Department of Computer Science
Faculty of Science
Pavol Jozef Šafárik University

Peter Vojtáš (Ed.)

ITAT 2005
Information Technologies - Applications and Theory

Workshop on Theory and Practice
of Information Technologies, Proceedings
Slovakia, September 2005

Department of Computer Science
Faculty of Science
Pavol Jozef Šafárik University

Peter Vojtáš (Ed.)

ITAT 2005
Information Technologies - Applications and Theory

Workshop on Theory and Practice
of Information Technologies, Proceedings
Slovakia, September 2005

Preface

The 5th workshop **ITAT'05 – Information Technology – Applications and Theory** (`http://ics.upjs.sk/itat`) was held in Horský Hotel Akademik, Račkova dolina (`http://www.akademik.sk/`) located 1000 meter above the sea level in Western Tatra, Slovakia, in September 2005.

ITAT workshop is a place of meeting of people working in informatics from former Czechoslovakia (official languages for oral presentations are Czech, Slovak and Polish; proceedings papers can be also in English).

Emphasis is on exchange of information between participants, rather than make it highly selective. Workshop offers a first possibility for a student to make a public presentation and to discuss with the "elders". A big space is devoted to informal discussions (the place is chosen at least 1000 meter above the sea level in a location not directly accessible by public transport).

Thematically workshop ranges from foundations of informatics, security, through data and semantic web to software engineering. In this year the emphasis is on data, information and knowledge.

Papers are divided into following classes:

- original scientific papers;
- practical applications;
- student papers
- didactics of informatics;
- extended abstracts;
- tutorial and
- work in progress.

All papers were refereed by at least two independent referees. There were 50 abstracts submitted. Due to an error one original paper is listed at the end of proceedings.

The workshop was organized by Institute of Informatics of University of P.J. Šafárik in Košice; Institute of Computer Science of Academy of Sciences of the Czech Republic, Prague; Faculty of Mathematics and Physics, branch Informatics of Charles University, Prague; Faculty of Informatics and Information Technology of the Slovak Technical University, Bratislava and Slovak Society for Artificial Intelligence.

Partial support has to be acknowledged from the slovak grant VEGA 1/0385 /03 "Intelligent search, retrieval and processing of information", the slovak IT project "Tools for knowledge acquisition and organization from heterogeneous information sources", and projects of the Program Information Society of the Thematic Program II of the National Research Program of the Czech Republic 1ET100300419 "Intelligent models, algorithms, methods and tools for the semantic web realization" and 1ET100300517 "Methods for intelligent systems and their application in data mining and natural language processing".

Peter Vojtáš

Program Committee

Gabriela Andrejková, *University of P.J. Šafárik, Košice, SK*
Mária Bieliková, *Slovak University of Technology in Bratislava, SK*
Viliam Geffert, *University of P.J. Šafárik, Košice, SK*
Ladislav Hluchý, *Slovak Academy of Sciences, Bratislava, SK*
Karel Ježek , *The University of West Bohemia, Plzeň, CZ*
Jozef Jirásek, *University of P.J. Šafárik, Košice, SK*
Jana Kohoutková, *Masaryk University, Brno, CZ*
Rastislav Lencses, *University of P.J. Šafárik, Košice, SK*
Pavol Návrat, *Slovak University of Technology in Bratislava, SK*
Ján Paralič, *Technical University in Košice, SK*
Karel Pala, *Masaryk University, Brno, CZ*
Dana Pardubská, *Comenius University, Bratislava, SK*
Martin Plátek, *Charles University, Prague, CZ*
Jaroslav Pokorný, *Charles University, Prague, CZ*
Karel Richta, *Czech Technical University, Prague, CZ*
Václav Snášel, *Technical University VŠB in Ostrava, CZ*
Vojtěch Svátek, *University of Economics in Prague, CZ*
Jirka Šíma, *Institute of Computer Science, AS CR, Prague, CZ*
Július Štuller, *Institute of Computer Science, AS CR, Prague, CZ*
Peter Vojtáš, (chair), *University of P.J. Šafárik, Košice, SK*
Filip Železný, *Czech Technical University, Prague, CZ*

Organizing Committee

Hanka Bílková, *Institute of Computer Science, AS CR, Prague, CZ*
Peter Gurský, *University of P.J. Šafárik, Košice, SK*
Stanislav Krajči, *University of P.J. Šafárik, Košice, SK*
Rastislav Lencses, (chair), *University of P.J. Šafárik, Košice, SK*

Organization

ITAT 2005 Information Technologies – Applications and Theory was organized by

University of P.J. Šafárik, Košice, SK

Institute of Computer Science, AS CR, Prague, CZ

Faculty of Mathematics and Physics, Charles University, Prague, CZ

Faculty of Informatics and Information Technology of the Slovak Technical University, Bratislava, SK

Slovak Society for Artificial Intelligence, SK

Table of Contents

Pôvodné vedecké práce 1

Adding semantics to stateful web services 3
M. Babík, L. Hluchý

Statická typová kontrola XSLT programů 13
D. Bednárek

Reasoning in multi-agent systems 23
I. Budinská, B. Frankovič

Aplikácia zhlukovania fuzzy konceptov v doméne textových dokumentov ... 31
P. Butka

Focused web crawling mechanism based on page relevance 41
E. Gatial, Z. Balogh, M. Laclavík, M. Ciglan, L. Hluchý

Genetické učení závislostních analyzátorů 47
T. Holan

Comparison of different approaches to overlapping clustering of binary variables 55
D. Húsek, A. Keprt, H. Řezanková, A.A. Frolov, P. Polyakov, V. Snášel

Conversion of regular expressions into 2-realtime automata 65
Ľ. Ištoňová

Ne-metrické vzdálenosti 75
M. Kolovrat, V. Snášel

Learning with kernel based regularization networks 83
P. Kudová

Generování a zpracování rozdílových dat stohově orientovaného systému ... 93
J. Kulhánek, D. Obdržálek

Sparse binary matrices 103
J. Martinovič, J. Dvorský, V. Snášel

On average complexity of InsertSort 117
P. Mlynárčik

Attacking PGV-like rate-2 cryptographic compression functions 123
R. Ostertág, M. Stanek

Alternatívna charakterizácia synchronizovaného alternovania 133
D. Pardubská, J. Wiedermann

O roztržitosti a volnosti slovosledu pomocí restartovacích automatů 145
M. Plátek, F. Mráz, F. Otto, M. Lopatková

Použití Hierarchického asociativního modelu neuronové sítě pro zlepšení lokalizace autonomního robota 157
J. Štanclová, D. Obdržálek

Pravděpodobnostní pohled na víceatributové dotazy v distribuovaných systémech 167
M. Vomlelová, P. Vojtáš

Zásobníkové automaty s orákulem 177
M. Žemlička

Praktické nasadené aplikácie 189

Frekvenční slovník češtiny 191
J. Hlaváčová

Ph.D. a študentská sekcia 199

Budování česko-anglického slovníku pro strojový překlad 201
O. Bojar

Virtuální lidé: jak representovat jejich znalosti 213
C. Brom

Paralelní řešení binární faktorové analýzy 223
A. Keprt

Building ontology for VirGIS system 233
Z. Linková, R. Nedbal

Why to use a type system for processing XML? 243
P. Loupal, M. Valenta

Stromová algebra v logike prvého rádu 251
R. Novotný, S. Krajči

Odhad struktury dat a induktivní logické programování 263
M. Řimnáč

Data pozičně závislá a jejich dopad v mobilních databázích 273
R. Špánek

Didaktika informatiky 279

Znalosti a dovednosti v přípravě informatiků (nezakusíš – nepochopíš) 281
J. Král

Studijní materiály s využitím DocBook stylu pro souběžnou českou a anglickou výuku C# 291
P. Šaloun

Rozšírené abstrakty 301

Aproximace vysokodimenzionálních dat pomocí perceptronových sítí 303
V. Kůrková

Semantic annotation based on regular expressions 305
M. Laclavík, E. Gatial, Z. Balogh, O. Habala, G. Nguyen, L. Hluchý

Query optimization by genetic programming 307
S.S.J. Owais, P. Krömer, V. Snášel

Tutoriály 309

Získávání logických tvrzení z dat jako významný směr dobývání znalostí z dat 311
M. Holeňa

Work in progress 323

Rámec pre prezentáciu informácií reprezentovaných ontológiou 325
M. Bieliková, V. Grlický, J. Kuruc

Problémy recyklování systému automatického překladu 335
O. Bojar, P. Homola, V. Kuboň

Design of ontology used for knowledge sharing 345
I. Budinská, B. Frankovič, G. Nguyen, L. Hluchý

Skrývání informace kompresní metodou kvantováním vektorů 353
J. Dvorský, E. Ochodková, V. Snášel

Representing content structure for e-learning systems 361
J. Dvořáková, J. Katreniaková, B. Rovan

Podpora navigácie v otvorených informačných priestoroch 371
K. Matušíková, M. Bieliková

Identifikácia zmien v ontológiách 381
M. Tury, M. Bieliková

Pôvodné vedecké práce II 391

Evoluce replikačních algoritmů v stohově orientovaných systémech 393
J. Dokulil, J. Yaghob, F. Zavoral

PÔVODNÉ VEDECKÉ PRÁCE

Adding semantics to stateful web services*

Marian Babík and Ladislav Hluchý

Department of Parallel and Distributed Computing, Institute of Informatics
Slovak Academy of Sciences
Marian.Babik@saske.sk, Ladislav.Hluchy@savba.sk

Abstract. Web Service Resource Framework (WSRF) is a recent effort of the grid community to facilitate modeling of the stateful services. Semantic Web Services (SWS) initiative is proposing a standard called OWL-S, to help the intelligent service provisioning i.e. automated discovery, composition and invocation of the services. In this paper we provide an overview of both frameworks and analyse their complementary features. We will also describe the OWL-S extensions necessary to cover the services introduced by the WSRF standard. Analysis of the requirements as well as description of the architecture is provided. A flood-forecasting application scenario is described, which have been used to evaluate the architecture.

1 Introduction

Recently, Web service (WS) technologies are gaining importance in the implementation of the distributed systems, especially grids. One such example is the Web Service Resource Framework (WSRF) [4], which extends the current WS technologies by modeling the stateful services. Design and development of the service oriented distributed system is quite common and there are several emerging WS initiatives, which tries to automate the process of discovery, composition and invocation of services. The semantic web services are a typical example, showing the potential of how ontological modeling can improve the shortcomings of the service oriented computing.

In this paper we propose the semantic description of the stateful web services introduced by the WSRF specifications. We also describe a corresponding distributed broker based architecture for discovery and invocation of both stateful and stateless services. We provide a brief overview of the semantic web services specification (OWL-S) and show how it can be used to model the basic stateful services as described by the WSRF.

* The research reported in this paper has been partially financed by the EU in the project IST-2004-511385 K-Wf Grid and Slovak national projects, VEGA No. 2/3132/23: Effective tools and mechanisms for scalable virtual organizations and Tools for acquisition, organization and maintenance of knowledge in an environment of heterogeneous information resources 2004-2007 SPVV 1025/04.

2 Semantic web services

One of the challenges of the loosely coupled distributed systems is the ability to dynamically discover and integrate the services needed by the applications. Interoperability among services is especially important in the distributed environments hosting a large number of services, i.e. grids. Semantic descriptions facilitates the process by expressing the characteristics of the service, which can later be used for their discovery, composition and invocation. The rich semantics needed for such descriptions can be provided by the ontologies.

OWL-S is an ontology-based approach to the semantic web services [1]. The structure of the ontology consists of a service profile for advertising and discovering services, a process model which supports composition of services, and service grounding, which associates profile and process concepts with underlying service interfaces. Service profile (OWL-S profile) has functional and non-functional properties. The functional properties describe the inputs, outputs, preconditions and effects (IOPE) of the service. The non-functional properties describe the semi-structured information intended for human users, e.g. service name, service description, and service parameter. Service parameter incorporates further requirements on the service capabilities, e.g. security, quality-of-service, geographical scope, etc. Service grounding (OWL-S grounding) enables the execution of the concrete Web service by binding the abstract concepts of the OWL-S profile and process to concrete messages. Although different message specifications can be supported by OWL-S, the widely accepted Web Service Description Language (WSDL) is preferred [2].

Other frameworks for semantic descriptions of the services, such as WSMF [17] and IRS [19], provide more elaborate service descriptions. Since we have focused mainly on the discovery and invocation, OWL-S was sufficient for our purposes. We have also considered the level of implementation and the number of tools available for WSMF and IRS.

3 WSRF

WS-Resource Framework (WSRF) is a set of specifications, which are based on the concept of modeling state as stateful resource and codify the relationship between Web Services (WS) and stateful resource in terms of a set of conventions on current (i.e. stateless) WS technologies [4]. A stateful resource is defined as having specific state data expressible as an XML document and a well defined life-cycle. It can be acted upon by one or more stateful services, e.g. files in a file system or rows in a relational database are considered a stateful resource. Fig. 1 shows a sample implementation scenario for the stateful service in the globus toolkit environment (GT4) [5].

In the example a virtual bookstore, which can buy and sell books is modeled. A sample stateful resource is a book, which is defined by various resource properties, e.g. title, author, price, etc. Adding a particular book is performed by contacting a service factory, which creates an instance of the resource (binding of

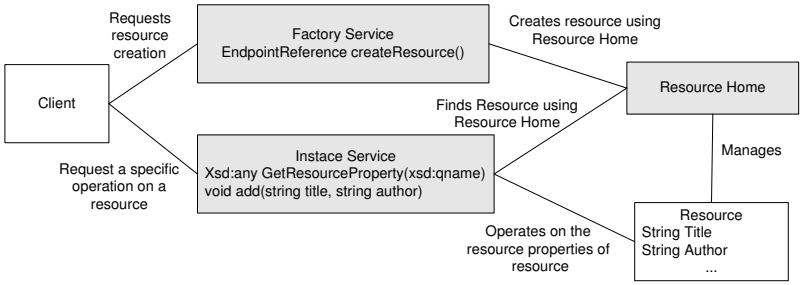


Fig. 1. Sample implementation scenario for the WS-Resource.

service and particular resource) and returns an identifier, i.e. so-called endpoint reference. The client can interact with the given resource by calling the methods exposed by the service instance interface. The pairing of the service instance and the resource is called a WS-Resource. There can be various different interfaces for the same resource, e.g. for buying books, selling books, listing books, etc. The actual resource instances are homed by the Resource home, which can be implemented as either persistent or transient store. Detailed description of the WSRF is beyond the scope of this paper; for additional details see [4].

4 Overview

In this section we provide an overview of the requirements of the broker based architecture motivated by the OWL-S Broker [14]. Essentially, broker based communication involves three parties, namely, provider of the services and resources, a client (requestor) requesting realization of its goals, and broker, which mediates the communication between provider and requestor.

The proposed broker is focused only on the automated discovery and invocation of services. The operations which should be performed by the broker include (1) the interpretation of the capabilities of the services, advertised by the provider, (2) interpretation of the requests from the clients, (3) discovery of the service, both keyword and semantic based, and (4) execution of the corresponding services.

In OWL-S, the service capabilities are described by the corresponding IOPE (inputs, outputs, preconditions and effects). Such description can be partly annotated from the WSDL description of the service and is sufficient for the stateless service. Stateful services, i.e. WS-Resources, however, are composed of a service and a stateful resource. Since resource as well as corresponding service are well defined, it is possible to semi-automatically map the resource description with a domain specific ontology. This can enable automated discovery and execution for the stateful services. A possible ontological mapping supporting OWL-S and WSRF is described in the next section.

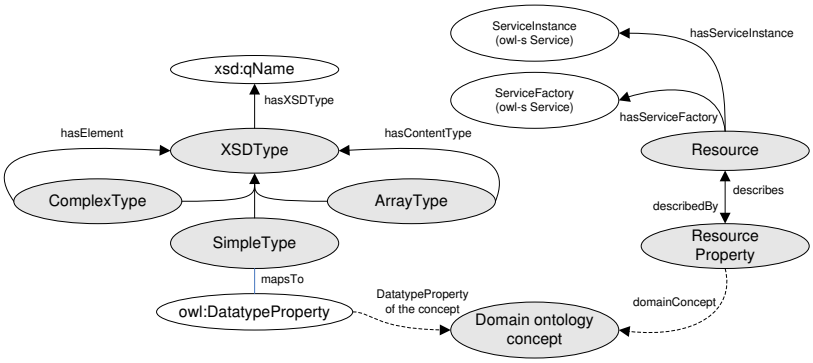


Fig. 2. A simplified schema of the ontological model (right) and datatype mapping (left) for WSRF services.

5 Service discovery

In our system, the process of service discovery has three parties, client, provider and broker. The provider is publishing the profile's functional and non-functional properties, called advertisements, to a broker. The client queries the broker for the given functional and non-functional properties and receives descriptions of all the relevant services. Service registry is comparing the request to the advertisements. This is called matchmaking of services. Since requests usually do not match exactly the advertisements, it is necessary to derive the corresponding match based on the subsumption, e.g. service reserving places in Cantonese restaurant should also match queries for Chinese restaurant. There are different ways of performing semantic matching of services [16]; however, in our case we'll only consider hierarchy of classes, properties and explicitly defined equivalence. In the implementation we have defined the degrees of matching by subsequent matching rules. The rules are based on the subsumption of classes, properties and also consider explicitly stated equivalence, i.e. the OWL construct owl:sameAs.

We have designed a simple ontology, which describes WS-Resource, i.e. resource, resource properties and its association with instance service and service factory, see Fig. 2. A resource class was defined having one or multiple instances of resource properties. The instances of resource properties are described by the identification (xsd:qname) and association to a relevant domain ontology concept. The resource instance has two object properties, which map the given resource to a service instance and a service factory. Since resource properties are already defined as an XML Schema, the ontological description can be created semi-automatically. The datatype mappings for the WS-Resource is described in the next section. It should be noted that both ontological and datatype mappings describe only the resource properties types, the actual instances of resource properties are implementation dependent.

In the process of implementation, there were several well known issues in the OWL-S profile, which were related to the OWL language and the limitations of the description logics in the description of services. A good survey of the OWL-S shortcomings is described in [8]. One of the major issues in the profile description is the conditional model. Since conditions refer to the concrete parameter instances, which are not known before execution, it is necessary to define the variables.

The latest version of OWL-S is addressing the issue and specifies the conditions by providing a class *Expression*. *Expression* is defined by the language used, e.g. SWRL [11], RDF [3], KIF [6], and the actual expression. Based on the language, the expression is incorporated as either XML Literal or String Literal. In our case we have used KIF for expressing conditions, therefore it was necessary to execute the given KIF expression on the instances of the service profiles.

6 Service invocation

Service grounding for the stateful service differs in the description of the resource properties document, which is specified in XML Schema [10]. It is thus essential to define a datatype mapping between XML Schema and OWL-S. Datatype mapping in OWL-S grounding is based on the RDFS and eliminates previous shortcomings, which were based on the XSLT approach. Currently, a mapping between XML schema datatypes and OWL, is based on semantics and uses a sample RDFS ontology, as described in [8]. The mapping is depicted in Fig. 2. It can be seen that the mapping reflects the nested data structures of the XML Schema. XML Schema SimpleType is mapped to a corresponding OWL type, which is bound to the datatype property of the OWL-S grounding. Together with complex and array types, a mapping through RDF property is used to map the elements to XSDTypes.

The basic WS-Resource has a well defined interface for accessing and modifying its resource properties and their values. This interface has three main methods, namely, getResourceProperty, setResourceProperty and getMultipleResourceProperties. The datatype mapping for the stateful services is based on mapping the inputs and outputs of these methods to a datatype property of the domain concept. The actual mapping is based on the RDFS mapping schema, which is complemented by the ontological mapping to the *Resource* class. Schematic representation of the process is shown in Fig 2.

7 Architecture

The architecture is shown in Fig. 3. The overall schema has three main components, namely, the client, the broker and the service providers. The broker's role is to mediate client requests for the service capabilities and match them with the description received from the service providers. The broker has knowledge base (KB) and reasoning engine based on the JTP [9], which is used to store

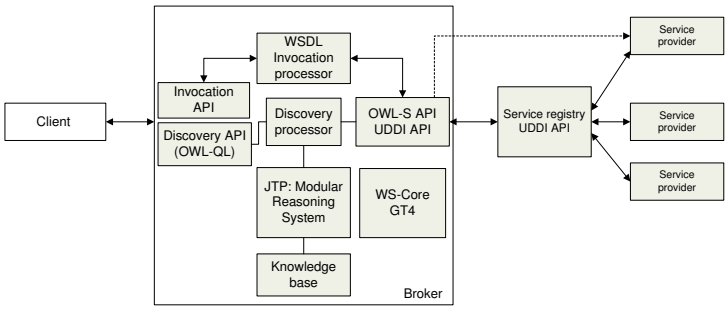


Fig. 3. Main components of the broker based architecture.

the service profiles and groundings and to perform the reasoning necessary for the process of matchmaking. The process of matchmaking is coordinated by the discovery processor, which receives the client's requests and queries the KB to find the relevant results. It is also used to process the descriptions from the service registries. In order to coordinate parsing of the description of the stateful service, access to the GT4 WS-core libraries is necessary [5]. The actual invocation of the service is performed by the WSDL invocation processor, which is also interconnected with the GT4 WS-Core.

The broker needs to interact with the service providers to get the WSDL and OWL-S specifications. This is accomplished by using the UDDI standard for service registries [7]. The corresponding mapping from OWL-S to UDDI was based on [13] and extended to support the stateful service descriptions. The client is accessing the broker through discovery and invocation API. Discovery API supports semantic matchmaking as well as keyword based lookups, which are implemented using the native UDDI API. Further, in order to access the OWL-S ontologies directly, the client can use the OWL-QL language to query the knowledge base [12]. This is supported by the fact that in the current implementation OWL-QL is transformed into a KIF query, which is native to our reasoning engine. The invocation API combines the OWL-S API [1] and GT4 invocation methods [5].

8 Application scenario

The Flood Forecast Simulation Cascade as shown in Fig. 4 is a hydrometeorological application, already well tested and developed during the project CrossGrid [15]. The application is a cascade (a natural workflow) of several simulation stages, which leads to the final result - a prediction of a potential flood. The cascade begins with meteorological prediction of weather for a short future period. This prediction is then recomputed into possible watershed, which in turn affects the water level of the target river. This water level is computed in the hydrological stage of the application. The resulting hydrograph (time series of

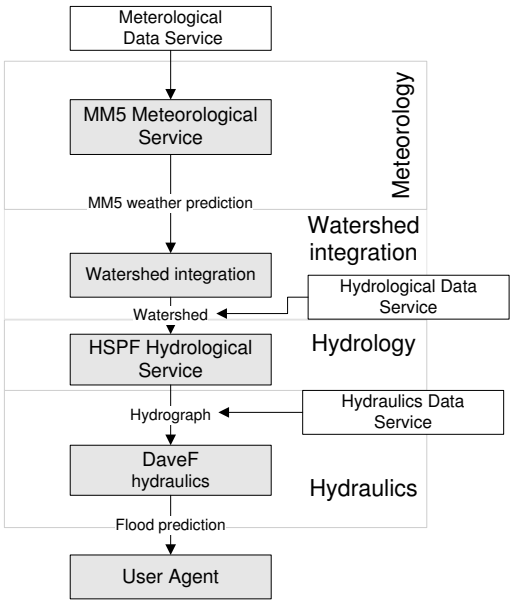


Fig. 4. Flood prediction scenario showing both stateful and stateless services.

water level values) is fed into the final stage of the cascade - a hydraulic prediction. This - using detailed terrain model of the target flooded area - computes the water flow in the target area, water depth and flow vectors.

In our application scenario the meteorological, hydrological and hydraulic methods are simple web (stateless) services. The inputs and outputs of the services are associated with the domain ontology concepts. In contrast, meteorological hydrological and hydraulic data services are stateful services, which are storing resources (in this case files in a file system). The services are also described by the non-functional parameter, i.e. geographic scope, which corresponds to the geographical scope of the data they are able to provide. The resource properties, i.e. descriptions of the file content, are modeled by the domain ontology. The goal of the discovery is to find the appropriate stateful services for a given region providing data in the required data format (specified by the resource properties). The subsequent invocation procedure executes the best suitable instances of services. Although only a static workflow is assumed, employing the OWL-S process model can enable more complex and dynamic workflows.

9 Related work

The Web Service Modeling Framework (WSMF)[17] is an industry scale framework for semantic web service discovery, execution and composition. It is a joint

effort of the European research projects on the Semantic Web and Semantic Web Services. It has three development areas concerning conceptual model (WSMO), the representation language (WSML) and the execution framework (WSMX). Although WSMF approach is much more profound and shows many significant contributions to modeling semantic web services, the level of implementation and the development support at the time of evaluation was unacceptable for our purposes. However, since WSMF has gained an enormous momentum in the last year, it will be considered in our future work.

The Internet Reasoning Service (IRS-II)[19] is a Java framework for publishing, locating, composing and executing semantic web services. IRS-II is modeling services based on the tasks, that needs to be fulfilled and the problem-solving-methods (PSM), that can be used to solve specific tasks. It utilizes a formal language called Operational Conceptual Modeling Language (OCML). It supports the specification and operationalization of functions, relations, classes, instances and rules. This proves to be more suitable for procedural knowledge representation than OWL. The process of assigning services the corresponding tasks has to be done manually, which is unscalable and introduces many possibilities for failure.

METEOR-S [18] attempts to add semantics to the basic web service descriptions by adding semantics to current industry standards. It is an effort of the LSDIS Lab of the University of Georgia. Although the framework presents an interesting bottom-up approach to the semantic descriptions, the service annotation and WSDL-S, semantic extension of the WSDL, are not scalable in terms of functionality to the level of semantic information provided by the previous approaches.

10 Conclusion

We have described a broker based architecture for automated service discovery and invocation. We have shown a flood prediction application scenario, that we have used to evaluate the initial implementation. In the future we would like to extend our work to consider OWL-S process model as well as further WSRF specifications, such as WS-Notification, WS-Security, WS-GRAM, WS-transfer, etc.

References

1. A. Ankolekar et.al, OWL-S: Semantic Markup for Web Service, 2003, `http://www.daml.org/services/owl-s/1.1`
2. E. Christensen, F. Cubera, G. Meredith, S. Weerawarana, Web Services Description Language (WSDL) 1.1, Technical report, WWW Consortium, 2001, `http://www.w3c.org/TR/wsdl`
3. Resource Description Framework, `http://www.w3.org/RDF/`
4. Web Service Resource Framework, `http://www.globus.org/wsrf/`
5. Globus Toolkit, `http://www-unix.globus.org/toolkit/`

6. Knowledge Interchange Format, draft to American National Standard, 1998, http://logic.stanford.edu/kif/dpans.html
7. Universal Description, Discovery and Integration Protocol Specification, UDDI Consortium, 2001, http://www.uddi.org/specification.html
8. Pitfalls of OWL-S – A practical Semantic Web Use Case, S. Balzer and T. Liebig and M. Wagner, Proceedings of the 2nd International Conference on Service Oriented Computing (ICSOC004), 2004
9. JTP: A System Architecture and Component Library for Hybrid Reasoning, R. Fikes, J. Jenkins, and G. Frank, Proceedings of the Seventh World Multiconference on Systemics, Cybernetics, and Informatics. Orlando, Florida, USA, 2003
10. Extensible Markup Language (XML), XML Protocol Working Group, 2004, http://www.w3.org/TR/2004/REC-xml11-20040204/
11. A semantic web rule language combining OWL and RuleML (SWRL), Ian Horrocks et.al., 2003, http://www.daml.org/2003/11/swrl/
12. R. Fikes, P. Hayes, I. Horrocks, OWL-QL: A Language for Deductive Query Answering on the Semantic Web, KSL Technical Report 03-14, 2003
13. N. Srinivasan, M. Paolucci and K. Sycara, Adding OWL-S to UDDI, implementation and throughput, First International Workshop on Semantic Web Services and Web Process Composition, (SWSWPC 2004) 6-9, 2004, San Diego, California, USA
14. M. Paolucci, J. Soudry, N. Srinivasan, and K. Sycara, Untangling the Broker Paradox in OWL-S, Proceedings of AAAI 2004 Spring Symposium
15. CrossGrid consortium, CrossGrid Technical Annex, 2004, http://www.crossgrid.org
16. L. Li and I. Horrocks, A software framework for matchmaking based on semantic web technology. In Proc. of the Twelfth Inl. Word Wide Web Conference (WWW 2003), pp. 331-339, ACM, 2003
17. Fensel D. and Bussler C., The Web Service Modeling Framework WSMF, Eletronic Commerce: Research and Applications, 1, 2002
18. P. Rajasekaran and J. Miller and K. Verma and A. Sheth, Enhancing Web Services Description and Discovery to Facilitate Composition, International Workshop on Semantic Web Services and Web Process Composition, 2004
19. Motta E. and Domingue J. and Cabral L. and Gaspari M., IRS-II: A Framework and Infrastructure for Semantic Web Services, 2nd International Semantic Web Conference (ISWC2003), Sundial Resort, Sanibel Island, Florida, USA, 2003

Statická typová kontrola XSLT programů*

David Bednárek

Katedra softwarového inženýrství, Matematicko-fyzikální fakulta
Univerzita Karlova v Praze
david.bednarek@mff.cuni.cz

Abstrakt Většina jazyků pro popis XML transformací, včetně XSLT, nemá explicitní typový mechanismus a programátorské chyby tak často vedou k formálně korektním programům produkujícím nechtěné výsledky. Idea automatické statické typové kontroly těchto jazyků vzhledem k danému vstupnímu či výstupnímu DTD zaujala mnoho výzkumníků, většina dosud publikovaných metod se však zaměřuje na samostatně stojící výrazy jazyka X-Path a nelze je snadno rozšířit na celý jazyk XSLT. V tomto článku je prezentováno kriterium vnitřní korektnosti XSLT programu jako kombinace dosažitelnosti a užitečnosti vzhledem k danému vstupnímu DTD a je popsán postup výpočtu dosažitelnosti a užitečnosti pro podmnožinu jazyka XSLT a konzervativní aproximace těchto vlastností pro zbylé konstrukce jazyka XSLT. Popsaná statická analýza může napomoci ve snaze o tvorbu korektních XSLT programů, kromě toho je možné vlastnosti zjištěné touto metodou využít při řešení dalších úloh, jako transformace XSLT na SQL nebo optimalizace vyhodnocování XSLT.

1 Úvod

Většina jazyků určených ke zpracování XML postrádá typový systém, což ponechává bez povšimnutí řadu běžných chyb, které se většinou projevují generováním nesprávného výstupu. Několik metod statické analýzy a navazující typové kontroly je popsáno např. v článcích [4,8] pro XPath a pro řadu nepříliš rozšířených formalismů jako JWIG [1], XDuce [6], k-pebble transducers [10] a další [11]. Pro XSLT byla popsána pouze přibližná validace výstupního XML vzhledem k vstupnímu DTD [7], navíc omezená na XSLT nevyužívající parametry typu node-set. V tomto článku se budeme zabývat statickou kontrolou vnitřní konzistence XSLT programů vzhledem ke vstupnímu DTD, to jest vlastnostmi jako dosažitelnost šablon či XPath fragmentů v nich použitých. Popsaná metoda analýzy zvládá významnou podmnožinu jazyka XSLT a je možné ji použit i v dalších oblastech, které se zabývají statickou analýzou XSLT programů. Z nich je nejpodstatnější problematika optimalizace vyhodnocování [5] či konverze do SQL [9], většina popsaných technik se však zabývá pouze jazykem XPath. Pro některé z těchto technik může zde popsaná metoda sloužit jako prostředek k rozšíření na XSLT programy.

* Projekt programu "Informační společnost" Tematického programu II Národního výzkumného programu České republiky, projekt číslo 1ET100300419.

Tento příspěvek je uspořádán takto: Po stručném popisu omezení, kladených na zpracovávané XSLT, bude kapitola 2 pojednávat o abstrakcích XML dokumentů, DTD, XPath výrazů a XSLT programů používaných v tomto článku. Na základě těchto abstrakcí pak budou definovány pojmy dosažitelnosti a užitečnosti jako kritéria vnitřní korektnosti XSLT programů. Kapitola 3 popíše základní principy konverze XSLT programů do abstraktní reprezentace. Kapitola 4 pak definuje klíčový pojem linearity XSLT programu a kritérium linearizovatelnosti. Na lineární formě jsou pak stručně popsány algoritmy další transformace a závěrečná konstrukce množiny dosažitelných typů hodnot pro každý výraz v programu, přičemž typem je zde rozuměn odkaz do abstraktní formy DTD omezující vstupní strom. V závěru bude uvedena asymptotická složitost uvedených transformací a analýzy jako celku.

1.1 Uvažovaná podmnožina XSLT

Stejně jako většina prací, zabývajících se problematikou statické analýzy XPath a XSLT, nebudeme v této práci uvažovat operace s textovým obsahem atributů a elementů. To především znamená vyloučení predikátů = a ! = a dalších řetězcových funkcí. Podobně vyloučíme operace s čísly, odvozenými ze vstupního XML dokumentu konverzemi či funkcemi *position*(), *last*(), *count*() a dalšími agregačními funkcemi. Toto vyloučení má svoje teoretické opodstatnění: Pomocí řetězcových operací je možné v XSLT simulovat Turingův stroj a je dobře známo, že statická analýza chování Turingova stroje je algoritmicky neřešitelný problém. Podobně bychom pomocí číselných operací mohli simulovat částečně rekurzivní funkce, jejichž chování je opět algoritmicky nerozhodnutelné.

Z obdobných důvodů neuvažujeme XSLT programy, které čtou fragmenty výstupního stromu. Takové programy byly v XSLT 1.0 zakázány, řada XSLT procesorů však takové chování umožňovala (například pomocí nestandardní funkce *nodeset*). V XSLT 2.0 je takové chování povoleno a explicitně specifikováno pomocí pojmu dočasný strom. XSLT programy se čtením výstupních/dočasných stromů však mohou snadno simulovat chování Turingova stroje a jejich statická analýza tak není obecně algoritmizovatelná. Jedinou povolenou možností, jak zpracovat parametr či proměnnou obsahující výstupní strom, zůstává okopírování do jiného výstupního stromu instrukcí $\langle copyof \rangle$. Vzhledem k tomu, že se v tomto článku nezabýváme analýzou výstupního stromu, ale pouze vnitřním chováním a korektností XSLT programu, můžeme analýzu parametrů a proměnných obsahujících výstupní stromy omezit na kontrolu jejich využití.

2 Abstraktní reprezentace DTD a XSLT

Tato kapitola popisuje abstraktní reprezentaci DTD (případně XML-Schema), popisujícího formát vstupního dokumentu, a analyzovaného XSLT programu. Tyto dvě reprezentace jsou vstupem vlastního algoritmu statické analýzy a na jejich základě je také definován význam pojmu vnitřní korektnosti XSLT programu vzhledem k danému DTD.

Obě abstraktní reprezentace se odvolávají na abstraktní reprezentaci XML dokumentu, zvanou FR-strom, nad kterou je vystavěna abstraktní forma XPath výrazů.

Abstrakce formou automatů či gramatik jsou používány velmi často (např. v [2], [3]), ovšem v poněkud odlišných formách.

2.1 Abstrakce vstupního XML dokumentu

Abychom zjednodušili zacházení se soupravou os, používaných v XPath pro navigaci ve vstupním XML dokumentu, budeme XML dokument abstrahovat jako binární strom nazývaný FR strom. Jeho uzly odpovídají původním uzlům XML dokumentu, z každého uzlu tohoto stromu však vycházejí (nejvýše) dvě hrany, značené F (first) a R (right). Hrana F vždy odkazuje na prvního syna v původním XML dokumentu, hrana R pak odkazuje na dalšího sourozence v původním dokumentu. Ačkoliv se tato abstrakce na první pohled vzdaluje od obvyklého chápání XML dokumentu a komplikuje nejběžnější způsoby navigace v něm, byla úspěšně použita v řadě teoretických analýz, a jak ukáže tento článek, nemusí být ani nevýhodná pro praktické zpracování XML.

Pro navigaci v FR stromě budeme používat zmíněné symboly F a R označující sestup po příslušných hranách stromu, a doplníme je o symboly B (back) a L (left) pro výstup proti směru hrany F resp. R. Pomocí čtveřice symbolů $\{F, R, B, L\}$, nazývané abeceda atomických os, a jazyka regulárních výrazů pak můžeme popsat všechny původní osy jazyka XPath podle tabulky 1.

child	$F.R*$
parent	$L*.B$
descendant	$F.R*.(F.R*)*$
ancestor	$L*.B.(L*.B)*$
descendant-or-self	$(F.R*)*$
ancestor-or-self	$(L*.B)*$
preceding-sibling	$L.L*$
following-sibling	$R.R*$
preceding	$(L*.B)*.L*.L.F.(F\|R)*$
following	$(L*.B)*.R.(F\|R)*$

Tab. 1. Transformace os jazyka XPath do abecedy atomických os.

Každý uzel FR-stromu nese stejně jako v původním XML dokumentu jméno elementu. Protože při analýze XSLT musíme zvládnout i odkazy na atributy (tj. osu attribute), zařadíme atributy mezi syny, tj. na osu $F.R*$, a budeme je od elementů odlišovat pouze vhodnou pojmenovávací konvencí.

2.2 Abstrakce DTD

Pokud je množina přípustných vstupních dokumentů omezena pomocí DTD nebo XML-Schema, je možné toto omezení snadno převést na omezení aplikovaná na FR strom, a to následujícím postupem:

DTD a XML-Schema se obvykle abstrahují jako rozšířený konečný automat $A = (V, Q, \delta, q_I, Q_F)$
zpracovávající abecedu V složenou ze jmen elementů (a atributů),
s množinou stavů Q,
přechodovou funkcí $\delta : Q \times V \to Q \times Q$
počátečním stavem $q \in Q$
a množinou koncových stavů $Q \subseteq Q$

Tento automat určuje přípustné posloupnosti elementů (a atributů) z abecedy V. Jeho přechodová funkce $\delta(q, e) = (q_r, q_f)$ určuje pro daný stav q a element e kromě obvyklého následujícího stavu q_r rovněž iniciální stav q_f (téhož automatu) určený pro rozpoznávání povolených posloupností elementů uvnitř elementu e.

2.3 Abstrakce XPath výrazů

XPath výrazy typu node-set jsou abstrahovány v souladu s abstrakcí vstupních XML dokumentů pomocí abecedy atomických os. Navigace popsaná jako regulární výraz nad atomickými osami je doplněna filtry, testujícími hodnotu booleovského výrazu složeného z atomických filtrů. Atomickým filtrem může být test jména elementu uzlu, zapisovaný jako $\langle e \rangle$, test neprázdnosti závislého výrazu typu node-set, nebo odkaz na booleovskou proměnnou. Celý výraz typu node-set je sjednocením nezávislých atomických výrazů typu node-set, jejichž prefixem je kontext, kořen dokumentu nebo odkaz na proměnnou typu node-set. Přesnou definici podává gramatika na obrázku 1.

2.4 Abstrakce XSLT programu

Normalizovaný XSLT program je abstrahován jako následující struktura

$$X = (ST, UT, IT, FET, t_R, NSParam, BParam, NSP, BP, S, U, I, FE)$$

kde $Tmpl = ST \cup UT \cup IT \cup FET$ je množina jmen šablon disjunktně rozdělená do čtyř možných tvarů šablon, $t_R \in Tmpl$ je počáteční šablona programu. $NSParam$ je množina jmen parametrů typu node-set, $BParam$ je množina jmen Booleovských parametrů,

$NSP \subseteq Tmpl \times NSParam$
$BP \subseteq Tmpl \times BParam$

jsou přiřazení formálních parametrů šablonám s tím, že počáteční šablona parametry nemá:

$NSP(t_R) = \emptyset \wedge BP(t_R) = \emptyset$

Zbývající součásti definují těla šablon jednotlivých druhů:

$NSExpr \rightarrow IndepNSExpr$	nezávislý výraz typu node-set
$NSExpr \rightarrow NSExpr \vert IndepNSExpr$	sjednocení
$IndepNSExpr \rightarrow IndepPrefix$	nezávislý výraz bez navigace
$IndepNSExpr \rightarrow IndepPrefix\,.\,NSPath$	nezávislý výraz s navigací
$IndepPrefix \rightarrow \mathbf{context}$	kontextový uzel
$IndepPrefix \rightarrow \mathbf{root}$	kořen dokumentu
$IndepPrefix \rightarrow \$NSParam$	odkaz na parametr typu node-set
$NSPath \rightarrow AtomicAxis$	navigace o jeden krok
$AtomicAxis \rightarrow F \mid R \mid B \mid L$	atomický krok
$NSPath \rightarrow [BExpr]$	filtr
$NSPath \rightarrow NSPath\,.\,NSPath$	zřetězení
$NSPath \rightarrow (NSPath \vert NSPath)$	alternativa
$NSPath \rightarrow (NSPath)*$	iterace
$BExpr \rightarrow IndepBAtom$	nezávislý Booleovský atom
$BExpr \rightarrow DepBAtom$	závislý Booleovský atom
$BExpr \rightarrow \mathbf{not}(BExpr)$	negace
$BExpr \rightarrow BExpr\ \mathbf{and}\ BExpr$	konjunkce
$BExpr \rightarrow (BExpr\ \mathbf{or}\ BExpr)$	disjunkce
$IndepBAtom \rightarrow \$BParam$	odkaz na Booleovský parametr
$IndepBAtom \rightarrow \mathbf{boolean}(IndepNSExpr)$	test neprázdnosti nezávislého výrazu
$DepBAtom \rightarrow \mathbf{boolean}(DepNSExpr)$	test neprázdnosti závislého výrazu
$DepBAtom \rightarrow < ElementName >$	test typu elementu
$DepNSExpr \rightarrow NSPath$	navigace od aktuálního uzlu
$IndepBExpr \rightarrow IndepBAtom$	nezávislý Booleovský atom
$IndepBExpr \rightarrow \mathbf{not}(IndepBExpr)$	negace
$IndepBExpr \rightarrow IndepBExpr\ \mathbf{and}\ IndepBExpr$	konjunkce
$IndepBExpr \rightarrow (IndepBExpr\ \mathbf{or}\ IndepBExpr)$	disjunkce

Obr. 1. Gramatika abstrahovaných X-Path výrazů.

$S : ST \rightarrow \mathbf{seq\ of}\ Call$ šablony typu sekvence tvořené posloupností volání jiných šablon

$U : UT \rightarrow NSExpr$ šablony typu $\langle use \rangle$ vyhodnocující daný výraz typu node-set

$I : IT \rightarrow BExpr \times Call$ šablony typu $\langle if \rangle$ volající jinou šablonu v závislosti na Booleovské podmínce

$FE : FET \rightarrow NSExpr \times Call$ šablony typu $\langle foreach \rangle$ volající jinou šablonu pro každý uzel daný řídícím výrazem typu node-set. Každé volání šablony $c \in Call$ je pak popsáno takto:

$c = (t, NSWP, BWP)$

kde $t \in Tmpl$je volaná šablona,

$NSWP : NSP(t) \rightarrow NSExpr$

$BWP : BP(t) \rightarrow BExpr$

jsou přiřazení skutečných parametrů formálním parametrům volané šablony.

Výpočet XSLT programu pro daný vstupní dokument d je abstrahován jako strom, jehož každý uzel odpovídá vyvolání některé ze šablon. Každé vyvolání je popsáno aktivačním záznamem ve tvaru čtveřice

$i = (t, context, NSV, BV)$
kde $t \in Tmpl$ je volaná šablona,
$context \in Node(d)$ je kontext - jeden z uzlů vstupního stromu d
$NSV \subseteq NSP(t) \times Node(d)$ je ohodnocení node-set parametrů šablony množinami uzlů
$BV : BP(t) \to Boolean$je ohodnocení Booleovských parametrů.
Kořenem stromu je aktivační záznam počáteční šablony
$i_0(d) = (t_R, root(d), \emptyset, \emptyset)$.

Strom výpočtu musí přirozeným způsobem odpovídat sémantice volaných šablon:

Každý aktivační záznam šablony typu sekvence má počet potomků odpovídající délce sekvence v těle šablony. Aktivační záznam šablony typu $\langle foreach \rangle$ má tolik potomků, kolik uzlů vstupního stromu je obsaženo v hodnotě jejího řídícího výrazu odpovídající ohodnocení parametrů. Aktivační záznam šablony typu $\langle if \rangle$ má nejvýše jednoho potomka, který je přítomen tehdy, je-li testovaný výraz pravdivý pro dané ohodnocení parametrů. Aktivační záznam šablony typu $\langle use \rangle$ je vždy listem stromu.

Ohodnocení parametrů v každém aktivačním záznamu ve stromě výpočtu je přirozeným způsobem určeno hodnotami parametrů jeho rodiče. V případě šablony typu $\langle foreach \rangle$ kontext potomků iteruje přes příslušný výraz, u ostatních typů je kontext zděděn od rodiče.

Důležitou vlastností stromu výpočtu je skutečnost, že výskyt a ohodnocení uzlu je určeno pouze rodičem uzlu a jednotlivé cesty se vzájemně neovlivňují, a nedochází ani k ovlivňování rodičů potomky. To znamená, že dosažitelnost aktivačních záznamů můžeme zkoumat na základě relace call specifikující vzájemné volání aktivačních záznamů:

$call(d) \subseteq I(d) \times I(d)$
kde $I(d)$ je množina všech aktivačních záznamů ve tvaru odpovídajícím vstupnímu stromu d. Strom výpočtu pak obsahuje právě uzly z množiny

$call^*(d)[\{i_0(d)\}]$, kde $call^*(d)$ je reflexivní a tranzitivní uzávěr relace $call(d)$ a $i_0(d)$ je aktivační záznam počáteční šablony.

2.5 Kritéria vnitřní konzistence XSLT programu

Abstrahovaný XSLT program je považován za vnitřně konzistentní, splňuje-li kritéria dosažitelnosti a užitečnosti.

Definition 1. *Šablona $t \in Tmpl$ je dosažitelná vzhledem k danému DTD, pokud existuje dokument d odpovídající tomuto DTD, v jehož stromě výpočtu je šablona obsažena, to jest existuje aktivační záznam $i = (t, context, NSV, BV)$ takový, že $i \in call^*(d)[\{i_0(d)\}]$.*

Definition 2. *Výraz nebo podvýraz X typu node-set, uvedený v těle šablony t, je dosažitelný vzhledem k danému DTD, jestliže existuje dokument d odpovídající tomuto DTD a aktivační záznam $i = (t, context, NSV, BV) \in call^*(d)[\{i_0(d)\}]$ ve stromě výpočtu pro d takový, že hodnota výrazu X v kontextu i je neprázdná.*

Definition 3. *Booleovský výraz nebo podvýraz X, uvedený v těle šablony t, je užitečný vzhledem k danému DTD, jestliže existují dokumenty d_1 a d_2 odpovídající tomuto DTD a aktivační záznamy*
$i_1 = (t, context_1, NSV_1, BV_1) \in call^*(d_1)[\{i_0(d_1)\}]$
$i_2 = (t, context_2, NSV_2, BV_2) \in call^*(d_2)[\{i_0(d_2)\}]$
takové, že hodnota výrazu X v kontextu i_1 je true a hodnota výrazu X v kontextu i_2 je false.

Definition 4. *XSLT program je vnitřně konzistentní vzhledem k danému DTD, pokud je každá jeho šablona dosažitelná, každý výraz a podvýraz typu node-set v ní uvedený je dosažitelný, a každý Booleovský výraz a podvýraz v ní je užitečný.*

3 Konverze XSLT programu do abstraktní reprezentace

Z množiny XSLT elementů (kterým budeme pro odlišení od vstupních XML elementů říkat instrukce), je před samotnou analýzou nutné odstranit ty, které mají komplikovanou sémantiku, a nahradit je ekvivalentním vyjádřením pomocí jednodušších instrukcí. V první fázi zredukujeme jazyk XSLT na tuto sadu instrukcí:

$\{use, foreach, if, variable, calltemplate, withparam\}$

Instrukce $\langle use \rangle$ je nová instrukce nahrazující výstupní instrukce $\langle copyof \rangle$, $\langle valueof \rangle$ a $\langle copy \rangle$ a slouží jako konzument XPath výrazu, který v těchto výstupních instrukcích byl (případně implicitně) použit. V naší analýze nezkoumáme obsah výstupního stromu, ale pouze XPath výrazy, které je při výpočtu nutné vyhodnocovat, a proto je tato instrukce dostatečnou náhradou původních instrukcí.

Instrukci $\langle applytemplates \rangle$ lze převést na kombinaci $\langle foreach \rangle$, $\langle choose \rangle$, $\langle when \rangle$ a $\langle calltemplate \rangle$ s tím, že volané šablony se zároveň převedou z režimu mode/match do režimu name. Tento převod není triviální, neboť je nutné projít všechny šablony s příslušným atributem mode (včetně implicitně definovaných) a převést jejich výrazy match na podmínky instrukcí $\langle when \rangle$ otočením pořadí a orientace jejich XPath kroků. Jednotlivé $\langle when \rangle$ instrukce je pak nutné seřadit podle priorit příslušných šablon.

V další fázi zpracování je XSLT program upraven do normalizované podoby, která přináší především jednodušší a srozumitelnější popis algoritmů. Normalizace spočívá v odsunutí všech vnořených struktur do samostatných šablon tak, aby každá šablona obsahovala pouze jednu konstrukci. Po konverzi XPath výrazů do tvaru s atomickými osami výsledná forma XSLT programu odpovídá abstraktní reprezentaci popsané v předchozí kapitole.

4 Test dosažitenosti a užitečnosti

Test dosažitelnosti a užitečnosti XSLT programu má dvě fáze: Transformace abstraktní reprezentace XSLT do tzv. lineární podoby a analýza dosažitelnosti a užitečnosti výrazů v této lineární podobě. Ne každý XSLT program může být

do lineární podoby transformován, vlastnosti nelinearizovatelných programů jsou však natolik exotické, že se v praxi vyskytují řídce. Jako ilustraci uveďme, že nelinearizovatelným programem je možno simulovat činnost lineárně omezeného Turingova stroje.

4.1 Lineární šablony

Zápisem $X[p = v]$ označme hodnotu výrazu X, ve kterém je za parametr p dosazena hodnota v.

Definition 5. *Výraz X je lineární vůči parametru p, jestliže platí:*
$X[p = v_1 \cup v_2] = X[p = v_1] \cup X[p = v_2]$ *pro X a p typu node-set*
$X[p = v_1 \cup v_2] = X[p = v_1] \vee X[p = v_2]$ *pro Booleovské X a p typu node-set*
$X[p = v_1 \vee v_2] = X[p = v_1] \cup X[p = v_2]$ *pro X typu node-set a Booleovské p*
$X[p = v_1 \vee v_2] = X[p = v_1] \vee X[p = v_2]$ *pro Booleovské X a p*

Definition 6. *Šablona t je lineární vůči parametru p, jestliže současně platí, že:*
1. Je-li p použit v řídícím výraze instrukce $for-each$, if nebo set, je tento výraz jediným výrazem v celé šabloně t, ve kterém se p vyskytuje.
2. V každém volání šablony ze šablony t je p použit nejvýše v jednom ze skutečných parametrů.
3. Každý výraz v šabloně, ve kterém se p vyskytuje, je lineární vůči p.

Definition 7. *Šablona t je lineární, je-li lineární vůči každému svému parametru.*

Definition 8. *XSLT program je lineární, pokud je každá jeho šablona lineární.*

Užitečnost lineárního tvaru programu spočívá v tom, že sledujeme-li nějakou statickou lineární vlastnost kterékoliv jeho šablony (například dosažitelnost nebo horní odhad jakéhokoliv výrazu v ní), je tato vlastnost lineární vůči každé jiné šabloně. To umožňuje další úpravy XSLT programu tak, jak jsou popsány v odstavci 4.2, aniž by byla změněna sledovaná vlastnost.

4.2 Převod XSLT do lineárního tvaru

Definition 9. *Graf závislosti parametrů XSLT programu je orientovaný graf, jehož vrcholy jsou dvojice $\langle t, p\rangle$, kde t je šablona a p je její parametr, přičemž hrana $\langle\langle t_1, p_1\rangle, \langle t_2, p_2\rangle\rangle$ je v grafu tehdy, pokud šablona t_1 volá šablonu t_2, přičemž jejímu formálnímu parametru p_2 je přiřazen výraz obsahující parametr p_1. Hrana se nazývá netriviální, pokud tento výraz není identita.*

Theorem 1. *XSLT program je převoditelný do lineární formy právě tehdy, pokud v jeho grafu závislosti parametrů neexistuje cyklus obsahující netriviální hranu.*

Důkaz tohoto tvrzení je mimo prostorové možnosti tohoto článku.

Převod do lineární formy může zvětšit XSLT program na kvadratickou velikost.

4.3 Analýza lineárního tvaru

Lineární tvar XSLT programu umožňuje jeho další transformaci těmito úpravami:

Každé volání šablony se skutečným parametrem tvaru $A|B$ lze nahradit sekvencí dvou volání s parametrem A resp. B. Každé volání šablony se skutečným parametrem tvaru $A.B$ lze nahradit voláním pomocné šablony t_1 s pomocným parametrem $p = A$, ze které bude volána původní šablona t s parametrem $p.B$.

Každé volání šablony t se skutečným parametrem tvaru $A.B*$ lze nahradit voláním pomocné šablony t_1 s pomocným parametrem $p = A$, ze které bude sekvenčně volána původní šablona t s parametrem p a rekurzivně šablona t_1 s parametrem $p.B$.

Žádná z těchto úprav nenarušuje statické lineární vlastnosti programu.

Těmito úpravami je možno program transformovat na tvar, ve kterém je každý skutečný parametr typu node-set ve tvaru

$NSParam\,.\,AtomicAxis$

nebo

$NSParam\,.\,[BExpr]$

Následkem toho může každý node-set parametr obsahovat nejvýše jeden uzel zpracovávaného stromu. To umožňuje další statickou analýzu vlastností programu vůči danému DTD, neboť nyní je možno sledovat všechny aktivace dané šablony programu jako n-tice stavů DTD automatu generující FR-strom. Na základě sledování dosažitelnosti těchto n-tic je pak možno určit pro všechny parametry a výrazy v XSLT programu množinu stavů DTD automatu, kterým mohou odpovídat uzly FR-stromů obsažené v hodnotách těchto výrazů. Speciálně, pokud je tato množina prázdná, je výraz nedosažitelný, a pokud se jedná o kontext šablony, je nedosažitelná i tato šablona. Obdobným způsobem je možno rozhodovat i o užitečnosti booleovských výrazů v programu.

5 Závěr

Popsaným postupem je možno v případě, že XSLT program splňuje podmínku linearizovatelnosti (kterou je možno otestovat v polynomiálním čase), určit dosažitelnost všech šablon a dosažitelnost resp. užitečnost výrazů v něm s ohledem na dané DTD omezující vstup programu.

Toto určení dosažitelnosti je proveditelné v čase a prostoru $O(m \cdot n)$, kde m je velikost DTD a n maximální počet parametrů šablony v linearizovaném tvaru, přičemž platí, že n je menší, než velikost původního XSLT programu před linearizací. Tato exponenciální složitost je sice poněkud odstrašující, prvotní experimenty však ukazují, že skutečný počet dosažitelných n-tic bývá významně menší a jejich vyčíslení je tedy prakticky proveditelné.

Další výzkum v oblasti statické analýzy by měl být zaměřen na praktické ověření uvedené metody na vhodné sadě veřejně dostupných XSLT. Tím se jednak zjistí praktické časové nároky metody, jednak úspěšnost statické analýzy v odhalování chyb v XSLT programech. Pro srovnání, v práci [7] je popsána

praktická aplikace statické validace výstupu XSLT programu, kterou bylo odhaleno nezanedbatelné množství chyb i v programech, které bývají považovány za bezchybné.

Metoda linearizace programu a případně další úpravy linearizovaného programu popsané v tomto článku mohou být užitečné i mimo oblast validace: Například řada metod, konvertujících XPath výrazy na SQL, může být pravděpodobně rozšířena i na linearizované XSLT programy. Další výzkum by se tedy měl věnovat i této oblasti.

Reference

1. Aske Simon Christensen, Anders Muller, Michael I. Schwartzbach, Static Analysis for Dynamic XML. In Proceedings of 1st Workshop on Programming Languages Technology for XML (PLAN-X 2002), 2002, `http://citeseer.ist.psu.edu/709823.html`
2. Boris Chidlovskii, Using Regular Tree Automata as XML Schemas. In Proceedings of the IEEE Advances in Digital Libraries 2000 (ADL 2000), `http://citeseer.ist.psu.edu/chidlovskii99using.html`
3. F. Neven, "Extensions of Attribute Grammars for Structured Document Queries". In Proc. 7th Intl. Workshop on Database Programming Languages (DBPL'99), 1999, `http://sherry.ifi.unizh.ch/neven99extensions.html`
4. Frank Neven and Thomas Schwentick, "XPath Containment in the Presence of Disjunction, DTDs, and Variables", International Conference on Database Theory, 2003, `http://citeseer.ist.psu.edu/neven03xpath.html`
5. G. Gottlob, C. Koch, and R. Pichler, "Efficient Algorithms for Processing XPath Queries". In Proc. of the 28th International Conference on Very Large Data Bases VLDB 2002, `http://citeseer.ist.psu.edu/gottlob02efficient.html`
6. H. Hosoya and B. C. Pierce, Xduce: A Statically Typed xml Processing Language. ACM Transactions on Internet Technology, 2002. Submitted for publication, `http://sherry.ifi.unizh.ch/hosoya02xduce.html`
7. Mads Kristian Østerby Olesen, "Static Validation of XSLT Transformations", Master's Thesis, University of Aarhus, 2004
8. P.T. Wood, Containment for XPath Fragments under DTD Constraints. ICDT 2003, this volume, `http://citeseer.ist.psu.edu/wood03containment.html`
9. Rajasekar Krishnamurthy, Raghav Kaushik, and Jeremy F. Naughton, " XML-to-SQL Query Translation Literature: The State of the Art and Open Problems" `http://www.cs.wisc.edu/~sekar/`
10. T. Milo, D. Suciu, and V. Vianu, Typechecking for XML Transformers. In PODS 00, May 2000. To appear, `http://sherry.ifi.unizh.ch/milo00typechecking.html`
11. W. Martens and F. Neven, Typechecking Top-Down Uniform Unranked Tree Transducers. Manuscript, `http://sherry.ifi.unizh.ch/554628.html`

Reasoning in multi-agent systems*

Ivana Budinská and Baltazár Frankovič

Institute of Informatics, Slovak Academy of Sciences
Dubravska cesta 9, 845 07 Bratislava, Slovakia

Abstract. The paper describes amulti-agent support system for discrete processes control design. The system is intended for discrete production processes. The proposed support system consists of several intelligent agents. The core of the systems is created by adatabase of modelling, control and simulation tools (algorithms) for discrete manufacturing processes and an intelligent decision system. The goal is to utilize information from the system's database and support aprocess control design on the basis of designer's requirements. The system consists of three modules: modelling, simulation and control.The intelligent decision system provides relevant reasoning about suitable algorithms and tools in respect of designer's requirements and description of a process.

Keywords: multi-agent systems, intelligent control, discrete processes, knowledge management

1 Introduction

Recently, an importance for computer aided decision support has raised rapidly. Decision support systems for producion systems have to involve three phases of production system control design. In the first phase, amodel of production system has to be constructed; next phase is a development of acontrol algorithms; and final phase is to simulate and verify the proposed solution. This procedure determines the basic architecture of the support system. There are many problems connected to the system development. For start, the knowledge has to be represented in such aform that is either human or computer readable. For second, all data and knowledge about specific domains has to be stored, archivated and organized for future reuse. Next, there is importance to create new knowledge on the base of stored and archivated information. The scope of the paper is to present aMA architecture to handle knowledge and to provide decision support for discrete processes modeling, control and simulation.

* The research in this area has been partially supported within the Slovak national projects NAZOU (SPVV 1025/2004), MARABU (APVT 51 011602), RAPORT (APVT-51-024604), VEGA 2/1101/21, and the IST project K-Wf Grid (EU RTD FP6-511385).

2 Architecture of the system

The MAS consists of four modules. Each of them is created by one ore more agents. Fig 1 depicts a general architecture of the system modules.

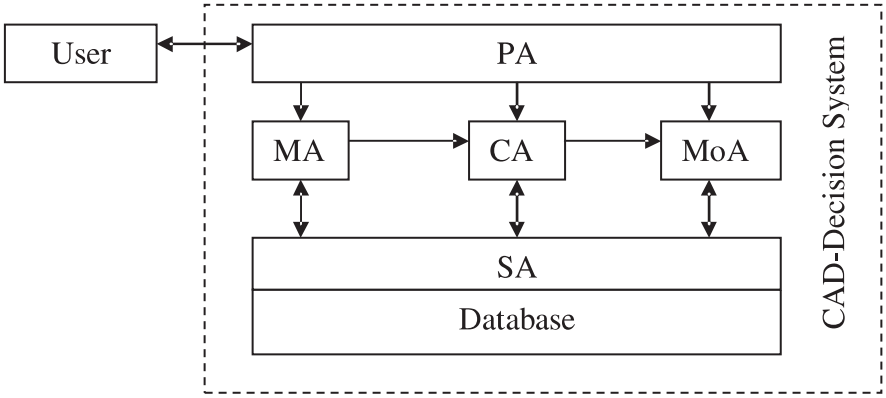


Fig. 1. Intelligent agents in the support system for control systems design.

The system consists of a modeling agent (MA), controll design agent (CA), and monitoring agent (MoA) A personal agent (PA) enables communication between users and the system. Agents have access to the central database via search agent (SA). Central database stores list of methods and tools for modelling and control design of production proecsses, either continuous or discrete. The database stores also problem's descriptions and thair solutions, as it is described later in the paper. The central control module is thecore of the system. It provides intelligent interface for users, interface between other three modules, in addition, searching and browsing functions on the basis of case-based and rule-based reasoning (CBR, RBR) [1] for performing actions with the aim to provide users relevant and valuable information.

3 Knowledge management and reasoning

In case-based reasoning (CBR) systems expertise is embodied in a library of past cases, rather than being encoded in classical rules. Each case typically contains a description of the problem, plus a solution and/or the outcome. The knowledge and reasoning process used by an expert to solve the problem is not recorded, but is implicit in the solution.

To solve a current problem: the problem is matched against the cases in the case base, and similar cases are retrieved. The retrieved cases are used to suggest a solution, which is reused and tested for success. If necessary, the solution is then revised. Finally the current problem and the final solution are retained as part of a new case.

Case-based reasoning is liked by many people because they feel happier with examples rather than conclusions separated from their context. A case library can also be a powerful corporate resource, allowing everyone in an organization to access the corporate case library when handling a new problem.

In CBR support systems advices are provided on the basis of users requirements. User formulates her/his requirements through intelligent interface. User's requirements create a special case for which the system has to advice a solution (a model, control algorithm and simulation of results). The system searches for similar cases in databases, because it is very unlikely that there is the same case in the database as the user has formulated. Searching for similar cases can be based on rules (rules for similar cases must be formulated very general to find appropriate solution) or on the base of similarity degree. Rule based reasoning seams to be easier for application but on the other hand it is very difficult to specify good rules that are general enough and to find not very many cases in the database. The rules have to be defined in the beginning of the system work and it is not easy to add new rules into the system while working. Case based reasoning involves developer to define some principles how to evaluate which case is similar enough.

All case-based reasoning methods have in common the following process:

- retrieve the most similar case (or cases) comparing the case to the library of past cases;
- reuse the retrieved case to try to solve the current problem;
- revise and adapt the proposed solution if necessary;
- retain the final solution as part of a new case.

There are a variety of different methods for organizing, retrieving, utilizing and indexing the knowledge retained in past cases.

Retrieving a case starts with a (possibly partial) problem description and ends when a best matching case has been found. The subtasks involve:

- identifying a set of relevant problem descriptors;
- matching the case and returning a set of sufficiently similar cases (given a similarity threshold of some kind); and
- selecting the best case from the set of cases returned.

Retaining the case is the process of incorporating whatever is useful from the new case into the case library. This involves deciding what information to retain and in what form to retain it; how to index the case for future retrieval; and integrating the new case into the case library.

4 Case-based reasoning in a support system for modeling, simulation and control design of dynamic system

CBR is built on a common principle: new situations are solved by exploiting and adapting solutions, which were used in similar past situations. The most important aspects of historical situations are indexed and stored in a common database. New situations are described by using some basic attributes; afterwards,

the similar, existing situations are identified and extracted from the knowledge base. Finally, the previous problem solutions are retrieved and the revised solutions are proposed for the current situation.

The first important requirement for the support system is to build a database for storing information from numerous situations and their solutions. The second requirement is to define a set of rules for classification and extraction of data; and the last one is to design a method for decision making, so that the agents can identify what is best for the user in the current situation. Due to the limited length of this paper, we shall deal with two first problems. The third problem related to decision-making algorithms will be considered in the future.

4.1 Case library for CBR

A case library is a relational database, which involves the important features of each historical situation [2]. Because the system has to work with different systems and kinds of information, the case library must be generic enough to represent all systems and their associated information. Determining which information is essential for storing is too difficult, since it is impossible to record all information and, on the other hand, because of practical reasons, the database should not be too large.

Other problems related to the database include the format used for data representation, e.g., number, text, graph, etc., and the method used for encoding/decoding these data. In the system, historical cases are represented by a number of basic attributes, which are indexed in a hierarchical scheme. These attributes are expressed by number or text (the example shown in Figure 2).

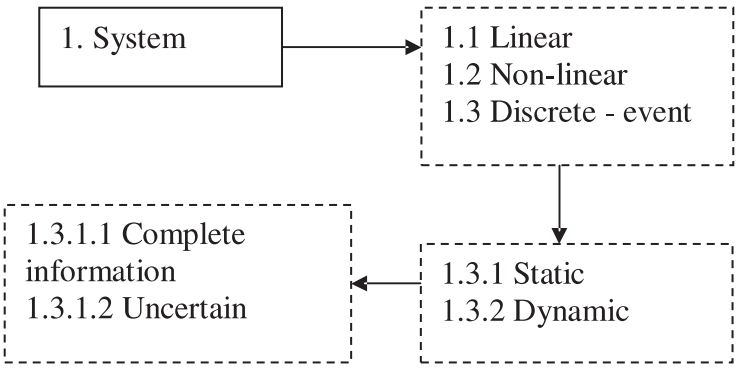


Fig. 2. Example of data representation.

The main database consists of two parts. The first part stores a description of all historical situations provided by the experts or added by users during customization of the CAS-Decision System. A description of past situation consists

of a number of basic attributes defined in advance. For example, situation si is a combination of a number of attributes as follows:

Let EL be a set of all possible attributes.

$si = \{el_1 \cup el_2 \cup ... \cup el_n\}; \forall i \neq j \in [1, n]; el_i \neq el_j \in EL$

In this representation, all the attributes are different each from other and they are sorted in a fixed order in order to facilitate search and retrieval.

The second part implies all solutions associated with the cases stored in the first part of the database. Solutions mean which method was used for modeling, designing the control system or simulation.

4.2 Case entry - the user's interface

The manner that the user describes the given situation should be standardized. All the users exploit the same manner for entering the required information. It allows the agents saving a lot of time of classifying and translating input data to an appropriate form. For example, first, the user should enter a type of system like in Figure 2 (linear, discrete, etc.), afterwards, the attributes expressing system dynamics, and so on. All these operations are performed by a PA, which provides the user with a feasible interface for this purpose.

4.3 Reasoning algorithms for extracting information

Let us assume that the user sends the system a description of the current situation (through a PA and in a correct form, which is understandable for agents). The first thing that the agents have to do is to find a similar situation in the library, afterwards, to revise and adapt the solutions applied in the selected case to the current one. Due to the limited scope of this paper, extracting past solutions and their retrieval to adapt to the current case are omitted.

To find a similar case, the agents can use queries to search. Since a relational database is considered to be used, the similar case could be found in a time linear with the number of cases stored in the database. However, the dilemma is how to achieve a case that is identical with or very close to the current one described by the user. Two cases are identical if all their attributes are the same; unfortunately the existence of the identical case is rather rare. In many instances, the agents can find only a case that is "very close" to the target one. Exploiting the similarity degree and an inductive reasoning method can help the agents distinguish situations and choose one of them that are closest to the new instance.

The first method works on the following principle: the agents calculate the similarity between two arbitrary situations by comparing all their attributes. Relationships among attributes are evaluated by any number (real or fuzzy) that reflects how much a solution of one situation is useful for the second one, with respect to these attributes. The similarity degree is defined by a combination of those partial relationships.

The second possible method is an inductive reasoning that works as follows: starting with the most important attribute, the agents sort and filter all cases

that have the same or to a certain degree the same attribute like the target one. This cycle continues with less important attributes, until only one candidate remains. The last case remained is considered as the most similar to the target one.

The important requirement of the both methods is the identification of relationships among attributes. Due to awide variety of attributes, classifying precisely the dependences among them is impossible. For that reason, the use of afuzzy classification is preferred, because of its capabilities to express anumber of difficult relations among attributes that other classification methods cannot do. For example, the following relation: "an algorithm for modeling system type Acould be applied to system type B with the same or very good quality", could be easily expressed by exploiting fuzzy sets.

Let $re(el_1, el_2) : \{ELXEL\} \rightarrow [0, 1]$ be a relation expressing thedependence between two attributes $(el_1, el_2) \in EL$ There are some important properties of this relation.

- $re(el_1, el_2) \neq re(el_2, el_1)$ resp. $\neq (1 - re(el_2, el_1))$ i.e. this relation is not symmetric or inverse.

- $re(el_1, el_2) = 1$ when the solution proposed for element el_2 could be applicable to element el_1 without changes. For example, el_1 is a linear system with complete information; el_2 is a linear system with parametrical uncertainty.

- $re(el_1, el_2) = 0$ when both the elements have disjoint domains of effects, i.e. the solution proposed for el_1 is useless for element el_2. For example, el_1 is a discrete event system and el_2 is a linear system.

- $\forall el \in EL; re[\emptyset, el] = 1$ and $re[el, \emptyset] \geq 0$

The similarity degree between two arbitrary situations $Sim(,)$ is defined as follows:

Let us consider two situations si_1 and si_2 with the same number of attributes. $si_1 = \{el_{1,1}, el_{1,2}, ..., el_{1,n}\}$ and $si_2 = \{el_{2,1}, el_{2,2}, ..., el_{2,n}\}$

If the numbers of attributes of each situation are not equal, we can add an empty attribute $\emptyset$ to them. Let us define matrix $W \in [0, 1]^{n \times n}$ as follows:

(1) $w_{i,j} = re(el_{1,j}, el_{2,j})$

Then,

(2) $Sim(si_1, si_2) = W^T Q W$

Where $Q = \{q_{i,j}\}^{n \times n}$ is a symmetric and positively definite matrix of the same dimension $(n \times n)$ as W. Each member $q_{i,j}$ of matrix Q expresses a weight of how much a relation $re(el_{1,i}, el_{2,j})$ can influence the dependence between both situations. Given a target case $si_{current}$ the case that maximizes a function $Sim(si, si_{current})$ is considered as the most similar one, and its solution could be applied or adapted to solve the current situation.

Inductive reasoning method extracts the desired situation by performing a search of a decision tree, which involves all possible historical cases satisfying certain conditions. A decision tree is generated as follows: starting with the most important attribute - propose it is el_1, all cases that satisfy the following conditions are added to the decision tree.

$re(el_1 \mid_{(si)}, el_1 \mid_{(si_{current})}) \geq \alpha$
where coefficient α is a low bound that is used to restrict a set of candidate situations. The classification process continues with less important attributes, until only one candidate solution remains.

After finding a similar situation to the current one, the agents extract the solution that was used in this case (a method for modeling, designing the control system and simulation) and retrieve it appropriately to a solution for the current situation. The user accepts the recommendation and applies it into a practice. The last step that the CAS-Decision System has to do is to observe the real behavior of the system after the user has applied all the methods proposed by the agents. The results achieved by an observation are intended to evaluate the selected methods and for later use.

5 Conclusion

The paper introduces amulti-agent architecture of asupport system for modelling, control and simulation of dynamic systems. The detailed description of amodel for CBR is given. The presented CBR algorithms may be used also in another knowledge management systems, e.g. for knowledge management for administration Processes [5].

References

1. http://www.aiai.ed.ac.uk/links/cbr.html
2. http://wwwagr.informatik.uni-kl.de/ bergmann/wimo/ Modeling of knowledge for CBR
3. Dang T-Tung, Improving Plan Quality through Agent Coalitions, IEEE International Conference on Computational Cybernetics - ICCC'04, Vienna, Austria, 2004, ISBN 3-902463-02-3, 171–176
4. Frankovic B., Budinska I., Single and Multi-Machine Scheduling of Jobs in Production Systems. In: Tzafestas, S.G. (Editor): Advances in Manufacturing. Decision, Control, and Information Technology. Springer Verlag, London, 1998
5. Laclavik M., Balogh Z., Hluchy L., Krawczyk K., Dziewierz M., Kitowski J., Majewska M., Knowledge Management for Administration Processes In: Proceedings of Znalosti 2004, February 2004, VSB-Technick univerzita Ostrava, ISBN 80-248-0456-5, 248–255.

Aplikácia zhlukovania fuzzy konceptov v doméne textových dokumentov

Peter Butka

Katedra kybernetiky a umelej inteligencie, Fakulta elektrotechniky a informatiky
Technická univerzita v Košiciach, Letná 9, 041 20 Košice, Slovenská republika
Peter.Butka@tuke.sk

Abstrakt Jednou z možností spracovania tabuľkových dát je použitie formálnej konceptovej analýzy (FCA). Jej možnou aplikáciou je identifikácia zhlukov podobných objektov v množine dát. Výpočtová zložitosť tvorby formálnych konceptových zväzov s veľkých vstupných kontextov (množiny textových dokumentov) je značná, preto má zmysel hľadať metódy pre efektívnejšie generovanie konceptov, napr. kombináciou s inými algoritmami a použitím heuristík. Rozšírením klasickej FCA je verzia pracujúca s reálnymi hodnotami v objekt-atribútovej tabuľke, jedna z existujúcich možností takejto fuzzifikácie bola použitá v tejto práci. Následne popisujeme aplikáciu aglomeratívneho zhlukovania nad fuzzy konceptami v doméne textových dokumentov (písaných v anglickom jazyku). Algoritmom nájdené zhluky sú na konci tohto procesu popísané množinou charakterizujúcich termov (slov). Pre prácu z dokumentmi bola použitá nami navrhnutá knižnica JBOWL pre spracovanie a reprezentáciu textových dokumentov.

1 Úvod

Zhlukovanie a formálna konceptová analýza (FCA) sa používajú pri hľadaní vzorov v dátach. Zhlukovanie zoskupuje objekty s navzájom podobnou charakteristikou do klastrov (zhlukov). FCA identifikuje podobnosť medzi množinami objektov na základe ich spoločných atribútov. Problémom v zhlukovaní je často výber vhodnej podobnostnej metriky, najvýraznejším problémom konceptovej analýzy je obrovské množstvo konceptov a časovo veľmi náročné (často prakticky nerealizovateľné) spracovanie.

Jednoduché prepojenie klasickej FCA a zhlukovania nad binárnymi dátami, tak ako ju zaviedli Ganter a Wille v [1], popísali v [2] Rice a Siff. Definovali špeciálnu vzdialenostnú (pseudo)metriku pre prípad klasického "crisp"konceptového zväzu na binárnej objektovo-atribútovej tabuľke dát. Navrhnutú metriku použili pre aglomeratívne zhlukovanie konceptov, čím sa značne redukuje počet konceptov (zhlukov).

V [3] autor ukazuje aplikáciu tohto prístupu pre určitý prípad fuzzy zovšeobecnenia konceptového zväzu. Vstupom je tabuľka s reálnymi hodnotami, autor upresňuje postup pri fuzzifikácii takéhoto prípadu, definuje potrebné zobrazenia a vzdialenostnú funkciu pre tento prípad.

Táto práca popisuje aplikáciu uvedeného prípadu v doméne textových dokumentov. Pre experimenty bola použitá množina 420 článkov z denníka Times z obdobia 60-tych rokov minulého storočia. Jedným z problémov pri textových množinách je interpretácia nájdených zhlukov. Pre jej uľahčenie sme na nájdené koncepty aplikovali metódu LabelSOM [5]. Pôvodne bola používaná pre nájdenie charakteristických termov zhlukov v samoorganizujúcich sa mapách (ako naznačuje názov). Je to však metóda skúmajúca odchýlky vektorov (variabilitu) príkladov v zhluku od centroidu zhluku, preto sa dá ľahko aplikovať aj tu. V našom prípade objekty tvoriace koncept (zhluk) sú jeho príklady a aritmetický priemer ich váhových vektorov je pre algoritmus LabelSOM centroidom. Pre prácu s textovými dokumentmi a implementáciu algoritmov bola použitá nami implementovaná knižnica JBOWL [6].

V kapitole 2 je stručne charakterizovaný uvažovaný prípad fuzzifikovaného konceptového zväzu, v ďalšej kapitole je definovaná použitá vzdialenostná funkcia a zhlukovací algoritmus. Obidve tieto kapitoly náplňou vychádzajú z [3]. Vo štvrtej kapitole je možné nájsť krátky popis algoritmu LabelSOM. Nasledujúca kapitola sa venuje implementácii celého procesu a experimentom s textovými dokumentmi. Krátke zhrnutie a myšlienky pre ďalšiu prácu sa nachádzajú v záverečnej kapitole.

2 Fuzzy konceptový zväz

Majme množinu atribútov A, množinu objektov B a fuzzy reláciu R, pričom $R : A \times B \to [0,1]$. Túto reláciu potom reprezentuje tabuľka dát, kde $R(a,b)$ vyjadruje hodnotu atribútu b objektu a.

Definujme zobrazenie $\tau : P(B) \to [0,1]^A$, ktoré priradí každej množine X prvkov z B nejakú funkciu $\tau(X)$, hodnotou ktorej v bode $a \in A$ je:

$$\tau(X)(a) = \min\{R(a,b) : b \in X\}, \tag{1}$$

t.j. táto funkcia priradí každému atribútu aspoň takú hodnotu, akú minimálne dosahujú všetky objekty z X.

Definujme ďalej zobrazenie $\sigma : [0,1]^A \to P(B)$, ktoré priradí každej funkcii $f : A \to [0,1]$ množinu:

$$\sigma(f) = \{b \in B : (\forall a \in A)\, R(a,b) \geq f(a)\}, \tag{2}$$

t.j. objekty ktoré "dominujú"danej funkcii f v každom atribúte.

Je ľahké ukázať, že tieto zobrazenia majú nasledovné vlastnosti:

$$X_1 \subseteq X_2 \to \tau(X_1) \geq \tau(X_1) \tag{3}$$

$$f_1 \geq f_2 \to \sigma(f_1) \subseteq \tau(f_1) \tag{4}$$

$$X \subseteq \sigma(\tau(X)) \tag{5}$$

$$f \subseteq \tau(\sigma(f)) \tag{6}$$

Potom dvojica zobrazení $< \tau, \sigma >$, ktorá spĺňa tieto podmienky, tvorí tzv. Galoisovu konexiu.

Definujme zobrazenie $cl : P(B) \rightarrow P(B)$ ako kompozíciu zobrazení τ a σ, t.j. nech pre každú množinu X je $cl(X) = \tau(\sigma(X))$. Z predchádzajúcich vlastností spájaných zobrazení je možné ľahko vidieť, že cl je uzáverovým operátorom, t.j. spĺňa nasledovné podmienky:

$$X \subseteq cl(X) \tag{7}$$

$$X_1 \subseteq X_2 \rightarrow cl(X_1) \subseteq cl(X_2) \tag{8}$$

$$cl(X) = cl(cl(X)) \tag{9}$$

Hlavnú úlohu pri identifikácii konceptov (rovnako ako v "crisp"prípade) hrajú také podmnožiny objektov $X \subseteq B$, pre ktoré je $X = cl(X)$. Potom pár $< X, \tau(X) >$ sa nazýva *fuzzy koncept*, pričom X je extentom a $\tau(X)$ intentom daného konceptu. Potom množina L všetkých extentov, t.j. $L = \{X \in P(B) : X = cl(X)\}$, usporiadaná inklúziou je zväz, v ktorom sú operácie definované nasledovne: $X_1 \wedge X_2 = X_1 \cap X_2$ a $X_1 \vee X_2 = cl(X_1 \cup X_2)$. Tento zväz potom nazývame *fuzzy konceptovým zväzom*.

3 Vzdialenostná funkcia a zhlukovací algoritmus

Pre potreby zhlukovania je potrebné definovať vzdialenostnú funkciu (metriku), podľa ktorej je možné určiť navzájom blízke koncepty. Autor v [3] uvádza funkciu, ktorá zovšeobecňuje (pseudo)metriku dvojice autorov Rice a Siff pre fuzzy prípad:

$$\rho(X_1, X_2) = 1 - \frac{\sum_{a \in A} \min\{\tau(X_1)(a), \tau(X_2)(a)\}}{\sum_{a \in A} \max\{\tau(X_1)(a), \tau(X_2)(a)\}} \tag{10}$$

Ďalej autor ukazuje, že ρ nadobúda hodnoty z intervalu [0,1], je pseudometrikou na $P(B)$ a metrikou na L. Explicitný dôkaz je možné nájsť v [4].

Zhlukovací algoritmus je ten istý, ktorý použili Rice a Siff v [2], akurát použijeme uvedenú funkciu ρ pre fuzzy prípad. Pseudokód algoritmu vyzerá nasledovne:

$C \leftarrow D \leftarrow \{cl(\{b\}) : b \in B\}$
while $(\mid D \mid > 1)$ *do*
{
  $m \leftarrow \min\{\rho(X_1, X_2) : X_1, X_2 \in D, X_1 = X_2\}$
  $\varepsilon \leftarrow \{< X_1, X_2 > \in D \times D : \rho(X_1, X_2) = m\}$
  $V \leftarrow \{X_1 \in D : (\exists X_2 \ \in D) < X_1, X_2 > \in \varepsilon\}$
  $N \leftarrow \{X_1 \vee X_2 :< X_1, X_2 > \in \varepsilon\}$
  $D \leftarrow (D \setminus V) \cup N$
  $C \leftarrow C \cup N$
}

Premenná D je množina konceptov (zhlukov) v aktuálnej iterácii, C je zjednotenie všetkých množín D od začiatku po aktuálnu iteráciu. Premenná m je minimálna vzdialenosť dvojíc z D. Potom množina ε obsahuje "hrany", t.j. dvojice klastrov z D, ktorých vzdialenosť je m, V obsahuje "vrcholy"(konce hrán). Prvky v N sú nové zhluky, nová iterácia nahradzuje v D prvky z V prvkami z N.

4 Algoritmus LabelSOM

Tento prístup k (všeobecne) centroidným modelom sa snaží určiť tie zložky centroidného vektora, ktoré sú najdôležitejšie pri mapovaní vstupných vektorov na dané centrum. To sa deje určením veľkosti prispenia každej zložky vektora voči celkovej Euklidovskej vzdialenosti medzi vstupným vektorom a váhovým vektorom centra.

Metóda LabelSOM [5] je založená na pozorovaní, že po naučení SOM-u sa zložky vektora váh neurónu čo najviac podobajú odpovedajúcim zložkám vstupného vektora vo všetkých vstupných signáloch, ktoré boli namapované na tento jeden neurón, ako aj na niektoré zložky vektorov susediacich neurónov. Zložky vektora, ktoré majú približne tú istú hodnotu v rámci celej množiny vstupných vektorov namapovaných na konkrétny neurón, popisujú a pomenúvajú vlastne príznaky všetkých vstupných dát tohto neurónu. Preto môžeme tieto zložky vektora váh považovať za popis daného neurónu. Kvadratická odchýlka pre všetky jednotlivé príznaky slúži ako miera pre ich relevantnosť ako názvy tried. Vyberú sa tie zložky vektora, ktorých hodnota chyby je blízka nule. Kvadratická odchýlka vektora sa vypočíta pre každý neurón i ako priemerná vzdialenosť medzi zložkami vektorov všetkých vstupných dát namapovaných na neurón i a zložkami vektora váh tohto neurónu. Aplikácia tohto prístupu v našom prípade je jednoduchá, príklady určitého konceptu ("neurónu") sú priamo jemu prislúchajúce objekty a váhový vektor konceptu (centroidu, "neurónu") je aritmetický priemer váhových vektorov daných objektov konceptu.

Nech C_i je množina objektov x_j konceptu i. Vypočítaním vzdialeností pre všetky zložky vektora k cez všetky vektory x_j ($x_j \in C_i$) dostaneme kvadratickú chybu vektora q_i pre všetky koncepty i:

$$q_{ik} = \sum_{x_j \in C_i} \sqrt{(m_{ik} - x_{jk})^2}\,, \quad k = 1, \ldots, n \tag{11}$$

Vybraním tých zložiek vektora, ktorých hodnota chyby je blízka nule sa získa zoznam atribútov, ktoré sa nachádzajú vo veľkom počte vstupných vektorov (dokumentov) namapovaných na tento neurón a preto popisujú charakteristiky dát pre daný zhluk. Tieto atribúty slúžia ako názvy resp. popisy (labels) jednotlivých konceptov.

Tento postup však pri textových dokumentoch naráža na problémy. Kvôli veľkej dimenzionalite príznakového priestoru a charakteristike reprezentácie príznakových vektorov dokumentov sa zvyčajne vyskytuje veľký počet zložiek vstupného vektora, ktorých hodnota je nulová (existuje veľa termov - slov, ktoré nie sú obsiahnuté v danej skupine dokumentov). Tieto termy majú obvykle hodnotu kvadratickej chyby rovnú nule a preto by boli vybrané ako charakteristické termy. Dôsledkom by bolo, že neurón by bol prakticky popísaný len termami, ktoré sa v daných dokumentoch ani nevyskytujú (samozrejme vzhľadom na daný zhluk). Úlohou analýzy textov je však popísať zhluky pomocou príznakov, ktoré sú prítomné v danom zhluku.

Riešenie je určiť tie zložky váhového vektora, ktoré na jednej strane preukazujú približne tie isté hodnoty pre všetky objekty špecifického konceptu a na druhej strane majú vysokú celkovú hodnotu váhového vektora, čo indikuje ich dôležitosť. Na dosiahnutie toho je potrebné definovať prah t, aby boli vybrané iba atribúty s veľmi nízkou hodnotou kvadratickej chyby, ktoré vykazujú hodnotu váhového vektora nad prahom t. V tomto prípade je t vlastne minimálna hodnota (dôležitosť) atribútu potrebná na to, aby bol vybraný pre popis zhluku. Zhluky sú potom prehľadne popísané charakterizujúcimi slovami, ktoré z konceptov vyextrahoval LabelSOM.

5 Experimenty s kolekciou textových dokumentov

Pre naše experimenty sme vybrali kolekciu 420 článkov v anglickom jazyku z denníka Times zo 60-tych rokov minulého storočia. Príklad časti dokumentu z kolekcie:

COMMUNISTS GETTING TOGETHER ?

-DESPISE THE ENEMY STRATEGICALLY, "WROTE MAO TSE-TUNG IN HIS HANDBOOK FOR REVOLUTIONARIES, -BUT TAKE HIM SERIOUSLY TACTICALLY . -LAST WEEK, DRAWING A BEAD ON HIS ENEMY IN THE KREMLIN, MAO DID BOTH . HE MADE HIS TACTICAL POINT BY ACCEPTING MOSCOW'S CALL FOR BILATERIAL NEGOTIATIONS OVER THE SINO-SOVIET RIFT AND INVITING NIKITA KHRUSHCHEV TO PEKING . -THE COMMUNIST MOVEMENT HAS REACHED A CRITICAL JUNCTURE . THE TIME HAS COME WHEN DIFFERENCES HAVE TO BE SETTLED, "DECLARED THE RED CHINESE, PROPOSING THAT RUSSIA'S PREMIER STOP OFF IN PEKING ON THE WAY TO CAMBODIA, WHERE A STATE VISIT BY KHRUSHCHEV HAS BEEN DISCUSSED FOR SOME TIME. ALTERNATIVELY, SUGGESTED MAO, A RED CHINESE DELEGATION COULD GO TO MOSCOW TO DISCUSS THE SQUABBLE . BUT THE HAUGHTY, PATRONIZING TONE OF THE MESSAGE, OFFERING TO MEET THE SOVIETS ONLY -ON THE BASIS OF FULL EQUALITY, ÄND REPEATING THE OLD DEMANDS FOR A ONE-WAY COMPROMISE, WOULD MAKE IT TOUGH FOR RUSSIA TO ACCEPT . EVEN IN THE EVENT OF A MEETING, SAID ONE HONG KONG EXPERT, -WE CAN EXPECT LITTLE BUT A VERBAL AGREEMENT WHICH WILL SOON BREAK DOWN .

5.1 Predspracovanie

Predspracovanie textových dokumentov v našom prípade pozostávalo z nasledujúcich krokov:

1. Tokenizácia
2. Eliminácia neplnovýznamových slov
3. Úprava slov na základný tvar (stemming)
4. Výber termov na základe ich frekvencie výskytu

V prvom kroku (tokenizácia) bol vstupný text transformovaný na tokeny. Tokenizácia je nevyhnutnou súčasťou predspracovania, ostatné kroky sú voliteľné (používajú sa kvôli redukcii priestoru termov).

Získané tokeny predstavujú základný slovník spracovávanej množiny dokumentov. V nasledujúcom kroku sú z tejto množiny odstránené neplnovýznamové slová (stopwords - spojky, častice, zámená, ...) porovnaním so zoznamom slov v tzv. stop-liste.

V ďalšom kroku sú zostávajúce tokeny transformované na ich základné tvary - stemy (ako napríklad "wins", "win"sú transformované na jeden koreň - "win"). Dosiahneme tak opätovné zmenšenie slovníka (pre rôzne morfologické tvary zostane len akýsi koreň slova - stem). Pre naše účely sme pri stemovaní použili pre anglický text známy Porterov algoritmus [7], založený na orezávaní sufixov.

Posledný krok na základe frekvencie výskytu slov takto upraveného slovníka odstráni tie slová, ktoré sa vyskytovali v texte v príliš malom počte dokumentov. Preto sa ponechajú len tie tokeny (termy), ktorých frekvencia výskytu dosiahla aspoň minimálnu zvolenú hodnotu frekvencie výskytu. Skúmaná kolekcia dokumentov je potom reprezentovaná pomocou dokument-term matice. Pre váhovanie termov sme použili *tfidf schému*, ktorá priradzuje každej dvojici dokument-term reálne číslo (príslušnú váhu) nasledovne:

$$w_{ij} = tfidf_{ij} = tf_{ij} \log\left(\frac{ndocs}{df_i}\right), \tag{12}$$

kde tf_{ij} je frekvencia výskytu j-tého termu v i-tom dokumente, $ndocs$ je počet dokumentov v kolekcii a df_i je počet dokumentov obsahujúcich daný j-tý term (dokumentová frekvencia). Vektory boli následne normalizované na jednotkovú dĺžku. Z takto (pomocou knižnice JBOWL) vytvorených vektorov sme vytvorili vstupnú objekt-atribútovú (dokument-term) maticu (formálny fuzzy kontext), pričom jej reprezentácia je vo forme množiny "riedkych"vektorov.

Minimálny prah pre frekvenciu bol výskyt termu aspoň v troch dokumentoch (po orezaní zostalo 5779 termov), maximálna hodnota prahu bola 40 (po orezaní 405 termov).

5.2 Nastavenie parametrov

Pri takto používanom prístupe k hľadaniu konceptov (zhlukov) je parametrom po predspracovaní dokumentov do podoby dokument-term matice už len prah pre výber termov v algoritme LabelSOM, tento sa v experimentoch pohyboval v intervale $< 0.1, 0.7 >$, v dôsledku toho sa počet termov popisujúcich jednotlivé koncepty pohyboval do 10 termov (pre prah nastavený na 0.7), resp. viac než 50 termov (pre prah nastavený na 0.1). Rozumný (interpretovateľný) počet termov v tomto prípade vychádza pri nastavení prahu $t \geq 0.4$.

5.3 Generovanie výstupných súborov

V našej implementácii sú generované automaticky dva typy súborov:

1. Popis priebehu zhlukovania konceptov
2. Popis konceptov

Prvý typ súboru popisuje priebeh zhlukovania, t.j. na začiatku počet riadkov a stĺpcov matice, potom postupne každú úroveň hierarchie v aglomeratívnom procese zhlukovania a príslušnú minimálnu vzdialenosť v aktuálnej iterácii, na konci je celkový počet zhlukov.

Druhý typ súboru podrobnejšie popisuje nájdené koncepty vytvorené v procese zhlukovania charakterizujúcimi termami získanými pomocou algoritmu LabelSOM (použitím "stredov"konceptu ako centroidov) pri zvolenom prahu t. Potom popis jedného konceptu môže vyzerať nasledovne:

Concept(526):
IntSet 526(| 526 | = 2) = {107 212 }
Labels(526):
ideolog 0.11408524076826629
tactic 0.12226496392377192
delai 0.12486534805002342
kremlin' 0.14005905590662923
mao 0.2054933319030411

V tomto popise konceptu je prvom riadku v zátvorke číslo konceptu, v druhom riadku je uvedená mohutnosť konceptu, teda počet prvkov (IntSet je len pozostatok z implementácie, trieda pre prácu s indexami množín s vhodnými metódami a vlastnosťami, ktorá je v našej knižnici implementovaná) konceptu a výpis množiny indexov dokumentov v koncepte.

Nasleduje množina charakterizujúcich termov (labels) usporiadaných podľa váhy, ktorá vystihuje vypočítanú odchýlku testovaného termu (atribútu) voči "stredu"daného konceptu.

5.4 Príklad interpretácie

Problematickou časťou pri spracovaní dát pomocou formálnej konceptovej analýzy je interpretovateľnosť výsledkov, najmä pre veľké kontexty (čo je aj náš prípad). Pre uľahčenie tohto postupu sme sa rozhodli využiť LabelSOM a ním vyextrahované termy. V tejto podkapitole ukážeme príklad dvoch zhlukov a ich spojenia do väčšieho zhluku, ako aj pohľad na charakterizujúce termy týchto zhlukov (konceptov). Pri tomto príklade je prah t algoritmu LabelSOM rovný 0.5 (t.j. relatívne menší počet vybraných termov).

Prvým príkladom je zhluk dvoch dokumentov, ktorý bol vo výstupnom súbore popísaný nasledovne:

Concept(467):
IntSet 467(| 467 | = 2) = {6 59 }
Labels(467):
indian 0.08490372125158631
abdullah 0.08586436482593596
sheik 0.09099057480577845
provinc 0.12588705362934366
negoti 0.12735558187737947
nehru 0.1429329060272742
kashmir 0.14964545301943158
india 0.16980744250317262
pakistan 0.2006574773903399

Oba dokumenty v tomto zhluku popisujú snahu o dohodu medzi Pakistánom a Indiou v probléme Kašmíru. Ide o hľadanie priestoru pre rozhovory a dohody (mierové vyjednávania - negotiations). Popisuje sa problematika a zúčastnené strany, provincie v Kašmíre.

Concept(565):
IntSet 565(| 565 | = 2) = {180 416 }
Labels(565):
ocean 0.09490853266934317
nehru 0.0992100221716187
lawyer 0.10429760442209315
seventh 0.1142504215539828
due 0.11489799073203827
sea 0.11809352714594763
settlement 0.1351823936626069
india 0.14029765507194006
round 0.17453770888417258
indian 0.2104464826079101

Tento zhluk sa venuje bližšie Indii, pohľadu na prekvapujúci útok Číny, narušenie ich neutrality (ako aj aktiváciu americkej siedmej - seventh - letky nad Indickým oceánom) a schválenie pomerne vysokej čiastky na zbrojenie, schválenej predsedom vlády, ktorého meno je Nehru.

V procese zhlukovania sa tieto dva zhluky (koncepty) spojili do jedného, výsledkom bol nasledujúci popis:

Concept(666):
IntSet 666(| 666 |=4) = {6 59 180 416 }
Labels(666):
seventh 0.0856878161654871
sea 0.08857014535946073
area 0.09374258193410527

partit 0.09701606728320546
nehru 0.11550805298449338
negoti 0.12253821922301163
abdullah 0.12879654723890394
settlement 0.1351823936626069
sheik 0.13648586220866768
pakistan 0.1401009101054956
indian 0.1476751019297482
india 0.15505254878755634
delhi 0.16703038564665282
provinc 0.16806225759554166

Ako je vidieť, po spojení vypadli niektoré špecifické termy predchádzajúcich zhlukov, napríklad Kašmír, o ktorom sa v druhom koncepte nehovorilo.

5.5 Diskusia a problémy

Po týchto experimentoch je priestor pre formuláciu niekoľkých poznámok:

- Aglomeratívne zhlukovanie konceptov je z hľadiska budovania spodných vrstiev hierarchie konceptov zaujímavé, pri veľkých kontextoch je (okrem časovej náročnosti) však postupné posúvanie v hierarchii problematické a neprehľadné, zhoršuje sa aj možnosť interpretácie. Použitie algoritmu LabelSOM síce uľahčilo proces interpretácie, stále je však tento proces pri veľkom vstupnom kontexte značne subjektívny a nejednoznačný.

- Práca z textovými dokumentmi si vyžaduje dosť komplexný prístup k predspracovaniu, jednoduché postupy vedú k výrazným problémom pri interpretácii výsledkov.

- Vlastnosti klasickej vektorovej reprezentácie množiny textových dokumentov (veľmi riedky a silne nehomogénny priestor, klasické problémy z vysokou dimenzionalitou) spôsobujú veľké problémy pri použití implementovaných metód. Zhlukovanie konceptov pomocou min/max metriky je dokonca problematickejšie ako v prípade napr. modelu v podobe SOM-ov a klasických metrík pre vzdialenosť medzi dokumentmi, ich množinami a modelmi centroidov. V našom prípade dokonca kvôli vlastnostiam priestoru min/max metrika so zvyšujúcim sa počtom dokumentov v konceptoch degraduje na niekoľko diskrétnych hodnôt (samozrejme sa nevyhne ani klasickým problémom, ktoré majú aj iné metriky na vektorovej reprezentácii textov).

- Algoritmus LabelSOM je zameraný na prácu s centroidom ako strojovo naučeným modelom (naučené váhy neurónu), výpočet centroidu ako aritmetického stredu pravdepodobne nie je najvhodnejší postup pre jeho aproximáciu, možno by bolo zaujímavé vyskúšať práve prvé zo zobrazení (teda minimum) v kompozícii zobrazení z kapitoly 2.

6 Záver

Cieľom tejto práce bolo otestovať jeden prístup ku generovaniu fuzzy konceptov pomocou zhlukovania, ktorého účelom je zmenšenie počtu generovaných konceptov, a to na množine textových dokumentov. Ako testovacia množina bol použitý korpus 420 článkov denníka Times z obdobia 60-tych rokov minulého storočia. Pre uľahčenie a sprehľadnenie interpretácie sme na vytvorené zhluky (koncepty) aplikovali algoritmus LabelSOM (kde potrebným centroidom sa stal aritmetický priemer dokumentov konceptu) a získali tak charakteristické termy konceptu.

Naše experimenty potvrdili problémy pri práci s vektorovou reprezentáciou textov, vychádzajúce z vlastnosti priestoru a veľkosti kontextu. Použitá min/max metrika "zachováva"problémy iných metrík v takomto priestore, navyše sa objavil aj problém s jej nejednoznačnosťou pre rozlíšenie vzdialenosti konceptov pri zvyšovaní úrovne v rámci zhlukovania.

Do budúcnosti sa nám ako vhodnejšia alternatíva zdá práca s malými množinami podobných dokumentov (do 10), pričom tieto by boli získavané iným postupom, napr. pomocou hierarchického SOM-u. FCA by potom mohla pomôcť vybudovať z malých skupín dokumentov časti ontológie (hierarchie konceptov), ktoré by sa spätne v hierarchii SOM-ov spájali do väčších celkov.

Práca prezentovaná v tomto príspevku vznikla za podpory Vedeckej grantovej agentúry Ministerstva školstva SR a Slovenskej akadémie vied (VEGA) v rámci projektu č.1/1060/04 s názvom "Klasifikácia a anotácia dokumentov pre sémantický webä za podpory nemecko-slovenského vedeckého projektu DAAD č.8/2004 "Využitie objavovania znalostí v textoch pre extrakciu metadát a sémantické vyhľadávanie".

Referencie

1. Ganter B., Wille, R., Formal Concept Analysis. Mathematical Foundations, Springer Verlag, 1997
2. Rice M.D., Siff M., Clusters, Concepts and Pseudometrics. Preprint, `http://science.slc.edu/~msiff/papers/mfcsit2000.pdf`, 2000
3. Krajči S., Clustering Algorithm Via Fuzzy Concepts. In Proceedings of DATESO 2003 Conference, Desná, 2003, 94–100
4. Krajči S., Vektorová min/max metrika. In Proceedings of ITAT 2002, Malinô Brdo, 2002, 119–123
5. Rauber A., On the Labeling of Self-Organizing Maps. In Proceedings of IJCNN, Washington, 1999
6. Bednar P., Butka P., Paralic J., Java Library for Support of Text Mining and Retrieval. In Proceedings of Znalosti 2005, Stará Lesná, 2005, 162–169
7. Porter M.F., An Algorithm for Suffix Stripping. In V Program, 14(3), 1980, 130–137

Focused web crawling mechanism based on page relevance*

Emil Gatial, Zoltán Balogh, Michal Laclavík,
Marek Ciglan, and Ladislav Hluchý

Institute of Informatics, Slovak Academy of Sciences
Dúbravská cesta 9, Bratislava 84237, Slovakia
{emil.gatial, balogh.ui, laclavik.ui, marek.ciglan, hluchy.ui}@savba.sk

Abstract. In this article we propose a system for focused web crawling used in a domain of job offers. This work is performed within the scope of NAZOU project. Results of this paper will be focuse on algorithms for web page content analysis in order to specify the depth of crawling and identification of page relevance. An open source tool called Arachnid is used as a ground crawling technique for web page downloading and furthermore it is extended by the proposed decision algorithm that identifies whether the page belongs into the specified domain or not. In this article, the simple text analysis technique is described, which is based on keyword search from a specified domain, its synonyms and prescribed word orders. We conclude with the probability evaluation of the page relevance and other possible techniques that could enhance the crawling process.

1 Introduction

The web crawler is a program that automatically traverses the web by downloading the pages and following the links from page to page [1]. A general purpose of web crawler is to download any web page that can be accessed through the links. Contrary to this approach, a focused crawler tries to download only pages with specific topic to avoid the irrelevant web documents and reduce network traffic. The most recent scientific results in the domain of large scale web crawler engines come from techniques described for search engines such as Google, Yahoo, etc. One of the first focused crawlers was described in paper [2]. This focused crawler is based on the keyword relevancy evaluation and is composed of two hypertext mining programs: *classifier* and *distiller*. The *classifier* component evaluates the relevance of the page and the *distiller* identifies the hypertext links that points to many relevant pages. Another very interesting focused crawler was proposed by Cho [3]. The system processes the "most important" pages first for the efficiency reasons. Authors took into account various measures of importance for a page e.g. similarity to a driving query, number of pages pointing to this page

* This work is supported by the project K-Wf Grid EU RTD IST FP6-511385, VEGA No. 2/3132/23, APVT-51-024604 and SPVV 1025/2004.

(backlinks), pagerank and location (in a hierarchy). The pagerank defines the qualitative attribute that is evaluated as a weighted sum of the pageranks of the pages which point to it. There are three kinds of focused crawling strategies:

- BestFirst crawler: priority queue ordered by similarity between topic and page where link was found,
- PageRank crawler: crawls in pagerank order, recomputes ranks every 25th page and
- InfoSpiders: uses neural net, back propagation, considers text around links.

2 Overview of proposed crawler

Proposed kind of focused crawler is divided into two components. The first component crawls the web sites and makes decision about the depth of site crawling. It employs an Arachnid, an open source tool for web crawling, and implements the simple site mapping and decision algorithm. The second component computes the document relevancy according to the keywords given for particular domain. The components are tightly coupled to provide real functionality of focused crawler.

2.1 Web crawler component

As mentioned earlier this component uses Arachnid tool based on Java. It allows to define the steps after the page is downloaded by extending the interface of Arachnid. The crawler starts with the predefined URL list of initiation sites and set of keywords recorded as the regular expressions. The algorithm of focused web crawler is described in the following pseudo code.

```
0. Initialize the list of links on sites and setup the keywords.
   Pull out the link from the list of links on sites.
   Create empty list of links from local pages.
1. Use the link to download the page into memory. Find the links
   on the page and add them to the list of links from local
   pages.
2. If a page has been downloaded in the past, then check if
   the content of current page differs from the downloaded one.
   If it differs then go to next step, otherwise process another
   page (step 6).
3. Store the copy of page into the temporary storage and find
   keywords on the page. Build a relevancy ratio according to
   the found keywords.
4. Create the metadata about the downloaded page (i.e. URI of
   the page, date of download, page size, relevancy ratio,
   location in temporary storage, URL of parent site, etc.).
5. Make a decision whether to continue with crawling the pages
   on the next level. This decision depends on the rates of
```

relevant and irrelevant pages. If the parent doesn't contain links or it contains small amount of links on relevant pages then remove the links from the list and go to the next step.
6. If there is any link in the list of links from the local pages, then pull out the next link and go to the step 1 otherwise go to the next step.
7. If there is any link in the list of links on sites, then pull out the link and go to the step 1, otherwise go to the next step.
8. Delete the obsolete pages from the temporary storage.

Several basic functionalities must be implemented in focused crawler:

- Recognition of already visited pages,
- Detection of the page change and make updates,
- Page absence detection and invalidate the content of downloaded page and
- Calculation of the depth of crawling.

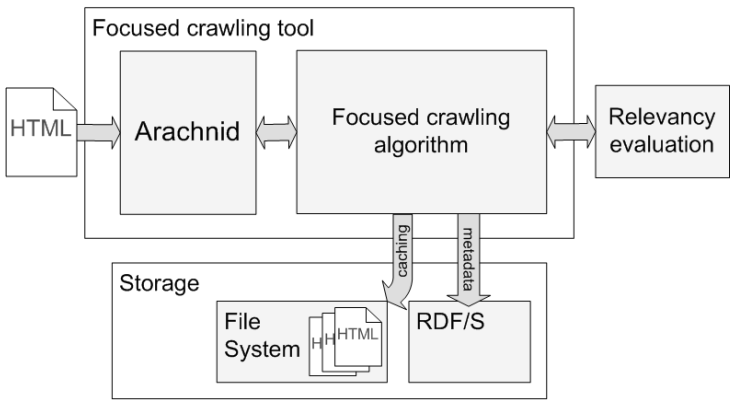


Fig. 1. Block schema of focused crawler.

The proposed focused crawler (Fig 1) writes copy of relevant page into the cache and writes metadata about successfully downloaded pages into an RDF database. Metadata include elements such as page URI, date and time of download, file name in a cache, and other metadata.

2.2 Relevance evaluation component

Internet offers many irrelevant information, therefore we are interested in the information processing of a specific domain. Furthermore, there is no rule for web page formatting and layout. To deal with such variety of sources, it must be

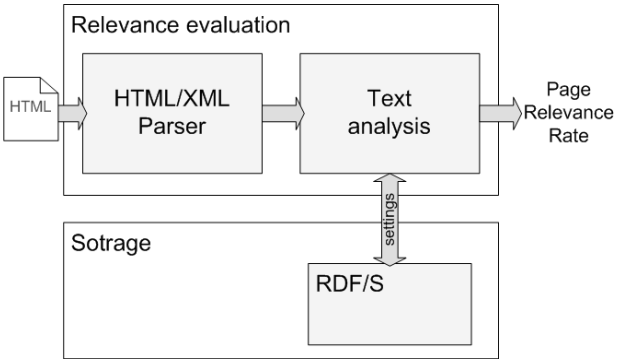


Fig. 2. Block schema of relevance evaluation component.

designed tool for page content processing that takes into account characteristic keywords relevant for given domain of use and its synonyms. The component (Fig 2) receives the page content for relevancy evaluation. It parses the content and tries to match the defined set of keywords. A single keyword is defined as a regular expression that describe format which identify (with some probability) whether the page falls into the investigated domain. Every keyword is coupled with the weight, that comes into the relevancy evaluation process of entire page.

3 Method of relevance evaluation

Relevancy evaluation of downloaded web page is the most important method in the process of focused crawling. The time of web site processing is highly influenced by the speed and quality of relevancy evaluation of a web page and therefore this is the critical part of web crawler. Method that evaluates the relevancy of downloaded page comprises the context identification, evaluation of particular weight for given keyword and calculation of page relevancy.

3.1 Context identification

The quality of relevancy evaluation is proportional to the dimension of dictionary, which stores relevant keywords. Therefore it is important to handle large dictionary of the most relevant keywords for given domain. For example, some keywords can be identified for the domain of job offers:

- "at least [\d]*" years experience : statement identified as an requirement for an applicant,
- "salary: [\d.]*" : statement for an offered salary

Moreover, the synonyms for specific keywords (e.g. salary, position, etc.) have to be defined and put into the page processing to process variety of pages. The dictionary of relevant words could change from time to time. Therefore the use

of a tool for automatic synonym update can enhance relevance evaluation. For example it could be created using the WordNet[4] dictionary located at CognitiveScienceLaboratory of Princeton University. Such tool should be executed in initialization phase or whenever the dictionary of relevant words is changed.

3.2 Weight of keyword

As it was mentioned earlier the rate of relevancy is computed as weighted sum of each occurrence of keyword in the content of page. Therefore, the component for relevancy evaluation must maintain the number of occurrences for particular keyword in all relevant pages and in all irrelevant pages. The weight value, that expresses the relevancy rate, could be simply evaluated as:

$$W(kw) = \frac{N_r(kw)}{P_r} - \frac{N_i(kw)}{P_i}\,, \tag{1}$$

where $W(kw)$ is relevancy weight for the keyword "kw", $N_r(kw)$ is number of keyword "kw" occurrences in pages evaluated as relevant, $N_i(kw)$ is number of keyword "kw" occurrences in pages evaluated as irrelevant, P_r is number of relevant pages and P_i is number of irrelevant pages.

The weight $W(kw)$ express the difference between the rates of keyword "kw" found in relevant and irrelevant pages. The $W(kw)$ shouldn't be greater than 1.0 provided that the keyword "kw" can be found in the content of relevant page only once. This method enables dynamically identify the relevant, neutral and irrelevant keywords. The keywords identified as the most irrelevant should be excluded from keyword set and substituted with others. The values of variables that come into W(kw) evaluation must be updated after every calculation of page relevance.

3.3 Calculation of page relevance

The rate of relevancy for particular page can be treated as sum of weights for the largest relevant group of keywords. It can be calculated according to the following formula:

$$R(page) = \frac{\sum_{kw \in K} W(kw)}{N_K(page)}\,, \tag{2}$$

where $R(page)$ is the relevancy rate of page for a given domain characterised by set of keywords, $W(kw)$ is weight of keyword "kw" and the $N_K(page)$ is number of keywords found in the content of page. The sum is evaluated for every keyword "kw" from the set of keywords K found in the page content.

The value of predefined threshold states whether the page is relevant or not. However, the value of $W(kw)$ depends on number of pages evaluated as relevant and irrelevant, the convenient value of threshold and the set of keywords could separate the miscellaneous pages into groups of relevant and irrelevant for a given domain.

Other relevance evaluation mechanisms, such as pagerank, backlinks, evaluation of location (in hierarchy) and other text processing techniques could be included into the calculation of page relevance.

4 Crawling strategy

The relevance evaluation is used, particularly, to make decision whether to download the page or not, but it can be used to influence the depth of crawling. The crawling algorithm walks through the links found in the content of the page, identifies pages that weren't yet put into crawling process and calculates the relevance for particular page. If most of the pages were recognised as irrelevant, the algorithm could stop the further crawling for that site. However, depth of crawling can be defined implicitly, but the described crawling strategy is able to exclude irrelevant sites and process relevant sites into much greater depth.

5 Conclusion and future work

The described focused crawling mechanism is able to process web sites and identify the relevant page according to its content text analysis. Moreover, the described crawling strategy could stop crawling process for irrelevant sites. The research, implementation and testing of crawling and page relevance evaluation algorithms are to be performed within the scope of NAZOU project on two pilot applications. The proposed focused crawler is designed to provide rough web pages and simple metadata for semantic annotation tool which is decribed in [5]. Such annotation tool produces information inevitable for presentation layer [6] which provides sophisticated features such as presentation in different languages and adaptive presentation of information.

References

1. Koster M.: The Web Robots Pages, 1999.
`http://info.webcrawler.com/mak/projects/robots/robots.html`
2. Chakrabarti S., van den Berg M., Dom B., Focused Crawling: A New Approach to Topic-Specific Web Resource Discovery. In Proceedings of the 8th International WWW Conference, Toronto, Canada, May 1999,
`http://www8.org/w8-papers/5a-search-query/crawling/index.html`
3. Cho J., Garcia-Molina H., Page L., Efficient Crawling Through URL Ordering. In Proceedings of the 7th International WWW Conference, Brisbane, Australia, April 1998,
`http://www7.scu.edu.au/programme/fullpapers/1919/com1919.htm`
4. Web page of WordNet project at CognitiveScienceLaboratory of PrincetonUniversity, `http://wordnet.princeton.edu/obtain`
5. Laclavík M., Balogh Z., Gatial E., Habala O., Nguyen G., Hluchý L., e-Collaboration and Knowledge Sharing based on Text Notes. In Proceedings of the 8th International Conference Informatics, Bratislava, Slovak Republic, June 2005, 151–154
6. Návrat P., Bieliková M., Rozinajová V., Methods and Tools for Acquiring and Presenting Information and Knowledge in the Web. In: Proc. International Conference on Computer Systems and Technologies - CompSysTech 2005, Varna 2005

Genetické učení závislostních analyzátorů*

Tomáš Holan

KSVI, Matematicko-fyzikální fakulta, Univerzita Karlova v Praze
Tomas.Holan@mff.cuni.cz

Abstrakt Použití genetických algoritmů pomohlo k nalezení parametrů jazykového modelu a zlepšení výkonu statistických syntaktických analyzátorů češtiny.

1 Úvod

Nejprve uveďme, že závislostním analyzátorem v tomto textu budeme rozumět program, jehož vstupem bude vždy jedna věta (obsahující ke každému slovu jednu morfologickou značku) a výstupem budou indexy závislostí udávající pro každé slovo index jeho nadřízeného slova, případně skutečnost (při hodnotě indexu rovné nule), že pro toto slovo nadřízené slovo neexistuje. Výstupem tedy není množina stromů ani právě jeden strom, výstupem jsou závislosti jednotlivých slov. Úspěšnost takového programu budeme měřit procentem správně určených závislostí.

V [5] a [6] jsme popsali několik pokusů o vytvoření českého statistického závislostního analyzátoru. Analyzátory jsou učeny na trénovací množině vět Pražského závislostního korpusu a na testovací množině vět téhož korpusu je vyhodnocována jejich úspěšnost měřená počtem správně určených závislostí.

Statistické analyzátory používají různé způsoby určování závislostí zachycujících syntaktickou strukturu věty, přičemž v jednotlivých krocích hrají roli pravděpodobnosti jevů odvozené z četností výskytu těchto jevů ve větách a stromech trénovací množiny.

Problémem všech statistických analyzátorů je nedostatečný počet dat a z něj plynoucí nutnost kompromisu mezi přesným popisem jevů, který ovšem trpí nedostatečnou četností popisovaných jevů v trénovací množině a mezi popisem jevů obecnějších, které sice mají dostatečnou četnost, ale zase jsou málo specifické. Proto syntaktické analyzátory často sledují jevy na různých úrovních obecnosti nebo naopak specifičnosti a při rozhodování těmto jevům přikládají různou váhu.

Pokud tyto váhy i úrovně obecnosti jednotlivých atributů popisujících jevy považujeme za parametry modelu jazyka, můžeme použít genetické algoritmy pro nalezení hodnot vedoucích k lepšímu výkonu analyzátoru. Tímto způsobem se podařilo zlepšit výkon několika dříve popsaných závislostních analyzátorů českého jazyka.

* Tato práce byla podporována grantem „Informační společnost" pod projektem 1ET100300517.

Následující kapitoly nejprve popisují data, jednotlivé typy analyzátorů, potom parametry, které by mohly ovlivňovat jejich výkon a nakonec geneticky nalezené hodnoty těchto parametrů a výkon analyzátorů při těchto nastaveních.

Je zde zmíněna i jedna úprava pomáhající zachovat různorodost populace v průběhu genetického učení.

2 Data

Pro účely učení i vyhodnocování byl použit Pražský závislostní korpus, konkrétně 73088 vět z adresáře `anal_general_training`, které byly použity pro učení *(trénovací množina)* a 7319 vět z adresáře `anal_general_devtest` *(testovací množina)*, které sloužily pro měření úspěšnosti *(výkonu)*.

Z vět byly použity údaje o slovním tvaru, morfologické značce a indexu nadřízeného slova.

Pražský závislostní treebank, PDT 1.0, byl vyvinut Ústavem formální a aplikované lingvistiky a Centrem komputační lingvistiky University Karlovy (viz `http://ufal. mff.cuni.cz/` a `http://ckl.mff.cuni.cz/` nebo [3]).

Nyní popíšeme dílčí analyzátory, z jejichž výsledků je sestavován konečný výstup popisovaného závislostního analyzátoru.

3 Analyzátory L2R a R2L

Analyzátor L2R pracuje jako zásobníkový automat, který v každém kroku čte slovo ze vstupní věty a ukládá ho na zásobník nebo provádí redukci obsahu zásobníku, při které vyznačuje hrany-závislosti.

Povolené akce analyzátoru tedy jsou:

- **čtení:**
  první nepřečtený symbol ze vstupu (*výhled*) je přidán na vrchol zásobníku, *výhled* se posouvá na další slovo
- **vypouštění vnitřního prvku pod vrchol:**
  jeden *vnitřní prvek* je vypuštěn a je vyznačena závislost mezi odpovídajícím slovem (závislé) a slovem odpovídajícím *vrcholu* (řídící)
- **vypouštění vrcholu pod vnitřní prvek:**
  je vypuštěn *vrchol* zásobníku a je vyznačena závislost mezi odpovídajícím slovem (závislé) a slovem odpovídajícím některému *vnitřnímu prvku* vrcholu zásobníku (řídící)
- **vypouštění vrcholu jako kořene:**
  je vypuštěn *vrchol* zásobníku bez vyznačení hrany (jako kořen stromu)

Posloupností těchto akcí může analyzátor vytvářet stromy projektivní i neprojektivní, i když není možné vytvořit úplně všechny možné stromy.

Během zpracování trénovací množiny analyzátor zná ke každé větě strom a tedy ví, jaké akce má provádět. Tuto znalost ukládá ve formě *pravidel*. *Pravidla* obsahují *popis situace analyzátoru* a *četnosti* udávající, kolikrát byla během zpracování trénovací množiny v dané situaci použita ta která akce.

Jak již bylo zmíněno v úvodu, bude-li popis situace přesný, například bude-li složen z morfologických popisů nebo dokonce plných slovních tvarů několika slov v zásobníku a několika slov na vstupu, potom budou četnosti udávající, kolikrát v dané situaci automat provedl jakou akci, velice nízké, typicky s hodnotami 0 nebo 1 a při analýze bude záležet spíše na náhodě, zda automat tu kterou konkrétní situaci během učení potkal nebo ne. Taková pravidla budou mít malou věrohodnost.

Naopak, zvolíme-li popis situace obecný, sestávající například jen z údaje o slovním druhu slova na vstupu, bude takové pravidlo mít dostatečnou četnost, ale malou přesnost, nedokáže dostatečně odlišit různé situace, například morfologické shody a neshody, a rozhodování vybírající akci s vyšší četností bude neoprávněně potlačovat akce, které v těchto spojených situacích obsadily druhé nebo třetí místo.

Proto jsme už u analyzátorů popsaných v [6] použili celkem čtyři typy pravidel, od obecných k specifickým, a protože obecnější pravidla musí nutně mít vyšší četnosti, kompenzovali jsme to tím, že jsme při skládání jejich četností přihlíželi k vahám jednotlivých pravidel. To, jaké mají být tvary pravidel a jaké mají být váhy pro skládání jejich četností, jsme (tehdy) určili zčásti odhadem a zčásti experimentálně.

Konkrétně jsme používali tyto popisy situací:

- **4-pravidla:**
  vnitřní prvek, vrchol, výhled, výhled+1 (slovo za výhledem)
  váha 0.002
- **3-pravidla:**
  vnitřní prvek, vrchol, výhled
  váha 10^6
- **2-pravidla:**
  vnitřní prvek, vrchol
  váha 1.00
- **1-pravidla:**
  vrchol
  váha 10^{-21}.

Svou roli ovšem hraje i to, jak přesně popisujeme slova, ať již na vstupu nebo uvnitř zásobníku. V uvedeném případě jsme ve všech případech používali stejné popisy slov (s výjimkou ,,vyjmenovaných slov" jako čárka, tečka, spojka ,,a", kde jsme použili úplný slovní tvar), odhadem a pokusy, stejně jako u vah, jsme dospěli k použití prvních deseti znaků z patnáctiznakové morfologické značky. Pokud je nám známo, i analyzátory jiných autorů používají ve všech případech stejné popisy slov.

Analyzátor L2R na testovací množině dosahoval výkonu 70.86% správně určených závislostí. Analyzátor R2L, který se ve všem shodoval s analyzátorem L2R až na to, že při učení i při měření výkonu postupoval od konce věty směrem k začátku, dosahoval na testovací množině výkonu 73.44% správně určených závislostí.

Doplňme, že v citovaném textu je popsán ještě analyzátor ANALOG, který pracuje tak, že pro analyzovanou větu hledá úseky shodující se s úseky vět trénovací množiny a z nich potom přebírá údaje o závislostech. Kvalita shody je přitom dána délkou shodného úseku, ale i mírou shody popisů jednotlivých slov. O tomto analyzátoru se zde zmiňujeme proto, že jsme ho použili a použijeme pro skládání výsledků s geneticky vylepšenými analyzátory.

4 Genetické vylepšení poprvé

Prvním krokem v genetickém vylepšování analyzátorů bylo povolení různých popisů slov na různých pozicích pravidel. Podíváme-li se na formát 4-, 3-, 2- a 1-pravidel, napočítáme deset pozic, na kterých se používá popis slova.

Typ popisu slova na těchto pozicích jsme tedy spolu se čtyřmi reálnými hodnotami použitými jako váhy pravidel, vzali jako řetězec parametrů analyzátoru a spustili jsme genetické hledání hodnot dávajících nejlepší výsledky.

Možné popisy slova byly vždy části patnáctiznakové morfologické značky, konkrétně prvních 0, 1, 2, 9, 10 a 15 znaků a nebo spojení prvního a páteho znaku (slovní druh a pád), tyto hodnoty budeme dále značit -,1,2,9,10,15 a 1+5.

Genetická informace sestávala z již zmíněných 10 hodnot výčtového typu určujících použitou část morfologické značky a dále ze čtyř reálných hodnot určujících váhu pravidel jednotlivých typů.

Genetické hledání pracovalo s populací velikosti 50. Do nové generace byl vždy přenesen nejúpěšnější jedinec (elita) a 25 jedinců vybraných jako ten lepší z náhodně vybrané dvojice. Zbytek populace do celkového počtu vznikl křížením z náhodně vybraných rodičů z elity a vybraných jedinců a uplatněním mutace.

Křížení probíhalo tak, že zvlášť pro posloupnosti značek a pro posloupnost vah bylo vždy nalezeno místo, v němž povede hranice křížení a nově vzniklý jedinec vznikl slepením úseků genetických informací rodičů.

Mutace u jedinců vzniklých křížením s pravděpodobností 1/2 měnila informaci o morfologické značce (náhodný výběr) a o váze pravidla (vynásobení hodnotou e^r, kde r byla náhodná hodnota z rozsahu $(-2, 2)$. Pravděpodobnost mutace byla nalezena experimentálně, to, že mutace vah má stejnou pravděpodobnost jako mutace u morfologických značek, bylo napraveno v další versi.

Genetickým hledáním byl výkon analyzátoru R2L zlepšen ze 73.44% na 73.90%, výsledná pravidla a váhy byly:

– **4-pravidla:**
  vnitřní prvek: 1+5, vrchol: 15, výhled: 2, výhled+1: - (ano, nula!)
  váha $2,6489122130 \times 10^{10}$

- **3-pravidla:**
  vnitřní prvek: 10, vrchol: 10, výhled:10
  váha $2,9048849665 \times 10^{6}$
- **2-pravidla:**
  vnitřní prvek: 15, vrchol: 2
  váha $8,8861105205 \times 10^{6}$
- **1-pravidla:**
  vrchol: 1+5
  váha $2,4787521767 \times 10^{-21}$.

5 Genetické vylepšení podruhé

Výše popsaným způsobem jsme se zbavili dvou parametrů analyzátoru, jejichž hodnota do té doby byla pouze odhadnutá – popisů slov a vah pravidel. Stále nám však zůstávaly předem určené tvary pravidel.

Další krok proto spočíval ve využití již povolené hodnoty popisu slova 0 *ignorovat* a změně všech pravidel na stejný formát.

Všechna pravidla se tedy nyní dívají na dvě slova obsažená v zásobníku a na dvě slova na výhledu, s tím, že si na různých pozicích různých pravidel mohou všímat různých částí morfologické značky, včetně možnosti tuto značku zcela ignorovat. Každé z těchto „uniformních" pravidel má také svou váhu, která určuje sílu jeho odpovědi při výběru prováděné akce.

Parametry analyzátoru nyní tvoří 16 údajů o tom, jak definovat popis slova na čtyřech pozicích čtyř pravidel, a tři čísla-váhy (protože záleží pouze na vzájemném poměru vah, ne na jejich absolutní hodnotě), je váha prvního pravidla zafixovaná na 1,00). Navíc jsme zjemnili repertoát sledovaných částí značek na prvních 0, 1, 2, 3, 4, 5, 9, 10 a 15 znaků a kombinaci prvního a pátého znaku (1+5, slovní druh a pád).

Výkon analyzátoru R2L se nad uvedených 73.90% zlepšit nepodařilo, výkon analyzátoru L2R byl genetickým hledáním zlepšen z původních 71.05% na 72.50%. Výsledná pravidla a váhy byly:

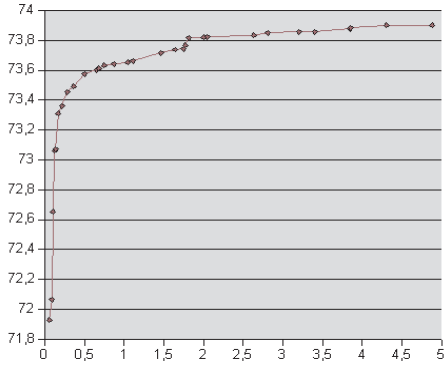
- **A-pravidla:**
  vnitřní prvek: 4, vrchol: 3, výhled: -, výhled+1: -
  váha $1,00$
- **B-pravidla:**
  vnitřní prvek: 9, vrchol: 15, výhled: 9, výhled+1: 1+5
  váha $4,6002445921 \times 10^{7}$
- **C-pravidla:**
  vnitřní prvek: -, vrchol: 1+5, výhled: 5, výhled+1: -
  váha $5,0451135548 \times 10^{-13}$
- **D-pravidla:**
  vnitřní prvek: 2, vrchol: 2, výhled: 2, výhled+1: -
  váha $4,6256553151 \times 10^{3}$

Pokud jde o parametry, genetické hledání pracovalo s populací velikosti 20. Elita byla opět tvořena nejúpěšnějším jedincem. Dále bylo vybráno 10 prvků populace jako vítězů z náhodně vybraných dvojic a tyto všechny vybrané prvky byly dále setříděny podle kvality a porovnány, na duplicitní jedince byla uplatněna mutace.

Další prvky populace do celkového počtu byly získány křížením a to tak, že byly postupně bráni již vybraní jedinci (včetně elitního), každému byl náhodně určen partner a jejich křížením, způsobem, jaký už byl popsán výše, byl vytvořen nový jedinec. Na všechny prvky populace potom, kromě elity, byla s pravděpodobností 1/2 uplatněna mutace.

Mutace jedince vypadala tak, že s pravděpodobností 1/10 mohl být pozměněn kterýkoliv údaj o morfologické značce, nejvýše však o dva kroky na výše uvedené stupnici přípustných hodnot a potom s pravděpodobností také 1/10 mohl být pozměněn kterýkoliv z parametrů udávajících váhu pravidla; změna spočívala ve vynásobení parametru hodnotou e^r, kde r bylo náhodné číslo z rozsahu $(-2/10, +2/10)$.

V této versi genetického hledání již byly odděleny pravděpodobnost mutace prvků genetické informace udávajících použitou část morfologické značky a pravděpodobnost mutace prvků určujících váhy pravidel, přesto byly nakonec použity stejné hodnoty 1/10. Tyto údaje, stejně jako ostatní, jsou výsledky provedených experimentů, prohledávaný stavový prostor je přitom veliký, výpočet ohodnocovací funkce pomalý (1-2 minuty) a presentované hodnoty jsou právě jen *hodnoty, u kterých to dobře vyšlo*. Nemůžeme proto zaručit, že by s jinými parametry hledání neprobíhalo lépe ani vyloučit, že nenalezneme lepší řešení, trochu nám může napovědět jen křivka znázorňující kvalitu nejlepšího jedince v závislosti na čase hledání a skutečnost, že z provedených pokusů uvedené parametry vedly k nejlepším výsledkům.



Obr. 1. Ohodnocení nejlepšího jedince (%) podle doby výpočtu (dny).

6 Kombinace analyzátorů

Už v textu [6] jsme popsali způsoby, jak rozhodování několika (tří) analyzátorů spojit do jednoho nad-analyzátoru. Uvádíme tam své zkušenosti s ,,chytrým" způsobem, který sleduje úspěšnost, s níž jednotlivé analyzátory určují závislosti pro jednotlivé druhy slov (hodnoceno podle morfologické značky) i výsledné zjištění, že ,,primitivní" způsob, kdy shoda libovolných dvou analyzátorů přehlasuje výsledek třetího analyzátoru a v případě, že k žádné shodě nedojde, platí výsledek předem určeného člena trojice, dává lepší výsledky.

Takovýmto spojováním výsledků analyzátorů L2R, R2L a ANALOG s výkony po řadě 70.86%, 73.44% a 71.82% jsme tehdy dospěli k výkonu 76.46% správně určených závislostí.

Použijeme-li pro spojování výsledky geneticky vylepšených analyzátorů, zlepší se výkon uvedené kombinace analyzátorů z 76.46% na 76.92%.

V témže textu jsme také popisovali zlepšení výsledků analyzátorů M.Collinse (82.49%, viz [4], [1]) a E.Charniaka (84.28%, viz [2]), které v kombinaci s našimi třemi analyzátory dosahovaly výkonu 85.76%. Použití vylepšených variant našich analyzátorů tento výsledek posunulo na 86.02%.

7 Další kroky

Mohli bychom pokračovat tím, že například zvětšíme počet pravidel nebo rozsah sledované části zásobníku nebo vstupu, výsledky by určitě neměly být horší, protože genetické hledání vždy může neužitečné informace ignorovat. Problém je ovšem v rychlosti hledání, která s růstem délky genové informace rychle klesá.

Výše uvedené údaje jsou výsledky hledání na populaci o 50 resp. 20 jedincích, kde výpočet měl končit po sto generacích bez zlepšení.

Dosažení i tohoto limitu je však problematické, protože výpočet ohodnocení jedince zahrnuje naučení analyzátoru na trénovací množině a následně zanalyzování všech vět testovací množiny, což na použitém hardware (Athlon 2500+, 512MB RAM) trvalo jednu až dvě minuty.

8 Pokus o uchování diversity

V každém kroku genetického učení jsou vždy vybíráni jedinci postupující do další generace, některé nich potom zkřížíme a některé (ze všech) necháme zmutovat. Stává se přitom, že v populaci máme více jedinců stejných nebo velmi podobných a je to jev nežádoucí, protože snižováním různorodosti populace ztrácíme výhodu vzájemného ovlivňování různých jedinců. Navrhované řešení je následující:

Po tom, co jsme určili kvalitu všech jedinců a před tím, než začneme s výběrem, pro každého jedince:

– nejprve spočteme vzdálenost nejbližšího jedince s vyšším původním ohodnocením, označíme ji V, za vzdálenost považujeme rozdílnost genenetických informací,

– potom jako nové ohodnocení jedince vezmeme staré ohodnocení vynásobené hodnotou $1 - \frac{P}{1+V}$, tj. u jedinců hodně odlišných a tvořících lokální extrémy snížíme hodnotu jen málo, naopak u jedinců v blízkosti úspěšnějších členů populace snížíme ohodnocení hodně a tím zmenšíme i jejich šanci postoupit do další generace.

Přestože nám tento návrh zní rozumně, praktické pozitivní efekty jsme nepozorovali. Pro důkladné prozkoumání této techniky by bylo třeba zvolit příklad s rychlejším výpočtem fitness funkce, abychom mohli provést dostatečný počet experimentů.

9 Závěr

Pomocí genetického hledání parametrů jazykového modelu se nám podařilo zlepšit výsledky dříve popsaných statistických analyzátorů českých vět.

Tyto výsledky stále nedosahují kvalit nejlepších analyzátorů (Collins: 82.49% [4], [1], Charniak: 84.28%, [2]), ale překračují publikované výsledky českých programů (Zeman: 74.1%, Žabokrtský: 75.2%, [7]) i dříve publikované výsledky autora. Zároveň byly vylepšeny i výsledky složených analyzátorů využívajících výstupy Collinse a Charniaka a to z 85.76% na 86.02%.

Výsledky byly získány učením, přičemž k výpočtu ohodnocovací funkce genetického hledání byla použita testovací množina, to může zpochybňovat jejich korektnost. Věříme však, že celková velikost naučených hodnot (16-krát 4 bity plus tři desetinná čísla) je příliš malá na to, aby mohla obsahovat informace specifické právě pro tuto testovací množinu více než sedmi tisíc vět.

Reference

1. M. Collins, Head-Driven Statistical Models for Natural Language Parsing, PhD thesis, Iniversity of Pensylvania, 1999
2. E. Charniak, A Maximum-Entropy-Inspired Parser, In: Proceedings of NAACL, Seattle, Washington, 2000
3. J. Hajič, Building a Syntactically Annotated Corpus: The Prague Dependency Treebank, Issues of Valency and Meaning, Karolinum, Praha 1998, 106-132
4. J. Hajič, M.Collins, L.Ramshaw, C. Tillmann: A Statistical Parser for Czech, ACL'99, Maryland USA, 1999
5. T. Holan, Tvorba závislostního syntaktického analyzátoru, MIS2004 Sborník semináře, MATFYZPRESS, Praha 2004, Eds. David Obdržálek, Jana Tesková, 16-26
6. T. Holan, Tvorba závislostního syntaktického analyzátoru — rok poté, MIS2005 Sborník semináře, MATFYZPRESS, Praha 2005, Eds. David Obdržálek, Jana Tesková, in print
7. D. Zeman, Parsing with a Statistical Dependency Model, PhD. thesis, Universita Karlova, Praha 2005

Comparison of different approaches to overlapping clustering of binary variables

Dušan Húsek[1], Aleš Keprt[2,5], Hana Řezanková[3], Alexander A. Frolov[4], P. Polyakov[4], Václav Snášel[5]

[1]Academy of Sciences of the Czech Republic, Prague
[2]Palacký University, Olomouc
[3]University of Economics, Prague
[4]Russian Academy of Sciences, Moscow
[5]VŠB Technical University of Ostrava, Ostrava–Poruba
dusan@cs.cas.cz, ales.keprt@vsb.cz, rezanka@vse.cz

Abstract. This paper describes and compares some possible solutions on how to use standard statistical procedures, blind search, formal concepts, Hopfield-like neural network, and genetic algorithms for finding overlapping binary factors. The application of factor analysis as a base for further investigation and interpretation of the factor loading matrix by means of fuzzy cluster analysis is suggested. Some experiments were performed with using statistical packages STATISTICA and S-PLUS, and our own software implementations.

1 Introduction

As the size of analyzed databases increases, methods for dimensionality reduction become more important. Factor analysis is a well known method for this purpose. It is based on investigation of correlations between variables. The input data matrix can be expressed approximately by means of multiplication of two other matrices. The first one contains factor scores and the other one contains factor loadings. In this paper we focus on the modification of factor analysis in which both input variables and new (latent) variables are dichotomous (binary). We are interested in finding overlapping binary factors.

We can mention the analysis of the input matrix X which contains information on N text documents and M terms. We want to express these terms by means of L factors. Let us consider the case in which the input matrix contains only values "one" (the term is contained in the document) and "zero" (the term is not contained in the document). This matrix can be decomposed into two matrices. Matrix F expresses the occurrence of L groups of terms in individual documents, and matrix A expresses which terms are contained in particular groups. Both of these matrices also contain the values "zero" and "one" only.

According to [4], we suppose that each M-dimensional case (row) $\mathbf{x}_i$ ($i = 1, 2,\ldots,N$) of the analyzed data matrix X can be expressed as a logical multiplication $\mathbf{x}_i \approx \mathbf{f}_i \otimes \mathsf{A}$, where $\mathbf{f}_i$ is the L-dimensional vector of factor scores and A is an $L \times M$ matrix of factor

loadings. Such a case is considered in which $\mathbf{a}_l \in \mathrm{B}_p^M$ * (pM is maintained constant, M is number of variables) and $\mathbf{f}_i \in \mathrm{B}_{p_f}^L$ (L is the number of factors). Here p and p_f are sparseness† of factor loadings row vectors $\mathbf{a}_l$, and sparseness of factor scores row vectors $\mathbf{f}_i$ respectively.

We are interested in different techniques for finding vectors of factor loadings $\boldsymbol{\alpha}_l$ from the data. According to our experience, no commercial product exists for this purpose. Formerly, the special procedures for Boolean factor analysis existed in BMDP (its name was 8M). We tried to use it, but we didn't obtain satisfactory results. The problem of BMDP's 8M is that the process leads to finding only a local minimum of the vector error function.

2 Suggested solutions

In this paper, we compare different techniques for solving this task. Firstly we are interested in finding an approximate solution by means of standard statistical procedures implemented in the statistical packages STATISTICA (6.0) and S-PLUS (4.5). There is the task of finding overlapping clusters of asymmetric binary variables. We are interested in assignment of variables to clusters with the known number of them.

Secondly, we tried our own developed algorithms for binary factor analysis. Blind search (see [6]) and formal concepts (see [7]) based algorithms are of so called "hard computing" kind, while the Hopfield-like neural network (see [4]), and genetic algorithm GABFA (see [6]) are of so called "soft computing" kind. We try to compare all these mentioned algorithms to do the best possible factor–based binary clustering.

2.1 Statistical methods

Statistical techniques for overlapping clustering are often discussed theoretically, but procedures for solving tasks of this type are rarely implemented in commercial statistical packages. Fuzzy cluster analysis (in the S-PLUS system) could be used for this purpose but it supports only clustering of the cases, so the input matrix needs to be transposed. However, if we applied this technique for simulated data files, the solution did not correspond to input setting on the basis of which the data were generated.

One possible way of finding the overlapping clusters of variables by methods in common use is to combine results obtained by different techniques for clustering. This consists mainly in different agglomerative techniques of hierarchical cluster analysis or techniques of multidimensional scaling. The suitable similarity measures

* $\mathrm{B}_p^M = \{\mathbf{X} \mid X_j \in \{0,1\}, P\{X_j = 1\} = p \quad \forall j = 1,2,...,M\}$

† Sparseness = number of active elements / total number of elements

for asymmetric binary data are needed. Applications of the process mentioned above are described in [11].

The other way of obtaining an approximate solution of the task defined above is the use of factor analysis as the base for the further analysis. We came to a conclusion that the suitable technique for revelation of factors is the interpretation of the factor loading matrix by fuzzy cluster analysis. In this manner we obtain the solution which is a good representation of the data generated for the analysis. We can obtain similar results by using multiple correspondence analysis and interpretation of coordinates by fuzzy cluster analysis in the same manner as in the previous case.

As mentioned before, for simplification we suppose that the number of factors (L) is known. In some cases factor interpretation can be difficult. That's why we decided to interpret results by fuzzy cluster analysis in S-PLUS. For this purpose, it is important to determine a less number of factors (K) for factor analysis. We suggest setting this number so that the following relations were satisfied:

$$K_{\min} = 2,\ 2^{K_{\max}-1} < L \ \text{ and } 2^{K_{\max}} \geq L\,.$$

In the same manner we can determine a number of coordinates for multiple correspondence analysis.

As the results from fuzzy cluster analysis we obtain L clusters. If the value of a membership coefficient is greater than or equal to $1/L$ then the variable belongs to the particular cluster (this value would be replaced by 1); in the opposite case the variable is not assigned to this cluster (this value would be replaced by 0).

2.2 Blind search and formal concepts

The basic algorithm called "Blind Search" simply searches for the solution among possible combinations of F and A matrices. This is obviously 100 percent exact but also extremely computationally expensive, so this kind of solver in its basic implementation is simply unusable.

In order to make the blind search based solver more usable, we did several changes to it. Mainly, all empty rows and empty columns are removed. Moreover, we can ignore both cases and variables with too low or too high number of 1s. We can also remove duplicate rows if we remember the repeat–count of each row. We call it multiplicity. The algorithm is described in detail in paper [6].

If we want to speed up the simple blind-search algorithm (we obviously want to), we can try to find some factor candidates, instead of checking out all possible bit-combinations. The technique which can help us significantly is Formal Concept Analysis (FCA). FCA is based on concept lattices but we actually work with formal concepts only, so we need only a small portion of FCA theory to do our factor analysis. As described in paper [7], all factors can be found among the formal concepts. We only need to know the number of factors we are searching for.

2.3 Hopfield-like neural network

As shown in paper [4], a modified version of binary Hopfield neural network can also be used to find binary factors in terms of Boolean factor analysis. We proposed a method for sufficiently nonlinear case that naturally follows from Hebbian-like learning rule and properties of Hopfield-like neural network dynamics. This conceptual mechanism seems plausible in frame of memory storage and processing in brain. This algorithm has several additional requirements on analyzed data (which must be fulfilled in order let it work correctly), but is able to work with much larger data then other algorithms presented here, including the statistical procedures (see [4] for details).

2.4 Genetic algorithm

Genetic algorithms (GAs) are well known global optimizers. With well defined vector error function, we can use GAs to find the binary factors. In paper [8] we described two different genetic algorithms which were successfully used to find the binary factors. That paper concludes the described genetic algorithms by saying that algorithm called GABFA may be assumed as the best algorithm for binary factor analysis, so far known. In this paper, we are about to test it on a real data matrix of text documents and words in them.

2.5 Discrepancy

We use discrepancy error function to compare the quality of results obtained by different methods. This function is defined as the number of different bits in input data set X and the product of F and A (in this order). (Note that these matrices are not quantified by all presented methods, so we had to perform additional computation in some cases, because discrepancy value is the only commonly accepted measure of quality of binary/boolean factor analysis.)

3 Experiments

We applied techniques mentioned above to the input data matrix characterizing a collection of text documents. We tried to analyze the data matrix of terms (words) occurrence in text documents. We analyzed the data published in Berry et al. [1] extended by one document to total count of 15 documents and 18 terms, which we assume as the good example of the real world data set. We set number of factors L=4.

3.1 Statistical methods

We applied both factor analysis and multiple correspondence analysis and then we used fuzzy cluster analysis for result interpretation. We interpreted only the part of

output about coordinates from multiple correspondence analysis, i.e. terms in connection with the value "one".

There were two factors in factor analysis and two coordinates in multiple correspondence analysis. The factor loadings and coordinates were interpreted by four clusters in fuzzy cluster analysis. On the basis of factor analysis we can identify the following four groups of terms (terms assigned to just one group are written in bold).

Solution 1. Factor analysis followed by fuzzy cluster analysis
Cluster 1: 0,1,3,4,8,15 **abnormalities**, age, **blood**, close, disease, **respect**
Cluster 2: 2,6,10,11,16,17 **behavior**, **depressed**, generation, **oestrogen**, rise, study
Cluster 3: 4,8,9,10,13,14,16 close, disease, **fast**, generation, **pressure**, **rats**, rise
Cluster 4: 1,5,7,12,17 age, **culture**, **discharged**, **patients**, study
discrepancy=42 (positive=37, negative=5)
computation time: several seconds

On the basis of multiple correspondence analysis we obtain a very similar result with the following difference: "behavior" is assigned to two groups and "generation" is assigned to just one group.

Solution 2. Multiple correspondence analysis followed by fuzzy cluster analysis
Group 1: 0,1,3,4,8,15 **abnormalities**, age, **blood**, close, disease, **respect**
Group 2: 2,6,11,16,17 behavior, **depressed**, **oestrogen**, rise, study
Group 3: 4,8,9,10,13,14,16 close, disease, **fast**, **generation**, **pressure**, **rats**, rise
Group 4: 1,2,5,7,12,17 age, behavior, **culture**, **discharged**, **patients**, study
discrepancy=42 (positive=34, negative=8)
computation time: several seconds

These results are rather bad. Before we go to compare it with the results of our own algorithms, we try to find out the reason of bad quality of these results. We suppose the problem is in the fact, that all variables were assigned to factors, what made it hard to use these "large" sets of variables to express of the original data. The reason of this state is in the proposed formula in setting the $1/L$ as the lower limit of membership function for adding a particular variable to a cluster (see section 2.1). We now replace this limit by simply requiring strong 50% (i.e. 0.5) membership to add a variable to a cluster. See solutions 3 and 4 for the results.

Solution 3. Factor analysis followed by fuzzy cluster analysis, 50% cut
Cluster 1: 0,3,15 abnormalities, blood, respect
Cluster 2: 6,11 depressed, oestrogen
Cluster 3: 9,13,14 fast, pressure, rats
Cluster 4: 5,7,12 culture, discharged, patients
discrepancy=30 (positive=27, negative=3)

Solution 4. Multiple correspondence analysis followed by fuzzy cluster analysis, 50% cut
Group 1: 0,3,15 abnormalities, blood, respect
Group 2: 6,11 depressed, oestrogen
Group 3: 9,10,13,14 fast, generation, pressure, rats
Group 4: 5,7,12 culture, discharged, patients
discrepancy=30 (positive=26, negative=4)

As we can see, the results are now much better. By limiting the membership function to 0.5, we also ensured that no overlapping occurs anymore. Since this change resulted in significant rise in solution quality, we may conclude, that standard statistical procedures have got the main bottleneck in finding the overlapping part of binary cluster.

In the following sections we are going to compare these results with our own algorithms.

3.2 Algorithm FCBFA

When using formal concepts based factorization algorithm, we obtain a different result to the ones shown above. Actually, we get more pieces of equally good solutions, of which only one randomly selected is listed here for example.

Solution 5. Algorithm FCBFA (a formal concepts based one)
settings: min.1 variable in a factor, i.e. 37 of 39 formal concepts used
Factor 1: 3,4,8,13 blood, close, disease, pressure
Factor 2: 0,1,3 abnormalities, age, blood
Factor 3: 6,11,17 depressed, oestrogen, study
Factor 4: 5,7,12 culture, discharged, patients
computation time: 0.1 seconds
discrepancy=29 (positive=25, negative=4)
similar solutions: 27 solutions with the same discrepancy

3.3 Genetic algorithm GABFA

The result of genetic algorithm GABFA depends only on the initial population. If we repeat the computation more times, we actually get many different solutions with this discrepancy value, because GAs use random numbers, so it is hard to get the very same solution in repeated runs. All our tests were performed with parallel GABFA with 5 populations of 30 individuals.

We started the GA's computation from a random population, or a random set of vectors with 2–10 1s in them. We got the same results in both cases, but not always in the case of random populations – they sometimes converged to local minima with discrepancy 30.

Solution 6. GABFA, parallel version with 5 populations of 30 individuals
Factor 1: 5,8,10,17 culture, disease, generation, study
Factor 2: 5,7,12 culture, discharged, patients
Factor 3: 6,11,17 depressed, oestrogen, study
Factor 4: 3,4,8,13 blood, close, disease, pressure
computation time: 0.1 second
discrepancy=29 (positive=25, negative=4)

3.4 Hopfield-like neural network

Our neural network found the following 4 factors, see solution 8.

Solution 8. Four factors of neural network algorithm.
Factor 1: 3,4,8,13 blood, close, disease, pressure
Factor 2: 0,1,3,15 abnormalities, age, blood, respect
Factor 3: 5,6,7,12,17 culture, depressed, discharged, patients, study
Factor 4: 9,10,13,14,16 fast, generation, pressure, rats, rise
discrepancy=31 (positive=24, negative=7)

These factors are a bit different to the results of the other methods. Each factor has assigned more variables than above, and still the overall discrepancy is quite good. The factors which "look similar" to some factors identified by the previous methods really are similar, usually because they are only variants of the same factor core (e.g. {0,1,3,15} and {0,1,3} are variants of factor core {0,3}).

3.5 Blind Search algorithm

We continue with Blind Search algorithm. As mentioned before, this algorithm can always find the best possible solution, as long as we can afford the time required to find it.

In the real world, we have to set some reasonable limits to blind searching in order to get finished in a reasonable time. Looking at the results of other methods (as listed above), we decided to limit number of 1s to exaclty three. See solution 9.

Solution 9. Blind Search (BS).
settings: exaclty 3 variables in a factor, i.e. 816 factor candidates
Factor 1: 0,1,3 abnormalities, age, blood
Factor 2: 5,7,12 culture, discharged, patients
Factor 3: 6,11,17 depressed, oestrogen, study
Factor 4: 9,10,14 fast, generation, rats
computation time: 220min 42s
discrepancy=29 (positive=24, negative=5)
similar solutions: 6 solutions with the same discrepancy

Note that BS found only 6 good solutions, compared to 27 good solutions found by FCBFA. If we want to find more good solutions by BS, be have to search widely. Solution 10 is an example of wider BS search which took 421 minutes.

Solution 10. Blind Search (BS).
settings: 2–3 variables in a factor, i.e. 816 factor candidates
Factors: See solution 9
computation time: 421min 53s
discrepancy=29 (positive=24, negative=5)
similar solutions: 12 solutions with the same discrepancy

3.5 Random Trial algorithm

We conclude our experiments with Random Trial algorithm. This "algorithm" does repeat trials of randomly generated candidate matrix A. No intelligence is involved here. We list the RT results here just to give an overview of what discrepancy values are really bad. When an algorithm gives a solution of discrepancy greater than RT, it's really bad solution. See solution 11.

Solution 11. Random Trial (RT) algorithm.
Factor 1: 0,1,2,6,11,12,15 abnormalities, age, behavior, depressed, oestrogen, patients, respect
Factor 2: 1,3,4,8,9,11,13 age,blood,close,disease,fast,oestrogen,pressure
Factor 3: 5,9,15 culture,fast,respect
Factor 4: 1,5,6,7,12 age,culture,depressed,discharged,patients
computation time: 0.3s
discrepancy=40 (positive=33, negative=7)

3.6 Evaluation of results

We can evaluate the correctness or quality of our results by the means of discrepancy error function. See table 1. We show also the computation time, although it is not listed for manual analysis procedures. Let's try to sort the algorithms and methods from the best to the worse.

According to discrepancy, the worst possible value equals to number of bits in the inverse input matrix, i.e. 270. Since we allow the matrix of factor scores to be a zero matrix, the real maximum discrepancy equals to the number of 1s in input matrix, i.e. 49. Knowing this value, we rate the results based on discrepancy as "how far is it from 49". This score is also listed in the table 1.

Algorithm	Discrepancy	Score	Time	Limitations
Blind Search	29 (6×)	20	220m	Exactly 3 variables in a factor
Blind Search	29 (12×)	20	421m	3–4 variables in a factor
FCBFA	29 (27×)	20	0.1s	Have to compute a concept lattice
GABFA/comb.	29	20	0.1s	Have to compute combinations
GABFA/random	29	20	0.3s	–
Neural network	31	18	n/a	Manual
Random trials	40	9	0.3s	–
Std. stat. proc's	42	7	seconds	Manual access, membership>=1/L
Std. stat. proc's	30	19	seconds	Manual access, membership>=0.5

Tab. 1. Comparison of algorithms.

1. FCBFA: score 20 (error 29), very fast, multiple solutions
   FCBFA performed best in our experiments, as it was both fast and precise. In addition, it is a multiple–solution finder, and found 27 good solutions here. Unfortunately, there is one hidden pitfall in this algorithm: As it needs a precomputed concept lattice or at least the formal concepts, this preprocessing step may slow the algorithm down in the case of larger data sets.

2. GABFA: score 20 (error 29), very fast
   Genetic algorithm performed also very well. It only finds one solution at a time, but can be rerun repeatedly to find more different good solutions.
3. Blind Search: score 20 (error 29), very slow, 12 solutions
   We found 12 good solutions with BS, but the computation time 421 minutes is all but satisfying. The blind search placed third in our experiments, because of its excellent results, but it would be the very last one in our chart if the data were larger.
4. Neural network: score 18 (error 31), manual
   Discrepancy of neural network's solution was 31, which is a bit higher than the one of higher placed algorithms. Its additional drawback is that it isn't fully automatical and needs an user/operator interaction.
5. Standard statistical procedures: score 7 (error 72), fast
   The results showed that standard statistical procedures available in statistical software packages are not a good option for performing the binary overlapping clustering. They are at the bottom of our table with result approximately third as good as the result of dedicated Boolean factorization methods (7 points compared to 20 points). The results are better when ignoring the overlapping parts of clusters, but this approach was used for this paper only, and not generally tested before.

4 Conclusion

In previous chapter we presented results obtained by several different algorithms and methods. Seeing that a specific procedure for Boolean factor analysis is missing in statistical packages, we have tried to find a way to determine L overlapping binary factors. We analyzed the input data matrix characterizing a collection of text documents.

Obviously, a special procedure for solving the described task would be a better solution. In recent years, we developed several methods or algorithms to perform a direct binary Boolean factor analysis. This paper presented the very first in-depth comparison of all these methods on a real world data, showing that our specifically binary methods are significantly better than standard statistical procedures found in statistical software packages. We only recommend using the described standard (nonbinary) statistical approach in the case you are for some reason limited to the standard statistical software and cannot use custom one.

Acknowledgement

The presented paper is financially supported by grant GA MSK(CZ)LN00B096 awarded by Ministry of Education of the Czech Republic.

References

1. Berry M.W., Dumais S.T. and Letsche T.A., Computational Methods for Intelligent Information Access. 1995, http://www.cs.utk.edu/~berry/sc95/sc95.html
2. Fichet B., Distances and Euclidean Distances for Presence-Absence Characters and Their Application to Factor Analysis. Multidimensional Data Analysis, Proceedings of a Workshop, DSWO Press, Leiden, 1986, 23–46
3. Frolov A.A., Húsek D. and Muraviev I.P., Informational Capacity and Recall Quality in Sparsely Encoded Hopfield-Like Neural Network: Analytical Approaches and Computer Simulation. Neural Networks 10 1997, 845–855
4. Frolov A.A., Húsek D., Muraviev I.P., Řezanková H., Snášel V. and Polyakov P., Features Extraction by Hopfield-Like Neural Network. Neural Network Engineering Experiences. University of Malaga, Malaga, 2003, 383–390
5. Hopfield J.J., Neural Network and Physical Systems with Emergent Collective Computational Abilities. Proc. Natl. Acad. Sci. USA 79, 1982, 2544–2548
6. Keprt A., Using Blind Search and Formal Concepts for Binary Factor Analysis. In Proceedings of Dateso 2004. VŠB Technická Univerzita Ostrava, Czech Republic; CEUR WS – Deutsche Bibliothek, Aachen, Germany; 2004, ISBN 80-248-0457-3 (VŠB TUO), 120–131, ISSN 1613-0073 (CEUR)
7. Keprt A. and Snášel V., Binary Factor Analysis with Help of Formal Concepts. In proceedings CLA / Concept Lattices and their Applications 2004. VŠB Technická Univerzita Ostrava, Czech Republic, 2004, 90–101, ISBN 80-248-0597-9
8. Keprt A. and Snášel V., Binary Factor Analysis with Genetic Algorithm. In proceedings of 4th IEEE International Workshop on Soft Computing as Transdisciplinary Science and Technology – WSTST 2005. Springer Verlag, Berlin Heidelberg, Germany, 2005, 1259–1268, ISBN 3-540-25055-7. In series Advances in Soft Computing, ISSN 1615-3871.
9. Perez-Vicente C. J. and Amit D., Optimized Network for Sparsely Encoded Patterns, J. of Physics A: Math. Gen. 22, 1989, 559–569
10. Řezanková H. and Húsek D., Comparison of the SAS, SPSS and STATISTICA Systems in the Area of Clustering Variables. Computational Statistics & Data Analysis (Statistical Software Newsletter) 41, 2002, 331–339
11. Řezanková H. and Húsek D., Nonlinear Factorization of Binary Variables in Statistical Packages. Mundus Symbolicus 11, 2003, No. 1, 35–44
12. Řezanková H., Húsek D., Smid J. and Snášel V., Clustering of Documents via Similarity Measures. CIC'03, CSREA Press, Las Vegas, 2003, 292–299

Conversion of regular expressions into 2-realtime automata*

Ľubomíra Ištoňová

Department of Computer Science, P. J. Šafárik University
Jesenná 5, 04154 Košice, Slovakia
istonova@science.upjs.sk

Abstract. We consider conversions of regular expressions into realtime finite state automata, in which the length of each ε-path is bounded by a fixed constant k. For $k = 2$, i.e., for automata in which no ε-path is longer than 2, we show that the conversion of a regular expression into such automaton produces only $O(n)$ states, $O(n)$ ε-free edges, and $O(n \log n)$ ε-edges. This result breaks the known lower bound $\Omega(n(\log n)^2/\log\log n)$, holding for 0-realtime automata, i.e., for automata with no ε-transitions.

Keywords Descriptional complexity, finite-state automata, regular expressions.

1 Introduction

The analysis of different tools describing formal languages not only with respect to their expressive power, but taking also into account descriptional complexity of these tools, is a classical topic of the formal language theory. Still, not all relations among descriptional complexity of different kinds of formalism are known, even when restricting to devices describing regular languages only. Despite the simplicity of regular languages, some problems concerning them are still open.

Regular expressions and finite automata without ε-transitions are two of the most popular formalisms for regular languages. The size of an automaton is measured by the number of its transitions, while the size of a regular expression is the number of occurrences of alphabet symbols in it.

It is known that a conversion of a nondeterministic ε-free finite automaton into an equivalent regular expression may cause an exponential blow-up [1]. For the converse direction, it was conjectured that the best conversion is an $O(n^2)$ conversion [8]. Only quite recently [5], [6], it was proved that, for each regular expression of size n, there exists an equivalent ε-free automaton with only $O(n(\log n)^2)$ transitions. After that, it has been shown that if we consider regular languages over a binary input alphabet, the above upper bound can

* This work was supported by the Science and Technology Assistance Agency under contract APVT-20-004104, and by the Slovak Grant Agency for Science (VEGA) under contract "Combinatorial Structures and Complexity of Algorithms."

be reduced to $O(n \log n)$ transitions [2]. This result was also generalized to the case of regular languages over a fixed alphabet with at most s input symbols: $O(s\, n \log n)$ edges are sufficient here. The upper bounds, which were presented above, are complemented in [7] by the lower bound $\Omega(n(\log n)^2/\log\log n)$.

Comparing $O(n(\log n)^2)$ with $\Omega(n(\log n)^2/\log\log n)$, that is, the upper and lower bounds, it seems at the first glance that no significant improvement is possible.

However, if we allow ε-transitions, the standard conversion of a regular expression into an equivalent automaton does not produce more than $O(n)$ transitions. The crucial drawback of such automaton is that, due to cycles consisting of ε-edges only, there may exist computation paths of unbounded length.

Therefore, it is quite natural to ask what we have to add to an ε-free automaton to break the lower bound $\Omega(n(\log n)^2/\log\log n)$, and still have an automaton with a linear computation time, in the length of the input. This condition can be even more restrictive, by requiring the automaton to execute only a constant number of steps in between reading any two consecutive symbols from the input.

We have named such machines, with the length of all ε-paths bounded by a constant, as "realtime" automata.[1] An automaton is *k-realtime*, if the length of any ε-path in it does not exceed the given fixed constant k.

In this paper, we shall show that already for $k = 2$, i.e., for automata in which no ε-path is longer than 2, the conversion of a regular expression produces only $O(n)$ states, $O(n)$ ε-free transitions, and $O(n \log n)$ ε-transitions.

In the general case, for $k > 2$, the descriptional complexity of k-realtime automata is still an open problem.

2 Preliminaries

Here we give some basic definitions and notation used throughout the paper. For a more detailed exposition on finite automata, the reader is referred to [4].

A *nondeterministic finite automaton* is a quintuple $M = (Q, \Sigma, \Delta, q_I, F)$, where Q is a finite set of states, Σ is a finite set of input symbols, $\Delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$ is a set of transitions (edges), $q_I \in Q$ is an initial state and $F \subseteq Q$ is a set of final (accepting) states. Here ε denotes an empty string. The *language accepted* by M is the set $L(M)$ consisting of all strings $w = a_1 \dots a_k \in \Sigma^*$, for which there exists a path of transitions connecting the state q_I with some state in F and labeled by $a_1 \dots a_k$.

A transition $x = (s_x, a, t_x) \in \Delta$ is *ε-free*, if $a \neq \varepsilon$, that is, if a is a standard input alphabet symbol. Otherwise, i.e., for $a = \varepsilon$, x is an *ε-transition*.

To make the notation of the paper more readable, the transition x will be presented in the form $s_x \xrightarrow{a} t_x$. If $a = \varepsilon$, it will be omitted, i.e., the edge will be

[1] Realtime devices play an important role in practice. Their formal definition dates back many years, down to [3]. Here we shall concentrate on the simplest realtime devices, without any auxiliary memory, which are one-way finite state automata.

in the form $s_x \to t_x$. The states $s_x, t_x \in Q$ are called the *source* and *target* states of the edge x, respectively.

A path consisting of several transitions will be displayed in a more compact form $q_1 \stackrel{a_1}{\to} q_2 \stackrel{a_2}{\to} q_3 \stackrel{a_3}{\to} \cdots \stackrel{a_{m-1}}{\to} q_m$. If all edges along this path are ε-transitions, we call such path an *ε-path*. The fact that the state q_m is reachable by some ε-path from q_1 can also be expressed in the form $q_1 \to^* q_m$. The same notation can be used to represent that two *edges* $x, y \in \Delta$ are connected by an ε-path: $x \to^* y$ indicates that there exists an ε-path from the target state of x to the source state of y. The meaning of combinations like $x \to^* q$ should be obvious.

The *length* of a path is, by definition, the number of edges along this path.

A *k-realtime automaton* M, where $k \geq 0$ is an integer constant, is a nondeterministic finite state automaton in which the length of each ε-path is bounded by k. As a special case, the automaton is 2-realtime, if each ε-path is of length at most 2. In other words, the machine cannot change its state, without reading an input symbol, more than two times in a row. Similarly, in 1-realtime automata, each ε-path degenerates into a single ε-edge. It should be clear that for $k = 0$ we get exactly the class of ε-free automata.

The language $L(\alpha)$, represented by a *regular expression* α, is defined in the usual way, by structural induction on α (see, e.g., [4]). We introduce, for technical reasons, a new unary operator of option "$^\diamond$", which is defined by $\alpha^\diamond \stackrel{\text{df.}}{=} \alpha + \varepsilon$. The *size of a regular expression* α, denoted by $s(\alpha)$, defined in the usual way, is the number of occurrences of alphabet symbols in α. that is, $s(\emptyset) = s(\varepsilon) = 0$, $s(a) = 1$, $s(\alpha_1^*) = s(\alpha_1^\diamond) = s(\alpha_1)$, and $s(\alpha_1 \cdot \alpha_2) = s(\alpha_1 + \alpha_2) = s(\alpha_1) + s(\alpha_2)$.

We shall also use *a representation* of a regular expression *by a binary tree*. In this tree, each binary operator is represented by an inner node with two sons corresponding to its two subexpressions, each unary operator by an inner node with one son for its only subexpression, and each occurrence of an alphabet symbol or special symbol "$\emptyset$" or "ε" represented by a leaf.

By a *subtree below a node* x we mean the tree consisting of the node x itself and all its descendants in the given tree.

In what follows, we shall need the following technical lemma. The lemma shows that each binary tree can be decomposed into two subtrees with a balanced number of leaves that were, initially, marked as "attended". One of these subtrees is below the separating inner node x. The second one consists of all remaining nodes of the original tree.

Lemma 1. *Let ϱ be a finite binary tree, with each leaf marked either as "attended" or as "ignored". The total number of leaves marked as "attended" is $k \geq 2$. Then there exists a node x such that the number of attended leaves in the subtree below the node x is at most $2/3 \cdot k$, but more than $1/3 \cdot k$.*

We present this lemma without a proof, which can be found in [2].

3 Conversion into small automata

In this section, we briefly describe the preprocessing phase, which converts the regular expression into a normal form and then into a nondeterministic automaton with at most $2n$ states and n ε-free transitions. This nondeterministic automaton will also contain some ε-transitions.

The conversion of the original regular expression into a normal form reduces the number of binary operators to at most $n-1$, moreover, the converted expression α satisfies the following properties: *(i)* either α degenerates to $\emptyset$ or ε, or, in the tree representation of α, *(ii)* each leaf corresponds to an alphabet symbol (all symbols $\emptyset$ and ε have been eliminated), and *(iii)* a son of a unary node corresponds either to a simple alphabet symbol or to a binary node for concatenation (the other unary node or a binary node for a union have also been eliminated).

The rules for such conversion were originally presented in Lemma 3.1 in [2].[2]

After that, the unary nodes in the tree representation of α are eliminated by unifying them with their sons. Thus, we get a binary tree in which we can find only the following types of nodes. An inner node corresponds either to a union $w_1 + w_2$, a simple concatenation $w_1 \cdot w_2$, an optional concatenation $(w_1 \cdot w_2)^\diamond$, or an iterated concatenation $(w_1 \cdot w_2)^*$. A leaf represents either a simple alphabet symbol $a \in \Sigma$, an optional symbol $a^\diamond$, or an iterated symbol a^*.

A regular expression being represented by a such binary tree can be converted into a nondeterministic automaton how the following theorem shows.

Theorem 1. **[2]** *Each regular expression α of size $n \geq 1$ can be replaced by en equivalent nondeterministic automaton M with at most $2n$ states and n ε-free transitions.*

The construction is quite standard. It should be pointed out, that the automaton M reflects the structure of the regular expression and also of its tree representation. Note also that the resulting automaton contains no transition ending in the initial state, neither starting in the unique final state. For the full version of the theorem as well as a detailed proof, the reader is referred to Theorem 3.2 in [2].

Before passing further, we shall present a realtime automaton used for small values of n.

Theorem 2. *For each regular expression of size $n \geq 1$, there exists an equivalent nondeterministic 1-realtime automaton with at most $2n + 1$ states, n ε-free transitions, and $n^2 + 1$ ε-transitions.*

4 Conversion into 2-realtime automata

Note that for each subexpression β in α, corresponding to a subtree bellow some node in the graph representing α, we have a subautomaton M_β in the

[2] The rules result from the fact that $(\alpha_1+\alpha_2)^* = (\alpha_1^\diamond \cdot \alpha_2^\diamond)^*$, and $(\alpha_1+\alpha_2)^\diamond = \alpha_1 + \alpha_2^\diamond$.

automaton M, constructed in Theorem 1. M_β has a single entry and a single exit point, denoted by en_β, ex_β, respectively.

The next definition generalizes the terminology of the so called "regions".

Definition 1. *Let β and $\beta_1, \ldots, \beta_\ell$ be some subexpressions of α, such that $\beta_1, \ldots, \beta_\ell$ are subexpressions of β, but β_i is not a subexpression of β_j, for $i \neq j$. The list $\beta_1, \ldots, \beta_\ell$ may also be empty. Then a $\beta - \beta_1, \ldots, \beta_\ell$* region *is a subgraph ϱ in the graph representing M, consisting of the inside part of M_β after removing the inside parts of $M_{\beta_1}, \ldots, M_{\beta_\ell}$.*

We call this subgraph an inside *part of the region. All remaining states and edges in M form an* outside *part. A* boundary *of a region consists of the boundary states for β and $\beta_1, \ldots, \beta_\ell$. The state en_β and $\mathrm{ex}_{\beta_1}, \ldots, \mathrm{ex}_{\beta_\ell}$ are entry points and $\mathrm{ex}_\beta, \mathrm{en}_{\beta_1}, \ldots, \mathrm{en}_{\beta_\ell}$ are exit points of the region.*

Note that a state may be, at the same time, an entry and also an exit point (for example, if $\mathrm{ex}_{\beta_i} = \mathrm{en}_{\beta_j}$, for some $i \neq j$). A relevant property of the regions is that they can be split into the subregions with a balanced number of ε-free edges. This is formalized in the next lemma, which is a variant of Lemma 1.

Lemma 2. *Let M be the automaton constructed in Theorem 1, let $\varrho = \beta - \beta_1 \ldots \beta_\ell$ be a region in M, for some subexpressions β and $\beta_1 \ldots \beta_\ell$, and let the total number of ε-free transitions inside the region ϱ be $k \geq 2$. Then ϱ can be decomposed into two disjoint regions ϱ_1 and ϱ_2 such that the number of ε-free transitions inside each of them is at least $1/3 \cdot k$, but at most $2/3 \cdot k$. More precisely, the list $\beta_1, \ldots, \beta_\ell$ can be partitioned into two disjoint lists $\beta'_1, \ldots, \beta'_{\ell'}$ and $\beta''_1, \ldots, \beta''_{\ell''}$, such that, for some subexpression β' in β, $\varrho_1 = \beta' - \beta'_1, \ldots, \beta'_{\ell'}$ and $\varrho_2 = \beta - \beta', \beta''_1, \ldots, \beta''_{\ell''}$.*

Note that the algorithm derived from Lemma 1, computing the separating inner node in the tree, can be easily modified to work directly with the regions in M, and thus to find a boundary between ϱ_1 and ϱ_2, i.e., the boundary states of $M_{\beta'}$. This only requires, in the tree representation of β, to mark all leaves in this tree that are not contained in the subtrees for $\beta_1, \ldots, \beta_\ell$ as "attended", but those in the subtrees for $\beta_1, \ldots, \beta_\ell$ as "ignored". Note that the leaves marked as "attended"correspond exactly to the k ε-free transitions in the region $\varrho = \beta - \beta_1, \ldots, \beta_\ell$. (For more details, see Lemma 4.2 in [2]).

Definition 2. *Let M be the automaton constructed in Theorem 1. Then, for each ε-free transition $x \in \Delta$, we shall construct two sets $\mathrm{In}(x) \subseteq Q$ and $\mathrm{Out}(x) \subseteq Q$, by the use of the following procedure:*

(i) Initially, the sets $\mathrm{In}(x)$, $\mathrm{Out}(x)$ are empty. We shall also keep track of a "current region" ϱ. Initially, ϱ is equal to the entire graph for M, i.e. $\varrho = \beta - \beta_1, \ldots, \beta_\ell$, where $\beta = \alpha$, the original regular expression, and $\beta_1, \ldots, \beta_\ell = \emptyset$ is an empty list.

(ii) Now let $\varrho = \beta - \beta_1, \ldots, \beta_\ell$ be the current region. Using the procedure described in Lemma 2 (based on Lemma 1), find β' splitting the region ϱ into two

subregions $\varrho_1 = \beta' - \beta'_1, \ldots, \beta'_{\ell'}$ *and* $\varrho_2 = \beta - \beta', \beta''_1, \ldots, \beta''_{\ell''}$*, with a balanced number of* ε*-free transitions. That is, the number of* ε*-free transitions in each subregion is between* $1/3 \cdot k$ *and* $2/3 \cdot k$*, where* $k \geq 2$ *is the number of such transitions inside* ϱ*.*

(iii) Next, we use the transition x *as a criterion for branching. We check where the transition* x *is located:*

(a) If the transition x *is located in the inside of* ϱ_1*, i.e., inside* $M_{\beta'}$*, then:*

- ϱ_1 *becomes the new current region, i.e., we let* $\varrho := \varrho_1$*.*
- *If there exists a path* $\mathrm{en}_{\beta'} \rightarrow^* x$*, insert the state* $\mathrm{en}_{\beta'}$ *into* $\mathrm{In}(x)$*.*
- *If there exists a path* $x \rightarrow^* \mathrm{ex}_{\beta'}$*, insert the state* $\mathrm{ex}_{\beta'}$ *into* $\mathrm{Out}(x)$*.*

(Here $\mathrm{en}_{\beta'} \rightarrow^* x$ *denotes a path connecting the state* $\mathrm{en}_{\beta'}$ *with the transition* x*, consisting of an arbitrary number of* ε*-transitions, including zero, i.e., the case in which the edge* x *springs directly from* $\mathrm{en}_{\beta'}$*. Similarly,* $x \rightarrow^* \mathrm{ex}_{\beta'}$ *denotes a path of* ε*-transitions connecting* x *with* $\mathrm{ex}_{\beta'}$*. Note that if such paths do not exist, no states are inserted into the respective sets* $\mathrm{In}(x)$ *and/or* $\mathrm{Out}(x)$*.)*

(b) If the transition x *is located in the inside of* ϱ_2*, i.e., outside* $M_{\beta'}$*, then:*

- ϱ_2 *becomes the current region, i.e., we let* $\varrho := \varrho_2$*.*
- *If there exists a path* $x \rightarrow^* \mathrm{en}_{\beta'}$*, insert the state* $\mathrm{en}_{\beta'}$ *to* $\mathrm{Out}(x)$*.*
- *If there exists a path* $\mathrm{ex}_{\beta'} \rightarrow^* x$*, insert the state* $\mathrm{ex}_{\beta'}$ *to* $\mathrm{In}(x)$*.*

(iv) Repeat the Steps (ii) and (iii) until the number of ε*-free transitions in the current region* ϱ *is reduced to* $k = 1$*. (At this moment, the unique* ε*-free transition remaining in* ϱ *is* x*).*

The sets $\mathrm{In}(x)$, $\mathrm{Out}(x)$ in Definition 2 are similar to the sets $\mathrm{In}(q)$, $\mathrm{Out}(q)$ in Definition 5.1 in [2]. The only substantial difference is that here we consider the sequence of the current regions along a trajectory zooming in the given ε-free transition x, rather than in a state $q \in Q$. Now we shall present some properties of our sets $\mathrm{In}(x)$, $\mathrm{Out}(x)$.

Lemma 3. *l:binout Let* M *be the automaton constructed in Theorem 1, with* n ε*-free transitions, for some* $n \geq 6$*. Then, for each* ε*-free transition* $x \in \Delta$*, neither of the sets* $\mathrm{In}(x)$*,* $\mathrm{Out}(x)$ *contains more than* $\frac{1}{\log(3/2)} \cdot \log n - 1.128$ *states.*

The proof is omitted due to lack of space, but it uses the same argument as Lemma 5.2 in [2].

Lemma 4. *l:inter Let* M *be the automaton constructed in Theorem 1. Let* $x, y \in \Delta$ *be two* ε*-free transitions in* M*,* $x \neq y$*, such that there exists a path* $x \rightarrow^* y$ *in* M*. Then* $\mathrm{Out}(x) \cap \mathrm{In}(y) \neq \emptyset$*.*

Proof. Let x, y be two transitions satisfying the assumptions of the lemma. Now consider, in parallel, sequences of regions for two instances of the procedure described in Definition 2, computing the sets $\mathrm{In}(x)$, $\mathrm{Out}(x)$, and $\mathrm{In}(y)$, $\mathrm{Out}(y)$.

Both processes start with ϱ being the entire graph for M and hence the complete path $x \rightarrow^* y$ being located in the inside of the current region ϱ. The procedure from Lemma 2, based on Lemma 1, finds β' splitting ϱ into two

subregions ϱ_1, ϱ_2 with a balanced number of ε-free transitions. It should be clear that, while both transitions x, y are located in the same subregion, the two processes computing $\mathrm{In}(x)$, $\mathrm{Out}(x)$ and $\mathrm{In}(y)$, $\mathrm{Out}(y)$ follow the same trajectory of current regions.

The shared trajectory starts branching at the moment when, after splitting the current region ϱ into two subregions ϱ_1, ϱ_2, the transitions x, y fall into different subregions. This moment must come, since x does not coincide with y and the number of ε-free edges in the current region goes down to $k = 1$. At the moment of branching, there are two cases to consider:

(a) x falls in ϱ_1, but y in ϱ_2 .

Since x, y are located in different subregions, the path $x \rightarrow^* y$ must pass through the boundary separating the regions ϱ_1 and ϱ_2. More precisely, it crosses the boundary from the inside of ϱ_1 out, and thus it has to pass through the state $\mathrm{ex}_{\beta'}$, the exit point of ϱ_1. Clearly, there exists a path $x \rightarrow^* \mathrm{ex}_{\beta'}$ and hence the state $\mathrm{ex}_{\beta'}$ is inserted into the set $\mathrm{Out}(x)$. Similarly, each path coming to the subregion ϱ_2 from ϱ_1 has to pass through $\mathrm{ex}_{\beta'}$, the entry point of ϱ_2, and thus there must exist a path $\mathrm{ex}_{\beta'} \rightarrow^* y$. Hence, $\mathrm{ex}_{\beta'}$ is inserted into $\mathrm{In}(y)$. Thus, $\mathrm{ex}_{\beta'}$ is inserted into $\mathrm{Out}(x) \cap \mathrm{In}(y)$.

(b) x falls in ϱ_2, but y in ϱ_1 .

Since there exists a path $x \rightarrow^* y$ and y falls in ϱ_1, the path has to pass through the state $\mathrm{en}_{\beta'}$, the entry point of ϱ_1. Thus, there exists a path $x \rightarrow^* \mathrm{en}_{\beta'}$. But then $\mathrm{en}_{\beta'}$ is inserted in $\mathrm{Out}(x)$. By the same argument we get that $\mathrm{en}_{\beta'}$ is inserted in $\mathrm{In}(y)$. Thus, $\mathrm{en}_{\beta'}$ is inserted into $\mathrm{Out}(x) \cap \mathrm{In}(y)$.

Summing up, we have that $\mathrm{Out}(x) \cap \mathrm{In}(y) \neq \emptyset$, which completes the proof. □

Now we are ready to construct a 2-realtime automaton equivalent to the automaton presented in Theorem 1.

Definition 3. *Let M be the automaton constructed in Theorem 1. Then an automaton M', a variant of M, is constructed as follows:*

(a) The set of states in M' is $Q' = Q \cup Q_s \cup Q_t \cup \{q'_I, q'_F\}$, where Q_s and Q_t denote, respectively, the new copies of source and target states for ε-free transitions in M. More specifically, for each ε-free transition $x \in \Delta$, connecting a state $\tilde{s}_x$ with some $\tilde{t}_x$, $\tilde{s}_x, \tilde{t}_x \in Q$, that is, $\tilde{s}_x \xrightarrow{x} \tilde{t}_x \in \Delta$, insert s_x, a new copy of $\tilde{s}_x$, into Q_s, and t_x, a new copy of $\tilde{t}_x$, into Q_t.

(b) The initial state of M' is a new state q'_I. If $\varepsilon \notin L(M)$, the only final state is a new state q'_F, i.e., we have $F' = \{q'_F\}$. However, if $\varepsilon \in L(M)$, then the initial state becomes also a final state, that is, $F' = \{q'_I, q'_F\}$.

(c) If $x \in \Delta$ is an ε-free edge in M, include the following transitions in Δ':

- *a transition $s_x \xrightarrow{x} t_x$, where $s_x \in Q_s$, $t_x \in Q_t$,*
- *transitions $q \rightarrow s_x$, for each $q \in \mathrm{In}(x)$,*
- *transitions $t_x \rightarrow q'$, for each $q' \in \mathrm{Out}(x)$.*

Here $s_x \xrightarrow{x} t_x$ denotes an ε-free edge connecting the states s_x, t_x, and labeled by the same input alphabet symbol as the ε-free transition x in M. Similarly, $q \to s_x$ and $t_x \to q'$ denote ε-transitions connecting the respective states.

(d) If, for some ε-free $x \in \Delta$, there exists a path $q_I \to^* x$ in M, include a transition $q'_I \to s_x$.

(e) If, for some ε-free $x \in \Delta$, there exists a path $x \to^* q_F$ in M, include a transition $t_x \to q'_F$.

(f) If, for some ε-free $x \in \Delta$, there exists a path $x \to^* x$ in M, include a transition $t_x \to s_x$.

It should be pointed out that we do not keep multiple copies of an edge connecting the same pair of states and labeled by the same symbol, and that no other edges are included in M'. The construction, presented in Definition 3, includes certain groups of transitions in M'. Figure 1 shows these different types of edges arising from the mentioned definition.

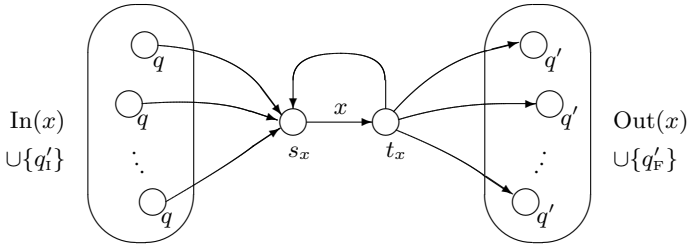


Fig. 1. Transitions in the automaton M', related to an ε-free edge x in M. The states s_x and t_x are "new" copies of the source and target states for the edge x. Edges without labels represent ε-transitions. Such edge is included only if there exists a path of ε-transitions connecting the corresponding states in M.

Theorem 3. *t:2ba The automaton M', constructed in Definition 3, is 2-realtime.*

Proof. Let us recall that an automaton is 2-realtime if and only if the length of each ε-path is at most 2. Using Definition 3, we examine all possible ε-paths in M'. There are the following cases to consider:

(i) Starting from the initial state q'_I, we can use an ε-edge $q'_I \to s_x$, for some $s_x \in Q_s$, by (d)) in Definition 3. Since s_x is a state from Q_s and there is no ε-edge starting from Q_s, this ε-path must end in s_x, and hence its length is bounded by 1.

(ii) Now we have to examine the paths starting from $s_x \in Q_s$. Though there are no ε-edges starting from s_x, we can still use an ε-free edge x from s_x to t_x, by (c) in Definition 3. Thus, we have to consider ε-paths starting from $t_x \in Q_t$.

(iii) Starting from some state $t_x \in Q_t$, we can find the following ε-paths:
   (a) $t_x \to q \to s_y$, for some $q \in Q$ and $s_y \in Q_s$, by (c) in Definition 3. There are no ε-edges starting from s_y. Thus, such ε-path is of length 2.
   (b) $t_x \to q'_F$, by (e) in Definition 3. Since there are no edges starting from q'_F, we have the ε-path of length 1.
   (c) $t_x \to s_x$, with $s_x \in Q_s$, by (f) in Definition 3. But there are no ε-edges starting from s_x. Hence, this ε-path is also of length 1.

There are no other ε-paths in M', except for those mentioned above. But then, from the previous discussion, we see that the length of any ε-path in M' is at most 2.

In particular, the only case producing an ε-path of length 2 is item (iii-a), representing a path connecting two ε-free transitions $x, y \in \Delta'$, $x \neq y$ (see Definition 3). In all other cases, an ε-edge is followed either by an ε-free edge or the final state q'_F. □

Theorem 4. *t:ekv The automata M and M', constructed in Theorem 1 and Definition 3, respectively, are equivalent.*

It is easy to show that the automaton M', constructed in Definition 3, uses at most $O(n \log n)$ ε-edges.

Theorem 5. *t:upb For each regular expression of size $n \geq 1$, there exists an equivalent nondeterministic 2-realtime automaton with at most $4n + 2$ states, n ε-free transitions, and $n \cdot (\frac{2}{\log(3/2)} \cdot \log n + 0.744)$ ε-transitions.*

For $n \geq 6$, the automaton of Definition 3 gives the required upper bounds. For $n = 2, \ldots, 5$, we use Theorem 2, for $n = 1$, the construction is trivial.

5 Conclusion

We have presented a conversion of regular expressions into nondeterministic 2-realtime automata, using at most $4n+2$ states, n ε-free transitions and $n \cdot (\frac{2}{\log(3/2)} \cdot \log n + 0.744)$ ε-edges. It would be quite easy to convert a 2-realtime automaton into a 1-realtime machine, which maintains the number of ε-free transitions and ε-transitions sufficiently small. Namely, the resulting 1-realtime automaton would have $O(n \log n)$ ε-free transitions as well as ε-transitions.

How many transitions are needed for a general case of k-realtime automata, where $k > 2$, is still an open problem. It might be possible that, by using ε-paths longer than 2, we may potentially save some transitions. This is related to the open question of whether there exists a fixed constant k such that, for each n, the resulting k-realtime automaton will have only $O(n)$ transitions. The best (rather trivial) construction of a realtime automaton with a linear number of transitions, known to the author, can be obtained by using the automaton of Theorem 1 as a starting point, in which we eliminate all cycles composed of ε-edges by unifying all states in such a cycle into a single state. However, this

results in an $(n-1)$-realtime automaton, that is, in $k = n-1$, which is not a fixed constant.

In Table 1, we give a survey on complexity of different types automata resulted from conversions of regular expressions.

Type of automata	Number of states	Number of ε-free edges	Number of ε-edges
ε-free automata (0-realtime)	$O(n)$	$O(n \log^2 n)$	0
1-realtime automata	$O(n)$	$O(n \log n)$	$O(n \log n)$
2-realtime automata	$O(n)$	$O(n)$	$O(n \log n)$
k-realtime automata ($k > 2$)	?	?	?
ε-automata	$O(n)$	$O(n)$	$O(n)$

Tab. 1. Conversion of regular expressions into different types of nondeterministic automata and their descriptional complexity.

References

1. A. Ehrenfeucht and P. Zieger, Complexity Measures for Regular Expressions. J. Comput. System Sci., 12, 1976, 134–46
2. V. Geffert, Translation of Binary Regular Expressions into Nondeterministic ε-Free Automata with $O(n \log n)$ Transitions. J. Comput. System Sci., 67, 2003, 451–72
3. S. Ginsburg, Algebraic and Automata-Theoretic Properties of Formal Languages. North-Holland, 1975
4. J. E. Hopcroft and J. D. Ullman, Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, 1979
5. J. Hromkovič, S. Seibert, and T. Wilke, Translating Regular Expressions into Small ε-Free Nondeterministic Automata. In Proc. Symp. Theoret. Aspects Comput. Sci., Lect. Notes Comput. Sci., 1200, Springer-Verlag, 1997, 55–66
6. J. Hromkovič, S. Seibert, and T. Wilke, Translating Regular Expressions into Small ε-Free Nondeterministic Finite Automata. J. Comput. System Sci., 62, 2001, 565–88
7. Y. Lifšic, A Lower Bound on the Size of ε-Free NFA Corresponding to a Regular Expression. Manuscript, St. Petersburg State University, 2002
8. S. Sippu and E. Soisalon-Soininen, Parsing Theory, Vol. I: Languages and Parsing, EATCS Monographs of Theoret. Comput. Sci., 15, Springer-Verlag, 1988

Ne-metrické vzdálenosti

Michal Kolovrat and Václav Snášel

Katedra informatiky, FEI, VŠB - Technická univerzita Ostrava,
17. listopadu 15, 708 33, Ostrava-Poruba, Česká republika
{michal.kolovrat, vaclav.snasel}@vsb.cz

Abstrakt Hlavní myšlenkou tohoto příspěvku je ukázat, že kromě klasických metrik existuji i ne-metrické vzdálenosti. Obecně metrika (míra sloužící k měření vzdálenosti v metrických prostorech) musí splňovat 3 základní axiomy. Ty budou uvedemy v úvodní kapitole. Pro ne-metrické vzdálenosti většinou selhává axiom trojúhelníkové nerovnosti. Příspěvek bude obsahovat ukázky těchto ne-metrických vzdáleností, jejich vlastnosti a využití. Dále uvedeme, jak lze pomocí jednoduchého koeficientu modifikovat axiom trojúhelníkové nerovnosti, aby mohl být využit v ne-metrických vzdálenostech. V testech ukážeme, že prakticky pro každou ne-metrickou vzdálenost tento koeficient existuje a lze ho určit. Ukážeme také, zda-li a jak tento koeficient zavisí na vlastnostech dat. Uvedeme důkaz existence tohoto koeficientu a jeho teoretického odvození.

Keywords: Ne-metrická vzdálenost, Levenshtein, editační vzdálenost.

1 Úvod

Při indexování a vyhledávání v multimediálních datech je nutné řešit problematiku podobnosti objektů. Pro tyto objekty můžeme definovat několik prostorů, ve kterých mohou být reprezentovány. Jsou to vektorové, metrické a ne-metrické prostory. Podobnost je velmi často modelována pomocí metriky. V mnoha případech se však ukazuje metrika nevyhovující viz. [2].

V tomto článku bychom chtěli také ukázat, jak je možné použít datové struktury pro indexování metrických prostorů pro indexování jisté třídy ne-metrických vzdáleností.

Definice: Metrika v neprázdné množině M je zobrazení $p : M \times M \rightarrow R$ s těmito vlastnostmi:

1. p je nezáporná funkce taková, že $p(x, y) = 0$ právě tehdy, když $x = y$
2. $p(x, y) = p(y, x)$
3. $p(x, z) \leq p(x, y) + p(y, z)$ (trojúhelníková nerovnost)

První vlastnost (axiom) vyjadřuje pozitivitu metriky, druhý symetrii a třetí trojúhelníkovou nerovnost. Vektorový prostor je zároveň metrickým prostorem, avšak opačně toto vždy neplatí. V ne-metrických prostorech nejen že nelze definovat n-rozměrný vektor reprezentující objekt, ale nelze definovat také metriku.

To z toho důvodu, že v těchto prostorech lze obtížně pochopit intuitivní podobnost jako funkci dvou objektů popř. bodů v prostoru. Mnoho reálných aplikací je založeno na vektorovém modelu, ale existuje řada aplikací využívající ne-metrické vzdálenosti. Z jiného úhlu pohledu jsou metrické axiomy omezujícím faktorem funkcí, které však člověk intuitivně chápe a potřebuje. V různých oblastech se může zdát překážkou trojúhelníková nerovnost. Právě tento axiom nejčastěji selhává v ne-metrických prostorech.

Ne-metrické vzdálenosti se objevují v mnoha aplikačních oblastech, jako například řetězcové (DNA) porovnávání, filtrování zákazníků (porovnání zákazníka s jeho prototypem) a také u databází obrázků.

Máme-li metriku d, tak z ní můžeme vyrobit ne-metrickou vzdálenost $d^{\cdot}$. To lze provést například tak, že celou metriku umocníme kladným číslem. Obecný zápis převodu metriky na ne-metrickou vzdálenost v tomto konkrétním případě je $d \rightarrow d^n$, kde $n > 1$. Pro d^n pak selhává podmínka (axiom) trojúhelníkové nerovnosti. Získaná ne-metrická vzdálenost je velmi podobná svými vlastnostmi vzdálenosti L^p pro parametr $p \in (0, 1)$.

Tento příspěvek se zabývá několika konkrétními ne-metrickými vzdálenostmi, jejich vlastnostmi a využitím. Dále ukazuje modifikaci axiomu trojúhelníkové nerovnosti pomocí koeficientu κ (kappa) a jeho použití u ne-metrických vzdáleností. V testech ukážeme existenci takového koeficientu a pokusíme se zjistit, jak je závislý na datech a dimenzi. Uvedeme také teoretický důkaz a odvození tohoto koeficientu. Cílem tohoto výzkumu je do budoucna umět, pomocí metrické datové struktury, indexovat ne-metrická data nebo objekty z ne-metrických prostorů. Nejlepší strukturou pro takovou indexaci bude M-strom. Umí indexovat konečné metrické prostory. Modifikací M-stromu získáme strukturu pro indexování ne-metrické vzdálenosti splňující κ-trojúhelníkovou nerovnost.

Jednou ze základních metrických datových struktur je M-strom [4], [5], umožňující indexovat kolekci dokumentů modelovaných objekty metrického prostoru $M = (U, d)$, kde U je univerzum objektů (např. vektorů dokumentů) a d je metrika. Je založen na myšlence B^+-stromu, což je dynamická a vyvážená datová struktura. Konkrétní index M-stromu představuje hierarchii metrických regionů (každý uzel představuje jeden region), respektive hierarchii shluků objektů v těchto regionech. Struktura M-stromu je velmi jednoduchá. Listy obsahují záznamy samostatných indexovaných objektů, zatímco vnitřní uzly obsahují tzv. směrovací záznamy. Tyto záznamy popisují tzv. metrické regiony, které vymezují v metrickém prostoru oblast, v níž se nacházejí objekty uložené v listech příslušného podstromu. Metrický region je popsán hyper-koulí se středem v nějakém objektu a příslušným pokrývajícím poloměrem. Indexování objektů v M-stromu je realizováno pouze pomocí dané metriky d.

Jak bylo tedy naznačeno, M-strom indexuje pouze metrická data. Avšak modifikací metriky d a úpravou trojúhelníkové nerovnosti na κ-trojúhelníkovou nerovnost lze docílit toho, že pomocí modifikovaného M-stromu půjdou indexovat i ne-metrická data.

V tomto příspěvku se tedy budeme zabývat modifikací trojúhelníkové nerovnosti, jejím otestováním na ne-metrických vzdálenostech a ne-metrických datech. Dokážeme a teoreticky odvodíme existenci koeficientu κ.

2 Příklady ne-metrických vzdáleností

2.1 Ne-metrická L^p vzdálenost

L^p vzdálenost (nebo jinak p-vzdálenost) mezi dvěma vektory $\mathbf{x}$, $\mathbf{y}$, je definována jako:

$$d(x,y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{\frac{1}{p}}$$

Pro $p = 2$ označujeme tuto vzdálenost jako Euclidovskou, a pro $p = 1$ jako vzdálenost Manhattan. Tato vzdálenost je ne-metrická pro hodnotu parametru $p < 1$.

Trojúhelníková nerovnost pro ne-metrické L^p vzdálenosti. Je známo, že pro L^p vzdálenost, kde $p < 1$, není zcela zaručena platnost trojúhelníkové nerovnosti: mějme 3 body $q_1, q_2, q_3 \in R^n$, $d(q_1, q_3) > d(q_1, q_2) + d(q_2, q_3)$. Přesněji, ať $\mathbf{x}$ označuje vektor spojující q_1, q_2 a $\mathbf{y}$ vektor spojující q_2, q_3; potom

$$\left(\sum_{1}^{n} |x_i + y_i|^p\right)^{\frac{1}{p}} > \left(\sum_{1}^{n} |x_i|^p\right)^{\frac{1}{p}} + \left(\sum_{1}^{n} |y_i|^p\right)^{\frac{1}{p}}$$

Zde je důkaz existence koeficientu $\kappa > 1$ takového, že κ-ohraničená trojúhelníková nerovnost je splněna:

$$d(q_1, q_3) \leq \kappa(d(q_1, q_2) + d(q_2, q_3))$$

více přesněji,

$$\left(\sum_{1}^{n} |x_i + y_i|^p\right)^{\frac{1}{p}} \leq \kappa \left(\left(\sum_{1}^{n} |x_i|^p\right)^{\frac{1}{p}} + \left(\sum_{1}^{n} |y_i|^p\right)^{\frac{1}{p}}\right) \tag{1}$$

Nejdříve, z konkávnosti funkce $f(t) = t^p$ pro $p < 1$, a pro $a, b \geq 0$:

$$\frac{1}{2^{1-p}}(a^p + b^p) \leq (a+b)^p \leq a^p + b^p$$

kde první nerovnost pochází z $\frac{(a^p+b^p)}{2} \leq \left(\frac{a+b}{2}\right)^p$. Nechť $a = \left(\sum_{1}^{n} |x_i|^p\right)^{\frac{1}{p}}$, $b = \left(\sum_{1}^{n} |y_i|^p\right)^{\frac{1}{p}}$; potom

$$\left(\left(\sum_{1}^{n} |x_i|^p\right)^{\frac{1}{p}} + \left(\sum_{1}^{n} |y_i|^p\right)^{\frac{1}{p}}\right)^p = (a+b)^p$$

$$\geq \frac{1}{2^{1-p}}(a^p + b^p) = \frac{1}{2^{1-p}}\left(\sum_1^n |x_i|^p + \sum_1^n |y_i|^p\right) = \frac{1}{2^{1-p}}\sum_1^n (|x_i|^p + |y_i|^p)$$
$$\geq \frac{1}{2^{1-p}}\sum_1^n (|x_i| + |y_i|)^p$$
$$\geq \frac{1}{2^{1-p}}\sum_1^n |x_i + y_i|^p$$

Tedy (1) platí pro $\kappa = 2^{\frac{1-p}{p}}$. Dále pak, tato nerovnost nemůže být zlepšena; v nejhorším případě, pro který platí rovnost, je případ kdy n předpokládáme sudé, $\mathbf{x}$ je vektor, ve kterém jsou sudé prvky rovny 1 a liché prvky jsou nulové, a $\mathbf{y}$ je vektor s lichými prvky rovnými 1 a sudými rovnými nule.
Je nutné poznamenat, že pro menší p je větší koeficient κ, a dochází ke vzdálení od p-vzdálenosti.

Příklad ohraničené κ-trojúhelníkové nerovnosti. Jednoduchým příkladem aplikace koeficientu κ může být obecná euklidovská metrika a druhá mocnina této metriky. V první metrice mějme například vzdálenosti $d(x, z) = 1$, $d(x, y) = 0.7$ a $d(y, z) = 0.7$. V této metrice platí podmínka trojúhelníkové nerovnosti pro vzdálenosti mezi jednotlivými body. Nyní změníme tuto vzdálenost d na d^2 (použijeme druhou mocninu metriky). Vzdálenosti nyní budou $d(x, z) = 1$, $d(x, y) = 0.49$ a $d(y, z) = 0.49$. Toto již není metrika, protože došlo k porušení trojúhelníkové nerovnosti. Využijeme nerovnosti

$$d(q_1, q_3) \leq \kappa(d(q_1, q_2) + d(q_2, q_3))$$

pro náš příklad

$$d(x, z) \leq \kappa(d(x, y) + d(y, z))$$

a pro $\kappa = 2$ nerovnost platí.

Toto je velmi jednoduchý příklad aplikace κ v podmínce trojúhelníkové nerovnosti. Je zde vidět, že koeficient lze použít na obecnou metriku. Zde jsme totiž uvažovali euklidovskou metriku a její druhou mocninu, vzdálenosti byly uvedeny jen pro konkrétní představu.

2.2 Ne-metrická k-mediánová vzdálenost

Tato vzdálenost mezi dvěma vektory (nebo obrázky, reprezentovanými pomocí vektorů)
$\mathbf{x} = (x_1, x_2, \ldots, x_n)$ a $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ je definována jako:

$$d(x, y) = \text{k-median}\left\{|x_1 - y_1|, \ldots, |x_n - y_n|\right\}$$

kde operace k-median vrací k-tou hodnotu setříděného rozdílového vektoru.

2.3 Editační vzdálenost

Tato vzdálenost je podobná níže uvedené Levenshteinově. Editační vzdálenost mezi řetězcem f a řetězcem g, označená jako $d(f, g)$, je minimální počet operací editace (vložení, smazání, výměna a přesunutí) potřebných k transformaci řetězce f na řetězec g.

2.4 Levenshteinova vzdálenost

Levenshteinova vzdálenost (LD) je míra podobnosti mezi dvěma řetězci, tedy míra podobnosti mezi zdrojovým řetězcem (s) a cílovým řetězcem (t). Vzdálenost je počet operací mazání, vložení nebo substitucí potřebných k transformaci řetězce s na řetězec t. Například:

- Jestliže s je "test" a t je "test", pak LD(s,t) = 0, poněvadž žádné transformace nejsou potřeba. Řetězce jsou totiž identické.
- jestliže s je "test" a t je "tent", pak LD(s,t) = 1, poněvadž je třeba jedno nahrazení (změna "s" na "n") k transformaci s na t.

Čím více jsou rozdílné řetězce, tím větší je jejich vzdálenost. Levenshteinova vzdálenost je pojmenována podle ruského vědce Vladimíra Levenshteina, který navrhl tento algoritmus v roce 1965. Jak již bylo zmíněno, tato vzdálenost se velmi podobá editační vzdálenosti.

Algoritmus pracující s Levenshteinovou vzdáleností je využíván například při:

- Kontrole pravopisu
- Rozpoznávání řeči
- DNA analýze
- Detekci kopií a opisů

Editační a Levenshteinova vzdálenost jsou metriky v obecném případě. Avšak existují jejich modifikace pro práci s řetězci, které již metrické nejsou. Těmito modifikacemi se v tomto článku zabývat nebudeme, neboť testy budeme provádět na L^p a k-mediánové vzdálenosti.

3 Experimenty a testy

Cílem experimentů a testů bylo ukázat (na datech a vzdálenostech), že při neplatnosti trojúhelníkové nerovnosti lze najít koecifient κ takový, aby byla splněna κ-trojúhelníková nerovnost.

Testy byly prováděny následovně: vygenerovali jsme náhodná data (body s rovnoměrným rozložením v prostoru) a na tyto body jsme aplikovali konkrétní ne-metrickou vzdálenost. Vytvořili jsme tak všechny kombinace vzdáleností mezi vygenerovanými body. Pro zobrazení průběhu koeficientu κ v závislosti na datech bylo nutné hodnotu tohoto koeficientu vypočítat. To jsme provedli pomocí matematické úpravy κ-trojúhelníkové nerovnosti na tvar:

$$\kappa \geq \frac{d(x,z)}{d(x,y)+d(y,z)}$$

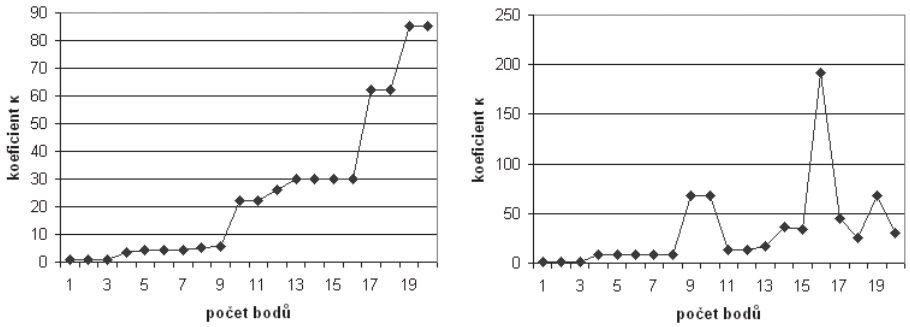
a do tohoto vzorce jsme postupně dosadili všechny předem vypočítané vzdálenosti mezi body. Pokud koeficient κ měl hodnotu menší než 1, bylo jasné, že trojúhelníková nerovnost je splněna. Důležitější však byl případ, kdy koeficient přesáhl hodnotu 1. Pro danou dimenzi a všechny kombinace vzdáleností mezi body jsme si vždy zapamatovali nejvyšší hodnotu koeficientu κ. Tím jsme vždy zaručili, ze pro všechny vzdálenosti platí κ-trojúhelníková nerovnost. Na následujících grafech je vidět průběh koeficientu κ v závislosti na použité vzdálenosti, dimenzi a množství dat.

3.1 Průběh koeficientu κ v závislosti na dimenzi a množství dat

Z testů je patrné, že vždy existuje konečný koeficient κ pro konečnou množinu dat. Tento poznatek bude využit v budoucím výzkumu, neboť jak již bylo zmíněno, modifikací (zavedením koeficientu) metrické datové struktury M-strom bude možné pomocí semimetriky (ne-metrické vzdálenosti) indexovat data. Jinak řečeno, bude možné indexovat metrická a ne-metrická data.

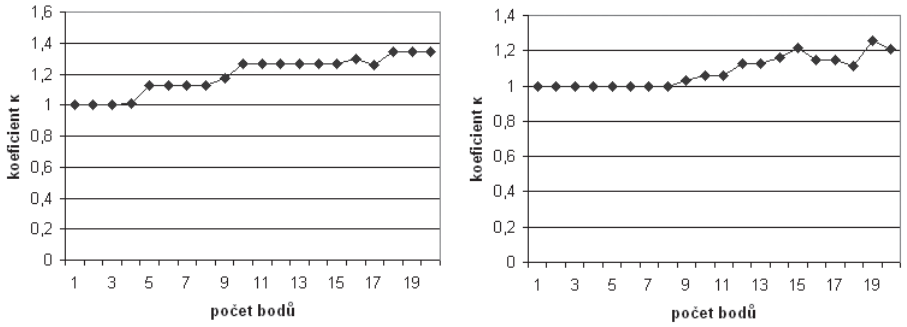
4 Závěr

V příspěvku jsme uvedli pojem metriky, metrických prostorů a možnosti indexování metrických dat. Vysvětlili jsme pojem ne-metrické vzdálenosti a naznačili, jak bude možné pomocí upravené metrické datové struktury M-strom indexovat ne-metrická data. Implementace a otestování této datové struktury bude součástí dalšího výzkumu.



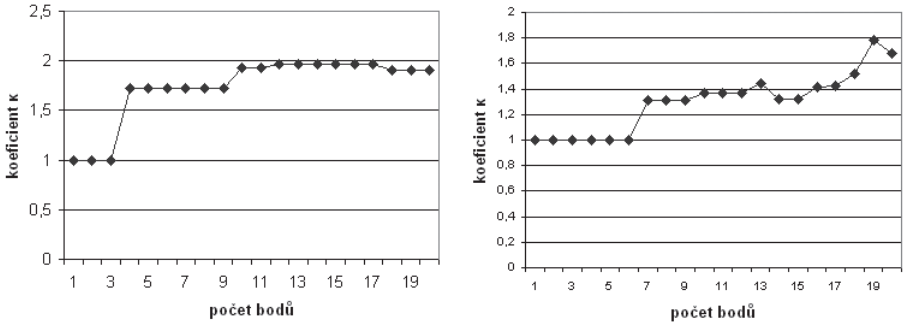
(a) Dimenze prostoru 4 (b) Dimenze prostoru 8

Obr. 1. K-mediánová vzdálenost.



(a) Dimenze prostoru 4 (b) Dimenze prostoru 8

Obr. 2. L^p vzdálenost (pro p=0.5).



(a) Dimenze prostoru 4 (b) Dimenze prostoru 8

Obr. 3. Druhá mocnina Euclidovské L^p vzdálenosti (p=2).

V další části příspěvku jsme uvedli konkrétní ne-metrické vzdálenosti, pro které selhává trojúhelníková nerovnost. Uvedli jsme teoretický důkaz existence parametru κ a vyčíslili jeho hodnotu pro konkrétní typ dat. Navrhli jsme tzv. κ-trojúhelníkovou nerovnost, která bude použita při indexování ne-metrických dat. Otestovali jsme námi upravenou trojúhelníkovou nerovnost a zjistili, že koeficient κ existuje a je závislý na typu dat a použité ne-metrické vzdálenosti. Na závěr tedy můžeme konstatovat, že pro konečnou množinu dat a tedy konečnou množinu nerovností lze najít také konečný koeficient κ. Tento koeficient bude sloužit pro datovou strukturu MK-strom jako parametr při indexování ne-metrických dat.

Reference

1. T. Skopal, M. Krátký, and V. Snášel, Metric and Semi-Metric Indexing of Vector Models in Information Retrieval Systems (in czech). In Proceedings of the 3rd Conference ZNALOSTI 2004, Brno, Czech Republic, 2004, 154-–165
2. D.W. Jacobs, D. Weinshall, and Y. Gdalyahu, Class Representation and Image Retrieval with Non-Metric Distances. Proceedings of International Conference Computer Vision 1998
3. A. Papadopoulos and Y. Manolopoulos, Distributed Processing of Similarity Queries. In Distributed and Parallel Databases, 2001, 67–92
4. T. Skopal, J. Pokorný, M. Krátký, and V. Snášel, Revisiting M-tree Building Principles. In Proceedings of the 7th East-European conference on Advances in Databases and Informations Systems (ADBIS), LNCS 2798, Springer-Verlag, Dresden,Germany, 2003, 148–162
5. T. Skopal, Pivoting M-tree: A Metric Access Method for Efficient Similarity Search. In Proceedings of the 4th annual workshop DATESO, Desná, Czech Republic, ISBN 80-248-0457-3, also available at CEUR, Volume 98, ISSN 1613-0073, `http://www.ceur-ws.org/Vol-98`, 2004, 21-–31

Learning with kernel based regularization networks *

Petra Kudová

Institute of Computer Science, Academy of Sciences of CR,
Pod Vodárenskou věží 2, P.0. Box 5, 182 07 Prague 8, Czech Republic
petra@cs.cas.cz

Abstract. The problem of *learning from examples*, also well known as *supervised learning*, is a subject of great interest at present. We discuss one approach to this problem – the kernel based regularization networks. We work with a learning algorithm introduced in [6]. Since regularization networks has very good theoretical background, we are interested in its practical aspects and applicability on real tasks. The key point of successful applying of a basic learning algorithm on a particular task is the choice of kernel type and a regularization parameter. We will describe techniques we use to estimate these 'explicit' parameters and discuss their performance. We will shortly introduce special types of kernels: *Product* kernels and *Sum* kernels (joint work with T. Šámalová). The performance of described algorithms will be demonstrated on experiments, including a comparison of different types of kernels.

1 Introduction

The problem of *learning from examples* (also called *supervised learning*) is a subject of great interest. The need for a good supervised learning technique stems from a wide range of application areas, covering various approximation, classification, and prediction tasks.

Kernel methods represent a modern family of learning algorithms. A comprehensive overview of kernel learning algorithms can be found in [8], [9], [1].

In this work we study one type of a kernel based learning algorithm, regularization network, a feed-forward neural network with one hidden layer, derived from the regularization theory. While it is well studied from the mathematical point of view, we are more interested in practical points of its application.

The theoretically derived learning algorithm has an extra parameters, that are supposed to be known in advance. They are the regularization parameter and the type of a kernel. Both are dependent on a current data task and there is no easy way, how to select them. In section 4 we will show a technique that we are using for their selection and propose a new algorithm based on evolutionary principles.

* This work was supported by GA ČR grant 201/05/0557 and by the Institutional Research Plan AV0Z10300504 "Computer Science for the Information Society: Models, Algorithms, Appplications"

2 Regularization networks

Our problem can be formulated as follows. We are given a set of examples (pairs) $\{(\boldsymbol{x}_i, y_i) \in R^d \times R\}_{i=1}^N$ that was obtained by random sampling of some real function f, generally in the presence of noise (see 1). To this set we refer as *a training set*. Our goal is to recover the function f from data, or to find the best estimate of it. It is not necessary that the function exactly interpolates all the given data points, but we need a function with a good *generalization*, that is a function that gives relevant outputs also for the data not included in the training set.

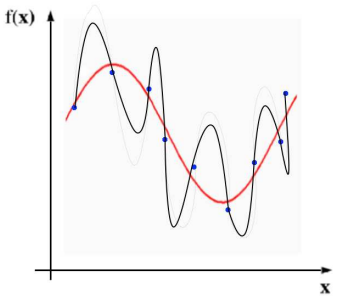


Fig. 1. The problem of learning from examples.

This problem is generally ill-posed, there are many functions interpolating the given data points, but not all of them exhibit also the generalization ability.

Therefore we have to consider some a priori knowledge about the function. It is usually assumed that the function is *smooth*, in the sense that two similar inputs corresponds to two similar outputs and the function does not oscillate too much. This is the main idea of the regularization theory, where the solution is found by minimizing the functional (1) containing both the data term and the smoothness information.

$$H[f] = \frac{1}{N}\sum_{i=1}^{N}(f(\boldsymbol{x}_i) - y_i)^2 + \gamma\Phi[f], \tag{1}$$

where Φ is called a *stabilizer* and $\gamma > 0$ is *the regularization parameter* controlling the trade-off between the closeness to data and the smoothness of the solution.

Poggio and Smale in [6] studied the regularization networks derived using a Reproducing Kernel Hilbert Space (RKHS) as the hypothesis space. Let $\mathcal{H}_K$ be an RKHS defined by a symmetric, positive-definite kernel function $K_{\boldsymbol{x}}(\boldsymbol{x}') = K(\boldsymbol{x}, \boldsymbol{x}')$. Then if we define the stabilizer by means of norm $||\cdot||_K$ in $\mathcal{H}_K$ and minimize the functional

$$\min_{f\in\mathcal{H}_K} H[f], \quad \text{where} \quad H[f] = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(\boldsymbol{x}_i))^2 + \gamma||f||_K^2 \tag{2}$$

over the hypothesis space $\mathcal{H}_K$, the solution of minimization (2) is unique and has the form

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} w_i K_{\boldsymbol{x}_i}(\boldsymbol{x}), \qquad (N\gamma I + K)\boldsymbol{w} = \boldsymbol{y}, \tag{3}$$

where I is the identity matrix, K is the matrix $K_{i,j} = K(\boldsymbol{x_i}, \boldsymbol{x_j})$, and $\boldsymbol{y} = (y_1, \ldots, y_N)$.

The solution (3) can be represented by a feed-forward neural network with one hidden layer and a linear ouput layer (see (2)). We refer to a network of this form as *regularization network*. The units of hidden layer realize the *kernel function* $K(\cdot, \cdot)$, with the first argument fixed to $\boldsymbol{c}_i$. We will refer to the vectors $\boldsymbol{c}_i$ as *centers*. In the optimal solution (3), they are fixed to the data points $\boldsymbol{x}_i$.

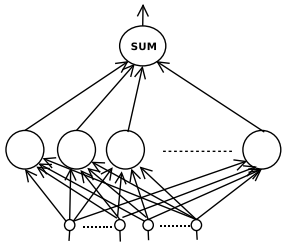


Fig. 2. Regularization network scheme.

3 Learning algorithm

The regularization network scheme derived in previous section leads to the RN learning algorithm (Algorithm 3.1).

Input: Data set $\{\boldsymbol{x}_i, y_i\}_{i=1}^N \subseteq \mathcal{R}^n \times \mathcal{R}$
Output: Regularization network.

1. Set the centers of kernels:

$$\forall i \in \{1, \ldots, N\} : \boldsymbol{c}_i \leftarrow \boldsymbol{x}_i$$

2. Compute the values of weights $w_1, \ldots, w_N$:

$$(N\gamma I + K)\boldsymbol{w} = \boldsymbol{y}, \tag{4}$$

where I is the identity matrix,
$K_{i,j} = K_1(\boldsymbol{c}_i, \boldsymbol{x}_j)$,
and $\boldsymbol{y} = (y_1, \ldots, y_N)$, $\gamma > 0$.

Algorithm 3.1. RN learning algorithm.

The algorithm is simple and effective for data sets of small and medium size, i.e. those for which we can solve the linear system (4) by means of available numerical software using current computational resources.

The tasks with huge data sets are hard to solve using this simple algorithm, and they lead to solutions of implausible size as well. Other algorithms should be used in such cases, one option can be RBF networks belonging to the family of generalized regularization networks (see [3] for a comparison of RBF networks and regularization networks). Alternatively, in [10] we proposed the *Divide at Impera* approach, dividing the task to several smaller subtasks, applying the basic algorithm on these subtasks and then obtaining the resulting network as a sum of sub-results.

In this paper, we will consider only tasks that are solvable using the Algorithm 3.1.

The strength of the algorithm stems from the fact, that the linear system we are solving is well-posed, i.e. it has a unique solution and the solution exists. (It has N variables, N equations, K is positive-definite and $(N\gamma I + K)$ is strictly positive.)

We are interested in its real performance in the first place. Therefore we are also asking the question, if the system is well-conditioned, i.e. insensitive to small perturbations of the data.

The rough estimate of well-conditioning is the condition number of the matrix $(N\gamma I + K)$. We know that the condition number is small, if $N\gamma$ is large, i.e. the matrix has a dominant diagonal. Unfortunately, we are not entirely free to choose γ, because with too large γ we loose the closeness to data. See figure 3b.

Both the numerical stability and the real performance of the algorithm depends significantly on the choice of the regularization parameter γ and the type of kernel. These parameters are explicit parameters of the algorithm and their optimal choice always depends on the particular task. We will discuss the techniques we use to estimate these parameters in the next section.

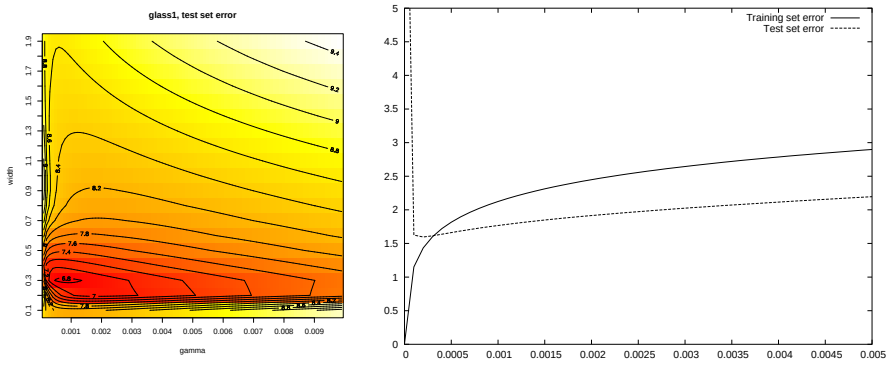


Fig. 3. a) The dependence of the error function (computed on test set) on parameters γ and b. b) The relation between γ and training and testing error.

4 Estimation of regularization parameters and types of kernel

As we showed in the previous section, the choice of the explicit parameters (γ and the type of kernel) are crucial for the successful application of the Algorithm 3.1. There exist no easy way for estimation of these parameters, usually some kind of an exhaustive search for parameters with the lowest cross-validation error is used (The choice of regularization parameter was discussed for example in [2].)

First we need a measure of a real performance of the network, that enables us to say that one particular choice of parameters is better than another. We use k-fold cross-validation to estimate a real performance of the network and its generalization ability.

Suppose we are given a training set of data $TS = \{\boldsymbol{x}_i, y_i\}_{i=1}^N \subseteq \mathcal{R}^n \times \mathcal{R}$. We split this data set randomly into k folds $TS_1, \ldots, TS_k$ such as $\bigcup_{i=1}^k TS_i = TS$ and $TS_i \bigcap_{i \neq j} TS_j \neq 0$. Then f_i is the network obtained by the Algorithm 3.1 run on the data set $\bigcup_{j \neq i} TS_j$. The cross-validation error is given by

$$E_{cross} = \frac{1}{k} \sum_{i=1}^{k} \frac{1}{|TS_i|} \sum_{(\boldsymbol{x},y) \in TS_i} (f_i(\boldsymbol{x}) - y)^2. \tag{5}$$

In our experiments we use Gaussian kernels, so the choice of a kernel type is reduced to the choice of Gaussian width. We use the *Adaptive grid search* (Algorithm 4.1) for estimation of the regularization parameter and the width of Gaussian kernels. The character of dependency of the error function on these parameters, as shown on Fig. 3a, enables us to start with a coarse grid and then create a finer grid around the point with the lowest cross-validation error.

Input: Data set $\{\boldsymbol{x}_i, y_i\}_{i=1}^N \subseteq \mathcal{R}^n \times \mathcal{R}$
Output: Parameters γ and a width b.

1. Create a set of couples $\{[\gamma, b]_i, i = 1, \ldots, K\}$, uniformly distributed in $< \gamma_{min}, \gamma_{max} > \times < b_{min}, b_{max} >$.
2. For each $[\gamma, b]_i$ for $i = 1, \ldots, K$ and for each couple evaluate the cross-validation error (5) E^i_{cross}.
3. Select the i with the lowest E^i_{cross}.
4. If the couple $[\gamma, b]_i$ is at the border of the grid, move the grid. (see Fig. 4a)
5. If the couple $[\gamma, b]_i$ is inside the grid, create finer grid around this couple. (see Fig. 4b)
6. Go to 2 and iterate until cross-validation error stops decreasing.

Algorithm 4.1. Adaptive grid search.

The winning values of parameters found by the Algorithm 4.1 are then used to run the Algorithm 3.1 on the whole training set.

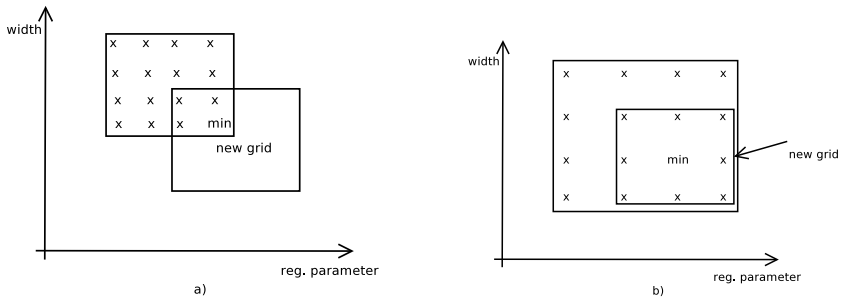


Fig. 4. a) Move grid, b) Create finer grid.

The disadvantage of this approach is the high number of evaluations of the Algorithm 3.1 needed during the search, and also observed over-fitting with respect to a particular partitioning to k folds. By the over-fitting we mean that it often happens that the real generalization ability of the network obtained using parameters with lower cross-validation error on the particular partitioning to k folds may be worse than the one of the network obtained using parameters with a higher cross-validation error.

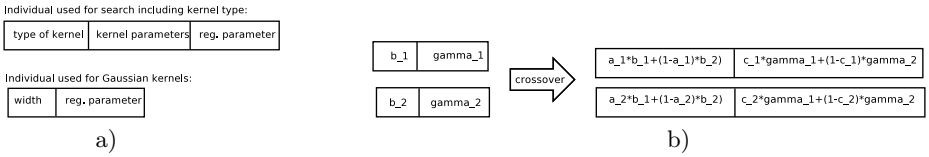


Fig. 5. a) Individuals, b) Crossover for Gaussian kernels.

Therefore we introduce a simple genetic algorithm to our search, see *Genetic parameter search* (Algorithm 4.2). We will work with a population of individuals (see Fig. 5a)), each representing one setting of parameters. Each individual is associated with its *fitness* value. This fitness should reflect the cross-validation error corresponding to the parameters represented by this individual. The lower the cross-validation error, the higher the fitness value.

Because the evaluation of fitness is very expensive in terms of time requirements, we will use so called *lazy evaluations*. By lazy evaluations we mean, that each time the fitness of the individual is being computed, only one fold is evaluated, i.e. only one term of the sum in (5) is evaluated. In other words, instead of the cross-validation error (5) we compute only estimate of it. Each evaluation of the fitness, another fold is evaluated and this estimate is made more accurate. In addition, this enables us to avoid the over-fitting by selecting a new random partitioning each time a fitness is evaluated.

New generations of individuals are created using operators *selection*, *crossover* and *mutation*. Mutation introduces small random perturbation to existing in-

Input: Data set $\{\boldsymbol{x}_i, y_i\}_{i=1}^N \subseteq \mathcal{R}^n \times \mathcal{R}$
Output: Parameters γ and a width b.

1. Create randomly an initial population P_0 of M individuals.
2. Reset individuals' counters.
   for $i = 1, \ldots, M : c_i = 0, err_i = 0$
3. $i \leftarrow 0$
4. $P_{i+1} \leftarrow$ empty set
5. $I_1 \leftarrow selection(P_i); I_2 \leftarrow selection(P_i)$
6. with probability p_{cross}: $(I_1, I_2) \leftarrow crossover(I_1, I_2)$
7. with probability p_{mutate}: $I_k \leftarrow mutate(I_k), k = 1, 2$
8. insert I_1, I_2 into P_{i+1}
9. if P_{i+1} has less then M individuals goto 5
10. for $j = 1, \ldots M : I_j \in P_{i+1}$
    divide the data set randomly to T_{train} and T_{eval}, run
    the Alg. 3.1 on T_{train} using parameters defined by I_j
    $E \leftarrow$ error of the resulting network on T_{eval}
    $err_j \leftarrow err_j + E$
    $c_j \leftarrow c_j + 1$
    fitness$(I_j) \leftarrow C - \frac{1}{c_j} err_j$
11. $i \leftarrow i + 1$
12. goto 4 and iterate until the fitness stops increasing

Algorithm 4.2. Genetic parameter search.

dividuals. Crossover (Fig. 5b), creates two new individuals from two existing individuals by choosing new parameters values randomly in the interval formed by the old values. Classical roulette-wheel selection is used (the higher the fitness, the higher the probability of being selected).

Another advantage of this simple evolutionary approach is that it can be simply modified to search also for the type of a kernel. Then the population consists of different types of individuals (species) and we have two possibilities of *crossover*. The former works, as it was described, only with individuals of the same type, so we have to ensure that always individuals of the same kind are selected. The latter combines different kind of kernels together using operator product and sum as described in the next section. We recommend to use the first type of crossover or aloud only simple combinations of kernels, otherwise the time-requirements of the search may increase significantly.

5 Construction of composite kernels

In [10] (joint work with T. Šámalová) we proposed two types of composite kernels, Product kernels and Sum kernels.

By a *Sum kernel* (see fig 6a) we mean a unit with n real inputs and one real output. It consists of two positive definite kernel functions $K_1(\boldsymbol{c}, \cdot)$, $K_2(\boldsymbol{c}, \cdot)$, both evaluating the same input vector. The output of the sum unit is computed as the sum $K_1(\boldsymbol{c}, \boldsymbol{x}) + K_2(\boldsymbol{c}, \boldsymbol{x})$.

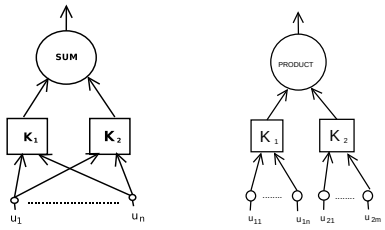


Fig. 6. a) Sum Unit, b) Product Unit.

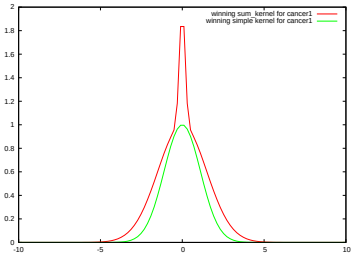


Fig. 7. Kernels found by parameter search for cancer1 data set.

By a *Product kernel* (see fig 6b) we mean an unit with $(n + m)$ real inputs and one real output. It consists of two positive definite kernel functions $K_1(\boldsymbol{c}_1, \cdot)$, $K_2(\boldsymbol{c}_2, \cdot)$, one evaluating the first n inputs and one evaluating the other m inputs. The output of the product unit is computed as the product $K_1(\boldsymbol{c}_1, \boldsymbol{x}_1) \cdot K_2(\boldsymbol{c}_2, \boldsymbol{x}_2)$.

In [10] we have shown that product (resp. sum) of two kernel function is again a kernel function and that products (resp. sums) of kernel functions can be used in the Algorithm 3.1 as a kernel. Product kernels are supposed to be used on data with different types of attributes, since different groups of attributes can be processed by different kernel functions. See also [4].

6 Experiments

The goal of our experiments was to demonstrate the performance of regularization networks and compare different kinds of kernels (Gaussian kernels, Product kernels and Sum kernels).

Gaussian kernels were used, regularization parameter and width were estimated by the Adaptive grid search (Alg. 4.1). Experiments with the Genetic parameter search (Alg. 4.2) are in progress.

The standard numerical library LAPACK [5] was used for linear system solving.

Benchmark data repository Proben1 (see [7]) containing both approximation and classification tasks was chosen. The short description of Proben1 tasks is

Task name	n	m	N_{train}	N_{test}	**Type**
cancer	9	2	525	174	class
card	51	2	518	172	class
flare	24	3	800	266	approx
glass	9	6	161	53	class
heartac	35	1	228	75	approx
hearta	35	1	690	230	approx
heartc	35	2	228	75	class
heart	35	2	690	230	class
horse	58	3	273	91	class
soybean	82	19	513	170	class

Tab. 1. Overview of Proben1 tasks. Number of inputs (n), number of outputs (m), number of samples in training and testing sets (N_{train},N_{test}). Type of task: approximation or classification.

listed in table 1. Each task is present in three variants, three different partitioning into training and testing data.

For each experiment we used separate data sets for training and testing. The normalized error was evaluated $E = 100\frac{1}{N}\sum_{i=1}^{N}||\boldsymbol{y}^i - f(\boldsymbol{x}^i)||^2$, where N is a number of examples and f is the network output.

	RN			Sum kernels			Product kernels			RBF	
Task	E_{train}	E_{test}	evals	E_{train}	E_{test}	evals	E_{train}	E_{test}	evals	E_{train}	E_{test}
cancer1	2.28	**1.75**	96	0.00	1.77	664	2.68	1.81	396	2.31	2.11
cancer2	1.86	3.01	76	0.00	**2.96**	624	2.07	3.61	412	1.91	3.12
cancer3	2.11	2.79	97	0.00	**2.73**	292	2.28	2.81	415	1.66	3.19
card1	8.75	10.01	126	8.81	10.03	472	9.22	**9.99**	6618	8.12	10.16
card2	7.55	**12.53**	101	0.00	12.54	376	7.96	12.90	6925	8.05	12.81
card3	6.52	12.35	113	6.55	12.32	387	6.94	12.23	6930	6.77	**12.09**
flare1	0.36	0.55	76	0.35	0.54	200	0.36	**0.54**	1008	0.37	**0.37**
flare2	0.42	0.28	60	0.44	**0.26**	164	0.42	0.28	756	0.41	0.31
flare3	0.38	0.35	36	0.42	**0.33**	164	0.40	0.35	1092	0.37	0.38
glass1	3.37	6.99	165	2.35	**6.15**	439	2.64	7.31	567	5.10	6.76
glass2	4.32	7.93	137	1.09	**6.97**	699	2.55	7.46	667	4.93	7.96
glass3	3.96	7.25	72	3.04	**6.29**	724	3.31	7.26	424	5.80	8.06
heart1	9.61	**13.66**	57	0.00	13.91	600	9.56	13.67	472	9.96	14.05
heart2	9.33	13.83	57	0.00	13.82	260	9.43	13.86	503	6.36	**11.67**
heart3	9.23	15.99	117	0.00	15.94	324	9.15	16.06	487	6.95	**12.02**
hearta1	3.42	4.38	134	0.00	4.37	532	3.47	4.39	1175	3.08	**4.36**
hearta2	3.54	4.07	134	3.51	4.06	478	3.28	4.29	476	3.36	**4.05**
hearta3	3.44	4.43	122	0.00	4.49	372	3.40	4.44	514	3.19	**4.29**
heartac1	4.22	**2.76**	496	0.00	3.26	483	4.22	**2.76**	1944	2.26	3.69
heartac2	3.50	3.86	342	0.00	**3.85**	500	3.49	3.87	1346	1.78	4.98
heartac3	3.36	**5.01**	405	3.36	**5.01**	1588	3.26	5.18	200	1.66	5.81
heartc1	9.99	16.07	900	0.00	**15.69**	470	10.00	16.08	3528	6.07	16.17
heartc2	12.70	**6.13**	917	0.00	6.33	680	12.37	6.29	3375	7.99	6.49
heartc3	8.79	12.68	121	0.00	**12.38**	760	8.71	12.65	496	7.13	14.35
horse1	7.35	**11.90**	121	0.20	**11.90**	408	14.25	12.45	1644	10.57	11.96
horse2	7.97	15.14	117	2.84	**15.11**	768	12.24	15.97	1332	10.04	16.80
horse3	4.26	**13.61**	85	0.18	14.13	328	9.63	15.88	479	9.88	14.56
soybean1	0.12	**0.66**	57	0.11	**0.66**	367	0.13	0.86	351	0.28	0.73
soybean2	0.24	**0.50**	85	0.25	0.53	175	0.23	0.71	463	0.38	0.60
soybean3	0.23	0.58	89	0.22	**0.57**	367	0.21	0.78	367	0.31	0.72

Tab. 2. Comparisons of errors on training and testing set for RN with Gaussian kernels, Sum kernels, Product kernels and RBF. The lowest testing error for each task is highlighted.

In table 2 errors on training and testing set for regularization network with Gaussian kernels (RN), Sum kernels and Product kernels are listed, together with number of evaluations needed to estimate the explicit parameters. Error values obtained by RBF are added to compare the performance of regularization networks with other standard neural network approach.

At Fig. 7 a kernel consisting of sum of two Gaussians found by parameter search is shown. This kernel showed an interesting behavior on several data sets. The error on the training set is almost a zero (rounded to zero) and still the generalization ability of the network is good, i.e. the error on testing set is not high. This is caused by the fact, that the kernel consists of two Gaussians, one being very narrow. The diagonal in matrix K from (4) is dominant and so regularization member is not needed, precisely γ is near to zero.

7 Conclusion

We discuss kernel based regularization network learning technique. We showed that the performance of basic RN algorithm depends on the choice of explicit parameters (kernel type, reg. parameter). We proposed two techniques for estimation of these parameters, the Adaptive grid search and the Genetic parameter search.

On experiments we demonstrated the performance of RN algorithm with parameters estimated by the Adaptive grid search.

We have shown two special types of composite kernels, Product Kernels and Sum Kernels. We compared performance of RN with classical kernels, Product kernels and Sum kernels, again using the Adaptive grid search for parameter estimation.

By parameter search for Sum kernels we found an interesting type of kernel, that exhibit a good generalization on many tasks, even without regularization member.

References

1. F. Girosi, M. Jones, and T. Poggio. Regularization theory and Neural Networks architectures. *Neural Computation*, 2:219–269, 7 1995.
2. S. Haykin. *Neural Networks: a comprehensive foundation*. Tom Robins, 2nd edition, 1999.
3. P. Kudová. Comparison of kernel based regularization networks and RBF networks. In *Proceedings of ITAT, 2004.*, 2004.
4. P. Kudová and Šámalová T. Product kernel regularization networks. In *Proceedings of ICANNGA, 2005*. Springer-Verlag, 2005.
5. LAPACK. Linear algebra package, `http://www.netlib.org/lapack/`.
6. T. Poggio and S. Smale. The mathematics of learning: Dealing with data. *Notices of the AMS*, 50:536–544, 5 2003.
7. L. Prechelt. PROBEN1 – a set of benchmarks and benchmarking rules for neural network training algorithms. Technical Report 21/94, Universitaet Karlsruhe, 9 1994.
8. B. Schoelkopf and A. J. Smola. *Learning with Kernels*. MIT Press, Cambridge, Massachusetts, 2002.
9. J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.
10. T. Šámalová and P. Kudová. Sum and product kernel networks. Technical Report 935, Institute of Computer Science, AS CR, 2005.

Generování a zpracování rozdílových dat stohově orientovaného systému*

Jiří Kulhánek and David Obdržálek

Katedra softwarového inženýrství MFF UK, Praha
j.kulhanek@centrum.cz, david.obdrzalek@mff.cuni.cz

Abstrakt Příspěvek se zabývá problematikou generování rozdílových dat z centrálního úložiště informačního systému pro import do jednotlivých provozních systémů. Některé provozní systémy přijímají data od replikačních mechanismů přímo, do některých jsou data vložena až poté, co jsou lidskou obsluhou zkontrolována a odsouhlasena či odmítnuta. To je třeba sledovat pro zajištění konzistence dat v informačním systému jako celku.
Metoda uložení dat a základní principy navržené pro implementaci celkového řešení ve formě datového stohu byly již popsány a publikovány především po stránce teoretické; v praxi jsme se museli vypořádat s konkrétními problémy danými vlastnostmi reálných připojovaných provozních systémů i s problémy danými způsobem použití replikačních mechanismů uživateli.
Popisované řešení je nyní provozováno v pilotním provozu na reálných datech a je postupně uváděno do běžného provozu.

1 Úvod

V rámci tvorby Informačního systému, který má za úkol zastřešit dosud nesouvisející aplikace, je jedním z významných úkolů propagace změn dat mezi jednotlivými aplikacemi. Tento článek vznikl v návaznosti na praktické zkušenosti a zjištění učiněná při implementaci systému ukládajícím data pomocí formmátu Datového stohu.

Teoretický návrh použitého stohového systému [1] bylo při skutečné implementaci propagace dat mezi systémy potřeba doplnit a rozšířit tak, aby nebyly omezeny schopnosti stohového systému spravovat a propagovat data a jejich změny a zároveň aby nebyly nijak dotčeny konkrétní uživatelské požadavky a vyžadované postupy pro práci se začleňovanými aplikacemi. V následujícím textu uvedeme přehled systému a jeho konkrétní implementaci: v kapitole 2 jsou stručně shrnuty základní vlastnosti stohového systému, v kapitole 3 jsou popsány způsoby propagace změn mezi aplikacemi, v kapitole 4 konkrétní implementace těchto postupů a v kapitole 5 jsou výsledky práce shrnuty.

* The work was partly supported by the project 1ET100300419 of the Program Information Society of the Thematic Program II of the National Research Program of the Czech Republic.

2 Informační systém založený na principu datového stohu

Pro uložení dat v centrálním úložišti Informačního systému byla navržena struktura Datového Stohu [1], [2], kdy je jeden záznam s N atributy v konvenčním datovém modelu v Datovém Stohu reprezentován množinou N záznamů hodnot jednotlivých atributů a sadou podpůrných metadat. Tento způsob je výhodný především pro zajištění flexibility ukládaných dat na rozdíl od tradičního způsobu uložení dat v řádkách reprezentujících množinu spolu souvisejících hodnot - atributů [3], [4]. Celý systém totiž dovoluje v čase měnit formát datových záznamů (především přidávat a odebírat atributy), což by v případě konvenčního modelu sdíleného lokálními aplikacemi bylo teoreticky obtížné a v reálném světě víceméně vyloučené. Datový Stoh také udržuje kompletní informaci o historii záznamů i v situaci, kdy se změní datový model, což by v konvenčním systému bylo také jen velmi těžce dosažitelné.

3 Propagace změn

V případě změny dat v lokální aplikaci se může jednat o změnu dat s jinými aplikacemi nesdílenými nebo sdílenými. Některá data jsou výhradně v doméně správy jediné aplikace, pak se jejich změny nikam nereplikují. Jiná data mohou být využívána i jinými aplikacemi, než tou, ve které došlo ke změně, a tehdy jsou změnová data zaslána centrálnímu úložišti, které umožní jejich propagaci do dalších lokálních aplikací. V tomto smyslu tedy propagace dat používá mechanismus "multimaster system with centralized replication" – ke změnám dochází na více místech, o replikaci se stará jedna centrální jednotka, které všechny lokální zdroje změn svá data předávají.

Základní vlastností navrženého Informačního systému je neaktivní propagace dat – centrální úložiště změny od aplikací nezasílá aktivně všem ostatním zúčastněným aplikacím, pouze je registruje a aplikaci je předá až tehdy, kdy si aplikace sama o vydání těchto dat požádá (to snižuje nároky na výkon centrální jednotky i na komunikaci mezi centrálou a aplikacemi, aniž by to snížilo užitnou hodnotu systému). Zde v tomto článku se budeme zabývat procesem synchronizace dat centrálního úložiště a aplikace od chvíle, kdy si aplikace požádá centrum o změnová data po chvíli, kdy aplikace tato data může do své databáze začlenit.

Proces synchronizace dat, v návrhu Datového Stohu označený z pohledu aplikací jako import, může být dvojího druhu – automatický a potvrzovaný:

- Automatický režim je používán typicky aplikacemi (nebo též provozními systémy), které datům přicházejícím od centrálního úložiště plně důvěřují,
- Potvrzovaný režim je použit tehdy, kdy je třeba importovaná data před zavedením do databáze aplikace ještě lidskou obsluhou potvrdit (či změnu zamítnout).

V obou případech je postup zjištění potřebných dat stejný, jsou však předávána aplikacím různým způsobem: v automatickém režimu jsou obvykle předávána

prostřednictvím tabulek v databázi, v potvrzovaném režimu jsou obvykle nejprve uživateli zobrazena pomocí HTML stránek WWW serveru a po odsouhlasení vygenerována do podoby souboru formátu XML, který je aplikací posléze zpracován. Tento rozdílný postup byl vybrán s ohledem na typické začleňované aplikace, jejich vlastnosti a požadavky: v automatickém režimu pracují převážně aplikace, které nově vznikají nebo které jejich autoři upraví pro použití v zastřešujícím Informačním systému, v potvrzovaném režimu pracují aplikace, ve kterých není možno provést tyto změny a data se do jejich databází vkládají speciálními postupy.

Oba způsoby přenosu změnových dat jsou v dalším textu probrány podrobně. V následujících sekcích jsou mechanismy použité pro generování dat popsány v těchto třídách:

1. Změněná data
2. Nová data
3. Mazaná data
4. Převod centrálního identifikátoru na lokální

Následně je uvedeno řešení dvou problémů souvisejících s propagací změn do provozních systémů - omezení generovaných dat podle domény dat konkrétního provozního systému a problém paralelního spouštění žádostí o propagaci změn.

3.1 Výběr rozdílových dat

Když si provozní systém zažádá u centrální replikační jednotky o vygenerování dat pro import, hledají se ve stohu data změněná od posledního importu dat do tohoto systému. Taková data je možné dohledat vzhledem k tomu, že u každé datové položky v centrálním úložišti evidujeme i čas její poslední změny.

Zároveň je nutno zohlednit situaci, kdy se od zahájení importu do skutečného získání dat mohou do centra propagovat nové změny z jiných provozních systémů. Je tedy nutné omezit i horní hranici vyhledávaných a generovaných změn časem zahájení dané relace. Kvůli zachování konzistence v časových závislostech je proto třeba po skončení importu zapsat jako čas jeho proběhnutí dobu, kdy byl zahájen, nikoli kdy byl dokončen.

Tato základní časová podmínka pro výběr dat je dále rozšířena o technická omezení, která říkají, že se neimportují data změněná daným provozním systémem, a ta data, pro která je nalezena ve stohu položka importující aplikace potvrzující platnou hodnotu. Tím je pokryta situace, kdy od posledního importu dva systémy poslaly do centra stejnou změnu, přičemž tato aplikace až jako druhá – pak jsou při importu sice časové závislosti pro generování změny splněny, ale přesto jde o již známou hodnotu kterou není třeba aplikaci posílat.

Mezi různými hodnotami, které posílají provozní systémy do centra, však mohou vznikat konflikty. Aby bylo možné takový konflikt metodicky vyřešit, je každému zúčastněnému systému přidělena relevance neboli "míra spolehlivosti" aplikace. Je-li do centra nová hodnota poslána aplikací, jejíž relevance je větší než hodnota relevance aktuálně platné hodnoty, je tato nahrazena nově příchozí.

Je-li její relevance menší, pak k nahrazení nedojde a nově příchozí hodnota je zaznamenána jako "tiše ignorovaná". Přítomnost ignorované položky ve Stohu pro rozhodné časové období importu je příznakem pro zahrnutí platné hodnoty mezi importní data, čímž v provozním systému dojde k sesynchronizování hodnoty s aplikací mající větší relevanci.

3.2 Propagování nových dat do aplikací

Při importu do provozního systému jsou uvedeným postupem zjištěna data, která byla změněna jinými systémy od doby minulého importu. Propagování pouze změn ale neřeší všechny požadavky kladené aplikacemi na replikační mechanismus. Pokud například dva systémy vedou záznamy o stejné osobě a v jednom systému je k této osobě přidána nová přechodná adresa jako nový záznam v tabulce kontaktů (a tento záznam je exportován do centrálního úložiště), informace by se již do druhého systému nepřenesla, neboť tento druhý provozní systém příslušný záznam nezná (dosud neexistoval) a tedy není uveden v jeho mapování.

To je řešeno pomocí tzv. insertů, tedy dalších položek, které jsou přidány ke změnovým datům, vygenerovaným na základě předchozích podmínek. Tato data jsou zároveň označena příznakem určujícím, že jde o insertovaná data. Zvláštní kapitolou jsou pak atributy takto insertované entity, kterými je odkazováno na jinou entitu, která je ovšem již importující aplikaci známá. Zde je, kvůli zpracování dat na straně provozního systému, třeba neoznačovat tuto položku jako insertovanou. Naopak, pokud je atributem entity, která je platně mapovaná, odkaz na vkládanou entitu, je nutno položku jako insertovanou označit.

Takto jsou do aplikace importovány entity, které jsou jí mapovány ve zvláštním stavu označujícím nové mapování provoznímu systému neznámé entity. Zde pak vzniká problém, že u tohoto mapování dosud neexistuje lokální identifikátor. To je řešeno jeho dočasným nahrazením centrálním identifikátorem entity. Provozní systém ví, že pokud je mu poslána položka s příznakem insert, je lokální identifikátor pouze pomocný a musí si vytvořit svůj vlastní identifikátor vyhovující jeho potřebám. Při příštím poslání dat do centra je pak poslána i tato nová hodnota s platným lokálním identifikátorem a je založeno platné mapovaní.

Nově vkládané entity pro provodní systém však nemají platné mapování, toto je třeba vytvořit. Toho je možno dosáhnout dvěma cestami Jednodušší z nich funguje na principu přimapování dat z pohledů vytvořenými nad stohovými daty. Tyto pohledy znázorňují data ve stohu v klasické podobě relačních databází a z nich jsou vybírány určité entity. Ty, které jsou pro aplikaci neznámé, jí jsou přimapovány. Provozní systém si zadává explicitní žádost o provedení.

Druhý způsob, mechanismus nazvaný "orákulum", umožňuje zohlednit vztahy entit. Orákulum je předpis, který stanoví množinu entit vztažených ke zdrojové entitě nějakého konkrétního typu (např. ve zmiňované situací orákulum určuje, že kontakty patří k osobě). Před započetím generování dat pro import se tato pravidla vykonají v přesně definovaném pořadí, přičemž jejich úkolem je pro entity daného typu vyhledat všechny entity jiného typu, které jsou v definované relaci s výchozí entitou, a které systém dosud namapované nemá. Takto nalezené

entity jsou aplikaci přiřazené k insertování. Pomocí orákula je tedy možno provoznímu systému předložit spolu s nově zakládanou entitou i všechny dodatečné informace, které provozní systém potřebuje ke správnému sestavení věty tvořící nový záznam v jeho databázi.

3.3 Oznámení o zrušení platnosti dat, smazání entity

Třetí skupinou dat, která jsou přidána do výstupu pro importující provozní systém jsou informace o smazaných entitách. V příkladu s kontaktem k dané osobě to například může znamenat zrušení adresy přechodného pobytu. Taková entita může být z databáze provozního systému odstraněna, a ve stejném smyslu je zrušena i entita typu kontakt, která tuto adresu ve Stohu reprezentuje. Jakmile jeden provozní systém takto smaže nějakou entitu a v rámci exportu svých dat do Stohu tuto změnu dá Informačnímu systému na vědomí, je pro všechny aplikace, které ji také znají, převedeno stávající platné mapování do stavu znázorňujícího, že entita má být smazána. To ovšem není povinné, jedná se pouze o doporučení a je na každém systému, zda u sebe položku skutečně zruší nebo s ní bude dál pracovat. Pokud ji zachová, pošle ji při příští příležitosti do centra, čímž se mapování opět vrátí do platného stavu. Pokud by ji ale smazal, nebude entita začleněna mezi daty poslanými centru a entita tak přestává být platně namapovaná pro danou aplikaci. V centrálních datech je takové zrušení entity zaznamenáno změnou jejího stavu na "zrušená", k fyzickému smazání však nedojde. Je tedy možné kdykoli později zjistit stav znalostí o entitě k libovolnému datu i poté, kdy dojde ke smazání této entity v provozních systémech.

3.4 Převod centrálního identifikátoru na lokální

Před posláním položek participujícímu systému jsou identifikátory jednotlivých entit převedeny z centrem používaného jednoznačného identifikátoru v rámci Stohu na jejich lokální protějšek, se kterým pracuje daný systém. Tento převod však nemusí být jednoznačný, ale může být typu 1:N, tedy jeden centrální identifikátor může být mapován na více lokálních. To může nastat tehdy, když jsou schémata některých zapojených systémů stará, ne zcela dobře navržená a nekorespondují se schématem používaným v centru.

3.5 Omezení výběru dat pro jednotlivé provozní systémy

Popsané podmínky vymezují okruh dat, která by mohla být importována dohromady do všech zapojených systémů. Jeden provozní systém ovšem pracuje jen s omezeným okruhem dat, protože zná a zpracovává pouze některé konkrétní záznamy - entity. Aby provoznímu systému nebyla posílána data, která tento provozní systém nezajímají, je v centrálním úložišti vedena tabulka, kde je pro každý systém uveden seznam všech lokálních identifikátorů entit tímto systémem zpracovávaných (tzv. mapování globálních identifikátorů entit na lokální identifikátory dané aplikace). Pro přenesení dat do importujícího systému jsou vybrána jen ta, pro které je zde nalezeno platné mapování.

Takto je výběr omezen na základě konkrétních dat, zároveň zde ale může být i omezení provedené na základě jejich obecného typu. Každá entita uložená ve Stohu má konkrétní typ a obsahuje konkrétní atributy, rovněž typované. Toto je určeno pomocí popisu vztahů entit a jejich atributů, přičemž tento popis může být různý pro každý ze systémů. Takový popis je při zapojování aplikace do Informačního systému dodán společně se seznamem typů entit a atributů, které se do připojeného systému mají importovat. Tímto způsobem může být tedy omezen výběr na celé entity, ale také jen na podmnožiny jejich atributů. Stejným způsobem jsou určeny i entity a atributy, které daná aplikace propaguje do centra (export). Oddělení těchto popisů pro import a export odpovídá běžné situaci, kdy některé informace jsou systémem do centra pouze exportovány a jiné jsou systémem pouze importovány.

3.6 Paralelismy v replikačních mechanismech

Jedním z dalších problémů, na jejichž řešení je třeba pečlivě dbát, je paralelní spouštění žádostí o propagaci změn do provozních systémů a dále souběh těchto importů se změnami zasílanými provozními systémy do centra.

Jediný zápis do databáze v centru se při importu provádí až po jeho úspěšném dokončení, jinak dochází pouze ke čtení dat, takže nic nebrání souběžnému generování dat pro různé provozní systémy. Je ale třeba zakázat spouštění nového generování dat aplikace, pro kterou již jedno generování právě probíhá. To by pouze zbytečně zatěžovalo centrum a především by to vedlo k nesrovnalostem v návaznosti importů aplikace.

Druhým místem možné kolize je souběžný import a export a to pro kombinaci jakýchkoliv systémů, protože vždy hrozí riziko neprázdného průniku zpracovávaných dat. Export, tj. zasílání dat do centra tedy musí probíhat vždy v oddělené transakci. Ke vzniku mezer v časové souslednosti tedy nedochází, neboť čas posledního importu je definován jako čas jeho zahájení a ten je zapisován až po potvrzení provozním systémem, že se skutečně podařilo data přijmout.

4 Implementace popsaných mechanismů

Generování a předání dat aplikaci, která si zažádala o import, může probíhat buď přímo, kdy systém přijímá automaticky všechna data, které mu centrum poskytne nebo s mezikrokem, ve kterém jsou data nejprve zobrazena obsluze a ta je musí potvrdit. Dále lze ještě rozlišit dvě kategorie podle způsobu přenosu dat – ten může probíhat buď přímo kopírováním dat mezi tabulkami databází nebo "offline" cestou pomocí XML souborů. Při ní jsou potřebná data uložena do souboru a předána na úložiště, odkud si je importující systém vyzvedne a zpracuje. Z praktických důvodů konkrétního nasazení Informačního systému probíhá přenos s využitím XML souborů jen ve spojení s potvrzováním v mezikroku.

Tyto metody a jejich kombinace dodržují obecná pravidla pro generování a zpracování dat tak, jak byla popsána v předchozím textu. Při implementaci jsme se však rozhodli konkrétní implementace oddělit a jednotlivé části

přizpůsobit specifickým požadavkům daného mechanismu. Nutnost používat několik různých způsobů propagace dat je zapříčiněna velkou rozdílností provozních systému, které jsou k centru připojeny a to jak rozdíly v jejich požadavcích připojení, tak rozdíly ve schopnostech používat některé konkrétní technologie. První důvod souvisí s tím, že některé aplikace jsou natolik svébytné, že není možné, aby se nová data do nich propagovala přímo. Proto byl zaveden mezikrok, kdy jsou data k importu do aplikace zobrazena nejdříve člověku, který ji obsluhuje. Naopak u jiných systémům toto potvrzení není ani žádoucí (příkladem takové aplikace může být knihovní aplikace, která získává informace o osobách oprávněných k výpůjčce z aplikace studijního oddělení, avšak její obsluha není oprávněna taková data nepřijmout). Druhý důvod pak je způsoben zastaralostí některých zapojených systémů, kde vzhledem ke způsobu uložení dat aplikace není možno přistoupit ke kopírování na úrovní databází přičemž není možno aplikaci upravit tak, aby zmíněný způsob přenosu mohla také použít. Nad danou aplikací je proto postavena mezivrstva zajišťující komunikaci s centrem, která příjme vygenerovaný XML soubor, zpracuje ho a získaná data poskytne aplikaci dle jejich konkrétních specifických požadavků.

4.1 Průběh importu pro přenos bez potvrzení pomocí SQL

Tento způsob, tzv. "SQL import" je z hlediska systému nejjednodušším pro vytvoření a přenos dat. Na straně centra se pomocí soustavy dotazů nad databázemi získají data k importu a zapíší se do tabulky určené k této výměně. Odtud jsou vyzvednuty modulem provozního systému, který je propaguje do pracovního prostoru importující aplikace.

Tabulka pro předávání dat pomocí SQL má formát, kterým je jednak zachycena struktura dat (XSD popisem), zároveň však využívá stohovou formu. Každý záznam této tabulky tedy zachycuje jednu hodnotu atributu zaznamenanou v příslušném datovém sloupci, a pro účely identifikace současně obsahuje odkaz na odpovídající element v XSD a na identitu, ke které tento záznam patří Dále je zde ještě nesena informace o tom, jestli jde o změnu, novou hodnotu nebo položku ke smazání.

Při tomto druhu importu není třeba žádných dalších rozšiřujících mechanismů a vše probíhá přímočaře. Z výkonového hlediska bylo ale příslušnou akci třeba dobře naplánovat a implementovat, neboť zvláště při zapojení nového systému dochází k přenosu značného množství dat (řádově i statisíce položek), a při vyvolávání importu není možné předem budoucí objem dat nijak odhadnout.

4.2 Zobrazení dat pro jejich potvrzení před samotným importem

Jak bylo řečeno, některé systémy se neobejdou bez potvrzení každé jednotlivé položky před jejím importováním. Po zvážení dopadů na komfort obsluhy i konzistenci dat v Informačním systému jako celku bylo rozhodnuto, že vygenerovaná data jsou implicitně určena k převzetí lokálním systémem a jejich případné odmítnutí musí obsluha provést explicitně. Data ke zobrazení jsou oproti "SQL

importu“ generována do tabulky s jiným schématem uložení dat, které více vyhovuje potvrzovací aplikaci. Zde není třeba klást důraz na strukturu dat, jak je definována pomocí XSD, místo toho se entity na sebe váží jen přes své lokální identifikátory. Uživateli je pak zobrazen seznam entit, u kterých dochází ke změně (ať už ke změně hodnoty atributu, jeho přidání nebo smazání), a pro každou její atributy. Uživatel označí ty změny, které nechce přijmout. Kromě změnových a insertovaných položek je zobrazen i seznam entit určených ke smazání, ale vzhledem k tomu, že logika toho, zda bude entita skutečně odmítnuta, je závislá až na samotném systému, nelze toto smazání odmítat.

Zapojení lidského elementu do procesu propagace dat do systému s sebou nese jednu komplikaci. Aby obsluha mohla smysluplně rozhodnout o přijetí dat, potřebuje k nim obvykle ještě doplňkové informace. Například při změně rodinného stavu osoby ze ”svobodná“ na ”vdaná“ by tato změna samotná nenesla pro uživatele dostatek informace. Uživateli je třeba poskytnout například celé jméno osoby a případně i další údaje, které mu data více přiblíží. Která data jsou takto popisná, je definováno ve speciálních meta-tabulkách, kde je uložen seznam tzv. popisných atributů entity, jež se mají zobrazit vždy, když je importním algoritmem generována libovolná položka této entity. V řadě případů by ani takovéto vypisování údajů jen z dané entity nebylo dostatečné. Pokud se změní například adresa osoby, je potřeba identifikovat tuto osobu a nestačí uvést všechny atributy z entity ”adresa“. Potvrzovací aplikace však nemá (obecně nemůže mít) přístup k databázi provozního systému, veškerá zobrazovaná data musí poskytnout centrální relokační mechanismus. Proto lze pro každý typ entity definovat typy entit, které jsou jí nadřazené. Pak se do výsledné množiny přidají i všechny popisné atributy z takto nadřazené entity a toto se provádí rekurzivně. Uživatel má pak možnost se při prohlížení importních dat postupně zanořovat a pro entitu si zobrazovat celý strom nadřazených entit.

Protože popisné atributy slouží uživateli pro správnou detekci entity, je při jejich změně třeba kromě nového (změněného) stavu třeba uživateli předložit i stav původní (typickým případem je změna příjmení osoby; bez znalosti původního příjmení není obvykle možno osobu přesně identifikovat). Z hlediska časové platnosti atributů je tedy třeba dohledat i hodnota platnou v čase posledního importu do daného systému, což je díky časovým známkám snadno splnitelné. Potvrzovací aplikace pak zobrazí uživateli oba údaje.

4.3 Odmítnutí importu dat do provozního systému

Pokud obsluha označí v potvrzovací aplikaci některá data k odmítnutí, jsou tato data nejenom vynechána při následném potvrzení importu, ale informace o odmítnutí je propagována zpět do centra.

Je-li odmítnut atribut nesoucí změnu již existující hodnoty, je do stohu vložena kopie platné hodnoty s identifikátorem odmítající aplikace a se speciálním příznakem ”explicitně odmítnutá“. Takovéto odmítnuté hodnoty je pak možno zobrazit pomocí administrátorské aplikace, díky které obsluha může dohlížet na případné problémy a řešit konflikty v datech dvou aplikací. Je třeba operativně reagovat na nesrovnalosti, komunikovat přímo s obsluhou provozních

systému, rozhodovat, která verze dat je správná a tuto změnu pomocí administrátorské aplikace direktivně v centrálních datech uplatnit.. Tímto způsobem se předchází šíření překlepů a dalších chyb a v případě jejich výskytu se usnadňuje jejich identifikace a rychlá oprava.

Pokud je odmítnuta aplikaci neznámá entita určená k insertování, je pro ni zrušeno mapování, na jehož základě se nová entita do systému importuje. Zároveň je ale nutno mezi mapování zavést stav specifikující, že tato entita nemá být importována ani v budoucnu. Výskyt tohoto mapování je pak indikátorem pro mechanismy vyhledávající aplikaci nové entity, že tato entita má být vynechána.

4.4 Přenos dat potvrzených k importu pomocí SQL

Pokud mají být potvrzená importní data přenesena do systému přímo kopírováním mezi databázemi, tak jak bylo popsáno, postup pro vlastní provedení je jednoduchý. Změnová data se z tabulky, odkud je čerpala potvrzovací aplikace, překopírují do tabulky určené k SQL přenosu. Doplňková zobrazovací data a položky, které byly odmítnuty, se pochopitelně nekopírují.

4.5 Doplnění potvrzených dat o další pomocná data

Pokud bude přenos potvrzených dat probíhat pomocí XML souborů, je situace složitější. Vrstva, která na straně provozního systému komunikuje s centrem, může ke změnovým datům potřebovat doplňující informace. Pro provozní systémy, které jsou takto obsluhovány, totiž nemusí být možné dohledat všechna potřebná data nutná ke složení databázové řádky, takže je nutno taková data při importu přímo poskytnout.. Data, reprezentovaná ve stohu několika entitami, musí být "slepena" dohromady.

Toto doplnění údajů se provádí na základě přítomnosti speciálního příznaku u atributu přidáním dat mezi data, ze kterých se provádělo zobrazení k potvrzení. Odtud jsou pak všechna data vybírána pro uložení do XML.

Algoritmus generování dat se opírá o meta-tabulku se seznamem entit a jejich atributů, které jsou importovány. V této tabulce může být u atributu identifikátor určující, nakolik je atribut citlivý na změnu, tzv. "senzitivita". Rozlišují se dva druhy sensitivity. Nižší znamená, že ke změnovému atributu jsou vygenerovány i všechny další atributy dané entity. Vyšší sensitivita značí potřebu znát i data k entitě s tímto atributem vztažená. Například pro změnu údaje ve smlouvě může být potřeba znát i některé informace o osobě. Postupuje se tedy tak, že jsou rekurzivně generována všechna data entit, na které je z dané entity odkazováno nebo které naopak na entitu odkazují. Protože to by samo o sobě mohlo v obecném případě vést v podstatě ke generování všech dat známých aplikaci (a to pro každý takový atribut), některé typy entit jsou tzv. zarážkami, na nichž se rekurze zastaví. Zarážky byly určeny na základě analýzy konkrétních dat a vztahů mezi nimi v průběhu testovacího provozu.

4.6 Přenos dat pomocí XML souboru

Jakmile je proveden i tento poslední krok a všechna data pro přenos jsou připravena, je možno přistoupit k vytvoření XML souboru pro přenos. Obecný formát tohoto souboru je jednotný pro všechny systémy: v kořenu jsou elementy s identifikátorem importujícího systému, čas vygenerování, číslo udávající, podle které verze XSD byla data do XML uložena, a elementy s vlastními daty rozdělené podle toho, zda jde u změnová data, inserty nebo rušené entity. Potomky těchto elementů jsou pak elementy pojmenované podle typu entity a jejich potomky jsou elementy pojmenované podle atributů. Tato jména jsou právě určena ve zmiňovaném XSD. Obsahem atributových elementů jsou pak samotná importovaná data.

Jako XML parser byl použit Xerces-C++. Průchodem platných dat generovaných pro zobrazení, se v pořadí definovaném přijímajícím systémem ukládají do DOM stromu reprezentovanému v paměti. Ten je po svém vytvoření uložen do souboru se jménem identifikujícím aplikaci, pro kterou byl vytvořen a s čítačem importu do daného systému. Pomocí něj je zajištěna kontrola na straně importující aplikace, že nedošlo ke ztrátě některého ze souborů v řadě importů. Následně je soubor zašifrován a spolu s veřejným klíčem odeslán do datového úložiště, odkud je vyzvednut modulem provozního systému. Ten v pravidelném intervalu hledá nový importní soubor v úložišti.

5 Shrnutí

Při vývoji Informačního Systému s použitím stohového formátu dat bylo nutné pro přenos dat od centra k provozním systémům navrhnout a implementovat konkrétní postupy a metodiky, aby bylo možno zajistit požadovanou funkčnost systému jako celku. Vzhledem k velké šíři postupně zapojovaných aplikací docházelo k mnohým úpravám původně navrhovaných mechanismů vedoucích především ke zlepšení uživatelského komfortu. Systém je nyní provozován v pilotním provozu s reálnými daty a je možno konstatovat, že současný stav umožňuje propagaci všech typů požadovaných dat. I v dalším vývoji systému očekáváme změny na základě vlastností nově připojovaných systémů.

Reference

1. D. Bednarek, D. Obdrzalek, J. Yaghob, F. Zavoral, Synchronisation of Large Heterogenous Data Using DataPile Structure, In: P. Vojtáš(ed.), ITAT 2003, Information Technologies-Applictions and Theory, Proceedings, Slovakia, 2004, 113–121
2. D. Bednarek, D. Obdrzalek, J. Yaghob, F. Zavoral, Data Integration Using DataPile Structure. In proc. of the 9th East-European Conference on Advances in Database and Information Systems, Tallinn, Estonia, 2005
3. M. Lenzerini, Data Integration: A Theoretical Perspective. In Proc. of the 21st ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2002), 2002, 233–246
4. E. Schallehn, K. Sattler, and G. Saake, Extensible and Similarity-Based Grouping for Data Integration. In 8th Int. Conf. on Data Engineering (ICDE), San Jose, CA, 2002

Sparse binary matrices*

Jan Martinovič, Jiří Dvorský, and Václav Snášel

Department of Computer Science, VŠB – Technical University of Ostrava
17. listopadu 15, 708 33 Ostrava, Czech Republic
{jan.martinovic,jiri.dvorsky,vaclav.snasel}@vsb.cz

Abstract. Many of these computations have matrix character. Although amount of memory in computers grows very rapidly, there are still matrices that are bigger than available memory. But many of these matrices are sparse, so that storage only non-zero values can solve the problem. Special case of sparse matrices are binary sparse matrices i.e. matrices where only binary values 0 and 1 are allowed. Large sparse binary matrices play important role in computer science, and in many modern information retrieval methods. These methods, such as clustering, web graph computations, web link analysis [6], binary factor analysis [7] performs huge number of computations with such matrices, thus their implementation should be very carefully designed.

1 Introduction

Numerical computations represent serious problem for generations of mathematicians. There were not suitable device to make the computations, only human being. Development of computers gives to people power to perform computations, which were impossible in past. Many of these computations have matrix character. Although amount of memory in computers grows very rapidly, there are still matrices that are bigger than available memory. But many of these matrices are sparse, so that storage only non-zero values can solve the problem. Special case of sparse matrices are binary sparse matrices i.e. matrices where only binary values 0 and 1 are allowed. Large sparse binary matrices play important role in computer science (indexing of class hierarchy [3]), and in many modern information retrieval methods. These methods, such as clustering, web graph computations, web link analysis [6], binary factor analysis e.g. [7] performs huge number of computations with such matrices, thus their implementation should be very carefully designed (e.g. sparse matrix multiplication [11]).

This paper is organized as follows. Section 2 describe state-of-art in sparse matrix implementation. A previously published approach for sparse matrix storage [10] based on finite automata is given in Section 2.1. The multi-dimensional approach for sparse matrix storage is introduced in Section 2.2. Sparse binary matrices storage schema based on finite automata that allows incremental building of the matrix is described in Section 3. In Section 4 preliminary experimental results are shown. Finally, we conclude with a summary of contributions and discussion on future work.

* This work was supported by GAČR No. 201/05/P145.

2 Short survey of sparse matrix storage

Let M be a sparse matrix of order $n \times m$. The matrix M can be be efficiently processed, if the zero elements of M are not stored. There are many methods for storing the data (see for instance [1]). Here we will discuss several sparse matrix storage methods.

Compressed Row Storage (CRS) – The Compressed Row Storage (CRS) format puts the subsequent nonzeros of the matrix rows in contiguous memory locations. Assuming we have a nonsymmetric sparse matrix M, we create 3 vectors: one for floatingpoint numbers (val), and the other two for integers (col_{ind}, row_{ptr}). The val vector stores the values of the nonzero elements of the matrix M, as they are traversed in a rowwise fashion. The col_{ind} vector stores the column indexes of the elements in the val vector. That is, if $val(k) = a_{i,j}$ then $col_{ind(k)} = j$. The row_{ptr} vector stores the locations in the val vector that start a row, that is, if $val(k) = a_{i,j}$ then $row_{ptr(i)} \leq k < row_{ptr(i+1)}$. By convention, we define $row_{ptr(n+1)} = n_{nz} + 1$, where n_{nz} is the number of nonzeros in the matrix M. The storage savings for this approach is significant. Instead of storing n^2 elements, we need only $2n_{nz} + n + 1$ storage locations.

Compressed Column Storage (CCS) – Analogous to Compressed Row Storage there is Compressed Column Storage (CCS), which is also called the *Harwell-Boeing sparse matrix format* [4]. The CCS format is identical to the CRS format except that the columns of M are stored (traversed) instead of the rows. In other words, the CCS format is the CRS format for M^T.

The CCS format is specified by the 3 arrays val, row_{ind}, col_{ptr}, where row_{ind} stores the row indices of each nonzero, and col_{ptr} stores the index of the elements in val which start a column of M.

Example 1. Let M be a matrix of order 8×8.

$$M = \begin{pmatrix} 2&0&0&0&0&0&0&0 \\ 0&4&0&0&1&0&0&0 \\ 0&0&3&0&0&6&0&9 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&8&0&0&0&5&0&0 \\ 0&0&0&0&0&0&9&0 \\ 0&0&0&0&0&0&0&7 \end{pmatrix} \tag{1}$$

The CRS format for this matrix is then specified by the arrays val, col_{ind}, row_{ptr} given in Table 1. The CCS format for the matrix M is given in Table 2.

2.1 Sparse matrices and finite automata

Culik and Valenta [2] introduced finite automata for compression of bi-level and simple color images. A digitized image of the finite resolution $m \times n$ consists

val	2	4	1	3	6	9	1	1	8	5	9	7
col_{ind}	1	2	5	3	6	8	4	5	2	6	7	8

row_{ptr}	1	2	4	7	8	9	11	12	13

Tab. 1. The CRS format for the matrix M in equation (1).

val	2	4	8	3	1	1	1	6	5	9	9	7
row_{ind}	1	2	6	3	4	2	5	3	6	7	3	8

col_{ptr}	1	2	4	5	6	8	10	11	13

Tab. 2. The CCS format for the matrix M in equation (1).

of $m \times n$ pixels each of which takes a Boolean value (1 for black, 0 for white) for bilevel image, or a real value (practically digitized to an integer between 0 and 256) for a grayscale image.

Sparse matrix can be viewed, in some manner, as simple color image too. Zero element of matrix corresponds to white pixel in bi-level image and nonzero element to black or gray-scale pixel.

Here we will consider square matrix M of order $2^n \times 2^n$ (typically $13 \leq n \leq 24$). In order to facilitate the application of finite automata to matrix description we will assign each element at $2^n \times 2^n$ resolution a word of length n over the alphabet $\Sigma = \{0, 1, 2, 3\}$ as its address. A element of the matrix corresponds to a subsquare of size 2^{-n} of the unit square. We choose ε as the address of the whole square matrix.

Its submatrices (quadrants) are addressed by single digits as shown in Figure 1(a). The four submatries of the matrix with address ω are addressed $\omega 0$, $\omega 1$, $\omega 2$ and $\omega 3$, recursively. Addresses of all the submatrices of dimension 4×4 are shown in Figure 1(b). The submatrix (element) with address 121 is shown on the right of Figure 1(c).

In order to specify a values of matrix of dimension $2^n \times 2^n$, we need to specify a function $\Sigma^n \to R$, or alternately we can specify just the set of non-zero values, i.e. a language $L \subseteq \Sigma^n$ and function $f_M : L \to R$.

This kind of storage system allows direct access to stored matrix. Each of elements can be accessed independently to previous accesses and access to each element has same, constant time complexity. Let M be a matrix of order $2^n \times 2^n$. Then time complexity of access is bounded by $O(\log_2 n)$. For detail information see [10].

Example 2. Let suppose matrix M given in example 1. The language $L \subseteq \Sigma^3$ is now

$$L = \{000, 003, 030, 033, 102, 121, 131, 203, 300, 303, 330, 333\}.$$

Then function f_M will have following values (see Table 3).

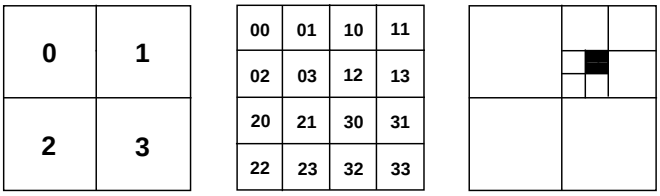


Fig. 1. The addresses of the submatrices (quadrants), of the subsmatrices of dimension 4×4, and the submatrix specified by the string 121.

$x \in L$	$f_M(x)$	$x \in L$	$f_M(x)$
000	2	203	8
003	4	300	1
030	3	303	5
033	1	330	9
102	1	333	7
121	6		
131	9		

Tab. 3. Positions in matrix M and corresponding values – function f_M.

2.2 Multi-dimensional sparse matrix storage

In order to a general multi-dimensional data structure can be used for the sparse matrix storage, the following definitions must be introduced.

Definition 1 (A matrix as tuples of 2-dimensional space). *Let M be a matrix of order $n \times m$ and $\Omega_{MT} = D_N \times D_M$ be an 2-dimensional discrete space (called matrix space), where $D_N = \{0, 1, \ldots, 2^{l_N} - 1\}$, $D_M = \{0, 1, \ldots, 2^{l_M} - 1\}$. It holds $n \leq 2^{l_N} - 1$, $m \leq 2^{l_M} - 1$. For all $a_{i,j} \in M$ there is mapping $\alpha : M \rightarrow \Omega_{MT}$ such that $\alpha(a_{i,j}) = (i, j)$.*

The mapping α transforms elements of matrix M to 2-dimensional space Ω_{MT}. The matrix space can be seen in Figure 2. The matrix space seems to be 3-dimensional. However, indices of matrix elements have to be indexed. The value of element is stored as non-index data. Consequently, only two coordinates must be indexed, so that the matrix space is only 2-dimensional.

3 Binary matrices and finite automata

Let M be a binary matrix of dimension $n_1 \times n_2$, where $n_1, n_2 \in \mathbb{N}$. Let $n = \max(n_1, n_2)$ and $size = 2^{\lceil \log_2 n \rceil}$. Finite automaton which is represented by matrix M is defined as:

$$A = (Q; \Sigma; \delta; q_0; F),$$

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,m} \\ a_{2,1} & a_{2,2} & \dots & a_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \dots & a_{n,m} \end{pmatrix} \longrightarrow \begin{array}{c|cccc} & 1 & 2 & \dots & m \\ \hline 1 & t_{1,1} & t_{1,2} & \dots & t_{1,m} \\ 2 & t_{2,1} & t_{2,2} & \dots & t_{2,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ n & t_{n,1} & t_{n,2} & \dots & t_{n,m} \end{array}$$

Fig. 2. A matrix as tuples of 2-dimensional space.

where:

- Q is the finite set of states;
- $\Sigma = \{0, 1, 2, 3\}$ is the input alphabet;
- $\delta : Q \times \Sigma \to Q$ is the transition function;
- q_0 is the initial state;
- $F = \{q_f\}$ is the set of (one) finite state.

The language L which is accepted by automaton A is defined:

$$L = \{w \in \Sigma^*; \exists x, y \in \mathbb{N}, 1 \leq x, y \leq n, M_{x,y} = 1, w = l(x, y, size)\}$$

where $l : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \Sigma^*$:

$$l(x, y, val) = \begin{cases} \varepsilon & \text{for } val = 0 \\ 3l(x - val, y - val, val/2) & \text{for } x \geq val \wedge y \geq val \\ 2l(x - val, y, val/2) & \text{for } x \geq val \wedge y < val \\ 1l(x, y - val, val/2) & \text{for } x < val \wedge y \geq val \\ 0l(x, y, val/2) & \text{for } x < val \wedge y < val \end{cases}$$

As can be seen function l produces words according to schema in Figure 1.

3.1 Reconstruction of the matrix

For each element of the matrix $M_{x,y}$ holds

$$M_{x,y} = \begin{cases} 1 \text{ automaton } A \text{ accepts } l(x, y, size) \\ 0 \text{ otherwise} \end{cases}$$

3.2 Construction of the automaton

Finite automaton for given binary matrix can be costructed in several ways e.g. using derivation of regular expression [5] or as composition of elementary automatons [9]. All these methods of construction have common disadvantage - the whole matrix i.e. complete language must be a priori known. Better choice is incremental construction of the automaton.

The incremental construction method starts with empty automaton (with exception of global initial and final state) and step by step builds sequence of

automata. As soon as new element is inserted to the matrix corresponding word is added to the automaton and the automaton is transformed to new one.

Transitions included in the automaton are represented by injective function $\delta \subseteq Q \times \Sigma \times Q$. Futher function $p : Q \times \Sigma^* \times \delta \to Q$ is defined, such that:

$$p(q_{in}, aw, \rho) = \begin{cases} p(p(q_{in}, a, \rho), w, \rho) & \text{if } w \neq \epsilon \ \wedge \ p(q_{in}, a, \rho) \in \rho \\ q & \text{if } w = \epsilon \ \wedge \ \exists q \in Q : \ \langle q_{in}, a, q \rangle \in \rho \\ \text{undefined} & \text{otherwise} \end{cases}$$

In order to the automaton A can accept new words it is necessary to add transitions to relation $\rho \subseteq \delta$ automatu A. For transformation of one automaton to another automaton holds:

1. $A_0 = (\{q_0, q_f\}; \Sigma; \rho_0; q_0; \{q_f\})$ and $\rho_0 = \emptyset$ and $L_0 = \emptyset$.
2. Let $A_i = (\{q_0, \ldots, q_k, q_f\}; \Sigma; \rho_i; q_0; \{q_f\})$ is given automaton.

   Then automaton accepting language $L_{A_{i+1}} = L_{A_i} \cup \{w \in L\}$ is defined as:

$$A_{i+1} = (\{q_0, \ldots, q_k, \ldots, q_{k+j}, q_f\}; \Sigma; \rho_{i+1}; q_0; \{q_f\}),$$

   where j is count of states $\notin Q_{A_i}$.

   For transitions it holds $\rho_{i+1} = add(\rho_i, \epsilon, w)$. Function $add : \delta \times \Sigma^* \times \Sigma^* \to \delta$ is defined:

$add(\rho, \alpha, a\beta)$:

```
q = p(p(q_0, α, ρ), a, ρ);
if β = ε ∧ q = undefined
   return ρ ∪ {⟨q, a, q_f⟩};
if β ≠ ε ∧ q = undefined
   return add(ρ ∪ {⟨q, a, q_ξ(ρ)⟩}, αa, β);
return add(ρ, αa, β);
```

   where $\alpha a \beta \in L$ and $\xi(\rho) : \delta \to \mathbb{N}$:

$$\xi(\rho) = i + 1; \ j_a, j_b \in \mathbb{N}, \ \forall \langle q_{j_a}, a, q_{j_b} \rangle \in \rho : \ i \geq \max(j_a, j_b)$$

High space complexity is main disadvantage of this algorithm. Number of states of the automaton is increased after inserting of every new word $w \in L$. Therefore after given number of instertions minimization of the automata is done. So as we will be able to insert other new words into automaton, function *add* should be changed. Modified definition of function *add* has following form:

$add(\rho, \alpha, a\beta)$:

$q = p(p(q_0, \alpha, \rho), a, \rho)$;
if $\beta = \epsilon \ \wedge \ q =$ undefined
  return $\rho \ \cup \ \{\langle q, a, q_f\rangle\}$;
if $\beta \neq \epsilon \ \wedge \ q =$ undefined
  return $add(\rho \ \cup \ \left\{\left\langle q, a, q_{\xi(\rho)}\right\rangle\right\}, \alpha a, \beta)$;
if $q \neq$ undefined $\ \wedge \ P(q) > 1$ then
begin
  $\rho_x = \left\{\left\langle q_{\xi(\rho)}, b, q_x\right\rangle; \forall b \in \Sigma, q_x = p(q_0, \alpha a b, \rho), \langle q, b, q_x\rangle \in \rho\right\}$;
  $\rho_x = \rho_x \cup \ \left\{\left\langle q, a, q_{\xi(\rho)}\right\rangle\right\}$;
  return $add((\rho - \{\langle q, a, p(q_0, \alpha a, \rho)\rangle\}) \ \cup \ \rho_x \ , \alpha a, \beta)$;
end
return $add(\rho, \alpha a, \beta)$;

where $\alpha a \beta \in L$ and functiom $P(q) : Q \to \mathbb{N}$ is defined:

$$P(q) = |\{\langle q_a, a\rangle ; q_a \in Q, a \in \Sigma : \ \delta(q_a, a) = q\}|$$

3.3 Minimization of the automaton

Basic incremental construction method does not consider any minimization of the automaton. Minimizing of the automaton is done by function min(*onLevel*), which is called with argument $onlevel = 1$. The function $\min : \mathbb{N} \times \delta \to \delta$ has following form:

$\min(onLevel, \rho)$:

change ρ by this definition
{
  $\forall \langle q_{1a}, a_1, q_{1b}\rangle, \langle q_{2a}, a_2, q_{2b}\rangle \in list(onLevel, \rho)$:
  $path(q_{1b}, \rho) = path(q_{2b}, \rho) \ \Rightarrow \ q_{1b} = q_{2b}$;
}
if $onLevel < size$ then return $\min(onLevel * 2, \rho)$;
else return ρ;

where relation $list : \mathbb{N} \times \delta \to \delta$:

$$list(level, \rho) = \{\langle q_{in}, a, p(q_{in}, a, \rho)\rangle ; \forall q_{in} \in Q \ \forall a \in \Sigma, yz \in L : |z| = \log_2(level) \wedge \ p(q_0, ya, \rho) = = p(q_{in}, a, \rho)\}$$

and function $path : Q \times \delta \to 2^Q$:

$$path(q, \rho) = \{q_{out}; \forall a \in \Sigma : \ \langle q, a, q_{out}\rangle \in \rho\}$$

The minimization algorithm supposes that every accepted word has equal length $|w| = \log_2(size)$.

Theorem 1. *Function* $\min(onLevel)$ *for given automaton A reduces its number of states to minimal possible value.*

Sketch of proof. The function min reduces automaton A from its final state towards initial state. The automaton can be viewed as graph, where states represent vertices and transitions edges. The graph is direct acyclic graph. Set of states (vertices) can be divided into subset (layers) according to their distance from final state – the distance is measured as number of transitions from the given state to final state (see [5]). The function min combines states q_i and q_j in each layer if a sequence of transitions from state q_i and q_j is the same. And consequently number of diffrent paths from each layer is minimal, there are no duplicate paths. For contradiction suppose that there are two different paths P_1 and P_2 from state q_i to the final state q_f. Each state in the finite automaton represents particular derivation of regular expression, which corresponds to the automaton. Two path from state q_i means that there are two possible sequences of consecutive derivations from the state q_i to the final state. This is contradiciton, because derivation of regular expression is unique.

Example 3. Let M be a matrix of size 4×6

$$M = \begin{pmatrix} 1\,0\,0\,0\,1\,0 \\ 0\,0\,0\,1\,0\,0 \\ 1\,0\,0\,0\,0\,0 \\ 1\,0\,0\,0\,0\,1 \end{pmatrix}$$

The matrix can be described by language $L_M = \{000, 013, 020, 100, 123, 022\}$. Initial automaton A_0 does not accept any language i.e. $L_0 = \emptyset$. Automaton A_1 is constructed from A_0 and accepts language $L_1 = \{000\}$. Automaton A_2 is extension of automaton A_1 and accepts language $L_2 = L_1 \cup \{013\} = \{000, 013\}$. In this way automaton A_6 accepts whole language L_M (see Fig. 3). This is expanded, non-minimized form of the automaton.

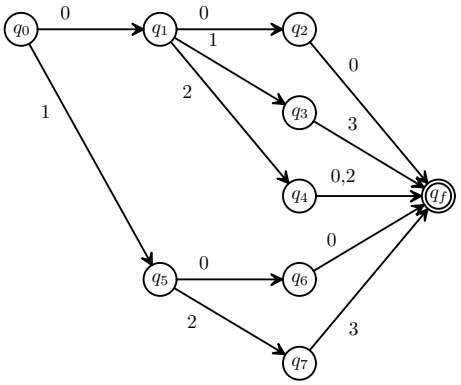


Fig. 3. Non-minimized automaton A_6.

The automaton A_6 will be minimized now. Minimalizing proccess starts in the final state q_f and goes to initila state of automaton. The minimizing proccess takes two steps in this case:

1. There are two transitions between q_2 and q_f and q_6 and q_f respectively. In both cases transition symbol is 0. These two transitions can be joined together and state q_6 is deleted (see Fig. 4).

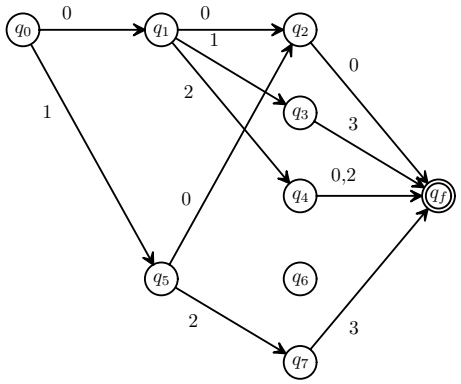


Fig. 4. Automaton A_6, state q_6 will be deleted.

2. There are also two transitions among q_3 and q_7 as initial states and q_f as final state. Transitions are labeled by symbol 3. As well as in the first step transitions are unified and state q_7 is deleted (see Fig. 5).

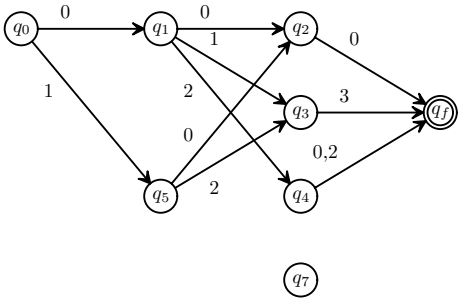


Fig. 5. Minimized automaton A_6, state q_7 will be deleted.

To demonstrate inserting of new element to the automaton, consider matrix M'.

$$M' = M + \begin{pmatrix} 0\,0\,0\,0\,0\,0 \\ 0\,0\,1\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0 \end{pmatrix} = \begin{pmatrix} 1\,0\,0\,0\,1\,0 \\ 0\,0\,\mathbf{1}\,1\,0\,0 \\ 1\,0\,0\,0\,0\,0 \\ 1\,0\,0\,0\,0\,1 \end{pmatrix}$$

Matrix M' differs from matrix M in element $M'_{2,3} = 1$. The element has corresponding word $w = 012$ which will be inserted into minimized automaton A_6. Insertion process starts in initial state q_0 and goes through automaton to final state q_f. In this case insertion requires three steps (number of steps are proportional to the length of word w):

1. Transition for symbol 0 exists, resulting state is q_1. No new state has to be added in this case.
2. Transition form q_1 for symbol 1 exists again, resulting state is q_3. New transition between q_1 and q_3 can not be added, because state q_3 is final state for more than one state - transition from state q_5 leads to this state too. The transition from q_1 to q_3 with symbol 1 induces, that word $w' = 122$ will be also inserted into automaton. To solve the problem, state q_3 have to be splited into two states - existing state q_3 and new state q_6. Transition from state q_1 leads to new state q_6 (see Fig. 6).
3. The last step is simple. There is no problem to add new transition from state q_6 to final state q_f for symbol 2 (see Fig. 7). Newly constructed automaton A_7 is also minimal.

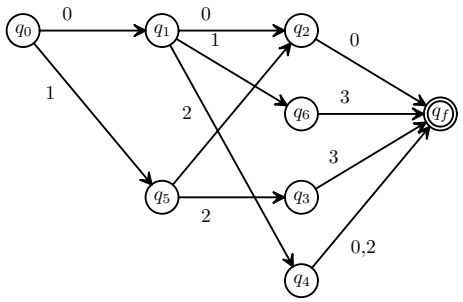


Fig. 6. State q_3 is splited, new state q_6 is inserted.

4 Experimental results

The algorithms described above were implemented to take care about several experiment. Experiments were performed on some small artificial data, but the most important experiment was done on real data taken from Enron Corpus [8].

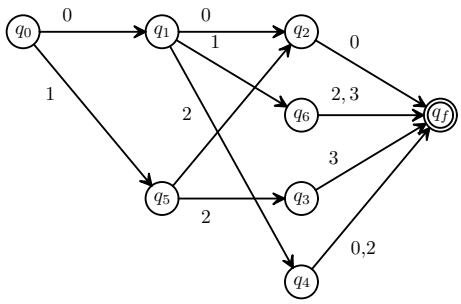


Fig. 7. Automaton A_7 for matrix M'.

Sparse matrix, in this case, represents document (email) × term matrix. Parameters of collection are:

- 300458 documents
- 286288 words
- 23013806 non-zero point in matrix document x terms.

Minimizing was done after each 10^4 documents (sparse vectors) were inserted. Table 4 show the number of states and other data taken from experiment ($size$ is dimension of the matrix (see section 3), c_1 is the number of states before minimizing, c_2 is the number of states after minimizing).

4.1 Structure of the automaton

The first, crucial parameter of constructed automaton is number of states and the way how rapidly the number of states is change with respect to the number of inserted matrix's elements. The second important parameter that affects efficient storage is the number of transitions from particular state i.e. it is important if the state has one, two, three or four transitions to other states. The results of experiment is given in table 5. The first column represents number of non-zero elements in matrix and the next columns represent number of states with 1, 2, 3 or 4 transitions. The last column gives the total number in automaton. All results are taken after minimizing process.

It is easy to see that the number of states rapidly grows and amount of used memory is growing in the same way. In order to find out if number of states depends on density of matrix, tests on artificially generated matrix were performed.

4.2 Artificially generated matrix

Test on artificial show how the number of states grows with respect to growing density of input binary matrix. Input matrices have size 1024×1024. Tests have following form:

size	c_1	c_2	$100c_1/c_2$ [%]
524288	1	1	100.00
262144	4	4	100.00
131072	8	8	100.00
65536	20	20	100.00
32768	70	70	100.00
16384	24	246	100.00
8192	917	917	100.00
4096	3599	3599	100.00
2048	14058	14058	100.00
1024	52399	52392	99.99
512	165644	164225	99.14
256	451932	395326	87.47
128	1090104	758133	69.55
64	2380000	1417686	59.57
32	4801431	2168092	45.16
16	8641007	2019972	23.38
8	13214698	645533	4.88
4	17284323	15947	0.09
2	20392616	15	0.00
1	1	1	100.00
Total	68493078	7656245	11.18

Tab. 4. Experimental results for Enron Corpus.

milions of one bit	1 edge	2 edges	3 edges	4 edges	total states
2	79085	245225	184928	179341	688579
4	128128	431934	356098	363186	1279346
6	196129	647262	527456	539097	1909944
8	258198	850765	700205	718520	2527688
10.	317766	1048291	872085	901511	3139653
12	360992	1214469	1031372	1081907	3688740
14	416066	1398530	1185357	1250925	4250878
16	471296	1583486	1344943	1423656	4823381
18	530333	1773468	1510167	1600480	5414448
20	581271	1951297	1671566	1777232	5981366
22	633157	2128006	1831026	1955279	6547468

Tab. 5. Test Enron.

1. Set density of matrix to $k = 5\%$.
2. Generate random $1024 \times 1024 \times \frac{k}{100}$ bits and insert them to matrix.
3. Minimize matrix.
4. Goto step 2 until density does not reach 95%.

Two tests were performed for randomly generated matrices. Results of tests are given in tables 6(a) and 6(b).

(a) Test 1

k	one bit	1 edges	2 edges	3 edges	s 4 edges	total states
5%	52425	766	4964	6090	6560	18380
15%	157275	91	1512	7132	19578	28313
25%	262125	65	1248	7370	29144	37827
35%	366975	65	1292	9352	39824	50533
45%	471825	60	1164	9680	50565	61469
55%	576675	16	510	6685	54380	61591
65%	681525	5	79	2880	47484	50448
75%	786375	5	11	771	37138	37925
85%	891225	5	6	93	28176	28280
95%	996075	5	6	4	18350	18365

(b) Test 2

k	one bit	1 edge	2 edges	3 edges	s 4 edges	total states
5%	52425	756	4970	6107	6557	18390
15%	157275	80	1466	7253	19841	28640
25%	262125	66	1241	7346	29192	37845
35%	366975	65	1287	9406	39711	50470
45%	471825	58	1133	9733	50453	61377
55%	576675	19	495	6762	54100	61376
65%	681525	6	92	2879	47500	50477
75%	786375	5	17	753	37143	37918
85%	891225	5	6	95	28179	28285
95%	996075	5	6	6	18361	18378

Tab. 6. Artificially generated matrix.

Performed tests show us that the number of states grows until the density of matrix reach 50%. If the density of matrix overcome this value, the number of states become smaller i.e. the corresponding automaton "exchanges" meaning of non-zero elements, and zero-bits become elements with sparse occurence.

5 Conclusion and future works

In this contribution the finite state automata approach to store binary sparse matrix was described. Our experiments prove the approach can serve as efficient binary sparse matrix storage. This approach allows access either to row or column in time $O(\log(n))$, where n is size of the matrix. In our future work, we would like further to test our approach over a huge real matrix and use this approach in clustering methods in information retrieval, formal concept analysis etc.

References

1. R. Barrett, M. Berry, T.F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H.V. der Vorst, Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition. SIAM, Philadelphia, PA, 1994
2. K. Culik, V.Valenta, Finite Automata Based Compression of Bi-Level and Simple Color Images. In Computer and Graphics 21, 1997, 61–68
3. P. Dencker, K. Drre, J. Heuft: Optimization of Parser Tables for Portable Compilers. ACM Transactions on Programming Languages and Systems 6(6), 1984, 546–572
4. I.S. Duff, R.G. Grimes, J.G. Lewis, Sparse Matrix Test Problems. ACM Trans. Math. Softw. 15(1), 1989, 1–14
5. J. Dvorský, Word-Based Compression Methods for Information Retrieval Systems, Ph.D. thesis. Charles University, Prague, Czech Republic, 2004
6. J. Dvorsky, P. Gajdos, E. Ochodkova, J. Martinovic, V. Snasel, Social Network Problem in Enron Corpus (accepted at ADBIS 2005), 2005
7. A. Keprt, V. Snášel, Binary Matrix Pseudo-Division and Its Application. In Proceedings of Znalosti 2005, 2005
8. B. Klimt, Y. Yang, Introducing the Enron Corpus. In Proceedings of First Conference on Email and Anti-Spam (CEAS), 2004
9. G. Rozenberg, E.A. Salomaa, Handbook of Formal Language, volume I. – III, Springer-Verlag Berlin, 1997
10. V. Snášel, J. Dvorský, V. Vondrák, Random Access Storage System for Sparse Matrices. In G. Andrejková and R. Lencses, editors, Proccedings of ITAT 2002, Brdo, High Fatra, Slovakia, 2002
11. R. Yuster, U. Zwick, Fast Sparse Matrix Multiplication. In Proceedings of 12th Annual European Symposium on Algorithms, 2004

On average complexity of InsertSort*

Peter Mlynárčik

Institute of Informatics, Faculty of Sciences UPJŠ, Košice
Jesenná 5, SK-04154 Košice, Slovakia
mlynarci@upjs.sk

Abstract. We consider well-known InsertSort sorting algorithm. It is known that the average complexity of comparisons for sorting n elements by this algorithm is $\frac{1}{4}n^2 + O(n)$. In this paper we derive this result by using specific decomposition of set of n-element permutations; based on this decomposition, we also obtain certain combinatorial identity involving binomial products.

1 Introduction

In this paper, we consider the sorting problem and the InsertSort algorithm applied on it. Each input sequence of length n for a sorting algorithm can be presented by permutation of set $\{1, 2, \ldots, n\}$. Let $Perm(n)$ denote the set of all permutations of the set $\{1, 2, \ldots, n\}$. For integers a, b, the symbol $[a, b]$ denotes the set of all integers c such that $a \leq c \leq b$; if $b < a$ then $[a, b]$ is empty. Let M be any set, then symbol $card(M)$ denotes the cardinality of M. We briefly describe how *InsertSort* ([1]) algorithm works. The general step is as follows: Let us assume that the permutation φ_{i-1} is already sorted at positions $[1, i-1]$ (at the beginning, we put $i = 2$). Now we insert the element $\varphi_{i-1}(i)$ among elements at positions $[1, i]$ to obtain a new permutation φ_i, where elements at positions $[1, i]$ are sorted. We can find the convenient position for the element $\varphi_{i-1}(i)$ easily; by comparing the element $\varphi_{i-1}(i)$ with elements $\varphi_{i-1}(i-1), \varphi_{i-1}(i-2), \ldots, \varphi_{i-1}(k)$, respectively, until $k = 0$ or $\varphi_{i-1}(k) < \varphi_{i-1}(i)$. After comparing, each element at position $j \in [k+1, i-1]$ is moved to position $j+1$, so it will be $\varphi_i(j+1) = \varphi_{i-1}(j)$ and the element $\varphi_{i-1}(i)$ is inserted at position $k+1$. Other elements at position $j \in [i+1, n]$ stay on their places, so it will be $\varphi_i(j) = \varphi_{i-1}(j)$. We obtain new permutation φ_i, such that its elements at positions $[1, i]$ are sorted. The whole process finishes when the element $\varphi_{n-1}(n)$ is inserted among elements at positions $[1, n]$ and we obtain permutation φ_n, which is the identity $(\varphi_n(1) < \varphi_n(2) < \ldots < \varphi_n(n))$. The related question is

> How many comparisons in average case are performed during the process of mentioned algorithm ?

* This work was supported by the Science and Technology Assistance Agency under contract APVT-20-004104, and by the Slovak Grant Agency for Science (VEGA) under contract "Combinatorial Structures and Complexity of Algorithms."

Let $C(\varphi)$ denote the number of comparisons for the permutation φ. Let $C_{average}$ denote the average number of comparisons in InsertSort. So

$$C_{average} = \frac{\left(\sum_{\varphi \in Perm(n)} C(\varphi) \right)}{n!}$$

In further we will express the $C_{average}$ by using the decomposition of $Perm(n)$.

2 Decomposition of permutations

In this section we show how we can decompose set $Perm(n)$ regarding to position $i \in [1, n]$. Let us consider each permutation from $Perm(n)$, where are just j elements at positions $[1, i-1]$ greater than element at position i. More precisely, we consider subset K_i^j of $Perm(n)$, which is defined in next definition

Definition 1. $K_i^j = \{\varphi \in Perm(n) \mid card(\{h \in [1, i-1] \mid \varphi(h) > \varphi(i)\}) = j\}$

It seems that for all $j \in [0, i-1]$, the cardinalities of sets K_i^j are equal. We show it in the Theorem1. At the first let us see the following example.

Example 1. We list all permutations of set $\{1, 2, 3, 4\}$. Let us consider position 3. On the right side of each permutation, there is the number of greater elements at positions $[1, 2]$. We can see, that we can decompose set $Perm(4)$ to three subsets K_3^0, K_3^1, K_3^2. Each subset has the same number of permutations.

1234	0	3124	1
1243	0	3142	0
1324	1	3214	2
1342	0	3241	0
1423	1	3412	2
1432	1	3421	2
2134	0	4123	1
2143	0	4132	1
2314	2	4213	2
2341	0	4231	1
2413	2	4312	2
2431	1	4321	2

It is clear that

$$Perm(n) = \bigcup_{j=0}^{i-1} K_i^j, \text{ and } \bigcap_{j=0}^{i-1} K_i^j = \emptyset \tag{1}$$

hence, the system $\mathcal{S} = \{K_i^j \subseteq Perm(n) \mid K_i^j \text{ is defined as in Definition 1}\}$ is the disjoint decomposition of $Perm(n)$.

After studying structure of sets in Definition 1, the first sketch of proof of following theorem was given by author (based on consideration about cardinality). In a personal communication with Peter Eliaš [3] a bijection between

K_i^j and K_i^{j+1} was found. Author presented Theorem 1 and proof of estimation of average complexity of InsertSort algorithm at seminar [2]. Based on this, Marián Dvorský found a shorter proof of Theorem 1 [4]. We present his proof with his permission [5].

Theorem 1. *Let n be positive integer, $i \in [1, n]$.*
Then for all $j \in [0, i-1]$ $card(K_i^j) = n!/i$.

Proof. The question is

> How many permutations we can choose from the set $Perm(n)$, such that there are exactly j elements greater at positions $[1, i-1]$ than element at position i ?

There are $\binom{n}{i}$ ways to choose i elements $p_1, p_2, \ldots, p_i$ from the set $\{1, 2, \ldots, n\}$. We can assume that the elements $p_1, p_2, \ldots, p_i$ are ordered such that $p_1 < p_2 < \ldots < p_i$. In this sequence there is an element p_{i-j}, which we can put to position i and other elements we put to positions $[1, i-1]$. It is clear that on that positions are just j elements greater than element p_{i-j}. The number of all possible ordering in field $[1, i-1]$ is equal exactly $(i-1)!$. The remaining elements from the set $\{1, 2, \ldots, n\}$ we put to field $[i+1, n]$ and there are $(n-i)!$ ways of ordering them. So we can choose $\binom{n}{i}(i-1)!(n-i)!$ permutations where are exactly j elements greater at positions $[1, i-1]$ than element at position i. So after adjusting previous expression we obtain number $n!/i$. The obtained expression does not contain value j, so for each $j \in [0, i-1]$ is result the same.

Based on previous result we can achieve next binomial equation.

Corollary 1. *Let n be positive integer, $i \in [1, n]$, $j \in [0, i-1]$. Then*

$$\sum_{l=i-j}^{n-j} \binom{n-l}{j} \binom{l-1}{i-j-1} = \binom{n}{i}$$

Proof. We express $card(K_i^j)$ in other way, as in proof of Theorem1, so we obtain another expression which will be equal to $n!/i$. Let us assume, that at position i is element $l \in [1, n]$. In the field $[1, i-1]$ must be j elements greater than l, hence there must be $i-j-1$ elements less than l, so it must be $(i-j) \leq l$. On the other hand l can not be greater than $n-j$, because there must exist at least j elements greater as l in $[1, n]$. So we obtain $(i-j) \leq l \leq (n-j)$. We have $n-l$ greater elements than l so we can choose j in elements $\binom{n-l}{j}$ ways to positions at field $[1, i-1]$. We have $l-1$ less elements than l so we can choose $i-j-1$ elements in $\binom{l-1}{i-j-1}$ the ways to positions in field $[1, i-1]$. Hence we can choose $\binom{n-l}{j}\binom{l-1}{i-j-1}$ ways elements at field $[1, i-1]$. If we count each ordering of choosed elements we obtain the number $\binom{n-l}{j}\binom{l-1}{i-j-1}(i-1)!$. Moreover, we have to add each ordering of remaining elements at field $[i+1, n]$ so at last we achieve the number $\binom{n-l}{j}\binom{l-1}{i-j-1}(i-1)!(n-i)!$. Summing it for all l we obtain the next expression

$$card(K_i^j) = \sum_{l=i-j}^{n-j} \binom{n-l}{j} \binom{l-1}{i-j-1} (i-1)!(n-i)!$$

Hence,

$$n!/i = \sum_{l=i-j}^{n-j} \binom{n-l}{j} \binom{l-1}{i-j-1} (i-1)!(n-i)!$$

Adjusting the previous expression we have

$$\sum_{l=i-j}^{n-j} \binom{n-l}{j} \binom{l-1}{i-j-1} = \binom{n}{i}$$

3 The average value of comparisons in InsertSort

As we define above

$$C_{average} = \frac{\left(\sum_{\varphi \in Perm(n)} C(\varphi) \right)}{n!}$$

Let us adjust the expression $\sum_{\varphi \in Perm(n)} C(\varphi)$. Let us denote $C_i(\varphi)$ the number of comparisons for inserting an element at position i. Obviously, $C(\varphi) = \sum_{i=2}^{n} C_i(\varphi)$. So we can write

$$\sum_{\varphi \in Perm(n)} C(\varphi) = \sum_{\varphi \in Perm(n)} \sum_{i=2}^{n} C_i(\varphi) \tag{2}$$

We can exchange the order of the sums, so we obtain

$$\sum_{\varphi \in Perm(n)} C(\varphi) = \sum_{i=2}^{n} \sum_{\varphi \in Perm(n)} C_i(\varphi)$$

According to 1, we have

$$\sum_{\varphi \in Perm(n)} C(\varphi) = \sum_{\varphi \in K_i^0} C_i(\varphi) + \sum_{\varphi \in K_i^1} C_i(\varphi) + \ldots + \sum_{\varphi \in K_i^{i-1}} C_i(\varphi)$$

For permutations from class K_i^j is $C_i(\varphi) = j + 1$ except of K_i^{i-1}, where the last comparing is not between elements, but it is only a test if position is equal to 1, so according to Theorem1, $\sum_{\varphi \in K_i^j} C_i(\varphi) = \frac{n!}{i}(j+1)$, for $j \in [0, i-2]$ and

for $j = i - 1$ it is $\sum_{\varphi \in K_i^{i-1}} C_i(\varphi) = \frac{n!}{i}(i-1)$. We replace sums to corresponding expression, then we obtain

$$\sum_{\varphi \in Perm(n)} C(\varphi) = \frac{n!}{i}1 + \frac{n!}{i}2 + \ldots \frac{n!}{i}(i-1) + \frac{n!}{i}(i-1) = \frac{n!}{i}(1+2+\ldots(i-1)+(i-1))$$

After further adjustings we obtain

$$\sum_{\varphi \in Perm(n)} C(\varphi) = \frac{n!}{i}\left(\frac{(i+1)i}{2} - 1\right) = n!\left(\frac{i+1}{2} - \frac{1}{i}\right)$$

Hence,

$$C_{average} = \left(\sum_{i=2}^{n} n!\left(\frac{i+1}{2} - \frac{1}{i}\right)\right)/n! = \sum_{i=2}^{n}\left(\frac{i+1}{2} - \frac{1}{i}\right) = \frac{1}{2}\sum_{i=2}^{n}(i+1) - \sum_{i=2}^{n}\frac{1}{i}$$

Finally we obtain

$$C_{average} = \frac{n^2+3n-4}{4} - \sum_{i=2}^{n}\frac{1}{i} \tag{3}$$

Theorem 2. *Let n be positive integer. Then the average number of comparisons of InsertSort algorithm for sorting n elements is bounded*

$$\frac{n^2+3n-4}{4} - (\log_2 n + 1) < C_{average} \leq \frac{n^2+3n-4}{4} - \frac{\log_2 n - 1}{2}$$

Proof. Let us consider the relation 3 and estimate $\sum_{i=2}^{n}\frac{1}{i}$. Let k be positive integer such that $2^k \leq n < 2^{k+1}$. Then we obtain estimation

$$\sum_{j=1}^{k}\frac{1}{2^j}2^{j-1} \leq \sum_{i=2}^{n}\frac{1}{i} < \sum_{j=1}^{k+1}\frac{1}{2^j}2^j$$

So, $k\frac{1}{2} \leq \sum_{i=2}^{n}\frac{1}{i} < k+1$. Hence $k = \lfloor \log_2 n \rfloor$ and we obtain

$$\lfloor \log_2 n \rfloor \frac{1}{2} \leq \sum_{i=2}^{n}\frac{1}{i} < \lfloor \log_2 n \rfloor + 1$$

$$\frac{log_2 n - 1}{2} \leq \sum_{i=2}^{n}\frac{1}{i} < \log_2 n + 1$$

At the last

$$\frac{n^2+3n-4}{4} - (\log_2 n + 1) < C_{average} \leq \frac{n^2+3n-4}{4} - \frac{\log_2 n - 1}{2}$$

4 Acknowledgement

I would like to thank Peter Eliaš for his idea in first form of proof of Theorem1 and Marián Dvorský for his proof which is presented in this paper.

References

1. Donald E. Knuth, The art of computer programming, Sorting and Searching, Vol. 3, Addison Wesley, 1973
2. Peter Mlynárčik, Presentation of Theorem1 and estimation of average complexity of comparisons of InserSort algorithm on seminar of Prof.Viliam Geffert in autumn 2004
3. Peter Eliaš, Personal communication, mail: elias@kosice.upjs.sk
4. Marián Dvorský, Presentation of proof of Theorem1 on seminar of Prof.Viliam Geffert in autumn 2004
5. Marián Dvorský, Personal communication, mail: marian@step.sk

Attacking PGV-like rate-2 cryptographic compression functions*

Richard Ostertág and Martin Stanek

Department of Computer Science
Faculty of Mathematics Physics and Informatics Comenius University
Mlynská dolina, 842 48 Bratislava, Slovak Republic
{ostertag, stanek}@dcs.fmph.uniba.sk

Abstract. The security of iterated hash functions relies on the properties of underlying compression functions. We study highly efficient compression functions based on block ciphers. The first systematic analysis of rate-1 compression functions was done by Preneel, Govaerts, and Vandewalle [6]. We propose a model for high-rate compression functions, and show that all natural generalizations of constructions from [6] to the case of rate-2 compression functions are not collision resistant. Moreover, the attacks are carried out by polynomially bounded adversaries.

Key words: Compression functions, block ciphers, provable security.

1 Introduction

Cryptographic hash functions are basic building blocks in many security constructions – digital signatures, message authentication codes, etc. Almost all modern hash functions are built by iterating a compression function according the Merkle-Damgård paradigm [3], [5]. Moreover, these compression functions are often based on some underlying block cipher. Even dedicated hash function like SHA-1 is based on a block cipher called SHACAL-1 [4].

The first systematic study of 64 block cipher based hash functions was done by Preneel, Govaerts, and Vandewalle [6]. Subsequently, Black, Rogaway, and Shrimpton [2] analyzed these constructions in black box model and proved that 20 of them are collision resistant up to the birthday-attack bound.

An important property of a hash function is performance. Therefore, one would like to maximize the rate of hash function – the number of message blocks processed with one block cipher transformation. Another way to design fast hash functions is the usage of keys from a small fixed set of keys in all block cipher transformations, thus enabling a pre-scheduling of keys. Classical constructions [6] are rate-1 and require rekeying for every message block. Recently, Black, Cochran, and Shrimpton [1] shown, that it is impossible to construct (block cipher based) provably secure rate-1 iterated hash function that use small fixed set of keys.

* This work was supported in part by VEGA grant No. 1/0131/03.

Our contribution. We propose a general model of (block cipher based) high-rate compression functions. We show that generalizations of rate-1 constructions by Preneel, Govaerts, and Vandewalle [6] to the case of rate-2 compression functions are not collision resistant. Thus, its possible to find two distinct inputs that yield the same output in more efficient manner than using birthday attack. Moreover, the attacks require only polynomially bounded adversaries asking constant number oracle queries (hence being "stronger" than attacks in standard black-box model).

Remark 1. We focus solely on collision resistance as the "most problematic" property of cryptographic hash functions. Moreover, the results of our analysis are mostly negative, so there is no need to study other properties.

The paper is structured as follows. Section 2 contains notions and definitions used in the paper. In addition, we present our model of high-rate compression functions. The analysis of rate-2 compression functions is presented in Section 3.

2 Background and definitions

The notation used in the paper follows closely the notation introduced in [1], [2]. Let V_m be a set of all m-ary binary vectors, i.e. $V_m = \{0,1\}^m$, and let $V_m^* = (V_m)^*$ be a set of all binary strings obtained as a concatenation of (zero or more) elements from V_m. Let k and n be positive integers. A block cipher is a function $E : V_k \times V_n \to V_n$, where for each key $K \in V_k$, the function $E_K(\cdot) = E(K, \cdot)$ is a permutation on V_n. Let $\mathrm{Bloc}(k,n)$ be the set of all block ciphers $E : V_k \times V_n \to V_n$. The inverse of block cipher E is denoted by E^{-1}.

A (block cipher based) compression function is a function $f : \mathrm{Bloc}(k,n) \times (V_a \times V_b) \to V_c$, where a, b, and c are positive integers such that $a + b \geq c$. We write the first argument as a superscript of compression function, i.e. $f^E(\cdot,\cdot) = f(E,\cdot,\cdot)$. An iterated hash of compression function $f : \mathrm{Bloc}(k,n) \times (V_a \times V_b) \to V_a$ is the hash function $H : \mathrm{Bloc}(k,n) \times V_b^* \to V_a$ defined by $H^E(m_1 \dots m_l) = h_l$, where $h_i = f^E(h_{i-1}, m_i)$ and h_0 is a fixed element from V_a. We set $H^E(\varepsilon) = h_0$ for empty string ε. We often omit superscripts E to f and H, when the block cipher is known from context. If the computation of $f^E(h,m)$ uses e evaluations of E then f (and its iterated hash H) is rate-r, where $r = (b/n)/e$. Often $n \mid b$, and the rate represents the average number of message blocks processed by single E transformation. For example, for $b/n = 3$ and $e = 2$ we get compression function of rate-$(3/2)$.

The experiment of choosing a random element x from the finite set S will be denoted by $x \xleftarrow{\$} S$.

Black-box model. An adversary A is given access to oracles E and E^{-1} where E is a block cipher. We write these oracles as superscripts, i.e. $A^{E,E^{-1}}$. We omit the superscripts when the oracles are clear from context. The adversary's task is attacking the collision resistance of a hash function H. We measure the

adversary's effort of finding collision as a function of the number of E or E^{-1} queries it makes. Notice that we assume information-theoretic adversary, i.e. the computational power of the adversary is not limited in any way.

Attacks in this model treat the block cipher as a black box. The only structural property of the block cipher captured by the model is the invertibility. The model cannot guarantee the security of block cipher based hash functions instantiated with block ciphers having significant structural properties (e.g. weak keys). On the other hand, the black-box model is stronger than treating the block cipher as a random function, because the adversary's ability to compute E^{-1}.

We define the advantage of an adversary in finding collisions in a compression function $f : \mathrm{Bloc}(k,n) \times (V_a \times V_b) \to V_c$. Naturally (h,m) and (h',m') collide under f if they are distinct and $f^E(h,m) = f^E(h',m')$. We also take into account a collision with empty string, i.e. producing (h,m) such that $f^E(h,m) = h_0$. We look at the number of queries that the adversary makes compare this with the probability of finding a collision.

Definition 1 (Collision resistance of a compression function [2]). *Let f be a block cipher based compression function, $f : \mathrm{Bloc}(k,n) \times (V_a \times V_b) \to V_c$. Fix a constant $h_0 \in V_c$ and an adversary A. Then the advantage of finding collisions in f is the probability*

$$\mathbf{Adv}_f^{\mathrm{comp}}(A) = \Pr\Big[E \xleftarrow{\$} \mathrm{Bloc}(k,n);\ ((h,m),(h',m')) \leftarrow A^{E,E^{-1}} : (h,m) \neq (h',m') \wedge f^E(h,m) = f^E(h',m') \vee f^E(h,m) = h_0\Big]$$

For $q \geq 0$ we write $\mathbf{Adv}_f^{\mathrm{comp}}(q) = \max_A\{\mathbf{Adv}_f^{\mathrm{comp}}(A)\}$ where the maximum is taken over all adversaries that ask at most q oracle (E or E^{-1}) queries.

Definition 2 (Collision resistance of a hash function [2]). *Let H be a block cipher based hash function, and let A be an adversary. Then the advantage of finding collisions in H is the probability*

$$\mathbf{Adv}_H^{\mathrm{coll}}(A) = \Pr\Big[E \xleftarrow{\$} \mathrm{Bloc}(k,n);\ (M,M') \leftarrow A^{E,E^{-1}} : M \neq M' \wedge H^E(M) = H^E(M')\Big]$$

For $q \geq 0$ we write $\mathbf{Adv}_H^{\mathrm{coll}}(q) = \max_A\{\mathbf{Adv}_H^{\mathrm{coll}}(A)\}$ where the maximum is taken over all adversaries that ask at most q oracle (E or E^{-1}) queries.

The following theorem forms a basis for construction of iterated hash functions (Merkle-Damgård paradigm). It shows that the collision resistance of compression function is sufficient for the collision resistance of its iterated hash function.

Theorem 1 (Merkle-Damgård [3], [5]). *Let $f : \mathrm{Bloc}(k,n) \times V_n \times V_n \to V_n$ be a compression function and let H be an iterated hash of f. Then $\mathbf{Adv}_H^{\mathrm{coll}}(q) \leq \mathbf{Adv}_f^{\mathrm{comp}}(q)$ for any $q \geq 1$.*

Remark 2. Birthday attack is a generic collision-finding attack on a compression/hash function. The advantage of birthday attack is $\Theta(q^2/2^n)$, where q is number of evaluations of the function and n is the length of output. Usually, a compression function f (hash function H) is called collision resistant up to the birthday-attack bound, or simply collision resistant if $\mathbf{Adv}_f^{\mathrm{comp}}(q) = O(q^2/2^n)$ ($\mathbf{Adv}_H^{\mathrm{coll}}(q)) = O(q^2/2^n)$). Since the birthday attack is always applicable, these equations can be stated equivalently as $\mathbf{Adv}_f^{\mathrm{comp}}(q) = \Theta(q^2/2^n)$ ($\mathbf{Adv}_H^{\mathrm{coll}}(q)) = \Theta(q^2/2^n)$)

2.1 A model of high-rate compression function

We define a model of high-rate compression function $f : \mathrm{Bloc}(k, n) \times (V_a \times V_b) \to V_a$. The model assumes following:

- The evaluation of compression function f uses a single query E to compute $f(h, m)$.
- The length of m is an integer multiple of the E's block length n, i.e. $b = rn$ where $r \geq 1$.

According to the second condition the compression function is rate-r. Let $f_1 : V_a \times V_{rn} \to V_n$, $f_2 : V_a \times V_{rn} \to V_k$, and $f_3 : V_a \times V_{rn} \times V_n \to V_a$ be arbitrary functions. The computation of the compression function $f : \mathrm{Bloc}(k, n) \times (V_a \times V_{nr}) \to V_a$ is defined as:

$$
\begin{aligned}
&\textbf{function } f^E(h, m) : \\
&X \leftarrow f_1(h, m) \\
&K \leftarrow f_2(h, m) \\
&Y \leftarrow E(K, X) \\
&\textbf{return } f_3(h, m, Y)
\end{aligned}
$$

When convenient we express m as a concatenation of n bit blocks. These r blocks are denoted by $m^{(1)}, \ldots, m^{(r)}$. Our model of high-rate compression function used in iterated hash is depicted in Fig. 1.

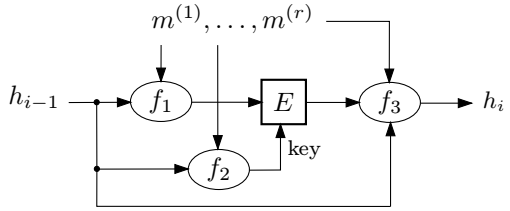


Fig. 1. Model of high-rate compression function.

The model is quite general – it covers all compression functions that takes r blocks of a message and process them using exactly one encryption transformation E. Notice that all rate-1 schemes from [6] (we call them PGV) are special instances of the model.

3 PGV-like rate-2 compression functions

The constructions of compression functions from block cipher often assume equal key and block lengths, i.e. $k = n$. Similarly, the output of compression function has usually the same length as the block, i.e. $a = n$. Thus, we consider rate-2 compression functions of the form $f : V_n \times V_{2n} \to V_n$.

Preneel, Govaerts, and Vandewalle [6] studied rate-1 compression functions. They considered all 64 compression functions f of the form $f(h, m) = E_a(b) \oplus c$ where $a, b, c \in \{h, m, h \oplus m, v\}$ (v is a fixed constant). As showed in [2], 12 compression function are collision resistant, and additional 8, though not collision resistant, form collision resistant hash functions.

A natural extension of above constructions to the case of rate-2 compression functions is the following scheme:

$$f(h, (m^{(1)}, m^{(2)})) = E_a(b) \oplus c, \tag{1}$$

where $a, b, c \in \{h, m^{(1)}, m^{(2)}, h \oplus m^{(1)}, h \oplus m^{(2)}, m^{(1)} \oplus m^{(2)}, h \oplus m^{(1)} \oplus m^{(2)}, v\}$. This way we obtain 512 compression functions.

Remark 3. Notice that the compression functions instantiated in the scheme fall in our model – $f_1(h, m) = b$, $f_2(h, m) = a$, and $f_3(h, m, Y) = Y \oplus c$.

We show that no compression function of the form (1) is collision resistant (for any function there exists an adversary that finds a collision and asks at most two queries). We partition these functions into distinct classes according attacks that find (at least one) collision. Summary of the classes is given in Table 1. For each class the table shows the number of compression functions in the class, and the number of oracle queries needed in collision finding attack.

class	functions	queries
1 – Superfluous Variables	169	0
2 – Balanced Combinations	133	0
3 – Compensations	150	2
4 – "Hard" Core	60	$2, 0$

Tab. 1. Collision classes of rate-2 compression functions.

Remark 4. There are compression functions that are vulnerable to multiple attacks using e.g. superfluous variables or balanced combinations. In such situation we assign particular function to the class with lowest number.

3.1 Class 1 – superfluous variables

First class contains all those compression functions that do not depend on all input vectors, i.e. at least one of h, $m^{(1)}$, $m^{(2)}$ is not required for computing the function. Examples of such compression functions are $E_h(h \oplus m^{(1)}) \oplus h$, $E_{m^{(2)}}(v) \oplus m^{(1)} \oplus m^{(2)}$, or $E_v(h \oplus m^{(2)}) \oplus h \oplus m^{(2)}$. Trivially, one can find many collisions in compression functions from this class. It suffices to vary the superfluous variable. Moreover, no oracle queries are needed to produce collisions.

3.2 Class 2 – balanced combinations

Our second class consists of those compression functions that are not in class 1, and have a balanced combination of two input vectors. Let $x_1, x_2 \in \{h, m^{(1)}, m^{(2)}\}$ be two distinct input vectors, i.e. $x_1 \neq x_2$. We call a combination $x_1 \oplus x_2$ balanced in compression function f, if every occurrence of x_1 in f's parameters a, b or c implies x_2 occurrence in the same parameters (and vice versa). Examples of compression functions in this class are:

$$E_{h \oplus m^{(1)} \oplus m^{(2)}}(m^{(1)} \oplus m^{(2)}) \oplus v,$$
$$E_{m^{(2)}}(h \oplus m^{(1)}) \oplus h \oplus m^{(1)},$$
$$E_{h \oplus m^{(2)}}(h \oplus m^{(1)} \oplus m^{(2)}) \oplus h \oplus m^{(1)} \oplus m^{(2)}.$$

It can be easily seen that collisions can be find without any oracle queries. Balanced combination $x_1 \oplus x_2$ allows choosing 2^n values pairs (x_1, x_2) without changing parameters a, b, and c. Hence, the value of f does not change either.

3.3 Class 3 – compensations

Third class of compression functions contains those functions (not in classes 1 and 2) that have some input vector solely in either parameter b or parameter c. Let $x \in \{h, m^{(1)}, m^{(2)}\}$ be such input vector. Let x appears only in f's parameter b, i.e. input of block cipher transformation. An adversary can find collision in the following way. He sets the output of f to some fixed value z. Similarly he sets the values of all input vectors except x randomly. Using a query to E^{-1} oracle the adversary can compute "suitable" x value. Repeating this procedure for different random choice of input vectors values and the same fixed z, the adversary obtains a collision for f. The situation for x appearing solely in parameter c is treated analogously.

Examples of compression functions in this class are:

$$E_{m^{(2)}}(m^{(1)}) \oplus h \oplus m^{(2)},$$
$$E_{m^{(1)}}(m^{(2)}) \oplus h \oplus m^{(1)} \oplus m^{(2)},$$
$$E_{h \oplus m^{(1)}}(m^{(1)} \oplus m^{(2)}) \oplus v.$$

3.4 Class 4 – "hard" core

There are 60 compression functions left after sorting the functions into classes 1, 2, and 3. Let us denote this set C. We call two functions f_1, f_2 permutation-equivalent, if f_1 can be obtained from f_2 by some permutation of its inputs. It can be easily observed that $\mathbf{Adv}^{\mathrm{comp}}_{f_1}(q) = \mathbf{Adv}^{\mathrm{comp}}_{f_2}(q)$ for any permutation-equivalent compression functions f_1, f_2, and any $q \geq 0$. Therefore the set C can be partitioned to equivalence classes. Since every equivalence class has 6 members, it suffices to analyze the collision resistance of any 10 permutation-nonequivalent compression functions (each one drawn from different equivalence class). One selection of these 10 functions is shown in Table 2.

i	a	b	c
1	$m^{(1)} \oplus m^{(2)}$	h	$h \oplus m^{(2)}$
2	$m^{(1)} \oplus m^{(2)}$	$h \oplus m^{(2)}$	h
3	$m^{(1)} \oplus m^{(2)}$	$h \oplus m^{(2)}$	$h \oplus m^{(2)}$
4	$m^{(1)} \oplus m^{(2)}$	$h \oplus m^{(2)}$	$h \oplus m^{(1)}$
5	$m^{(1)} \oplus m^{(2)}$	$h \oplus m^{(2)}$	$h \oplus m^{(1)} \oplus m^{(2)}$
6	$m^{(1)} \oplus m^{(2)}$	$h \oplus m^{(1)} \oplus m^{(2)}$	$h \oplus m^{(2)}$
7	$h \oplus m^{(1)} \oplus m^{(2)}$	$m^{(2)}$	$m^{(1)}$
8	$h \oplus m^{(1)} \oplus m^{(2)}$	$m^{(2)}$	$m^{(1)} \oplus m^{(2)}$
9	$h \oplus m^{(1)} \oplus m^{(2)}$	$m^{(1)} \oplus m^{(2)}$	$m^{(2)}$
10	$h \oplus m^{(1)} \oplus m^{(2)}$	$m^{(1)} \oplus m^{(2)}$	$h \oplus m^{(2)}$

Tab. 2. Permutation-nonequivalent compression functions of "hard" core class.

Now we show collisions in all 10 permutation-nonequivalent compression functions. Hence, all 60 functions from set C are not collision resistant. We refer compression functions from Table 2 as $f_1, \ldots, f_{10}$. For brevity, let 0 (1) be all-zero (all-one) vector in V_n, respectively. Let A be following collision finding adversary:

1. A sets $(h, m^{(1)}, m^{(2)}) = (0, 0, 0)$, i.e. $f_i(h, m^{(1)}, m^{(2)}) = E_0(0)$.
2. A asks two oracle queries and computes $x = E_0(0) \oplus E_1(0)$.
3. A solves following equations (a', b', c' denote corresponding linear combinations of h', $m'^{(1)}$, $m'^{(2)}$ for compression function f_i):

$$a' = 1$$
$$b' = 0$$
$$c' = x$$

For any solution $(h', m'^{(1)}, m'^{(2)})$ we have $f_i(h', m'^{(1)}, m'^{(2)}) = E_{a'}(b') \oplus c' = E_1(0) \oplus E_0(0) \oplus E_1(0) = E_0(0)$. Hence any solution different from $(0, 0, 0)$ yields a collision.

Let us illustrate adversary's computation on f_1. Adversary A solves following equations in step 3:

$$\begin{aligned} m'^{(1)} \oplus m'^{(2)} &= 1 \\ h' &= 0 \\ h' \oplus m'^{(2)} &= x \end{aligned}$$

Solution is $(h', m'^{(1)}, m'^{(2)}) = (0, 1 \oplus x, x) \neq (0, 0, 0)$, and A obtains a collision.

Adversary A can successfully find collisions for all functions $f_1, \ldots, f_{10}$, see Table 3, except for f_3 and f_4 due to the linear dependence of a', b', and c'. We produce collisions for these functions separately (moreover, no oracle queries are needed). It can be easily verified that $f_3(0,0,0) = f_3(1,1,1)$, and $f_4(0,0,0) = f_4(1,1,1)$.

i	h'	$m'^{(1)}$	$m'^{(2)}$
1	0	$1 \oplus x$	x
2	x	$1 \oplus x$	x
3	–	–	–
4	–	–	–
5	$1 \oplus x$	x	$1 \oplus x$
6	1	x	$1 \oplus x$
7	$1 \oplus x$	x	0
8	$1 \oplus x$	x	0
9	1	x	x
10	1	$1 \oplus x$	$1 \oplus x$

Tab. 3. Collisions for $(0,0,0)$ produced by adversary A.

Summarizing attacks from this section we obtain following theorem:

Theorem 2. *Let $E \in \mathrm{Bloc}(n,k)$. Let $f : V_n \times V_{2n} \to V_n$ be a compression function defined as $f(h,(m^{(1)},m^{(2)})) = E_a(b) \oplus c$, where $a,b,c \in \{h, m^{(1)}, m^{(2)}, h \oplus m^{(1)}, h \oplus m^{(2)}, m^{(1)} \oplus m^{(2)}, h \oplus m^{(1)} \oplus m^{(2)}, v\}$. Then $\mathbf{Adv}_f^{\mathrm{comp}}(2) = 1$.*

Remark 5. The attacks presented in this section do not use the full strength of black-box model – a computationally unbounded adversary. These attacks require just polynomially bounded adversary asking constant number oracle queries.

4 Conclusion

Many interesting questions arise from the results presented in the paper. We state two most prominent ones as open problems:

1. Are there any collision resistant compression functions with rate > 1?
2. Collision resistance of compression function is sufficient, but not necessary condition for collision resistance of iterated hash [2]. Are there any PGV-like rate-2 compression functions that result in collision resistant iterated hash function?

References

1. Black J., Cochran M., Shrimpton T., On the Impossibility of Highly-Efficient Blockcipher-Based Hash Functions, In Advances in Cryptology – Eurocrypt '05, LNCS 3494, Springer-Verlag, 2005
2. Black J., Rogaway P., Shrimpton T., Black-box analysis of the block-cipher-based hash-function constructions from PGV, In Advances in Cryptology – CRYPTO '02, LNCS 2442, Springer-Verlag, 2002
3. Damgård I., A design principle for hash functions, In Advances in Cryptology – CRYPTO '89, LNCS 435, Springer-Verlag, 1990
4. Handschuh H., Knudsen L., Robshaw M., Analysis of SHA-1 in encryption mode, In Advances in Cryptology – CT-RSA '01, LNCS 2020, Springer-Verlag, 2001
5. Merkle R., One way hash functions and DES, In Advances in Cryptology – CRYPTO '89, LNCS 435, Springer-Verlag, 1990
6. Preneel B., Govaerts R., Vandewalle J., Hash functions based on block ciphers: A synthetic approach, In Advances in Cryptology – CRYPTO '93, LNCS 773, Springer-Verlag, 1994

Alternatívna charakterizácia synchronizovaného alternovania*

Dana Pardubská[1] and Jiří Wiedermann[2]

[1] Katedra informatiky
Fakulta matematiky, fyziky a informatiky Univerzity Komenského
Mlynská dolina, 842 48 Bratislava, Slovensko
pardubska@fmph.uniba.sk
[2] Ústav informatiky Akademie věd České republiky
Pod Vodárenskou věží 2, 182 07 Praha 8, Česká republika
jiri.wiedermann@cs.cas.cz

Abstrakt V článku navrhneme nový výpočtový model, ktorého návrh je inšpirovaný základnými vlastnosťami bezdrôtových mobilných komunikačných sietí. V podstate ide o deterministický Turingov stroj obohatený o možnosť vytvárania nových komunikujúcich procesov, pričom "bezdrôtová"komunikácia medzi procesmi prebieha prostredníctvom explicitne určených kanálov. Dokážeme, že výpočtovo sú tieto stroje časovo aj priestorovo ekvivalentné synchronizovaným alternujúcim Turingovým strojom. To ukazuje, že synchronizované alternovanie možno nahradiť deterministickým paralelizmom s možnosťou neobmedzenej komunikácie a naopak.

1 Úvod

Prirodzeným dôsledkom vývoja bezdrôtových a mobilných komunikačných technológií je zvýšenie záujmu o skúmanie algoritmických aspektov odpovedajúcich výpočtových a komunikačných mechanizmov. Zdá sa, že doterajší výskum sa sústredil najmä na otázky konkrétnych algoritmov, pričom sa takmer úplne zabúdalo na zložitostné aspekty týchto výpočtov. Jedným z dôvodov môže byť nedostatok formálnych modelov pre takýto typ výpočtov. Ak sa pokúsime identifikovať základné výpočtové a komunikačné vlastnosti príslušných výpočtov, po značnom zjednodušení skončíme pri dynamických rekonfigurovateľných sieťach mobilných procesorov, ktoré na vzájomnú komunikáciu používajú "rádio", ktoré sa dá naladiť pre príjem či vysielanie na ľubovoľnom kanáli. Správu, vysielanú nejakým procesorom (vrcholom siete) na jednom kanáli, tak môžu počuť všetky na tento kanál naladené vrcholy.

Čo sa dá povedať o výpočtovej sile práve načrtnutého "výpočtového modelu"? Prináša — v porovaní s klasickými výpočtami — bezdrôtové mobilné

* Výskum bol čiastočne podporovaný grantom APVT-20-018902. Práca vznikla v rámci výskumného zámeru AV0Z10300504 a bola čiastočne podporovaná grantom 1ET100300517.

počítanie do výpočtov novú kvalitu? Odpovede na tieto otázky sú hlavnou motiváciou tohto článku. Aby sme na ne mohli odpovedať, budeme definovať formálny model, ktorý zachytáva hlavné črty mobilných bezdrôtových výpočtov, dôležité z hľadiska ich funkčnosti. Snaha o zachovanie istej realističnosti modelu nás doviedla k návrhu deterministického zariadenia. V podstate to bude Turingov stroj, ktorý bude schopný deterministicky spúšťať paralelné procesy s možnosťou vytvárať komunikujúce skupiny. Pri jeho návrhu sme sa snažili o transparentnosť komunikačného mechanizmu. Aby sa mohlo uskutočniť spojenie v rámci rôznych skupín procesov, musia si tieto skupiny procesov alokovať rôzne kanály (avšak rovnaké pre danú skupinu). Rovnaký mechanizmus medziprocesorovej komunikácie použijeme aj na identifikáciu ukončenia výpočtu. Spomínaná transparentnosť tak zviditeľňuje (a robí merateľnými) aktivity, ktoré sa v prípade alternujúcich Turingových strojov dejú "za scénouä sú z pohľadu výpočtovej zložitosti zadarmo.

Ani zďaleka netvrdíme, že náš model je ideálnym modelom mobilných bezdrôtových výpočtov. Napriek tomu vidíme jeho dva prínosy. Po prvé, náš model charakterizuje výpočtovú silu istého typu bezdrôtových mobilných výpočtov. Druhým, a možno dokonca dôležitejším prínosom tohto modelu je alternatívna charakterizácia výpočtového mechanizmu synchronizovaného alternovania, čo prináša novú charakterizáciu pamäťových tried PSPACE a EXPTIME. Ukazuje sa, že synchronizované alternovanie je ekvivalentné deterministickému paralelizmu s explicitným mechanizmom komunikácie.

V našej práci sme vychádzali z princípu alternovania (napr. [1]), ktorý sa v teórii zložitosti považuje za najelegantnejší nástroj na štúdium paralelných výpočtov, a z naväzujúcich prác [6], [3] a [2], ktoré rozširujú mechanizmus alternovanie o možnosti komunikácie medzi bežiacimi procesmi. Vnímať synchronizované alternujúce procesy ako bezdrôtovo komunikujúce mobilné procesy sa na prvý pohľad môže zdať prehnané; v skutočnosti to tak nie je. Už aj klasický alternujúci Turingov stroj v podstate využíva "bezdrôtovú"komunikáciu medzi bežiacimi procesmi pri určovaní terminácie výpočtu. Mobilita procesov je v ich dynamickom vzniku a zániku a tiež v tom, že nie sú viazané konkrétnym umiestnením v priestore.

Pri návrhu modelu sme upustili od nedeterminizmu, ktorý je základnou zložkou alternovania. Namiesto nedeterminizmu a synchronizácie sme zaviedli univerzálny mechanizmus deterministického vymieňania správ medzi procesmi. Hlavný výsledok práce je vcelku prekvapivý — tento nový mechanizmus nielenže dokázal nahradiť stratu nedeterminizmu a akceptačného mechanizmu alternujúcich Turingových strojov, ale zachoval aj výhodu, známu zo synchronizovaných Turingových strojov, a síce väčšiu silu synchronizovaného priestoru, ktorá je dokumentovaná ekvivalenciou tohoto priestoru so sychronizovaným exponenciálne väčším časom (pozri napr. [6], [3]). Dostali sme stroj, ktorý je ekvivalentný synchronizovaným alternujúcim Turingovým strojom.

Štruktúra článku je nasledovná. V časti 2 definujeme model, tzv. sieť Turingových strojov (WPTM — z anglického wireless parallel Turing machine), spolu s príslušnými zložitostnými mierami. V časti 3 sa venujeme základnej zlo-

žitostnej charakterizácii výpočtov takýchto strojov a nakoniec v časti 4 zhrnieme dosiahnuté výsledky.

2 Sieť Turingových strojov — WPTM

Pri vytváraní výpočtového modelu, v ktorom procesy komunikujú podobne ako prvky bezdrôtových mobilných sietí, využijeme schopnosť procesov alternujúcich Turingových strojov vytvárať v univerzálnom stave nové procesy. Túto schopnosť doplníme o špeciálny mechanizmus modelujúci bezdrôtový prenos informácie medzi procesmi naladenými na rovnaký kanál. Akceptačný mechanizmus alternujúcich strojov nahradíme flexibilným komunikačným mechanizmom.

Definícia 1 *Sieť k–páskových Turingových strojov (WPTM) s jednou vstupnou a jednou kanálovou páskou je systém $M = (k, Q, R, \Sigma, \Gamma, \Delta, q_0, r_0, q_{accept}, q_{reject})$, kde*

- *k je počet pracovných pások;*
- *$Q \times R$ je konečná množina stavov.*
  - *Q je množina pracovných stavov, q_0, $q_0 \in Q$ je počiatočný stav;*
  - *R je množina komunikačných stavov, ktorá obsahuje tri špeciálne stavy: počiatočný komunikačný stav r_0,* akceptujúci *stav q_{accept} a* zamietajúci *stav q_{reject};*
- *Σ je konečná vstupná abeceda ($\$ \notin \Sigma$ je tzv. zarážka);*
- *Γ je konečná pracovná abeceda ($\sharp \in \Gamma$ je prázdny symbol, $\sharp \notin \Sigma$);*
- *$\Delta \subseteq Q \times R \times (\Sigma \cup \{\$\}) \times \Gamma^{k+1} \times Q \times R \times (\Gamma - \{\sharp\})^{k+1} \times \{\leftarrow, \rightarrow\}^{k+2}$ je prechodová relácia.*

Prvkom Δ hovoríme prechody. Stroj má (iba čítaciu) vstupnú pásku so zarážkami, k pracovných pások a jednu kanálovú pásku; pracovné pásky spolu s kanálovou páskou budeme súhrnne nazývať pásky. Pásky sú jednosmerne nekonečné smerom doprava, číslovanie políčok začína od 0; na začiatku výpočtu sú prázdne.

Nech $\delta = \langle q, r, x, a_1, \ldots, a_{k+1}, q', r', a'_1, \ldots, a'_{k+1}, d_1, \ldots, d_{k+2} \rangle \in \Delta$ je prechod M. Podľa tohto prechodu stroj M, ktorý sa nachádza v pracovnom stave q, komunikačnom stave r, číta zo vstupu symbol x, z i-tej pásky a_i, $i = 1, 2, \ldots, k+1$, v jednom *kroku* dosiahne nový pracovný stav q', nový komunikačný stav r', napíše a'_i na i-tu pásku a posunie každú z $k+2$ hláv, $i = 1, 2, \ldots, k+1$, o jedno políčko v smere d_j (doľava alebo doprava), pre $j = 1, 2, \ldots, k+2$.

Konfigurácia M je prvok $Q \times R \times \Sigma^* \times ((\Gamma - \{\sharp\})^*)^{k+1} \times \mathbb{N}^{k+2}$, ktorý popisuje pracovný a komunikačný stav konečnostavovej riadiacej jednotky, vstup, neprázdne obsahy $k+1$ pások a $k+2$ pozícií hláv.

Hlavová konfigurácia stroja M je prvok $Q \times R \times (\Sigma \cup \{\$\}) \times \Gamma^{k+1}$, ktorý popisuje pracovný a komunikačný stav riadiacej jednotky spolu s obsahmi buniek snímanými jednotlivými hlavami.

Hovoríme, že prechod s novým komunikačným stavom r' *vysiela* stav $r' \in R$. Na prechody kladieme jedno syntaktické obmedzenie, tzv. pravidlo súhlasného

vysielania: *všetky prechody odpovedajúce rovnakej hlavovej konfigurácii vysielajú rovnaký komunikačný stav.* Líšiť sa môžu vo všetkých ostatných položkách – v novom pracovnom stave, v nových symboloch a v posunoch hláv. Ďalej hovoríme, že konfigurácia je *naladená na kanál* c ak obsahom kanálovej pásky naľavo od aktuálnej pozície hlavy je c; snímané políčko nie je súčasťou c. Ak c je neprázdny reťazec, hovoríme mu *číslo kanála/kanálové číslo*. Konfigurácia vysiela r' na kanáli c, ak je to konfigurácia naladená na kanál c, pričom na ňu aplikujeme prechod s novým komunikačným stavom r'.

Konfigurácia β je δ–nasledovník konfigurácie α vzhľadom na prechod $\delta \in \Delta$ (píšeme $\alpha \vdash^{\delta} \beta$), ak β vznikla z α aplikovaním prechodu δ. Prechodu $\alpha \vdash^{\delta} \beta$ hovoríme *jednoduchý krok* M.

Pripomíname, že konfigurácia môže mať viacero rôznych δ–nasledovníkov. Konfigurácia bez nasledovníka je *terminálna konfigurácia*.

Pri definovaní výpočtu stroja M nám pomôže niekoľko ďalších pojmov a definícií. Začneme definíciami projekcií, ktoré extrahujú z konfigurácie jej zložky.

Funkcia $Tuned : Q \times R \times \Sigma^* \times ((\Gamma - \{\sharp\})^*)^{k+1} \times \mathbb{N}^{k+2} \to (\Gamma - \{\sharp\})^*$ priraďuje konfigurácii číslo jej kanála. Miernym zneužitím tejto notácie môžme funkciu prirodzene rozšíriť na neprázdnu množinu konfigurácií $L \neq \emptyset$:

$$Tuned(L) = \begin{cases} c & \text{ak/práve vtedy keď } Tuned(\alpha) = c \text{ pre všetky } \alpha \in L \\ \perp & \text{ak existujú } \alpha, \beta \in L, \alpha \neq \beta \text{ a } Tuned(\alpha) \neq Tuned(\beta) \end{cases}$$

(symbol $\perp$ označuje nedefinovanú hodnotu.)

Analogicky definujeme funkciu $Broadcast : Q \times R \times \Sigma^* \times ((\Gamma - \{\sharp\})^*)^{k+1} \times \mathbb{N}^{k+2} \to R$, ktorá každej konfigurácii priradí stav, vysielaný prechodmi aplikovateľnými na túto konfiguráciu (vďaka pravidlu súhlasného vysielania je tento stav určený jednoznačne). Funkciu opäť rozšírime na neprázdnu množinu konfigurácií $L \neq \emptyset$:

$$Broadcast(L) = \begin{cases} b & \text{ak pre všetky } \alpha, \beta \in L, Tuned(\alpha) = Tuned(\beta) \\ & \text{a } Broadcast(\alpha) = b \\ \perp & \text{inak} \end{cases}$$

Poslednou projekciou definovanou na množine konfigurácií je $Comm : Q \times R \times \Sigma^* \times ((\Gamma - \{\sharp\})^*)^{k+1} \times \mathbb{N}^{k+2} \to R$, ktorá konfigurácii priraďuje jej komunikačný stav.

Nech $u, v \in R$ sú komunikačné stavy a α konfigurácia v komunikačnom stave u. Symbolom $\alpha|_{u:=v}$ označujeme tú konfiguráciu, ktorá vznikla nahradením (komunikačného) stavu u v pôvodnej konfigurácii α (komunikačným) stavom v.

Nech L je množina konfigurácií, $L_c \subseteq L$ podmnožina tých konfigurácií z L, ktoré sú naladené na c a nech $\alpha \vdash^{\delta} \beta$ je jednoduchý krok. Hovoríme, že konfigurácia γ je δ_L–*nasledovník* konfigurácie α vzhľadom na prechod δ *modifikovaný vysielaním* z L (označujeme $\alpha \vdash^{\delta_L} \gamma$,) ak je definovaná nasledovne:

$$\gamma := \begin{cases} \beta|_{Comm(\beta):=b} & \text{ak } L_{Tuned(\beta)} \neq \emptyset \text{ a } Broadcast(L_{Tuned(\beta)}) = b \\ \bot & \text{ak } L_{Tuned(\beta)} \neq \emptyset \text{ a } Broadcast(L_{Tuned(\beta)}) = \bot \\ \beta & \text{ak } L_{Tuned(\beta)} = \emptyset \end{cases}$$

Predchádzajúca konštrukcia popisuje vysielanie konfigurácií z L na tom kanáli, na ktorý je naladená konfigurácia β. Prvý prípad odpovedá korektnému vysielaniu, keď všetky konfigurácie súhlasne vysielajú rovnaký stav b. Druhý prípad popisuje situáciu, keď na sledovanom kanáli dochádza ku konfliktu vo vysielaní a nakoniec tretí prípad popisuje situáciu, keď sa na sledovanom kanáli nevysiela.

Nech ϱ je konfigurácia; *výpočtový strom* $T(\varrho)$ vzhľadom na prechodovú reláciu Δ stroja M je orientovaný, potenciálne nekonečný strom s koreňom ϱ, ktorého vrcholy sú konfigurácie stroja M. Definujeme ho rekurzívne:

1. ϱ je koreň $T(\varrho)$ v hĺbke $d = 0$;
2. Nech C_d je množina konfigurácií v hĺbke $d \geq 0$. Potom pre všetky neterminálne $\alpha \in C_d$ množina C_{d+1} pozostáva zo všetkých δ_{C_d}–nasledovníkov α; t.j. $C_{d+1} = \{\gamma | \exists \text{ neterminálna konfigurácia } \alpha \in C_d : \alpha \vdash^{\delta_{C_d}} \gamma\}$. Ak je niektorý z δ_{C_d}–nasledovníkov α nedefinovaný, potom je celý strom nedefinovaný. V strome $T(\varrho)$ vedie hrana z každého vrchola $\alpha \in C_d$ do každého jeho δ_{C_d}–nasledovníka $\gamma \in C_{d+1}$.

Pripomíname, že podmienka ("úplnosti") na konci bodu 2 zaručuje, že strom $T(\varrho)$ obsahuje pre každý vrchol α v $T(\varrho)$ množinu všetkých jeho nasledovníkov, ktorých komunikačné stavy môžu byť modifikované vysielaním.

Z popisu vidno, že výpočtový strom je vytváraný plne deterministicky: všetky prechody medzi konfiguráciami rovnako ako všetky medzi–procesorové komunikácie sú definované jednoznačne, bez možnosti akejkoľvek voľby.

Majme (potenciálne nekonečnú) cestu v strome $T(\varrho)$, ktorá začína vo vrchole ϱ. Postupnosti konfigurácií na tejto ceste budeme hovoriť *výpočtová cesta* so začiatkom v ϱ. Výpočtová cesta prirodzene reprezentuje výpočet M na tejto ceste, ale tiež *proces* odpovedajúci tejto ceste.

Hovoríme, že výpočtový strom $T(\varrho)$ je *akceptuje vstup* w ak navyše spĺňa nasledujúce podmienky:

1. *Konečnosť:* $T(\varrho)$ je konečný strom;
2. *Počiatočná podmienka:* $\varrho = (q_0, r_0, w, \underbrace{\nu, \ldots, \nu}_{k+1}, \underbrace{0, \ldots, 0}_{k+2})$ je *počiatočná konfigurácia*, ν označuje prázdny reťazec.
3. *Dohoda o akceptovaní:* listy $T(\varrho)$ sú terminálne konfigurácie v rovnakej hĺbke stromu, v komunikačnom stave q_{accept} a naladené na rovnaký kanál.

Z praktického pohľadu dohoda o akceptovaní hovorí, že všetky procesy zdieľajú v rovnakom čase spoločnú informáciu o tom, že vstupné slovo bolo akceptované. Ak nepoznáme čas, v ktorom má výpočet skončiť, tak akceptovanie

v rovnakej hĺbke zabezpečíme napr. pomocou bariérovej synchronizácie. Jej myšlienkou je periodické súčasné testovanie pripravenosti procesov na ukončenie výpočtu (pozri napr. [5]).

Analogicky sa definuje pojem výpočtového stromu zamietajúceho vstup.

Hovoríme, že M *akceptuje* w ak existuje výpočtový strom akceptujúci w. Množinu slov akceptovaných M budeme označovať $L(M)$.

Priestorová (kanálová) zložitosť konfigurácie je súčet dĺžok neprázdnych obsahov pracovných pások (dĺžka neprázdnej časti kanálovej pásky). Priestorová zložitosť výpočtového stromu T je maximum z priestorových zložitostí konfigurácií stromu T; analogicky definujeme kanálovú zložitosť výpočtového stromu T. Časová zložitosť výpočtového stromu T je jeho hĺbka.

Hovoríme, že M *pracuje v priestore* $S(n)$ (*s veľkosťou kanála* $C(n)$), ak pre každý reťazec $w \in L(M)$ dĺžky n existuje výpočtový strom akceptujúci w priestorovej zložitosti nanajvýš $S(n)$ (kanálovej zložitosti nanajvýš $C(n)$). Analogicky, M *pracuje v čase* $T(n)$, ak pre každý reťazec $w \in L(M)$ dĺžky n existuje výpočtový strom akceptujúci w časovej zložitosti nanajvýš $T(n)$.

Rozlišovanie priestorovej a kanálovej zložitosti je motivované bezdrôtovými mobilnými sieťami. Kanálová zložitosť odráža veľkosť príslušného komunikačného mechanizmu.

Na výsledný model sa môžeme pozerať aj ako na Π_1 stroj, ktorý je obohatený o mechanizmus medziprocesorovej komunikácie a má zmenený akceptačný mechanizmus alternujúcich Turingových strojov: kritérium akceptovania musí byť naprogramované v inštrukciách stroja. Vďaka svojej deterministickej povahe predstavuje WPTM do istej miery realistický model bezdrôtových mobilných výpočtov.

Všimnime si, že nahradením prechodovej relácie funkciou v definícii WPTM získame klasický Turingov stroj. Ľahko tiež vidno, že WPTM môže v lineárnom čase simulovať nedeterministický aj Π_1 Turingov stroj. Napriek tomu, že obe zariadenia majú odlišné akceptačné kritériá, WPTM sa ľahko prispôsobí obom. Ukážeme ešte silnejší výsledok: z hľadiska času sú WPTM polynomiálne ekvivalentné alternujúcim Turingovým strojom (a teda aj synchronizovaným Turingovým strojom).

3 Sila bezdrôtovej komunikácie

Začneme porovnaním WPTM s klasickými ATM.

Veta 1 *Nech $T(n)$ je časovo konštruovateľná funkcia, $T(n) \geq n$, A alternujúci Turingov stroj časovej zložitosti $T(n)$. Potom existuje WPTM M, ktorý simuluje A s časovou, priestorovou aj kanálovou zložitosťou $O(T(n))$.*

Náčrt dôkazu: Simuláciu spravíme v dvoch etapách. Najskôr vybudujeme výpočtový strom T_A a potom v jeho listoch spustíme výpočet akceptačného kritéria pre T_A. Kanálovú pásku pritom použijeme na identifikáciu vrcholov v T_A, resp. popis cesty, ktorou sa do nich dostaneme.

Môžeme predpokladať, že každý výpočtový strom T_A stroja A je binárny. M začína svoju prácu v počiatočnej konfigurácii stroja A. Realizuje všetky aplikovateľné prechody A nezávisle od toho, či bola konfigurácia existenčná alebo univerzálna; odpovedá to prechádzaniu T_A po úrovniach zhora nadol. V pracovnej pamäti si však procesy M pamätajú aj tzv. *trasu* — popis výpočtovej cesty v T_A z koreňa do konfigurácie c v hĺbke d, ktorá odpovedá príslušnému procesu v čase d. Vďaka predpokladu môžeme výpočtovú cestu reprezentovať binárnym číslom $(b_1, \ldots, b_d)_2,\ b_i \in \{0, 1\}$; niekedy mu hovoríme aj *číslo cesty* do c.

Súčasťou trasy je aj informácia o type jednotlivých vrcholov T_A(existenčný, univerzálny) na odpovedajúcej ceste. Kvôli druhej etape pridávame aj premennú $qual$, v ktorej budeme uchovávať *kvalitu* jednotlivých vrcholov na ceste. Kvalita vrchola v je prvok z množiny $\{\perp, Y, N\}$, ktorý zodpovedá nedefinovanej, akceptujúcej, resp. zamietajúcej hodnote. Hovorí o podstrome T_v výpočtového stromu T_A s koreňom v. Ak je T_v akceptujúci (zamietajúci) strom, kvalita v je Y (N); inak je kvalita v nedefinovaná, teda $\perp$ (pozri napr. [1]).

Trasa konfigurácie c uloženej vo vrchole u v hĺbke d je teda popísaná $d-$ticou $\langle (b_1, \square_1, qual_1), \ldots, (b_d, \square_d, qual_d)\rangle$, kde $b_i \in \{0, 1\}$, $\square_i \in \{\exists, \forall, |, \}$ a $qual_i \in \{\perp, Y, N\}$. Kvantifikátor $\exists$ zachytáva existenčné vetvenie v u, $\forall$ univerzálne vetvenie (s dvomi nasledovníkmi) a $|$ znamená "žiadne vetvenie– deterministická konfigurácia s jedným nasledovníkom. Popis trasy sa zmestí do priestoru $O(T(n))$.

Po odsimulovaní $T(n)$ krokov ($T(n)$ je časove konštruovateľná funkcia) stroja A začnú procesy M overovať podmienku akceptovania v A. Táto podmienka vyjadruje, ako sa majú skombinovať ödpovede"jednotlivých ukončených procesov, keď stúpame smerom ku koreňu T_A (pozri napr. [1]). Pri realizovaní "stúpania"využijeme informáce z popisu trasy. Postúpiť z úrovne d na úroveň $d-1$ môžme len vtedy, ak majú všetky procesy M "vyplnenú"hodnotu $qual_d$.

Odteraz $S_{\langle b_1,\ldots,b_d\rangle}$ označuje množinu procesov M, ktorých číslo cesty začína $(b_1, \ldots, b_d)_2$. Nezabúdajme, že hodnota $\square_i$ (pre $i = 1, \ldots, d,$) je pre všetky procesy z $S_{\langle b_1,\ldots,b_d\rangle}$ rovnaká.

V každom procese M vypočítame hodnoty $qual_d,\ d = T(n), T(n)-1, \ldots, 1$ rovnakým algoritmom. Algoritmus teda pozostáva z $T(n)$ kôl, pričom na konci každého kola d platí invariant: každý proces z $S_{\langle b_1,\ldots,b_d\rangle}$, má určené všetky hodnoty $qual_i,\ i = T(n), T(n)-1, \ldots, d$ a navyše, hodnota $qual_d$ je vo všetkých procesoch z $S_{\langle b_1,\ldots,b_d\rangle}$ rovnaká. (Nezabúdajme, že všetky procesy WPTM vykonávajú jednotlivé kroky algoritmu samostatne, ale naraz.)

Výpočet algoritmu začína v hĺbke $d = T(n)$; hodnota $qual_d$ sa v každom procese M nastaví na Y, N, alebo $\perp$ podľa toho, či proces akceptoval, zamietal alebo ešte neskončil výpočet. Keďže množiny $S_{\langle b_1,\ldots,b_d\rangle}$ sú pre všetky čísla ciest $b_1, \ldots, b_d$ jednoprvkové, invariant platí.

Predpokladajme, že invariant platí v hĺbke $T(n) \geq d > 1$; popíšeme algoritmus, ktorý zachová jeho platnosť aj pre d rovné $d-1$.

Každý proces z $S_{\langle b_1,\ldots,b_d\rangle}$ skontroluje hodnotu $\square_d$ na svojej trase.

Hodnota "|ödpovedá deterministickému kroku; v tomto prípade proces nastaví $qual_d := qual_{d+1}$.

Hodnota "∀"resp "∃"poukazuje na univerzálne, resp. existenčné vetvenie. V rámci $S_{\langle b_1,\ldots,b_d\rangle} = S$ preto existujú dve "sesterské"podmnožiny procesov $S_{\langle b_1,\ldots,b_d,0\rangle} = S_1$ a $S_{\langle b_1,\ldots,b_d,1\rangle} = S_2$. Množiny S_1 a S_2 sú disjunktné, $S_1 \cup S_2 = S$, všetky procesy v S_i, $i = 1, 2$, majú rovnakú hodnotu $qual^i_{d+1}$. Aby mohol proces z S vypočítať hodnotu $qual_d = qual^1_{d+1} \wedge qual^2_{d+1}$, resp. $qual_d = qual^1_{d+1} \vee qual^2_{d+1}$, potrebuje obe hodnoty $qual^1_{d+1}, qual^2_{d+1}$ poznať. Preto obe množiny (postupne) odvysielajú hodnotu svojho $qual^i_{d+1}$ všetkým ostatným procesom v S. Vysielanie sa uskutoční na kanáli číslo $(b_1 \ldots b_d)_2$: najprv vysielajú procesy z S_1, potom procesy z S_2. Keďže sú hodnoty $qual^i_{d+1}$ v rámci S_1, resp. S_2, rovnaké, ku konfliktu vo vysielaní nedochádza. Každý proces z S má potrebnú informáciu a môže vypočítať kvalitu príslušného vrchola v hĺbke d v súlade s pravidlami vyhodnocovania alternujúcich stromov (pozri napr. [1]). Je zrejmé, že všetky procesy v S vypočítajú rovnakú hodnotu $qual_d$. Všimnime si, že pre rôzne množiny $S_{\langle b_1,\ldots,b_d\rangle}$ prebieha potrebná komunikácia na rôznych kanáloch. Nastavením hodnoty $d := d - 1$ ďalšie kolo algoritmu končí.

Takto vyhodnocujeme jednu úroveň za druhou až kým pre $d = 1$ nedosiahne $qual_1$ vo všetkých procesoch svoju výslednú hodnotu. Vysielaním tejto hodnoty na kanáli číslo 0 sa všetky procesy dostanú do rovnakého komunikačného stavu q_{accept} alebo q_{reject} a simulácia končí.

Ľahko vidno, že overenie podmienky akceptovania pre A realizuje M v $O(T(n))$ krokoch.

□

Veta 2 *Nech $T(n)$ je časovo konštruovateľná funkcia, $T(n) \geq n$, a M WPTM časovej zložitosti $T(n)$. Potom existuje alternujúci Turingov stroj A, ktorý simuluje M v časovej aj priestorovej zložitosti $O(T^4(n))$.*

Náčrt dôkazu: Namiesto priamej simulácie M alternujúcim Turingovým strojom A ukážeme simuláciu M nedeterministickým Turingovým strojom N v priestore $O(T^2(n))$. Z toho na základe známych výsledkov o alternujúcich Turingových strojoch (napr. [1], Veta 3.1) vyplynie požadovaný výsledok. Navrhneme nedeterministický algoritmus, ktorý odsimuluje M v priestore $O(T^2(n))$.

Nech T_M je výpočtový strom stroja M so vstupom w. Popíšeme priestorovo efektívny nedeterministický peblovací algoritmus (stroj N) ktorý navštívi každý vrchol T_M. Uvedomme si, že peblovanie môžme použiť vďaka deterministickej povahe výpočtu M. V takých prípadoch je peblovanie jednou z možných stratégií modelovania priestorovo efektívneho prepočitávania prechádzaných konfigurácií (napr. [4]).

T_M je superpozíciou dvoch grafov: prvý z nich je binárny strom —budeme mu hovoriť *strom nasledovníkov*, ktorý znázorňuje históriu vzniku jednotlivých procesov. Druhý, tzv. *graf vysielania*, zachytáva vysielanie komunikačných stavov z konfigurácií do odpovedajúcich prijímajúcich konfigurácií.

Musíme zabezpečiť, že algoritmus navštívi každý vrchol T_M a v rámci tejto návštevy overiť, že príslušná konfigurácia bola vypočítaná správne. Tiež treba skontrolovať, že v T_M nedošlo k vysielacím konfliktom a že je splnená podmienka akceptovania/zamietania vstupu. K dosiahnutiu stanovených cieľov navrhneme nedeterministickú peblovaciu stratégiu.

Peblovanie pozostáva z dvoch častí: cieľom prvej fázy je položiť kameň na každý list T_M, v druhej časti sa kontroluje bezkonfliktnosť vysielania v T_M.

V prvej fáze kladieme kamene na vrcholy T_M podľa nasledujúcich pravidiel. Kameň sa vždy dá položiť na koreň T_M. Kameň vždy môžeme odstrániť z ľubovoľného vrchola. Kameň sa dá položiť na vrchol, ak je kameň položený na jeho bezprostrednom predchodcovi v strome nasledovníkov; v prípade, že je do pokrývaného vrchola vysielaný komunikačný stav, musí navyše obsahovať kameň aj jeden z jeho predchodcov v grafe vysielania. Cieľom tejto fázy je pokryť každý list T_M.

V druhej fáze skontrolujeme, že pri vysielaní nedošlo v T_M ku konfliktom. Spravíme to takto: pre všetky obsahy kanálovej pásky dĺžky $c : 0 \leq c \leq T(n)$ a pre každú hĺbku overíme, či všetky vysielajúce konfigurácie v T_M naladené na c vysielajú rovnaký komunikačný stav. Cieľom hry v tejto fáze je prechádzať stromom T_M jednu úroveň za druhou a podľa vyššie naformulovaných pravidiel postupne pokrývať všetky ich vrcholy.

Jedine v prípade, že prvá fáza ohlásila akceptujúci stav v každej terminálnej konfigurácii a všetky tieto konfigurácie sú naladené na rovnaký kanál a v priebehu druhej fázy sa nezistil žiaden vysielací konflikt, dochádza k tomu, že N akceptuje w.

Nedeterministická peblovacia stratégia je kombináciou dvoch algoritmov: deterministického, ktorý systematicky prechádza stromom nasledovníkov a nedeterministického, ktorý kladie kamene na vrcholy stromu s cieľom položiť kameň na konkrétny vrchol v T_M, ktorý bol určený deterministickým algoritmom. Kroky nedeterministického peblovacieho algoritmu sa generujú on–line, nie je potrebné si ich pamätať.

V prvej fáze prechádzame strom nasledovníkov prehľadávaním do hĺbky, v druhej zas prehľadávaním do šírky. Ak v tomto prehľadávaní neboli navštívené všetky vrcholy, prehľadávací algoritmus jednoznačne určí nový ďalší nenavštívený vrchol v, ktorý sa potom musí pokryť nedeterministicky.

Odhadnime počet kameňov, postačujúcich na realizáciu popísanej hry. Aby sme položili kameň na vrchol v stromu T_M v hĺbke d, musíme položiť kameň na dvoch jeho priamych predchodcov. Prvým je vrchol v_1 v strome nasledovníkov, z ktorého, s výnimkou komunikačného stavu, vieme vypočítať konfiguráciu uloženú vo vrchole v. Tento chýbajúci stav môžme doplniť prostredníctvom druhého vrchola v_2 v grafe vysielania. Oba vrcholy v_1 a v_2 ležia v hĺbke $(d-1)$. Označme $P(d)$ počet kameňov postačujúcich na pokrytie vrchola v stromu T_M v hĺbke d. Najprv — použitím $P(d-1)$ kameňov — pokryjeme vrchol v_1, necháme kameň na v_1 a odstránime (pre ďalšie použitie) $P(d-1)-1$ kameňov použitých pri pokrývaní v_1. Potom pokryjeme pomocou $P(d-1)$ kameňov vrchol v_2. Tentokrát však môžeme využiť $P(d-1)-1$ kameňov, ktoré sme uvoľnili po pokrytí vrchola v_1. Na pokrytie v_2 teda stačí jeden kameň navyše. Keďže jeden kameň je umiestnený na v_1, $P(d) \leq P(d-1)+1$ a $P(1)=1$, platí $P(d)=d$. Preto $T(n)$ kameňov v každej fáze pokrývacej hry stačí.

Keďže kameň v pokrývacej hre popísaného algoritmu je v skutočnosti konfigurácia M veľkosti $O(T(n))$, je priestorová zložitosť popísaného nedeterministického simulačného algoritmu $O(T^2(n))$.

□

V nasledujúcej časti sa budeme venovať zaradeniu WPTM do hierarchie známych zložitostných tried. Navyše k štandardným deterministickým triedam ako LOGSPACE, PSPACE a EXPTIME budeme používať ich analógie pre ATM, resp. WPTM. Označenie príslušných zložitostných tried vznikne pridaním prefixu A resp. W k už spomínaným triedam.

Dôsledok 1 *Pre každú konštruovateľnú funkciu* $T(n)$, $T(n) \geq n$,

$$\bigcup_{k>0} \mathsf{WTIME}(\mathsf{T}^k(\mathsf{n})) = \bigcup_{k>0} \mathsf{ATIME}(\mathsf{T}^k(\mathsf{n}))$$

t.j.,

$$\mathsf{WPTIME} = \mathsf{APTIME} = \mathsf{PSPACE}$$

Ďalej sa budeme zaoberať vzťahom medzi deterministickým a bezdrôtovým mobilným priestorom.

Veta 3 *Nech D je deterministický Turingov stroj priestorovej zložitosti $S(n)$. Potom D sa dá simulovať WPTM M kanálovej zložitosti $O(\log S(n))$ a priestorovej zložitosti $O(1)$.*

Náčrt dôkazu: Dôkaz postupuje podobne ako dôkazy podobných tvrdení o synchronizovanom alternovaní (napr. [2], [6]). Obsahy jednotlivých buniek jedinej pracovnej pásky DTM D sa udržiavajú v procesoch M. Vždy, keď hlava D dosiahne na pracovnej páske prázdny symbol, vytvorí M nový proces. Procesy sú číslované pozíciou bunky, aktuálny obsah ktorej udržiavajú. Na svojej pracovnej páske si teda proces číslo i pamätá symbol zapísaný v i-tej bunke, prítomnosť hlavy D na nej (yes / no), a stav stroja D v prípade, ak hlava sníma i–tu bunku. Číslo i (zapísané binárne) si i–ty proces uchováva na svojej kanálovej páske. Simulácia jedného kroku výpočtu D znamená aktualizáciu informácie (obsah, resp. stav) v tých procesoch M, ktoré odpovedajú bunkám zúčastneným v tomto kroku. "Pohyb hlavy"z i–teho procesu (bunky) sa realizuje vyslaním príslušnej "správy"ľavému, resp. pravému susedovi bunky na kanáli, ktorého číslo sa zhoduje s indexom príslušnej bunky — $(i-1)$ resp. $(i+1)$. Preladenie kanála teda spočíva v pripočítaní alebo odpočítaní 1 od aktuálnej hodnoty čísla kanála. Kanálová zložitosť je zrejme $O(\log S(n))$, priestorová $O(1)$.

□

Veta 4 *Nech $S(n) \geq \log n$, M WPTM priestorovej a kanálovej zložitosti $O(S(n))$. Potom existuje $c > 0$ také, že M sa dá simulovať deterministickým Turingovým strojom D v priestore $O(c^{S(n)})$.*

Náčrt dôkazu: Majme výpočtový strom T_M WPTM M so vstupom w. Ak M akceptuje w, tak M môže mať nanajvýš $O(c_1^{S(n)})$ rôznych procesov. Keďže priestorová zložitosť každého z nich je ohraničená $O(S(n))$ (uvedený priestor je postačujúci aj pre pozíciu hlavy na vstupe), tak pre vhodnú konštantu $c > c_1$ má D dosť miesta na to, aby zapísal všetky potrebné konfigurácie a krok za krokom simuloval výpočet M. D akceptuje, ak jeho výpočet odpovedá výpočtovému stromu M akceptujúcemu w.

□

Dôsledok 2 *Pre každú funkciu* $S(n) \geq \log n$

$$\mathsf{WSPACE(S(n))} = \bigcup_{\mathsf{c}>0} \mathsf{DSPACE(c^{S(n)})}$$

$$\mathsf{WLOGSPACE} = \mathsf{PSPACE} = \mathsf{WPTIME},\ \mathsf{WPSPACE} = \mathsf{EXPSPACE} = \mathsf{WEXPTIME}$$

Dôsledky 1 a 2 ukazujú, že základné zložitostné triedy — logaritmický priestor a polynomiálny čas — sú pre synchronizované alternujúce Turingove stroje (podľa [6] alebo [3]) a sieť Turingových strojov totožné. Zvlášť si všimnime, že podobne ako synchronizované Turingove stroje, WPTS používa svoj priestor optimálnym spôsobom — napr. logaritmický "bezdrôtový"priestor je ekvivalentný "bezdrôtovému"polynomickému času. Vzhľadom na základné triedy klasickej priestorovej zložitosti sú teda obidva modely ekvivalentné.

4 Záver

Okrem charakterizácie výpočtovej sily bezdrôtových sietí sme získali aj nový pohľad na podstatu klasického výpočtového zdroja — alternácie, resp. synchronizovanej alternácie. Hoci (synchronizované) alternujúce stroje so svojím neobmedzeným využívaním nedeterminizmu a zvláštnym mechanizmom akceptovania pôsobia ako pomerne nerealistický výpočtový model, získané výsledky ukazujú, že sú vlastne ekvivalentné deterministickým paralelným zariadeniam so schopnosťou šírenia správ na neobmedzenom počte rôznych kanálov. Tieto výsledky stavajú bezdrôtové mobilné siete medzi veľmi silné výpočtové zariadenia.

Referencie

1. Chandra A., Kozen D., Stockmeyer L., Alternation. Journal of the ACM 28, 1981, 114–133
2. Geffert V., A Communication Hierarchy of Parallel Computations. Theor. Comput. Sci. 198, No.1–2, 1998, 99–130
3. Hromkovič J., Karhumäki J., Rovan B., Slobodová A.: On the Power of Synchronization in Parallel Computations. Discrete Appl. Math. 32, 1991, 155–182
4. Hopcroft J. E., Paul J.W., Valiant L., On Time Versus Space. Journal of ACM 15, 1968, 414–427

5. Valiant L.G., A Bridging Model for Parallel Computation, Communication of the ACM 33, No. 8, 1990, 103–111
6. Wiedermann J., On the Power of Synchronization. Elektronische Informationsverarbeitung und Kybernetik 25(10), 1989, 499–506

O roztržitosti a volnosti slovosledu pomocí restartovacích automatů*

Martin Plátek[1], František Mráz[1,2], Friedrich Otto[2], and Markéta Lopatková[1]

[1] MFF UK Praha, Malostranské nám. 25, 118 00 Praha 1, Česká republika
martin.platek@mff.cuni.cz, mraz@ksvi.mff.cuni.cz,
lopatkova@ufal.mff.cuni.cz
[2] Fachbereich Mathematik/Informatik, Universität Kassel, 341 09 Kassel, Německo
otto@flower.theory.informatik.uni-kassel.de

Abstrakt Zavádíme nový typ restartovacích automatů, abychom získali prostředek umožňující klasifikaci jevů souvisejících s valencí a slovosledem u přirozených jazyků. Studujeme dva základní typy omezení výpočtů: *j-roztržitost* interpretujeme jako míru nesouvislosti valencí (redukcí) a *j-volnost* jako míru volnosti slovosledu. Ukazujeme, že roztržitost lze měřit stejně u jazyků s volným slovosledem (vysokým stupněm volnosti) jako u jazyků s pevným slovosledem (nízkým stupněm volnosti). Přínos vidíme i v exaktním rozlišení mezi roztržitostí a volností slovosledu.

1 Úvod

Z teoretického pohledu jsou restartovací automaty flexibilním zobecněním analytických gramatik (např. frázových, kategoriálních, čistých a závislostních gramatik). Základní motivací pro studium restartovacích automatů je modelování redukční analýzy vět přirozených jazyků. Každý restartovací automat definuje *syntaktický systém*, který je formalizací redukční analýzy [5]. Syntaktické systémy nesou přímo informaci o slovníku (abecedě) vstupních slov (symbolů), seznamu (slovníku) kategorií, jazyce větných forem (charakteristickém jazyce), systému redukcí a zprostředkovaně i o vstupním (povrchovém, terminálním) jazyce. Ve schopnosti rozlišovat a popisovat syntaktické systémy přirozených jazyků se projevuje lingvistická kompetence mluvčího. V článku navrhujeme nový typ restartovacích automatů umožňující adekvátní klasifikaci jevů souvisejících s valencí a slovosledem v přirozených jazycích (více viz 1.1).

V technické části ukazujeme nekonečnou škálu tříd syntaktických systémů a tříd jazyků definovanou pomocí restartovacích automatů a vhodných omezení jejich výpočtů. Naším hlavním cílem je formalizovat stupně volnosti slovosledu. Podobnou problematiku jsme řešili již dříve [3], [4]. Využívali jsme závislostní gramatiky a od nich odvozené pojmy. Restartovací automaty umožňují univerzálnější postup. Tento postup může být uplatněn (nepřímo) na různé typy

* První, druhý autor a čtvrtá autorka byli částečně podporovaní GA ČR, grant č. 201/04/2102 a programem 'Informační společnost', projekt č. 1ET100300517. Druhý a třetí autor byli podporování grantem Deutsche Forschungsgemeinschaft.

gramatik, např. na frázové gramatiky Chomského typu, kategoriální, Marcusovy, čisté i závislostní gramatiky. Jim odpovídající analyzátory lze přímočaře konstruovat jako restartovací automaty. Syntaktické systémy takto získané dovolují sílu těchto prostředků porovnávat (z lingvistického pohledu) výrazně adekvátněji než jen pomocí prosté generativní síly.

Omezujeme se na automaty, jejichž výpočetní síla není větší než lineárně omezené automaty. *Volně restartující automat*, krátce FRR-automat, je zařízení s jednou (pružnou) páskou, konečnou řídící jednotkou a pracovním oknem pevné velikosti. Každý výpočet je složen z posloupnosti *cyklů* a je uzavřen *koncovkou*. Během cyklu dochází ke zjednodušení obsahu pásky přepisováním. Počet přepsání v cyklu není v základním modelu omezen.

V práci jsou zavedeny a studovány dva základní typy omezení výpočtů. Tzv. *j-roztržitost* interpretujeme jako míru složitosti slovosledu, *j-volnost* interpretujeme jako míru volnosti slovosledu. Ukážeme, že roztržitost lze měřit stejně u jazyků s volným slovosedem (vysokým stupněm volnosti) jako u jazyků s pevným slovosledem (nízkým stupněm volnosti). Důležitým terminologickým přínosem práce je exaktní rozlišení mezi složitostí (roztržitostí) a volností slovosledu v jazykových jevech.

1.1 Lingvistická motivace

V této podsekci přímo navazujeme na text o redukční analýze [5]. Ilustrujme složitost slovosledu na dobře známé holandské konstrukci (příklad 1) s valenčním slovesným doplněním v infinitivu (tuto konstrukci nelze adekvátně popsat kontextovými gramatikami [7]). Holandština je považována za jazyk s pevným slovosledem, kde pozice jednotlivých větných členů je pevně dána, tj. syntaktické vztahy jsou určeny slovosledem.

Příklad 1 *Ik denk dat ik Cecilia de nijlpaarden zag voeren.*
(já-myslím-že-já-Cecilia-(člen)-hroši-viděl-krmit)
(Myslím, že jsem viděl Cecilii krmit hrochy.)

Obrázek 1. *Ik denk dat ik Cecilia* ***de nijlpaarden*** *zag* ***voeren****.*
$\longrightarrow$ *Ik denk* ***dat ik Cecilia zag*** $\mathbf{VG}_{inf}$*.*
$\longrightarrow$ *Ik denk* $\mathbf{VG}_{dat}$*.*

Pokud chceme tuto větu redukovat, postupujeme způsobem zachyceným na obr. 1 (redukovaná slova jsou zvýrazněna). V prvním kroku jsou zpracována slova *de* (člen), *nijlpaarden* (hroši) a *voeren* (krmit) (jde o sloveso a jeho valenční doplnění; povšimněme si, že přepisovaná slova netvoří 'spojitou komponentu'). Tato slova jsou přepsána slovem VG_{inf}, které patří do seznamu kategorií (zachycuje skutečnost, že byla vypuštěna infinitivní konstrukce). Ve druhém kroku redukce pokračuje přepsáním slov *dat* (že), *ik* (já), *Cecilia*, *zag* (viděl) a VG_{inf} kategoriálním slovem VG_{dat} (označuje vypuštěnou závislou klauzi uvozenou spojkou *dat* (že); tato závislá klauze je opět tvořena slovesem a jeho valenčními doplněními).

Podobné konstrukce, kde je adekvátní redukovat v jednom kroku slova, která se nacházejí na 'různých místech' věty, jsou poměrně běžné například i v češtině (příklad 2), která nám slouží jako zástupce jazyků s volným slovosledem – syntaktické vztahy jsou zde dány převážně jinými prostředky než pozicí ve větě (v češtině zejména bohatou flexí).

Příklad 2 *Petr na otázku nenašel odpověď.*

Obrázek 2. *Petr* `na otázku` *nenašel* `odpověď`.
$\longrightarrow$ *Petr nenašel* NG_{acc}.

Redukce této věty je znázorněna na obr. 2. V redukčním kroku jsou přepsána slova *na, otázku* a *odpověď* kategoriálním slovem NG_{acc} (zachycuje skutečnost, že byla vypuštěna jmenná skupina realizující valenční doplnění slovesa, jejíž 'hlavu' tvoří podstatné jméno v akuzativu; tato jmenná skupina je tvořena substantivem a jeho valenčním doplněním). Stejně bychom mohli postupovat pro skoro všechny permutace této věty.

Poznamenejme zde, že v [4] jsou ukázány české konstrukce, kde je vhodné redukovat větu na více místech. Počet přepisovacích pozic je v čekých větách v principu omezen počtem povinných valenčních pozic sloves a substantiv.

V sekci 3 se budeme věnovat dvěma typům formálních jazyků, které reprezentují jednak jazyky s valencemi modelovanými pomocí redukcí na pevně určených pozicích, jednak jazyky s valencemi modelovanými pomocí redukcí symbolů na proměnlivých pozicích, tedy nezávisle na slovosledu.

2 Definice

Pojmy budeme zavádět nepříliš formálně. Spoléháme přitom u čtenáře na jisté zkušenosti z teorie formálních jazyků.[3]

Volně restartující automat, krátce FRR-automat, je (nedeterministický) automat M s konečnou řídící jednotkou reprezentovanou konečnou množinou stavů Q a konečnou přechododovou relací (tabulkou) T, konečným vstupní slovníkem I, konečným kategoriálním slovníkem (abecedou) C a pracovním oknem velikosti k, kde $k \geq 1$. M pracuje na (pružné) pásce s levou zarážkou ¢ a pravou zarážkou \$. Pro vstupní větu $w \in I^*$ je vstupním zápisem na pásce ¢w\$. M začíná výpočet ve svém startovním stavu q_0 s oknem na začátku pásky, ve výhledu s levou zarážkou ¢ a prvními $k-1$ symboly w. Řízen svojí přechodovou tabulkou, M na základě aktuáního stavu q a obsahu okna u může provádět následující typy kroků:

(a) *posun doprava* – posune okno o jednu pozici doprava a změní stav,

[3] Abychom předcházeli nedorozuměním v lingvistické interpretaci navrhovaných pojmů, používáme často původní Chomského terminologii. Tedy např. *slovník* na místě abecedy, *věta* na místě slova.

(b) *přepisovací krok* – nahradí obsah okna u řetězem v, změní stav a posune okno na pravého souseda řetězu v. Navíc požadujeme, aby v bylo *zjednodušením* u, tedy aby obsahovalo méně vstupních symbolů než u, nebo aby obsahovalo stejný počet vstupních symbolů jako u, ale bylo kratší než u.
(c) na pravém konci pásky se M buď zastaví a ohlásí *přijmutí*, nebo se zastaví a ohlásí *odmítnutí*, nebo *restartuje*, což znamená, že přemístí okno na levý konec pásky a přepne se znovu do startovního stavu.

Konfigurace automatu M je řetěz $\alpha q \beta$ kde $q \in Q$, a buď $\alpha = \lambda$ a $\beta \in \{¢\} \cdot (I \cup C)^* \cdot \{\$\}$, nebo $\alpha \in \{¢\} \cdot (I \cup C)^*$ a $\beta \in (I \cup C)^* \cdot \{\$\}$; q reprezentuje momentální stav, $\alpha\beta$ je momentální obsah pásky, a je konvencí, že okno obsahuje prvních k symbolů β, nebo celé β, pokud $|\beta| \leq k$. *Restartovací konfigurace* je tvaru $q_0 ¢ w \$$, kde $w \in (I \cup C)^*$.

Libovolný výpočet M sestává z fází. Fáze, zvaná *cyklus*, začíná restartovací konfigurací, hlava se pohybuje doprava prováděním posunů doprava a přepisovacími kroky do chvíle, kdy je proveden restart, a tudíž nastane nová restartovací konfigurace. V případě, že fáze nekončí restartem, nazýváme ji *koncovkou*, neboť nutně končí zastavením automatu. Požadujeme, aby každý cyklus automatu M provedl alespoň jedno přepsání – tedy každá nová fáze začíná na jednodušším řetězu než ta předchozí. Používáme notaci $u \vdash_M^c v$ pro označení cyklu M, který začíná v restartovací konfiguraci $q_0 ¢ u \$$ a končí restartovací konfigurací $q_0 ¢ v \$$; relace $\vdash_M^{c^*}$ je reflexivním a transitivním uzávěrem $\vdash_M^c$. Pár $\mathsf{RS}(M) := ((I \cup C)^*, \vdash_M^c)$ je nazýván *redukčním systémem* indukovaným M.

Větná forma $w \in (I \cup C)^*$ *je přijímána pomocí* M, pokud existuje přijímající výpočet startující z $q_0 ¢ w \$$. Jako $L_{IC}(M)$ označujeme jazyk větných forem přijímaných (rozpoznávaných) pomocí M. Říkáme, že $L_{IC}(M)$ je *charakteristický jazyk* rozpoznávaný M. Jazyk $L_I(M) := L_{IC}(M) \cap I^*$ všech vstupních vět přijímaných pomocí M je *vstupní jazyk rozpoznávaný* M. Čtveřici $BS(M) := (I, C, L_{IC}(M), \mathsf{RS}(M))$ nazýváme *syntaktickým systémem* automatu M. Syntaktické systémy reprezentují lingvistickou kompetenci, tedy znalost větné stavby, odpovědnost za adekvátně navržený systém (zjednoznačňujících) kategorií a umění interpretovat jejich souvislost s větnou stavbou. Větná stavba je v syntaktických systémech reprezentována (hlavně) redukčním systémem. Povšimněme si, že $BS(M)$ obsahuje úplnou informaci o $L_I(M)$, $L_{IC}(M)$ a $\mathsf{RS}(M)$ a že $L_{IC}(M)$ přebírá významnější roli mezi jazyky reprezentovanými syntaktickým systémem. Poznamenejme ještě, že zcela analogicky lze zavést syntaktické systémy pro Chomského frázové grammatiky i pro nejrůznější typy kategoriálních, závislostních či jiných formálních gramatik. Tato skutečnost umožňuje porovnávat ('explikativní') sílu různých typů gramatik či analyzátorů pomocí syntaktických systémů.

Následující vlastnost FRR-automatů využíváme často implicitně v důkazech. Její zásluhou jsou FRR-automaty vhodným modelem redukční analýzy.

Fakt 1. (Zachovávání správnosti). *Nechť* M *je* FRR*-automat a nechť* $u, v \in (I \cup C)^*$. *Pokud* $u \vdash_M^{c^*} v$ *je částí přijímacího výpočtu* M, *potom* $v \in L_{IC}(M)$.

Dále se budeme věnovat omezením relevantním pro naše výpočty. Nejprve zavedeme pojem *monotonie*. Nechť $C := \alpha q \beta$ je *přepisovací konfigurace*, tedy konfigurace, v níž lze uplatnit přepisovací instrukci. Potom $|\beta|$ je *pravou distancí* C, kterou označíme $D_r(C)$. *Posloupnost přepisovacích konfigurací* $S = (C_1, C_2, \ldots, C_n)$ nazýváme *monotonní*, pokud $D_r(C_1) \geq D_r(C_2) \geq \ldots \geq D_r(C_n)$.

Nechť j je přirozené číslo. Budeme říkat, že posloupnost přepisovacích konfigurací $S = (C_1, C_2, \ldots, C_n)$ je *j-monotonní*, pokud existuje rozklad S do j podposloupností

$$S_1 = (C_{1,1}, C_{1,2}, \cdots, C_{1,p_1}), \ldots, S_j = (C_{j,1}, C_{j,2}, \cdots, C_{j,p_j}),$$

takový, že každá S_i, $1 \leq i \leq j$ je monotonní. Povšimněme si, že posloupnost přepisovacích konfigurací $(C_1, C_2, \cdots, C_n)$ *není* j-monotonní právě tehdy, když existují indexy $1 \leq i_1 < i_2 < ... < i_{j+1} \leq n$ takové, že $D_r(C_{i_1}) < D_r(C_{i_2}) < \ldots < D_r(C_{i_{j+1}})$.

Výpočet FRR-automatu M je nazýván *j-monotonní*, pokud odpovídající posloupnost přepisovacích konfigurací je j-monotonní. *Výpočet* je *j-přepisující*, pokud žádný z jeho cyklů neobsahuje více než j přepisovacích kroků. Nakonec, *výpočet* je *j-roztržitý* pokud je jak j-přepisující, tak j-monotonní.

Notace. Pro libovolnou třídu A automatů a libovolný index $Y \in \{I, IC\}$ bude $\mathcal{L}_Y(\mathsf{A})$ označovat třídu Y-jazyků rozpoznatelných automaty z A a $\mathcal{BS}(\mathsf{A})$ bude označovat třídu syntaktických systémů určených A. Jako (D)CFL budeme označovat třídu (deterministických) bezkontextových jazyků, pomocí $|w|_b$ počet výskytů symbolu b v řetězu w a jako $\subset$ budeme označovat vlastní inkluzi. Občas budeme užívat regulární výrazy místo regulárních jazyků, λ značí prázdnou větu a $\mathbb{N}_+$ značí množinu kladných celých čísel.

3 Vlastnosti slovosledu

V této sekci studujeme omezení, která reprezentují složitost a volnost slovosledu formálních jazyků, nebo přesněji jejich syntaktických systémů. Volnost slovosledu syntaktického systému znamená odolnost proti permutacím jejich větných forem a proti redukcím. Zaměříme se hlavně na studium syntaktických systémů a charakteristických jazyků, abychom zdůraznili jejich význam pro modelování lingvistické kompetence.

Definice 2. *Pro* FRR*-automat* M, $i \in \mathbb{N}_+$ *a* $Y \in \{I, IC\}$ *zavádíme:*

$L_Y(M, i) := \{\, w \in L_Y(M) \mid M$ *přijímá* w *pomocí* i*-roztržitého výpočtu* $\}$,
$BS(M, i) := (I, C, L_{IC}(M, i), \mathsf{RS}(M))$, *pokud* $BS(M) = (I, C, L_{IC}(M), \mathsf{RS}(M))$.

Zřejmě platí, že pro každý FRR-automat M, každé $Y \in \{I, IC\}$ a každé $i \geq 1$ $L_Y(M, i) \subseteq L_Y(M, i+1)$ a $BS(M, i) \subseteq BS(M, i+1)$.

Definice 3. *Nechť $i \in \mathbb{N}_+$, $Y \in \{I, IC\}$ a $\mathcal{A}$ označuje (pod)typ* FRR*-automatu. Budeme značit:*

$$\mathcal{L}_Y(i, \mathcal{A}) := \{\, L_Y(M, i) \mid M \textit{ je } \mathcal{A}\textit{-automat} \,\},$$
$$\mathcal{BS}(i, \mathcal{A}) := \{\, BS(M, i) \mid M \textit{ je } \mathcal{A}\textit{-automat} \,\}.$$

Následující věta vznikla pouhým přeformulováním výsledků z [6].

Věta 1. $\mathsf{DCFL} \subset \mathcal{L}_{IC}(1, \mathsf{FRR}) \subset \mathsf{CFL} = \mathcal{L}_I(1, \mathsf{FRR})$.

Na tomto místě je užitečné si uvědomit, že $\mathcal{BS}(1, \mathsf{FRR})$ obsahuje jako vlastní podtřídy třídy syntaktických systémů bezkontextových grammatik (bez vypouštěcích pravidel) či (standardních) kategoriálních grammatik.

Nyní (z praktických důvodů) zavedeme omezený typ FRR-automatu, který budeme označovat jako MRR-*automat*. Děláme to proto, že přechodová relace (tabulka) MRR-automatu M se zadává pomocí konečné posloupnosti *meta-instrukcí* tvaru

$$(E_1, u_1 \to v_1, E_2, u_2 \to v_2, E_3, \ldots, E_i, u_i \to v_i, E_{i+1}),$$

kde $E_1, \ldots, E_{i+1}$ jsou regulární výrazy a pro každé $n = 1, \ldots, i$, $u_n, v_n \in (I \cup C)^*$ jsou věty takové, že v_n je zjednodušením u_n. Pravidla $u_1 \to v_1$, $u_2 \to v_2$, …, $u_i \to v_i$ ztělesňují přepisovací kroky M. Při pokusu o provedení této meta-instrukce z konfigurace $q_0 ¢ w \$$ M neuspěje (a tedy skončí odmítnutím), pokud nebude w umožňovat faktorizaci tvaru $w = w_1 u_1 w_2 u_2 \ldots w_i u_i w_{i+1}$ takovou, že $¢w_1 \in L(E_1)$, $w_2 \in L(E_2)$, …, $w_{i+1}\$ \in L(E_{i+1})$, kde $L(E_n)$ označuje jazyk popsaný regulárním výrazem E_n. Na druhou stranu, pokud w má nějaké faktorizace tohoto tvaru, pak jedna taková faktorizace je vybrána nedeterministicky a $q_0 ¢ w \$$ je transformována do $q_0 ¢ w_1 v_1 w_2 v_2 w_3 \ldots w_i v_i w_{i+1} \$$. Abychom popsali koncovky přijímacích výpočtů M, používáme meta-instrukce tvaru $(¢ \cdot E \cdot \$, \mathsf{Accept})$, které přijímají věty z regulárního jazyka $L(E)$.

Pro každý MRR-automat M existuje číslo j_M takové, že žádný cyklus libovolného výpočtu M neobsahuje více než j_M přepisovacích kroků, to jest, MRR-automaty mají *omezené přepisování*. V tomto smyslu jsou MRR-automaty slabší než FRR-automaty.

Příklad 3. *Nechť M_1 je* MRR*-automat s $\{a, b\}$ jako vstupním i kategoriálním slovníkem (abecedou), který je zadán následujícími třemi meta-instrukcemi:*

(1) $(¢ \cdot a^+, abb \to b, b^* \cdot \$)$,
(2) $(¢ \cdot a^+, abb \to b, b^* \cdot a^+, abb \to b, b^* \cdot \$)$,
(3) $(¢ \cdot (ab + abab) \cdot \$, \mathsf{Accept})$.

Snadno nahlédneme, že $L_{IC}(M_1, 1) = \{\, a^n b^n \mid n > 0 \,\} \cup \{abab\}$, $L_{IC}(M_1, 2) = L_{IC}(M_1, 1) \cup \{\, a^n b^n a^n b^n \mid n > 0 \,\}$ a $L_{IC}(M_1) = L_{IC}(M_1, 2) \supset L_{IC}(M_1, 1)$.

Poznámka 1. $L_{IC}(M_1) \notin \mathsf{CFL}$ *a podle Věty 1 platí, že* $L_{IC}(M_1) \notin \mathcal{L}_{IC}(1, \mathsf{FRR})$.

Příklad 4. *Nechť M_2 je* MRR*-automat se stejnými slovníky jako M_1, který je zadán pomocí meta-instrukcí (1)–(3) z M_1 a následujících dvou meta-instrukcí:*

$$(4)\ (¢ \cdot a^+, abb \to b, b^* \cdot ab \cdot \$), \qquad (5)\ (¢ \cdot ab \cdot a^+, abb \to b, b^* \cdot \$).$$

Platí, že $L_{IC}(M_2, 1) = \{\, a^n b^n, aba^n b^n, a^n b^n ab \mid n > 0 \,\}$, $L_{IC}(M_2, 2) = \{\, a^n b^n a^m b^m \mid n > 0,\ m \geq 0 \,\}$ a $L_{IC}(M_2) = L_{IC}(M_2, 2)$.

Oproti $L_{IC}(M_1)$ je jazyk $L_{IC}(M_2)$ bezkontextový a $L_{IC}(M_2)$ dokonce může být rozpoznáván pomocí MRR-automatu M' takového, že využívá pouze 1-roztržitých (tedy neroztržitých) výpočtů, tudíž $L_{IC}(M_2) \in \mathcal{L}_{IC}(1, \mathsf{FRR})$. M' je zadán pomocí následujících meta-instrukcí

$$(1)\ (¢ \cdot a^+, abb \to b, b^* \cdot \$), \qquad (3)\ (¢ \cdot a^+, abb \to b, b^* \cdot a^+ \cdot b^+ \cdot \$),$$
$$(2)\ (¢ \cdot ab \to \lambda, a^+ \cdot b^+ \cdot \$), \qquad (4)\ (¢ \cdot ab \cdot \$,\ \mathsf{Accept}).$$

Dále budeme využívat následující příklady jazyků, kde $j \in \mathbb{N}_+$:

$$LC_j := \{\, c_0 w c_1 w \dots c_i w \dots c_{j-2} w c_{j-1} w \mid w \in \{a, b\}^* \,\},$$
$$LE_j := \{\, w \in \{a_1, ..., a_j\}^* \mid |w|_{a_1} = |w|_{a_2} = ... = |w|_{a_j} \,\}.$$

Poznamenejme, že formální jazyky LC_j ($j \geq 1$) jsou paralelou jevů v přirozených jazycích, kde jsou valence modelovány pomocí redukcí j-tic symbolů v pevně určených pozicích (s rostoucím počtem takových pozic), viz příklad 1. Formální jazyky LE_j ($j \geq 1$) modelují případy, kdy jsou valence modelovány pomocí redukcí j-tic symbolů ve zcela volně umisťovaných pozicích, viz příklad 2.

Příklad 5. *Nechť $j \geq 1$ a MC_j je* MRR*-automat s $\Sigma_j := \{a, b, c_0, c_1, \dots, c_{j-1}\}$ jako vstupním i kategoriálním slovníkem a MC_j je zadán následující posloupností meta-instrukcí, kde $\Sigma_0 := \{a, b\}$:*

$$(1)\ (¢c_0, a \to \lambda, \Sigma_0^* \cdot c_1, a \to \lambda, \Sigma_0^* \cdot c_2, \dots, \Sigma_0^* \cdot c_{j-1}, a \to \lambda, \Sigma_0^* \cdot \$);$$
$$(2)\ (¢c_0, b \to \lambda, \Sigma_0^* \cdot c_1, b \to \lambda, \Sigma_0^* \cdot c_2, \dots, \Sigma_0^* \cdot c_{j-1}, b \to \lambda, \Sigma_0^* \cdot \$);$$
$$(3)\ (¢c_0 c_1 \dots c_{j-1}\$, \mathsf{Accept}).$$

Je zřejmé, že platí $L_I(MC_j) = L_{IC}(MC_j) = LC_j$ a že MC_j je j-roztržité. Není těžké nahlédnout, že $LC_{j+1} \notin \mathcal{L}_{IC}(j, \mathsf{FRR})$, jelikož pro větu $x \in LC_{j+1}$ nestačí (v obecném případě) j přepisovacích operací v jednom cyklu na transformaci x na kratší větu patřící do LC_{j+1}. Tedy LC_{j+1} nemůže být charakteristickým jazykem pro žádný j-roztržitý FRR-automat.

Na druhou stranu pro každé $j \geq 1$ existuje FRR-automat $\overline{M}_j$ se vstupním slovníkem Σ_j a s odlišným kategoriálním slovníkem takový, že $L_I(\overline{M}_j, 2) = LC_j$. Uvažujme vstup tvaru $c_0 w_1 c_1 w_2 \dots c_{j-2} w_{j-1} c_{j-1} w_j$ s $w_1, \dots, w_j \in \{a, b\}^*$. $\overline{M}_j$ postupně porovnává sousední faktory w_i a w_{i+1} ($1 \leq i \leq j-1$) zleva doprava, k čemuž mu postačí dva průchody přepisů zleva doprava pro každý z těchto faktorů. Z toho vyplývá, že $\overline{M}_j$ je 2-roztržitý.

Tento příklad ilustruje fakt, že jazyk, pokud je posuzován jako vstupní jazyk nějakého FRR-automatu, může být klasifikován podstatně níže v hierarchii než

v případě, kdy je posuzován jako charakteristický jazyk nějakého FRR-automatu. Důvodem tohoto jevu je skutečnost, že pro vstupní jazyky není zaručeno zachovávání správnosti (příslušnosti k jazyku) během přijímacích výpočtů.

Věta 2.

(a) $\mathcal{L}_{IC}(\mathsf{FRR}) \subset \mathcal{L}_I(\mathsf{FRR})$, *a pro všechna* $i \in \mathbb{N}_+$,
(b) $\mathcal{L}_{IC}(i, \mathsf{FRR}) \subset \mathcal{L}_I(i, \mathsf{FRR})$,
(c) $\mathcal{L}_{IC}(i, \mathsf{FRR}) \subset \mathcal{L}_{IC}(i+1, \mathsf{FRR}) \subset \mathcal{L}_{IC}(\mathsf{FRR})$,
(d) $\mathcal{BS}(i, \mathsf{FRR}) \subset \mathcal{BS}(i+1, \mathsf{FRR}) \subset \mathcal{BS}(\mathsf{FRR})$.

Zavádíme pojmy, které umožní formalizovat rozpětí volnosti slovosledu u syntaktických sytémů a charakteristických jazyků FRR-automatů. Jazyky LE_j reprezentují jazyky s rostoucí volností slovosledu.

Definice 4.

(a) *Dvě věty* $u, v \in \{a_1, ..., a_j\}^*$ *nazýváme* Parikhovsky equivalentní *a značíme* $u \equiv v$, *pokud* $|u|_{a_i} = |v|_{a_i}$ *platí pro všechna* $1 \leq i \leq j$.
(b) *Nechť M je* FRR*-automat a nechť* $u, v \in L_{IC}(M)$. *Budeme říkat, že věta u je* M-transformovatelná *do v, což označujeme pomocí* $u \equiv\!>_M v$, *pokud* $u \equiv v$ *a pro každé* u_1 *splňující* $u \vdash^c_M u_1$, *kde* $|u| > |u_1|$, *existuje* v_1 *takové, že* $v \vdash^{c^*}_M v_1$ *a* $u_1 \equiv v_1$.
(c) FRR*-automat M je* redukčně úplný, *pokud pro každé* $i > 0$ *a každé* $u \in L_{IC}(M, i+1)$, *existuje věta* $v \in L_{IC}(M, i)$ *taková, že* $u \equiv\!>_M v$.

Příklad 6. MRR*-automat* M_1 *z příkladu 3 je redukčně úplný. Vezměme* $n \geq 2$ *a* $u := a^n b^n a^n b^n \in L_{IC}(M_1, 2) \setminus L_{IC}(M_1, 1)$. *Vyberme* $v := a^{2n} b^{2n} \in L_{IC}(M_1, 1)$, *potom platí* $u \equiv\!>_{M_1} v$. *Je totiž vidět, že ze vztahů* $u \equiv v$ *a* $u \vdash^c_{M_1} u_1$ *vyplývá* $u_1 = a^{n-1} b^{n-1} a^{n-1} b^{n-1}$. *Dále je vidět, že* $v_1 := a^{2n-2} b^{2n-2} \in L_{IC}(M_1, 1)$ *splňuje podmínky* $v \vdash^{c^*}_{M_1} v_1$ *a* $v_1 \equiv u_1$.

Definice 5. *Redukčně úplný* FRR*-automat M je nazýván* j-volným *pro nějaké* $j \in \mathbb{N}_+$, *pokud pro každé* $i = 1, \ldots, j-1$, $L_{IC}(M, i+1) \notin \mathcal{L}_{IC}(i, \mathsf{FRR})$. *Jazyk* L_{IC}, *resp. syntaktický systém BS je* j-volný *pokud existuje* j*-volný* FRR*-automat M takový, že* $L_{IC} = L_{IC}(M)$, *resp.* $BS = BS(M)$.

Pojem volnosti slouží pro porovnávání volnosti slovosledu.

Tvrzení 6. *Pokud* FRR*-automat M je* j*-volný pro nějaké celé číslo* $j > 1$, *pak M je také* $(j-1)$*-volný.*

Příklad 7. MRR*-automat* M_1 *z příkladu 3 je 2-volný, ale není 3-volný.* MRR*-automat* M_2 *z příkladu 4 je 1-volný, ale není 2-volný.*

Příklad 8. *Jazyk* LC_j *patří do třídy* $\mathcal{L}_{IC}(j, \mathsf{FRR})$, *ale není* j*-volný pro libovolné* $j \geq 2$. *Například, pokud M je* MRR*-automat takový, že* $LC_j = L_{IC}(M)$, *potom* $L_{IC}(M, i)$ *je konečný pro všechna* $1 \leq i < \lceil j/2 \rceil$. *Povšimněme si pro zajímavost, že podjazyk* $\{ c_0 a^n c_1 a^n \ldots c_{j-1} a^n \mid n \geq 0 \}$ *jazyka* LC_j *je obsažen ve třídě* $\mathcal{L}_{IC}(\lceil j/2 \rceil, \mathsf{FRR})$.

Tvrzení 7. *Pro $j \geq 2$ je jazyk LE_j $(j-1)$-volný, ale není j-volný.*

Důkaz. Nechť $j \geq 2$ a M_j je (deterministický) MRR-automat s oběma slovníky rovnými $\Sigma_j := \{a_1, \ldots, a_j\}$, který je definován pomocí meta-instrukcí následujícího tvaru, kde π může být libovolnou permutací množiny indexů $\{1, 2, \ldots, j\}$ a i může nabývat libovolné hodnoty z množiny indexů $3, \ldots, j-1$:

$$
\begin{aligned}
&(1.\pi)\ (\mathbb{c} \cdot a^*_{\pi(1)}, a_{\pi(1)}a_{\pi(2)} \to \lambda, \{a_{\pi(1)}, a_{\pi(2)}\}^*, a_{\pi(3)} \to \lambda,\\
&\qquad\qquad \ldots, (\Sigma_j \setminus \{a_{\pi(j)}\})^*, a_{\pi(j)} \to \lambda, \Sigma_j^* \cdot \$);\\
&(2.i)\ (\mathbb{c} \cdot a_2^*, a_2a_1 \to \lambda, a_1^*, a_3 \to \lambda, a_3^* \cdot a_1^*, a_4 \to \lambda, a_4^* \cdot a_1^*,\\
&\qquad\qquad \ldots, a_i^* \cdot a_1^*, a_{i+1} \ldots a_j \to \lambda, (a_{i+1} \ldots a_j)^* \cdot \$);\\
&(3)\quad (\mathbb{c} \cdot \$, \mathsf{Accept})\,.
\end{aligned}
$$

Pomocné tvrzení 1. $L_{IC}(M_j) = LE_j$ a každý výpočet M_j je $(j-1)$-roztržitý.

Důkaz. Pokud obsah pásky má tvar

$$w(i) := a_2^n a_1^{m_1} a_3^n a_1^{m_2} a_4^n \ldots a_i^n a_1^{m_{i-1}} (a_{i+1} \ldots a_j)^n$$

pro nějaká celá čísla $n, m_1, \ldots, m_{i-1} > 0$, potom lze uplatnit meta-instrukci $(2.i)$. Opakováním takovéto situace může M_j dospět k přijetí $w(i)$, právě když $w(i) \in LE_j$. Odpovídající přijímací výpočet je zřejmě i-roztržitý. Pokud obsah pásky má jiný tvar, pak M_j může uplatnit meta-instrukce typu $(1.\pi)$ a tedy v odpovídajícím cyklu M_j používá přesně $j-1$ vypouštěcích operací. Pokud sestavíme přepisovací (vypouštěcí) konfigurace výpočtu M_j do $j-1$ podposloupností takovým způsobem, že i-tá vypouštěcí konfigurace každého cyklu je uložena do i-té podposloupnosti pro všechna $i = 1, \ldots, j-1$, pak lze nahlédnout, že všechny $(j-1)$ takto získané podposloupnosti jsou monotonní. Vidíme také, že M_j rozpoznává LE_j. □

Můžeme se snadno přesvědčit, že M_j je redukčně úplný. Následujícím výsledkem lze uzavřít důkaz tvrzení 7.

Pomocné tvrzení 2. Pro všechna $i \in \{2, \ldots, j-1\}$, $L_{IC}(M_j, i) \notin \mathcal{L}_{IC}(i-1, \mathsf{FRR})$.

Důkaz. Uvažujme vstup tvaru $w(i)$ (viz výše) takový, že $n = m_1 + \ldots + m_{i-1}$, a všechna čísla $m_1, \ldots, m_{i-1}$ jsou dostatečně velká. Abychom převedli $w(i)$ na kratší větu také patřící do $L_{IC}(M_j, i)$, tak nejméně jeden výskyt každého symbolu $a_1, a_2, \ldots, a_j$ musí být vypuštěn. Pokud čísla $m_2, \ldots, m_{i-1}$ jsou větší než velikost okna uvažovaného automatu, musí automat provést minimálně i operací vypuštění, abychom vypustili alespoň po jednom výskytu každého symbolu. □

□

Následující věta je přímým důsledkem předchozího tvrzení. Jednotlivá tvrzení ukazují, že existují nekonečné škály syntaktických systémů a charakteristických jazyků s ohledem na stupně volnosti i stupně roztržitosti slovosledu. Ukazuje navíc, že roztržitost slovosledu má smysl měřit při libovolném stupni volnosti slovosledu.

Věta 3. *Pro všechna* $i, j \in \mathbb{N}_+, i < j$,
(a) $\mathcal{L}_{IC}((j+1)\text{-volný-FRR}) \subset \mathcal{L}_{IC}(j\text{-volný-FRR})$.
(b) $\mathcal{BS}((j+1)\text{-volný-FRR}) \subset \mathcal{BS}(j\text{-volný-FRR})$.
(c) $\mathcal{L}_{IC}(i, j\text{-volný-FRR}) \subset \mathcal{L}_{IC}(i+1, j\text{-volný-FRR})$.
(d) $\mathcal{BS}(i, j\text{-volný-FRR}) \subset \mathcal{BS}(i+1, j\text{-volný-FRR})$.

4 Závěrečné poznámky

V příspěvku jsme zavedli koncept *j-roztržitosti*, který slouží jako míra složitosti slovosledu (jednotlivých syntaktických jevů) přirozených jazyků, zatímco pojem *j-volnosti* vypovídá o stupni volnosti slovovsledu. Ukázali jsme, že míra složitosti slovosledu může sloužit jako vhodný parametr výpočtu pro restartovací automaty, což se podstatným způsobem promítá do konstrukcí syntaktických analyzátorů přirozených jazyků, a to zejména jazyků s volným slovosledem.

Viděli jsme, že složitost slovosledu může být způsobena jak jazykovými jevy založenými na pevné pozici v syntaktické konstrukci (ilustrováno konstrukcí z holandštiny – Příklad 1), tak jevy, u nichž je syntaktická struktura dána jinými prostředky než slovosledními (jako je např. valence v češtině – Příklad 2). Holandština, jazyk s pevným slovosledem, tvoří 2-roztržité konstrukce (se dvěma fixními pozicemi, kde dochází k přepisování), které ale nejsou 2-volné. Naopak podobné konstrukce v češtině jsou j-volné (kde j je dáno počtem valenčních doplnění valenci nesoucích členů věty, zejména sloves a substantiv): slovosled takových konstrukcí je možné zjednodušit při zachování redukční struktury (viz příklad z úvodu). Podobná pozorování platí i pro latinu, většinu slovanských jazyků a pro některé další typy jazyků, což nás vede k úvahám, které jsme prezentovali v našem příspěvku.

V blízké budoucnosti plánujeme charakterizovat *sílu* různých typů formalizmů, např. [1], [2], na základě pojmů navržených v tomto příspěvku. To se zdá být slibné zejména pro formalizmy založené na topologických omezeních, jako je [1], a to díky zde studovaným, ve své podstatě topologickým omezením.

Reference

1. R. Debusmann, D. Duchier, J. Niehren, The XDG Grammar Development Kit: Topological Dependency Trees: A Constraint-based Account of Linear Precedence. In: Proceedings of the MOZ04 Conference, 39th Annual Meeting of ACL, Charleroi/BEL, LNCS 3389, Springer, 2004, 190–201
2. R. Gramatovici, C. Martin-Vide, Contextual Grammars and Dependency Trees. In: B. Daille, E. Morin (eds.), Actes de TALN. 10e Conference sur le Traitement Automatique des Langues Naturelles, TALN 2003, Batz-sur-Mer, France, 2003, 135–144
3. T. Holan, V. Kuboň, K. Oliva, M. Plátek, Two Useful Measures of Word Order Complexity. In: A. Polguere, S. Kahane (eds.), Proceedings of the Coling '98 Workshop "Processing of Dependency-Based Grammars", University of Montreal, 1998, 21–28

4. T. Holan, V. Kuboň, K. Oliva, M. Plátek, On Complexity of Word Order. In: Les grammaires de dépendance - Traitement automatique des langues (TAL), Vol. 41, No. 1 (q.ed. S. Kahane), 2000, 273-300
5. M. Lopatková, M. Plátek, V. Kuboň, Závislostní redukční analýza přirozených jazyků, In: P. Vojtáš(ed.), ITAT 2004, Information Technologies-Applictions and Theory, Proceedings, Slovakia, 2005, 165–175
6. F. Otto, Restarting automata and their relations to the Chomsky hierarchy. In: Z. Ésik and Z. Fülöp (eds.), Developments in Language Theory, Proc. DLT'2003, LNCS 2710, Springer, Berlin, 2003, 55–74
7. M. Steedman, Linguistic and Computational Theories of Grammar. Course at University of Edinburgh, 2005,
`http://www.iccs.informatics.ed.ac.uk/~steedman/tl/home.html`

Použití Hierarchického asociativního modelu neuronové sítě pro zlepšení lokalizace autonomního robota

Jana Štanclová and David Obdržálek

Katedra softwarového inženýrství
Matematicko-fyzikální fakulta Karlovy Univerzity v Praze
{Jana.Stanclova, David.Obdrzalek}@mff.cuni.cz

Abstrakt V našem článku uvádíme Hierarchický asociativní model (HAM) neuronové sítě – sítě navržené zejména pro vybavování chybějících částí postupně předkládaných dat ve formě map nebo obrazů prostředí. Jednou z motivací k vytvoření prostředku pro doplňování map je jejich využití pro predikci cesty autonomně se pohybujících mobilních zařízení (robotů). Predikci cesty je pak možno využít jako další z metod sloužících k lokalizaci takového mobilního zařízení a lepšímu plánování jeho pohybu v předem známých prostředích.
Pro proces predikce cesty jsme použili námi navrženou HAM síť složenou z asociativních pamětí, které jsou uspořádány do více vrstev. Takový návrh modelu umožňuje odstranit základní nedostatky standardního modelu asociativní paměti - nemožnost ukládání a následného správného vybavování korelovaných dat a relativně malou kapacitu takové neuronové sítě. Vzhledem k tomu, že HAM síť je postupně dynamicky budována na základě konkrétních trénovacích vzorů, je tento model výhodnější, než předem fixně vytvořená neuronová síť, neboť struktura sítě je lépe přizpůsobena konkrétním podmínkám nasazení a je možné ji použít pro libovolně rozsáhlý soubor dat. Článek shrnuje postupný vývoj modelu, prezentuje některé dosažené výsledky a naznačuje směr dalšího vývoje a případného použití.

1 Úvod

Představme si situaci, že cestujeme krajinou, kde jsme již předtím někdy byli. Na konkrétním místě obvykle vidíme jen nejbližší okolí. Pokud se pohybujeme v již známém prostředí, pak jsme na základě scenérie okolo nás schopni vybavit si krajinu, kterou z našeho místa zatím nemůžeme vidět, ale víme, že bude za chvíli následovat (ve směru našeho dalšího putování). Jsme tedy schopni vybavit si např. co je za nejbližším kopcem, co je za nejbližší zatáčkou atd. Na základě vybavené krajiny můžeme vybavovat dále - tedy vzdálenější krajinu. Tato schopnost nám umožňuje rychle a bezpečně cestovat (např. včas se vyhnout překážkám a zbytečně nebloudit).

Pro jednoduché vybavování krajiny (resp. její zjednodušené reprezentace) je možné použít model standardní asociativní paměti. Tento základní model

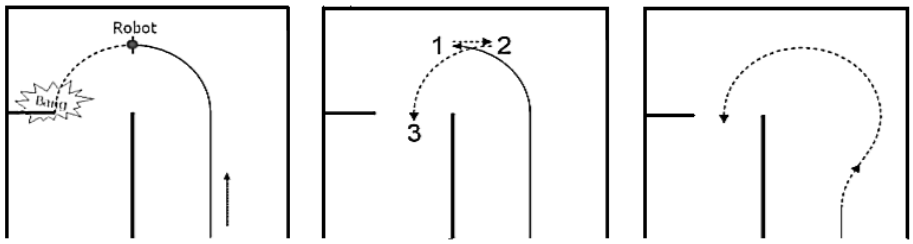
však není příliš použitelný při vybavování rozsáhlejší krajiny a to hned ze dvou zásadních důvodů. Prvním důvodem je relativně malý počet vzorů, které lze do modelu uložit a následně je správně vybavit. Druhým důvodem je fakt, že není zaručeno správné vybavování korelovaných vzorů. Pro vybavování krajiny jsme proto navrhli a implementovali model Hierarchické asociativní paměti, která by měla odstranit některé nedostatky základního modelu asociativní paměti. Tento model bychom chtěli použít v autonomních robotech jako další metodu pro zlepšení lokalizace robota v prostoru.

Další členění článku je následující: V druhé kapitole popíšeme motivaci, která nastiňuje jak by bylo možné použít model Hierarchické asociativní paměti pro zlepšení lokalizace autonomního robota. Ve třetí kapitole formálněji popíšeme princip vybavování krajiny. Další kapitola bude věnována základnímu modelu asociativní paměti, jenž představuje základní prvek našeho systému. Pátá kapitola bude zaměřena na popis našeho modelu Hierarchické asociativní paměti, který chceme použít pro vybavování. V šesté kapitole zmíníme některé zajímavé výsledky provedených experimentů a v poslední kapitole krátce vyhodnotíme získané výsledky a nastíníme naši další práci.

2 Motivace pro použití modelu při lokalizaci robota

Při návrhu mechanismu řízení autonomního robota jsou obvykle brány v úvahu různé typy zdrojů vstupních informací [2]: podněty z nárazníků a dalších senzorů, informace z odometrie, signál z kamery, signály z majáků a mnohé další (kontaktní/nekontaktní, aktivní/pasivní, ...). Většina metod používaných pro zpracování příchozích dat je však zatížena chybami a kumulativními chybami, takže správná informace o poloze robota nemůže obvykle být udržena s dostatečnou předností po delší dobu pouze na základě jediného zdroje dat. Například odometrická informace, kterou robot získá ze svých kol, nedokáže zaznamenat prokluz kol na kluzkém povrchu nebo změnu celkové polohy robota vyprovokovanou působením vnější síly na robota v bočním směru. Proto je pro určení polohy robota v jeho virtuálním světě v drtivé většině případů použito více různých metod zpracování informací z vnějšího světa.

Námi navržené vybavování mapy může být použito pro rozšíření palety takových metod. Výsledky z vybavování map mohou být například použity pro srovnání s dříve získanou (nebo naučenou) mapou prostředí a mohou pomoci eliminovat nebo alespoň snížit chybu pocházející z jiných zdrojů lokalizace robota. Představme si řídící a plánovací mechanismus robota, který má řídit robota při jeho pohybu mezi místnostmi uvnitř budovy. V průběhu postupu robota po chodbách je třeba na křižovatkách odbočovat tak, jak to vyžaduje stanovený jízdní plán. Dříve, než však robot skutečně odbočí, musí především zjistit, že je na křižovatce a určit, která konkrétní křižovatka to je. Pokud bychom implementovali navržené vybavování mapy na základě obrazu získávaného z kamery připevněné na robotovi, mohli bychom získat výstup, který je vlastně predikcí toho, co robot uvidí v nejbližší budoucnosti ("za rohem") - tedy právě křižovatku nebo významnou zatáčku.



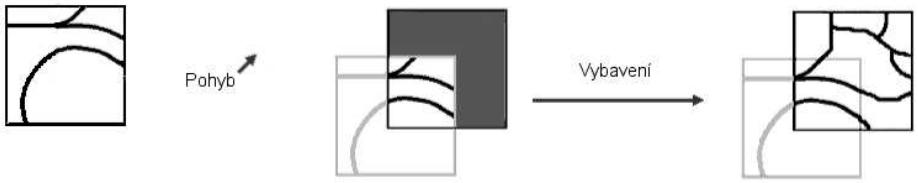
Obr. 1. Příklad motivace pro využití predikce mapy.

Toto vlastně znamená, že můžeme řídícímu mechanismu robota poskytnout poměrně důležitou informaci ještě před tím, než doopravdy přijede ke křižovatce nebo zatáčce (a může je zdetekovat standardními senzory). Řídící mechanismus tak jednak bude mít více času na rozhodnutí o další cestě, což může být významné v případě nízké výpočetní síly rozhodovací jednotky, jejíž malý výkon by vedl k tomu, že robot tu a tam zastaví, aby získal čas na propočtení další cesty. Řídící mechanismu také může začít manévrovat ještě před tím, než křižovatku nebo zatáčku zdetekuje jinými senzory, a tak může dosáhnout jednak plynulejšího pohybu (viz obrázek 1), nebo umožnit pohyb jako takový (např. při pohybu velkého robota v úzkých chodbách). Vstupem pro vybavování mapy nemusí také být jen obraz z kamery, vstupem mohou být data z libovolného typu senzoru za předpokladu, že jsme schopni na základě těchto dat vytvořit pomyslnou "mapu" těchto dat pro celé pracovní prostředí robota.

3 Proces vybavování krajiny

Ve Fukushimově modelu [3] je proces vybavování krajiny ve směru dalšího pohybu simulován pomocí tzv. korelační paměti (velmi podobné asociativní paměti). Princip vybavování krajiny je znázorněn na obrázku 2. Mapa krajiny je rozdělena na vzájemně se překrývající segmenty, které jsou uloženy během procesu učení do korelační paměti. Nejbližší okolí, které z daného místa vidíme, je reprezentováno tzv. aktuálním vzorem s egocentrickým systémem souřadnic (tj. stojíme vždy ve středu vzoru). Při pohybu se aktuální vzor posune tak, abychom byli vždy ve středu aktuálního vzoru. Posunutím obrazu podle směru našeho pohybu se v novém aktuálním vzoru se objeví prázdná oblast. Prázdná část odpovídá krajině, kterou jsme předtím nemohli vidět (např. byla skryta za zatáčkou). Nový aktuální vzor je tedy tvořen "známou" krajinou, kterou jsme viděli z předchozího místa, a "prázdnou" oblastí, která odpovídá "doposud neviděné" krajině. Takto vytvořený vzor (tzv. neúplný vzor) je předložen korelační paměti, aby vzor vybavila. Vybavený vzor by měl mít doplněnou prázdnou oblast na základě korelační matice.

Korelační paměť nezaručuje, že vybavování je invariantní vůči posunutí. Posunutý objekt tak, jak je zakódován, představuje pro korelační matici zcela jiný vzor. Proto předkládané neúplné vzory musí přesně odpovídat některému



Obr. 2. Vybavování krajiny inspirované Fukushimovým modelem.

z uložených vzorů (ve smyslu posunutí objektů). K odstranění tohoto problému je ve Fukushimově modelu použit princip tzv. pilotního vzoru. Pozice, ve které je dosaženo maximální korelace mezi "známou" částí neúplného vzoru a pilotním vzorem, se stane středem aktuálního vzoru. Proces vybavování vzoru může někdy selhat, proto Fukushimův model obsahuje tzv. monitorovaci obvod, který toto selhání detekuje. Je–li detekováno selhání procesu vybavování, vybavený vzor se nepoužije a proces vybavování je přesunut do jiného místa a zopakován [3].

4 Model asociativní paměti

Asociativní paměť je model neuronové sítě, kde jsou všechny neurony současně vstupní i výstupní (viz např. [5]). Neurony jsou mezi sebou propojeny pomocí orientovaných vazeb. Všechny váhy vazeb jsou symetrické a každý neuron je spojen se všemi ostatními neurony. Výstupem sítě je vektor výstupních hodnot všech neuronů asociativní paměti. Váhová matice W asociativní paměti s n ($n > 0$) neurony je matice $W = (w_{ij})$ velikosti $n \times n$, kde w_{ij} značí váhu mezi neuronem i a j.

Pro učení asociativní paměti lze použít tzv. Hebbovské učení. Na počáku procesu jsou váhy nastaveny na nulové hodnoty. Po předložení trénovacího vzoru $\boldsymbol{x}^k = (x_1^k, \ldots, x_n^k)$ dojde k adaptaci vah w_{ij} podle vztahu:

$$w_{ij} \leftarrow w_{ij} + x_i^k x_j^k \quad i, j = 1, \ldots, n \quad i \neq j$$

Vybavování asociativní sítě je iterativní proces. V každé iteraci je vybrán neuron, který aktualizuje svůj stav (podle znaménka potenciálu neuronu). Neuron, který nebyl vybrán k aktualizaci, svůj stav nemění. Lze ukázat [5], že asociativní paměť s asynchronní dynamikou (tj. výběr neuronu k aktualizaci náhodný a nezávislý) konverguje k lokálnímu minimu energetické funkce.

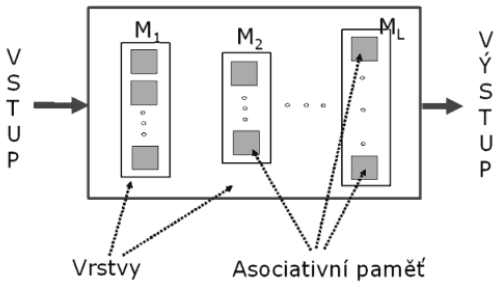
Asociativní paměť představuje základní model, který je možné použít pro zpracování obrazů a při jejich následném rozpoznávání. Asociativní paměť je schopna vybavit vzory, které byly do modelu dříve uloženy a nyní jsou předloženy k rozpoznání. Síť je schopna vybavit i vzory, které jsou "poničené" nebo "neúplné". Nedostatkem asociativní sítě je její relativně malá kapacita (přibližně $0.15n$, kde n je dimenze ukládaných vzorů [5]) a požadavek na ortogonalitu ukládaných

trénovacích vzorů. Ukládání korelovaných vzorů může způsobit problémy a dojde pravděpodobně ke ztrátě některých dříve uložených trénovacích vzorů, protože tzv. crosstalk (součet skalárních součinů vzorů s ostatními uloženými vzory) je nenulový [1].

5 Model Hierarchické asociativní paměti

Standardní asociativní paměť je schopna vybavit poškozené nebo neúplné vzory za předpokladu, že počet vzorů uložených v síti je relativně malý a že uložené vzory jsou ortogonální (nebo téměř ortogonální). Tyto předpoklady redukují možnost použití asociativních pamětí v praxi, kde je nutné pracovat s velkým množstvím dat, které jsou téměř vždy korelovaná. Abychom se vyhnuli těmto omezením, navrhli jsme model tzv. Hierarchické asociativní paměti (HAM–model) [6]. HAM–model byl inspirován modelem tzv. Kaskádové asociativní paměti [4], který umožňuje pracovat se speciálním druhem korelovaných vzorů. Naším cílem bylo vytvořit model schopný pracovat s velkým množstvím dat.

Hierarchická asociativní paměť H s L ($L > 0$) vrstvami je uspořádaná L–tice $H = (M_1, \ldots, M_L)$, kde $M_1, \ldots, M_L$ jsou konečné neprázdné množiny asociativních pamětí (tzv. lokální asociativní paměti); každá z lokálních asociativních pamětí má stejný počet neuronů $n (n > 0)$. Množina M_k $(k = 1, \ldots, L)$ se nazývá vrstva hierarchické paměti H. $|M_k|$ značí počet lokálních asociativních pamětí ve vrstě M_k $(k = 1, \ldots, L)$. Trénovací množina T hierarchické asociativní paměti H je uspořádaná L–tice $T = (T_1, \ldots, T_L)$, kde T_k $(k = 1, \ldots, L)$ je konečná neprázdná množina trénovacích vzorů pro vrstvu M_k. Struktura HAM–modelu je zobrazena na obrázku 3.

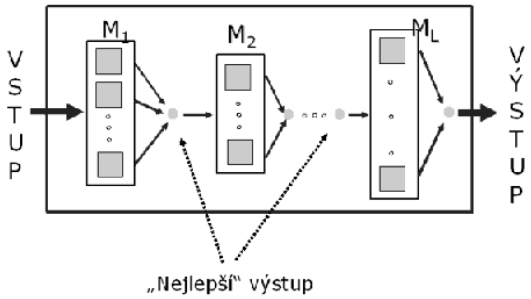


Obr. 3. Struktura Hierarchické asociativní paměti.

Pro učení HAM–modelu jsme navrhli tzv. dynamický učící algoritmus. Každá vrstva M_k $(k = 1, ..., L)$ je učena zvlášť a nezávisle na ostatních vrstvách (tento proces lze dělat paralelně pro všechny vrstvy). Trénovací vzory z množiny T_k $(k = 1, ..., L)$ jsou uloženy v lokálních asociativních sítí vrstvy M_k. Během učení

vrstvy M_k $(k = 1, ..., L)$ jsou trénovací vzory vybírány z příslušné trénovací množiny T_k sekvenčně a předkládány vrstvě M_k. Pro každý trénovací vzor z T_k se hledá "vhodná" lokální asociativní paměť ve vrstvě M_k (tj. taková lokální síť, kde by byl vzor uložen a následně vybaven zcela korektně) a vzor je uložen do této lokální sítě. Když není nalezena žádná "vhodná" lokální asociativní paměť ve vrstvě M_k, je dynamicky vytvořena nová lokální asociativní paměť a přidána na dané vrstvy HAM–modelu. Vzor je pak uložen do nově přidané lokální asociativní sítě.

Během procesu vybavování je na vstup HAM–modelu předložen vzor $\boldsymbol{x}$. Předložený vzor $\boldsymbol{x}$ je vstupním vzorem první vrstvy M_1 (tj. $\boldsymbol{x} = {}^1\boldsymbol{x}$). V každém kroku k $(1 \leq k \leq L)$ vrstva M_k spočte svůj výstup ${}^k\boldsymbol{y}$ jako odezvu na předložený vstup ${}^k\boldsymbol{x}$. Výstup ${}^k\boldsymbol{y}$ vrstvy M_k představuje vstup pro další vrstvu M_{k+1} (tj. ${}^k\boldsymbol{y} = {}^{k+1}\boldsymbol{x}$ pro $1 \leq k < L$). Výstup "poslední" vrstvy M_L představuje výstup $\boldsymbol{y}$ celého HAM–modelu. Proces vybavování HAM–modelu je zachycen na obrázku 4.



Obr. 4. Proces vybavování HAM–modelu.

Při procesu vybavování ve vrstvě M_k je vstupní vzor ${}^k\boldsymbol{x}$ předložen všem lokálním asociativním sítím na dané vrstvě, které spočtou svůj výstup ${}^k\boldsymbol{y}_i$ $(i = 1, \ldots, |M_k|)$. Výstup ${}^k\boldsymbol{y}$ vrstvy M_k je definován jako výstup, který je "nejpodobnější" vstupnímu vzoru ${}^k\boldsymbol{x}$ (tj. ${}^k\boldsymbol{y} = max\ \{podobnost({}^k\boldsymbol{x}, {}^k\boldsymbol{y}_i)\ \ i = 1, \ldots, |M_k|\}$). V experimentech používáme Hammingovu vzdálenost jako míru podobnosti vzorů, ale je možné použít i jiné míry podobnosti.

Navržený dynamický učící algoritmus HAM–modelu je jednoduchý a snadno implementovatelný, ale není optimální. Trénovací vzor zůstane uložen v té lokální sítí, kde by uložený vzor byl korektně vybaven. Při použití této metody učení však nejsme schopni nic říci o tom, jak se po uložení nového vzoru budou vybavovat dříve uložené vzory. Může dojít k situaci, že některé z dříve uložených vzorů začnou být vybavovány nesprávně nebo může dokonce dojít k jejich zapomenutí.

6 Simulace a analýza získaných výsledků

Jedna z nejdůležitějších vlastností neuronových sítí je jejich schopnost vybavovat. Při použití HAM–modelu v autonomních robotech jsou nezbytné robustní vybavovací schopnosti HAM–modelu (zejména při vybavování neúplných vzorů). Na základě těchto požadavků jsme navržený HAM–model modifikovali. Trénovací vzor byl uložen do takové lokální asociativní paměti (v příslušné vrstvě), kde byl vybaven korektně nejen uložený trénovací vzor ale také zašuměný trénovací vzor (tj. vzor, kde byly náhodně vybrané prvky změněny). Když taková lokální síť nebyla nalezena, byla vytvořena nová lokální asociativní síť a trénovací vzor byl uložen do nově vytvořené sítě.

V našich experimentech jsme se omezili na dvouvrstvý HAM–model. Vzory předkládané první vrstvě jsou tzv. vzory předků a pro druhou vrstvu tzv. vzory potomků. Tento dvouvrstvý model má poměrně jednoduchou geometrickou interpretaci. Můžeme předpokládat, že vstupní data jsou uskupena do shluků. Vzory předků mohou reprezentovat jednotlivé shluky (např. středy shluků) a vzory potomků pak vlastní data. Trénovací vzory pro první vrstvu jsou tzv. trénovací vzory předků a pro druhou vrstvu tzv. trénovací vzory potomků. Na začátku učení byla v každé vrstvě jen jediná lokální asociativní síť. Ostatní lokální sítě byly přidávány do jednotlivých vrstev dynamicky během procesu učení. Výsledný počet lokálních sítí tedy závisel na předkládaných datech.

Pro experimenty jsme vytvořili 100 sad náhodně vygenerovaných vzorů o rozměrech 15×15 bodů. Každá sada dat obsahovala 100 bipolárních vzorů (tj. vzory, které byly tvořeny jen hodnotami $+1$ nebo -1). Z každé sady dat bylo vybráno 25% vzorů s nejmenší kumulativní korelací s ostatními vzory a tyto vzory se staly trénovacími vzory předků. Zbylé vzory představovaly trénovací vzory potomků. Pro zlepšení rozpoznávacích schopností modelu jsme v druhé vrstvě nepracovali přímo se vzory potomků ale s tzv. rozdílovými vektory. Pro každý vzor potomka byl nalezen jeho odpovídající vzor předka (např. pomocí první vrstvy HAM–modelu) a rozdílový vektor byl vytvořen na základě předloženého vzoru potomka a nalezeného vzoru předka. Takto vytvořený rozdílový vektor byl předložen druhé vrstvě HAM–modelu k vybavení. Výsledný vybavený vzor byl zkonstruován na základě vybaveného rozdílového vektoru a odpovídajícího vybaveného předka (inverzím postupem k postupu vytváření rozdílového vektoru).

Na základě provedených experimentů s náhodně vygenerovanými daty jsme zjistili, že implementovaný HAM–model nebyl schopen rozpoznat korektně všechny uložené vzory (některé z uložených vzorů byly vybaveny nesprávně). Rozhodli jsme proto uměle omezit kapacitu všech lokálních asociativních pamětí v HAM–modelu na hodnotu menší než teoretická maximální kapacita $0.15n$, kde n je dimenze dat [5]. Zavedli jsme tzv. kapacitní koeficient c redukující maximální počet vzorů, které bylo možné uložit do každé lokální asociativní paměti. Maximální kapacita q lokální asociativní paměti byla definována vztahem:

$$q = 0.15 \cdot n \cdot c.$$

Pro standardní asociativní paměť nabývá kapacitní koeficient c hodnoty 1. Se snižujícími hodnotami koeficientu c klesala maximální kapacita lokálních sítí a zlepšovaly se schopnosti HAM–modelu rozpoznat (vybavit) uložené vzory. Získané výsledky jsou shrnuty obrázku 5.

c	recalled correctly
1	56,71%
0,9	56,83%
0,8	82,73%
0,7	89,05%
0,6	92,80%
0,5	97,15%
0,4	99,20%
0,3	99,95%
0,2	100,00%

Obr. 5. Procento korektně vybavených trénovacích vzorů s ohledem na kapacitní koeficient c.

V dalších experimentech jsme zvolili kapacitní koeficient lokálních sítí $c = 0.15$. Nejprve jsme nechali vybavit uložené vzory a měřili jsme počet chybně vybavených bodů v jednotlivých vzorech podle polohy bodu ve vzoru. Získané výsledky jsou na obrázku 6. Je vidět, že jednotlivé body byly vybaveny chybně v méně než 0.1% vzorů.

6	3	2	5	2	1	1	3	4	3	4	8	1	3	2
1	3	4	2	1	2	2	5	3	4	2	3	5	2	3
3	6	7	2	3	3	7	7	6	4	4	4	1	5	1
5	5	2	3	1	2	5	4	4	6	2	4	3	3	3
2	3	5	1	3	5	4	6	1	5	5	1	4	2	2
2	4	5	6	5	2	3	1	8	1	2	7	2	5	1
2	6	0	3	5	2	4	1	3	2	5	2	3	2	5
2	7	1	5	5	6	3	5	3	5	6	4	3	2	5
1	4	6	4	3	4	2	4	2	0	6	3	1	4	3
2	2	1	4	4	1	4	4	0	4	1	10	3	3	3
7	7	5	4	4	1	3	6	3	4	2	2	5	2	0
3	3	3	3	3	1	5	4	3	4	3	2	6	2	5
5	3	2	3	4	3	1	4	1	3	3	1	2	3	2
1	5	3	2	6	3	4	2	5	2	1	6	3	3	4
2	5	2	0	4	6	6	3	7	3	6	3	3	0	3

Obr. 6. Počet chybně rozpoznaných bodů v závislosti na poloze bodu ve vzoru při vybavování uložených vzorů (v desetinách promile).

Poté jsme se zaměřili na schopnosti HAM–modelu vybavit neúplné vzory, které obsahovaly 13% neznámých pixelů. To odpovídalo diagonálnímu posunu

vzoru o 1 bod (tj. každý prvek vzoru se posunul o 1 bod v daném diagonálním směru). Neúplné vzory byly rozděleny do 4 skupin podle směru posunu: ↘, ↗, ↙ a ↖. Směr posunu byl opačný k "poloze neúplné oblasti" ve vzoru. I v tomto experimentu jsme měřili počet chybně vybavených bodů v závislosti na poloze bod ve vzoru. Výsledky vybavování neúplných vzorů s posunutím ↘ jsou zachyceny na obrázku 7. Vybavování vzorů s jiným diagonálním posunutím přineslo velmi podobné výsledky. Při vybavování neúplných vzorů byla chybovost jednotlivých pixelů vyšší než při vybavování uložených vzorů. Chybovost jednotlivých pixelů ve "známé" části byla mezi $17.8\% - 19.5\%$ a v "neznámé" části mezi $21.5\% - 22.9\%$.

22,5	21,7	21,6	22,5	21,8	22,4	22,0	22,1	21,7	22,3	22,4	22,2	22,3	22,1	22,0
22,2	18,8	18,7	18,6	18,2	19,2	18,6	18,8	18,8	18,5	18,7	18,3	18,9	19,1	18,6
22,2	19,1	18,9	18,6	18,1	18,4	18,9	18,4	18,6	18,5	18,5	18,6	18,3	18,6	18,6
22,9	18,3	18,8	18,4	19,0	19,4	19,1	19,3	19,0	18,0	19,1	19,1	19,3	19,5	18,6
22,8	18,6	18,5	18,7	18,6	18,6	18,2	18,6	17,8	18,7	19,0	18,2	18,9	18,6	19,0
22,1	18,7	19,0	18,8	18,8	18,8	19,0	18,3	19,1	19,0	18,7	18,3	18,4	18,2	18,5
21,9	19,0	18,9	18,4	18,5	18,1	18,4	18,7	18,7	18,8	19,5	18,4	18,6	18,5	18,6
22,0	18,4	18,8	18,5	18,3	18,7	18,9	18,5	19,1	18,7	18,9	18,3	18,0	18,1	18,8
22,1	18,6	19,5	18,7	18,5	18,8	18,2	18,8	18,7	18,6	19,0	19,0	18,3	18,8	18,6
22,7	18,4	18,5	18,5	19,1	18,8	18,6	18,5	18,4	18,8	18,6	18,9	19,1	18,6	18,9
21,5	18,4	18,6	19,1	18,7	18,1	19,0	18,6	19,2	18,9	18,9	18,5	18,4	19,2	18,9
22,7	18,9	18,8	18,5	18,6	18,9	18,3	18,7	19,0	18,6	19,4	18,3	18,6	19,0	18,7
21,9	19,2	18,7	18,7	17,8	18,4	19,0	19,1	18,3	18,1	18,1	18,7	18,5	19,0	18,6
22,4	18,3	18,6	18,1	18,6	18,6	19,3	18,5	19,2	18,7	19,1	19,0	18,7	19,1	18,6
22,1	18,9	18,4	18,5	18,7	18,5	18,7	18,1	18,7	18,4	18,7	18,7	18,6	18,9	18,5

Obr. 7. Počet chybně rozpoznaných pixelů v závislosti na poloze pixelu ve vzoru při vybavování neúplných vzorů s posunutím ↘ (v procentech).

7 Závěr

V tomto článku jsme se zaměřili na aplikaci modelu asociativních pamětí pro predikci cesty autonomně se pohybujících mobilních zařízení. Tato metoda by mohla pomoci zlepšit lokalizaci mobilních zařízení v předem známých prostředích. Činnost standardního modelu asociativní paměti však závisí na počtu uložených vzorů a je velmi citlivá na vzájemnou korelaci uložených vzorů. Z tohoto důvodu jsme navrhli a implementovali model Hierarchické asociativní paměti, který by měl zlepšit schopnosti standardní asociativní paměti a umožnit pracovat s libovolným počtem vzorů (i korelovaných). Vybavovací schopnosti celého HAM–modelu mohou být zlepšeny umělým omezením kapacity jednotlivých asociativních sítí. Experimenty správnost omezení kapacit lokálních sítí potvrdily.

Důležitým momentem při činnosti HAM–modelu je správný výběr lokální asociativní paměti pro uložení předloženého vzoru. S tímto úzce souvisí požadavek, aby po uložení dalšího vzoru do HAM–modelu nedošlo k poničení či zapomenutí dříve uložených vzorů. V současné implementaci jsme použili základní

verzi dynamického učícího algoritmu, která však výše uvedený požadavek nezaručí. Nyní pracujeme na zlepšení učícího algoritmu a nalezení sofistikovanější metody učení, která by měla zlepšit robustnost modelu. Do budoucna bychom chtěli analyzovat časovou a prostorovou složitost našeho HAM–modelu.

Reference

1. D.J. Amit, H. Gutfreund, H. Sompolinsky, Information Storage in Neural Networks with Low Levels of Activity, Physical Review A, 35, 1987, 2293–2303
2. H.R. Everett, Sensors For Mobile Robots, ISBN 1–56881–048–2, A K Peters, 1995
3. K. Fukushima, Y. Yamaguchi, M. Okada, Neural Network Model of Spatial Memory: Associative Recall of Maps, Neural Network, Vol. 10, No. 6, 1997, 971–979
4. M. Hirahara, N. Oka, T. Kindo, A Cascade Associative Memory Model with a Hierarchical Memory Structure, Neural Networks, Vol. 13, No. 1, 2000, 41–50
5. J.J. Hopfield: Neural Networks and Physical System with Emergent Collective Computational Abilities, Proc. Natl. Acad. Sci. USA, 79: 1982, 2554–2558
6. I. Mrázová, J. Tesková, Hierarchical Associative Memories for Storing Spatial Patterns, Proceedings of ITAT 2002, 2002, 143-154

Pravděpodobnostní pohled na víceatributové dotazy v distribuovaných systémech

Marta Vomlelová[1] and Peter Vojtáš[2]

[1] Matematicko-fyzikální fakulta, Univerzita Karlova, Praha, Česká republika
marta@kti.mff.cuni.cz
[2] Ústav informatiky AV ČR, Pod Vodárenskou věží 2, 182 07 Praha 8
Česká republika
vojtas@cs.cas.cz

Abstrakt Článek se zabývá algoritmy hledání nejlepšího (k-nejlepších) objektů v distribuovaných systémech, zejména na webu. Předpokládáme, že každý zdroj vrací objekty uspořádané podle zvoleného kriteria (kriterií), otázkou je, které zdroje dotazovat, abychom našli nejlepší objekt(y) na co nejméně dotazů. Známé algoritmy jsou většinou založené na výpočtu gradientu, tj. lokálním odhadu. V tomto článku ukazujeme, že je často možné odhadnout celé pravděpodobnostní rozložení hodnot jevů (za předpokladu nezávislosti či znalosti modelu). Globální znalost pravděpodobnostního rozložení umožní lépe optimalizovat hledání nejlepších objektů, nicméně experimenty ukazují, že heuristické metody jsou v testovaných příkladech blízko optima. Výpočet optimálního zdroje pro dotaz vede k výpočtu několikanásobných intergrálů; přímý výpočet lze nahradit přibližným výpočtem (např. metodou Monte Carlo). Alternativní možností je přijmout diskutabilní předpoklad rovnoměrného rozložení a používat odvozené vzorce jako heuristiky.

1 Úvod

S rozvojem sémantického webu nabývá na významu potřeba hledání objektů s distribuovanými atributy. Typické atributy, které nelze vyčíst z jediného zdroje, jsou geografické údaje vážící se k rezervacím (hotelů apod.) a cenám. Navíc jsou uživatelovy preference většinou vágní, fuzzy.

Použijme jako příklad člověka organizujícího pracovní cestu letadlem, rezervaci hotelu a vypůjčení auta. Jednotlivými atributy tvořícími celkové preference jsou cena, vzdálenost, počet leteckých spojení apod. Na vyhledání optimálního řešení se nehodí dotazy AND a OR, protože jednotlivá kriteria mají vícestupňovou stupnici "kvality". Lepší je celkovou kvalitu vyjádřit agregační funkcí, např. váženým součtem jednotlivých kriterií, a hledat objekt s maximální hodnotou agregační funkce.

Tento problém formuloval R. Fagin [2], [1] a zavedl tzv. Faginův algoritmus. Poté spolu s dalšími [4] zavedl Threshold algoritmus. Ekvivalentní algoritmus navrhli i Nepal et. al. [7]. Güntzen et al. [5] definovali "quick–combine" algoritmus s heuristickým pravidlem, který seznam má být preferovaný. Nastev et al. [6] zobecnili tento problém na libovolná spojení v databázích.

Tyto algoritmy se explicitně nezabývají pravděpodobnostním rozložením veličin. Na základě lokálních vlastností agregační funkce vybírají nejslibnější sloupec. V této volbě je často skrytý předpoklad nezávislosti jednotlivých atributů a rovnoměrné rozložení jejich hodnot.

V tomto článku se explicitně věnujeme úvahám o pravděpodobnostním rozložení hodnot. Odhad pravděpodobnostního rozložení a přibližné řešení vede k lepší heuristice zvláště v případech, kdy se rozložení hodnot atributů silně liší od rovnoměrného. Nicméně experimenty ukázaly, že na testovacích příkladech nebyly nejlepší literaturou uváděné heuristiky daleko od optima.

V Sekci 2 uvádíme základní model, algoritmus a heuristiky popsané v literatuře. Sekce 3 popisuje pravděpodobnostní modely pro jednotlivé úkoly: potvrzení maxima, nalezení a potvrzení maxima a samotné nalezení maxima. Úkoly jsou řazeny podle vzrůstající složitosti odvození.

V závěru uvádíme výsledky experimentů.

2 Model, Threshold algoritmus a heuristiky

2.1 Model bez pravděpodobnostní znalosti

Mějme konečnou množinu objektů o velikosti N. Každý objekt x má m atributů $x_1, \ldots, x_m$. Hodnoty atributů jsou reálná čísla. Pro každý atribut i vytvoříme seznam L_i, uspořádaný sestupně podle hodnot atributu i obsahující všechny objekty. Na každém z těchto seznamů definujeme dvě funkce: objekt na j–té pozici v seznamu L_i značíme $r_i(j)$, hodnotu atributu i pro objekt x značíme $s_i(x)$. Máme dva možné přístupy k seznamům: buď sekvenčně, od prvního prvku vždy k následujícímu, nebo náhodným přístupem, kdy pro daný prvek x zjišťujeme hodnotu atributu i. Abychom mohli zjišťovat hodnotu atributů prvku x, museli jsme na tento prvek nejdříve narazit uspořádaným přístupem v některém ze sloupců. Do té doby o objektu x nevíme.

Dále je daná agregační funkce F, která kombinuje hodnoty jednotlivých atributů do jedné – agregované, sumární – kvality objektu x. Funkci F vyžadujeme monotónní ve všech argumentech a některé heuristiky navíc zleva spojitou, aby existovala derivace zleva. Tato funkce je vstupem všech následujících algoritmů. Agregovaná hodnota objektu x se značí $S(x)$ a spočte se: $S(x) = F(s_1(x), \ldots, s_m(x))$.

Naším úkolem je najít k nejlepších objektů (objektů s maximální $S(x)$), přičemž se snažíme minimalizovat čas, měřený v počtu sekvenčních přístupů do seznamů L_i.

2.2 Nalezení versus potvrzení

Experimenty ukazují, že i když maximum (nejlepších *málo* prvků) nalezneme, potřebujeme hodně času na to, abychom došli pod práh *threshold* a tím potvrdili, že nalezené prvky jsou opravdu nejlepší. Čas od prvního navštívení do potvrzení bývá často i delší než čas do prvního navštívení.

2.3 Zobecněná verze Threshold algoritmu (TA)

Mějme vektor $z = \langle z_1, \ldots, z_m \rangle$, ve kterém si budeme pamatovat pro každý sloupec i poslední pozici navštívenou sekvenčním přístupem (na počátku $z_i = 0$ pro $i = 1, \ldots, m$). Mějme heuristiku H, která nám řekne, kterým/kterými seznamy L_i máme pokračovat v hledání uspořádaným přístupem. Pro daný stav prohledání z heuristika vydá doporučení pro každý sloupec, $H(z) = \langle H(z)_1, \ldots, H(z)_m \rangle$, přičemž nikdy nemá význam se vracet na nižší hodnotu z_i, tj. $\forall i$ požadujeme $H(z)_i \geq z_i$ a alespoň v jednom soupci potřebujeme postoupit, tj. $\exists i$ tak, že $H(z)_i > z_i$. Heuristika nám tím definuje *množinu kandidátů* příštího uspořádaného přístupu $\{i \leq m : H(z)_i > z_i\}$.

Jednotlivé algoritmy se liší v heuristice H (či rodině heuristik) a v zacházení s kandidáty na další krok. Některé heuristiky přistupují paralelně ke všem kandidátům, my se budeme zabývat jen těmi, které přistupují ke kandidátům po jednom (náhodně).

Threshold algoritmus hledá nejlepších k objektů včetně jejich potvrzení. Pro každý sloupec si pamatuje hodnotu posledního navštíveného objektu v daném seznamu, $u_i = s_i(r_i(z_i)))$. Víme, že nenavštívené objekty nemohou mít hodnotu tohoto atributu vyšší, proto žádný nenavštívený objekt nemůže mít vyšší hodnotu agregované funkce než $threshold = F(u_1, \ldots, u_m)$.

Algoritmus je založen na tom, že jakmile máme k objektů s hodnotou $S(x)$ vyšší než *threshold*, můžeme skončit prohledávání. Neviděný objekt totiž nemůže mít v žádném sloupci i lepší hodnotu než odpovídající u_i a protože předpokládáme funkci F monotónně rostoucí, ani agregovaná hodnota $F(u_1, \ldots, u_m)$ nemůže být lepší než $S(x)$.

Threshold algoritmus $TA(H, k)$ [4]

1. $z \leftarrow \langle 0, \ldots, 0 \rangle$
2. aktualizujeme $H(z)$
3. Náhodně vybereme jedno i, pro které $H(z)_i > z_i$. Ve sloupci L_i sestoupíme o jeden záznam a přiřadíme $z_i \leftarrow z_i + 1$; jiná z_j se nemění.
4. Pro nově objevený objekt x zjistíme hodnoty jeho ostatních atributů $s_j(x)$ a spočteme agregovanou hodnotu $S(x) \leftarrow F(s_1(x), \ldots, s_m(x))$. Je–li tato hodnota jednou z k nejvyšších zatím nalezených, zapamatujeme si x a $S(x)$ (při rovnosti hodnot vybereme libovolný).
5. Aktualizujeme $u_i \leftarrow s_i(r_i(z_i)))$ a $threshold \leftarrow F(u_1, \ldots, u_m)$. Pokud k nejlepších nalezených objektů ma hodnotu vyšší než *threshold*, končíme, jinak celý cyklus opakujeme od bodu 2.
6. Vrátíme k nejlepších objektů spolu s jejich hodnotami.

2.4 Používané heuristiky

Ze známých heuristik uvedeme dvě základní a jejich kombinaci.

Güntzer et all. [5] se snaží co nejrychleji snížit hodnotu prahu *threshold*. Heuristiku lze snadno popsat jako aproximaci derivace funkce $S(x)$ podle jednotlivých složek r_i, kde se výsledné kriterium rovná součinu derivace F podle

s_j násobeným odhadem derivace vnitřní funkce, tj. derivace $s_j(r_j)$ podle r_j. Vnitřní funkce není spojitá, proto derivaci autoři nahrazují rozdílem hodnot za p posledních kroků. Heuristika námi označovaná TA maximalizuje:

$$\Delta_j = \left(\frac{\partial F}{\partial x_j}(s_1(r_1(z_1)), \ldots, s_m(r_m(z_m)))\right)^{-} \cdot (s_j(r_j(z_j - p)) - s_j(r_j(z_j)))$$

V experimentech heuristiku referujeme *Güntzen*.

Alternativní heuristikou, značíme *Gurský*, navrženou Gurským & all. v [8] na základě experimentů a intuice, je nahradit rozdíl hodnot $(s_j(r_j(z_j - p)) - s_j(r_j(z_j)))$ přímo hodnotou $s_j(r_j(z_j))$. Další alternativou navrženou v [8] je výše uvedené heuristiky střídat.

3 Pravděpodobnostní model

Označíme $P(s_i(x) \geq r)$ pravděpodobnost, že se hodnota i–tého atributu objektu x větší nebo rovná hodnotě r. Vzhledem k tomu, že možné hodnoty r bereme z reálných čísel, je tato pravděpodobnost rovnosti konkrétní hodnotě často nulová, proto zavedeme distribuční funkci $p_i(s_i(x) = r) = \frac{\partial P(s_i(x) \geq r)}{\partial r}[r]$. Přijměme relativně silný předpoklad, že jednotlivé atributy jsou na sobě nezávislé, tj. pravděpodobnost sdruženého jevu $s_i(x) \geq r_i$ a zároveň $s_j(x) \geq r_j$ se rovná součinu těchto pravděpodobností pro libovolné kombinace jednotlivých sloupců.

Na základě znalosti distribučních funkcí p_j a tohoto předpokladu jsme schopni spočítat pravděpodobnostní rozložení hodnot s_i pro libovolný sloupec.

3.1 Potvrzení maxima

Předpokládejme, že jsme sekvenčními přístupy v jednotlivých sloupcích sestoupili na pozice $\langle z_1, \ldots, z_m \rangle$ a že objekt s maximální hodnotou $S()$ je $x_{\max}$. V okamžiku prvního sekvenčního přístupu na $x_{\max}$ ještě nemáme jistotu, že se mezi nenavštívenými prvky nenachází objekt lepší, proto musíme ještě dále prohledávat, dokud nedojdeme na taková $\langle z_1^{(p)}, \ldots, z_m^{(p)} \rangle$, že hodnoty s_i na těchto pozicích dají dohromady méně než $S(x_{\max})$, tj.

$$F(s_1(r_1(z_1^{(p)})), \ldots, s_m(r_m(z_m^{(p)}))) \leq S(x_{\max}). \tag{1}$$

Snažíme se minimalizovat očekávaný počet sekvenčních kroků, tj. hledáme $\min \sum_j (z_j^{(p)} - z_j)$, kde minimalizujeme přes m–tice $\langle z_1^{(p)}, \ldots, z_m^{(p)} \rangle$ splňující podmínku (1) a pro každé j je $z_j^{(p)} \geq z_j$.

Podobně jako heuristiky z literatury přijmeme zjednodušení, které dovolí prohledávat jen v jednom sloupci. Během prohledávání se nejlepší sloupec vždy přepočítává, tedy v průběhu výpočtu se může měnit.

Takto zjednodušený problém už můžeme vyřešit jednoduchým výpočtem. Pro každý sloupec j hledáme takové $z_j^{(t)}$, aby očekávaná hodnota $s_j(r_j(z_j^{(t)}))$ zajišťovala "podlezení maxima", tj. aby

$$F(s_1(r_1(z_1)), \ldots, s_j(r_j(z_j^{(t)})), \ldots, s_m(r_m(z_m))) \leq S(x_{\max}),$$

a vybereme sloupec s minimální pravděpodobností, že je hodnota s_j mezi hodnotami $s_j(r_j(z_j))$ a $s_j(r_j(z_j^{(t)}))$.

Pokud navíc předpokládáme funci F v jednoduchém tvaru lineární kombinace jednotlivých atributů s váhami $w_1, \ldots, w_m$, tj. $w_j = \frac{\partial F}{\partial s_j}$ a $F(s_1(r_1(z_1)), \ldots, s_m(r_m(z_m))) = \sum_{j=1}^{m} w_j \cdot s_j(r_j(z_j))$, můžeme potřebné $s_j(r_j(z_j^{(t)}))$ vypočítat:

$$s_j(r_j(z_j^{(t)})) = s_j(r_j(z_j)) - \frac{\sum_{j=1}^{m} w_j \cdot s_j(r_j(z_j)) - S(x_{\max})}{w_j}$$

tedy

$$s_j(r_j(z_j)) - s_j(r_j(z_j^{(t)})) = \frac{\sum_{i=1}^{m} w_i \cdot s_i(r_i(z_i)) - S(x_{\max})}{w_i} \tag{2}$$

Čitatel zlomku nezávisí na j, proto ho označíme $const = \sum_{i=1}^{m} w_i \cdot s_i(r_i(z_i)) - S(x_{\max})$ a ve vhodnou chvíli budeme ignorovat úplně.

Pokud známe pravděpodobnost, že záznam má hodnotu s_j v intervalu $\langle s_j(r_j(z_j)), s_j(r_j(z_j^{(t)}))\rangle$ a počet ještě nenavštívených záznamů, známe i očekávaný počet záznamů, které musíme sekvenčně navštívit do potvrzení maxima.

Obecné rozložení Pravděpodobnost hodnoty s_j z intervalu $\langle s_j(r_j(z_j)), s_j(r_j(z_j^{(t)}))\rangle$ samozřejmě záleží na rozložení jednotlivé veličiny, jsou–li veličiny závislé, tak i na hodnotách ostatních veličin.

V našich experimentech jsme na základě znalosti náhodného rozložení vygenerovali vzorek 1000 hodnot a odhadli pravděpodobnost. Při znalosti funkčního popisu rozložení by bylo možné odvodit pravděpodobnost symbolicky, nicméně simulace je použitelná vždy. Odhad pravděpodobnostního rozložení můžeme získat např. z předchozích dotazů. I v průběhu prohledávání podle sloupce L_j získáváme informace o rozložení hodnot v ostatních sloupcích různých od L_j.

Rovnoměrné rozložení Předpokládáme–li pravděpodobnost hodnot rovnoměrnou, je:

$$P(s_j \geq threshold) = \frac{s_j(r_j(z_j)) - s_j(r_j(z_j^{(t)}))}{(s_j(r_j(z_j)) - 0}$$

a po dosazení (2) do čitatele dostaneme:

$$P(s_j \geq threshold) = \frac{const}{w_j} \cdot \frac{1}{(s_j(r_j(z_j)) - 0}$$

Očekávanou počet záznamů spočteme vynásobením pravděpodobnosti počtem v daném slouci nenavštívených záznamů $N - z_j$.

Pokud navíc převedeme minimalizaci pravděpodobnosti na maximalizaci převrácené hodnoty, dostaneme výraz v jazyce heuristik Sekce 2.4:

$$\max_{j=1\ldots m} \left\{ \frac{1}{(N - z_j)} \cdot \frac{\partial F}{\partial x_j} \cdot x_j \right\}. \tag{3}$$

Od heuristiky intuitivně odvozené v [8] se liší pouze o činitel $\frac{1}{N-z_j}$, který se uplatní jen při velkých rozdílech v z_j v jednotlivých sloupcích.

3.2 Nalezení a potvrzení

Pokud chceme maximum nalézt a potvrdit, tak by bylo ideální nejdřív uhodnout hodnotu maximálního prvku $S(x_{\max})$ a pak v průběhu potvrzování této hodnoty na daný prvek musíme narazit.

Samozřejmě, že hodnotu maximálního prvku se dovíme teprve, až ho poprvé sekvenčním přístupem navštívíme, nicméně jeho hodnotu můžeme odhadnout na základě pravděpodobnostních rozložení hodnot jednotlivých atributů.

Předpokládejme, že jsme dosud neviděli N záznamů. Za odhad hodnoty maximální hodnoty prvku zvolíme takovou hodnotu, že na 50% existuje objekt s hodnotou rovnou či vyšší. Samozřejmě můžeme volit i jinou hodnotu než 50%, ale 50% odpovídá mediánu, tj. "prostřední" hodnotě, což je dobrý bodový odhad pravděpodobnostního rozložení.

Narozdíl od odstavce 3.1 pro maximum je důležitá souhra hodnot v jednotlivých atributech, tj. je lepší provádět simulaci přes všechny atributy zároveň.

Pravděpodobnost, že aspoň jeden objekt z N má hodnotu vyšší než t spočteme jako jedna mínus pravděpodobnost, že všechny objekty mají hodnotu nižší. Formálně to lze zapsat:

$$P(\max_{N \text{ cases of } \hat{x}} F(\hat{x}) \geq f_0) = 1 - \left[\int_{x_1,\ldots,x_n} \delta_{F(x_1,\ldots,x_n)<f_0} \cdot \Pi_{i=1,\ldots,n} p(X_i = x_i) \right]^N$$

Odtud odvodíme, jak malou horní část rozložení na F máme brát, konkrétně

$$\alpha_N = 1 - [1 - \alpha]^{\frac{1}{N}}$$

pro $\alpha = 0.5$ tedy $\alpha_N = 1 - [0.5]^{\frac{1}{N}}$. Pro $N = 1000$ dostáváme $\alpha_N = 0.000692907$, tj. při skoro rovnoměrném rozložení hodnot veličin můžeme za odhad vzít absolutní maximum funkce S, např. nabývá–li veličina hodnot od nuly do jedné, za odhad bereme 1 nezávisle na hodnotě prvního navštíveného objektu.

Pravděpodobnostní rozložení na F můžeme v jednoduchých případech řešit přímým výpočtem, ale vždy můžeme simulací získat rozumný odhad. Pokud jsme vygenerovali N_V vzorků, vezmeme za odhad ten $\alpha_N \cdot N_V$–tý největší.

3.3 Nalezení maxima

Uvažujme jeden konkrétní sloupec. První záznam má jistě větší šanci být maximem než záznam druhý (a další), protože má vyšší hodnotu v tomto sloupci

a o ostatních sloupcích nevíme ani v jednom případě nic. Chceme–li maximum nalézt na nejmenší počet sekvenčních přístupů, měli bychom hledat v tom sloupci, kde je největší pravděpodobnost, že první sekvenčně navštívený záznam má maximální hodnotu.

V obecném případě musíme vypočíst či aproximovat pravděpodobnostní rozložení hodnot v jednotlivých sloupcích a vybrat ten nejlepší.

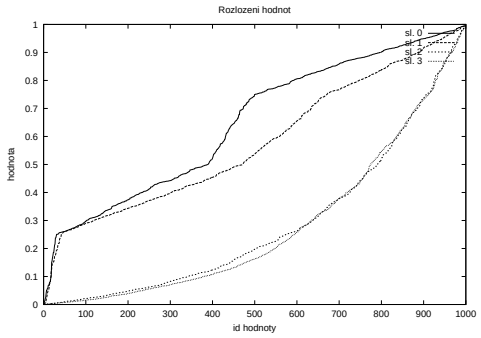
Pokud ale je funkce F lineární kombinací jednotlivých atributů a atributy jsou na sobě nezávislé, situace se podstatně zjednoduší. Předně, součet složek ve funkci F lze zaměnit se součtem přes jednotlivé proměnné, tj. sečíst příspěvky jednotlivých proměnných a tak získat odhad celkové hodnoty.

Pro každý sloupec tedy nejprve zjistíme jeho očekávanou hodnotu, $E(s_i) = \int P(s_i) \cdot s_i$. Pravděpodobnostní rozložení funkce F ve sloupci j pak bude dáno váženým součtem středních hodnot ostatních sloupců plus váhou vynásobenou střední hodnotou *maximálního* prvku v daném sloupci, pro zjednodušení ho odhadneme $s_j(r_j(z_j))$, pokud jsme už ve sloupci nějaký záznam viděli, jinak maximální hodnotou s_j. Pro porovnání dvou sloupců nám tedy stačí jejich střední hodnoty a maximální hodnoty, protože příspěvky ostatních sloupců jsou pro oba stejné. Proto k prohledávání vybíráme sloupec s (váženým) maximálním rozdílem mezi maximální a očekávanou hodnotou, tj. maximalizujeme:

$$\max_{j=1\ldots m} \{(s_j(r_j(z_j)) - E(s_j)) \cdot w_j\}\,.$$

4 Experimenty

Experimenty jsme prováděli na simulovaných datech, která měla silně nerovnoměrné rozložení hodnot jednotlivých atributů. První dva atributy měly po částech rovnoměrné rozložení, druhé dva exponenciální a výrazně méně časté vysoké hodnoty. Distribuční funkce jednotlivých atributů je zobrazena na obrázku 1.



Obr. 1. Hodnoty jednotlivých atributů pro 1000 prvních datových řádků. Jednotlivé atributy byly na sobě nezávislé.

Výsledky experimentů jsou shrnuty v tabulce na obrázku 2. Jednotlivé testy jsme prováděli na stejných datech, ale s různou funkcí F: vždy šlo o lineární kombinaci atributů, měnily se váhy. První sloupec obsahuje váhy atributů daného experimentu, následující čtyři sloupce se týkají úlohy *najít a potvrdit maximum*, poslední tři úlohy maximum *pouze najít*, bez jeho potvrzení.

	potvrdit				nalezt		
váhy	náš	Güntzen	Gurský	sl.	sl.	nalez. heur.	nalez. op.
$\langle 1,9,0,3\rangle$	**284**	363	313	2 (1)	1	21	21
$\langle 3,9,0,1\rangle$	**223**	224	262	1	1	21	21
$\langle 9,1,0,3\rangle$	**249**	524	289	2	0	65	65
$\langle 9,3,0,1\rangle$	715	**713**	891	0 (2)	0	535	**21**
$\langle 3,0,9,1\rangle$	**94**	95	103	1	1	7	7
$\langle 9,0,1,3\rangle$	**250**	525	291	2	0	18	18
$\langle 9,0,3,1\rangle$	**354**	680	405	1	0	18	18

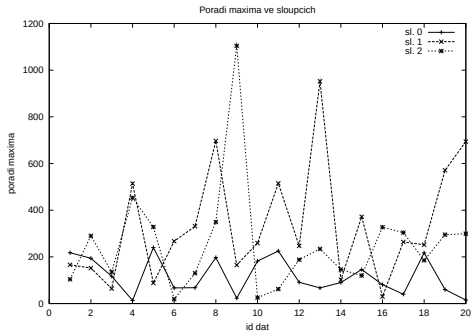
Obr. 2. Hodnoty jednotlivých atributů pro 1000 prvních datových řádků. Jednotlivé atributy byly na sobě nezávislé.

Námi navrhovaný algoritmus založený na pravděpodobnostním rozložení byl nejlepší ve většině případů (2. sloupec). Pro srovnání jsou v tabulce uvedeny počty sekvenčních přístupů dvou nejlepších heuristik z literatury (3. a 4. sloupec), které nám poskytl P. Gurský.

U úlohy nalezení maxima je snadné najít optimální řešení – nejlepší je procházet ten sloupec, ve kterém se maximální prvek vyskytuje nejdříve. Naše heuristika doporučila skoro vždy nejlepší sloupec, v jednom případě se diametrálně lišila od optima (prohledala 535 záznamů místo optimálních 21). To je dáno stochastickou povahou úlohy – maximum v této úloze na většině dat s daným rozložením je nejblíže ve sloupci 0, který doporučujeme prohledávat, ale právě v těchto datech je daleko lepší přistupovat přes sloupec 2. Pro ověření tohoto vysvětlení jsme generovali 20 různých vzorků dat ze stejného pravděpodobnostního rozložení. Pozice maximálního prvku v jednotlivých sloupcích na jednotlivých datech jsou zobrazeny na obrázku 3. Tento experiment potvrdil hypotézu – většinou je pořadí v 0. sloupci nejnižší, ale v některých případech vítězí jiný sloupec.

5 Závěr

Porovnali jsme v literatuře uváděné heuristiky daného problému s pravděpodobnostně optimálním řešením (za zjednodušujících předpokladů). Prohledávání založené na znalosti pravděpodobnosti dává lepší výsledky než heuristiky, nicméně se od heuristik řádově neliší.



Obr. 3. Pozice maxima v jednotlivých sloupcích silně závisí na konkrétních datech (tj. na náhodě). Graf ukazuje pozice maxima v různých sloupcích pro 20 vzorků dat.

Poděkování

Tato práce byla podporovaná výzkumným záměrem MS002162038 a projektem Informační společnost 1ET100300517. Děkujeme také P. Gurskému za zpracování dat algoritmy z [5] a [8] pro porovnání s našimi výsledky.

Reference

1. R. Fagin, Combining Fuzzy Information from Multiple Systems. In In 15th ACM Symposium on Principles of Databases Systems, Montreal, 1996, 216–226
2. R. Fagin, Combining Fuzzy Information from Multiple Systems. J. Comput. System Sci. 58, 1999, 83–99
3. R. Fagin, Combining Fuzzy Information: An Overview. ACM SIGMOID Record 31, Database Principles Column, June 2002, 109–118
4. R. Fagin, A. Lotem, and M. Naor, Optimal Aggregation Algorithms for Middleware. In In Proc. 20th ACM Symposium on Principles of Database Systems, 2001, 102–113
5. U. Güntzer, W. Balke, and W. Kiessling, Optimizing Multi-Feature Queries for Image Databases. In Proceedings of the 26th VLDB Conference, Cairo, Egypt, 2000.
6. A. Natsev, Y-C. Chang, J.R. Smith, C-S. Li, and J.S. Vitter, Supporting Incremental Join Queries on Ranked Inputs. In Proc. 27th Very Large Databases Conference, Rome, Italy, 2001, 281–290
7. S. Nepal and M. V. Ramakrishna, Query Processing Issues in Image (Multimedia) Databases. In Proc. 15th International Conference on Data Engineering, 1999, 22–29
8. Gurský P., Lencses R., and Vojtáš P., Algorithms for User Dependent Integration of Ranked Distributed Information. Technical report, Institute of Informatics, P.J. Šafárik University, Košice, 2004.
9. J. Pokorný and P. Vojtáš, A Data Model for Flexible Querying. In A. Caplinskas and J. Eder, editors, Proc. ADBIS'01, volume 2151 of Lecture Notes in Computer Science, Berlin, 2001, Springer Verlag, 280–293

Zásobníkové automaty s orákulem*

Michal Žemlička

Univerzita Karlova, Matematicko-fyzikální fakulta,
Katedra softwarového inženýrství,
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
michal.zemlicka@mff.cuni.cz

Abstrakt Klasické zásobníkové automaty kódují výhled do stavů. To je výhodné zejména pro analýzu zdola. V uvedeném případě tak nedochází ke ztrátě rychlosti – i když za cenu významného nárůstu velikosti analyzátorů. Pro potřeby analýzy shora či téměř shora je kódování výhledu do stavů nevyhovující, či přinejmenším musí být na vybraných místech obcházeno.
Pokusili jsme se v tomto příspěvku zobecnit práci s výhledem a představujeme dva modely výběru správného přechodu zásobníkového automatu: Zásobníkové automaty s orákulem a zásobníkové automaty s výhledem. Tyto modely jsou dvěma různými úrovněmi zobecnění téhož přístupu, a proto je představujeme současně.
Ukážeme, že použití obecného orákula může vést až za hranice bezkontextových jazyků. Dále ukážeme, že zásobníkové automaty s výhledem (ať již nejvýše k nebo právě k) jsou ekvivalentní klasickým zásobníkovým automatům.
Použití automatů s výhledem nejvýše k umožňuje pro většinu běžných jazyků výrazně zredukovat jejich prostorovou náročnost. Využití automatů s výhledem nejvýše k pro analýzu shora či téměř shora umožňuje získat přehledný a srozumitelný kód velmi blízký ručně psaným analyzátorům rekurzivním sestupem.

1 Úvod

Standardní model zásobníkových automatů (ZA, viz např. [1], [2]) pracuje s výhledem tak, že jej zakóduje do stavů ZA (viz např. obrázky 6.7 a 6.8 v [3]). Tento model je výhodný pro teoretickou analýzu vlastností zásobníkových automatů, ale neodráží jejich implementaci v překladačích – zejména v případě kompilátorů s analýzou rekurzivním sestupem. Dosud používaný model neodpovídá intuici, že analýza výhledu je proces rozhodující, který přechod je v dané konfiguraci možný. Konfigurací se rozumí aktuální stav, nepřečtená část vstupu a obsah zásobníku. V praxi většinou využíváme jen stav, prefix nepřečtené části vstupu a vrchol zásobníku. Správný přechod bývá určen analýzou výhledu o délce k symbolů (typicky $k \geq 1$). Také připomeňme, že pro programovací jazyky je zapotřebí zkoumat výhled delší než 1 pouze velmi vzácně.

* Tento výzkum byl částečně podporován programem "Informační společnost" jako projekt 1ET100300517.

Problémem výběru vhodného přechodu se budeme zabývat na dvou úrovních: obecnější (zásobníkový automat s orákulem, ZAO) a praktičtější (zásobníkový automat s výhledem, ZAV).

Je smysluplné se zabývat výběrem vhodného přechodu pouze v konfiguracích, kdy je možné aplikovat pro různé výhledy více různých přechodů. Také se zdá rozumné, zkoumat jen takovou část výhledu, která je nezbytná pro uvedené rozhodnutí. V běžných programovacích jazycích je jen velmi málo míst, kde je třeba analyzovat výhled delší než 1. Jinde vystačíme s výhledem kratším. Na mnoha místech dokonce není třeba rozhodovat (a tedy zkoumat výhled) vůbec. Toto nám dovoluje držet velikost analyzátorů i pro jazyky vyžadující výhled větší než 1 v rozumných mezích bez toho, že bychom zhoršili efektivitu analýzy.

Myšlenka zkoumání výhledu může být zobecněna na obecné orákulum, tj. na něco (černou skříňku), co provádí příslušná rozhodnutí. Orákulum pak může být implementováno například pomocí konečného automatu nebo jako nástroj komunikující s uživateli či jinými aplikacemi nebo automaty. V takovém případě může být koncept orákula použit jako teoretický základ pro popis některých systémů orientovaných na služby [4].

Jiné využití analýzy s orákulem vidíme v lingvistice, kdy se může stát, že analyzátor není schopen správně zpracovat předložený text (např. proto, že daná konstrukce je nová či vzácná či z jiného důvodu není ošetřena tvůrci analyzátoru), a tak může být rozumné v dané situaci spolupracovat s uživatelem (nechat jej správné řešení vybrat). Jsou-li tyto situace dostatečně vzácné, může to být dokonce řešení optimální. Zásahy uživatele mohou být následně využity pro případné učení analyzátoru. Je také možné, aby analyzátor volal (je-li to třeba) specializované aplikace řešící specifické lingvistické úlohy tak, jak je to obvyklé pro spolupráci služeb v systémech orientovaných na služby. Využití orákula v oblastech softwarového inženýrství a komputační lingvistiky je předmětem dalšího výzkumu.

2 Zásobníkový automat s orákulem

Zásobníkový automat s orákulem je (nedeterministický) akceptor složený z orákula, řídící jednotky, čtecí hlavy pohybující se jedním směrem po vstupní pásce a zásobníku. Jeho výpočet je řízen množinou přechodů. Z množiny přechodů vybírá pro danou konfiguraci orákulum množinu přechodů aktuálně přípustných.

Definice 1 (Zásobníkový automat s orákulem) Zásobníkový automat s orákulem *(ZAO) je uspořádaná dvojice* $A = (P, O)$*, kde* $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ *je zásobníkový automat (dále nazývaný* základní automat*) a* orákulum O *je funkce, která pro každou konfiguraci* C *základního automatu* P *vrací* množinu přípustných přechodů $\delta'(C)$*. Je-li* $C = (q, w, \alpha)$*, kde* $q \in Q$*,* $w \in \Sigma^*$*, a je-li* $\alpha \in \Gamma^*$*, definujeme* $\delta(C) = \{(q, a, Z) \to (q', \beta) \in \delta | \exists w' : aw' = w\}$*. Pak* $O(C, \delta(C)) = \delta'(C) \subseteq \delta(C)$*.*

Pohyb *zásobníkového automatu s orákulem je dán přechody základního automatu, kde pro každou konfiguraci* C *jsou orákulem vybrány pouze pohyby z* $\delta'(C)$*.*

Intuitivně je P (nedeterministický) zásobníkový automat bez výhledu. P v A se v každé konfiguraci poradí s orákulem, jaké přechody jsou v dané konfiguraci možné.

Poznámka 2 *Může se zdát, že parametr $\delta(C)$ orákula je redundantní, jelikož může být kdykoliv spočten z C. Oddělení tohoto výpočtu od orákula umožňuje zkonstruovat orákulum méně závislé na základním automatu.*

Konvence 3 *Je možné psát, že ZAO $A = (Q, \Sigma, \Gamma, \delta, O, q_0, Z_0, F)$, místo $A = (P, O)$, kde $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$.*

Konvence 4 *V případě, že orákulum pracuje deterministicky pouze na základě stavu, prefixu nepřečtené části vstupu a části zásobníku, můžeme zapisovat prefix nepřečtené části vstupu využitý orákulem k výpočtu do hranatých závorek ('[',']') hned za čtený symbol. Podobně budeme zapisovat do hranatých závorek použitý téměř prefix zásobníku hned za vrchol zásobníku.*

Abychom zjednodušili popis přechodů poblíž konce vstupní pásky, předpokládáme, že vstup je na konci doplněn zarážkou $\$^k$ (používáme augmentovanou gramatiku).

Značení: Prefix řetězce délky k značíme $First_k$.

Poznámka 5 *Orákulum nemusí být závislé jen na základním automatu a jeho konfiguraci. Může využívat i interakci s reálným světem (např. s uživateli). Nemusí být ani deterministické. Zde se však omezíme na deterministické chování orákula.*

Vrací-li orákulum pro každou konfiguraci nejvýše jeden přechod, ZAO je deterministický. Orákulum s výhledem k je takové orákulum, které ke svým výpočtům využívá pouze k symbolů výhledu, stavu základního automatu a vrcholu zásobníku. Formálně $O(C, \delta(C)) = O(C_1, \delta(C_1))$, kde $C = (q, w, \alpha)$ a $C_1 = (q, a_1...a_k, Z)$ pro nějaké $q \in Q$, $w \in \Sigma^*$, $\alpha \in \Gamma^*$, $a_1...a_k = First_k(w \cdot \$^k)$ a $Z = First_1(\alpha)$.

Využívá-li orákulum ke svému rozhodování pouze konfiguraci, je deterministické (pracuje deterministicky; může však stále vracet i víceprvkovou množinu přípustných přechodů), a tak může být specifikováno jako zobrazení

$$O : Q \times \Sigma^* \times \Gamma^* \to 2^{Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to Q \times \Gamma^*}.$$

V případě, že orákulum používá pouze omezenou část konfigurace, je možné používat jako vstup orákula pouze tuto omezenou část konfigurace. (Příklad: LL(k) analýza – orákulum využívá pouze stav a prvních k symbolů nepřečtené části vstupu.)

Definice 6 (Konfigurace s výhledem k) *Pro každou konfiguraci $C = (q, w, \alpha)$ můžeme definovat* konfiguraci s výhledem k $K = (q, First_k(w), First_1(\alpha))$.

Příklad 7 *Mějme ZAO, jehož orákulum využívá ke svému rozhodování pouze konfiguraci s výhledem k* $K = (q, a_1..a_k, Z)$*. Pohyb ZAO z dané konfigurace do stavu* q'*, čtení symbolu* a_1 *a nahrazení vrcholu zásobníku řetězcem* α *budeme zapisovat*

$$(q, a_1[a_2...a_k], Z) \to (q', \alpha).$$

Případ, kdy není čten žádný symbol, zapíšeme takto:

$$(q, [a_1...a_k], Z) \to (q', \alpha).$$

3 Příklady orákula

Podívejme se, jak mohou pracovat různá orákula:

Triviální: Orákulum pro žádnou konfiguraci vůbec neomezuje množinu použitelných přechodů (nic nedělá). Budeme jej značit Tr.

Klasické: Orákulum pro každou konfiguraci vybere nejvýše jeden přechod v závislosti na stavu, vrcholu zásobníku a právě k prvních dosud nepřečtených symbolů ze vstupu. To odpovídá klasické deterministické analýze (např. LL(k), LR(k)). Toto orákulum nazveme orákulem s výhledem k.

Definice 8 (Sourozenecké automaty) *Říkáme, že dva ZAO* $A = (P_A, O)$*,* $B = (P_B, O')$ *jsou* sourozenecké*, pokud* $P_A = P_B$*. Také říkáme, že* A *je sourozencem* B *a* B *je sourozencem* A*.*

Lemma 9 *Mějme ZAO* $A = (P, Tr)$*. Pak* A *a* P *procházejí týmiž konfiguracemi a rozpoznávají týž jazyk.*

Důkaz. V obou případech je počáteční konfigurace táž: (q_0, w, Z_0). Tr neomezuje původní množinu přechodů. Proto ZAO má v každé konfiguraci původní množinu přechodů. Oba automaty tedy procházejí týmiž konfiguracemi a rozeznávají týž jazyk.

Definice 10 (Zjemnění orákula) *Mějme dva sourozenecké ZAO* $A = (P, O)$ *a* $A' = (P, O')$*. Říkáme, že* O je zjemněním O'*, pokud pro každou konfiguraci* C *základního automatu* P $O(C) \subseteq O'(C)$*. Také říkáme, že* A je zjemněním A'*.*

Poznámka 11 *Orákulum může být libovolně silné – můžeme se tak dostat za hranice bezkontextových jazyků.*

Příkladem může být orákulum vybavené zásobníkem. Má-li toto orákulum přístup k zásobníku automatu, je možné rozpoznávat i jazyk $a^n b^n c^n$*. Stačí, když si orákulum "spočítá" b, a "donutí" automat, aby odpočítal stejný počet c.*

4 Zásobníkové automaty s výhledem

Zaměřme se na deterministická orákula, která ke svému výpočtu využívají pouze stav, omezenou část prefixu nepřečtené části vstupu a vrchol zásobníku a která vybírají z možných přechodů nejvýše jeden. Omezme proto zásobníkové automaty s orákulem na zásobníkové automaty s výhledem.

Definice 12 (Zásobníkový automat s výhledem) *Říkáme, že zásobníkový automat s orákulem je* zásobníkovým automatem s výhledem právě *k (ZAV!k), je-li jeho orákulum deterministické a pro každý stav, vrchol zásobníku a k-prefix nepřečtené části vstupu určí nejvýše jeden přechod.*

Obdobně říkáme, že zásobníkový automat s orákulem je zásobníkovým automatem s výhledem nejvýše *k (ZAV≤k), pokud jeho orákulum na základě stavu, vrcholu zásobníku a prefixu nepřečtené části vstupu o délce nejvýše k deterministicky vybere nejvýše jeden přechod.*

Říkáme, že zásobníkový automat s výhledem nejvýše k je minimální, *využívá-li jeho orákulum nejmenší prefix nepřečtené části vstupu nezbytný pro výběr jediného přechodu.*

Poznámka 13 *Každý ZAV!k je ZAV≤k.*

Konvence 14 *Nebude-li právě důležité rozlišovat ZAV!k a ZAV≤ k, budeme psát ZAV.*

Může být užitečné rozšířit pojem přechod pro ZAV tak, aby bylo možné rozlišit stejné přechody pro různé výhledy.

Úloha výběru přechodu s použitím výhledu se liší od ostatních úloh řešených Moorovými stroji [5]: zde není třeba přečíst celý vstup, aby se ověřilo, zda patří do daného jazyka; úkol je vybrat variantu výpočtu mající naději úspěšně dokončit výpočet.

Na optimalitu orákula můžeme mít rozdílné požadavky. Je možné "vypsat" všechny možné výhledy a převést tento seznam na strom (viz **a)** v obr. 1). Je možné minimalizovat strom tak, aby byly použity jen nejkratší možné výhledy dostatečné pro rozlišení jednotlivých přechodů – tj. odpovídající ZAV≤k je minimalizován (případ **b)** v obr. 1). Můžeme rezignovat na požadavek, aby orákulum mělo tvar stromu (který usnadňuje rozšiřování analyzátoru) a použít automat se strukturou acyklického orientovaného grafu (**c)** v obr. 1). Orákulum může být optimalizováno na velikost, což povede k řešením s nejmenším počtem stavů jako v případě **d)** obr. 1.

Uvedená diskuse naznačila, že jednoduchá aplikace obvyklé minimalizace automatů nemusí být tím nejlepším řešením. Mnohdy bychom došli k něčemu jinému, než je požadováno v def. 12. Výběr "optimálního" typu orákula záleží na očekávaném použití analyzátoru.

Příklad 15 (Různé přístupy k zacházení s výhledem) *Uvažujme situaci, kdy je třeba vybrat pomocí výhledu z několika přechodů vycházejících z daného stavu. Tato úloha může být řešena pomocí acyklického konečného automatu. Mějme přechody:*

$t_1 = (q, [adk], Z) \to (q_1, Z)$,
$t_2 = (q, [ael], Z) \to (q_2, Z)$,
$t_3 = (q, [bgm], Z) \to (q_3, Z)$,
$t_4 = (q, [cel], Z) \to (q_1, Z)$,
$t_5 = (q, [cfn], Z) \to (q_4, Z)$.

Mohou být použita přinejmenším 4 základní řešení uvedená na obr. 1:

1. *Automat se strukturou stromu testující celé dostupné výhledy délky k. Zkoumá výhled délky 3 a má 14 stavů.*
2. *Automat se strukturou stromu testující výhledy minimální délky dostatečné pro rozlišení uvedených přechodů. Zkoumá výhled 1-2 symboly a má 8 stavů.*
3. *Automat se strukturou acyklického orientovaného grafu testující výhledy minimální délky dostatečné pro rozlišení uvedených přechodů. Zkoumá výhled 1-2 symboly a má 7 stavů.*
4. *Automat s minimálním testováním. Využívá výhled délky 3 a má 7 stavů. Je možné testovat pouze 3. symbol, což zjednoduší test.*

Všechny uvažované přechody jsou ε-přechody, a proto je možné označit koncové stavy automatu implementujícího orákulum stavy, kam se má základní automat dostat, místo toho, abychom použili jako značky celé přechody.

Definice 16 (Výhled pro daný přechod) *Pro každý přechod t ZAV$\leq k$ definujeme*

$$lookahead(t) = \begin{cases} a_1 a_2 \dots a_m & \text{for } t = (q, a_1[a_2 \dots a_m], Z) \to (q', \alpha) \\ a_1 \dots a_m & \text{for } t = (q, [a_1 \dots a_m], Z) \to (q', \alpha) \end{cases}$$

kde $m \leq k$.

Lemma 17 *Ke každému zásobníkovému automatu s výhledem nejvýše k A existuje sourozenecký zásobníkový automat s výhledem právě k A′ takový, že A′ je zjemněním A a $L(A') = L(A)$.*

Důkaz. Doplněním výhledů přesně na délku k pomocí $(\Sigma \cup \$)^*$ získáváme hledaný automat.

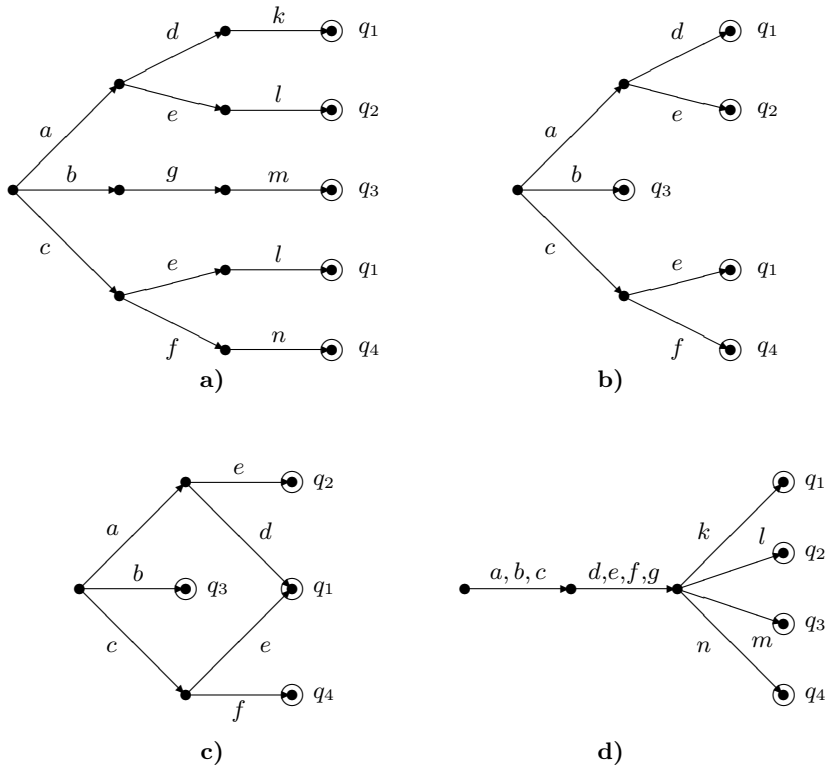
Takto získaný automat je ve většině případů možné ještě dále zjemnit spočtením skutečných výhledů.

Věta 18 *Ke každému ZAV$\leq k$ A existuje sourozenecký minimální ZAV$\leq k$ A′ takový, že A je zjemněním A′ a $L(A') = L(A)$.*

Důkaz této věty je uveden až za algoritmem 31.

Definice 19 (Přijatelná konfigurace) *Nechť $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ je ZA(V). Říkáme, že konfigurace $C = (q, w, Z)$ je* přijatelná *P, pokud $\exists C' = (q', \varepsilon, \alpha)$, kde $q' \in F$ takové, že $C \vdash^* C'$.*

AC(P) označuje všechny přijatelné konfigurace P.



Obr. 1. Různé automaty pro týž výhled.

Definice 20 (Situace (analýzy)) *Nechť $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ je ZA. Dvojice $(q, Z) \in Q \times \Gamma$ nazveme* situací *ZA P.*

Definice 21 *Všechny přechody začínající v situaci $s = (q, Z)$ nazýváme (q, Z)-*přechody *(nebo s-*přechody*).*

Definice 22 *$Clean_\Sigma$ je homomorfismus definovaný na $(\Sigma \cup \{\circ, \$\})^*$, $\{\circ, \$\} \cap \Sigma = \emptyset$ takový, že*

$$Clean_\Sigma(a) =_{df} \begin{cases} a \ \forall a \in \Sigma \\ \varepsilon \ \forall a \notin \Sigma \end{cases}$$

Definice 23 (Možné řetězce výhledu) *Nechť $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ je ZA a $s = (q, Z)$ je libovolná situace analýzy P. Množina všech možných řetězců výhledu délky nejvýše k situace analýzy s je*

$$LAS_{P,\leq k}(s) = \left\{ a_1 ... a_j \in \bigcup_{0 \leq i < j \leq k} \Sigma^i \$^{j-i} \mid (q, Clean(a_1 ... a_j), Z) \in AC(P) \right\}$$
$$\cup \left\{ a_1 ... a_j \in \bigcup_{0 \leq j \leq k} \Sigma^j \mid \exists w \in \Sigma^* : (q, a_1 ... a_j w, Z) \in AC(P) \right\}$$

Množina všech možných řetězců výhledu délky právě k situace analýzy s je

$$LAS_{P,!k}(s) = \left\{a_1...a_k \in \bigcup_{0\leq i<k} \Sigma^i\$^{k-i} \mid (q, Clean(a_1...a_k), Z) \in AC(P)\right\} \\ \cup \left\{a_1...a_k \in \Sigma^k \mid \exists w \in \Sigma^* : (q, a_1...a_k w, Z) \in AC(P)\right\}$$

Mnohé věci se snáze představují v grafické podobě. Zacházení s výhledy si proto ukážeme na výhledových stromech.

Definice 24 (Stromová reprezentace množiny řetězců) *Nechť Δ je množina symbolů (abeceda) a S je množina řetězců nad Δ. Nechť $T = (V, E)$ je strom. Říkáme, že T je* stromová reprezentace *S, existuje-li značení $l : E \to \Delta \cup \{\$\}$ takové, že spojením značek podél cesty p od kořene do listu (nazýváme takovou cestu* úplnou*) dává řetězec r z S. (Pak říkáme, že r je reprezentován p.) Hrany vycházející z libovolného uzlu jsou ohodnoceny různými symboly. Navíc by mělo platit, že různé úplné cesty v T reprezentují různé řetězce z S a že existuje bijekce mezi cestami od kořene k listu v T a mezi řetězci v S.*

Definice 25 (Výhledový strom, navigační strom) *Stromovou reprezentaci T množiny řetězců výhledu délky k pro situaci s nazýváme* výhledovým stromem pro *s.*

Jsou-li navíc uzly výhledového stromu T pro s označeny množinami přechodů z P, které jsou pro situaci s a pro část výhledu odpovídající cestě od kořene do daného uzlu použitelné, nazýváme takový strom navigačním pro *s.*

Definice 26 (Hloubkově minimální orákulum a výhled) *Říkáme, že orákulum je* hloubkově minimální*, používá-li pouze nejkratší prefix nepřečtené části vstupu nezbytný pro jednoznačný výběr vhodného přechodu. Výhled hloubkově minimálního orákula (nebo navigační strom jej reprezentující) nazýváme také hloubkově minimální.*

Algoritmus 27 (Minimalizace navigačního stromu)
Vstup: Navigační strom T;
Výstup: Hloubkově minimalizovaný navigační strom T;
Method:

```
BEGIN
    WHILE existuje nelistový uzel n označený singletonem DO
        Smaž podstrom zakořeněný v n s výjimkou n samotného;
END;
```

Lemma 28 *Algoritmus 27 je konečný.*

Důkaz. Algoritmus v každém kroku vypustí z konečného grafu alespoň jeden vrchol, musí se proto po konečném počtu kroků zastavit.

Lemma 29 *V každém kroku alg. 27 je T navigačním stromem pro nějakou pevně danou situaci.*

Důkaz. Na počátku je T navigačním stromem pro situaci s.

Změny T v každém kroku spočívají v mazání části (podstromu bez kořene) T. Z podmínky cyklu plyne, že kořen mazaného "podstromu" (označme jej n) je označen jediným přechodem m.

Pak všechny uzly na cestách z n do všech listů podstromu n jsou také označeny $\{m\}$ nebo prázdnou množinou. Odstranění těchto uzlů neovlivní fakt, že T je navigační strom pro s.

Dále si všimněme, že pokud T_i a T_j jsou dvě různé modifikace T vyskytující se v průběhu alg. 27 a T_i se vyskytlo dříve než T_j, pak T_j je zjemněním navigačního/výhledového stromu T_i.

Lemma 30 *Navigační strom vrácený alg. 27 je prefixově minimální.*

Důkaz. Předpokládejme pro spor, že je možné dále optimalizovat výsledný navigační strom. Pak musí existovat nějaký nelistový uzel označený singletonem. Existuje-li takový uzel, alg. 27 vstoupí do cyklu a situaci napraví [spor].

Algoritmus 31 (Minimalizace výhledů)
Vstup: Libovolný ZAV$\leq k$ A
Výstup: Hloubkově mimimální ZAV$\leq k$ A' takový, že $L(A') = L(A)$
Metoda: Pro každou situaci analýzy automatu A spusť alg. 27.

Důkaz (Věty 18). Alg. 31 vrátí pro každý ZAV$\leq k$ A minimalizovaný ZAV$\leq k$ A' takový, že $L(A') = L(A)$.

Věta 32 *Ke každému ZAV!k existuje DZA P takový, že $L(A) = L(P)$.*

Důkaz je uveden o dvě strany dále – zatím si připravíme, co je třeba.

Následující algoritmus se zdá známým, ale jelikož se nám nepodařilo jej najít v literatuře, rozhodli jsme se jej zde uvést.

Algoritmus 33 (Konstrukce ekvivalentního DZA k ZAV!k)
Idea: Vytvoříme stavy, koncové stavy a přechody DZA, abecedu i zásobníkové symboly ponecháme. Přechodvou funkci rozdělíme na části odpovídající čtení uvnitř textu (δ_{RdChr}) a na jeho konci (δ_{RdEnd}), ε-krokům (δ_{Move}) i samotné inicializaci – a to jak pro dostatečně dlouhý vstup (δ_{IniC}), tak i pro ten krátký (δ_{IniE}).

Vstup: ZAV!k $A = (Q_A, \Sigma, \Gamma, \delta_A, q_0, Z_0, F_A)$.
Výstup: DZA $P = (Q_P, \Sigma, \Gamma, \delta_P, (q_0, \circ^k), Z_0, F_P)$.
Postup:
$Q_P = Q_A \times (\Sigma \cup \{\circ, \$\})^k$ $\quad$ {$\circ$ *reprezentuje neinicializovaný symbol*}
$F_P = \{(q, \$^k) | q \in F_A\}$
$\delta_{RdChr} = \{((q, a_1...a_k), a_{k+1}, Z) \to ((q', a_2...a_{k+1}), \alpha) | (q, a_1..a_k, Z) \to (q', \alpha) \in \delta_A\}$
$\delta_{RdEnd} = \{((q, a_1...a_k), \varepsilon, Z) \to ((q', a_2...a_k\$), \alpha) | (q, a_1..a_k, Z) \to (q', \alpha) \in \delta_A\}$
$\delta_{Move} = \{((q, a_1...a_k), \varepsilon, Z) \to ((q', a_1...a_k), \alpha) | (q, a_1..a_k, Z) \to (q', \alpha) \in \delta_A\}$

$\delta_{IniC} = \{((q_0, a_1 ... a_j \circ^{k-j}), a_{j+1}, Z_0) \to ((q_0, a_1 ... a_{j+1} \circ^{k-j-1}), Z_0)\}$
$\delta_{IniE} = \{((q_0, \circ^{k-j} a_1 ... a_j), \varepsilon, Z_0) \to ((q_0, \circ^{k-j-1} a_1 ... a_j \$), Z_0)\}$
$\delta_P = \delta_{IniC} \cup \delta_{IniE} \cup \delta_{Move} \cup \delta_{RdChr} \cup \delta_{RdEnd}$

Definice 34 *Pro Q_P a Q_A z alg. 33 definujme $State : Q_P \to Q_A$ jako*

$$State((q, a_1 \dots a_k)) = q.$$

Dále definujme Input mapující konfigurace P do Σ^* tak, že

$$Input((q, a_1 \dots a_k), w, \alpha) = Clean(a_1 \dots a_k) \cdot w.$$

Lemma 35 *Mějme automaty A a P z algoritmu 33. Nechť A při výpočtu nad w prochází konfiguracemi $C_0, \dots, C_m$. Pak P při výpočtu nad w prochází konfiguracemi $K_{-k}, \dots, K_m$. Platí, že pokud $K_i = ((q, a_1 \dots a_k), w, \alpha)$, pak $C_i = (State((q, a_1 \dots a_k)), Input(K_i), \alpha)$ pro $0 \le i \le m$.*

Navíc platí, že $K_i = ((q_0, \circ^{-i} a_1 \dots a_{k+i}), u_{k+i}, Z_0)$ pro $-k \le i \le 0$ a kde $a_1 \dots a_{k+i} u_{k+i} = w$.

Důkaz. Výpočet můžeme rozdělit na dvě části: v první shromažďuje DZA informaci o výhledu, ve druhé provádí samotný výpočet.

V inicializační fázi výpočtu zásobníkového automatu P prochází tento automat pouze konfiguracemi odpovídajícími počáteční konfiguraci zásobníkového automatu s orákulem A.

Na začátku je A v konfiguraci $C_0 = (q_0, w, Z_0)$. P začíná v konfiguraci $K'_{-k} = ((q_0, \circ^k), w, Z_0)$. Platí, že $(State((q_0, \circ^k)), Input(K_{-k}), \alpha) = (q_0, w, Z_0) = C_0$.

V jednotlivých krocích (pohybech) inicializace automatu P vykonává tento přechody podle $\delta_{IniC} \cup \delta_{IniE}$, přičemž nejdříve jsou vykonávány přechody z δ_{IniC} (je-li na vstupu alespoň jeden symbol), a pak případně z δ_{IniE} (je-li vstup kratší, než k symbolů).

V každé z inicializačních konfigurací $K_{-k}..K_{-1}$ zásobníkového automatu P je možné provést právě jeden přechod. $K_i = ((q_0, \circ^{-i} a_1 \dots a_{k+i}, u_{k+i}, Z_0)$ pro nějaké i, $-k \le i < 0$. Dále platí, že $u_i = a_{i+1} u_{i+1}$. Pak existuje jediný přechod $((q_0, \circ^{-i} a_1 \dots a_{k+i}), a_{k+i+1}, Z_0) \to ((q_0, \circ^{-i-1} a_1 \dots a_{k+i+1}), Z_0)$. Tento přechod končí v konfiguraci $K_{i+1} = ((q_0, \circ^{-i+1} a_1 \dots a_{k+i+1}), u_{k+i+1}, Z_0)$. Pro všechny tyto konfigurace tedy platí, že

$$\forall i : \ -k \le i \le 0 : \ (State((q, \circ^{-i} a_1 \dots a_{k+i})), Input(K_i), \alpha) = (q_0, w, Z_0) = C_0.$$

Samotný běh automatu P probíhá v krocích odpovídajících krokům automatu A. Nechť A je v konfiguraci $C_i = (q, a_1 \dots a_{k+1} v, Z\alpha)$, $i \ge 0$ a P je v konfiguraci $K_i = ((q, a_1 \dots a_k), a_{k+1} v, Z\alpha)$. Pak musí nastat jedna ze situací níže (popis dle A):

1. čtení symbolu, výhled obsahuje pouze symboly z Σ;
2. čtení symbolu, výhled obsahuje i konec vstupu;
3. ε-přechod;
4. žádný přechod nejde provést.

Diskutujme jednotlivé případy:

1. Nechť je pro A použitelný přechod $(q, a_1[a_2 \dots a_k], Z) \rightarrow (q', \beta)$ vedoucí do konfigurace $C_{i+1} = (q', a_2 ... a_{k+1} v, \beta\alpha)$. Pro P je pak použitelný jediný přechod $((q, a_1 ... a_k), a_{k+1}, Z) \rightarrow ((q', a_2 ... a_{k+1}), \beta)$ vedoucí do konfigurace $C'_{i+1} = ((q', a_2 ... a_{k+1}), v, \beta\alpha)$.
   Platí, že $C_{i+1} = (State((q', a_2 \dots a_{k+1})), Input(C'_{i+1}), \beta\alpha)$.
2. Nechť je pro A použitelný přechod $(q, a_1[a_2 \dots a_j \$^{k-j}], Z) \rightarrow (q', \beta)$ vedoucí do konfigurace $C_{i+1} = (q', a_2 ... a_j, \beta\alpha)$. Pro P je pak použitelný jediný přechod $((q, a_1 ... a_j \$^{k-j}), \varepsilon, Z) \rightarrow ((q', a_2 ... a_j \$^{k-j+1}), \beta)$ vedoucí do konfigurace $C'_{i+1} = ((q', a_2 ... a_j \$^{k-j+1}), \varepsilon, \beta\alpha)$.
   Platí, že $C_{i+1} = (State((q', a_2 \dots a_j \$^{k-j+1})), Input(C'_{i+1}), \beta\alpha)$.
3. Nechť je pro A použitelný přechod $(q, [a_1 \dots a_k], Z) \rightarrow (q', \beta)$ vedoucí do konfigurace $K_{i+1} = (q', a_1 ... a_{k+1} v, \beta\alpha)$. Pro P je pak použitelný jediný přechod $((q, a_1 ... a_k), \varepsilon, Z) \rightarrow ((q', a_1 ... a_k), \beta)$ vedoucí do konfigurace $C'_{i+1} = ((q', a_1 ... a_k), a_{k+1} v, \beta\alpha)$.
   Platí, že $C_{i+1} = (State((q', a_1 \dots a_k)), Input(C'_{i+1}), \beta\alpha)$.
4. Není-li pro A možný žádný přechod, z definice P plyne, že není k dispozici ani žádný přechod pro P. Jedinou výjimkou je počáteční konfigurace A (a tedy i inicializační konfigurace P), kdy P může projít všemi konfiguracemi odpovídajícími počáteční konfiguraci A.

Důkaz (Věty 32). Vhodným automatem je např. ten, který je vytvořen alg. 33. Z lemmatu 35 plyne, že ZAV a DZA procházejí právě odpovídajícími konfiguracemi, a tak rozpoznávají týž jazyk.

Věta 36 *Ke každému DZA P existuje ZAV A takový, že $L(A) = L(P)$.*

Důkaz. Stačí vybrat ZAV s výhledem 1, který pro ε-kroky provádí daný přechod pro libovolný výhled.

5 Závěr

Představili jsme dva modely odpovídající reálným analyzátorům rekurzivním sestupem přesněji než zásobníkové automaty. První, obecnější, model – zásobníkový automat s orákulem – přidává ke klasickým zásobníkovým automatům možnost omezit v průběhu výpočtu množinu aplikovatelných přechodů, případně využít zásahů zvenčí (uživatele).

Rozdělení problému na analýzu a omezování použitelných přechodů umožňuje udržovat analyzátor přehledný. To zjednodušuje vývoj a údržbu takto navržených analyzátorů. Možnost ovlivnit výpočet zvenčí je výhodná pro situace, kdy rozpoznávaný jazyk není zcela specifikován (např. pro přirozené jazyky je nemožné vytvořit přesnou gramatiku, a tedy ani vytvořit odpovídající analyzátor, neboť se průběžně mění).

Výpočetní síla zásobníkových automatů s orákulem závisí na síle orákula. Použití jednoduchého orákula (včetně výhledu k symbolů) nemění rozpoznávanou

třídu bezkontextových jazyků. Přináší pouze větší přiblížení intuici než existující model či zefektivnění výpočtu. Použití silného orákula (např. majícího neomezený výhled nebo vlastní zásobník) umožňuje rozpoznávat jazyky, které nejsou bezkontextové.

Druhý model (automaty s výhledem) jsou zvláštním případem zásobníkových automatů s orákulem, kdy jako orákulum je použit konečný automat, který využívá výhledu nejvýše k symbolů, určuje, které přechody jsou použitelné. Očekávaným typickým použitím tohoto modelu je analýza deterministických bezkontextových jazyků, kdy orákulum vždy vybírá jediný přechod nebo ukončuje výpočet z důvodu neodpovídajícího vstupu.

Reference

1. Aho A.V., Ullman J.D., The Theory of Parsing, Translation and Compiling. Volume I.: Parsing. Prentice-Hall, Englewood Cliffs, N.J., 1972
2. Chytil M., Automaty a gramatiky. SNTL, Praha, 1984
3. Sippu S., Soisalon-Soininen E., Parsing Theory. Volume II: LR(k) and LL(k) Parsing. Number 20 in EATCS. Springer Verlag, Berlin, 1990
4. Král, J., Žemlička, M., Architecture and Modeling of Service-Oriented Systems. In Vojtáš, P., Bieliková, M., Charon-Bost, B., Sýkora, O., eds.: SOFSEM 2005 Communications, Bratislava, Slovakia, Slovak Society for Computer Science, 2004, 71–80
5. Moore E.F., Gedanken Experiments on Sequential Machines. In Shannon, C.E., McCarthy, J., eds.: Automata Studies, Princeton, Princeton University Press, 1956

PRAKTICKÉ NASADENÉ APLIKÁCIE

Frekvenční slovník češtiny

Jaroslava Hlaváčová

Ústav formální a aplikované lingvistiky
Matematicko–fyzikální fakulta Karlovy univerzity v Praze
Malostranské nám. 25, Praha 1
hlava@ufal.mff.cuni.cz

Abstrakt V loňském roce vyšel v nakladatelství Lidové noviny nový Frekvenční slovník češtiny obsahující 50 tisíc nejběžnějších českých slov v základním tvaru. Byl vytvořen na základě vyváženého korpusu současné češtiny SYN2000, který obsahuje 100 miliónů slovních tvarů. Pro výběr slov z korpusu a jejich uspořádání ve slovníku nebyla použita prostá frekvence slov v korpusu, ale tzv. průměrná redukovaná frekvence, která bere v úvahu nejen počet výskytů daného slova, ale i jeho rozložení v rámci korpusu. Tím se zabrání upřednostnění těch slov, které se vyskytují jen v jednom omezeném úseku korpusu, na úkor slov s nižší frekvencí, ale rovnoměrným zastoupením napříč celým korpusem, která jsou obecně považována za běžnější. Ve svém příspěvku vysvětlíme principy použité při vytváření nového frekvenčního slovníku, zejména zmíněnou průměrnou redukovanou frekvenci.

1 Úvod

Frekvenční slovníky jsou velmi populární, a to především ze dvou důvodů:

- teoretického : přinášejí zajímavý vhled do slovní zásoby a ukazují, která slova se nacházejí v jádru a která na periferii jazyka;
- praktického : mohou být přímo použity v praxi, především při výběru hesel pro nové překladové nebo výkladové slovníky nejrůznějších žánrů.

V celém příspěvku budeme používat výraz slovo pro slovo v základním tvaru, jinak také lemma.

Frekvence je jednou z nejpoužívanějších slovních charakteristik. Je často hlavním kritériem pro lexikografy, kteří se rozhodují, která slova zařadit do slovníku a která ne. Frekvence nějakého slova se spočítá snadno — je to počet jeho výskytů v nějakém textu. Když porovnáme frekvenci všech slov z daného textu, okamžitě vidíme, která slova jsou běžná. Tento výsledek se však vztahuje pouze k textu, který jsme použili pro výpočet, nikoliv k jazyku jako celku. Frekvence, tedy počet výskytů slova, závisí samozřejmě na textu. Nejen na jeho délce, ale i na jeho předmětu, autorovi, stylu a dalších vlastnostech. Můžeme shromáždit sbírku textů obsahující texty nejrůznějších žánrů, od nejrůznějších autorů, napsaných nejrůznějšími styly. Kdybychom byli schopni shromáždit veškeré existující texty, napsané i vyslovené, mohli bychom spočítat pravou frekvenci všech slov v jazyce. Takový úkol je ale pochopitelně nemožné splnit, musíme si vystačit s určitým vzorkem textů — s jazykovým korpusem.

2 Jazykový korpus

Čím větší korpus, tím spolehlivější fakta o jazyce z něj můžeme odvodit. Velikost ovšem není jedinou vlastností ovlivňující výsledky. Záleží také na jeho složení, na poměru typů textů, které ho tvoří. Potřebujeme reprezentativní korpus vybudovaný z velkého množství textů nejrůznějších druhů. Tím zvýšíme pravděpodobnost, že ve výsledném korpusu nalezneme všechny jazykové jevy, které chceme studovat. Základem pro vytvoření Frekvenčního slovníku češtiny byl Český národní korpus, jeho verze SYN2000 ([4]). Obsahuje 100 milionů slovních tvarů a byl vytvořen s ohledem na reprezentativnost. Jeho hrubé složení ukazuje následující tabulka [1]:

1. beletrie – 15%;
2. odborné texty – 25%;
3. publicistika – 60%

Korpus SYN2000 byl automaticky lemmatizován a morfologicky označkován (viz [5]). Každé slovní formě byla jednoznačně přiřazena jeho základní forma, tzv. lemma, a morfologická značka. Pro výpočet frekvencí jsme použili lemmata, ne slovní formy. To by bylo také zajímavé, ale frekvenční slovník lemmat bude mít nepochybně větší uplatnění v praxi.

Původní verze korpusu SYN2000 obsahovala určité množství chyb nejrůznějšího druhu — překlepy pocházející z originálních textů, chyby způsobené převodem do jednotného formátu, chyby vzniklé při automatické lemmatizaci a morfologické anotaci, a především disambiguaci homonymních tvarů. Jednou z možností bylo chyby ignorovat a sestavit frekvenční slovník plně automaticky. Při tomto přístupu by se frekvenční charakteristiky některých slov zkreslily, proto bylo rozhodnuto přistoupit k opravám. Při stomilionovém objemu bylo ovšem samozřejmě vyloučeno provádět opravy ručně. Nebudeme se zde zabývat konkrétními přístupy k jednotlivým typům chyb, ty jsou podrobně popsány v [7], jenom se zmíníme o poloautomatických opravách chyb disambiguačních.

Disambiguační chyby byly asi nejvážnějším problémem, se kterým se tým autorů potýkal. Disambiguace homonymních tvarů byla, jak už bylo řečeno, prováděna automaticky pomocí statistických metod ([3]), které pochopitelně nezaručují naprosto správný výsledek. Některé homonymní tvary jsou velmi časté, nebylo tedy možné je zkontrolovat všechny ručně (např. slovní tvar *bez*, který může být nominativ nebo akuzativ podstatného jména, nebo velmi běžná předložka, má v korpusu SYN2000 frekvenci 85 541.) Od každého homonymního tvaru jsme tedy vybrali náhodný vzorek 200 výskytů a na něm spočítali poměr frekvencí lemmat připadajících v úvahu. Pokud byl tento poměr menší než 5%, méně četná alternativa byla vyloučena a celková frekvence slovního tvaru byla připsána četnějšímu lemmatu. Tím se vyloučily málo užívané tvary, které byly díky statistické disambiguaci přiřazeny chybně, zejména šlo o přechodníky (např. *červeně* nebývá přechodník od *červenat*, i když je to správný tvar) a vokativy

[1] Všechny kategorie jsou v originálním korpusu ještě dále rozčleněny na jemnější podkategorie. Těch je ale několik desítek, použili jsme proto jen hrubší dělení.

(např. tvar *plocho* je častěji příslovce z výrazu *na plocho* než 5. pád lemmatu *plocha*). Tímto způsobem se ale vloudily do upraveného korpusu chyby, které v něm původně nebyly. Např. homonymnímu slovnímu tvaru *let*, který může být buď od lemmatu *let*, nebo od lemmatu *rok*, bylo na základě právě popsaného postupu přiřazeno "natvrdo" lemma *rok*, které je tak časté, že i nezanedbatelné absolutní množství tohoto tvaru od lemmatu *let* tvořilo méně než 5% celkové frekvence. Tím se samozřejmě posunula frekvence obou lemmat. Celkově však se výsledný korpus spíše vylepšil.[2]

Výsledný, poloautomaticky opravený korpus dostal název fsc2000. Výsledky dotazů na korpus fsc2000 tak plně odpovídají číselným charakteristikám Frekvenčního slovníku češtiny, což nelze říci o původním zdroji dat SYN2000.

3 Absolutní frekvence a průměrná redukovaná frekvence

Máme-li k dispozici jazykový korpus, není nic jednoduššího než spočítat výskyty všech jeho slov. Je-li navíc reprezentativní a dostatečně velký, jsou výsledky spolehlivější. Nikdy však nebude možno překonat nerovnoměrné rozložení slov, které výslednou frekvenci ovlivní. Vždy se objeví texty s neobvyklou koncentrací určitého slova (hrdina románu, název nově objevené chemické sloučeniny v nedělní příloze novin, jméno vesnice, kde došlo k důležité mezinárodní schůzce, ...). Toto slovo má v korpusu vyšší relativní frekvenci vůči ostatním slovům, než by byla jeho relativní frekvence v jazyce jako celku. Lexikografové, kteří rozhodují, která slova zahrnout do svého (nikdy ne neomezeného) slovníku, znají tento problém velmi dobře. Především u slov s nižší frekvencí je třeba postupovat často podle intuice spíše než podle korpusové frekvence. Vlastně by potřebovali vyjádřit běžnost slov, ne jejich frekvenci.

Právě z tohoto důvodu jsme se rozhodli použít pro náš frekvenční slovník ne absolutní frekvenci, ale tzv. průměrnou redukovanou frekvenci, která lépe odpovídá naznačenému termínu běžnost než frekvence absolutní. Její hodnoty jednoznačně odhalí slova, která se vyskytují v korpusu jen v jednom nebo několika málo shlucích, i když jejich absolutní frekvence je třeba vysoká. Průměrná redukovaná frekvence (ARF — average reduced frequency) takových slov je mnohem menší než ARF slov se stejnou absolutní frekvencí, ale s rovnoměrnějším rozložením v korpusu.

Naznačíme zde postup odvození ARF, přesný popis najde laskavý čtenář v [2].

Korpus se skládá z pozic, každá pozice je obsazena právě jedním slovem. Očíslujme si pozice korpusu čísly od 1 do N. N je tedy délka celého korpusu. Mějme slovo mající v korpusu absolutní frekvenci f. Rozdělme si celý korpus na f segmentů stejné délky, tedy N/f (pro jednoduchost a názornost zde předpokládejme, že N je dělitelné f). Kdyby bylo dané slovo rozloženo v korpusu rovnoměrně, každý segment by obsahoval právě jeden jeho výskyt. Jelikož tomu tak většinou není, obsahují některé segmenty více výskytů, jiné žádný. Počet segmentů obsazených alespoň jedním výskytem nazýváme redukovanou

[2] Zajímavé by bylo spočítat frekvenční slovník na neupravené verzi SYN2000 a výsledky porovnat, ale pokud vím, nikdo se tím nezabýval.

frekvencí. Redukovaná frekvence má jednu nepříjemnou vlastnost. Slovo vyskytující se v malém shluku má redukovanou frekvenci buď 1 nebo 2 podle toho, zda hranice segmentů padne náhodou doprostřed shluku výskytů nebo vně. Abychom tento nedostatek odstranili a učinili tak míru objektivnější, počítáme průměrnou redukovanou frekvenci, tedy aritmetický průměr redukovaných frekvencí přes všecky možné začátky prvního segmentu. Za tím účelem si představujeme korpus ne jako úsečku, ale jako kružnici; za poslední pozicí hned následuje první pozice korpusu. Při dělení korpusu potom máme stále stejný počet segmentů o stejné délce.

Průměrná redukovaná frekvence daného slova se spočítá podle následujícího vzorce:

$$ARF = \frac{1}{v} \sum_{i=1}^{f} \min\{d_i, v\}.$$

kde $v = N/f$ a d_i označují vzdálenosti mezi jednotlivými výskyty slova v korpusu. Konkrétně označíme-li si n_1,n_2,...,n_f čísla pozic, na kterých se vyskytuje dané slovo, potom $d_i = n_i - n_{i-1}$ pro každé $i = 2, \ldots, f$ a $d_1 = n_1 + (N - n_f)$, což je vzdálenost mezi posledním a prvním výskytem slova v cyklickém uspořádání korpusu.

3.1 Vlastnosti ARF

Přestože je v názvu použito slovo frekvence, může ARF nabývat (a většinou nabývá) neceločíselných hodnot, jak je vidět ze vzorečku.

ARF může nabývat libovolné hodnoty z intervalu $< 1, f >$. Nejnižší hodnoty nabývá pouze pro slova s absolutní frekvencí 1, nejvyšší potom pouze pro slova rozložená v korpusu naprosto rovnoměrně (což jsou většinou opět jen slova s absolutní frekvencí 1).

Slovo nacházející se v jednom malém úseku korpusu má ARF o málo větší než 1. Závisí více na délce úseku, ve kterém se slovo vyskytuje, než na jeho absolutní frekvenci, jak moc se bude ARF lišit od jedničky. Slovo vyskytující se ve dvou shlucích, má ARF o málo větší než 2, podobně pro další malá přirozená čísla. ARF slov vyskytujících se v korpusu víceméně rovnoměrně se blíží svojí hodnotou hodnotě absolutní frekvence.

Kromě ARF jsou v článku [2] uvedeny ještě další dvě míry, které by bylo také možno použít ve smyslu běžnosti. Je to průměrná čekací doba (AWT — average waiting time)

$$AWT = \frac{1}{2}\left(1 + \frac{1}{N}\sum_{i=1}^{f} d_i^2\right)$$

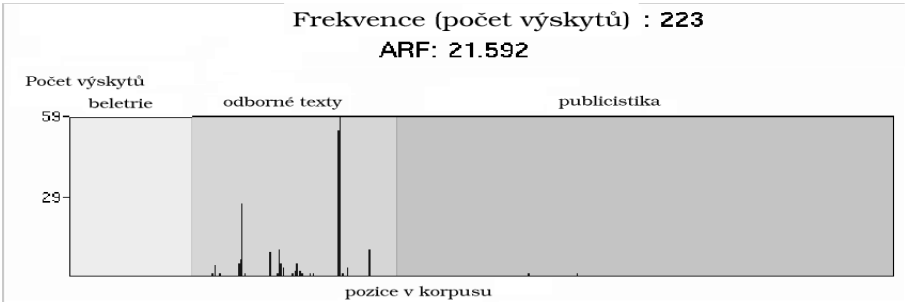
a průměrná logaritmická vzdálenost (ALD — average logarithmic distance)

$$ALD = \frac{1}{N}\sum_{i=1}^{f} d_i \log_{10} d_i.$$

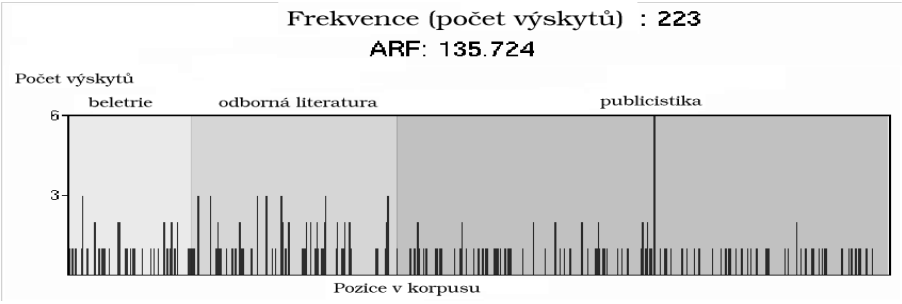
Hlavní důvody, proč byla vybrána právě ARF, jsou tyto:

1. ARF je implementována v korpusovém vyhledavači Bonito ([6]), který používá většina uživatelů Českého národního korpusu; výsledky vyhledávek nad korpusem fsc2000 tedy přesně odpovídají hodnotám ve Frekvenčním slovníku češtiny;
2. ARF je ze všech tří měr nejpřímočařejší, mimo jiné i nejsnáze vysvětlitelná.

Podívejme se nyní na rozdíl mezi absolutní a průměrnou redukovanou frekvencí na příkladech z frekvenčního slovníku. Slova *molekulový* a *nahromadit* mají v korpusu fsc2000 stejnou frekvenci 223.



Obr. 1. Rozložení výskytů slova *molekulový* v korpusu fsc2000.



Obr. 2. Rozložení výskytů slova *nahromadit* v korpusu fsc2000.

Obrázky 1 a 2 ukazují jejich rozložení v korpusu. Vodorovná osa představuje celý korpus — jeho začátek, tedy první slovo, leží vlevo. Odstíny šedi rozlišují tři základní žánry popsané výše. z obrázků je vidět, že korpus byl sestaven tak, aby dokumenty jednotlivých žánrů nebyly v korpusu promíchány.

Frekvence slova (počet jeho výskytů v korpusu) je vyznačena na svislé ose, která má na obrázcích rozdílné měřítko — vždy se počítá podle konkrétních dat vyčtených z korpusu, aby výsledný obrázek měl stejnou velikost. Obrázky byly upraveny na základě výstupu z korpusového manažeru Bonito ([6]), jehož autorem je Pavel Rychlý z Masarykovy univerzity v Brně.

Vidíme, že první slovo (*molekulový*) se vyskytuje převážně v odborné části korpusu, zatímco druhé slovo (*nahromadit*) je rozloženo mnohem rovnoměrněji po celé délce korpusu. Průměrná redukovaná frekvence rozdíl mezi mezi rozložením, a tím i mezi běžností obou slov velmi názorně odráží.

4 Frekvenční slovník češtiny

Slovník ([1]) má dvě verze — elektronickou na CD a klasickou knižní papírovou. Obě verze obsahují 50 000 nejběžnějších českých obecných slov — apelativ (tedy ne vlastních jmen — těm je věnována zvláštní tabulka) spočítaných z korpusu fsc2000. Nejde tedy o slova nejfrekventovanější, k výběru slov do slovníku byla použita výše popsaná průměrná redukovaná frekvence ARF.

Hlavní část slovníku je v knižní verzi řazena abecedně a pro každé slovo uvádí kromě ARF i frekvenci absolutní. Ve skutečnosti ani ARF, ani absolutní frekvence není důležitá. Důležitá jsou pořadí slov podle těchto charakteristik. Ta jsou samozřejmě ve slovníku uvedena také. Poslední číselné charakteristiky uváděné u každého slova jsou frekvence ve třech hlavních žánrech. Vzhledem k tomu, že žánry nejsou v korpusu zastoupeny rovnoměrně, jsou tyto frekvence normalizovány. Normalizovaná frekvence v žánru vyjadřuje frekvenci, jakou by dané slovo v daném žánru mělo, kdyby každý ze tří žánrů tvořil právě jednu třetinu korpusu. Pro snazší porovnávání jsou normalizované frekvence v žánrech uvedeny v procentech.

Slovník tedy obsahuje pro každé slovo následující číselné charakteristiky:

- ARF : průměrná redukovaná frekvence
- rank ARF : pořadí podle ARF
- FRQ : absolutní frekvence
- rank FRQ : pořadí podle FRQ
- bel. : frekvence v žánru beletrie
- odb. : frekvence v žánru odborné literatury
- pub. : frekvence v žánru publicistiky

Podívejme se nyní na naše příklady z předchozí části. Přestože jejich absolutní frekvence je stejná, jejich průměrné redukované frekvence se liší, a podle toho se liší i odpovídající pořadí. Poslední tři sloupce říkají, že 99% všech výskytů slova *molekulový* se nacházejí v odborných textech, 1% v publicistice a 0% v beletrii. Druhý příklad – *nahromadit* – vykazuje mnohem rovnoměrnější zastoupení v rámci celého korpusu i ve všech třech žánrech. Odpovídá to obrázkům uvedeným výše.

slovo	rank ARF	ARF	rank FRQ	FRQ	bel. (%)	odb. (%)	pub. (%)
molekulový	37 502	22	18 959	223	0	99	1
nahromadit	14 970	136	18 915	223	34	43	23

Kromě hlavního abecedně řazeného seznamu 50 000 nejběžnějších českých slov obsahuje papírová verze Frekvenčního slovníku češtiny ještě:

- frekvenční slovník 20 000 apelativ řazených podle FRQ
- frekvenční slovník 20 000 apelativ řazených podle ARF
- frekvenční slovník 2 000 vlastních jmen řazených podle ARF
- frekvenční slovník 1 000 zkratek řazených podle ARF

Frekvenční slovník řazený podle ARF je podmnožinou hlavní části, což ovšem neplatí o frekvenčním slovníku řazeném podle absolutní frekvence. V korpusu fsc2000 se totiž nachází 15 slov, jejichž pořadí podle absolutní frekvence je menší než 20 000, ale pořadí podle průměrné redukované frekvence je vyšší než 50 000. z tohoto důvodu se těchto 15 slov ocitlo v menším seznamu, ale nedostalo se do hlavní části. Všechna tato slova jsou odborné termíny z různých oblastí vědy (fyzika, biologie, informatika, ...), např. *suprematismus*, *repertorium*, *rezistor*, *heparin*. Tato slova skutečně nepatří mezi 50 000 nejběžnějších českých slov, i když jejich absolutní frekvence je poměrně vysoká.

4.1 CD

CD přiložené ke knižní verzi Frekvenčního slovníku češtiny má téměř totožný obsah. Slůvko téměř jsme použili proto, že neobsahuje oněch 15 slov uvedených na konci předchozího paragrafu.

Najdeme na něm tedy 50 000 nejběžnějších českých slov podle průměrné redukované frekvence. Navíc je CD vybaveno prohlížečem EFES, který uživateli umožňuje uspořádat data podle libovolné z uvedených charakteristik i vyhledávat data podle nejrůznějších podmínek. Tedy nejen podle abecedy či jedné z frekvencí. Je možné vyhledávat podle koncového řetězce nebo i libovolného podřetězce, což umožní vytvářet frekvenční slovníky např. slovních kořenů, předpon či přípon. Vyhledávací podmínky lze kombinovat pomocí boolovských operátorů.

5 Závěr

Na příkladu nového Frekvenčního slovníku češtiny jsme ukázali nevhodnost absolutní korpusové frekvence pro řazení slov podle běžnosti v jazyce. Mnohem užitečnější se jeví průměrná redukovaná frekvence, která zohledňuje nerovnoměrnosti, nacházející se ve větší či menší míře v každém, sebepečlivěji vytvořeném korpusu. Pokud je nám známo, byla tato charakteristika použita pro frekvenční slovník poprvé.

Reference

1. Čermák et al., Frekvenční slovník češtiny. Praha, NLN 2004. ISBN 80-7106-676-1
2. Savický P., Hlaváčová J., Measures of Word Commonness. Journal of Quantitative Linguistics. , Swets & Zeitlinger Vol. 9, No. 3, 2002, 215-231
3. Hajič J., Hladká B., Tagging Inflective Languages. Prediction of Morphological Categories for a Rich, Structured Tagset. ACL-Coling'98. Montreal, Canada, August 1998, 483–490
4. Czech National Corpus: `http://ucnk.ff.cuni.cz/`
5. Hajič J., Disambiguation of Rich Inflection. (Computational Morphology of Czech) Praha, Karolinum 2004
6. `http://nlp.fi.muni.cz/projects/bonito/`
7. Křen M.: Frequency Dictionary of Czech: A Detailed Processing Description. Submitted for print
8. Těšitelová M. et al., Kvantitativní charakteristiky současné češtiny. Praha, Academia 1985

Ph.D. A ŠTUDENTSKÁ SEKCIA

Budování česko-anglického slovníku pro strojový překlad*

Ondřej Bojar

Ústav formální a aplikované lingvistiky MFF UK
Malostranské náměstí 25, Praha 1, 118 00, Česká republika
bojar@ufal.mff.cuni.cz

Abstrakt Příspěvek popisuje probíhající práce na česko-anglickém slovníku pro strojový překlad. Slovník vychází z dostupných strojově čitelných slovníků, kombinací ručních a automatických metod jsou doplňovány informace nutné pro nasazení slovníku v automatickém systému. Jedná se zejména o morfologická omezení nutná pro uplatnění hesel, popis syntaktické struktury hesel a překladové ekvivalenty slovesných rámců.

1 Úvod

Strojový překlad je obtížnou úlohou, kterou nelze uspokojivě řešit bez rozsáhlých a kvalitních slovníků, zejména v případě, že chceme dát přednost strukturálnímu přístupu před přístupem čistě statistickým (pro podrobný přehled celé škály metod strojového překladu viz např. [1]). V případě překladu mezi češtinou a angličtinou se strukturálním překladem experimentují zejména [2], kteří překládají z češtiny texty ekonomické povahy. Naše práce je pak spjata s projektem GAČR 405/03/0914, jehož cílem je přizpůsobit překladový systém Ruslan [3] pro překlad do angličtiny místo původní ruštiny.

V současné době není k dispozici žádný vhodný slovník pro strojový překlad. Z toho důvodu např. i [2] používají zjednodušený slovník s překladovými hesly[1] omezenými na jedno slovo na české straně a nejvýše dvouslovné pevné spojení na straně anglické. Je zřejmé, že při takovém omezení systém nemůže správně překládat např. slovesné vazby ap.

Naším cílem je tedy obohatit strukturu slovníku, tak aby vyhovovala potřebám (zejména strukturálního) strojového překladu, a připravit slovník pro překlad z češtiny do angličtiny.

V sekci 2 popisujeme dostupné zdroje dat pro náš úkol a ilustrujeme některé jejich problémy. Sekce 3 se podrobně věnuje našim krokům při čištění slovníků a doplňování informací, které jsou nezbytné pro nasazení v systému automatického překladu. Samostatnou sekci 4 věnujeme našemu novému algoritmu určenému pro automatický sběr překladů slovesných rámců.

* Práce na tomto projektu je podporována granty MŠMT LC536, GAČR 201/05/H014 a GAUK 351/2005.

[1] V celém textu budeme pojmem HESLO označovat překladový pár jednoho českého a jednoho anglického výrazu.

2 Dostupné strojově čitelné slovníky a jejich problémy

Při budování slovníku pro strojový překlad je jistě vhodné v maximální možné míře využít existující elektronicky dostupné zdroje. V našem případě se jedná zejména o slovníky WinGED[2] a [4] a rovněž nelze opomenout EuroWordNet[3].

Dostupné elektronické slovníky s výjimkou EuroWordNetu jsou budovány zásadně pro lidského uživatele a z tohoto důvodu je nelze přímo použít při automatickém zpracování. Problémy zejména působí nekonzistentní anotace informací o slovníkových heslech a chybějící údaje, které jsou nutné pro jednotlivé kroky strojového překladu.

2.1 Nekonzistentní anotace metainformací

Dostupné slovníky mají tvar dvousloupcového textu, kde na levé straně je výraz český a na pravé straně výraz anglický. Samotný text na jednotlivých řádcích však v řadě případů obsahuje nejen čistý základní tvar překládaného hesla, ale též dodatečné informace *o* výrazu, např.:

- stručnou poznámku o stylu či dialektu pro dané konkrétní heslo: *am.*, *mat.*
- oddělovač vyznačující více variant výrazu, z nichž lze při překladu užít vždy jen jednu: *be liable/subordinate to*
- konstrukce označující nepovinnou část výrazu: *(auto)stop*, *(na)líčení tváře*
- poznámky popisující tvaroslovné vlastnosti výrazu: *průdušky pl.* pro plurál, *pyramida f* pro ženský rod
- poznámky charakterizující syntaktické vlastnosti výrazu, zejména syntaktické rámce sloves ap.: *accession to = vstoupení do* nebo *adjudge sb. to be guilty = uznat vinným koho*

Tyto cenné informace bohužel nejsou v heslech vyznačovány žádnou standardizovanou metodou. Pro lidského uživatele není problém takové informaci v heslu porozumět a zejména ji z hesla vyjmout, automatický systém však nemá žádný podklad pro to, aby např. v překládané větě nehledal text *pl.*, když chce použít heslo *průdušky pl.*, nebo aby při generování naopak nepřidal do textu tuto "netextovou" část hesla.

2.2 Chybějící morfologická informace

Pro úspěšné použití slovníkového hesla v automatickém systému je nutné poskytnout počítači informaci o slovním druhu jednotlivých slov výrazu a případně o dalších povinných morfologických rysech hesel (např. povinný genitiv, povinnou negaci, povinné množné číslo ap.). Až na velmi řídké výjimky však slovníková

[2] `http://www.rewin.cz/`

[3] EuroWordNet ([5]) je možné použít jako překladový slovník, obsahuje však celkem pouze cca 10 000 podstatných jmen a cca 3 000 sloves (bez informace o vazbách, s výjimkou zvratné částice *se,si*), což je o řád méně než ostatní slovníky.

hesla tyto informace neobsahují a s ohledem na vysokou míru tvaroslovné nejednoznačnosti češtiny v řadě případů není morfologická informace pro počítač jasná. Počítač pak nemůže rozhodnout, zda je překladové heslo oprávněn použít, nebo zda musí hledat jinou variantu, protože slovní druhy neodpovídají. Tabulka 1 ilustruje tento problém ukázkou českých výrazů ze slovníkových hesel.

Substantivum a substantivum/adjektivum	**Správná interpretace**
husa divoká	substantivum adjektivum
kniha účetní†	substantivum adjektivum
napětí dovolené†	substantivum adjektivum
chyba měření	substantivum substantivum
plán prací†	substantivum substantivum
rozsah měření	substantivum substantivum
Číslovka/sloveso a substantivum	**Správná interpretace**
tři prdele†	číslovka substantivum
pět švestek	číslovka substantivum
pět chválu	sloveso substantivum

Tab. 1. Příklady nejednoznačnosti slovních druhů v překladových slovnících. Výrazy označené † umožňují i druhou interpretaci, typicky poněkud směšnou.

2.3 Chybějící syntaktická informace

Tvůrci překladových slovníků typicky neuvádějí žádné informace o syntaktické struktuře překladových hesel a ani nedodržují žádná pevná pravidla, která by umožnila tuto informaci automaticky u hesel doplnit. (Např. řídící slovo výrazu by mohlo být uváděno zásadně na prvním místě, ne vždy se tak však děje.)

Syntaktická informace (vnitřní syntaktická struktura hesla) je však pro strukturální překladový systém velmi podstatná. Systém totiž typicky provede větný rozbor dříve, než vyhledává překladové ekvivalenty. Víceslovný výraz je pak typicky reprezentován více uzly ve stromové struktuře věty a pokud slovníková hesla tuto strukturu nepopisují, je obtížné v této fázi identifikovat, zda se heslo ve větě skutečně vyskytlo.

3 Postup čištění a doplňování potřebných informací

V této sekci stručně popíšeme provedený postup čištění a doplnění potřebných informací do dostupných elektronických slovníků WinGED a GNU-FDL.

Ve všech případech ručního čištění jsme zásadně postupovali po blocích, metodou velmi podobnou klasickému "rozděl a panuj", neboť tento postup vede k nejvyšší dosažitelné vnitřní konzistenci a současně je i nejrychlejší. Před samotnou ruční korekcí nejprve shromáždíme k sobě všechna slovníková hesla s daným nejednoznačným nebo nepřesným rysem a pak pracujeme jen na této

podmnožině. Můžeme samozřejmě tuto podmnožinu opět rozdělit podle dodatečných kritérií a v řadě případů je možné dopracovat se k podmnožině dostatečně specifické na to, aby bylo možné korekci provést na všech heslech najednou.

3.1 Identifikace metainformací

V prvním kroku jsme identifikovali všechna častá slova (dle výskytu ve slovníku), která typicky kódují určitou informaci o hesle, např. *pl.*, *am.*, *někoho*, *sb.*, *someone* ap. Rovněž jsme prověřili všechna hesla, která končí potenciální předložkou. Ve všech těchto případech bylo na základě překladového ekvivalentu možné rozhodnout, zda dané slovo vyjadřuje morfologickou informaci, informaci o stylu, "slot" (tj. zástupný symbol pro další rozvití), nebo zda jde o pevnou součást hesla. Např. v hesle *mít o sobě vysoké mínění = think something of oneself* označují slot pouze slova *sobě* a *oneself*, ne však slovo *something*.

V této fázi jsme rovněž roznásobili všechna hesla, která kódovala více překladových variant současně.

3.2 Zjednoznačnění slovních druhů

Ke zjednoznačnění slovních druhů jsme zatím přistoupili pouze na české straně. Slovní druhy na anglické straně bude možné částečně odvodit ze slovního druhu na české straně a ve sporných případech zvolíme analogickou techniku jako pro češtinu.

Slovníková hesla jsme nejprve analyzovali pomocí morfologického analyzátoru [6] a víceznačné případy jsme roztřídili ručně.[4] Postupovali jsme přitom opět v blocích podle konkrétního typu morfologické nejednoznačnosti, jak však ilustruje tabulka 1, není možné úplně se vyhnout čtení slovníkových hesel.

3.3 Doplnění morfologických omezení

Morfologická omezení hesel popisují, které hodnoty morfologických rysů jsou pro která slova v hesle povinná a která volná, případně s dodatečným požadavkem shody daného rysu mezi více slovy hesla. Morfologická omezení pak budou využita při analýze textu ke kontrole, zda výskyt daných slov skutečně vyjadřuje slovníkové heslo, nebo jde o náhodnou kolokaci. S ohledem na náš směr překladu z češtiny do angličtiny jsme se zatím soustředili na identifikaci morfologických omezení pro českou stranu hesel.

Nutná morfologická omezení generujeme automaticky na základě příkladů dostupných v korpusu[5]. Pro každé slovníkové heslo automaticky vyhledáme v korpusu věty, které obsahují všechna lemata (základní tvary slov) daného hesla v blízkém sousedství. Mezi slovy přitom mohou být vložena jiná slova

[4] Je zřejmé, že pro tento účel není vhodné použít automatickou desambiguaci (tagger), neboť slovníkovým heslům chybí větný kontext nezbytný pro správný chod taggeru.

[5] Konkrétně používáme Český národní korpus [7].

a na pořadí slov nezáleží. Nalezeným výskytům přiřazujeme váhu tak, aby vyšší váhu měly výskyty bez vložených slov a případně se souvislým závislostním grafem.[6] Seznam vážených výskytů je prohledán a je ověřeno, zda dané výskyty splňují některé z předem definovaných typů omezení. Dosud jsou v našem repertoáru předem definovaných omezení omezení unární (např. test typu "pád je akuzativ" nebo "číslo je jednotné") a binární ("pád u prvního a druhého slova se shodují" ap.). Pokud alespoň 75 % celkové váhy tvoří výskyty splňující dané omezení, pokládáme omezení za povinně platné pro dané heslo. (Zbylých 25 % přisuzujeme náhodným souvýskytům slov.)

Přestože je popsaný algoritmus velmi jednoduchý a přestože použité korpusy obsahují často chybnou (automaticky generovanou) morfologickou a závislostní informaci, jsme s výsledky automatické extrakce platných omezení spokojeni. Většina z hesel, u nichž se podařilo v korpusu najít alespoň 10 výskytů, získává správnou sadu omezení. Příklad takto získaných omezení je uveden v tabulce 2. Jako problematická se jeví zejména hesla složená z velmi běžných slov. Běžná slova se totiž poměrně často vyskytnou blízko sebe i v případě, že netvoří dohromady hledané heslo. Značná část pozorovaných výskytů pak neodráží omezení daného hesla a náš práh 75 % tato omezení nepřijme. Např. pro spojení *bohatý člověk* nebylo možné automaticky potvrdit požadavek shody čísla a rodu, shodu pádu se potvrdit podařilo.

Český výraz	**Unární omezení**	**Binární omezení**
za nízkou cenu	RR–4 AAF** NNF*4	pčr:2=3
v jediném dnu	RR–6 AA*S* NNIS*	číslo:2=3
v jiném směru	RR–6 AAI** NNIS6	pád:1=3 číslo:2=3 rod:2=3
v jiném stavu	RR–6 AA*S* NNIS*	
v jistém smyslu	RR–6 AAIS6 NNIS6	pád:1=2 pád:1=3 pčr:2=3
bohatý člověk	AA*** NNM**	číslo:1=2
získané informace	AAFP* NNFP*	pčr:1=2
zkušební provoz	AAIS* NNIS*	pčr:1=2
první světová válka	CrFS* AAFS* NNFS*	pčr:1=2 pčr:1=3 pčr:2=3

Unární omezení vyjadřujeme českou poziční morfologickou značkou (pro reference viz PDT, [8]), kde znak "*" označuje pozici, jejíž hodnota závisí na kontextu (tj. pozici, na niž se žádné unární omezení nevztahuje).
Binární omezení určují, které hodnoty musí být mezi jednotlivými slovy výrazu sdíleny. Např. "pčr:2=3" vyjadřuje, že pád, číslo i rod druhého a třetího slova se musí shodovat.

Tab. 2. Příklady automaticky extrahovaných morfologických omezení.

3.4 Doplnění syntaktických omezení

Syntaktické informace (závislostní vztahy mezi slovy výrazu) jsme zatím doplnili pouze do české části hesel. Ve většině případů bylo možné strukturu stano-

[6] V případě, že použitý korpus obsahuje již i větnou strukturu, jako např. PDT [8] nebo PCEDT [9], lze ji přímo použít. V případě ostatních korpusů spoléháme na automatickou syntaktickou analýzu, jak ji generuje parser [10] adaptovaný pro češtinu.

vit jednotně pro drtivou většinu hesel se shodnou posloupností slovních druhů jednotlivých slov (např. u všech hesel obsahujících adjektivum následované podstatným jménem je řídícím slovem substantivum). Pro zbývající hesla s velmi různorodými posloupnostmi slovních druhů je ruční metoda příliš náročná a dáváme přednost hledání typické struktury korpusu, podobně jako popisujeme v sekci 3.3.

4 Automatické získávání překladových slovesných rámců

Termínem PŘEKLADOVÝ SLOVESNÝ RÁMEC budeme označovat specifický typ překladového hesla.[7] Kromě českého slovesa a jeho anglického protějšku překladový rámec obsahuje též seznam dvojic popisující české a odpovídající anglické formy slovesných doplnění. Příkladem může být heslo: *dělit=divide na+akuzativ =into*.

Slovníková hesla po ručním očištění (viz sekce 3.1) sice v řadě případů obsahují u slovesných hesel informaci o vazbách a jejich překladech, specifické zaměření ekonomických textů však způsobuje, že řada užívaných vazeb ve slovníku vůbec není zahrnuta. Z tohoto důvodu se věnujeme i vývoji automatické metody, která z paralelního korpusu překladová slovesná hesla získá.

Extrakcí (povrchových) slovesných rámců z různých typů korpusů se zabývá též [12] a další práce, na něž odkazuje. Náš cíl je však odlišný. Zatímco citovaní autoři usilují o zodpovězení otázky, která z pozorovaných slovesných doplnění jsou nedílnou součástí slovesného rámce (complements), a která jsou na slovese nezávislá (adjuncts), my potřebujeme vědět: "Jakou vazbu je třeba užít na anglické straně, jestliže na české straně bylo doplnění daného druhu.

4.1 Metoda získávání překladových rámců

Pro získávání překladových rámců užíváme paralelní závislostní korpus [9], který jsme navíc obohatili automatickou metodou (GIZA++, [13]) o párování jednotlivých slov k sobě. Obecně je možné naši metodu nasadit na jakýkoli paralelní text po doplnění (automatické) závislostní analýzy v obou jazycích a spárování textů po slovech.

V prvním kroku získáváme POZOROVANÉ PŘEKLADOVÉ RÁMCE takto: zaměříme se na všechny výskyty českých sloves, k nimž je párováním jednoznačně připojeno sloveso anglické. Pro všechna doplnění českého slovesa pak zjistíme odpovídající doplnění anglického slovesa: sledujeme, ke kterým uzlům v anglické větě vedou párovací hrany z celého podstromu studovaného českého doplnění. Jako odpovídající anglické doplnění zvolíme to doplnění, do jehož podstromu vede nejvíce sledovaných párovacích hran. V sekci 4.3 popisujeme problémy, na něž tento jednoduchý přístup naráží.

[7] Je třeba upozornit na odlišnost naší definice překladového rámce od valenčních rámců zachycovaných ve valenčním slovníku češtiny VALLEX (nejnověji viz. [11]) nebo PDT-VALLEX (odkaz viz VALLEX), které se soustředí na zachycení hloubkových syntaktických rysů.

Ve druhém kroku získané překladové rámce automaticky očišťujeme, a to několika různými způsoby:

- Bez očištění (označení `raw`): použijeme přímo pozorované překladové rámce.
- Odstranění málo četných typů doplnění (`freq`): ze všech rámců odstraníme všechny překladové páry forem doplnění, které nebyly pozorovány (napříč všemi slovesy) dostatečně často.
- Odstranění nespolehlivých párování (`giza`): na základě dodatečné informace ze systému GIZA++ odstraníme všechny rámce pozorované ve větách, kde si GIZA++ nebyla výsledným párováním slov dostatečně jista.
- Pouze "velmi jednoduché české věty (`vjv`): Použitím algoritmu popsaném v práci [14] odstraníme všechny rámce pozorované ve složitých českých větách. Složité české věty představují pro automatickou syntaktickou analýzu větší problém a [14] ukazuje, jak je možné kvalitu pozorovaných českých rámců zlepšit, pokud se omezíme na jednodušší větné konstrukce.

Třetí volitelný krok navazuje na očištěné rámce a používá statistické metody nebo metody strojového učení k tomu, aby dále zjednodušil množinu očištěných rámců a dospěl k finální verzi překladových rámců určené pro nasazení v překladovém systému. Dosud jsme vyzkoušeli jednu z možných metod, algoritmus Apriori ([15])[8], jinou možností je využít některou z metod navrhovaných v práci [12] pro identifikaci těch doplnění, která jsou pro dané sloveso typická. V překladových rámcích pro dané sloveso bychom pak uváděli jen překlady typických doplnění a překlady ostatních doplnění bychom překládali pro všechna slovesa společně.

4.2 Vyhodnocení

Abychom vyhodnotili, který z navržených postupů získávání a čištění překladových rámců poslouží v překladovém systému pravděpodobně nejlépe, připravili jsme ruční korekcí sadu 140 testovacích vět. Testovací věty obsahují celkem 400 výskytů 200 různých sloves a tato slovesa mají dohromady celkem 1005 doplnění.

Vzhledem k tomu, že celý systém překladu ještě není zprovozněn, implementovali jsme tři jednoduché algoritmy, které mají za úkol pro dané sloveso a jeho

[8] Algoritmus Apriori a další podobné algoritmy byly navrženy pro podporu prodeje: rozborem množiny uskutečněných obchodních transakcí, kde každá transakce představuje množinu zakoupených artiklů, algoritmus identifikuje časté vztahy mezi vybranými artikly. (Kdo si koupil chléb, koupil si často i máslo.) Alternativně může být výstup tohoto algoritmu vyjádřen jako typické podmnožiny uskutečněných transakcí. Analogie s naší situací je zřejmá: jedné transakci odpovídá jeden výskyt slovesa v paralelním korpusu a každá překladová dvojice český slot–anglický slot pak představuje jeden artikl. Algoritmus identifikuje, které podmnožiny překladových dvojic jsou pro dané sloveso typické.

pozorovaná česká doplnění odpovědět, jaké formy budou mít odpovídající doplnění anglická.[9] Všechny tři se opírají o očištěný seznam rámců (pro stručnost říkejme prostě "slovník"), liší se však v tom, jakým způsobem konstruují pro pozorovaný český rámec anglickou odpověď:

- Algoritmus A – překlad slot po slotu bez ohledu na sloveso. Tento hladový algoritmus napřed předzpracuje slovník a pro každý český slot si poznamená, jakou formu má nejčastěji příslušný anglický slot. Při samotném běhu pracuje slot po slotu a zásadně volí tento vítězný ekvivalent.
- Algoritmus B – překlad slot po slotu s ohledem na sloveso. Podobně jako A předzpracuje slovník a překlad volí slot po slotu, vítězný překlad však volí jen ze slotů pozorovaných u daného českého slovesa. V případě, že ve slovníku u daného slovesa nějaká česká forma zcela chybí, můžeme algoritmus B kombinovat s A a zkusit navrhnout překlad slotu bez ohledu na slovesa. U jiného slovesa totiž daná česká forma viděna být mohla a možná i její překlad dobře poslouží zde. Tento kombinovaný algoritmus označme "BA".
- Algoritmus C – překlad podle vítězného rámce. Pro vstupní pozorovaný rámec algoritmus nejprve najde mezi slovníkovými rámci daného slovesa takový rámec, který je pozorovanému rámci nejpodobnější (ev. více rámců se stejnou mírou podobnosti). Pak překládá slot po slotu, ale překladové ekvivalenty volí jen na základě vybraných rámců. V případě, kdy se nepodaří nějaký slot přeložit, protože vítězný rámec překlad daného slotu vůbec neobsahoval, je opět možné překlad navrhnout některou z jednodušších metod. Konkrétně používáme kombinace CBA a CB.

Pročištění	**Apriori**	**Algoritmus**	**F-score**	**Přesnost**	**Pokrytí**
`giza`	Apriori	A	68.4	52.9	96.7 ← nejlepší F-score
`giza`	Apriori	CBA	66.4	50.5	96.7
`giza`	Apriori	BA	66.1	50.2	96.7
`giza`	Bez Apriori	BA	66.1	49.8	98.0
`raw`	Bez Apriori	A	58.2	41.3	98.9 ← baseline
`freq`	Apriori	BA	56.5	52.9	60.6 ← nejlepší přesnost
`vjv`	Apriori	BA	55.9	41.4	86.3 ← nejlepší z `vjv`
`vjv`	Apriori	C	30.0	33.5	27.2 ← nejhorší výsledek

Tab. 3. Přesnost a pokrytí algoritmů pro překlad forem slovesných doplnění.

Tabulka 3 shrnuje základní ukazatele o kvalitě predikce pro vybrané kombinace předzpracování a použité metody. Přesnost je počet správně zvolených

[9] Je zřejmé, že české věty je možné přeložit do angličtiny více různými způsoby, volit různá slovesa s rozdílnou sadou forem doplnění ap. Pro jednoduchost však tento problém necháváme zatím stranou a opíráme se o jediný referenční překlad.

překladů z celkového počtu odevzdaných překladů, pokrytí je počet slotů, k nimž algoritmus navrhl nějakou odpověď, z celkového počtu slotů. Výsledky jsou uspořádány podle tzv. F-score, které představuje geometrický průměr přesnosti a pokrytí.

Nejlepšího výsledku jsme dosáhli při použití nejprostší metody A opírající se o data po odstranění vět s nespolehlivým párováním a po použití algoritmu Apriori pro zjednodušení pozorovaných překladových rámců. S mírným odstupem následují kombinace CBA a BA na stejně předzpracovaných rámcích. Již samotné odstranění nespolehlivě spárovaných vět (bez nasazení metody Apriori) stačí algoritmu BA k dosažení velmi podobných výsledků. Tyto nejlepší výsledky představují zlepšení o 10 procentních bodů v přesnosti za cenu nepatrného poklesu v pokrytí proti metodě, kterou lze považovat za zcela základní (baseline): metoda A na nijak nefiltrované pozorované rámce.

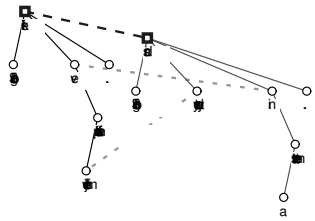
Jak je z výsledků zřejmé, největší vliv na kvalitu predikce překladové formy má kvalitní párování slov. Čištění pomocí Apriori má rovněž svůj význam, z implementovaných metod A, B a C však nejlépe funguje nejjednodušší metoda A. Kvalitě predikce naopak nejvíce ublížilo, když jsme překladové rámce získávali pouze z vět, kde česká věta je "velmi jednoduchá, `vjv`. Vysvětlení pro tento neúspěch spočívá pravděpodobně v tom, že popsaný algoritmus kombinující dva závislostní stromy pomocí párovacích hran nenajde v (typicky krátkých) velmi jednoduchých větách dostatek podkladů pro rozumné přiřazení slotů k sobě.

4.3 Pomíjené problémy překladu syntaktického okolí sloves

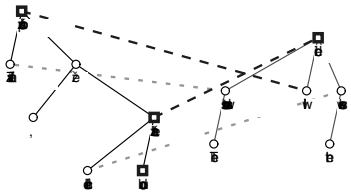
Při našich prvních experimentech pro jednoduchost přehlížíme významné problémy způsobené syntaktickou odlišností češtiny angličtiny, jak ji pozorujeme v korpusu ekonomických textů (PCEDT). V dalším výzkumu se však na tyto problémy zaměříme podrobněji a zohledníme je v našem algoritmu automatizované extrakce.

Posun doplnění. Jak ukazuje obrázek 1, v některých případech mírně odlišný překlad způsobí nemožnost přesného párování jednotlivých doplnění na sebe. Vhodným řešením by bylo takové případy identifikovat a do slovníku zařadit jen ty sloty, které si odpovídají.

Záměna řídících uzlů (Head Switching). U některých konstrukcí dochází při přechodu od českého závislostního stromu k anglickému k záměně řídících uzlů. (Např. ve větě *Nákupní firmy se* mohou *stát ziskovými. = Malls can* become *profitable.* je jako kořen v češtině použito způsobové sloveso, v angličtině se však užívá typicky sloveso plnovýznamové.) Je třeba pečlivě odlišovat situace, kdy je záměna způsobena pouze jinou dohodou o anotaci a kdy jde o závažnější syntaktickou odchylku (viz obrázek 2). Dobrým rámcem pro toto odlišení by mohla být závislostní redukční analýza, [16].



Obr. 1. Sigoloff řekl ve včerejším prohlášení ... Sigoloff said yesterday in a statement ...



Obr. 2. Zavírání způsobí , že dělníci budou zahálet ... The shutdowns will idle the workers ...

5 Shrnutí a další plány

Popsali jsme postup prací, které probíhají na přípravě česko-anglického slovníku pro strojový překlad. Dostupné strojové slovníky kombinací ručních a automatických metod obohacujeme o potřebné morfologické a syntaktické údaje.

V dalším výzkumu se kromě dalšího zlepšování metod automatické identifikace slovesných rámců zaměříme též na možnosti formálního zachycení pozorovaných syntaktických odlišností ve slovníku.

Reference

1. Dorr B.J., Jordan P.W., Benoit J.W., A Survey of Current Paradigms in Machine Translation. Technical Report CS-TR-3961, 1998
2. Čmejrek M., Cuřín J., Havelka J., Czech-English Dependency-based Machine Translation. In: EACL 2003 Proceedings of the Conference, Association for Computational Linguistics, 2003 83–90 MSM113200006, LN00A063
3. Hajič J., RUSLAN: an MT System between Closely Related Languages. In: Proceedings of the Third Conference on European Chapter of the Association for Computational Linguistics, Association for Computational Linguistics, 1987, 113–117
4. Svoboda M., GNU/FDL English-Czech Dictionary, 2001 `http://slovnik.zcu.cz/`
5. Pala K., Smrž P., Building Czech Wordnet. Romanian Journal of Information, Science and Technology 7, 2004, 79–88
6. Hajič J., Disambiguation of Rich Inflection - Computational Morphology of Czech. Volume I. Prague Karolinum, Charles University Press, 2001, 334 pp.
7. Kocek J., Kopřivová M., Kučera K., eds., Český národní korpus - úvod a příručka uživatele. FF UK - ÚČNK, Praha, 2000
8. Böhmová A., Hajič J., Hajičová E., Hladká B., The Prague Dependency Treebank: Three-Level Annotation Scenario. In Abeillé, A., ed.: Treebanks: Building and Using Syntactically Annotated Corpora. Kluwer Academic Publishers, 2001
9. Čmejrek M., Cuřín J., Havelka J., Hajič J., Kuboň V., Prague Czech-English Dependecy Treebank: Syntactically Annotated Resources for Machine Translation. In: Proceedings of LREC 2004, Lisbon, 2004
10. Charniak E., A Maximum-Entropy-Inspired Parser. In: Proceedings of NAACL-2000, Seattle, Washington, USA, 2000 132–139

11. Lopatková M., Bojar O., Semecký J., Benešová V., Žabokrtský Z., Valency Lexicon of Czech Verbs VALLEX: Recent Experiments with Frame Disambiguation. In: Proceedings of TSD 2005, 2005, (in press)
12. Zeman D., Sarkar A. , Learning Verb Subcategorization from Corpora: Counting Frame Subsets. In: Proceedings of the Second International Conference on Language Resources and Evaluation (LREC 2000), Athens, Greece, ELRA, 2000
13. Och F.J., Ney H., A Systematic Comparison of Various Statistical Alignment Models. Comput. Linguist. 29, 2003, 19–51
14. Bojar O., Towards Automatic Extraction of Verb Frames. Prague Bulletin of Mathematical Linguistics, 2003, 101–120
15. Agrawal R., Imieliński T., Swami A., Mining Association Rules between Sets of Items in Large Databases. In: SIGMOD'93: Proceedings of the 1993 ACM SIGMOD international conference on Management of data, New York, NY, USA, ACM Press (1993) 207–216
16. Lopatková M., Plátek M., Kuboň V., Závislostní redukční analýza přirozených jazyků. In: Proceedings of ITAT 2004, Košice, Slovakia, University of P. J. Šafařík, 2005, 165–174

Virtuální lidé: jak representovat jejich znalosti

Cyril Brom

Karlova Universita, Matematicko-fyzikální fakulta
Malostranské nám. 2/25, Praha, Česká republika
brom@ksvi.mff.cuni.cz

Abstrakt. Fenomén virtuálních lidí proniká v poslední době i mimo oblast počítačových her. Jedná se softwarové agenty, kteří imitují lidské chování v nějakém virtuálním světě a mají simulované, obvykle graficky zobrazené tělo. V tomto článku je představen studentský softwarový projekt IVE, zaměřený na modelování rozsáhlých světů s virtuálními lidmi. V rámci projektu jsou řešeny hlavní problémy: a) jak representovat svět tak, aby byl snadno rozšiřitelný, tedy mimo jiné aniž by bylo nutné používat pro agenty technik strojového učení, b) jak automaticky zjednodušovat simulace v místech, kam se uživatel „nedívá“. Řešení obou problémů je v článku nastíněno.

1 Úvod

Virtuální lidé jsou inteligentní softwaroví agenti (ve smyslu Wooldridge [14]), kteří imitují lidské chování v prostředí simulujícím přirozený svět a kteří mají virtuální tělo modelující tělo lidské. Prostředí a agentovo tělo jsou obvykle zobrazeny graficky.

Cílem článku je seznámit čtenáře s tímto fenoménem, představit studentský softwarový projekt IVE [6] (*intelligent virtual environment*) a problémy, které jsou v jeho rámci řešeny. Kapitola 2 blíže pojednává o virtuálních lidech – je zpřesněna jejich definice, jsou zasazeni do kontextu softwarového inženýrství a aplikované umělé inteligence a jsou popsány oblasti aplikací. Kapitola 3 vymezuje problémy, na než se projekt IVE zaměřuje. Kapitoly 4 a 5 nastiňují, jakými způsoby jsou problémy řešeny. Příbuzné práce a práce, ze kterých IVE těží, jsou diskutovány průběžně.

2 Virtuální lidé – co jsou zač?

Každý virtuální člověk (*artificial human*, *virtual being*) jedná ve virtuálním světě a vykonává úkoly, jež předem vymezil programátor nebo designér. Rozhraním do světa je *umělé tělo*. Pro označení řídící funkce agenta se často používá pojem *umělá mysl* (*artificial mind*). Jde o funkci, která řeší problém výběru akcí (*action selection problem*) – tj. co má agent provést v dalším kroku simulace v závislosti na aktuálním stavu svého těla, své náladě a záměrech (tj. stavu mysli) a stavu okolního prostředí. Typicky jedná virtuální člověk v následujícím cyklu: 1) přijme podněty od prostředí, 2) zpracuje podněty, 3) vybere následující akci, 4) odešle své rozhodnutí prostředí.

V následujícím textu bude termín „virtuální člověk“ volně zaměňován za „aktor“ (*virtual actor*).

Následující vlastnosti dobře vymezují virtuální lidi od podobných softwarových agentů:

- virtuální lidé pouze *předstírají*, že se chovají jako lidé; mají uživatele *oklamat*, tj. chovat se uvěřitelně (*believable*), uspět v Turingově testu (prožívání skutečnosti, proces myšlení ap. do domény virtuálních lidí nepatří),
- mají tělo a simulují většinou performativní akce, tj. akce, jejichž efekt je „fyzicky vidět“ (šachový program není virtuální člověk, leda by měl tělo a pohyboval figurkami po 3D/pseudo-3D šachovnici),
- jednají v prostředí, jenž je dynamické (tj. mění se i bez přičinění agenta a nečeká se, až agent naplánuje, co bude dělat), nedeterministické (tj. výsledek akce nelze vždy přesně předpovědět) a pouze částečně pozorovatelné (agent vnímá svými senzory vždy jen část prostředí, nikoli celé prostředí naráz); obvykle dochází na interakci s člověkem,
- očekává se, že virtuální lidé budou navrhováni designéry, konstruktéry nebo umělci, tedy nikoli lidmi s hlubokým IT vzděláním.

Virtuální lidé pronikají posledních několik let i mimo oblast počítačových her. Přehled udává například [12], v následujícím je vyjmenováno několik nejtypičtějších oblastí použití. Některé projekty jsou čistě výzkumné, jiné jsou již nasazeny v praxi.

- **Výukové aplikace** (*educational applications*)**:** Pedagogičtí aktoři učí studenty pracovat s drahým zařízením (př. vysokotlaký kompresor) nebo jednat v situaci, kterou lze jen obtížně nasimulovat (př. voják v Bosně uklidňuje matku, jejíž dítě srazilo armádní auto – viz obr. 1). Vše se odehrává ve virtuálním prostředí.

Obr. 1: Výuková aplikace pro vojáky mírové mise v Bosně. Armádní vozidlo přejelo bosenského chlapce. Vidíme jeho matku a lékaře (chytré agenty), v popředí seržanta (tutora, který žáka–vojáka provádí lekcí) a v pozadí dav zvědavců (jednoduché agenty). Převzato z [5].

- **Virtuální vyprávění** (*virtual storytelling*)**:** Jde o „přehrání příběhu na počítači“, přičemž příběhem se myslí např. krátká novela nebo divadelní hra. Jedná se o aplikace z pomezí umění. Často je uživateli dovoleno zasahovat do průběhu děje. Narozdíl od počítačové hry je kladen silný důraz na narativní strukturu děje.
- **Film:** Virtuální lidé nahrazují komparsisty při davových scénách. Ovšem stovkami postav nepohybuje animátor obrázek po obrázku, nýbrž má každá postava svůj program a vlastnosti a výsledná interakce vzniká emergentně (představme si například souboj dvou armád).
- **Virtuální performance:** Jde o čistě umělecké projekty, jejichž médiem je virtuální realita.
- **Léčba fobií:** Při léčení fobií je možné použít metodu, kdy je pacient opakovaně vystavován situacím navozujícím jeho fobii. Ukazuje se, že pacienti reagují, i když jsou situace modelovány ve virtuálním prostředí s virtuálními aktory.
- **Verifikace návrhu strojů.** Inženýři, kteří navrhnou v nějakém CAD systému model stroje, si kladou otázku, jestli je stroj navržen dobře z hlediska ovládání. Tím se myslí, jestli může stroj ovládat jak malý, tak velký člověk, jestli ovládání není nebezpečné (př. horká plocha těsně vedle často používané páčky) atp. Verifikaci modelu lze provést ve virtuálním prostředí provedením série experimentů s virtuálními lidmi.

Podle typu aplikace musí vývojáři řešit některé z následujících problémů:

- **Návrh struktury chování.** Je třeba říci, jaké úkoly bude virtuální člověk provádět a jak budou strukturované.
- **Problém výběru akcí.** Řeší se, jak v daném kontextu vybrat akci, kterou má aktor provést. Problém je, že virtuálnímu člověku nelze předem vložit deterministická procedura, ani se nemůže zastavit a pět minut přemýšlet nad dalším „krokem“ (jako například šachový program). Prostředí se mění aktorovi „pod rukama“.
- **Grafická uvěřitelnost.** Agent se musí pohybovat co nejvíce „lidsky“. Ukazuje se, že důležitou roli hrají i takové jemnosti jako exprese emocí či pohyb očí.
- **Věrohodná fyzikální simulace.** Agenti jednající v 3D prostředí nemohou například procházet skrze předměty.
- **Udržení dějové linie.** Je-li povolena interakce s uživatelem, může tento mít tendenci jednat proti smyslu předem navrženého děje.
- **Komunikace.** Specielně ve výukových aplikacích musí virtuální učitelé mluvit přirozeným jazykem studenta a také porozumět psané nebo mluvené řeči.

V každém případě musí být výsledek výpočetně efektivní; virtuální lidé nejsou teoretické modely, ale reálně pracující aplikace.

Šíře aplikací a řešených problémů ukazuje, že virtuální lidé jsou interdisciplinární fenomén. Přibližně lze obory, do nichž virtuální lidé spadají, vymezit následovně. Hlavními disciplínami jsou softwarové inženýrství, počítačová grafika a aplikovaná umělá inteligence. Vzhledem k tomu, že předlohou modelům je skutečný člověk, přichází silná inspirace z psychologie a etologie (jde např. o teorii lidského chování nebo modelování emocí). Nezanedbatelný je i vliv biologie a neurologie. K simulaci 3D světů jsou nutné fyzikální modely.

Je třeba jasně říci, že virtuální lidé nejsou a nemají být vševědoucí golemové ani živé bytosti. Každý virtuální člověk jedná v omezené doméně a jeho návrháři řeší pouze problémy, které bezprostředně vyplývají ze zamýšleného použití. Od virtuálních lidí nelze očekávat „skutečnou inteligenci". Jako příklad si vezměme e-Františka sedícího ve virtuální kanceláři za účelem navození situace, v níž pacient trpí úzkostí. E-František se nebude zaobírat tím, co bude dělat večer, nýbrž bude „přemýšlet", jak se má „poošít" v další vteřině, aby vypadal jako člověk, jehož bolí záda.

3 IVE – problémy a motivace

Projekt IVE [6] je studentský softwarový projekt, realizovaný od listopadu 2004 na MFF UK, Praha. Finálním cílem je vytvořit universální nástroj na prototypování chování virtuálních lidí jednajících v rozsáhlých světech. V tuto chvíli se řeší pouze první fáze, jejímž cílem je navrhnout obecnou architekturu virtuálního světa, která bude umožňovat snadnou rozšiřitelnost a efektivní simulaci rozsáhlých světů, a implementovat prototyp. Nyní budou blíže vysvětleny pojmy *snadná rozšiřitelnost* a *efektivní simulace*.

Problém 1: Snadná rozšiřitelnost. Aplikace s virtuálními lidmi bývají typicky vyvíjeny inkrementálně, často je požadována pozdější rozšiřitelnost (uvažme např. plug-iny v počítačových hrách). Vzniká tak potřeba mechanismu, který umožní virtuálním lidem interagovat i v novém prostředí, např. s předměty, které přibyly a žádný aktor se s nimi dosud nesetkal. Ovšem není možné požadovat po uživatelích nebo designérech, aby přepisovali kvůli novému objektu či úkolu kompletně celou simulaci. Natož pak aby používali složitých technik umělého učení (pokud jejich použití není hlavním záměrem aplikace). Jde tedy o adaptabilitu na nové podněty, respektive o její předstírání.

Snadnou rozšiřitelností se rozumí to, že do virtuálního světa bude možno přidávat předměty nebo úkoly pro virtuální lidi jako plug-iny, aniž by se používalo technik učení nebo se výrazně modifikovalo to, co už bylo naprogramované dříve. Kapitola 4 popisuje, jak je tento požadavek v IVE řešen.

Problém 2: Efektivní simulace rozsáhlých světů. Existuje mnoho teoretických modelů lidského rozhodování. Problém je, že modelují systém (člověka), který je do značné míry paralelní. Prostým přenesením modelu na PC se sériovou architekturou získáme NP-úplné algoritmy. To nemusí vadit, je-li aktor sám, ovšem většinou jich ve virtuálním světě jedná více.

Efektivní simulací rozsáhlého světa se intuitivně rozumí to, že relativně veliký svět (řádově desítky postav) poběží na současném běžném PC „rozumě". Neznamená to ale nezbytně, že algoritmy řízení virtuálních lidí budou polynomiální. Kapitola 5 popisuje, jak je tento požadavek v IVE řešen.

Stojí za podotknutí, že i izolované řešení pouze těchto dvou problémů (tj. aniž by existoval proklamovaný universální prototypovací nástroj) je přínosné. Snadná rozšiřitelnost je požadavkem téměř všech aplikací s virtuálními lidmi a efektivní simulace (ve výše uvedeném smyslu) většiny počítačových her a virtuálních vyprávění.

4 Snadná rozšiřitelnost = vhodná representace

Klíčem k snadné rozšiřitelnosti virtuálního světa je jeho representace. Uvědomme si, že jednání virtuálního člověka je determinované množinou objektů a jejich stavů a povolených transformací stavů objektů. Jsou to právě tyto objekty a transformace (dále *akce*), které je nutné representovat na počítači.

Typický přístup je *symbolický*, kdy zjednodušeně řečeno objektům, které chceme simulovat, přiřadíme nálepky jako „konev12“, „pozice_v_mapě27_12“ a relace jako „množství_vody(konev12, 25)“. Není cílem se zde pouštět do debaty o tom, jestli je tento přístup biologicky plausibilní, či jestli již byl nebo nebyl překonán, faktem nicméně zůstává, že z hlediska designu virtuálních lidí je nevhodný. Důvod je prostý: symboly jsou právě jenom nálepky a mají význam pouze pro designéra, avšak nikoli pro virtuálního člověka. S takovou formální representací se dá dobře plánovat, ale to je asi tak vše. Chceme-li přidat hadici, se kterou lze zalévat podobným způsobem jako s konví, musíme každému aktorovi, který s hadicí může přijít do styku, tento fakt „sdělit“ – tj. přímo ho zapsat do jeho umělé mysli – nebo to aktora naučit. Navíc jelikož je virtuální svět jen omezeně vnímatelný, musí si aktor pro potřeby rozhodování vytvářet v paměti kopii světa (symbolů). Pomněme, co se asi tak stane, když přijde virtuální zahradník k záhonu, kde roste 150 mrkví.

Problém je, že výše uvedeným způsobem se pouze budují jakési ontologie objektů a jejich vlastností založené na fyzikálním chápání světa. Virtuální lidé ale málokdy něco takového potřebují. Podstatná informace je, že předmět lze sebrat a že s ním lze zalévat, nikoli, že leží na dlaždici27_12 a je v něm 25 jednotek vody. Jiná klíčová informace může být, že na zahradě je možné provést určité akce, které způsobí, že aktor bude držet v ruce jídlo (např. mrkev). Zkrátka: je třeba jiný typ (symbolické) representace.

Tato representace vychází z chápání světa prostřednictvím ontologie akcí (nebo, chceme-li, služeb), které nám předměty mohou poskytnout. Aktor tak má určité záměry (např. najíst se, zalévat) a předměty okolo něj mu nabízí („křičí“ na něj), že tyto záměry uspokojí a jak. Dá se říci následující:

1. Předmět je agent (ve smyslu Wooldridge [14]), který obsahuje kompletní návod pro aktora, jak provést danou interakci (tj. obsahuje řídící skript). Obsahuje i zobrazovací informace pro grafický prohlížeč.
2. Aktor, který chce naplnit daný záměr, se „rozhlédne“ po světě, nalezne předmět, který nabízí, že uspokojí jeho záměr, „napojí“ se na něj a nechá se jím dále navigovat. Nové předměty uspokojující již existující záměry tedy mohou být nahrávány do prostředí jako plug-iny, aniž by byl nutný zásah do mysli aktora.

Pochopitelně nemizí potřeba převodu fyzikální representace (v konvi je 25 jednotek vody) na representaci orientovanou na akce (konev uspokojí záměr zalévat). Ovšem tento problém nyní neřeší mysl aktora, nýbrž konkrétní předmět – problém výběru akcí byl dekomponován. Svět (předměty) nyní „hovoří“ přímo jazykem záměrů. To je novinka, v klasické representaci svět hovořil jazykem fyzikálních vlastností, se kterými si mysl musela nějak poradit.

Tento přístup se nazývá metoda chytrých předmětů (*smart object approach*), byl navržen v [7] a úspěšně implementován a rozšiřován mj. v [1], [8], [11]. Chytré

předměty není těžké hierarchicky vnořovat (dveře nabízí otevření a v rámci toho klika nabízí stisknutí) – tím lze lehce vyřešit problém se stovkami mrkví. Aktor totiž mrkve vůbec neuvidí, uvidí pouze záhon nabízející buď přímo uspokojení hladu, nebo zisk předmětu, jenž hlad uspokojí. V druhém případě za aktora vybere mrkev k vytrhnutí a spustí skript, jenž způsobí, že se mrkev ocitne v aktorově ruce.

Přístup lze rozšířit na pravděpodobnostní (aktor pokaždé jedná trochu jinak) – a takové implementace také existují. Řeší se i problémy s viditelností a s pamětí (aktor chce sebrat konev, ví kde je, ale nevidí ji).

Metoda chytrých předmětů je založena na teorii afordancí, což je teorie lidského vnímání percepčního psychologa J. J. Gibsona [4] (1979). Gibson tvrdí, že máme (lidé a zvířata) tendenci vnímat svět nikoli prostřednictvím fyzických vlastností objektů (má to čtyři nohy, desku a opěradlo, *ergo* jde o židli), nýbrž skrze afordance. To jsou jakési nabídky, které nám prostředí činí – „...the *affordances* of the environment are what it *offers* the animal, what it *provides* or *furnishes* [4, p.127]". Klíčové je, že jsou relativní k aktorům (pes má jiné afordance než člověk) a trvalé a nezávislé na záměrech agenta, ba dokonce ani na schopnosti je vidět. Umět vidět afordance a mít určité afordance jsou dvě různé věci. Bližší vhled podá např. [3].

IVE rozšiřuje metodu chytrých předmětů třemi směry. Jsou zavedeny *chytré procesy*, *chytré rady* a *géniové*. V následujícím jsou tato rozšíření popsána.

Chytré procesy. Pokud je „inteligence" schovaná do chytrých předmětů, zkomplikuje se tím popis interakce mezi více objekty (např. opice shazuje ze stromu banány klackem nebo člověk zavírá pero víčkem). Proto byl zaveden koncept *chytrých procesů*. Jde o podobné agenty jakými jsou chytré předměty, jen nejsou vázány na žádný konkrétní objekt. Spíše jde o jakési „transcendentální entity", které agentovi nabízí, co ve virtuálním světě může provádět. Proces může popisovat interakci jak mezi aktorem a jedním zdrojem (zde získáváme původní chytré předměty), tak mezi aktorem a více předměty, stejně jako interakci mezi více agenty. Takové objekty nebo aktoři se pak stanou *zdroji* procesu. Neříkáme však „virtuální Pepa má skript A, který pracuje se zdrojem B", nýbrž „virtuální Pepa má záměr Z, který může být uspokojen nějakým procesem, který může pracovat s nějakými objekty – a to právě takovými objekty, které mohou figurovat v daném procesu jako zdroje."

Chytré rady. Uživatel často očekává, například na základě grafické vizualizace, že různí aktoři se ve stejných situacích budou chovat různě. Tak např. tlustá Barka, chce-li uspokojit záměr „najíst se", dá přednost hromadě cukroví, zatímco sportovec Pepa by raději salát. Pro řešení tohoto problému jsou v IVE zavedeny speciální funkce, *rady*, které radí aktorům, jaký proces nebo variantu procesu mají provádět v daném kontextu, aby uspokojili nějaký záměr. Výkonnou část rady lze vidět jako fuzzy if-then pravidlo operující nad vlastnostmi aktora a objektů, jež se mají účastnit daného procesu. Dá se říci, že chytré rady zprostředkovávají aktorům vidění těch správných (resp. správně instanciovaných) procesů. Ilustrováno je to ve scénáři na obr. 2.

Géniové. Aktoři se musí rozhodovat nejen jaký proces (resp. jak instanciovaný proces) mají spustit za účelem naplnění daného záměru (tzv. *means-ends reasoning*), ale také jaký záměr budou uspokojovat (tzv. *deliberative reasoning*). V tom jim chytré rady mohou pomoci, ale ne vždy je tento základní typ rozhodování vhodný.

1. Jsem aktor líná Barka. Mám hlad. Je tu někdo, kdo mi ho „uspokojí“?
2. Ano, jsem proces, co uspokojí tvou žravost. Mám tu koláče a salát. Co bys raději?
3. Barka: Nevím. Nevíš o někom, procese, kdo by mi poradil?
4. Proces. Ano, mám tady chytrou radu. Tady ji máš.
5. Barka spouští chytrou radu a předává jí vektor svých vlastností.
6. Chytrá rada: V tvém případě rozhodně doporučuji koláče.
7. Barka: Procese, uspokoj mě prostřednictvím koláčů.
8. Proces se spustí na objektu „hromada koláčů“.

Obr. 2. Ukázka komunikace dvou agentů (aktora a procesu) a chytré rady tohoto procesu. Na příkladu je vidět, že aktor se dokáže najíst, aniž by musel „vidět“ byť jen jediný konkrétní koláč. Podobná komunikace může proběhnout i při obecném aktorově rozhodování mezi libovolnými *n* procesy.

Rady jsou pouze fuzzy if-then pravidla a občas je třeba použít jinou techniku (např. podívat se pomocí dopředného plánování, jak akce pravděpodobně dopadne). Obecně se dá říci, že pro různé kontexty se hodí jiné techniky. Proč?

Uvažme jako příklad Pepu v hospodě. Pepa bude pít pivo. Ovšem nikoli proto, aby uspokojil svůj záměr „bavit se“ nebo „nemít žízeň“, nýbrž proto, že uživatel očekává, že v hospodě chlap, byť virtuální, bude pít pivo. Při rozhodování se tedy využije například vlastnost „jak_je_Pepa_veliký_opilec“. Na poušti se ovšem pro podobné rozhodování použije vlastnost „biologická_žízeň“, jejíž hodnota se v hospodě vůbec nesledovala. Navíc rozhodování dalších aktorů se bude odvíjet od stejného schématu (až na to, že někdo bude např. raději pít pivo černé, jiný vodku atp.).

Je tedy vhodné dekomponovat rozhodování aktorů podle situací, ve kterých se mohou nacházet. Slovy plánovacích technik: každá situace vyžaduje vlastní specifické informace pro rozhodování, jakousi vlastní „heuristiku“. Za tím účelem je v IVE zaveden koncept géniů.

Génius je samostatná inteligence, která umí po určitou dobu převzít v daném prostředí a situačním kontextu jednoho či více aktorů a řídit je. Původní mysl aktora je buď zcela uspaná, nebo se podílí na řešení dílčích úkolů (např. uhýbání před překážkami). Pepu tak v hospodě bude řídit génius hospody, zatímco na poušti génius pouště. Koncept géniů navíc umožňuje rozšiřitelnost, která nám doposud chyběla – o nové záměry.

Této technice se říká předávání rolí (*role-passing*) a byla implementována např. v [10]. Obvykle se ale role předává pouze jednomu aktorovi. Génius je obecnější koncept. Dokáže řídit více aktorů naráz a tím umožňuje centrální rozhodování a synchronizaci aktorů – například při rozhodování na poušti o tom, kdo dostane láhev s vodou jako první a kdo jako poslední. To umožňuje třebas vygenerování barvitého dialogu (což se lépe, než kdyby všichni aktoři jednali samostatně, provede centrálně). Společný génius je jakousi sdílenou myslí.

Chytré procesy, chytré rady, zdroje procesů (tj. předměty a aktoři) a géniové tvoří základní strukturu IVE světa. Pro kapitolu 5 bude klíčové, že tato struktura je hierarchická.

Dodejme, že tato struktura může připomínat architekturu Belief-Desire-Intention (viz např. [14]). To je pravda, v IVE je ale myšleno i na jiné přístupy a jejich kombinace. Konkrétně mají aktoři možnost odmítnout chytré rady a „heuristiku“, jež se jim nabízí v podobě hierarchické struktury procesů – a mohou se rozhodnout používat jen takové procesy, které jsou už přímo základními transformacemi vlastností objektů (tzv. *atomické procesy*). To umožňuje imitovat učení. Dále mají možnost odmítnout nechat se vést cizím géniem (např. Pepa se může v hospodě rozhodovat sám, chce-li). Nicméně přístup zvolený v IVE si zatím neporadí s řídícími technikami, jež nejsou založené na hierarchickém rozkladu úloh (např. [9]).

5 Efektivní simulace = ne-simulace

Pro rozsáhlé simulace je charakteristické, že uživatel sleduje pouze část prostředí. To znamená, že ta místa, na než se uživatel „nedívá“, není nutné simulovat přesně – a tedy je možno ušetřit výpočetní prostředky. Tento fakt je hojně reflektován v počítačové grafice používáním techniky úrovně detailů (*level of detail*, dále *LOD*). Není třeba počítat povrch hlavice iónského sloupu, když má celý sloup méně než jeden pixel.

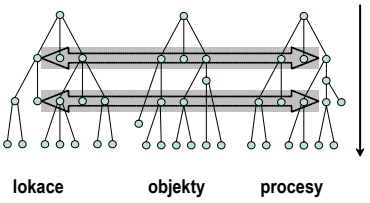
V oblasti simulace chování je faktu občas využíváno v počítačových hrách – leč většinou jenom tak, že se místa mimo výhled uživatele nesimulují vůbec. To často způsobuje nekonzistenci děje – uživatel si „podvodu“ všimne. Můžeme „podvádět“, ovšem uživatel nesmí naše klamy poznat. V IVE je tento problém řešen robustním a kompaktním způsobem, který bude nyní popsán. Klíčové pro řešení je, že znalosti virtuálních lidí jsou representovány hierarchicky prostřednictvím chytrých procesů.

Předně je třeba pro celý virtuální svět určit, jak precisně má být v tom kterém místě simulován. Každé místo musí mít stanovenu *úroveň detailu*. Vyšší úroveň koresponduje s místy, na něž se uživatel dívá, s místy sousedícími s výhledem a konečně s místy, kde se odehrávají události důležité pro další průběh děje.

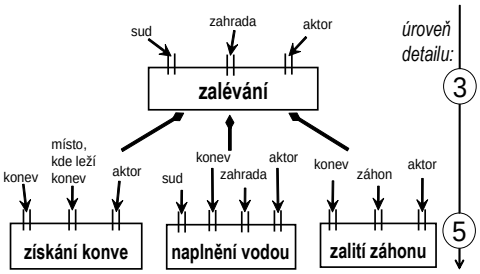
Dále je třeba stanovit, jak konkrétně má simulace pro danou úroveň detailu proběhnout. Právě k tomu je v IVE využito hierarchické dekompozice procesů. Každý proces je přímo asociován se skriptem, jenž provádí požadovanou transformaci světa, a každá úroveň v hierarchii procesů odpovídá jedné úrovni detailu. Čím vyšší detail, tím hlouběji v hierarchii procesů jsme. Například zahradu lze zalít tak, že je odečteno příslušné množství vody ze sudu a je změněna všem záhonům vlastnost „zalitost“ (to je nízká úroveň detailu). Anebo lze na zalévání použít podprocesy pro nalezení a zvednutí konve, pro naplnění konve (opakované) a pro zalití každého jednotlivého záhonu (to je vyšší úroveň detailu). Schématicky je to znázorněno na obr. 4.

Každý proces dokáže odhadnout dobu svého trvání. Pro procesy ve vrstvě korespondující s aktuální úrovní detailu je tento odhad vzat jako skutečná doba trvání a konec procesu je naplánován do simulačního *kalendáře*. Pochopitelně je možné, že proces bude předčasně přerušen, stejně tak může dojít k zvýšení úrovně detailu v průběhu nějakého procesu – tyto situace jsou v IVE řešeny.

Vedle hierarchické dekompozice procesů je třeba representovat ještě další dvě dekompozice. Jednak je nutná dekompozice lokací, k nimž budou vázány úrovně detailů a v nichž budou probíhat procesy. A dále je potřeba dekompozice objektů –



Obr. 3: Při horizontálním řezu hierarchiemi si úrovně vzájemně odpovídají. Procesy a objekty můžeme „nahrávat“ do hierarchie jako plug-iny.



Obr. 4.: Dekompozice zalévání na subprocesy. Každý obdélník značí proces s požadovanými zdroji.

tím se myslí to, že všechny objekty patří k nějaké nadřazené komponentě, což je buď objekt (např. klika je součástí dveří) nebo lokace (např. půllitr je součástí hospody). Při nízké úrovni detailu podřazený objekt nebude existovat. Při zvýšení úrovně bude tento vygenerován nadřazenou komponentou. Objektem se rozumí i aktor; např. virtuální vesničané v místě, které se právě nesimuluje, vůbec nebudou existovat.

Aby celý systém fungoval, je potřeba dekompozice udělat tak, že povedeme-li trojicí hierarchií libovolný horizontální řez, dostaneme vždy procesy, lokace a objekty, které spolu vzájemně souvisí (viz obr. 3). Například konev musí existovat s alespoň stejnou úrovní detailu, se kterou existuje i proces zalévání s konví. Obecně zřejmě nelze provést ony tři dekompozice tak, aby řez byl možný; problém budou činit minimálně rekurzivní procesy. Otázka konkrétních podmínek, za kterých dekompozice provést lze, zůstává zatím otevřená.

Nástin, jak použít pro LOD simulace virtuálních lidí hierarchické konečné automaty, je podán v [2], jde ale jen o myšlenku. Částečná implementace založená na technice předávání rolí je popsána v [13]. Přístup v IVE je robustnější, protože počítá i s dekompozicí lokací a objektů – v místech, kam se nikdo nedívá, nejen že aktor nedostane svou roli a nebude nic provádět, takové místo vůbec nebude existovat.

Existuje mnoho zajímavých situací, které při použití LOD techniky vznikají. Uvažme například přemisťování předmětů mimo jejich nadřazené komponenty (půllitr na zahradě), procesy probíhající ve více lokacích (vozík vyjíždí z dolu, kde je nižší úroveň detailu než v lokaci nad dolem) a koneckonců podstatná je i otázka režie přepínání mezi různými úrovněmi detailu. To vše bude diskutováno v samostatném článku někde jinde.

6 Závěr

V tomto článku byl představen fenomén virtuálních lidí a typy jejich aplikací. Mimo počítačové hry jde například o virtuální vyprávění a drama, generování scén do filmu, výukové aplikace či verifikování návrhu strojů. Byl rovněž uveden výčet hlavních problémů, které se v těchto aplikacích řeší.

Dále byl představen studentský softwarový projekt IVE. Zkratka IVE znamená inteligentní virtuální prostředí (*intelligent virtual environment*). Inteligence simulátoru spočívá ve dvou skutečnostech. Za prvé v tom, že prostředí pomáhá řídit virtuální lidi pomocí chytrých procesů. Aktoři sami být inteligentní nemusí, nýbrž se mohou nechat inteligentně řídit okolím – jejich znalosti jsou částečně representovány v prostředí, nikoli v jejich mysli. Tento přístup výrazně usnadňuje design a rozšiřitelnost. Druhý aspekt inteligence IVE spočívá v zjednodušování simulace v místech, kam se uživatel „nedívá“. To je možné díky tomu, že znalosti virtuálních lidí jsou representovány hierarchicky.

Prototypový scénář implementovaný v rámci projektu [6] zahrnuje ves s několika domy, hospodou, dva doly a několik desítek aktorů.

Poděkování. IVE je implementováno na MFF UK, Praha studenty 4. ročníků: O. Šerým, T. Pochem, P. Šafratou, J. Kubrem, J. Kulhánkem a Z. Šulcem, jímž tímto patří dík. Cenné komentáře k článku měli: T. Holan a E. Dufková. Projekt je částečně podpořen grantem „Information Society“ pod číslem projektu: 1ET100300517.

References

1. Badler N., Allbeck J., Zhao L., Byun M., Representing and Parameterizing Agent Behaviors. In: *Proceedings of Computer Animation*, IEEE Computer Society, Geneva, Switzerland, 2002, 133-143
2. Champandard A.J., AI Game Development: Synthetic Creatures with learning and Reactive Behaviors. New Riders, USA, 2003
3. McGrenere J., Ho W., Affordances: Clarifying and evolving a concept. In: Proceedings of Graphics Interface 2000, 2000, 179-186
4. Gibson J.J., The Ecological Approach to Visual Perception. Boston: Houghton Muffin, 1979
5. Marsella S., Gratch J., Rickel J. Expressive Behaviors for Virtual Worlds. In: Prediger, H., Ishizuka, M. (eds.): Life-like Characters Tools, Affective Functions and Applications, Cognitive Technologies Series, Springer, Berlin, 2003
6. IVE, projektová stránka: `http://mff.modry.cz/ive/`
7. Kallmann. M., Thalmann D., Modeling Objects for Interaction Tasks. In: Proceedings of EGCAS 98, Lisbon, Portugal, 1998, 73-86
8. Kallmann M., Thalmann D., Modeling Behaviors of Interactive Objects for Virtual Reality Applications. In: Journal of Visual Languages and Computing Vol. 13, 2002
9. Maes, P. The agent network architecture (ANA). In: SIGART Bulletin, 2/4, 1991
10. McNamee B., Dobbyn S., Cunningham P., O´Sullivan C., Men Behaving Appropriately: Integrating the Role Passing Technique into the ALOHA system. In: Proceedings of the Animating Expressive Characters for Social Interactions.(2002
11. Peters C., Dobbyn S.,. Mac Namee B., O'Sullivan C.:, Smart Objects for Attentive Agents. In: WSCG 2003 short papers proceedings, Czech Republic, 2003
12. Prendinger H., Ishizuka M., Introducing the Cast for Social Computing: Life-like characters. In: Prediger, H., Ishizuka, M. (eds.): Life-like Characters. Tools, Affective Functions and Applications, Cognitive Technologies Series, Springer, Berlin, 2004, 3-16
13. O'Sullivan C., Cassell J., Vilhjálmsson H., Dingliana J., Dobbyn S., McNamee B., Peters C., Giang T., Level of Detail for Crowds and Groups. In: Computer Graphics Forum, 21(4) (2002) 733-742
14. Wooldridge, M.: An Introduction to MultiAgent Systems. John Wiley & Sons, 2002

Paralelní řešení binární faktorové analýzy

Aleš Keprt

Katedra informatiky, PřF, Univerzita Palackého, Olomouc
Katedra informatiky, FEI, VŠB–Technická Univerzita, Ostrava
Aley@Keprt.cz

Abstrakt Faktorová analýza, neboli faktorizace, je výpočetně velmi náročná. K jejímu rychlejšími řešení by mohl přispět paralelní výpočet s využitím více procesorů. Tento příspěvek představuje takové paralelní řešení. První část textu popisuje paralelní faktorizační algoritmus, druhá část pak představuje provedené experimenty a jejich vyhodnocení.

Keywords: binární/booleovská faktorová analýza, paralelní výpočet

1 Binární faktorová analýza

1.1 Definice problému

Binární (booleovská) faktorová analýza slouží k faktorové analýze dichotomických (binárních) dat. Tato analýza se liší od klasické faktorové analýzy (viz [17]) binárních dat, a to i přesto, že cíl a model jsou (symbolicky) stejné.

Cílem je vyjádřit p proměnných $\mathsf{X} = (x_1, x_2, \ldots, x_p)$ pomocí m faktorů ($\mathsf{F} = f_1, f_2, \ldots, f_m$), kde $m << p$ (m je mnohem menší než p). Tento model může být zapsán jako rozklad matice na součin dvou mnohem menších matic.

Pro n pozorování (případů, cases) mohou být matice dat X, faktorových skóre "factor scores" F a faktorových zátěží "factor loadings" A vyjádřeny jako $\mathsf{X} = \mathsf{F} \odot \mathsf{A}$, kde prvky všech matic mají hodnoty 0 nebo 1.

1.2 Rozdíl mezi klasickou a binární faktorovu analýzou

Booleovská faktorová analýza se od té klasické liší tím, že při násobení matic se používá booleovská aritmetika, takže "scores" i "loadings" jsou binární. Aritmeticky dochází k rozdílu jen při sčítání $1 + 1 = 1$, algoritmicky jde však o dvě velmi odlišné úlohy (právě z důvodu jiné použité aritmetiky klasické metody pro BFA "nefungují").

V klasické faktorové analýze je hodnota (score) každého případu (case) pro daný faktor lineární kombinací všech proměnných: všechny proměnné s velkými loadings přispívají k celkové hodnotě (score). V booleovské faktorové analýze má případ jedničkové score, když má odezvu pro kteroukoliv proměnnou dominantní ve faktoru (dominantní = nemá nulový loading), jinak má score nulové. V klasické faktorové analýze je také žádoucí mít každou proměnnou asociovanou s jedním faktorem (proměnná by neměla mít velké loadings u více faktorů najednou). Naopak v booleovské faktorové analýze proměnná jedničkový loading u více faktorů najednou mít může.

1.3 Míra kvality řešení

Pro každou matici X rozklad rozklad na $\mathsf{F} \odot \mathsf{A}$ samozřejmě neexistuje. Kvalitu konkrétního řešení posuzujeme porovnáním vstupní matice (X) se součinem ($\hat{\mathsf{X}} = \mathsf{F} \odot \mathsf{A}$).

$$d = \sum_{i=1}^{n} \sum_{j=1}^{p} |\hat{x}_{i,j} - x_{i,j}|$$

Funkce discrepancy d je tedy rovna počtu rozdílných bitů mezi vstupní matici a součinem $\mathsf{F} \odot \mathsf{A}$ (tedy vždy nezáporná hodnota). Cílem BFA je najít takový rozklad, jehož hodnota d je co nejmenší.

2 Výběr algoritmu k paralelizaci

Je známo několik algoritmů, které lze k výpočtu BFA použít. Není jich však mnoho, proto lze na úvod provést diskuzi nad vhodností každého algoritmu k paralelizování.

Prvním typem algoritmů jsou ty, které jsou paralelizovatelné jen velmi obtížně, nebo vůbec. Patří sem všechny klasické statistické procedury, kterými lze výpočet BFA simulovat (viz [16]), neboť tyto provádíme použitím standardních statistických softwarových balíků, jejichž činnost nemůžeme ovlivnit. Obtížně paralelizovatelný je také výpočet neuronovou sítí (viz [5]).

Druhým typem algoritmů jsou ty, které lze paralelizovat částečně. Patří sem zejména algoritmus FCBFA (viz [12]), který sestává ze dvou samostatných částí. V první části hledáme formální koncepty, což nelze dost dobře paralelizovat. Ve druhé části pak mezi koncepty hledáme faktory a tento proces lze paralelizovat velmi dobře. Dále bychom sem mohli zařadit algoritmus GABFA (viz [13]), který sice lze dobře paralelizovat, ale paralelní verze nedává zcela stejné výsledky jako verze klasická neparalelní. (Pokud jsou to řešení dobrá/kvalitní, pak tento rozdíl samozřejmě nemusí vadit.)

Třetím typem algoritmů jsou ty, které lze paralelizovat úspěšně a efektivně. Základním algoritmem tohoto typu je Blind Search (viz [11]), který je navíc algoritmem nejpomalejším, takže jeho paralelizace je zároveň nejvíc žádoucí. Stejně tak druhá část algoritmu FCBFA může být úspěšně paralelizována (je navíc podobná algoritmu Blind Search a lze je tedy sloučit dohromady). Budeme-li akceptovat jisté odchylky mezi klasickou a paralelní verzí algoritmu GABFA, můžeme i jej považovat za dobře paralelizovatelný.

Hlavním cílem této práce bylo vytvořit paralelní verzi algoritmu Blind Search (BS), což by zároveň umožnilo paralelně provádět druhou fázi algoritmu FCBFA. Výsledky této práce zároveň mohou posloužit jako vodítko k paralelizaci dalších algoritmů pro řešení BFA, stejně jako jiných typů datové analýzy obecně.

3 Paralelní programování v oblasti výzkumu

V oblasti výzkumu a vědeckotechnických výpočtů jsou pro paralelní programování nejrozšířenější knihovny pro posílání zpráv PVM a MPI.

3.1 Parallel Virtual Machine (PVM)

PVM (viz [6]) je knihovna nabízející sadu funkcí pro jazyky C, C++ a Fortran. Nedílnou součástí PVM je také prostředí, které vytváří tzv. virtuální stroj (virtual machine), na kterém program běží. PVM umožňuje spouštět programy na vzdálených počítačích (které jsou součástí virtuálního stroje) příkazem `spawn` podobně, jak to umí standardní knihovna jazyka C na lokálním počítači (což je velmi pohodlné). Spuštěné procesy spolu mohou komunikovat pomocí sady funkcí na posílání a přijímání zpráv, bez ohledu na to, na kterém konkrétním počítači právě běží.

Výhodou PVM je jeho snadné použití a je k dispozici pro Linux, Windows i další systémy, je zcela zdarma a nemusí se ani moc složitě instalovat.[1] Naopak nevýhodou PVM je výkon, který v aplikacích, které posílají zprávy často a/nebo větší množství dat, není optimální.[2]

Shrnutí hlavních rysů PVM:
– Umožňuje explicitní i implicitní alokaci uzlu pro každý proces
– Pouze asynchronní blokující odesílání zpráv
– Blokující i neblokující příjem zpráv
– Hromadná (skupinová) komunikace (operace barrier, broadcast a reduction)
– Počet ani délka zpráv není omezena
– Je garantováno doručení zpráv, včetně jejich pořadí

3.2 Message Passing Interface (MPI)

MPI (viz [15]) je oproti PVM novější a efektivnější framework pro paralelní výpočty. MPI definuje jen rozhraní (v podobě sady funkcí) pro komunikaci pomocí zpráv s cílem přenositelnosti programů na úrovni zdrojového kódu. Naopak některé další důležité body zde nejsou řešeny (např. spuštění procesů) a závisejí na konkrétní implementaci MPI. Proto také může být použití MPI ve srovnání s PVM poněkud obtížnější. Naproti tomu MPI vykazuje vyšší propustnost (menší latency i vyšší přenosovou rychlost než PVM) při posílání zpráv. Tohoto zrychlení je dosaženo zejména podporou speciálního síťového protokolu, zatímco PVM pracuje pouze na bázi protokolu IP.

Výhody MPI jsou však významné jen ve specifických podmínkách, protože použití zvláštního síťového protokolu má svá úskalí a programy využívající MPI jsou částečně vázány na konkrétní implementaci. Přitom samozřejmě i samotná vyšší efektivita programů běžících na MPI, která je velmi často uváděna v mnohé literatuře, je přímo závislá na kvalitě konkrétní implementace MPI. Přitom těchto implementací, jak komerčních, tak nekomerčních, je samozřejmě vícero

[1] Otázka jednoduchosti nebo složitosti instalace závisí také na použitém operačním systému. Například použití PVM ve Windows úplně snadné není.

[2] Otázka optimálního výkonu posílání zpráv samozřejmě závisí na implementaci a vlastnosti PVM se mohou v budoucnu změnit, tj. zlepšit v porovnání s alternativami. Vývoj v posledních letech ovšem směřuje spíše jiným směrem než k PVM.

a nelze očekávat, že všechny budou stejné. Stavět na MPI tedy může být v konečném důsledku příliš "drahé" (práce vynaložená navíc oproti PVM neodpovídá získaným výhodám).

3.3 Nevýhody zapříčiněné použitím PVM

Zhodnocením popsaných vlastností PVM a MPI jsme se rozhodli pro paralelní faktorizaci použít PVM. Problémem by snad mohl být již zmíněný fakt, že PVM obvykle bývá při posílání zpráv pomalejší než MPI. Tento nedostatek PVM se dá potlačit posíláním zpráv pomocí funkce `pvm_psend` (packed send), která odesílá data bez dalšího zpracování (bez konverze datových typů, opakovaného kopírování apod.). Podrobnější rozbor situace je možno najít v článku [2].

4 Paralelní verze algoritmu

4.1 Cíle

Hlavním cílem, kterému se vše ostatní podřizuje, je **dosažení kratších výpočetních časů** na běžných počítačích. Získat jeden velmi výkonný počítač je obvykle mnohem dražší, než získat větší množství počítačů běžných. Bylo by samozřejmě chybou očekávat, že nasazení většího množství počítačů automaticky povede ke zkrácení výpočetních časů.[3] Proto je potřeba použitý algoritmus pozměnit, aby použití více počítačů (přesněji: výpočetních uzlů) bylo vůbec možné.

BS je dobře paralelizovatelný algoritmus, lze tedy očekávat, že existuje takový způsob paralelizace, kdy výkon paralelního virtuálního stroje poroste zhruba lineárně s rostoucím počtem výpočetních uzlů.

4.2 Princip "pool of tasks"

Jelikož skupina běžných počítačů má obvykle větší výpočetní výkon než tzv. superpočítač (viz např. [4]), jednou z žádaných vlastností paralelního výpočtu je také schopnost běžet na virtuálním stroji, kde se rychlosti jednotlivých uzlů mohou až výrazně lišit.[4] Je proto vhodné postavit paralelní výpočet na principu "pool of tasks", kdy jeden řídící proces (dle názvosloví PVM nazývaný "master task") zadává úkoly ostatním (pracovním) procesům (v PVM "worker task"). Po vyřešení úkolu dostává každý pracovní proces další úkol.

Pro zvýšení efektivity jsou všechny pracovní procesy nastartovány příkazem `pvm_spawn` jednou na začátku celého výpočtu. Tím odpadá čas, který by bylo nutné vynaložit pro spouštění procesů před každým dílčím úkolem. Samozřejmě lze předpokládat, že operační systém umí optimalizovat opakované spouštění

[3] Viz podobně chybnou úvahu: Jeden taxík mě zaveze do školy za 1 hodinu. Proto dva taxíky mě zavezou do školy za půl hodiny.

[4] Dnes je běžné, že jedna pracovní stanice má desetinásobný výpočetní výkon než stanice jiná.

a ukončování de facto stejného procesu tak, že prakticky nestojí žádný čas, avšak inicializace procesu, tj. načtení a připravení všech dat nutných k vlastnímu výpočtu, vždy nějaký čas zabere. A právě tento čas ušetříme tím, že každý proces startujeme jen jednou.

4.3 Zadávání úkolů a střádání mezivýsledků

Algoritmus přidělování úloh řídícím procesem funguje přibližně takto:

1. Spusť všechny pracovní procesy.
2. Každému pracovnímu procesu zadej první dílčí úkol.
3. Dokud nejsou vypočítány všechny dílčí úkoly, opakuj toto:
   - Čekej na odpověď od libovolného pracovního procesu. (Odpověď obsahuje základní informaci o řešení dílčího úkolu, čili minimální zjištěnou odchylku).
   - Je-li právě přijatá hodnota odchylky zatím nejlepší, zapamatuj si její hodnotu a řešitele.
   - Jsou-li ještě nějaké nezadané dílčí úkoly, zadej pracovnímu procesu další k řešení. Připoj také informaci o zatím nejlepším řešení.
4. Pracovnímu procesu, který našel nejlepší řešení, pošli žádost o jeho zaslání.
5. Přijmi řešení.
6. Všem pracovním procesům pošli pokyn k ukončení.

4.4 Stanovení dílčích úkolů

Dílčími úkoly jsou samozřejmě algebraicky řečeno podprostory prostoru, ve kterém hledáme minimum chybové funkce. V našem případě je dělení prostoru prováděno podle podle matice A, dílčí úkoly jsou definovány hodnotami vektoru a_1, tj. prvního řádku matice A. Zadání dílčího úkolu pracovnímu procesu pak spočívá v odeslání pouhých 2 hodnot: Odešleme binární zakódování vektoru a_1 a odchylku doposud nejlepšího řešení.

Po dokončení výpočtu dílčího úkolu odesílá pracovní proces odpověď v podobě pouze jedné hodnoty – odchylky nejlepšího doposud nalezeného řešení. Tím současně dává řídícímu procesu najevo, že dokončil výpočet a očekává zadání dalšího úkolu. Místo zadání dalšího dílčího úkolu může také obdržet žádost o poslání celého řešení nebo ukončení procesu.

4.5 Zatížení sítě

Paralelní algoritmus popsaný v předchozích odstavcích způsobuje jen minimální zatížení komunikační sítě. PVM přímo nepodporuje vícevláknové programy a odesílání zpráv je částečně blokující operace, takže malý počet i objem zpráv při výpočtu je velmi příznivý pro celkovou rychlost. Jediná delší zpráva je závěrečné poslání celého řešení, to však probíhá pouze jednou na konci výpočtu.

4.6 Úprava pro použití knihovny MPI

V komunitě odborníků zabývajících se PVM a MPI lze sledovat určité snahy o pozvolný "přechod" od PVM k novějšímu a výkonnějšímu MPI. To je samozřejmě pochopitelné. Převodem programů z PVM do MPI se zabývají např. práce [10] a [8]. Pečlivé srovnání PVM a MPI můžeme najít např. v [7]. MPI má všeobecně lépe vyřešeno posílání zpráv, zatímco PVM řeší problematicku komplexněji, včetně virtuálního stroje. Naopak projekt PVMPI (viz [3]) se snaží obě rozhraní sloučit a vytvořit tak rozhraní nové, nabízející výhody obou z nich.

Přepsání faktorizátoru do MPI by bylo poměrně snadné. Dle zmíněné práce [8] je stávající verze dokonce více podobná stylu MPI než obvyklé aplikaci PVM.

5 Praktické testy

Paralelní faktorizaci jsme zkoušeli na několika testovacích úlohách. Přitom není klíčový význam jednotlivých úloh, ale pouze naměřené časy.

5.1 Tabulka časů

Tabulka 1 shrnuje výsledky všech provedených testů, uvedeny jsou výpočetní časy sériové i paralelní verze programu na různých počítačích. Není-li uvedeno jinak (viz poznámky pod tabulkou), každý test byl prováděn opakovaně, aby nejméně třikrát trval výpočet přibližně stejně dlouho (odchylka max. $\pm 1\%$). Údaj o počtu procesů značí vždy počet pracovních procesů, paralelní výpočet vždy řídí jeden řídící proces, který není do součtu započítán.

Tabulka 2 vysvětluje vybavení jednotlivých počítačů. Uvedeny jsou jen typ operačního systému, typ a taktovací frekvence hlavního procesoru (CPU) a použitý překladač jazyků C/C++. Počítače byly vždy propojeny běžnou 100Mbit lokální sítí. Ostatní parametry počítačů pro nás nejsou až tak důležité.

5.2 Srovnání sériové a paralelní verze programu

Kde to mělo smysl, provedli jsme srovnání sériové verze s paralelním výpočtem v jednom procesu. V tabulce je vidět, že časy jsou velmi podobné, rozdíly jsou příliš malé na to, abychom z nich vyvozovali nějaké závěry.

5.3 Horní část tabulky

Horní část tabulky srovnává výpočty na Phoenixu. Jedná se o klasické PC se dvěma procesory Intel Xeon. I když se jedná o multiprocesor se sdílenou pamětí, výpočet probíhal pomocí posílání zpráv. První dva řádky uvádějí časy jednoprocesového výpočtu, třetí řádek (Phoenix 2CPU) ukazuje výpočet s využitím obou procesorů. Hodnoty relativního i absolutního zrychlení jsou u všech úloh podobné. Absolutní efektivita je ve všech případech zhruba 0.92. Přitom je

konfigurace	počet procesů	úloha 1 p3 4 3	úloha 2 p3 5 3	úloha 3 p2 3 3
Phoenix	serial	0:11	4:49	30:16 (*2)
Phoenix	1	0:11	4:01	31:25
Phoenix 2CPU	2	0:06	2:11	16:12 (*1)
relativní zrychlení S_r	2	1.83	1.84	1.94
absolutní zrychlení S	2	1.83	1.84	1.87
relativní efektivita E_r	2	0.92	0.92	0.97
absolutní efektivita E	2	0.92	0.92	0.93
1× U404	serial	0:46	20:56	126:46
1× U404	1	0:48	17:17	126:29
10× U404	10	0:05	2:48	12:46
Phoenix + 10× U404	11	0:04	1:45	9:10
absolutní zrychlení S	10	9.20	6.17	9.91
absolutní efektivita E	10	0.92	0.62	0.99
1× U402	serial	0:17	6:53	60:54
1× U402	1	0:18	5:58	60:59
6× U402	6	0:04	1:20	11:19 (*1)
11× U402	11	0:02	0:54	6:01 (*2,3)
11× U402	22	0:03	1:04	5:52 (*2,3)
absolutní zrychlení S	6	4.25	4.48	5.38
absolutní zrychlení S	11	8.50	6.63	10.12
absolutní efektivita E	6	0.71	0.75	0.90
absolutní efektivita E	11	0.77	0.60	0.92

(*1) pouze 1 test
(*2) pouze 2 testy
(*3) testy vykazovaly velký rozptyl (více než 1% celkového času)

Tab. 1. Naměřené časy.

počítač	procesor	operační systém	překladač C/C++
Phoenix	2× Pentium 4 Xeon 2.8 GHz	Debian Linux 3.0	GCC
U404	Pentium 3 450 MHz	Debian Linux 3.0	GCC
U402	Pentium 4 2.0 GHz	Windows 2000 A. S.	VS++.NET

Tab. 2. Konfigurace testovacích počítačů.

zvláště zajímavé, že hodnoty jsou ve všech případech stejné. Během testů se jasně ukázalo, že systém Linux vykazuje velmi stabilní výkon, měřené výpočetní časy prakticky vůbec nekolísají. I když nebylo možno vyhradit Phoenix pouze pro prováděné testy, odchylky měření při jednotlivých testech byly do 0.1%.

5.4 Prostřední část tabulky

Zde jsou uvedeny výpočetní časy v heterogenním prostředí. Uzly U404 jsou PC propojená 100Mbit ethernetem a každý z nich je mnohem pomalejší než proce-

sory Phoenixu, dohromady jsou však rychlejší, na některých úlohách dokonce rychlejší než oba procesory Phoenixu dohromady. Při nasazení jednoho procesoru Phoenixu + všech 10 uzlů U404 bylo dosaženo nejlepších časů.

Poměry časů U404 versus Phoenix 2CPU se liší podle typu úlohy, což je důsledkem povahy úloh a zásadního faktu, že zvyšující se počet procesorů ve virtuálním stroji může negativně ovlivnit výpočet tím, že na jeho konci se může dlouho čekat na jeden pomalý uzel.

Konfiguraci PVM je možno velmi snadno nastavit tak, aby virtuální stroj využíval oba procesory Phoenixu. Toto jsme i zkoušeli a naměřený výkon odpovídal předpokladům. V tabulce nejsou hodnoty uvedeny, protože z technických důvodů (vytížení jinými výpočty) nebylo možno řádně provést všechny testy.

5.5 Spodní část tabulky

Spodní část tabulky uvádí časy testů na uzlech U402 se systémem Windows 2000 Advanced Server. Jedná se o počítače s výkonnými procesory Pentium 4, avšak není na nich Linux. Testy ve Windows mj. vykazovaly mnohem větší rozptyl hodnot, opakované testy se lišily až o 20 sekund.

Při použití všech 11 uzlů U402 bylo dosaženo vůbec nejlepších časů z celého testu. Testy byly provedeny na 1, 6 a 11 uzlech, proto je možno uvést zajímavá srovnání. Tabulka ukazuje absolutní hodnoty zrychlení a efektivity. Je na první pohled vidět, že paralelní výkon Windows se blíží Linuxu jen u velké úlohy 3. Menší úlohy (časově méně náročné) vykazují o $20-30\%$ nižší absolutní efektivitu výpočtu, což je znatelně horší výsledek. Přesto nelze jednoznačně konstatovat, že Windows je horší než Linux. Použité instalace Windows byly velmi "bohaté" a výpočty mohly být ovlivněny na pozadí běžícími procesy.

Poslední test byl za účelem zjištění, zda by ve Windows nebylo výhodné alokovat dva procesy na procesor (celkem 22 procesů). Tato teze se obecně nepotvrdila, i když třetí úloha byla spočítána o 8 sekund rychleji.

5.6 Shrnutí

Testy byly příliš rozsáhlé na to, aby je bylo možno stručně shrnout. Úloha 2 vykazovala ve všech použitých konfiguracích virtuálního stroje výsledky, které byly překvapivé. Naopak úlohy 1 a 3 dopadly dle očekávání a navzájem velmi podobně. Vzhledem k tomu, že se jedná o úlohy s velmi rozdílnou časovou náročností, lze konstatovat, že testy proběhly velmi dobře, PVM se osvědčilo a nevykázalo během několika stovek hodin testů ani jednu vadu nebo poruchu(!). Problémy byly pouze se spouštěním virtuálního stroje, pro něž PVM standardně využívá službu RSH, kterou Linux ani Windows z bezpečnostních důvodů standardně neposkytují. Rovněž vytvořený faktorizační program fungoval spolehlivě. Program samotný, ani počítače, na kterých běžel, nevykázaly během testů vůbec žádnou chybu ani poruchu.

Za zajímavý výsledek můžeme považovat lepší výsledky Linuxu oproti Windows, i když oba systémy v testu neměly plně rovnocenné postavení.

6 Další varianty paralelního výpočtu

6.1 Multithreading

PVM ani MPI přímo nepodporují multithreading (vícevláknové programování). Přitom právě kombinace různých postupů a principů vedou k nejoptimálnějšímu řešení. Proto zde nastiňuji BS algoritmus upravený pro použití kombinace posílání zpráv a více vláken v každém procesu. (Nebylo implementováno.)

Řídící proces – 1.vlákno: První vlákno řídícího procesu má za úkol přijímat zprávy a ukládat je do bufferu, ze kterého je postupně čte a zpracovává druhé vlákno.

Řídící proces – 2.vlákno: Druhé vlákno řídícího procesu čte zprávy z bufferu a zpracovává je způsobem popsaným v kap.4.3, s tím rozdílem, že každý pracovní proces dostává vždy jeden dílčí úkol dopředu. To je proto, aby v okamžiku dopočítání jednoho dílčího úkolu mohl okamžitě začít počítat další. Nečeká tedy na poslání výsledku a příjem dalšího úkolu.

Druhý zásadní rozdíl je v chování v okamžiku, kdy řídící proces v obdrží výsledek, jehož odchylka je menší než doposud nejmenší odchylka. Tato nová nejmenší hodnota je ihned poslána všem pracovním procesům.

Pracovní proces – 1.vlákno: První vlákno pracovního procesu se stará o přijímání zpráv. Zadané úkoly dává do bufferu, ze kterého je čte druhé vlákno.

Příchozí zpráva může také obsahovat pouze informaci o novém doposud nejlepším řešení. V tom případě toto vlákno přímo přepíše hodnotu odchylky v proměnné, kterou používá druhé vlákno při výpočtech. Prohlédnutím stávajícího programu jsem dospěl k závěru, že toto přepsání proměnné je skutečně možné provést bez neočekávaných následků. (Vlákna není nutno synchronizovat.)

Pracovní proces – 2.vlákno: Druhé vlákno pracovního procesu provádí vlastní výpočet. Zadané úkoly má připravené v bufferu od prvního vlákna. Výsledky předává třetímu vláknu. Algoritmus zaručuje[5], že v okamžiku dopočítání jednoho dílčího úkolu je již připravené zadání dalšího.

Pracovní proces – 3.vlákno: Zajišťuje odeslání odpovědi řídícímu procesu. Při použití MPI je možné toto vlákno zcela vynechat a nahradit ho použitím příkazu pro neblokující odeslání zprávy. Výsledný efekt by tak byl velmi podobný, avšak troufám si tvrdit, že použitím tohoto vlákna vždy dosáhneme stejných nebo lepších výsledků než při použití funkcí MPI.

6.2 Zadávání úkolů předem

Myšlenku zadávání úkolů předem, popsanou v předchozích odstavcích, lze realizovat i na stávající verzi programu. Problémem však je, že při zadávání dílčích úkolů předem vzniká problém s oznamováním zatím nejmenší odchylky. Bez

[5] až na technické meze

znalosti této hodnoty se výpočet zpomaluje, protože každý proces hledá nové lepší řešení jen vzhledem k jeho vlastnímu doposud nejlepšímu řešení, nikoli ke globálně zatím nejlepšímu řešení.

Zadávání úkolů předem tedy znamená na jedné straně zrychlení a na straně druhé zpomalení. Velmi výhodné je tam, kde síťová komunikace je velmi pomalá (např. modem, satelit), naopak bezcenná je na moderních clusterech, kde je posílání zpráv velmi rychlé.

Reference

1. BMDP (Bio-Medical Data Processing), balík statistických programů. SPSS, `http://www.spss.com/`
2. Casanova H., Dongarra J., Jiang W., The Performance of PVM on MPP Systems. University of Tennessee, Knoxville (Ten., USA), 1995
3. Fagg G.E., Dongarra J.J., PVMPI: An Integration of the PVM and MPI Systems. Calculateurs Paralleles, 2, 1996
4. Foster I., Designing and Building Parallel Programs. Addison-Wesley Inc., Reading (USA), 1995, ISBN 0-201-57594-9
5. Frolov A.A., Sirota A.M., Húsek D., Muraviev I.P., Polyakov P.A., Binary Factorization in Hopfield-Like Neural Networks: Single-Step Approximation and Computer Simulations 2003
6. Geist A., et al., PVM: Parallel Virtual Machine, MIT Press, Cambridge (Mes., USA), 1994
7. Geist G.A., Kohl J.A., Papadopoulos P.M., PVM and MPI: A Comparison of Features, 1996
8. Heistand S., Converting PVM to MPI. NAS NASA, 1999
9. Jakl O., Grygárek P., Výkonové charakteristiky počítačů IBM RISC System/6000 při vědeckotechnických výpočtech. Ústav geoniky AVČR a VŠB-TU, Ostrava, 1999
10. Jakl O., Starý J., Paralelní výpočty v prostředí MPI. Ústav geoniky AVČR a VŠB TU, Ostrava, 1998
11. Keprt A., Using Blind Search and Formal Concepts for Binary Factor Analysis. In proceedings of Dateso 2004. VŠB Technická Univerzita Ostrava, Czech Republic; CEUR WS – Deutsche Bibliothek, Aachen, Germany; 2004, 120–131, ISBN 80-248-0457-3 (VŠB TUO), ISSN 1613-0073 (CEUR).
12. Keprt A., Snášel V., Binary Factor Analysis with Help of Formal Concepts. In proceedings of CLA / Concept Lattices and Their Applications. VŠB Technická Univerzita Ostrava, Czech Republic, 2004, 90–101, ISBN 80-248-0597-9
13. Keprt A., Snášel V., Binary Factor Analysis with Genetic Algorithms. In proceedings of 4th IEEE International Workshop WSTST 2005. Springer Verlag, Berlin Heidelberg, Germany, 2005, 1259–1268, ISBN 3-540-25055-7. In series Advances in Soft Computing, ISSN 1615-3871.
14. Keprt A., Snášel V., Pseudo-dělení binárních matic a jeho aplikace. In proceedings of Znalosti 2005. Vysoké Tatry, Slovakia. Published by VŠB Technická Univerzita, Ostrava, 2005, 41–50, ISBN 80-248-0755-6
15. MPI: A Message-Passing Interface Standard, Message Passing Interface Forum, University of Tennessee, Knoxville (Ten., USA), 1995
16. Řezanková H., Húsek D., Comparison of the SAS, SPSS and STATISTICA Systems in the Area of Clustering Variables. In Computational Statistics & Data Analysis (Statistical Software Newsletter), Vol. 41, 2002, 331–339
17. Überla K., Faktorenanalyse (2.vydání). Springer-Verlag, Berlin-Heidelberg, 1971. Slovenský překlad: Alfa, Bratislava, 1974

Building ontology for VirGIS system

Zdeňka Linková and Radim Nedbal

Ústav informatiky, Akademie věd ČR,
Pod Vodárenskou věží 2, 182 07 Praha 8
linkova@cs.cas.cz, nedbal@cs.cas.cz

Abstract. Ontologies play an important role in a knowledge representation. It involves ontology development as well as ontology re-use. Among various fields, where ontologies can be useful, is the GIS (Geographical Information System) data area. The goal of the research described in this paper is to develop a specific ontology for a given GIS domain. At first, we describe a general methodology and main tools for ontology development. Then a new ontology that covers data used in a VirGIS integration system is presented. The paper describes the VirGIS specified ontology as well as a list of spatio-temporal data ontologies that are available and possible to use for a general data features description.

1 Introduction

The development and usage of geographical information systems (GIS) is based, among other things, on knowledge representation. Ontologies, as specifications of conceptualizations, are possible tools to be employed in this context. Ontologies play an important role in information processing. They enable sharing terms used for information description and thereby they also provide a basis for data sharing, data processing, and, of course, data integration.

Our aim was to build an ontology for a given GIS data domain. It had cover at least data provided by the integration system VirGIS [1].

The paper is organized as follows: Section 2 gives a brief description of GIS; Section 3 provides basic ontology theory and ontology development. Section 4 presents our research description. It introduces VirGIS data and their modelling for ontology building purpose.

2 GIS

Geographical information systems (GIS) [2, 3] are generally used to analyze and visualize spatio-temporal information. Originally developed for the creation of thematic maps, GIS support data capture, data storage, and data analysis. The power of GIS comes from the ability to relate different information in a spatial context and to obtain details about this relationship. GIS, therefore, can reveal important new information that leads to better decision making.

Unlike a flat paper map, where what you see is what you get, a GIS can present many layers of different information. These geographic data are thought

as layers of information. Each layer represents a particular theme or feature. One theme could be made up of all the roads in an area, another theme could represent all the lakes in the same area. These themes can be laid on top of one another, creating a stack of information about the same geographic area. A GIS combines layers of information about a place to give a better understanding of that place. What layers of information are combined depends on a purpose (e.g. finding the best location for a new store, analyzing environmental damage, etc.). The way data have been stored or filed as layers of information in a GIS makes it easier to perform complex analyses.

The use of GIS can encourage cooperation and communication among organizations. Standardization eases the exchange of digital information among users of different systems. One idea to provide interoperable solutions and applications for geospatial services, data, and applications is to define "simple features" in modelling GIS data. The starting point for modelling of geographic information is a geographic feature. A feature is an abstraction of a real world phenomenon. A geographic feature is a feature associated with a location relative to the Earth. A digital representation of the real world can be thought of as a set of features. The Open Geospatial Consortium [4] Reference Model (ORM) [5] describes a framework for the ongoing work of enhancing and enabling interoperability for technologies involving spatial information and location.

3 Ontologies

Ontologies [6], [7] were developed in the framework of artificial intelligence (AI) to facilitate knowledge sharing and reuse. Ongoing research on ontologies can be found in the computer science community. The reason of ontologies popularity is that they promise a shared and common understanding of some domain that can be communicated between people and application systems. Ontologies are crucial for knowledge interoperation; sharing the same ontology is a precondition to data sharing and data integration. Ontologies are also central to the Semantic Web [7], [8], because they allow applications to agree on the terms and consequently to communicate. They are a key factor for enabling interoperability in the Semantic Web.

The term "ontology" has been used in many ways and across different communities. A popular definition of the term ontology in computer science is: an ontology is a formal, explicit specification of a conceptualization. A conceptualization refers to an abstract model of some phenomenon in the world. However, a conceptualization is never universally valid. Ontologies have been set out to overcome the problem of implicit and hidden knowledge by making the conceptualization explicit. An ontology may take a variety of forms, but it will necessarily include a vocabulary of terms and some specification of their meaning.

Ontologies may help also in GIS data sharing and processing. The need to share geographic information is evident. Today, there is a huge amount of data gathered about the Earth, computers throughout the world are connected, and the use of GIS has become widespread. The support and use of multiple ontolo-

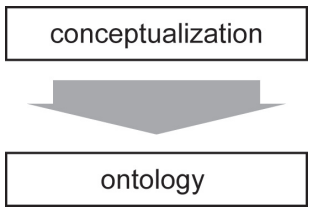


Fig. 1. Conceptualization and ontology.

gies should be a basic feature of systems that should be able to solve semantic heterogeneity to make use of the amount of information available.

3.1 Ontology development

Ontologies aim at modelling and structuring domain knowledge. The purpose is to provide understandable domain description, which may be used and shared across applications and groups of people. It requires methodologies that cover all aspects [9]. Therefore an ontology development follows a cycle [7] containing several phases.

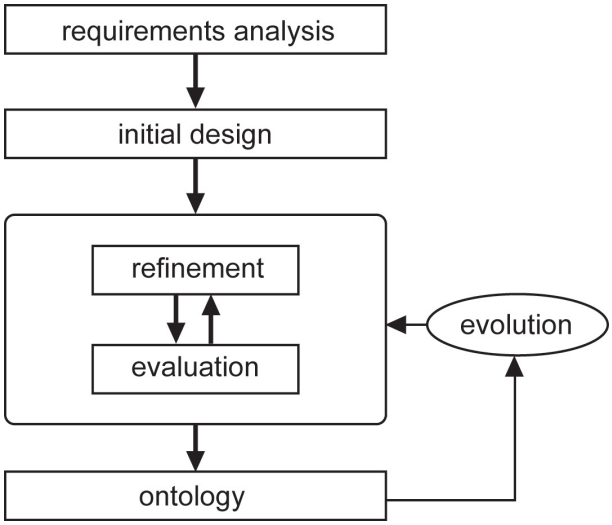


Fig. 2. Ontology lifecycle.

– The ontology lifecycle starts with a requirements analysis. In this step, it is important to consider the breadth of the ontology and richness of the modelled domain. A key issue is the balance between specificity and reusability.

- Based on the requirements specification, the initial ontology is designed. Thorough analysis of various kinds of knowledge sources (free texts, semi-structured sources, and structured sources) is important. Here ontology learning, i.e. the combination of linguistic analysis, information extraction, statistical techniques and machine learning is a promising area of research. The output of the initial design step is some preliminary conceptual structure.
- In the next step, the basic ontology design is refined and evaluated. To support efficient and effective refinement, tools for restructuring and enriching data are needed. It is possible to exploit already existing conceptual sources, like thesauri, database schemata, other ontologies etc. Created conceptual structure needs to be evaluated with respect to its requirements and eventually more specified.
- The ontology lifecycle does not end with refinement and evaluation steps. In real world, things are changing, and so should do ontologies. To handle the evolution and maintenance of ontologies, one needs to explore and to formalize the kinds of relationships that may rise between different ontology versions.

3.2 Ontology languages and tools

Contemporary ontology languages are based on the XML syntax. It is a consequence of the fact that XML (eXtensible Markup Language) [10] has become a standard language for information exchange on the Web. Also the common RDF (Resource Description Framework) [11] syntax is based on the XML. RDF was developed by the W3C (the World Wide Web Consortium) [12] as a framework to describe Web resources.

Its extension, RDF Schema [13], is RDF's vocabulary description language. RDFS provides mechanisms for describing groups of related resources and the relationships between these resources. It allows the representation of concepts, concept taxonomies and binary relations.

However it is not very expressive. For more exact description of knowledge, a richer language is needed. Therefore three more languages have been developed as alternative to RDF(S): OIL (Ontology Interchange Language) [14], DAML+OIL [15], and OWL. OWL (Web Ontology Language) [16] is a product of W3C and is presented as an ontology language for the Semantic Web. It allows representing not only concepts, taxonomies, binary relations, but also cardinalities, richer type definitions and other characteristics.

A large number of organizations have been exploring the use of OWL, with many tools currently available. The Working Group of W3C is maintaining a list of implementations and demonstrations [17]. Most of the systems currently using DAML, OIL and DAML+OIL are now migrating to OWL. In addition, a number of ontology language tools, such as the widely used Protg system [18], now provide OWL support.

It is reasonable to assume that ontologies could be available on the market. As ontology development technology evolves, the benefits of ontology use

will outweigh the costs of developing them. With the success of this approach, large-scale repositories of ontologies will be available in diverse disciplines. Also a commercial production is possible. However, the available quantity of ontological knowledge is modest and their quality, too. Some types of objects have been the objects of ontology study, some objects have received little attention.

An option is to use an ontology library containing specialized ontologies of domain and tasks. There is a large number of ontologies available on the Web. There is a DAML ontology library [19], which contains about 280 examples written in OWL or DAML+OIL (a converter from DAML+OIL to OWL is also available on the Web). The library organizes hundreds of ontologies in a variety of different ways (keyword, organization, submission date, etc.). In addition, several large ontologies have been released in OWL. And as in other research areas, there have been also some projects of ontology development in GIS data field (e.g. [20]-[23]).

4 Building ontologies for GIS

As an ontology design tool, Protg System [18] was used. Protg is an integrated software tool used by system developers and domain experts to develop ontologies and knowledge-based systems. Protg has been developed by the Stanford Medical Informatics (SMI) at Stanford University. It is an open source, standalone application with an extensible architecture. It holds a library of plugins that add more functionality to the environment. Protg's OWL Plug-in now provides support for editing Semantic Web ontologies. There is also a list of the currently made ontologies on the Protg Ontologies Library page [24]. It is a small but hopefully growing selection of existing OWL ontologies that one can use.

As a part of the research, OWL ontologies on the Internet were explored. Some of them are suited for the geographical information system. They fulfill geographic information standards (ISO, OpenGIS®Consortium - OGC, or standard by Federal Geographic Data Committee - FGDC) [22]. The list is given in Figure 3.

ISO/CD TS 19103	Ontology for Conceptual Schema Language
ISO 19107:2003	Ontology for Geographic Information - Spatial Schema
ISO 19108:2002	Ontology for Geographic Information - Temporal Schema
ISO/FDIS 19109	Ontology for Geographic Information - Rules for Application Schema
ISO/FDIS 19110	Ontology for Geographic Information - Methodology for Feature Cataloguing
ISO 19111:2003	Ontology for Geographic Information - Spatial Referencing by Coordinates
ISO 19112:2003	Ontology for Geographic Information - Spatial Referencing by Geographic Identifier
ISO 19115:2003	Ontology for Geographic Information - Metadata
ISO 19115:2003	Ontology for Geographic Information - Metadata Application
FGDC	Ontology for Content Standard for Digital Geospatial Metadata
OGC	Ontology for Topic-2: Spatial Referencing by Coordinates
OGC	Ontology for Geography Markup Language (GML3.0)

Fig. 3. List of OWL ontologies based on norms.

Our aim is to develop a new ontology for specific GIS area. More generally, we would like to help to develop a new version of the VirGIS integration system [1] that should be Semantic Web based - in order to provide more integration capabilities. For more general data description, our first step was development of an ontology covering at least data provided by this system.

4.1 VirGIS data

VirGIS is a mediation platform that provides an integrated view of geographic data. The VirGIS system is composed of data sources and a mediator over them. This mediator, which is called a GIS mediator, provides a global virtual view allowing local sources to be accessed as one integrated source.

For querying, a client uses global terms and schema. The GIS mediator rewrites this query, poses it against local data sources, then composes final answer from local answers, and returns the result to the client.

Currently, VirGIS is implemented as an integration system of satellite images. Figure 4 illustrates local and global sources of VirGIS. As local sources are used subsets of schemas drawn from SPOT and IKONOS catalogues and QUICK_LOOK database.

SPOT	
Attribute	Type
date_	Date
sun_elev	numeric
satellite	string
sat_id	numeric
key	string
the_geom.	Polygon

IKONOS	
Attribute	Type
date_acqui	Date
sun_el	numeric
satellite	string
sat_id	numeric
key	string
the_geom	Polygon

VIRGIS	
id	string
name	string
satid	string
date	Date
sun_elevation	numeric
url	string
geom	Polygon

QUICK LOOK	
Attribute	Type
key	string
filename	string

Fig. 4. Local and global satellite schemas.

SPOT and IKONOS catalogues provide information about satellites; QUICK_LOOK refers to a sample of small images that give an overview of satellite images supplied in the catalogue. The role of the global source is played by the VIRGIS mediated schema. The VIRGIS schema contains just one entity VIRGIS with following attributes:

- string *id* (a common id for the different region photographed)
- string *name* (the name of the satellite that takes the photo)
- string *satid* (the id for the satellite)
- date *date* (the date when the photo was taken)
- numeric *sun_elevation* (the sun elevation when photo was taken)
- string *url* (the url where the real photo is saved)
- polygon *geom* (the geometry of the region photographed)

According to this schema description, the aim was a development of an ontology satisfying the VirGIS data semantics. It had to cover not only the global schema, but also the local ones and relationships among them.

4.2 The VirGIS Ontology

The aim was a description of satellite image knowledge in a VirGIS ontology. In ontology re-use, we can consider only some general spatial ontology for basic geometric features. The VirGIS data area itself is not covered with any existing GIS ontology. A new ontology for this purpose is needed.

The proposed VirGIS specified ontology comes out of the data model described above. The main domain concepts and their relationships are depicted in Figure 5 by means of ISA tree.

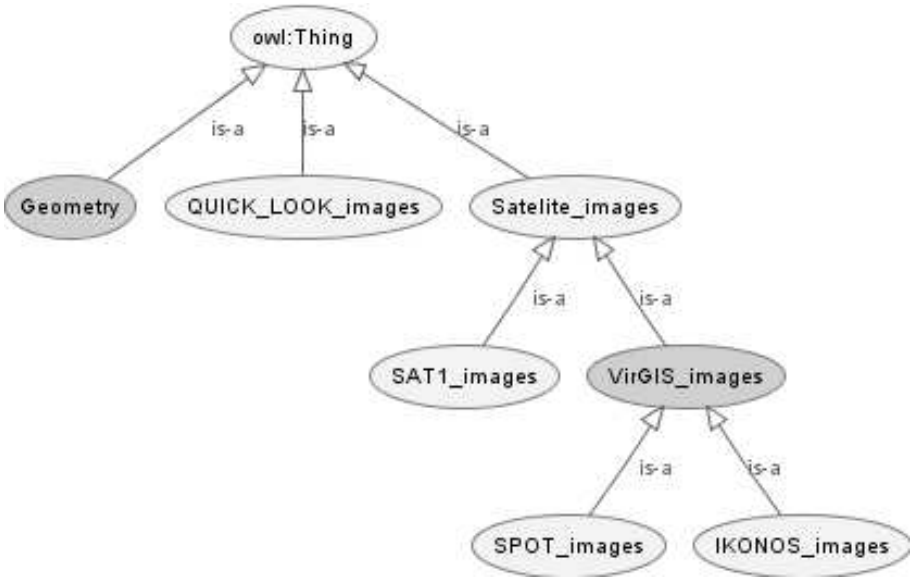


Fig. 5. ISA diagram of the model.

Observe that each node corresponds to one concept. `IKONOS_images` and `SPOT_images` refer to local sources; `VirGIS_images` refers to the global mediated source. The fact that every image contained in IKONOS or SPOT database is also contained in VirGIS induces the corresponding concepts relationship that can be understood as set inclusions:

$$\begin{aligned} IKONOS_images &\subseteq VirGIS_images, \\ SPOT_images &\subseteq VirGIS_images, \end{aligned} \tag{1}$$

Analogical relationship applies to `VirGIS_images` and `Satelite_images` concepts. Observe that there is an additional class `SAT1_images` in the model. It contains satellite images not integrated in VirGIS_images. Finally, an inherent feature of the OWL data model is the unique superclass `THING` being the superclass of all other classes.

In OWL, a `owl:Class` construct is used for concept indication and `rdfs:subClassOf` construct for expressing the concept relationships corresponding to set inclusion relations:

Example 1.
```
<owl:Class rdf:ID="SPOT_images">
   <rdfs:subClassOf rdf:resource="#VirGIS_images" />
</owl:Class>
```

The `rdfs:subClassOf` construct expresses inclusion relationship on both set and conceptual level. Therefore, the above OWL code example implies SPOT_images being conceptually more specific than VirGIS_images.

In OWL, classes are also characterized by means of properties, i.e. attributes of corresponding concepts. Properties definitions are to represent the semantic relationships of the corresponding concepts and their attributes.

Observe that SPOT and IKONOS use semantically equivalent attributes without any common name convention. In addition, VirGIS introduces its own identifiers for respective attributes. `date_` (SPOT), `date_acqui` (IKONOS) and `date` (VirGIS) represent semantically equivalent attributes for instance. This is solved with mapping of mediation integration in VirGIS. However, it can naturally be expressed on the semantic level, by means of OWL.

With regard to the above discussion and considering the inclusion (1), it follows:

$$(\forall image \in SPOT_images)(date_(image, DD/MM/YY) \rightarrow date(image, DD/MM/YY)),$$

which defines the semantic relationship of the binary predicates `date_` and `date`. The relationships between other predicates can be expressed analogically.

In OWL, `rdfs:subPropertyOf` construct is used for expressing such semantic relationships:

Example 2.
```
<owl:DatatypeProperty rdf:about="#date_">
    <rdfs:subPropertyOf rdf:resource="#date" />
</owl:DatatypeProperty>
```

This relationship is more vague than the relationship of equivalence. However, the relationship of "subPropertyOf" mirrors SPOT_images being conceptually more specific than VirGIS_images.

For completeness, there is an additional class in the model. `Geometry` class contains geometric elements, designed for geometry type properties description. In case that richer geometry is needed, geometry classes from existing spatial ontologies listed in Figure 3 can be imported. At this time, the presented ontology is suitable for VirGIS data description. It can be enriched in case more capabilities should be needed.

5 Conclusion

Ontologies are crucial in data description. According to this fact, a new ontology in GIS data area was developed. Particularly, this ontology describes sources and data in the VirGIS integration system. As other ontologies, also this one should be evolved in order to follow the evolution of things it describes. In this case, it can for instance mean ontology enrichment along with adding new sources to VirGIS. Although ontologies are very powerful tools in data processing, there is still a lack of available and suitable ontologies in many areas. We believe that our research and development can contribute to increase number of usable ontologies and can help in VirGIS data integration task.

Acknowledgements

This work was supported by the project 1ET100300419 of the Program Information Society (of the Thematic Program II of the National Research Program of the Czech Republic) "Intelligent Models, Algorithms, Methods and Tools for the Semantic Web Realization", by the project of Czech-French Cooperation Barrande 2004-003-1, 2: "Integration de donnes sur le Web - applications aux Systmes d'Information Gographique (2003-2005), and by the Institutional Research Plan AV0Z10300504 "Computer Science for the Information Society: Models, Algorithms, Applications".

References

1. O. Boucelma, F.-M. Colonna, Mediation for Online Geoservices. in Proc. 4th International Workshop Web & Wireless Geographical Information System, W2GIS 2004, Korea, November, 2004
2. Your Internet Guide to Geographic Information Systems. http://www.gis.com
3. Geographic Information Systems. http://erg.usgs.gov/isb/pubs/gis_poster/
4. Open Geospatial Consortium, Inc. (OGC). http://www.opengeospatial.org/
5. OpenGIS® Reference Model. http://portal.opengis.org/files/?artifact_id=3836
6. Y. Ding, D. Fensel, M. Klein, B. Omelayenko, "The Semantic Web: Yet Another Hip?". Data & Knowledge Engineering 41, 2002, 205–227
7. J. Euzenat, Research challenges and perspectives of the Semantic Web. Report of the EU-NSF Strategic Research Workshop, Sophia-Antipolis, France, October, 2001
8. M.-R. Koivunen, E. Miller, "W3C Semantic Web Activity". In the Proceedings of the Semantic Web Kick/off Seminar, Finland, 2001
9. O. Corcho, M. Fernndez-Lpez, A. Gmez-Prez, "Methodologies, Tools and Languages for Building Ontologies. Where is Their Meeting Point?". Data & Knowledge Engineering 46, 2003, 41–64
10. Extensible Markup Language (XML). http://www.w3.org/XML/
11. Resource Description Framework (RDF). http://www.w3.org/RDF/
12. W3C (WWW Consortium). http://www.w3.org

13. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, http://www.w3.org/TR/2004/REC-rdf-schema-20040210, February, 2004
14. OIL. http://www.ontoknowledge.org/oil
15. About the DAML Language. http://www.daml.org/about.html
16. Web Ontology Language (OWL). http://www.w3.org/2004/OWL
17. OWL Implementations. http://www.w3.org/2001/sw/WebOnt/impls
18. The Protégé Ontology Editor and Knowledge Acquisition. http://protege.stanford.edu/index.html
19. DAML Ontology Library. http://www.daml.org/ontologies/
20. U. Visser, H. Stuckenschmidt, G. Schuster, T. Vögele, Ontologies for geographic information processing. Computers & Geosciences 28, 2002, 103–117
21. F.T. Fonseca, M.J. Egenhofer, C.A. Davis Jr., K.A.V. Borges, "Ontologies and Knowledge Sharing in Urban GIS". Computers, Environment and Urban Systems 24, 2000, 251-271
22. List of OWL Ontologies based on Norms. http://loki.cae.drexel.edu/~wbs/ontology/list.htm
23. N. Lammari, E. Métais, Building and Maintaining Ontologies: a Set of Algorithms. Data & Knowledge Engineering 48, 2004, 155–176
24. Protégé Ontologies Library. http://protege.stanford.edu/ontologies/ontologies.html

Why to use a type system for processing XML?

Pavel Loupal and Michal Valenta

Department of Computer Science, FEE CTU Prague,
Karlovo nám. 13, 121 35 Praha, Czech Republic
loupalp@fel.cvut.cz, valenta@fel.cvut.cz

Abstract. In this paper we discuss an approach of manipulating XML data with help of a type system. There is an existing framework – XML-λ– that is built on a type system and uses typed λ calculus for creating a query language. We try to explore possible utilization of this framework and show an example of searching for equivalences in XPath queries that could be favorable for optimizing such queries. This paper does not offer a complete solution yet but is rather intended to open this topic for discussion.

1 Introduction

In February 2004 we started a project of implementation of a XML-native database system. However, there are similar projects with the same goal which started two or three years before. Let us mention TIMBER [1], Niagara [2], Tamino [3], and Xindice [4] as examples.

Some of above mentioned systems claim to implement (partially or fully) query languages which are likely to become broadly accepted standards - i.e. either XPath 2.0 [5] or XQuery [6]. The others, especially TIMBER, claim to state an general and consistent query-tree algebra at the first.

TIMBERs approach seems to be more systematic, although it does not follow the mainstream in the branch. The mainstream is represented mainly by XPath 2.0 and XQuery 1.0 which are developed under the W3C protection and they are both near to become W3C standards.

Commonly used data model for all these languages and systems is tree-like model. It is natural XML data representation (let us remember popular DOM representation here).

But there is also another, by our belief attractive and interesting, data model for XML data representation functional model. It was introduced in [7] in the form of a special type system for regular expression types (see section 2 for details). Together with the data model a query language, which is very close to the syntax of λ calculus, was stated. The language is called XML-λ.

We found the basic ideas of XML-λ very interesting and attractive mainly due to its simplicity and consistency with respect to large and complicated formal semantic and data model of XQuery. Hence we decided to use XML-λ principles in our XML-native database system.

Naturally, the study of XML-λ and its related type system led us to discussion of positive and negative aspects of the language with respect to XQuery (which we regard as a mainstream and hence referential XML-query language). In this paper we focus on topics concerning querying data with formalism.

The rest of the paper is organized as follows. Section 2 outlines concept of the XML-λ framework and section 3 describes its querying capability. In section 4 we suggest one of possible utilization of a type system-based framework – an approach for optimizing XPath queries.

2 The XML-λ framework

XML-λ is a framework for manipulating XML data proposed originally by Pokorný [7], [8]. Let us briefly describe its concept. Basic idea comes from the typed λ calculus [10] and hence the name XML-λ was chosen.

At the first *set of all abstract elements* E is introduced. E means an element universe – it is a space where every possible XML element can be constructed and accommodated.

Informally, *element type* can be seen as a subset of E. We can also say, that element type in XML-λ one-to-one corresponds with element type in XML document description (remember that each element of a well-typed XML document has its element type – regardless to used notation – XML Schema, DTD, Relax NG, ...).

Then – from the point of view of functional data model – *element objects* are (partial) functions over E, whose values are concrete instances of an element in a given XML document. Object elements change in time because they represent actual state of database.

```
<!ELEMENT book (author+, name, price)>
<!ELEMENT author (first, last)>
<!ELEMENT first #PCDATA>
<!ELEMENT last #PCDATA>
<!ELEMENT name #PCDATA>
<!ELEMENT price #PCDATA>
```

Fig. 1. Example DTD.

Let us consider DTD shown in Figure 1 and appropriate fragment of XML data 2.

Then information about authors "Jaroslav Pokorny" and "Karel Richta" are regarded as two object elements i.e. (partial) functions from $E \to E \times E$. At the same time, there is also a function *author* of type $E \to E \times E$. This function is partial, it means it works only for some arguments from E (some *abstract*

```
<author>
   <first> Jaroslav </first>
   <last> Pokorny </last>
</author>
<author>
   <first> Karel </first>
   <last> Richta </last>
</author>
```

Fig. 2. A Fragment of XML data.

elements). For elements containing only $PCDATA$ values there are functions of type $E \to PCDATA$.

Definition XML-λ data model follows by introduction of type system for regular expressions. This type systems (marked as T_E) is in [7] build up sequentionally using systems T and T_{reg}. (This approach is not necessary, but it is very illustrative).

First prototype implementation of the framework that utilizes it only as a query language is available in [9]. Basic overview of the language is shown in following section.

3 Querying XML

Querying is the crucial task – besides manipulating (inserting, updating and deleting) – that is essential for any kind of data. XQuery and XPath query languages (both W3C working drafts) are today the most popular options for retrieving information from XML. XQuery is tightly coupled with XPath. It uses XPath expressions for selecting appropriate nodes in XML tree and encapsulates it with desired output structure of elements. The result of the query is then a document fragment.

In comparison with these languages queries in XML-λ are slightly different but the concept remains similar – we have to search for pieces of data matching the query and then construct the output. The most important difference is the way how the data are treated. Let us have a look at the functional approach for constructing a query language based on XML-λ framework.

Pokorný [7] suggests the *language of terms*. This language uses variables and constants of type T and operations known from λ calculus: applications, projections and λ-abstractions. Then – considering our DTD – we can write queries e.g. for retrieving first names of all authors with following method:

$$\lambda a\,(/book/author(a))$$

This query returns the *author* function and for consequent evaluating we have to use projection to this intermediate result

$$\lambda x\,(/book/author(a)\ and\ a/first\ =\ x)$$

(This method is similar to axis evaluation in XPath). This query returns content of all $first$ elements stored within *author* elements in the document. Function *author* is applied to variable a and returns a tuple of abstract elements (e_1, e_2). Then a projection of $first$ to a is evaluated and stored into variable x. Content of this variable is returned as the result of evaluation.

4 Query optimization

One of possible applications of typing transformation is *query optimization*. Let us consider DTD shown in Figure 1 and XPath queries `/book/name` and `//name`. With the knowledge of the DTD we can say that these two questions will always return the same result, or in other words they are equivalent.

Most of the query engines contain also a query optimizer that is responsible for query preprocessing and thus more effective query evaluation. One of the methods used by query optimizer can be query analysis and rewriting. This approach may consider also the principle based on above mentioned example.

XPath axis evaluation is in XML-λ replaced with function evaluation. Theoretically, we are also able to derive the type of a given function (query) in XML-λ. So, if it is possible to build up a type system for XML-λ in such a way that the group of functions (queries) with the same type represents the same (canonical) query, we can use it for query optimization.

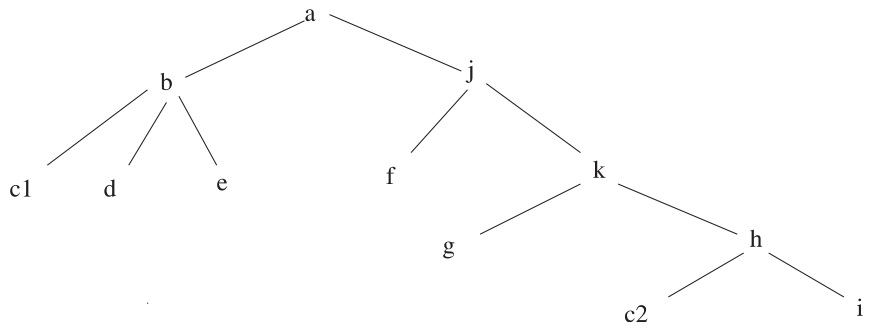


Fig. 3. XML data.

Let us now explain our idea in a more detail way: Consider a XML-data represented as a tree in Figure 3. Here the nodes represent (named) XML elements while the angles represent the child-parent relationships. At the first we have to accept an important assumption: *all elements in the document have unique names* (and types as we explain later). There are two nodes named `c` in Figure 3, but they have indexes `1` and `2` to enable distinguish between each other.

Suppose we assign a unique type to each element (we will use capitals of the element names for these reasons). So the element `c1` is of type `C1`, element `j` is of type `J` etc. The element `a` is of type `A`, but at the same time the type of `a` can be expressed as a composition of types of its subelements. We will use the notation `Z = (X,Y)` to express that the type `Z` consists of two types `X` and `Y`. We also need an reverse operation – extraction of the component. Let us write π_1 `(Z)` to address the first subtype of compound type `Z` (i.e. `X`). Here are four expressions of type construction from our example of XML data from Figure 3:

$$\texttt{C1} = \pi_1(\pi_1(\texttt{A}))$$

$$\texttt{C2} = \pi_1(\pi_2(\pi_2(\pi_2(\texttt{A}))))$$

$$\texttt{K} = \texttt{(G, (C2, I))))}$$

$$\texttt{A} = \texttt{((C1, D, E), (F, (G, (C2, I))))}$$

Now suppose we have a type system which is able to accept typing rules for composition type and it also has (typing) rules for XPath expressions `/` and `//`. Such system is then able to derive that the type of terms (expressions) `a/b/c1` and `a//c1` is `C1`. In other words we can conclude that these two terms (expressions or queries) are equivalent.

5 Sketch of the research

Let us now to sketch the above idea in the environment of a tool which implements a type system. We chose the LEGO proof assistant and ECC (Extended Calculus of Constructions) type system. At the beginning of our research we are trying to take up the work [11]. One result of [11] is a LEGO module which realizes DTD representation in ECC type system. We are going to use this already done work to approve our idea on a nontrivial case studies. Due to lack of the space we omit all introduction and explanation of LEGO proof development assistant, ECC type system, and the representation of DTD in this framework. The details of these topics can be found respectively in [13], [12], and [11].

XPath expression type derivation in LEGO. Let us suppose the following document structure:

```
B = (C, D)
A = (B, Q)
```

For representation of this structure in LEGO a *inductive type* `Record` can be used. The LEGO specification of our example looks like:

```
[C:Type] [D:Type] [Q:Type];
Record [B : Type]
Fields [c : C] [d :D];
Record [A : Type]
Fields [b : B] [q : Q];
```

LEGO accepts this definition and automatically derives the underlying declarations and definitions which enable working with (sequence) types `A` and `B`. For example for the type `B` it creates, except the type `B` itself, also the computation rule, the type eliminator, type constructor, and the terms for the component extraction (reduction rules). All these underlying definitions and declarations for the type `B` follow:

```
(* computation rule for inductive type B *)
[[C_B:B->TYPE][f_make_B:{c:C}{d:D}C_B
(make_B c d)][c:C][d:D]
    B_elim C_B f_make_B (make_B c d)  ==>
    f_make_B c d]

(* B type eliminator *)
decl  B_elim : {C_B:B->TYPE}({c:C}{d:D}C_B
(make_B c d))->{z:B}C_B z

(* B type constructor *)
decl  make_B : C->D->B

(* C component extraction *)
defn  c = B_elim ([_:B]C) ([c:C][_:D]c)
c : B ->C

(* D component extraction *)
defn  d = B_elim ([_:B]D) ([_:C][d:D]d)
d : B ->D
```

We also have to create similar rules for the type `A`. Now, we can let the system to derive the type of expression `c (b a)` (brackets are necessary for correct evaluation) which represents the XPath expression `a/b/c`:

```
Lego> [a : A]; (* local declaration *)
Lego> c (b a);
value = c  (b a)
type = C
```

At the end of this section we would like also mention another area of XML database processing which can potentially profit from employing of type systems theory and a functional point of view. The *queries over queries* are a hot topic not of XPath but of XQuery proposal.

In this case the queries are written completely in XML and thus it is possible to run query over the query formulation. Unfortunately this XML syntax of queries seems to be too complicated and hard to understand for humans, but it can potentially be processed by a machine. Here in the concept of queries over queries we feel an analogy with type systems hierarchy. For example with

construction of $\lambda 2$ type system which is done by introduction of a new (polymorphic) type constructor $\forall VT$ in $\lambda \rightarrow$ (simply typed lambda calculus). There can be found other examples in a $\lambda - cube$ classification. See [10] for details.

6 Conclusion and future work

This paper describes briefly the XML-λ framework that uses type system and typed λ calculus for manipulating XML data. We use this framework for solving problem of equivalence of XPath expressions. To evaluate our approach we use LEGO proof assistant for defining correct types and trying to define rules that can be used for formal proofing of equivalence of queries.

Another idea (not covered by this paper) is to use XML-λ for integrating different kinds of data – not only tree-structured data as XML but also relational data or any data with different data model. Functional approach offers a pure formal way of manipulating data that seems to be independent on data structures.

As already stated this paper does not describe concrete results but we try to discuss open topic discovered within our research. We plan to explore further the problem of searching for equivalence among XPath queries mainly to find out formal way to describe all XPath axis. Second research branch should focus on extending existing XML-λ implementation in way of enhancing it with aggregate and user defined functions.

References

1. J.V. Jagadish and col., TIMBER: A Native XML Database. VLDB Journal 11, Springer-Verlag, 2002, 274–291
2. Niagara system, University of Wisconsin. `http://www.cs.wisc.edu/niagara`
3. Tamino Developer Community QuiP, a W3C XQuery protoype. `http://www.softwareag.com/developer/quiip`
4. Apache Xindice - Native XML database. `http://xml.apache.org/xindice.`
5. XPath 2.0, working draft `http://www.w3.org/TR/xpath20/`
6. XQuery 1.0, working draft `http://www.w3.org/TR/xquery/`
7. J. Pokorný, XML Functionally. Proceedings of IDEAS 2000, B.C.Desai, Y. Kioki, M. Toyama (Eds.). IEEE Comp Society, 2000
8. J. Pokorný, XML-λ: An Extendible Framework for Manipulating XML Data. Proceedings of BIS 2002, Poznan
9. Šárek P., Implementation of the XML-λ Language (in Czech). Master Thesis, Dept. of Software Engineering, Charles University, Prague, 2002
10. H.P. Barendregt, Lambda Calculi with Types. Handbook of Logic in Computer Science. Oxford University Press, New York, 1992
11. M. Valenta, Using Type Systems in XML Data Transformations. PhD Thesis. Czech Technical University in Prague, 2002
12. Z. Luo, Computation and Reasoning. A Type Theory for Computer Science. ISBN 0 19 853835 9, 228 pages. Clarendon Press. Oxford 1994
13. Z. Luo, R. Pollack, LEGO Proof Development System: User's Manual. Dept. of Computer Science, University of Edinburgh, 1992, `http://www.dcs.ed.ac.uk/home/lego`

Stromová algebra v logike prvého rádu

Róbert Novotný a Stanislav Krajči

Institute of Computer Science, Faculty of Science, P. J. Šafárik University Košice
Jesenná 9, 040 01, Košice
r.novotny@szm.sk

Abstrakt. Definujeme formálny dátový model pre XML dokumenty a operátory stromovej algebry – navigáciu (výber rodičov, či detí uzla), Kleeneho uzáver (viacnásobné aplikovanie operátorov) a selekciu (filtrácia uzlov na základe booleovskej podmienky). S využitím týchto operátorov definujeme jazyk logiky prvého rádu, pomocou ktorého sa možno dopytovať na uzly v dátovom modeli.

1 Úvod

Jazyk XML (*eXtensible Markup Language*) je v súčasnosti takpovediac monopolným značkovacím jazykom. Jednou z častých aplikácií sú XML dokumenty XML slúžiace ako skladisko dát, resp. databáza. V takomto prípade je prirodzenou požiadavkou mať k dispozícii aparát na adresáciu jednotlivých prvkov XML dokumentu – príkladom je napr. adresovací jazyk XPath (pozri [4]). Popri tom vznikla potreba definovať formálny matematický model pre dopytovanie sa nad XML dokumentami (stromovú algebru), čím by vznikol vzťah podobný vzťahu "relačná algebra-SQL".

Naším cieľom je uviesť náhľad na formálny model pre stromovú algebru založený na ideách poskytnutých v článkoch [3] a [2]. Tento model by sa mal pohybovať v medziach logiky prvého rádu. Podrobnosti a technické detaily možno nájsť v práci [1].

2 Úvodné definície a označenia

Na začiatok definujeme niektoré základné pojmy a štruktúry, ktoré budeme často používať.

Definícia 1. *Systém podmnožín množiny A budeme označovať $\mathcal{P}(A)$. Systém jednoprvkových podmnožín množiny A budeme označovať $\mathcal{P}^1(A)$.*

Definícia 2. Zoznamom *dĺžky n nad množinou M nazývame usporiadanú n-ticu $\langle a_0, \ldots, a_{n-1} \rangle$, kde $(\forall i \in \{0, \ldots, n-1\})$ $a_i \in M$. Ak Z je zoznam, potom $Z[i]$ označuje i. prvok zoznamu Z.*

Definícia 3. *Majme abecedu* A = Písmená ∪ ŠpeciálneZnaky ∪ OstatnéZnaky ∪ Číslice, *kde* Písmená = {a, . . . , ž} ∪ {A, . . . , Ž} *je množina znakov slovenskej abecedy,* Číslice = {0, . . . , 9} *je množina cifier,* ŠpeciálneZnaky = {!, @, #} *je množina špeciálnych znakov a* OstatnéZnaky *je množina ASCII znakov nepatriacich do množiny* Písmená ∪ ŠpeciálneZnaky ∪ Číslice.

Písmená a slová nad abecedou A *budeme značiť* `neproporcionálnym typom písma`.

Definujme rôzne druhy slov nad abecedou A, ktoré budeme používať. Predovšetkým budeme rozlišovať reťazce a čísla.

Definícia 4.

- *Slovo nezačínajúce špeciálnym znakom sa nazýva* reťazec *(string). Množinu všetkých reťazcov označujeme* String.
- *Slovo v tvare, ktorý reprezentuje nasledovná rozšírená Backus-Naurova forma sa nazýva* číslo.

$$\textit{číslica} := \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9}$$

$$\textit{číslo} := \texttt{-?}\ \textit{číslica}\texttt{+ (.}\ \textit{číslica}\texttt{+)?}$$

  Množinu všetkých čísiel označujeme Number.
- *Slovo* `#` *sa nazýva* identifikátor koreňového uzla.

V mnohých programovacích jazykoch bývajú reťazcové a číselné literály často špeciálne odlíšené (napr. uvádzanie reťazcov v úvodzovkách). My však budeme reťazce a čísla rozlišovať výhradne na základe predošlej definície. Podľa nej je slovo `5` vždy číslom a reťazce `5` v našej abecede nebudú povolené.

3 Dátový model

Náš dátový model je inšpirovaný dátovým modelom v práci [3]. Je tvorený predovšetkým základnou konečnou množinou Node. Prvky tejto množiny budú prislúchať jednotlivým hlavným prvkom XML dokumentu, ktorými sú elementy, textové uzly a atribúty. Na tom, aké prvky má obsahovať základná množina, nezáleží; asi najjednoduchšie je za prvky základnej množiny považovať prirodzené čísla.

Vlastnosti prvkov základnej množiny (zodpovedajúce vlastnostiam uzlov v dokumente XML) a vzťahy medzi jednotlivými prvkami (najčastejším je vzťah "rodič-dieťa") sú popisované niekoľkými totálnymi zobrazeniami zo základnej množiny. Špeciálne miesto medzi vzťahmi má usporiadanie uzlov. To je potrebné v prípade, že potrebujeme vybrať napr. i-tého potomka uzla.

Definícia 5. *Majme neprázdnu konečnú alebo spočítateľnú množinu* Node. *Prvok* $u \in$ Node *sa nazýva* uzol. *Majme nasledovné totálne zobrazenia:*

- label : Node → String, *ktoré reprezentuje značku ("tag") uzla.*

- $\mathsf{value} : \mathsf{Node} \to \mathsf{String}$ *predstavujúce hodnotu uzla.*
- $\mathsf{parent} : \mathsf{Node} \to \{\emptyset\} \cup \mathcal{P}^1(\mathsf{Node})$*, ktorého funkčná hodnota je buď prázdna množina (ak uzol nemá rodiča), alebo obsahuje jednoprvkovú množinu obsahujúcu rodiča.*
- $\mathsf{isAttribute} : \mathsf{Node} \to \{0, 1\}$ *špecifikuje, či uzol je atribútovým uzlom.*
- $\mathsf{children} : \mathsf{Node} \to \{\emptyset\} \cup \mathcal{P}(\mathsf{Node})$ *udáva množinu uzlov, ktoré sú deťmi uzla.*
- $\mathsf{order} : \mathsf{Node} \to \mathbb{N}$ *určuje lineárne usporiadanie prvkov množiny* Node*.*

Uzlovou štruktúrou *nazývame usporiadanú trojicu*

$$\langle \mathsf{Node}, \langle \mathsf{label}, \mathsf{value}, \mathsf{parent}, \mathsf{isAttribute}, \mathsf{children} \rangle, \mathsf{order} \rangle,$$

pre ktorú platí:

1. *Pre každé dva uzly* $u_1, u_2 \in \mathsf{Node}$ *platí*

$$\mathsf{parent}(u_2) \ni u_1, \text{ akk } u_2 \in \mathsf{children}(u_1).$$

   To zaručuje korektnosť uzlovej štruktúry. Ak uzol r je rodičom uzla u (vzťah vyjadrený zobrazením parent), potom u sa nachádza v množine detí uzla r (vzťah vyjadrený zobrazením children).

2. *Zobrazenie* order *je bijekcia, ktorá spĺňa podmienky:*
   (a) *Ak* $u_1, u_2 \in \mathsf{Node}$ *a* $\mathsf{parent}(u_2) \ni u_1$*, tak* $\mathsf{order}(u_1) < \mathsf{order}(u_2)$.
   (b) *Ak* $u \in \mathsf{Node}$ *a* $u.\mathsf{children} = \{u_0, \ldots, u_n\}$*, tak pre* $i \neq j$*;* $i, j \in \{0, \ldots, n\}$ *je* $\mathsf{order}(u_i) \neq \mathsf{order}(u_j)$.
3. *Existuje jediný uzol* $u \in \mathsf{Node}$*, pre ktorý platí*

$$\mathsf{parent}(u) = \emptyset \wedge \mathsf{isAttribute}(u) = 0 \wedge \mathsf{label}(u) \neq \varepsilon.$$

   Tento uzol sa nazýva koreňom*.*
4. *Pre* $u \in \mathsf{Node}$ *platí:*
   (a) *Ak* $\mathsf{isAttribute}(u) = 1$ *alebo* $\mathsf{label}(u) = \varepsilon$*, tak* $|\mathsf{children}(u)| = 0$.
   (b) *Ak* $\mathsf{isAttribute}(u) = 1$*, potom* $\mathsf{label}(u) \neq \varepsilon$.
   (c) *Ak* $\mathsf{isAttribute}(u) = 0$ *a* $\mathsf{label}(u) \neq \varepsilon$*, potom* $\mathsf{value}(u) = \varepsilon$.

   Atribútové a textové uzly teda nemôžu mať deti. Zároveň atribútový uzol musí mať neprázdnu značku.

Definícia 6. *Uzol* $u \in \mathsf{Node}$ *sa nazýva*

- atribútovým uzlom*, ak* $\mathsf{isAttribute}(u) = 1$*,*
- textovým uzlom*, ak* $\mathsf{isAttribute}(u) = 0$ *a zároveň* $\mathsf{label}(u) = \varepsilon$*,*
- elementovým uzlom*, ak nie je atribútovým ani textovým uzlom.*

Tieto typy uzlov zodpovedajú špecifikácii W3C.

4 Stromová algebra

Operátory stromovej algebry majú nasledovnú myšlienku:

- Dva operátory navigácie umožňuje vyberať uzly, ktoré sú k danému uzlu vo vzťahu "rodič-dieťa". Operátor hornej navigácie vráti uzol, ktorý je rodičom daného uzla, operátor dolnej navigácie zas vráti množinu potomkov daného uzla
- Operátor Kleeneho uzáveru dovoľuje viacnásobné skladanie operátorov stromovej algebry. Ďalej poskytuje možnosť opakovať aplikáciu daného operátora, až kým sa nedosiahne pevný bod príslušného operátora.
- Operátor selekcie umožňuje filtrovať množinu daných uzlov na základe booleovskej podmienky.

4.1 Operátory navigácie

Idea dolného operátora navigácie je nasledovná: jeho prvým argumentom je uzol, ku ktorému budú hľadať "rodinného príslušníka. Druhý argument zabezpečí, že vo funkčnej hodnote operátora budú len uzly s požadovanou značkou. Alternatívne je možné vybrať všetky uzly, bez ohľadu na značku.

Definícia 7 (operátor ν). *Nech $U \subseteq \mathsf{Node}$ a nech $s \in \mathsf{String}$.* Dolným navigačným operátorom ν *nazývame funkciu*

$$\nu : \mathcal{P}(\mathsf{Node}) \times \mathsf{String} \to \mathcal{P}(\mathsf{Node})$$

definovanú nasledovne:

$$\nu(U,s) = \begin{cases} \{d \in \mathsf{Node} : |\mathsf{parent}(d)| = 1 \wedge \mathsf{parent}(d) \subseteq U\}, & ak\ s = *, \\ \{d \in \mathsf{Node} : |\mathsf{parent}(d)| = 1 \wedge \mathsf{parent}(d) \subseteq U \wedge \mathsf{label}(d) = s\}, & inak. \end{cases}$$

Je užitočné si všimnúť, že operátor dolnej navigácie vyberá uzol ľubovoľného typu, teda elementové, atribútové i textové uzly. V prípade, že druhý argument nie je $*$, sú vybrané len elementové a atribútové uzly (keďže textové uzly majú prázdnu značku).

Operátor hornej navigácie je postavený na zobrazení parent. Funkčnou hodnotou je zjednotenie množín obsahujúcich rodičov uzlov v argumente operátora.

Definícia 8 (operátor π). *Nech $U \subseteq \mathsf{Node}$.* Horným navigačným operátorom π *nazývame funkciu*

$$\pi : \mathcal{P}(\mathsf{Node}) \to \mathcal{P}(\mathsf{Node})$$

definovanú nasledovne:

$$\pi(U) = \bigcup_{u \in U} u.\mathsf{parent}$$

4.2 Operátor Kleeneho uzáver

Kleeneho uzáver umožňuje viacnásobné skladanie operátorov stromovej algebry, s postupným kumulovaním výsledkov z jednotlivých iterácií. Kleeneho operátor vypočíta hodnotu iterácie $(i+1)$ a výsledok zjednotí s hodnotami z iterácií i, $(i-1), \ldots, 1$.

V prípade, že počet požadovaných iterácií nie je známy, Kleeneho uzáver umožňuje iterovať operátor dovtedy, kým sa nedosiahne pevný bod operátora, ktorý iterujeme.

Definícia 9 (Kleeneho uzáver). *Majme dve ľubovoľné konečné alebo spočítateľné množiny $\mathfrak{U}$, $\mathfrak{V}$ a množiny $U \subseteq \mathfrak{U}$, $V \subseteq \mathfrak{V}$. Nech $f : \mathfrak{U} \times \mathfrak{V} \to \mathfrak{U}$. Potom aplikáciu operátora Kleeneho uzáveru na funkciu f (označujeme f^*) definujeme nasledovne:*

$$f^{(1)}(U,V) = f(U,V)$$
$$f^{(i+1)}(U,V) = f^{(i)}(U,V) \cup f(f^{(i)}(U,V),V)$$

$$f^* = \bigcup_{i \in \mathbb{N}} f^{(i)}$$

4.3 Operátor selekcie

Myšlienka tohto operátora je jednoduchá – ponechať na výstupe len tie uzly, ktoré spĺňajú booleovskú podmienku obsahujúcu logické spojky (konjunkciu, disjunkciu a negáciu), funkcie pracujúce s uzlami, reťazcami a číslami a booleovské funkcie. Medzi funkcie, ktoré sú povolené v tejto podmienke však spadajú aj operátory stromovej algebry a funkcie prislúchajúce zobrazeniam z definície uzlovej štruktúry.

Zároveň musíme formalizmus tejto booleovskej podmienky prispôsobiť medziam logiky prvého rádu. Preto booleovskú podmienku definujeme ako formulu jazyka prvého rádu bez kvantifikátorov, ktorý budeme označovať $\mathfrak{J}_\sigma$.

Syntax Vytvorme teraz jazyk $\mathfrak{J}_\sigma$ pre podmienku v selekcii. Definujeme teda jazyk

$$\mathfrak{J}_\sigma = \left\langle \mathcal{D}_\sigma, \mathrm{typ}_{\mathcal{C}_\sigma}, \mathrm{typ}_{\mathcal{V}_\sigma}, \mathrm{arita}_{\mathcal{F}_\sigma}, \mathrm{arita}_{\mathcal{P}_\sigma} \right\rangle.$$

– Množina dátových typov jazyka $\mathfrak{J}_\sigma$ určená nasledovne

$$\mathcal{D}_\sigma = \{\textsc{String}, \textsc{Number}, \textsc{Node}, \textsc{NodeSet}\}.$$

– Majme množinu konštantových symbolov

$$\mathcal{C}_\sigma = \mathsf{String} \cup \mathsf{Number} \cup \{\#\}.$$

Totálna funkcia $\mathrm{typ}_{\mathcal{C}_\sigma}$ je zobrazenie $\mathrm{typ}_{\mathcal{C}_\sigma} : \mathcal{C}_\sigma \to \{\textsc{String}, \textsc{Number}, \textsc{Node}\}$, definované nasledovne:

  - Ak $c \in \mathsf{String}$, potom $\text{typ}_{\mathcal{C}_\sigma}(c) = \text{STRING}$.
  - Ak $c \in \mathsf{Number}$, potom $\text{typ}_{\mathcal{C}_\sigma}(c) = \text{NUMBER}$.
  - Ak $c = \mathtt{\#}$, potom $\text{typ}_{\mathcal{C}_\sigma}(c) = \text{NODE}$.
- Nech S je množina slov, ktoré nezačínajú slovom `$0`. Množina premenných jazyka $\mathcal{V}_\sigma$ je definovaná ako

$$\mathcal{V}_\sigma = S \cup \{\mathtt{\$0}\}$$

  Budeme uvažovať *množinu špeciálnych premenných* $\mathcal{V}_{\check{\text{S}}} \subset \mathcal{V}_\sigma$ definovanú $\mathcal{V}_{\check{\text{S}}} = \{\mathtt{\$0}, \mathtt{\$1}, \mathtt{\$2}, \ldots\}$, pričom platí, že ak $v \in \mathcal{V}_{\check{\text{S}}}$, tak $\text{typ}_{\mathcal{V}_\sigma}(v) = \text{NODE}$. Na tejto množine definujme lineárne usporiadanie, na základe ktorého môžeme vytvoriť zoznam prvkov množiny $\mathcal{V}_{\check{\text{S}}}$. Potom je možný napr. nasledovný zápis: $\mathcal{V}_{\check{\text{S}}}[2005] = \mathtt{\$2005}$.
- Konečná množina funkciových symbolov $\mathcal{F}_\sigma$ je definovaná takto:

$$\mathcal{F}_\sigma = \{\mathtt{navigationDown}, \mathtt{navigationUp}, \mathtt{selection}, \ldots\}.$$

  Ak arita funkciového symbolu je zobrazenie $\text{arita}_\mathcal{F} : \mathcal{F} \to \bigcup\{\mathcal{D}^n \times \mathcal{D} : n \in \mathbb{N}^+\}$, platí:
  - $\text{arita}_\mathcal{F}(\mathtt{navigationDown}) = \langle\langle \text{NODESET}, \text{STRING}\rangle, \text{NODESET}\rangle$,
  - $\text{arita}_\mathcal{F}(\mathtt{navigationUp}) = \langle\langle \text{NODESET}\rangle, \text{NODESET}\rangle$,
  - $\text{arita}_\mathcal{F}(\mathtt{selection}) = \langle\langle \text{NODESET}, \text{STRING}\rangle, \text{NODESET}\rangle$.

  Samozrejme, množinu funkciových symbolov možno doplniť podľa potreby – napr. o symboly zodpovedajúce funkciám z definície uzlovej štruktúry, reťazcovým, či numerickým funkciám z normy XPath, prípadne o iné vhodné operácie (napr. funkciové symboly pre numerické operácie).
- Množina predikátových symbolov $\mathcal{P}_\sigma$ je podobne tvorená vhodne zvolenými predikátovými symbolmi – napr. porovnania, prípadne symboly zodpovedajúce booleovským funkciám z normy XPath. Príkladom môže byť množina

$$\mathcal{P}_\sigma = \{\mathtt{isAttribute}, =_{\mathtt{N}S}, =_{\mathtt{R}}, =_{\mathtt{S}}, =_{\mathtt{N}}\}.$$

  Ak zobrazenie $\text{arita}_\mathcal{P} : \mathcal{P} \to \bigcup\{\mathcal{D}^n : n \in \mathbb{N}^+\}$ udáva aritu predikátového symbolu, potom napr.
  - $\text{arita}_\mathcal{P}(=_{\mathtt{N}S}) = \langle \text{NODESET}, \text{NODESET}\rangle$,
  - $\text{arita}_\mathcal{P}(=_{\mathtt{R}}) = \langle \text{NUMBER}, \text{NUMBER}\rangle$,
  - $\text{arita}_\mathcal{P}(=_{\mathtt{S}}) = \langle \text{STRING}, \text{STRING}\rangle$,
  - $\text{arita}_\mathcal{P}(=_{\mathtt{N}}) = \langle \text{NODE}, \text{NODE}\rangle$.

Termy. Z definície jazyka $\mathfrak{J}_\sigma$ vyplýva, že konštanty v termoch tohto jazyka sú len typu STRING alebo NUMBER alebo NODE.

Substitúcia premenných. Zo substitúcií má pre nás význam len jediná – substitúcia, ktorá "posúva" špeciálne premenné. Tá všetky špeciálne premenné "o jednu nižšie v zmysle lineárneho usporiadania. Ak použijeme nie celkom presnú notáciu, špeciálna premenná $\$i$ sa touto substitúciu nahradí špeciálnou premennou $\$i-1$, pričom špeciálna premenná `$0` nie je touto substitúciou ovplyvnená.

Definícia 10. *Substitúciu, ktorá zobrazuje*

$$\mathcal{V}_{\check{S}}[i] \mapsto \mathcal{V}_{\check{S}}[i-1], \text{ pre každé } i > 0$$

budeme označovať posun.

Formuly. V úvode k tejto časti sme spomínali, že v našom jazyku nebudeme vo formulách využívať kvantifikátory. Aby sme predišli terminologickým a formálnym rozporom, formulu neobsahujúcu kvantifikátory budeme nazývať 0-formulou. 0-formula v jazyku $\mathfrak{J}_\sigma$ vlastne zodpovedá tomu, čo sme v úvode tejto časti vágne nazývali "booleovskou podmienkou". Definujeme si preto nasledovné pomocné označenia.

Definícia 11. Elementárnou σ-podmienkou *budeme nazývať atomickú formulu nad jazykom* $\mathfrak{J}_\sigma$. *σ-podmienkou budeme nazývať každú 0-formulu jazyka* $\mathfrak{J}_\sigma$. *Množinu všetkých σ-podmienok budeme označovať* σ-CONDITION.

Sémantika a interpretácia. Pred definíciou interpretácie jazyka $\mathfrak{J}_\sigma$ je potrebné si uvedomiť, že mnohé funkcie záležia od konkrétnych zobrazení v danej uzlovej štruktúre. Budeme predpokladať, že nasledovná interpretácia pracuje s pevne danou uzlovou štruktúrou, ktorá sa v procese interpretácie nemení.

Definujeme interpretáciu $\mathfrak{M}$ jazyka $\mathfrak{J}_\sigma$, pričom $\mathfrak{M} = \langle \mathfrak{M}_{\mathcal{D}_\sigma}, \mathfrak{M}_{\mathcal{C}_\sigma}, \mathfrak{M}_{\mathcal{F}_\sigma}, \mathfrak{M}_{\mathcal{P}_\sigma} \rangle$, kde v usporiadanej štvorici máme postupne interpretáciu dátových typov, konštantových, funkciových a predikátových symbolov.

Dátové typy. Interpretáciu (čiže domény) dátových typov definujeme nasledovne:
- $\mathfrak{M}_{\mathcal{D}_\sigma}(\text{NODE}) = \mathsf{Node}$.
- $\mathfrak{M}_{\mathcal{D}_\sigma}(\text{NODESET}) = \mathcal{P}(\mathsf{Node})$.
- $\mathfrak{M}_{\mathcal{D}_\sigma}(\text{STRING}) = \mathsf{String}$.
- $\mathfrak{M}_{\mathcal{D}_\sigma}(\text{NUMBER}) = \mathbb{R}$.

Konštantové symboly. Definujeme nasledovnú interpretáciu konštantových symbolov jazyka $\mathfrak{J}_\sigma$. Nech $c \in \mathcal{C}_{\mathfrak{J}_\sigma}$.
- Ak $c \in \mathsf{String}$, potom $\mathfrak{M}_{\mathcal{C}_\sigma}(c) = c$.
- Ak $c \in \mathsf{Number}$, potom $\mathfrak{M}_{\mathcal{C}_\sigma}(c) = \mathsf{valueOf}(c)$.
- Ak $c = \#$, potom $\mathfrak{M}_{\mathcal{C}_\sigma}(c) = u$, kde $u \in \mathsf{Node}$ je koreňom uzlovej štruktúry.

Funkciové symboly. Definujeme nasledovnú interpretáciu funkčných symbolov jazyka $\mathfrak{J}_\sigma$.
- Nech $\texttt{navigationDown} \in \mathcal{F}_\sigma$ a platí, že $\texttt{navigationDown}\langle \text{NODESET}, \text{STRING} \rangle : \text{NODESET}$. Nech $U \in \mathcal{P}(\mathsf{Node})$, $s \in \mathsf{String}$. Potom

$$\mathfrak{M}_{\mathcal{F}}(\texttt{navigationDown})(U, s) = \nu(U, s)$$

- Nech $\texttt{navigationUp} \in \mathcal{F}_\sigma$ a nech platí, že $\texttt{navigationUp}\langle \text{NODESET} \rangle : \text{NODESET}$. Nech $U \in \mathcal{P}(\mathsf{Node})$. Potom

$$\mathfrak{M}_{\mathcal{F}}(\texttt{navigationUp})(U) = \pi(U)$$

V prípade, že sme v syntaxi definovali ďalšie funkciové symboly, je nutné v tejto časti uviesť ich interpretáciu. Zatiaľ sme neuviedli interpretáciu funkciového symbolu `selection`. To však napravíme po tom, čo uvedieme definíciu operátora σ.

Predikátové symboly. Interpretácia opäť záleží na zvolení množiny predikátových symbolov v $\mathfrak{J}_\sigma$. Príklady interpretácie môžu byť nasledovné:

- Nech $\texttt{isAttribute} \in \mathcal{P}_\sigma$ a platí, že $\texttt{isAttribute}\langle\textsc{Node}\rangle$. Nech $u \in \textsc{Node}$. Potom

$$u \in \mathfrak{M}_\mathcal{P}(\texttt{isAttribute}), \text{ akk } \mathsf{isAttribute}(u) = 1.$$

- predikátové symboly $=_{NS}$, $=_\texttt{N}$, $=_\texttt{S}$, $=_\texttt{R}$ sú postupne interpretované reláciami rovnosti medzi prvkami množiny Node, $\mathcal{P}$(Node), String, Number.

Splniteľnosť. V klasickej logike prvého rádu sa definuje splniteľnosť formuly v interpretácii pri danom ohodnotení. My tento pojem budeme označovať $\mathfrak{M} \models \varphi[\mathfrak{e}]$ (v interpretácii $\mathfrak{M}$ je splniteľná formula φ pri ohodnotení $\mathfrak{e}$.

Zmena hodnoty zobrazenia. Pri využívaní funkcie ohodnotenia premenných, resp. termov budeme často potrebovať zmeniť hodnotu konkrétneho zobrazenia v danom bode. Ak máme napr. funkciu $f : \mathbb{N} \to \mathbb{N}$, kde $f(0) = 1$ a $f(1) = 2$, potom môžeme napr. zmeniť funkčnú hodnotu f v bode 0 napr. na π. Označenie tejto operácie je v nasledovnej definícii.

Definícia 12. *Majme funkciu jednej premennej $f(x)$ a majme množinu $\{\langle a_0, b_0\rangle, \langle a_1, b_1\rangle, \ldots, \langle a_n, b_n\rangle\}$, kde b_i je rovnakého dátového typu ako a_i, pre každé $i \in \{0, \ldots, n\}$. K zobrazeniu f definujeme zobrazenie so zmenenými funkčnými hodnotami v bodoch $a_1, \ldots, a_n$ nasledovne:*

$$\begin{matrix} f(x) \\ {\scriptstyle a_0 \mapsto b_0} \\ \vdots \\ {\scriptstyle a_n \mapsto b_n} \end{matrix} = \begin{cases} b_i, & \text{ak } x = a_i \\ f(x), & \text{inak.} \end{cases}$$

Operátor selekcie Operátor selekcie bude mať tri argumenty: množinu, z ktorej sa budú vyberať uzly; reťazec predstavujúci σ-podmienku a ohodnotenie premenných v σ-podmienke.

Spomenuli sme už, že σ-podmienka má medzi svojimi funkciovými symbolmi všetky doteraz definované operátory stromovej algebry, dokonca aj operátor selekcie (hoci sme jeho interpretáciu ešte nedefinovali). To umožňuje v σ-podmienke používať vnorené podmienky (napr. vyberať tie elementy so značkou `kapitola`, ktoré majú párny počet detí vlastniacich atribút s hodnotou `5`).

Predpokladajme najprv, že σ-podmienka neobsahuje vnorené podmienky (teda neobsahuje funkciový symbol zodpovedajúci operátoru selekcie). Do výslednej množiny sa dostanú len uzly, ktoré spĺňajú zadanú podmienku. σ-podmienka φ je splnená (pravdivá), ak platí v interpretácii $\mathfrak{M}$ a pri ohodnotení $\mathfrak{e}$ platí $\mathfrak{M} \models \varphi[\mathfrak{e}]$.

Ako zistíme, či podmienka je splnená pre daný uzol? Na to slúži špeciálna premenná `$0`, ktorú musí σ-podmienka obsahovať. Pri vykonávaní interpretácie sa (okrem iného) ohodnocujú premenné. Špeciálna premenná `$0` sa ohodnotí uzlom, pre ktorý sa overuje splnenie σ-podmienky. Preto σ-podmienka musí obsahovať aspoň jednu špeciálnu premennú.

Teda máme akýsi algoritmus:

1. Vezmi uzol u z množiny v prvom argumente.
2. Pri interpretácii ohodnoť špeciálnu premennú `$0` v σ-podmienke uzlom u.
3. Ak je σ-podmienka pri danom ohodnotení premenných v interpretácii splniteľná, vlož u do výslednej množiny. V opačnom prípade prejdi na krok 1.
4. Skonči, ak si prešiel všetky uzly množiny v prvom argumente.

Situácia je trochu komplikovanejšia v prípade, že σ-podmienka obsahuje vnorené podmienky. To možno dosiahnuť použitím funkciového symbolu pre operátor selekcie v σ-podmienke. Algoritmus je podobný, ako v prípade bez vnorených podmienok. Predstavme si však situáciu, že operátor $\mathfrak{I}$ začne interpretovať funkciový symbol pre selekciu napr. do operátora σ_1. Prvý argument sa interpretuje do množiny $\mathcal{P}(\mathsf{Node})$, nech je ním množina U_0. Druhý argument je typu STRING, ktorý sa interpretuje do prvku množiny String (označme ho s_0). Reťazec s_0 predstavuje vnorenú podmienku, ktorou by malo byť slovo jazyka $\mathfrak{J}_\sigma$ predstavujúce 0-formulu. Interpretovaná funkcia σ_1 je však operátor selekcie, ktorého má opäť dva parametre: množinu uzlov (označme ju U_1 a reťazec pre σ-podmienku, s_1. Tento reťazec (reprezentujúci vnorenú podmienku) by opäť mal obsahovať nejakú špeciálnu premennú, ktorou sa budú ohodnocovať uzly z množiny U_1 pri zisťovaní pravdivosti σ-podmienky s_1. Aby sme predišli nejasnostiam pri interpretácii, túto špeciálnu premennú vo vnorenej podmienke označíme `$1`. Slovo reprezentujúce vonkajšiu podmienku obsahuje teda dve špeciálne premenné: `$0`, ktorou sa ohodnocujú uzly prvého argumentu vonkajšej selekcie a `$1` ohodnocovaná uzlami z vnútornej selekcie. Ak by sme mali prípad troch vnorených podmienok, tak špeciálnou premennou v treťom vnorení je `$2`. Aby sme predišli problémom s konfliktami premenných, musíme zabezpečiť, aby vnútorná podmienka neobsahovala premennú z vonkajšej podmienky.

Po zavedením viacerých špeciálnych premenných však potrebujeme vyriešiť jeden problém. Majme vnorenú podmienku, ktorá podľa predošlého návrhu musí obsahovať špeciálnu premennú `$1`. Ked nadíde chvíľa, že vnútorný operátor σ začne interpretovať vnorenú podmienku podľa prípadu bez vnorených podmienok, nastane problém. Operátor vyhodnotenia pravdivosti chce ohodnotiť špeciálnu premennú `$0`. Táto špeciálna premenná však vo vnútornej podmienke nie je, keďže sme ju zakázali. Je tam len premenná `$1`. Riešenie tohto problému nám však poskytne substitúcia posun (pozri definíciu 10). Táto premenná posunie všetky špeciálne premenné "o jednu nižšie". Čiže špeciálnu premennú `$1` substituuje premennou `$0`. Tým získame σ-podmienku, s ktorou môžeme pracovať tak ako v prípade bez vnorenej podmienky.

Výsledný postup je teda nasledovný:

1. Vezmi uzol u z množiny v prvom argumente.
2. Vezmi σ-podmienku obsahujúcu premennú `$1` a neobsahujú premennú `$0`.
3. Aplikuj na σ-podmienku substitúciu do formuly $\overline{\overline{\text{posun}}}$, ktorá vznikne ako rozšírenie zobrazenia substitúcie do premenných podľa definície formuly.
4. V procese interpretácie σ-podmienky ohodnoť špeciálnu premennú `$0` uzlom u.

5. Ak je σ-podmienka pri danom ohodnotení premenných v interpretácii splniteľná, vlož u do výslednej množiny. V opačnom prípade prejdi na krok 1.
6. Skonči, ak si prešiel všetky uzly množiny v prvom argumente.

Ale na to, aby prípad pre vnorené podmienky fungoval aj na začiatku interpretovania nevnorenej podmienky, potrebujeme vyššie spomenutý popis trochu poupraviť. Každá σ-podmienka, ktorá je parametrom operátora selekcie musí obsahovať premennú `$1` a nesmie obsahovať premennú `$0`. Prvá vec, ktorá sa vykoná pred spustením interpretácie, je aplikácia substitúcie $\overline{\overline{\text{posun}}}$ na σ-podmienku. Tá nám prevedie túto σ-podmienku na podmienku, ktorá bude určite obsahovať premennú `$0`.

Pristúpme teraz k formálnej definícii operátora selekcie, ktorý sa pridržiava načrtnutých ideí.

Definícia 13 (operátor σ). *Majme interpretáciu $\mathfrak{M}$. Nech $U \subseteq \mathsf{Node}$. Nech φ je σ-podmienka obsahujúca špeciálnu premennú `$1` typu* Node *a neobsahujúca špeciálnu premennú `$0` typu* Node*. Nech $\mathfrak{e}$ je ohodnotenie premenných vo φ a $\mathfrak{I}$ je vyhodnotenie pravdivosti v interpretácii $\mathfrak{M}$.* Operátorom selekcie σ *nazývame funkciu*

$$\sigma : \mathcal{P}(\mathsf{Node}) \times \mathsf{String} \times \mathcal{E} \to \mathcal{P}(\mathsf{Node})$$

definovanú nasledovne:

$$\sigma(U, \varphi, \mathfrak{e}) = \begin{cases} \left\{ u \in U : \mathfrak{M} \models \overline{\overline{\text{posun}}}(\varphi) \left[\begin{smallmatrix} \mathfrak{e} \\ \$0 \mapsto u \end{smallmatrix} \right] \right\}, & \text{ak } \varphi \in \sigma\text{-Condition} \\ \text{nie je definovaná}, & \text{inak.} \end{cases}$$

Poznámka 1. Funkciu $\sigma(U, \varphi, \mathfrak{e})$ budeme niekedy označovať ako $\sigma_{\mathfrak{e}}(U, \varphi)$.

Definícia 14. *Nech $\mathfrak{e}$ je ohodnotenie premenných. Pre funkciový symbol* `selection` *jazyka $\mathfrak{J}_\sigma$, kde* `selection`$\langle$NodeSet, String$\rangle$: NodeSet*, definujeme pre $U \in \mathcal{P}(\mathsf{Node})$, $s \in \mathsf{String}$:*

$$\mathfrak{M}_{\mathcal{F}}(\texttt{selection})(U, s) = \sigma_{\mathfrak{e}}(U, s),$$

kde $\sigma_{\mathfrak{e}}$ je zobrazenie z predošlej definície 13.

5 Záver

V práci sme formálne podali definíciu dátového modelu a sady operátorov stromovej algebry. Ďalej sme definovali jazyk logiky prvého rádu pre booleovské podmienky v operátore selekcie. Samozrejme, v rozširovaní a dopĺňaní stromovej algebry možno pokračovať. Do budúcnosti je možné prezentovať formálny preklad syntaxe jazyka $\mathfrak{J}_\sigma$ do gramatiky jazyka XPath.

Reference

1. Novotný, R., Biblická konkordancia. Diplomová práca. Ústav informatiky. Prírodovedecká fakulta Univerzity Pavla Jozefa Šafárika, Košice, 2005
2. Beech, D., Malhotra, A., Rys, M., A Formal Data Model and Algebra for XML. Communication to the W3C, September 1999
3. Rjaško, J., XML a stromy. Diplomová práca. Ústav informatiky. Prírodovedecká fakulta Univerzity Pavla Jozefa Šafárika, Košice, 2004
4. XML Path Language (XPath), Version 1.0, W3C Recommendation 16 November 1999 [online] [citované 29. mája 2005] `http://www.w3.org/TR/xpath`
5. Krajči, S., Matematická logika. [online] [citované 1. júna 2005] `http://ics.upjs.sk/~krajci/skola/vyucba/ucebneTexty/logika/logika.pdf`

Odhad struktury dat a induktivní logické programování

Martin Řimnáč

Ústav informatiky, Akademie věd ČR,
Pod Vodárenskou věží 2, 182 07 Praha 8
rimnacm@cs.cas.cz

Abstrakt Odhadování struktury dat je jednou z možností, jak automatizovaným způsobem interpretovat data. Ta mohou být popsána pomocí modelu funkčních závislostí, vytváření takového modelu lze srovnat s některými technikami strojového učení. Tento příspěvek shrnuje vybrané základní techniky induktivního logického programování a analyzuje je z pohledu metody odhadování struktury dat. Ukazuje se, že techniky induktivního logického programování lze v některých případech převést právě odhadování struktury dat.

Jednou z klíčových otázek dnešní doby je zpřístupnění informací nejen lidem, ale i výpočetním prostředkům, které by na dotaz uživatele získaly potřebné informace a provedly by na jejich základě potřebné úkony (integrace z více zdrojů, odvozování informací, zahrnutí předdefinovaných uživatelských profilů nebo preferencí) tak, aby uživatel získal informaci ve zkompletované, přehledné formě.

Možnou odpovědí na tyto otázky je vize sémantického webu [1], tedy rozšíření současných webových zdrojů o dokumenty navíc vhodné ke strojovému zpracování. Tyto dokumenty by vedle samotné informace obsahovaly i její popis - sémantiku, což by umožnilo strojové zacházení s těmito dokumenty. V dnešní době se jeví jako perspektivní formát RDF (Resource Description Framework) [2], popisující realitu pomocí binárních predikátů $vlastnost(objekt, subjekt)$ s návazností na deskripční logiky jako odvozovacího mechanizmu, nebo formát OWL (Web Ontology Language) [3], mapující realitu pomocí vztahů mezi množinami.

Druhou možností je využití vybraných typů současných webových zdrojů, extrakce informace z nich pomocí web-minnigových metod a uložení těchto informací do databází. K tomu je však potřeba znát strukturu dat (alespoň na pokrytí požadavků na normální formu databázového modelu), přičemž lze využít dále nastíněné metody odhadování struktury dat [4]. Mezi takové zdroje dat patří především webová rozhraní pro databázové pohledy, např. elektronické obchody.

1 Odhadování struktury dat

Odhadování struktury dat vychází z metod dekomponujících databázový model [5], [6], [7]. Úlohou těchto metod je upravit vstupní model popsaný pomocí množiny funkčních závislostí na nový model tak, aby splňoval další požadavky

jako vyšší normální formu či automatické rozšíření modelu o další vlastnosti, např. modely označované jako multi-level secure [8]. Výstupem těchto metod je model popsaný množinou funkčních závislostí. Aby byl výčet dekompozičních metod kompletní, uveďme ještě metody označované jako vertical a horizontal partitioning [9,10], sloužící k dekompozici modelu s ohledem na paralelizaci přístupu k datům.

Na rozdíl od výše uvedených přístupů, metoda odhadování struktury dat [4] získává model pouze z dat, vstupem metody je tabulka dat a výstupem je model, minimální množina funkčních závislostí platných na množině vstupních dat.

Metoda je primárně vyvíjena jako doplněk současných web-miningových metod [11]. Ty operují především nad metadaty webových stránek a zprostředkovávají o nich souhrnné informace. Navrhovaný doplněk rozšiřuje tyto metody o extrakci samotných prezentovaných dat a podrobuje je analýze a další agregaci.

1.1 Základní vlastnosti funkčních závislostí

V tomto odstavci zopakujeme některé základní vlastnosti známé z teorie relačních databází.

Pokud atributy jsou vzájemně funkčně závislé, mají shodnou velikost aktivních domén.

$$A_1 \to A_2 \wedge A_2 \to A_1 \Rightarrow \|\mathscr{D}_\alpha(A_1)\| = \|\mathscr{D}_\alpha(A_2)\| \tag{1}$$

Množina funkčních závislostí vykazuje transitivitu, tedy:

$$A_1 \to A_2 \wedge A_2 \to A_3 \Rightarrow A_1 \to A_3 \tag{2}$$

Funkční závislost mezi atributy může existovat pouze v případě, kdy velikost aktivní domény závislého atributu není větší nežli velikost aktivní domény atributu, na němž závisí.

$$A_1 \to A_2 \Rightarrow \|\mathscr{D}_\alpha(A_1)\| \geq \|\mathscr{D}_\alpha(A_2)\| \tag{3}$$

Triviální funkční závislosti jsou ty, které nepopisují vlastnosti modelu, platí nezávisle na něm. Mezi ně patří např.

$$\begin{aligned} A_i &\to A_i \\ A_i &\to \oslash \end{aligned} \tag{4}$$

Komplexní atribut slučuje několik atributů v jeden celek. Pokud (komplexní) atribut H funkčně závisí na (komplexním) atributu G, funkční závislost atributu H na atributu G rozšířeném o libovolný další atribut je triviální.

$$G \to H \Rightarrow G' \to H \quad \forall G' \supset G \tag{5}$$

1.2 Algoritmus odhadu struktury dat

Mějme množinu vstupní tabulku dat (relaci) o n sloupcích a hledejme minimální množinu funkčních závislostí, která daná data popisuje.

Algoritmus inicializujeme prvním prvkem množiny dat. V této chvíli můžeme hovořit o modelu obsahujícím n^2 funkčních závislostí $A_j \rightarrow A_i$, budeme-li uvažovat též komplexní atributy, pak model pokrývá $n!$ (i triviálních) funkčních závislostí.

Výstupem algoritmu je `kostra modelu`, minimální množina funkčních závislostí reprezentující elementární vazby v modelu. Taková množina neobsahuje žádné triviální funkční závislosti (4, 5) a obsahuje pouze ty funkční závislosti, které tvoří jádro množiny všech netriviálních funkčních závislostí.

Hledání takového jádra vůči jejímu tranzitivnímu uzávěru je však NP úplná úloha [5] a nemá jedinečné řešení. Proto navržený algoritmus vytvoří v prvním kroku triviálním způsobem kostru modelu a takto vytvořenou kostru v každém kroku aktualizuje, přičemž problém aktualizace je již polynomiálně řešitelný.

Povšimněme si, že po přidání prvního záznamu každý atribut je extenzionálně funkčně závislý na každém jiném atributu (neexistuje žádný další záznam, který by takovou závislost porušoval), tyto závislosti jsou vzájemné.

Přidejme nyní další záznam. Předpokládejme, že některé atributy nabývají stejné hodnoty a některé hodnoty jiné. To podle (3) znamená, že bude porušena některá z funkčních závislostí.

V okamžiku, kdy je do úložiště přidán další záznam, je nutné nejprve aktualizovat kostru modelu. Abychom zachovali daný požadavek "elementárnosti vazeb v kostře", definujme primární kritérium uspořádání atributů s ohledem na (3) tak, že

$$\|\mathscr{D}_\alpha(A_i)\| < \|\mathscr{D}_\alpha(A_j)\| \Rightarrow i < j \tag{6}$$

Během aktualizace kostry při změně pořadí atributů může dojít k situaci, kdy existuje kratší hrana, která je doplňkem hran tvořících kostru (délku hrany $\delta(A_i, A_j)$ reprezentuje rozdíl pozic v uspořádání atributů). V tomto případě se tato hrana stává hranou tvořící kostru na úkor delší z hran.

Předpokládejme, že kostra modelu je aktualizována, otestujme nyní, zda-li některé funkční závislosti nejsou porušeny.

Protože kostra tvoří jádro množiny, pokud je porušena jakákoli funkční závislost v modelu, musí být porušena některá z funkčních závislostí $A_j \rightarrow A_i$ tvořící kostru. Díky tomuto faktu není nutné testovat při každém kroku všechny funkční závislosti, ale pouze ty, které tvoří kostru (takových je nejvýše $2n$). V případě, kdy je taková funkční závislost porušena, je nutné navíc testovat i funkční závislosti se stejnou stranou a aktualizovat kostru (najít jiné funkční závislosti s minimální délkou spojující atributy původně spojené přes A_i a A_j).

Přidejme další záznamy a předpokládejme, že při testování funkčních závislostí v aktuálním kroku je porušena závislost $A_j \rightarrow A_i$, přičemž v minulosti byly porušeny i funkční závislosti $A_k \rightarrow A_i$ a $A'_k \rightarrow A_i$ a mezi atributy A_k, A'_k a A_i neexistuje žádná funkční závislost. To může vést ke vzniku netriviální závislosti na komplexním atributu $\{A_j, A_k, A'_k\}$.

Vložme nový `komplexní atribut` jako `virtuální atribut` do modelu. Virtuální atribut reprezentující kartézský součin je použit, aby nebylo nutné úlohu řešit v prostoru hypergrafů. Otestujme funkční závislost $\{A_j, A_k, A'_k\} \rightarrow A_i$. Pokud je splněna, virtuální atribut je v modelu ponechán. Pokud arita komplexního atributu je větší než 2, testujeme pomocí dekompozice komplexního atributu, zda-li daná závislost není triviální (5). Všechny získané netriviální funkční závislosti jsou přidány do kostry modelu a model je rozšířen funkční závislosti z nového uzávěru.

Jak je patrné, vznik netriviální funkční závislosti na komplexním atributu o m atributech je možný za podmínky, že existuje $m \leq m'$ (alespoň m z celkových m') porušených funkčních závislostí na atributu A_i, který bude na hledaném komplexním atributu závislý a které nemají mezi sebou žádné jiné funkční závislosti. Operace hledání komplexního atributu je nepolynomiálně složitá, je potřeba otestovat celkem až k kombinací (dekompozice atributu), přičemž (při uvažování celočíselného dělení)

$$k = \max_{\forall i < m'} \binom{m'}{i} = \binom{m'}{m'/2} = \frac{m'!}{(m'/2)!^2} \tag{7}$$

Jedinou možností, kdy bude hledání komplexního atributu polynomiálně řešitelné, je případ, kdy velikost aktivní domény komplexního atributu roste pomaleji než velikost aktivní domény atributu na komplexním atributu závislém.

Algoritmus 1

```
A .. množina atributů
D .. množina dat
C .. matice pokrytí
S .. kostra modelu
C = {c_ij = 1, i ≠ j, ∀i, j ∈ 1..|A|}
S = {A_i → Aj : |i − j| = 1}
Pro ∀d ∈ D
{
    ulož d do úložiště
    aktualizuj velikost domén a podle změn i pořadí atributů v S a C
    while (zmena)
   {
      ∀(i, j, k), i < k < j : (A_i → A_j) ∈ S ∧ (A_k → A_j) ∈ S ∧ C_ik = 1
        S = S − {A_i → A_j} ∪ {A_i → A_k}
      ∀(i, j, k), i < k < j : (A_i → A_j) ∈ S ∧ (A_i → Ak) ∈ S ∧ C_kj = 1
        S = S − {A_i → A_j} ∪ {A_k → A_j}
    }
    testuj ∀(A_i → A_j) ∈ S
    {
      pokud A_i → A_j porušeno
      {
       testuj ∀(A_u → A_v), kde C_iv = 1 ∨ C_uj = 1
       pokud porušeno, C_uv = 0
      }
      S = S − (A_i → Aj)
      rozšiř kostru
      testuj komplexní atributy
    }
}
```

1.3 Vlastnosti modelu

Uspořádejme modely podle počtu všech funkčních závislostí. Označme n počet atributů, m počet záznamů, M_k pak model po k-tém záznamu. Podle Algoritmu 1 počet hran monotónně klesá, tedy

$$|M_0| = n! \tag{8}$$

$$M_i < M_j \Rightarrow |M_i| > |M_j| \tag{9}$$

$$\forall M_k : M_0 \leq M_k \leq M_m \leq M_\infty \tag{10}$$

Označíme-li M_∞ nejpřesnější (logický) model reality, poslední nerovnost značí, že model vrácený algoritmem může oproti tomuto modelu obsahovat navíc některé funkční závislosti. Tento rozdíl může být způsoben nereprezentativními daty (jedná se o algoritmus strojového učení, kde reprezentativnost dat hraje marginální roli), jednak granularitou hodnot domén jednotlivých atributů (konečný počet záznamů versus nespočitatelné domény atributů v realitě). Z těchto důvodů hovoříme o odhadu struktury, nikoli o její rekonstrukci. První problém lze vyřešit integrací dat z více zdrojů, druhý pak představuje principiální limit metody.

Výhodou odhadování struktury dat je, že výsledný model není závislý na pořadí vstupních dat, což vyplývá z definice funkční závislosti.

2 Induktivní logické programování

Induktivní logické programování (ILP) [12,13] je úloha strojového učení, jejíž úkolem je naučit se odvodit predikát P_H z literálů ve znalostní bázi B, přičemž tyto literály tvoří fakta nad množinou konstant K.

Vstupem těchto algoritmů je znalostní báze B a funkce $\epsilon \rightarrow \{\ominus, \oplus\}$, $\epsilon \subset K_P^a$ určující pravdivostní ohodnocení predikátu P_H arity a na trénovací množině. Výstupem je množina hypotéz H definujících predikát P_H.

Pro jednoduchost uvažujme pouze základní verze algoritmů, hledaná hypotéza se skládá z literálů, jejichž argumenty jsou pouze proměnné, báze dat obsahuje pouze literály s konstantami, neuvažujeme rekurzivní definice a všechny literály jsou deterministické.

2.1 GOLEM

Jedním ze základních algoritmů ILP je algoritmus GOLEM [12],[16]. Je založen na principu označovaném jako "relative least general generalization".

V jednoduchosti lze říci, že algoritmus převede literály na jejich modely a ty skládá podle konstant tak, aby všechny kombinace konstant v literálu nepokrývaly žádný negativní příklad. Tedy např.

$$p(K_1, K_2), p(K_3, K_4), p(K_1, K_4) \rightsquigarrow p(K_{1,3}, K_{2,4}) \tag{11}$$

Pak kombinuje tyto modely podle množin konstant, tedy např.

$$p(K_{1,3}, K_{2,4}), q(K_{1,3}) \rightsquigarrow p(K_{1,3}, K_{2,4}) \wedge q(K_{1,3}) \tag{12}$$

Pokud tedy převedeme hledaný predikát P_H a literály $b \in B$ na jejich modely, pomocí výše nastíněných operací lze odvodit generalizovaný model hypotézy o predikátu P_H a tento model zpětně převést na množinu literálů popisující hypotézu H.

2.2 FOIL

Dalším algoritmem ILP je algoritmus FOIL [12,14,15], který volí jiný způsob hledání klauzulí, oproti algoritmu GOLEM pracujícího na bázi generalizace, volí specializaci. Oproti uvedené notaci místo pravdivostní funkce do algoritmu vstupuje množina pozitivních ϵ^+ a negativních ϵ^- příkladů.

Algoritmus 2

ϵ^+ .. množina pozitivních příkladů
ϵ^- .. množina negativních příkladů
B .. množina literálů - znalostní báze
H .. množina hypotéz $H = \emptyset$
while $(\epsilon^+ \neq \emptyset)$
{
  Specialization algorithm $(\epsilon^+, \epsilon^-, B)$
  {
    $Q = \emptyset$
    $\epsilon_S^+ = \epsilon^+$, $\epsilon_S^- = \epsilon^-$, $B_S = B$
    while $(e_S^- \neq \emptyset \wedge B_S \neq \emptyset)$
    {
      Pro $\forall b \in B_S$:
        Hledej literál b nejlépe rozdělující ϵ^+ a ϵ^-
      $\epsilon_S^+ = \{e \in \epsilon_S^+ \text{ , } e \text{ splňuje } b\}$
      $\epsilon_S^- = \{e \in \epsilon_S^- \text{ , } e \text{ splňuje } b\}$
      $Q = Q \cup \{b\}$
      $B_S = B_S - \{b\}$
    }
  }
  Postprocess $(P \leftarrow Q)$
  $\epsilon^+ = \epsilon^+ - \{e \in \epsilon^+, e \text{ splňuje } Q\}$
  $H = H \cup (P \leftarrow Q)$
} until $(\epsilon^+ \neq \emptyset)$

Algoritmus 2 začíná s prázdnou množinou hypotéz.

Nejprve zavolá specializační algoritmus, jehož úkolem je postupně složit optimální tělo Q z literálů b. Tyto literály jsou hledány přes celou bázi znalostí B, mírou optimality se volí entropie. Pro hledání dalšího literálu se omezí množina pozitivních a negativních příkladů pouze na ty, které jsou předchozími literály těla Q splněny. Pokud literál nepokrývá žádný negativní příklad, není nutná další specializace a aktuální tělo je zvoleno jako optimální kandidát.

V opačném případě je nutné dále specializovat. Pokud již specializovat nelze (všechny literály ze znalostní báze byly již použity), algoritmus výběru končí. Pokud takovou kombinaci literálů zařadíme do množiny hypotéz H, nebude výsledná hypotéza konzistentní.

Specializační algoritmus vrací optimální tělo pro hypotézu. Toto tělo ještě můžeme upravit pomocí postprocesního algoritmu, který eliminuje přebytečné literály v těle Q (takové literály, které sice mají velkou entropii, avšak na výsledné ohodnocení nemají sami o sobě vliv).

Všechny pozitivní příklady, které jsou pokryty tělem Q, jsou z množiny pozitivních příkladů odebrány a množina hypotéz je rozšířena o klausuli $P \leftarrow Q$.

Pokud existují stále hypotézou nepokryté pozitivní příklady, algoritmus se opakuje.

2.3 Jiné algoritmy

V přehledu nebyly ještě uvedeny algoritmy používající inverzních metod (inverzní rezoluce, inverzní dedukce - ALEPH, CIGOL, PROGOL).

Dalším přístupem je transformovat úlohy ILP na jiné algoritmy strojového učení, např. na algoritmy klasifikační [12,13], kdy oddělující nadplocha je dána pomocí pravdivostní funkce $\epsilon \rightarrow \{\ominus, \oplus\}$ predikátu P_H a jako relevantní atributy klasifikační úlohy jsou hodnoty pravdivostního ohodnocení literálů b ve znalostní bázi B (úlohu je nutné převést do výrokové formy). Nejčastěji se pro tyto účely využívají algoritmy generující rozhodovací stromy. Zmiňme i data-miningové systémy LINUS a DINUS [12], kterých lze rovněž použít pro účely ILP.

3 Řešení problému induktivního logického programování pomocí odhadování struktury dat

Nejprve je nutné zadání úlohy ILP transformovat na zadání metody odhadu struktury dat. To je možné vytvořením tabulky výroků a spuštění algoritmu odhadu struktury dat na tuto tabulku. Výslednou množinu funkčních závislostí je pak nutné znovu transformovat do množiny klausulí tak, aby výsledek byl shodný s výsledky ILP metod.

3.1 Transformace úlohy

Výsledkem ILP je množina hypotéz nad literály, jakým způsobem závisí hodnoty ohodnocení literálů na ohodnocení hledaného predikátu. Takový výsledek metoda odhadu struktury neposkytuje, určuje jen, které predikáty jsou relevantní pro určení pravdivostního ohodnocení hledaného predikátu P_H, vrací množinu predikátů, na kterých ohodnocení hledaného predikátu funkčně závisí.

Pro potřeby metody je nutné ILP úlohu transformovat do výrokové formy [12] pomocí některé propozicionalizační metody, nejčastěji do tabulky pokrývající všechny výroky. Základem takové tabulky je pravdivostní ohodnocení hledaného predikátu P_H arity a podle hodnot jeho vstupních proměnných V_{H1} až V_{Ha} (atribut P_H udává, zda-li se jedná pozitivní $\oplus$ nebo negativní $\ominus$ trénovací příklad.

K této základní tabulce přidáme literály ze znalostní bázi B. Pro každý predikát s odpovídající kombinací proměnných bude vytvořen atribut A_i, který bude nabývat hodnoty podle uvedené notace, chybějící literál v bázi se ohodnotí negativním symbolem. V některých případech bude nutné zavést další volné proměnné pomocí atributů V_{xi}. Poznamenejme, že tabulka musí obsahovat všechny možné kombinace proměnných (i volných) v predikátech, což v některých situacích může vést ke kombinatorické explozi.

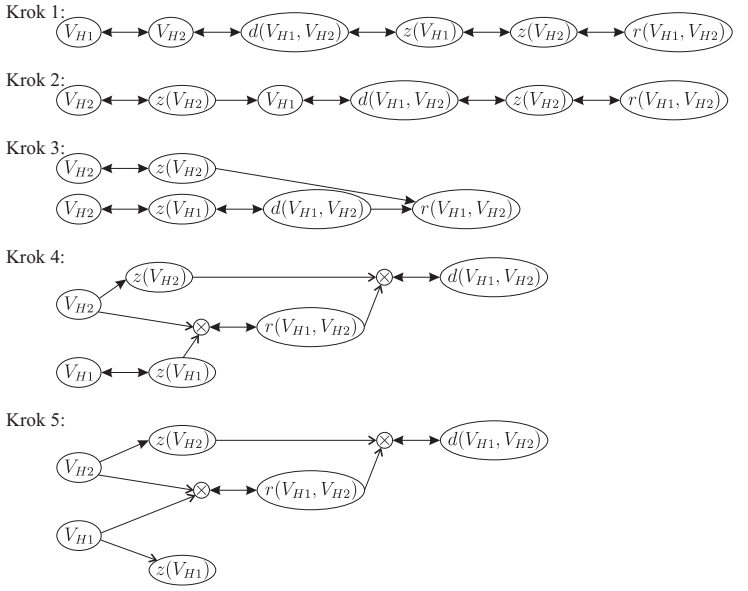
3.2 Příklad odhadu struktury

Pro ilustraci uveďme jednoduchý příklad takto vytvořené tabulky (13). Hledaným predikátem P_H je predikát $d(V_{H2}, V_{H1})$ popisující skutečnost, že objekt V_{H1} je dcerou objektu V_{H2}. Znalostní báze B obsahuje predikát $r(V_{H1}, V_{H2})$ (objekt V_{H1} je rodičem objektu V_{H2}) a predikát $z(V_{H1})$ (objekt V_{H1} je ženského pohlaví).

n	V_{H1}	V_{H2}	$d(V_{H2}, V_{H1})$	$z(V_{H1})$	$z(V_{H2})$	$r(V_{H1}, V_{H2})$	$r(V_{H2}, V_{H1})$	$r(V_{H1}, V_{H1})$	$r(V_{H2}, V_{H2})$
1	Eva	Tomáš	$\oplus$	$\oplus$	$\ominus$	$\oplus$	$\ominus$	$\ominus$	$\ominus$
2	Eva	Kamila	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\ominus$	$\ominus$	$\ominus$
3	Milan	Tomáš	$\ominus$	$\ominus$	$\ominus$	$\oplus$	$\ominus$	$\ominus$	$\ominus$
4	Eva	Milan	$\ominus$	$\oplus$	$\ominus$	$\ominus$	$\ominus$	$\ominus$	$\ominus$
5	Karel	Milan	$\ominus$	$\ominus$	$\ominus$	$\ominus$	$\ominus$	$\ominus$	$\ominus$

(13)

Následující obrázek (1) ukazuje vývoj modelu po každém přidání řádku z tabulky (13) podle Algoritmu 1.



Obr. 1. Ilustrativní příklad.

3.3 Transformace výsledku

Z konečného modelu vyplývá, které predikáty jsou relevantní pro určení ohodnocení hledaného predikátu P_H. Aby byl tento postup srovnatelný s algoritmy ILP, je nutné vrátit výsledek ve shodném tvaru, tedy jako seznam klausulí. Z tabulky vybereme pouze atributy odpovídající těm predikátů, na kterých je ohodnocení hledaného literálu závislé a ty řádky, které obsahují příklady z množiny pozitivních příkladů. Pro každou jedinečnou kombinaci ohodnocení literálů budeme

generovat klausule, je-li ohodnocení literálu v daném řádku negativní, bude literál v klausuli uveden s negací, je-li pozitivní, bude uveden bez negace.

Protože pro pravdivostní ohodnocení $d(V_{H1}, V_{H2}) = \oplus$ je podle (13) podmínkou, aby $r(V_{H1}, V_{H2}) = \oplus$ a zároveň $z(V_{H1}) = \oplus$, budou oba literály tvořící tělo podle výše uvedeného postupu klausule pozitivní, tedy

$$d(V_{H1}, V_{H2}) \leftarrow r(V_{H1}, V_{H2}), z(V_{H1}) \tag{14}$$

Takto vygenerované klausule nemusí být optimální, však je lze upravit pomocí libovolného z algoritmů minimalizující boolské funkce nebo postprocesní úpravu (jako FOIL [12,14]) ověřující skutečný vliv literálu na výsledné přiřazení do kategorií trénovací množiny.

4 Srovnání metod

Výhodou nastíněné metody odhadování struktury dat je, že prostor možných řešení není prohledáván hladovým způsobem, jako např. u algoritmu FOIL, tedy nalézá všechna přípustná řešení. Představme si, že hledaný predikát lze odvodit více způsoby $P_H \leftarrow \{L_i^1\}$ a $P_H \leftarrow \{L_i^2\}$, přičemž $L^1 \cap L^2 = \oslash$. Algoritmus FOIL vrátí pouze množinu predikátů L^1 nebo L^2 (teoreticky náhodně vybere jednu z nich) [14], [15]. Uvažování více alternativ sice vede k tomu, že vygenerovaná teorie není minimální, ale lépe vystihuje daný predikát.

Algoritmus odhadování struktury dat nejen že může generovat teorie pro více literálů zároveň stejně jako např. metody použité v systému DINUS [12], ale může uvažovat i vztahy mezi hledanými literály.

Podle (7) je rozšiřování modelu o komplexní atribut (v ILP ekvivalentní specializaci či generalizaci) je nepolynomiálně složité vyjma speciálních případů nepokrývající problematiku ILP, složitost je dána aritou funkční závislosti, která nebývá nikterak velká. U algoritmu FOIL je však tato složitost podmíněna počtem predikátů, který je vyšší nežli arita predikátů, u algoritmu GOLEM se tato složitost odvozuje od velikosti domény proměnných. Naopak nevýhodou metody odhadu struktury dat je nutnost výroky převést do tabulky, přičemž závislosti lze hledat pouze v rámci jednoho řádku, tedy neumožňuje rekurzivní definice. To sice lze vyřešit analogickým způsobem jako rozšířená verze FOIL [15], avšak podstatně se zvyšuje počet volných proměnných a jim odpovídajících literálů.

Odhadování struktury dat principem zaručuje nezávislost na pořadí vstupních dat, což u některých inverzních metod ILP nelze zaručit.

Srovnání vyjmenovaných klasických algoritmů ILP s metodou odhadu struktury tedy nelze obecně shrnout, záleží případ od případu, kdy je použití jedné z metod efektivnější. Slabým místem metody odhadu struktury dat je především nutnost mít k dispozici celý záznam, který reprezentuje všechny možné kombinace proměnných v literálech. Zde se však rýsuje možnost použít mechanizmů podobných integraci dat a hledání globálního modelu dat, čímž by tato nepříjemná nutnost odpadla. Integrační mechanizmy při odhadování struktury dat jsou jedním z dalších předsevzatých úkolů a budou tématem na další práci.

Poděkování

Práce byla podpořena projektem 1ET100300419 programu Informační společnost "Inteligentní modely, algoritmy, metody a nástroje pro vytváření sémantického webu" a částečně i výzkumným záměrem AV0Z10300504 "Informatika pro informační společnost: Modely, algoritmy, aplikace".

Reference

1. Grigoris Antoniou, Frank van Harmelen, A Semantic Web Primer. MIT Press, 2004, ISBN: 0-262-01210-3
2. Eric Miller, Ralph Swick, Dan Brickley, Resource Description Framework, 2004, `<http://www.w3.org/RDF/>` [on-line]
3. Eric Miller, Jim Hendler, Web Ontology Language, 2005, `<http://www.w3.org/2004/OWL/>` [on-line]
4. Martin Řimnáč, Web Information Integration Tool - Data Structure Modelling. In Proceedings of DMIN 2005, 2005, 37-40, ISBN 1-934215-79-3
5. G. Grahme, K. Räihä, Database Decomposition into Fourth Normal Form. In Conference on Very Large Databases, 1983, 186–196
6. G. Ausiello, A. D'Atri, M. Moscarini, Chordality Properties on Graphs and Minimal Conceptual Connections in Semantic Data Models. In Symposium on Principles of Database Systems, 1985, 164–170
7. Bruno T. Messmer, Horst Bunke, Efficient Subgraph Isomorphism Detection: A Decomposition Approach. In IEEE Transactions on Knowledge and Data Engineering, 2000, 307–323
8. F. Cuppens, K. Yazdanian, A Natural Decomposition of Multi-level Relations. In IEEE Symposium on Security and Privacy, 1992, 273–284
9. B.N. Shamkant, R. Minyoung, Vertical Partitioning for Database Design – a Graphical Algorithm. In SigMod, 1989, 440–450
10. L. Bellatreche, K. Karlapalem, A. Simonet, Algorithms and Support for Horizontal Class Partitioning in Obect–Oriented Databases. In Distributed and Parallel Databases 8, Kluwer Academic Publisher, 2000, 115–179
11. A.A. Barfourosh, M.L. Anderson, H.R.M. Nezbad, D. Perlis, Information Retrieval on the World Wide Web and Active Logic: A Survey and Problem Definition, 2000, `http://citeseer.ist.psu.edu/barfourosh02information.html` [online]
12. Nada Lavrač, Sašo Džeroski, Inductive Logic Programming - Techniques and Applications, Ellis Hordwood, Chichester, ISBN: 0-13-457870-8, 1994
13. Sašo Džeroski, Nada Lavrač, Relational Data Mining, Springer-Verlag, Berlin, ISBN: 3-640-42289-7, 2001
14. Quilan J., Learning Logical Definitions from Relations. In Machine Learning 5(3), 1990, 239–266
15. Quilan J., Knowledge Acquisition from Structured Data – Using Deteminate Literal to Assist Search. In IEEE Expert 6(6), 1991, 32–37
16. Muggeton S., Feng C., Efficient Introduction of Logic Programs. In Proceding of the First Conference on Algorithmic Learning Theory, 1990, 368–381

Data pozičně závislá a jejich dopad v mobilních databázích*

Roman Špánek

Ústav informatiky Akademie věd ČR,
Pod Vodárenskou věží 2, 182 07 Praha 8, Česká republika
spanek@cs.cas.cz

Abstrakt Mobilní telekomunikace a jejich prudký vývoj, daly podmět k vzniku nového oboru v oblasti informačních technologií, *Mobilní databáze*. Ve své podstatě jsou sice velmi podobné distribuovaných systémům, avšak je zde celá řada omezení, která přináší mnoho problémů čekajících na řešení. Jedním z hlavních rozdílů mezi mobilní a distribuovanou databází je přítomnost *pozičně závislých dat*. Příkladem by mohl být dotaz na aktuální výši daní. Tento dotaz zodpovězený v různých státech, bude jistě dávat různé hodnoty. Pro správné zodpovězení takovéto otázky, je bezpodmínečně nutné znát aktuální pozici uživatele v síti. Článek představuje možné vylepšení stávajících řešení, za cenu minimálních úprav v implementacích. Přístup je založen na myšlence, že uživatel nemá možnost absolutně volného pohybu v reálném prostředí, což bývá obvykle předpokládáno, ale jeho pohyblivost je omezena všudypřítomnými překážkami. Náš postup využívá stávajících implementací a jejich vhodným začleněním a rozšířením se snaží otázku aktuální pozice uživatele efektivně zodpovědět.

1 Současný stav problematiky

Mobilní databáze jsou pojmem, který je v posledních letech zmiňován stále častěji, jak v kruzích akademických tak i v průmyslu. Přes všechnu věnovanou pozornost je tato oblast stále na počátku svého vývoje a mnoho problémů bylo řešeno pouze částečně nebo vůbec. Přeneseme-li se přes všudypřítomné problémy bezpečné komunikace, technologického zabezpečení přenosu atd., až k vyšším vrstvám v modelu OSI, nalezneme problémy spojené s aplikační logikou a daty vůbec. Přestože mobilní databáze jsou v mnoha rysech podobné distribuovaným databázovým systémům a dalo by se říci, že by většina postupů měla být bez větších problému použitelná i zde, narazíme na několik závažných rozdílů.

Než se jim však budeme věnovat, je třeba zavést několik pojmů. Dotazy v mobilním prostředí lze obecně rozdělit na *periodicky vysílaná* a *vyžádaná data*. První

* Práce byla podpořena projektem 1ET100300419 programu Informační společnost (Tématického programu II Národního programu výzkumu v ČR: Inteligentní modely, algoritmy, metody a nástroje pro vytváření sémantického webu) a výzkumným záměrem AV0Z10300504 "Informatika pro informační společnost: Modely, algoritmy, aplikace".

skupinu nejlépe charakterizují data týkající se dopravní situace či předpovědi počasí. Společným jmenovatelem je jim skutečnost, že jsou vysílána periodicky v předefinovaných časových intervalech všem uživatelům v dané lokalitě. Odpadá zde většina problémů, o kterých budeme hovořit v druhém případě tzv. vyžádaných dat. Jako vyžádaná data uvažujeme případy, kdy uživatel aktivně vydá dotaz do databáze a pokud byl správně položen jsou mu zaslána i požadovaná data. V tomto případě je nutné brát v úvahu tzv. *data závislá na místě položení dotazu*. Jinými slovy výsledek dotazu je odvislý od místa jeho položení, takže na příklad dotaz typu: "Kde je nejbližší hotel?" bude zřejmě dávat jiné výsledky pro uživatele v Praze a jiné pro uživatele v Bratislavě. A právě lokačně závislá data představují jeden s hlavních rozdílů a omezení, které část přístupů používaných v distribuovaných systémech staví mimo pole využitelnosti. Kapitola 2 bude věnována právě otázkám místně závislých dat a transakčnímu zpracovaní v mobilních databázích.

Jedním z dalších a neméně důležitých rozdílů, jsou omezené prostředky, ať již výpočetní nebo paměťové, případně omezená kapacita baterií a limitovaná šířka pásma bezdrátových linek. V takovémto prostředí je například velmi často používaný 2PL protokol příliš náročný na množství přenesených dat.

Zbytek příspěvku je dělen následně: druhá kapitola je věnována otázkám trans-akčního zpracování v mobilních databází. Následující kapitola stručně zmiňuje stávající situaci ve správě pozice uživatele a je především věnována našemu rozšíření a zpřesnění stávajících řešení. Poslední sekce je věnována závěru a shrnutí článku.

2 Transakce v mobilních databázích

Za zpracování mobilní transakcí je zodpovědný *koordinátor transakcí*. Jako koordinátor může být zvolen samotný uživatel nebo základní stanice mobilní sítě (ZS). ZS bývá doporučována vzhledem k tomu, že uživatel může být v průběhu zpracování transakce odpojen od sítě z různých důvodů (kapacita baterie, nedostupnost signálu atd.). Koordinátor má na starosti průběh zpracování celého dotazu, kompletaci dat a jejich přeposlání uživateli. Definice transakce v mobilních databázích je následující:
Mobilní transakce je trojice $< Fi, Li, FLMi >$*, kde* Fi *je množina fragmentů spuštění,* Li *je množina lokací a* $FLMi$ *je množina zobrazení fragmentů spuštění na lokace [1].*
Uživatel sestaví dotaz a přepošle jej koordinátorovi. Ten dotaz rozdělí na jednotlivé fragmenty, složky dotazu, které se musejí vykonat na oddělených místech nebo je oddělené zpracování efektivnější. Pak jsou fragmenty rozeslány do jednotlivých lokací. Předpokládejme, že všechny fragmenty byly úspěšně provedeny a byly zaslány zpět koordinátorovi. Ten by je měl přeposlat uživateli, který ovšem mohl změnit pozici nebo mohl být odpojen od sítě. Hlavně v případě změny lokace je důležité co nejefektivněji zjistit jeho aktuální pozici. Bude-li k dispozici mechanismus pro odvození další možné trajektorie na základě předešlé, pak bude tento úkol možné řešit s minimálním zatížením mobilní sítě.

3 Uživatelova pozice a její správa

Tato část bude věnována otázkám a stávajícím řešením v oblasti spravování pozice uživatele.

3.1 Stávající řešení

Správa pozice se dá rozdělit podle přístupu na dvě hlavní podskupiny a to *deterministický* a *pravděpodobnostní*. Deterministický přístup byl poprvé představen v [2] jako *Pure Paging Strategy*, jehož hlavní nevýhodou byla značná spotřeba především "bezdrátových" zpráv. Principem je nahlásit každou změnu pozice, ke které u uživatele došlo. Proto byly navrženy dokonalejší metody pro odvozování pozice. Hlavní rozdíl spočíval především v proceduře zajišťující nahlášení změny pozice uživatele v síti, kde docházelo k hlavní spotřebě zpráv. Aby bylo možné zprávy ušetřit, byla obvykle celá sít rozdělena na menší oblasti a změna pak byla nahlášena jen v případě, že uživatel překročil její hranici. DasBit a Mitra navrhli *Profile Based Scheme* [3], kde je uvažován i uživatelův dosavadní pohyb. Mobilní jednotky (MJ) byly rozděleny podle frekvence s jakou docházelo k přechodu hranic, na jednotky s malou frekvencí změny (Less Frequency Profile Changer (LFPC)) a uživatele rychle měnící pozici (Frequent Profile Changer(FPC). Zatímco LFPC uživatelé měly povinnost hlásit svou pozici s menší frekvencí, pak FPC uživatelé museli svoji pozici oznamovat častěji. Přístup lépe postihuje přirozené chování uživatelů a umožňuje šetrnější zacházení s kapacitou baterií a množstvím odeslaných zpráv. Jak bylo zmíněno v úvodu kapitoly, správu lokace lze rozdělit na dvě podskupiny. Druhou je pravděpodobnostní přístup, který využívá pravděpodobnostní rozdělení a přístupy k odhadu budoucí pozice MJ.

3.2 Náš příspěvek k řešení problému odhadu pozice uživatele

Náš přístup kombinuje deterministický postup s omezeními v pohybu, se kterými je uživatel denně konfrontován. Doposud uvedené postupy [2]-[14] předpokládají neomezené možnosti pohybu. Náš přístup naopak považuje přirozené překážky jako omezující faktor, kterým lze případné množství řešení výrazně snížit. Jako příklad lze uvést situaci, kdy uživatel jedoucí autem vydá dotaz na nejbližší benzínovou stanici. V této situaci je přirozené, že jede-li po silnici dálničního typu, není pro něj reálné využít benzínovou stanici, která je v opačném směru, přestože je v daném případě tou nejbližší.

I když lze předchozí případ chápat jako dobrou modelovou situaci, zůstává zde otázka, jak zachytit přirozené překážky (v našem případě síť silnic) způsobem přístupným výpočetní technice? Tato otázka byla úspěšně řešena v řadě publikací. V naší implementaci bychom rádi využili postup navržený pány Hage, Jensen, Pedersen, Speicys and Timko v [15], který je v současné době průmyslově nasazen v Dánsku. Systém se sestává z databáze, ve které jsou uloženy informace o silniční síti v Dánsku, přístupné i pomocí mobilních zařízení. Autoři diskutují čtyři možné způsoby reprezentace silniční sítě v databázi. Jako první je uvedena *kilometer-post* reprezentace, kde je celý systém popsán pomocí milníků

a vzdáleností mezi nimi. Tento způsob je vlastně reprezentace obecně známá z reálného světa. Druhou možností je *link-node* reprezentace založená na konceptu orientovaných a neorientovaných grafů. Uzly jsou položeny tak, aby zachycovaly důležité dopravní křižovatky a hrany reprezentují jednotlivé silnice spojující významné uzly. Přestože přesnost této metody je menší, metoda je především vhodná k plánování cest. Předposledním navrženým způsobem je *geografická reprezentace*, kde jsou zavedeny koordináty a pozice je měřena v geografických souřadnicích (například pomocí GPS). Jako poslední byla zmíněna *segmentová reprezentace*. Jedná se o kolekci segmentů, které na sebe navzájem navazují (za segment může být považována například silnice a bodem spojení je křižovatka). Segmentace umožňuje zachytit kompletní systém silnic a navíc přidat i detailnější informace jako například dopravní omezení.

Z výše uvedené implementace je pro naše potřeby nejvhodnější kombinace segmentové a geografické reprezentace, která nám umožní zachytit kompletní systém (např. silnic) s velkou přesností, která je hlavní předností geografické reprezentace.

V tuto chvíli můžeme přistoupit k bližší specifikaci uživatele. Jak je patrné ze zmíněného příkladu, je velmi vhodné vědět, zda-li bude modelováno chování chodce, motoristy nebo pasažéra na palubě letadla a to bez přílišné a obtěžující intervence ze strany uživatele. Nejprve zavedeme rozdělení uživatelů podle rychlosti pohybu. Pro naše potřeby budou postačují tři základní skupiny, i když případné rozšíření nebo naopak zúžení je možné. První skupinou budou *velmi rychle se pohybující MJ (VRMJ)*. Druhou skupinou budou *rychle se pohybující MJ (RMJ)* a poslední skupina bude tvořena *pomalými MJ (PMJ)*. Výše uvedeným skupinám lze dále přidělit druhy dopravních prostředků, neboť lze usuzovat, že do skupiny VRMJ budou patřit letadla a vysokorychlostní vlaky. Skupinu RMJ lze zobecnit na automobily a vlaky. Poslední skupina bude obsahovat pouze chodce. Podle tohoto rozdělení je možné provést další dělení podle možnosti pohybu a přirozených překážek. Je zřejmé, že například letadlo se pohybuje velmi rychle, ale obvykle v přímém směru, takže odvozování budoucí pozice je snadné vzhledem k známé předešlé trajektorii. Podobná, i když komplikovanější, situace je v případě vysokorychlostních vlaků. Na druhou stranu hustota sítě kolejí těchto vlaků je poměrně nízká. Podobně jednoduchá situace je v případě PMJ, kde je rychlost pohybu velmi malá. Přestože takoví uživatelé mají bezpochyby největší možnost změny trajektorie, je odvození budoucí pozice opět snadné, s přihlédnutím k vyžadované přesnosti, která je v tomto případě menší. Nejproblematičtější situací je skupina RMJ. Jejich rychlost je poměrně velká a navíc možnost změny trajektorie je také dostatečná. Podrobnější rozdělení na podskupiny podle rychlosti pohybu je v tomto případě vhodné. Je pravděpodobné, že vozidlo pohybující se po silnici dálničního typu poměrně velkou rychlostí, bude o to více omezeno v možnostech změnit směr jízdy, než-li pomalu se pohybující vozidlo ve městě. Výše uvedená reprezentace nám bude v těchto případech dávat další omezují podmínky na algoritmy navržené pro deterministické odvozování a počet možných řešení bude tímto klesat.

Odhad možné úspory je uveden na příkladu vozidla jedoucího po silnici dálničního typu dotazujícího se na nejbližší benzínovou stanici. V tomto případě bude dotaz nejprve položen na databázi udržující informace o segmentech dané silnice a navrácenými daty bude počet možných odboček. V kombinaci s přesnou geografickou reprezentací, je pak možné omezit data vrácená dotazem na benzínové stanice. Druhou možností je takto získané omezující podmínky připojit k dotazu a omezit počet vrácených položek výsledku.

Přestože předešlý odhad byl proveden pouze na ideální situaci je patrné, že by došlo k úspoře a to jak poslaných zpráv tak i velikosti dat vrácených dotazem. Na druhou stranu by bylo vhodné předložený koncept porovnat a ověřit v reálných podmínkách.

4 Závěr

Mobilní databáze představují rychle se vyvíjející oblast výpočetní techniky. Vzhledem k odlišnosti mobilních sítí nelze velkou část postupů navržených pro distribuované systémy použít. Jedním z hlavních rozdílů je přítomnost dat závislých na místě položení dotazu, resp. na pohybu uživatele. Ve spojitosti s tímto fenoménem je v oblasti mobilních databází věnována značná pozornost tzv. správě pozice uživatele, aby bylo možné místně závislé dotazy efektivně zodpovědět. Naším přínosem do otázky správy pozice uživatele v mobilních databázích je vylepšení stávajících řešení bez nutnosti dalekosáhlých změň v implementacích. Kombinací deterministického přístupu ke stanovení budoucí trajektorie mobilní jednotky a vhodné reprezentaci reálného prostředí pomocí segmentové a geografické reprezentace, by bylo možné v některých případech výrazně snížit množství vyslaných zpráv a zároveň omezit velikost dat vrácených dotazem. Hlavním předpokladem je možnost rozdělit MJ podle rychlosti pohybu a dále k těmto skupinám stanovit přípustné trajektorie. Omezení jsou dána především reálným prostředím. Jako příklady lze uvést silnice, cesty nebo například železnice.

Reference

1. V. Kumar, N. Prabhu, M. Dunham, Y. A. Seydim, TCON - A Timeout-Based Mobile Transaction Commitment Protocol. In Special Issue of IEEE Transaction on Computers 51 (10), 2002, 1212-1218
2. D. Plassmann, Location Management Strategies for Mobile Cellular Networks of 3rd Generation. In Proceedings on IEEE Vehicular Technology Conference, 1994
3. S. DasBit, S. Mitra, A Varying per User Profile Based Location Update Strategy for Cellular Network. In Proceedings on International Conference ICCT/WCC 2000, 2000
4. C. Rose, Minimizing the Average Cost of Paging and Registration: a Timer Based Method. In ACM/Baltzer Wireless Networks 2, 1996
5. S. Mitra, S. DasBit, A Location Management Strategy in Cellular Mobile Environment Using Distributed Searching. In Proceedings 3Gwireless'2001, 2001
6. U. Madhow, M. L. Honig, K. Steiglitz, Optimization of Wireless Resources for Personal Communications Mobility Tracking. In IEEE/ACM Transaction on Networking 3, 1995

7. I. F. Akyildiz, J. Ho, Y. B. Lin, Movement Based Location Update and Selective Paging for pcs Networks. In IEEE/ACM Transactions on Networking 4, 1996
8. R. Thomas, H. Gilbert, G. Mazioho, Influence of the Moving of the Mobile Stations on the Performance of a Radio Mobile Cellular Network. In Proceedings on third Nordic Seminar on Digital Land Mobile Radio Communication, 1988
9. H. Xie, S. Tabbane, D. Goodman, Dynamic Location Area Management and Performance Analysis. In 43rd IEEE Vehicular Technology Conference, 1993
10. S. Mitra, S. DasBit, A Cost Effective Location Update Strategy in Cellular Mobile Environment. In Proceedings on International Conference CIC, 2001
11. C. Rose, R. Yates, Minimizing the Average Cost of Paging under Delay Constraints. In ACM Journal of Wireless Networks 1, 1995
12. S. K. Das, S. K. Sen, Adaptive Location Prediction Strategies Based on a Hierarchical Network Model in Cellular Mobile Environment. In Proceedings 2nd International Mobile Computing Conference (IMCC), 1996
13. S. K. Das, S. K. Sen, A New Location Update Strategy for Cellular Networks and Its Implementation Using a Generic Algorithm. In Proceedings ACM/IEEE Conference on Mobile Computing and Networking, 1997
14. S. DasBit, P. Raha, S. Mitra, A Probabilistic Location Management Strategy in Cellular Mobile Environment. In Proceedings on IEEE TENCON, 2002
15. C.Hage, C.S.Jensen, T.B.Pedersen, L.Speicys, I.Timko, Integrated Data Management for Mobile Services in the Real World. In Proceedings of the 29th VLDB Conference, Berlin, Germany, 2003

DIDAKTIKA INFORMATIKY

Znalosti a dovednosti v přípravě informatiků (nezakusíš – nepochopíš)*

Jaroslav Král

Univerzita Karlova, Matematicko-fyzikální fakulta, Katedra softwarového inženýrství,
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
jaroslav.kral@mff.cuni.cz

Abstrakt Jedním z obtížných problémů výuky informatiky je dosažení stavu, kdy studenti nejen vyslechnou v přednášce, ale také pocítí na vlastní kůži, jaké jsou skutečné důsledky zdánlivě triviálních skutečností, jako že data mají být přesná nebo dostupná (a nejsou), že uživatel má vědět, co chce (a neví). Jedním z řešení jsou projekty, které studenti řeší od počátku (vyhledání partnerů) až k plnému nasazení v (komerční) praxi. Tento postup je vhodný pro cvičení k přednášce a může být použitelný i v diplomové práci. V tomto případě se cvičí především schopnost komunikace s uživateli. Přes různá úskalí lze dosáhnout poměrně dobrých výsledků. Pro chápání hloubky problémů kvality dat (např. dostupnost, věrohodnost, konsistentnost) je možné využít projekty vážící se na data ve státní správě. Ukážeme zkušenosti s projektem pro hodnocení kvality škol. Některé poznatky získané při řešení tohoto projektu mají širší platnost.

1 Úvod

Je všeobecně známo, že absolventi našich informatických oborů jsou velmi úspěšní na trhu práce a že mají velmi kvalitní (a to i v Evropském srovnání) přípravu na programování. Domníváme se, že jsou v průměru několikanásobně výkonnější, než je běžné ve světě. Jako programátoři jsou i dostatečně flexibilní.

Zdálo by se tedy, že je vše v pořádku a že není nutné se příliš zabývat otázkou, zda není nutné profil absolventa na univerzitách (včetně) informatických směrů studia na technikách) podstatněji měnit.

Bližší pohled však ukáže, že existují významné hrozby. Snad nejvýznamnější hrozbou je klesající zájem o informatické profese v předních zemích. Nezaměstnanost mezi informatiky je v Austrálii 16%, v Británii 18% a v USA podle regionů více než 15%. Hrozivější je skutečnost, že se IT přestává považovat za příležitost k dobré kariéře a výtahem k lepším platům. V uvedených zemích a nejen v nich klesá zájem o studium informatických oborů.

Jeden příklad. Prof. Riha z britské Kingstonské univerzity (účastník Systémové Integrace 2005) tvrdí, že během posledních dvou let poklesly počty studentů přijímaných na informatické směry studia na jeho universitě o 1/3 (ze 450

* Tento výzkum byl částečně podporován programem "Informační společnost" jako projekt 1ET100300517

na 300) a řada pedagogů musela opustit univerzitu. Je dosti rozšířený názor, že IT již v Anglii nenabízí dobré možnosti kariérního růstu a proto není výhodné informatiku studovat. Jako další možnou příčinu uvádí prof. Riha to, že mnozí studenti slyší naposledy něco z matematiky v 16 letech. Podobná situace v nezaměstnanosti informatiků je i v Austrálii (sdělení Prof. Voříška na konferenci Systémová Integrace 2004, Praha). Autor tohoto příspěvku se setkal s podobným hodnocením situace v USA v několika keynotes a v několika diskusích na konferenci IRMA 2004 v New Orleans. Se stejným hodnocením se autor setkal na konferenci SCC 2005 v Orlandu.

Není žádný důvod se domnívat, že to u nás bude jiné, i když máme jisté výhody a to nejen nižší platy programátorů u nás, znalost místní kultury a zvyků a potřebu znalosti češtiny při komunikaci s uživateli. Náš trh je příliš malý na to, aby nás čeština uživila. Problémem je, že můžeme tyto výhody ztratit (jestli je již neztrácíme) a dostat se do situace výše uvedených zemí. Smutné je, že nemáme dostatečná data umožňující přesnější hodnocení této hrozby. Mohlo by se zdát, že níže uvedená dosti extrémní stanoviska jsou za těchto okolností subjektivní. Tato námitka není, vzhledem k výše uvedeným okolnostem, oprávněná. Výše uvedené spíše všeobecné informace jsou natolik přesvědčivé, že ukazují na skutečné hrozby. Můžeme se přít o to, zda se hrozby uskuteční za rok, či za dva, či za pět let, ale můžeme stěží pochybovat, že se uskuteční.

V každém případě je nutno tento problém brát nanejvýš vážně a pokusit se o protiopatření.

2 Důvody hrozeb pro kariéru informatiků

Možnosti uplatnění počítačových odborníků jsou ohrožovány následujícími skutečnostmi:

1. Profilem osobnosti spojené s příliš úzkým zaměřením na úzce počítačové znalosti a dovednosti. To se často kombinuje s neschopností komunikovat s lidmi mimo informatiku (tento problém jsme nazvali [1] hackerským syndromem.
2. Přesunem činností do cizích zemí (outsourcing). To týká především programování takových systémů, jejichž činnost málo závisí na místní kultuře a jazyku a jimž lze konkurovat jen vyšší kvalitou prací.
3. Použitím vývojových nástrojů zvyšujících výkon a tedy snižujících potřebu klasických vývojových a provozních pracovníků.
4. Přebíráním kustomizovatelných systémů jako jsou produkty firem SAP nebo ORACLE.
5. Nedostatečný všeobecný základ znalosti (matematiky) a nedostatečný výcvik dovedností včetně schopnosti systematické tvrdé práce.
6. Neschopnost chápat takové skutečnosti, jako je nutnost spolupráce s neinformatiky, taktika a strategie vývoje softwaru, chápání potřeb, znalostí a jazyka uživatelů, úcta k uživatelům a chápání důsledků (ne)kvality dat pro formulaci požadavků a cest řešení. K tomu je nutné – avšak obtížné – řešit takové projekty, které umožní některé z uvedených problémů prožít a nejen slyšet a povrchně chápat.

Hlavní ohrožení je to, že produkce informatiků je velmi vysoká v důsledku dřívějšího rychlého růstu informatických technologií. Kapacity výuky informatiků jsou tomu přizpůsobeny, a hrozí, že budou nadměrné, až potřeba informatiků přestane růst. Zároveň si je třeba uvědomit, že na globální trh nově vstupuje obrovské množství talentů z Číny (viz konference SCC 2005).

3 Vývoj a provoz

V dalším se budeme převážně věnovat problematice přípravy informatiků-vývojářů (např. programátorů). Nebudeme dále uvažovat přípravu provozářů (správci sítě, správci dat, ...). Příprava vývojářů má oproti přípravě provozářů jistá specifika daná nutností velmi úzké spolupráce s uživateli včetně zvládnutí základů jejich znalostního oboru. Dalším problémem je nutnost zahrnout do specifikace požadavků aspekt kvality dat a otázky taktiky strategie zavádění systému.

Tyto problémy nehrají podstatnou roli při provozování systému (a tedy při přípravě provozářů), alespoň prozatím. V delší perspektivě jsou ohroženi i oni. Jsou klíčovým aspektem přípravy vývojářů – alespoň těch, kteří se účastní úvodních fází vývoje.

V čem je problém? Ukazuje se, že úspěch vývoje závisí na mnoha zdánlivých maličkostech a zdánlivě samozřejmých a tedy triviálních zásadách.

Je velmi obtížné přesvědčit studenty, že problém je právě v těchto pravidlech. Zkušenost ukazuje, že i podrobný výklad doložený zkušenostmi z praktických projektů studenty většinou nepřesvědčí. Není výjimkou, že absolventi tyto zkušenosti docení, až když je opomenou ve svém zaměstnání a obtížně napravují následky. Slyšel jsem názor absolventa z praxe, že některá doporučení (např. získat jako spojence tzv. časné uživatele a začít systém zavádět pomocí těchto spojenců a nikoliv pomocí počítačových fandů) měla pro ně klíčový význam. Nikdy ale před tím nevěřil, že je tomu tak. Na přednáškách studenti mnohým věcem uvěřili až po té, co prošli ohněm praktického projektu. A někteří to přijali až po té, co se s tím setkali v praxi. Nezkusíš-li, neuvěříš.

4 Profil a kvalita studia

Výše uvedené skutečnosti závisí na profilu a kvalitě zvládnutí znalostí a dovedností absolventy. Absolventi by měli být schopni spolupracovat s uživateli při vývoji a provozu softwarových systémů. To vyžaduje, aby profil studia podporoval schopnosti a znalosti usnadňující spolupráci, a, což je zvláště obtížné, alespoň zčásti umožnil výcvik potřebných dovedností. To však není jediná hrozba.

Kvalita znalostí získaných na vysoké škole závisí nejen na tom, co poskytne vysoká škola, ale je do značné míry determinována tím, jak jsou připraveni studenti na nižších stupních škol. Jsou důvody k tvrzení, že se kvalita výuky na středních školách snižuje. Informatika je ohrožena i z tohoto důvodu dvakrát. Špatná kvalita vzdělání na středních školách nás fakticky nutí snižovat kvalitu výuky tím, že musíme doplňovat to, co již studenti měli umět (a někdy to v podstatě ani nelze z důvodů časových, a někdy, jako v případě dovedností,

principiálních). Poněvadž studium informatiky (i jiných technických oborů) je relativně náročné, hrozí snížení zájmu o studium informatiky (technické obory obecně) i z tohoto důvodu.

Dalším negativním faktorem se může stát školné. Bude-li vztaženo k nákladům na studium, bude v informatice vyšší než u ekonomie, psychologie, sociologie, práv a jiných populárních oborů. Důvodem vyšších nákladů je potřeba rychlé obměny podpůrných prostředků, které v případě informatiky zastarávají velmi rychle. Níže uvidíme, že bychom mohli vliv školného měřit na existujících datech, že to ale z řady důvodů nelze uskutečnit. Svůj díl viny na tom nesou i informatici, že se dostatečně nesnaží situaci změnit.

Zdánlivě nejslabším negativním vlivem může být i to, že informatikům velmi rychle zastarávají jejich znalosti a dovednosti a informatici se proto musí stále velmi mnoho učit. Na to ale stále více berou ohled studenti při výběru své budoucí profese.

5 Uživatelé, profil studia a kvalita znalostí

Schopnost spolupráce s uživateli se stává rozhodující v mnoha informatických profesích a je podmínkou použití moderních způsobů vývoje např. agilních forem vývoje. Nutnou (avšak nikoliv postačující) podmínkou je přesvědčit studenty, že tomu tak opravdu je, a doplnit do kurikula takové předměty, které komunikaci s uživateli usnadní.

Vzhledem k různorodosti oborů aplikace je nutno zařadit takové přednášky, které usnadní zvládání různorodých znalostních oborů. Vhodným řešením pro softwarové odborníky je zvýšení důrazu na použití matematické statistiky a zařazení přednášek prezentujících způsob myšlení ekonomických (finance, finanční matematika) nebo experimentálních věd (fyzika, farmakologie, některá odvětví biologie). Podrobnosti lze nalézt v [2]. V tomto směru mají výhodu studenti technik a ekonomických univerzit. Výhodou informatiků z klasických univerzit (tam, kde je obor zcela informatický) je vcelku dobrá úroveň matematických znalostí a velmi kvalitní příprava v programování.

Na VŠE v Praze existuje tendence k víceoborovému studiu. Řada studentů MFF UK v Praze tento problém pociťuje a řeší jej tím, že absolvují vedle MFF UK např. VŠE nebo dokonce medicínu. Jedná se ale spíš o výjimky.

Dvojoborovost je náročná. Vyžaduje talent, vůli pracovat a dobré základy z nižších stupňů škol. Kvalita středoškoláků se ale spíše zhoršuje. To může být zásadní problém při realizaci kvalitního dvojoborového studia. Nároky na studenty mohou být příliš vysoké, studijní neúspěchy běžné a to následně sníží zájem o takové studium.

Tento problém nemusí být základní, jak lze ověřit na příkladu dvojoborového studia na učitelských oborech, kde studenti musí zvládnout základy dvou oborů (svých budoucích aprobací) i to, jak tyto poznatky předávat dál. Pro své komunikační schopnosti jsou absolventi učitelství informatiky někdy žádanější, než klasičtí informatici, i když jejich informatické dovednosti mohou být výrazně nižší.

6 Studentské projekty a dovednost spolupráce s uživateli

Spolupráci s uživateli je třeba nacvičit. Z podstaty věci vyplývá, že to není možné uvnitř školy – uživatelé jsou zpravidla (a měli by být) lidé mimo školu. To je značná komplikace a vyžaduje to některé ne zcela obvyklé postupy.

Jsou možné následující postupy (většina z nich již byla ověřena při výuce):

1. Malá skupina studentů (3–5) dostane za úkol sama v praxi najít téma na projekt. Pro tento projekt se ve spolupráci s uživateli realizuje celý vývojový cyklus od vyhledání "zákazníka" k předání systému a potvrzení "zákazníka", že systém převzal a případně za něj zaplatil. Zaplacení za projekt se považuje za jednoznačný úspěch, není to ale samozřejmě podmínkou.
   Projekty s vyhledáním partnera jsou velmi obtížné a mohou selhat. Proto jsou přípustné i projekty, které nezahrnují vyhledání partnera (zadavatele) v praxi, vizi řešení a specifikaci základních požadavků (je dodána zadavatelem a upřesňována během řešení). Studenti ale musí upřesňovat zadání a udělat návrh.
   Jednotlivé týmy si vytvářejí domovské stránky svých projektů, přes něž jsou dostupné všechny relevantní dokumenty a průběh *e*-konference. Týmy pravidelně referují o postupu prací. Zkušenost ukazuje, že ve skupinách do 30 lidí může být takový postup úspěšný a plní hlavní účel (komunikace s uživateli). Cvičení je rozděleno do dvou semestrů. V prvém semestru musí řešitelé vytvořit domovskou stránku projektu a zrealizovat návrh systému (většinou s použitím nástroje založeném na UML). Návrh je třeba obhájit s použitím standardních technik presentace. V druhém semestru se systém oživí a předá. Z hlediska právního purismu existují některá slabá místa uvedeného řešení. Zkušenosti z výuky ukazují, že jen asi 5–15 % studentů je schopno a ochotno spolupracovat s uživateli a vést projekty. Pro úspěch projektu bývá nutné, aby vedoucího projektu a jeho zástupce jmenoval po případné dohodě s členy týmu přednášející.
2. Diplomové práce vázané na reálné projekty (např. modernizace informačního systému *e*-obchodu). Výhodou takových diplomových prací je to, že se v nich mohou řešit úlohy většího rozsahu, než je obvyklé u "klasických" prací. Ve zmíněném příkladu *e*-obchodu se jedná o zkušenost se zavedením nového systému za existence systému starého. To je výhoda natolik významná, že vyvažuje níže uvedené nevýhody. S diplomovými pracemi je kupodivu více starostí než s týmovými projekty uvedenými výše. Uveďme hlavní problémy s diplomovými pracemi uvedeného typu.

– Praktický úkol, který je součástí práce, je natolik náročný, že diplomant nemá dost času na vypracování textu diplomové práce. To bývá kombinováno s tím, že se klade malý důraz na klasické požadavky, jako je rešerše možných řešení, důvody zvoleného postupu a případné nevýhody přijatého řešení. To bývá důsledek apriorních názorů uživatele. Často chybí závěry. Poznamenejme, že toto je jeden ze sporných zvyků současné praxe.
– Mohou být problémy se zveřejňováním programů vzniklých v rámci práce a také s předvedením systému.

– Je trochu obtížnější kontrolovat, zda to, co diplomant deklaruje jako svůj výsledek, je skutečně jeho dílem.

Domníváme se, že diplomové práce tohoto typu by měly být více podporovány. Podporovány by měly být i týmové diplomové práce. S použitím servisní orientace lze řešit problém ověření toho, co diplomant skutečně udělal. Další problém je to, že s takovými diplomovými pracemi nejsou zkušenosti, a proto se mnozí učitelé domnívají, že by se takové diplomové práce neměly zadávat.

Existují i potenciální legislativní problémy, které zatím naštěstí není nutné řešit. Není např. dořešeno, zda škola má k vytvořenému produktu jistá práva.

7 Studentské projekty a kvalita dat

Kvalita dat (KD) je stále významnějším aspektem informačních systémů. KD je multidimenzionální (víceaspektový koncept) závislý podle ISO 8124 na účelu, k němuž se data používají. Z uvedeného plyne, že existuje řada aspektů kvality dat. Zatím není shoda v tom, které aspekty kvality dat jsou nejdůležitější. Nejčastěji se vyhodnocují následující aspekty (též www.iqconference.org, nebo mitiq.mit.edu) nebo Haluk Kasnakoglu a Robert Mayo v příspěvku [3].

Relevantnost (Relevance) – míra, do jaké míry data splňují účel, pro který jsou používána.
Přesnost (Accuracy) – jak přesná jsou používaná data (měřeno obvykle statistickými charakteristikami pro chybu, např. směrodatná odchylka).
Včasnost (Timeliness) – za jakou dobu lze data aktualizovat.
Dostupnost (Accessibility) – jak jsou již existující data dostupná. To je problém především ve státní správě.
Porovnatelnost (Comparablity) – metrika hodnotící možnost porovnávat, ale také spojovat data z různých zdrojů.
Koherence (Coherence) – metrika vyjadřuje, do jaké míry byla data vytvořena podle z hlediska výsledku stejných pravidel.

Britský statistický úřad k těmto metrikám doplňuje metriku:

Úplnost (Completeness) – metrika udávající jaká část potenciálních dat je zachycena v databázi (viz [4]), případně zda statistické charakteristiky dat nejsou ovlivněny výběrovými efekty.

Autoři navrhují postup, jak z těchto metrik odvodit číselnou metriku celkové kvality dat.

Kvalitu dat je nutné nejen hodnotit, ale je třeba uplatňovat postupy hodnocení kvality dat jako je vylučování okrajových hodnot, doplňování chybějících dat, atd. Zde je problém studenty přesvědčit, jak je důležité hodnocení kvality dat statistickými metodami.

Problém s KD je v tom, že pokud student neprožije důsledky nedostatečné kvality dat na způsob řešení, neuvědomí si důležitost hodnocení kvality dat a není připraven správně na problémy s kvalitou dat reagovat.

Kategorie	Dimenze
Vnitřní, intrinsická (Intrinsic)	Přesnost (Accuracy) Objektivnost (Objectivity) Důvěryhodnost (Believability) Reputace (Reputation)
Dostupnost (Accessibility)	Dostupnost (Accessibility, též Availability) Bezpečnost přístupu (Access security)
Kontextuální (Contextual)	Relevantnost (Relevancy) Přínos (Value added) Včasnost (Timeliness) Úplnost (Completeness) Rozsah (Amount of data)
Reprezentační (Representational)	Interoperabilita (Interoperability) Srozumitelnost (Easy of Understanding) Výstižná a stručná reprezentace (Concise representation) Konsistentní reprezentace (Consistent representation)

Tab. 1. Metriky podle [5]. Podrobný popis významu jednotlivých metrik lze nalézt v [6].

Příkladem je použitelnost metody kritické cesty (CPM). Pokud nelze zanedbat fakt, že data (odhady dob provedení činností) jsou nepřesná (mají chybu větší než zcela malou), je nutno použít metodu kritického řetězce [7] založenou na specifických postupech nepoužitelných v CPM. Při přednášce je možné studenty seznámit s principiálními důsledky kvality dat. Je ale velmi obtížné je přesvědčit o tom, že je nutné tímto způsobem reagovat, a o tom, jak to může být obtížné.

Pro uvědomění si celé hrůzy nekvality dat a také pro seznámení se s tím, jaký horor je práce pro státní správu, jsme se studenty řešili úlohu, jak hodnotit kvalitu středních škol s možností rozšíření na školy vysoké [8].

Výsledky tohoto projektu jsou zajímavé i obecně. Minimálním úkolem byla realizace jednoduchého portálu usnadňujícího nalezení škol a existujících informací o nich, tj. stránek obsahujících informace o dané škole (např. výsledky inspekcí). Zároveň se provedla diskuse, která informace o škole je nejvhodnější. Došlo se k závěru, že by to mohly být následující ukazatelé:

- průměrná velikost příjmu absolventa (pro daný obor, danou školu, atd.) jistou dobu po absolutoriu;
- úspěšnost na VŠ (přijímačky, bakaláři, magistři);
- míra nezaměstnanosti mezi absolventy (i po více letech po absolutoriu).

Podařilo se ověřit, že potřebná data pro tyto ukazatele existují, ale že jsou nepoužitelná. Důvodem kupodivu nebylo hlavně to, že data nebyla úplná. Malé problémy byly i s formátem a konsistencí dat. Hlavní problém je v dosažitelnosti dat v kombinaci s tím, že potřebná data jsou nesmyslně rozptýlena na více

než pěti serverech a že jsou v nich naprosto nepochopitelné koncepční závady (problém je s atributem kvality zvaným v anglické literatuře *accessibility* – dostupnost). Příklady:

- údaje o nezaměstnanosti se zveřejňují jen pro čerstvé absolventy škol a ne pro dřívějších absolventy. Potřebná data jsou na pracovních úřadech k dispozici, nelze je však zveřejňovat.
- VŠ nezveřejňují informace o svých studentech (pravděpodobně z důvodu ochrany osobních údajů) – s výjimkou toho, kdo byl přijat. Nikoliv však o tom, kdo absolvoval. Informace o přijatých je zatížena tím, že ne všichni, kteří uspěli u přijímacích zkoušek skutečně do dané školy nastoupí (příslušná data jsou fakticky zatížena skrytými duplicitami).
- Formáty dokumentů o školách (např. inspekčních zpráv) jsou velmi volné a neumožňují automatizované zpracování. Může to být důsledkem toho, že se problematikou škol zabývá řada institucí, které ale nespolupracují.

Podle dostupných informací existují data umožňující výpočet přiměřené výšky příjmu v závislosti na volitelné množině atributů: obor, škola, pohlaví a rok absolvování. Je ale asi nutné kombinovat data z penzijního systému, pracovních úřadů, UIV (Ústav informací ve vzdělávání), MŠMT (Ministerstvo školství, mládeže a tělovýchovy), MPSV (Ministerstvo práce a sociálních věcí), finančních úřadů a dalších zdrojů. To však není možné z důvodu ochrany osobních dat. Instituce se nedohodly jak přejít na klíče bez interního významu. Pro studenty nebyly tyto skutečnosti překvapením. Velikost z toho plynoucích problémů však překvapením byla. Je typické, že studenti dovedli navrhnout postupy anonymizace klíčů, nebyli však ochotni zjistit, proč se anonymizace klíčů ve státní správě neprovádí a jak by se měl případně tento problém řešit.

8 Poučení nejen pro studenty

S využitím dat z různých zdrojů je možné řešit i takové otázky, jak závisí velikost příjmu na tom, kolik hodin matematiky bylo "naordinováno" během studia (případně na jakém typu škol a v jakém oboru). Tato informace by mohla být argumentem proti rozhodnutí senát a spol. nezařadit matematiku do povinných maturit. Veřejnost – a pedagogové zvláště – by měla mít přístup k informacím o vlivu výuky matematiky a jiných předmětů na kvalitu pracovní síly a kariérní možnosti. O tom, že nedostupnost těchto údajů je chyba, málokdo soudný pochybuje. Například na konferenci Systémová integrace 2005 si na nedostatek matematického vzdělání stěžovali zástupci IT firem a dávali to do souvislosti s tím, že informatiků je na trhu práce dost, ale dobrých informatiků málo. To je výtka i pro učitele informatiky. Jednou z našich chyb je to, že nejsme schopni veřejnosti vysvětlit a doložit, že:

- matematika je důležitým předpokladem úspěchu u přijímacích zkoušek na mnoha VŠ;

– matematika jako prakticky jediný předmět (snad s výjimkou informatiky) cvičí na středních školách nejen znalosti, ale i dovednosti, schopnost dodržovat stanovený postup i to, že někdy prostě nelze nahradit dovednost záplavou slov;
– znalost matematiky zvyšuje šanci na pracovním trhu.

Při existujících datech by neměl být problém sledovat trendy zaměstnanosti a při spolupráci se zaměstnavateli i trendy do budoucnosti.

Je neomluvitelné, že VŠ nezveřejňují data o úspěšnosti svých studentů a o tom, kdo absolvuje, tak, aby data umožňovala hodnotit úspěšnost nejen vysokých škol samých, ale také škol středních.

Je neomluvitelné, že se neověřují důsledky zavedení školného (soukromé školy je mají již dnes) na kariérní úspěšnost absolventů a rozhodnutí se ponechává na předsudcích a lidové tvořivosti politiků.

Výše uvedený studentský projekt ukázal, kde je hlavní chyba. Je neomluvitelné, že ÚOOÚ považuje za svůj úkol zakazovat a nestará se vůbec o to, aby draze sebraná data mohla sloužit pro řízení státu a pro řízení podniků, či pro rozhodování občanů a nemizela jak v černé díře.

Hlavním závěrem je, že se data sbírají, ale neanonymizují pro obecné použití s minimální ztrátou informace. Důvodem může být to, že je dosti takových školských i jiných institucí, které nemají zájem na tom, aby se sledovala úspěšnost absolventů jednotlivých škol. Skutečnost však může být ještě horší – může to být neschopnost maskující se jako neochota.

Všude se mluví o vzdělání a podpoře vzdělanosti. Do školství směřuje ke sto miliardám. Budu-li se ale jako otec starat, kam dát dítě, aby se v životě dobře uplatnilo, musím se dnes spoléhat na pověsti, drby a málo spolehlivé ukazatele. Vláda není ochotna (a opozice je podle skutků s ní zajedno) investovat pár milionů do systému, který by umožnil zvýšit objektivnost hodnocení škol a provádět efektivní školskou politiku.

9 Závěr

Informatici by se měli zlepšit situaci v hodnocení škol a obecně ve využití dat státní správy. Situace na trhu práce se pro informatiky rychle mění. Nikoliv k lepšímu nebo snazšímu. Vstupujeme do globální konkurence. Stačí navštívit nějakou prestižní konferenci a vidět tam talentované účastníky z Číny, abychom si uvědomili, co globalizace znamená. Nevyhneme se víceoborovosti studia a důrazu na počáteční etapy vývoje softwaru. Pokud se kvalita středních škol dále zhorší, nebudou studenti volit informatiku, poněvadž tato se jim bude jevit jako obtížný, ale ne příliš dobře placený obor. A ti, kteří se přihlásí, nebudou (vzhledem k nedostatku obecných základů z nižších stupňů škol) schopni dosáhnout kvalitnějších výsledků. To se projeví zhoršením výsledků informatiků v praxi, snížením jejich kariérních možností a tedy i dalším poklesem zájmu o studium IT. Kruh se uzavře. Situace s daty ve státní správě a nejen tam je další hrozbou pro informatiku, neboť je bariérou pro to, aby byly využity možnosti a potenciály

informatiky. I to je hrozbou pro informatiku jako celek. A nejen pro informatiku – nedokážeme-li využít informace, které můžeme získat z existujících dat, dostáváme se do konkurenční nevýhody proti těm, kteří svá data dokáží náležitě využít. Z tohoto hlediska je současná politika ÚOOÚ nezodpovědná. V malém to vidíme na příkladu hodnocení škol.

Dovětek

Projektu se zúčastnili i slovenští studenti. Ti zjistili, že ač se situace na Slovensku v jednotlivostech liší, zásadní bariéra nedostupnosti dat je stejná jako v České republice.

Reference

1. Král J., Žemlička M., Mathematical Statistics in SW Engineering Education. In Khosrow-Pour M., ed.: Information Technology and Organizations: Trends, Issues, Challenges and Solution, Hershey, PA, USA, Idea Group Publishing, 2003, 889–890
2. Král J., Žemlička M., Koubková A., Opomíjené aspekty profilu absolventů informatických oborů. In Vojtáš, P., ed.: ITAT 2004 Information Technologies – Applications and Theory, Košice, Prírodovecká fakulta Univerzity Pavla Jozefa Šafárika v Košiciach, 2005, 149–163
3. Kasnakoglu H., Mayo R., FAO Statistical Data Quality Framework: A Multi-Layered Approach to Monitoring and Assessment, 2004
`http://unstats.un.org/unsd/ accsub/2004docs-CDQIO/1-FAO.pdf`
4. Haworth M.F., Martin J., Delivering and Measuring Data Quality in UK National Statistics, 2001, `http://www.fcsm.gov/01papers/Haworth.pdf`
5. Wang R.Y., Ziad M., Lee Y.W., Data Quality. Volume 23 of The Kluwer International Series on Advances in Database Systems, Springer, 2001
6. Pipino L., Lee Y.W., Wang R.Y., Data Quality Assessment. Communications of the ACM 45, 2002, 211–218
7. Goldratt E.M., Critical Chain. North River Press, Great Barrington, MA, 1997
8. Bajana D., Borovcová A., Čurn J., Fojta R., Štefan Ignáth, Malohlava M., Rudišin M., Šafrata P., Šrajer M., Šulc Z., Zelenay P., Výstupy projektu "Hodnocení kvality škol", 2005

Studijní materiály s využitím DocBook stylu pro souběžnou českou a anglickou výuku C#

Petr Šaloun

Katedra informatiky FEI VŠB-TU Ostrava, 17. listopadu 15, 708 33 Ostrava-Poruba
Petr.Saloun@vsb.cz

Abstrakt Pro současnou tvorbu studijních materiálů odpovídající požadavkům e-vzdělávání v jednom autorském dokumentu byla realizována rozšíření DocBook stylu vytvořeného nad XML. Vytvořené styly jsou připraveny pro výstup do formátů HTML, XHTML, PostScript a PDF. Nově byly vytvořeny XSL styly, které odpovídají provedeným změnám v DTD souborech.

1 Úvod

Řada předmětů nejen v oblasti informačních technologií je studentům nabízena souběžně v českém a anglickém jazyce. S tím souvisí potřeba přípravy studijních materiálů, které vychází z jednotných osnov předmětu. Tak byly vytvořeny kompletní studijní texty pro výuku programovacího jazyka C#. Texty představují slajdy přednášek, studijní texty, multimediální rozšíření a materiály tvořící náplň cvičení včetně vzorových komentovaných příkladů.

Studijní texty pro C# vytvořit tým řešitelů během druhé poloviny roku 2004. Tým si práci rozdělil běžným způsobem – zvlášť byly připravovány přednášky a cvičení, návaznosti byly řešeny úvodní koordinací osnov. Na základě osnov byl vytvořen výklad dané látky. Jeho zjednodušení a překlad do angličtiny se staly východiskem k prezentacím přednášek. Tak se podařilo připravit studijní materiály česky a anglicky. Údržba autorských souborů, využívajících styl DocBook, z nichž jsou cílové materiály generovány, je ovšem náročná. Souběžně pracující jiný tým diplomantů pracoval na úpravách stylů DocBook pro potřeby e-vzdělávání, studenti dokončili a obhájili své práce [1], [2], [3], [4] v roce 2005.

Práce vznikla jako výsledek projektů FRVŠ 2004/341 *Multimediální studijní opory pro předměty programování v jazyce C/C++ a C#*, transformačního a rozvojového projektu MŠMT pro VŠB-TU Ostrava 2005/542 *Navazující Mgr. studium Informačních technologií kvalitněji a paralelně anglicky II* a projektu FRVŠ 411/2005 "Elektronické publikování – rozšíření předmětu".

2 Použité technologie

Komplexní podporou pro přípravu multimediálních studijních opor je široké téma, které jsme řešili a řešíme na katedře již od 90. let. Vycházíme ze skutečnosti,

že základem všech výukových materiálů je obsahově kvalitní text připravený ve vhodném stylu a založený na technologiích, které jsou perspektivní a stabilní. Takový materiál lze pak bez větších problémů možno vložit do požadovaného LMS, nebo prezentovat samostatně.

Při přípravě materiálů byl použit obecný přístup využitelný pro další výukové texty. Jako základ byl použit DocBook styl vytvořený nad XML a jeho rozšíření pro potřeby přípravy studijních opor pro e-vzdělávání. Nově byly vytvořeny XSL styly, které odpovídají provedené změny v DTD souborech. Pro multimediální prvky byla přidána podpora animací Macromedia Flash a Java appletů. Pro zjednodušení práce budoucích uživatelů nově upravených stylů byla začleněna podpora technologie XInclude, která umožňuje autorům dělit jednotlivé zdrojové XML soubory na menší logické části se zachováním podpory validace. V DocBooku prozatím neexistovala možnost mít pro sazbu výpisy kódů s automatickým zvýrazněním syntaxe. Tato možnost byla implementovaná na úrovni stylů a zautomatizované zvýraznění bylo provedeno pomocí jazyka ELisp.

Implementace podpory tvorby prezentací je založena na ověřeném projektu DocBook Slides. Zvoleny byly dva postupy. Jeden spočívá ve tvorbě prezentací od začátku a druhý v určení vhodných částí stávajících opor k vytvoření prezentace. Tím je umožněna současná tvorba výukových textů a slajdů přednášek. Údržba jednoho zdrojového dokumentu, z nějž se podle potřeby vygeneruje text nebo prezentace, je mnohem snadnější, než udržování konzistence mezi různými zdroji při jejich tvorbě nebo opravách. Vytvořené styly jsou připraveny pro výstup do formátů HTML, XHTML, PostScript a PDF.

3 Úpravy DocBooku

Základní myšlenkou je vytvoření nových stylů. Většinou se vlastní transformační pravidla nevytvářejí úplně od začátku, ale využívají se již hotové transformační styly. Na jejich základě se provedou rozšíření, anebo úprava stávajících stylů.

1. **úprava stylů i DTD souborů** – náročné na neustálou synchronizaci změn. Může se ale vždy ověřit *validita* dokumentu, tzn. splňuje předepsané vlastnosti popsané v DTD.
2. **úprava stylů bez úpravy DTD souborů** — neklade žádné nároky na synchronizaci změn. Avšak již se nemůže **nikdy** ověřit validita dokumentu.

3.1 Úpravy DTD

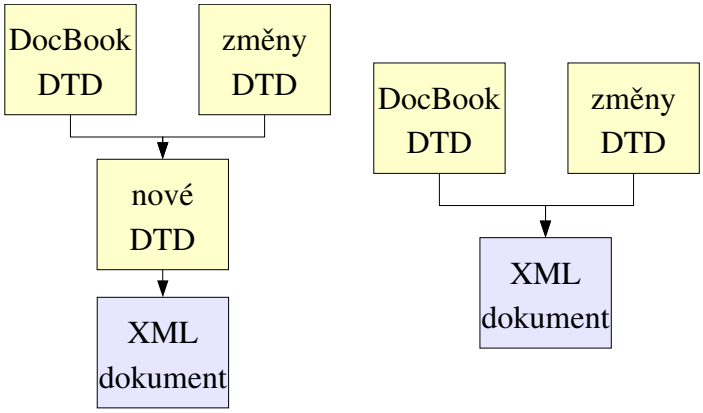
Oproti standardu technologie DocBook vznikly zejména tyto úpravy stylu:

- ikony s informačními oblastmi,
- podpora animací ve formátu Flash,
- podpora Java appletů,
- zvýraznění syntaxe programovacích jazyků,
- tvorba prezentací na základě XML dokumentů se studijní oporou,
- možnost rozdělení kapitol, sekcí a vkládaných zdrojových kódů do jednotlivých souborů pomocí technologie *XInclude*.

DocBook má velmi dobře napsané DTD soubory a je možné v nich provádět modifikace, které mohou být provedeny třemi způsoby.

1. přímo,
2. nové DTD,
3. rozšiřující vrstva *(customization layer)*.

Nejlepším způsobem se ukázalo využití *rozšiřující vrstvy*, která umožňuje přechod na nové verze DocBooku bez ztráty provedených změn. Ke spojení změn s DocBookem dochází na úrovni psaného XML dokumentu.



Obr. 1. Vkládání DTD souborů do XML dokumentu.

Při novém rozšíření vznikla dvě nová DTD. Jedno je pro psaní opor pomocí DocBooku a druhé je pro úpravu prezentací, viz obrázek 1. Pro elementy, které byly přidány pro výběr elementů vhodných pro tvorbu prezentace na základě XML dokumentu s oporou, DTD soubor neexistuje. Je to dáno tím, že tyto elementy se mohou objevit na jakémkoli místě dokumentu a popsání vzájemných možných vztahů není prakticky uskutečnitelné.

3.2 XInclude

Společně s technologií XML vzniká i specifikace technologie XInclude, viz [8], a XPointer, viz [9]. Oboje je nepřímo využitelné v DocBooku. Standardní DocBook neobsahuje elementy pro vkládání dokumentů. Ponechává se na programátorovi, jaký přístup k modularizaci zvolí. Nejznámější je vkládání pomocí entit. To však nese jistá omezení:

1. není možné mít v dokumentu specifikaci DTD,
2. dokument nebude nikdy validní.

Jednotlivé dokumenty, které se vkládají pomocí XInclude, mohou obsahovat další dokumenty vkládané technologií XInclude. Zanořování umožňuje vytvářet nekonečné hierarchie dokumentů, které mohou být kdykoliv použity při psaní jiných dokumentů.

3.3 Propojení s DocBookem

Největším problémem bylo nalezení vhodných míst ve struktuře DocBooku pro nově vytvořené prvky. Ne všechny nové elementy jsou použitelné jako samostatný celek, a proto byly již zařazeny do jiných nově vytvořených prvků. Všechny nejvrchnější prvky se musí napojit na strukturu dokumentů DocBooku. Na to tvůrci DocBook DTD pamatovali a zajistili řešení za použití entit.

3.4 DTD používané s DocBookem

Kromě standardních DTD souborů jsou s technologií DocBook spjata další DTD.

Simplified DocBook – nejznámější je asi *zjednodušený DocBook*, nad ním se obvykle staví další technologie. Spravuje ho stejná organizace jako normální DocBook a jedná se o podmnožinu klasického DocBooku. Současná verze 1.1 je založena na DocBooku 4.3.

DocBook Slides – podpora tvorby prezentací (DTD je využito i v našich stylech), která je založena na zjednodušeném DocBooku. Společně s DTD jsou šířeny i XSL styly pro výsledné transformace.

MathML – často používané ve spojení s DocBookem, udržuje konsorcium W3. Cílem je umožnit zápis matematických výrazů. Problémem zůstává ruční zápis a výstup do PDF. XSL-FO procesory dnes tento druh zápisu vzorců často nezvládají. V současnosti jsou k dispozici nástroje umožňující konvertování TEXového zápisu na MathML.

SVG – používá se pro zápis vektorové grafiky pomocí XML. V projektu bylo SVG využito pro ikony zvýrazňujících oblastí při výstupu do PDF. SVG DTD spravuje konsorcium W3.

3.5 Úpravy XSL

Pro přidání vlastních elementů do DocBooku byly nejprve vytvořeny příslušné DTD, ve kterých byla stanovena výsledná struktura XML dokumentu. V tu chvíli již byl známý logický význam všech prvků. Během celkové modifikace byly styly neustále při vývoji testovány na souběžně vznikajících oporách včetně autorovy.

Základním stavebním kamenem se staly styly od Normana Walshe, které jsou dobře udržované. Jejich použitím se dá dosáhnout velmi kvalitního rozvržení objektů při výstupu. Pro modifikaci XSL jsou k dispozici tři možnosti:

1. změny parametrů u XSLT procesoru,
2. přímo,
3. rozšiřující vrstva.

K modifikaci byla využita poslední metoda, tedy rozšiřující vrstva, která je z hlediska jazyka plnohodnotným XSL souborem. Originální styly se přitom připojují na úrovni rozšiřující vrstvy.

4 Prezentace a jejich příprava

Pro tvorbu prezentací existují styly Normana Walshe. Projekt se nazývá DocBook Slides a skládá se z DTD souborů a XSL stylů. DTD je postaveno nad *zjednodušeným DocBookem*, viz [6]. To zaručuje velmi jednoduché použití uživatelem, protože je k dispozici pouze omezená skupina elementů DocBooku. Ke stylům Normana Walshe musely být přidány rozšiřující XSL vrstvy k uspokojení požadavků e-vzdělávání. Všechny nově přidané elementy byly umístěny do modifikačního DTD souboru, aby byla možná validace zdrojových XML souborů prezentací.

Další možný a podporovaný přístup je začlenění tvorby prezentací již na úroveň XML dokumentů se studijní oporou. Pro tento přístup bylo nezbytné napsat nové pomocné styly.

Pro tvorbu prezentací od začátku bylo vytvořeno DTD, které odráží změny přidané oproti standardnímu DocBook Slides. DTD bylo vytvořeno pomocí nového souboru, který slučuje rozšiřující vrstvu s originálním DTD, viz levá strana obrázku 1. Toto DTD může být aplikováno i na XML soubor popisující prezentaci, který vznikl z opory. Spojení s originálním DTD bylo realizováno pomocí entit. Byly začleněny podpory pro vybrané zvýrazněné oblasti, podpory pro flashové animace, Java applety a výpis kódu. Prezentace je možné vytvářet do HTML, XHTML a XSL-FO formátů. Pro HTML i XHTML formát je k dipozici několik na sobě nezávislých transformačních stylů, které ovlivňují výsledný vzhled celé prezentace na úrovni generovaného kódu.

1. **rámce** *(frames)* – vlevo je menu, vpravo snímky,
2. **tabulka** – levý sloupec obsahuje menu, pravý sloupec snímky,
3. **čisté umístění** *(plain)* – prezentace je bez menu.

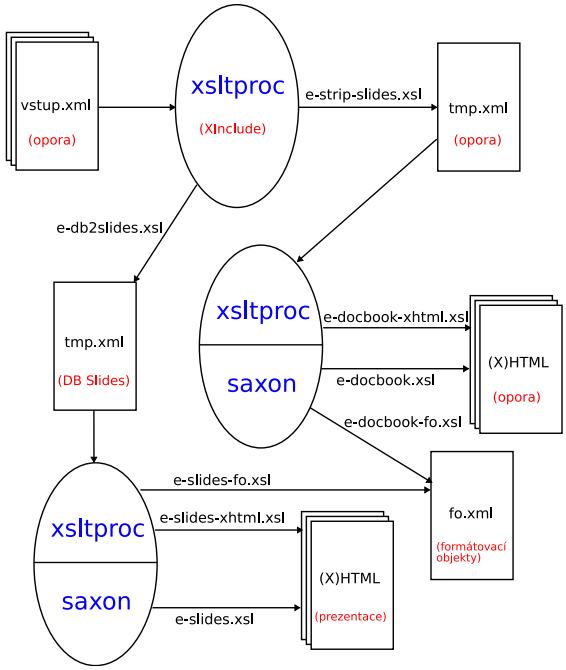
Všechny formáty obsahují navigační prvky, kterými lze posouvat prezentace po jednotlivých snímcích dopředu nebo dozadu. Dále je k dispozici odkaz na obsah celé prezentace. Na obrázku 2 je znázorněno použití jednotlivých stylů při překladu včetně použitých XSLT procesorů.

5 Návrh a tvorba grafických prvků

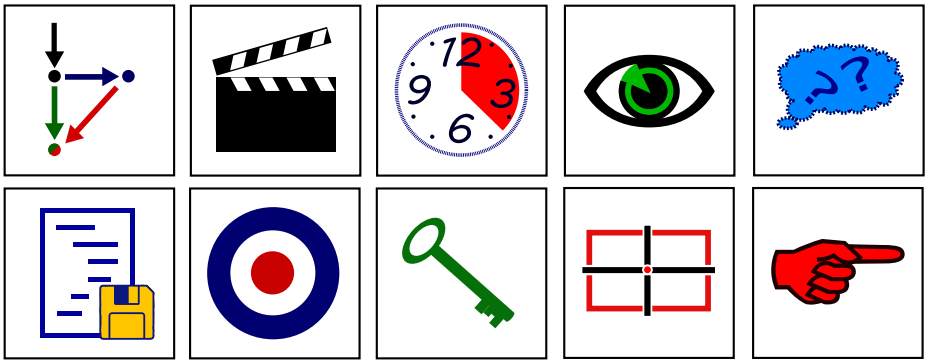
Každý kurz by měl obsahovat graficky názorné prvky, které usnadňují navigaci a porozumění textu. K těmto grafickým prvkům je vhodné připojit piktogramy, které urychlí orientaci v materiálu.

Několik piktogramů bylo vytvořeno v rámci minulých diplomových prací [5]. V rámci [1], [2], [3] byly vytvořeny nové grafické prvky včetně piktogramů.

Po definování základních druhů ikon se řešil problém jejich formátu. Formát musel být vektorový a převeditelný na rastrovou grafiku. Jako nejvhodnější formát byl zvolen SVG, za rastrovou verzi PNG. Formát SVG byl zvolen pro jeho snadnou převeditelnost do dalších vektorových formátů, což usnadňuje umístění v PDF dokumentu. Rastrová varianta byla zvolena pro verzi výukové opory, která bude založena na (X)HTML. Příklad několika ikon je na obrázku 3.



Obr. 2. Přehled a použití všech vytvořených stylů.



Obr. 3. Ukázka vytvořených piktogramů.

Piktogramy jsou jen jednou částí graficky definovaných objektů, které zvýrazňují text. Pro barevné odlišení logických částí textu bylo zapotřebí upravit kaskádový styl, který je na transformovaný dokument aplikován v jeho finální fázi. Protože není možné barevné pozadí k textu patřící určitým ikonám zvolit náhodně, byla navržena barevná schémata (barvy pozadí a barvy okraje

zvýrazněné části textu), která odpovídají (a ladí) s barvami použitými v jednotlivých ikonách. Barevné kombinace byly voleny s ohledem na přehlednost, aby podle jednotlivých odstínů a tónů barev čtenář okamžitě poznal typ textu, který je ve zvýrazněné oblasti obsažen. Pro následné zvýšení přehlednosti je tato oblast doplněna piktogramem.

Správný kurz má správně dávkované učivo. Tento požadavek souvisí se stylem struktury opory. Protože primárním formátem výstupu je (X)HTML, které je určeno pro výuku, lze s tímto vědomím poměrně kvalitně rozdělit látku na jednotlivé prvky. S vědomím, že DocBook každou podkapitolu ve formátu (X)HTML vygeneruje jako samostatnou stránku, lze efektivně určit rozdělení obsahu opory tak, aby text na monitoru v rozsahu jedné webové stránky nebyl příliš dlouhý.

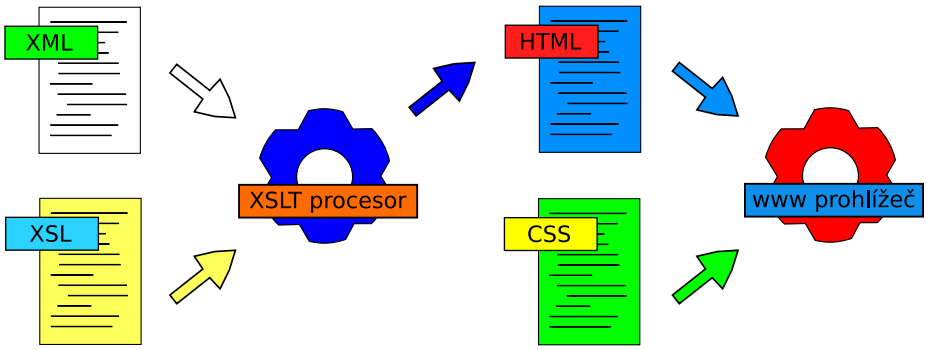
6 Použití XSLT šablon pro generování výukových opor

V této fázi tvorby výukové opory autor má vytvořenu výukovou oporu ve formátu XML. Text je logicky rozčleněn do jednotlivých částí a kapitol. Zbývá dokument v DocBooku převést do požadovaných formátů. Těmito formáty jsou především HTML (resp. XHTML) a PDF.

Pro sestavení výsledných formátů ze zdrojových souborů lze použít několik variant, podrobněji viz [1]. Styly se pro generování výukové opory aplikují způsobem popsaným v následující podkapitole.

6.1 Převod dokumentů z DocBooku

Převod zdrojového kódu výukové opory do HTML formátu je proces, který je zobrazen na obrázku 4.

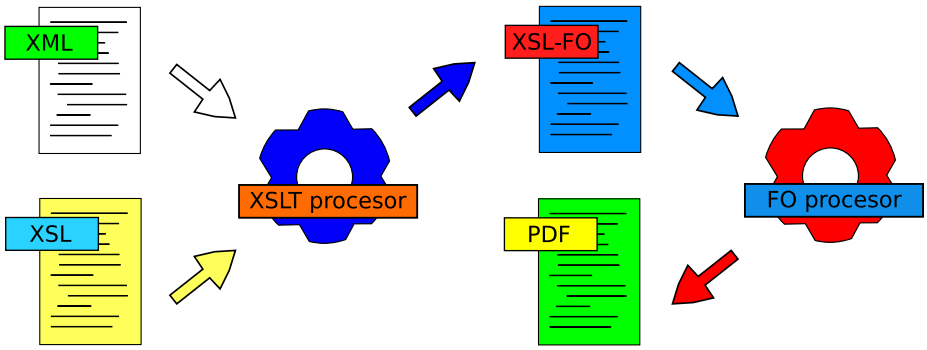


Obr. 4. Generování HTML formátu.

V prvním kroku se použije *XML dokument* (zdrojový kód opory). Ten je následně zpracován *XSLT procesorem* za využití *XSL stylů*. Tento styl byl vytvořen v rámci [1]. Kromě nezbytných pravidel pro výstup do HTML obsahuje

i pravidla, pomocí nichž jsou do výstupu XSLT procesoru přidány piktogramy a další prvky usnadňující navigaci výsledným materiálem. Výstupem XSLT procesoru jsou *HTML dokumenty*, které jsou zobrazovány *internetovým prohlížečem*. Aby HTML dokument obsahoval veškeré grafické prvky, je HTML dokument provázán s *kaskádovými styly* (CSS). Výsledkem jsou dokumenty, mezi nimiž existují hypertextové odkazy, materiál je zpřehledněn pomocí piktogramů a graficky zvýrazněných oblastí.

Pro generování druhého formátu, kterým byl určen PDF, se používá alespoň v první polovině procesu stejný postup. Proces je vidět na obrázku 5.



Obr. 5. Generování PDF formátu.

Zdrojové kódy opory jsou opět zpracovány *XSLT procesorem*, ale za pomoci odlišných *XSL pravidel*, než je tomu v prvním případě. Výstupem XSLT procesoru není hned výstupní dokument, ale *soubor formátovacích objektů* (XSL-FO). Protože XSL-FO je samo o sobě prakticky nepoužitelné, musí být tyto objekty dále zpracovány *FO procesorem*, který vygeneruje výsledný formát (PS, PDF). V případě této práce je výstupem FO procesoru *PDF dokument*.

Z popsaných postupů plyne, že díky různým XSL pravidlům lze dokument z DocBooku převést na jakýkoliv formát, pro který existuje XSL styl.

6.2 Potřebné programové vybavení

Jak bylo v předchozí podkapitole uvedeno, pro generování potřebných formátů jsou zapotřebí XSLT a FO procesory. Za XSLT procesory byly zvoleny programy *Saxon* a *xsltproc*.

Při převodu z DocBooku do HTML je nejprve spuštěn XSLT procesor *xsltproc*. Je použit zejména kvůli kvalitní podpoře spojování dokumentů, které používají technologii XInclude. Program je použit při generování XHTML formátu, který není dalším XSLT procesorem podporován. V druhém kroku při generování opory je využit *Saxon*. Výstupem je sada provázaných HTML dokumentů.

Pro generování PDF formátu výukové opory je zapotřebí FO procesor, který zpracuje soubor formátovacích objektů, jenž je výstupem uvedených XSLT procesorů. Protože v současnosti není k dispozici dostatečně funkční program šiřitelný zdarma, byl ke generování využit program *XEP* v akademické licenci. *XEP* podporuje práci s SVG formátem, ve kterém byly vytvářeny piktogramy a také velmi dobře implementuje XSL-FO standard.

6.3 Instalace nástrojů pro aplikaci šablon

Pro transformaci výukových opor do cílových formátů je zapotřebí množství souborů a programů, které jsou specificky nastaveny. Protože instalace jednotlivých částí programového vybavení je věcí poměrně časově náročnou, byl z tohoto důvodu v rámci [1], [2], [3] sestaven již hotový balík nástrojů, který obsahuje veškeré nástroje potřebné pro konverzi zdrojových souborů výukové opory do formátů HTML, XHTML a PDF:

- DTD DocBook v4.3 a v4.2 (pro website);
- DTD DocBook website v2.5.0;
- DTD DocBook slides v3.3.1;
- ukázkové uživatelské XSL styly;
- XSL styly Normana Walshe v1.68.1;
- FOP v0.20.5;
- Libxml v2.6.1.5 (xsltproc a xmllint);
- knihovnu resolver, zabezpečující podporu XML katalogů;
- Saxon v6.5.3;
- Xalan v2.4.0;
- podklady pro počeštění XEPu (XEP samotný vyžaduje licenci);
- funkční dávkové soubory;
- ukázkové testovací příklady.

7 Problémy

Ne zcela vyřešeno je současné použití nástroje *xsltproc* a XML katalogů na některých systémech Windows XP. Částečné řešení spočívá v používání systémových identifikátorů. Další problémy se projevily při používání FO procesoru XEP, kde vyvstaly problémy s dělením slov a zobrazení diakritiky s rodinou písem Courier fontu Nimbus. Poslední problém je ne zrovna ideální zobrazení HTML a XHTML kódu v prohlížeči Internet Explorer, které je způsobeno špatnou podporou kaskádových stylů ze strany prohlížeče.

8 Závěr

Výsledkem projektu je vytvoření multimediální výukové opory pro předmět *Programování v C#*. Součástí projektu se staly nově vytvořené XSL styly, které odpovídají změnám provedeným v DTD souborech. Pro multimediální prvky byla přidána podpora animací Macromedia Flash a Java appletů.

Vytvořené styly jsou připraveny pro výstup do formátů HTML, XHTML, PostScript a PDF. Ty byly v rámci této práce vybrány jako vhodné z hlediska e-vzdělávání.

Při psaní opor se ukázalo, že zvolený formát je pro přípravu multimediálních učebních textů vhodný, i když klade vyšší nároky na autora. Příprava materiálu je velmi časově náročná záležitost a od autora vyžaduje odborné znalosti nejen z oblasti XML.

Připravené styly byly úspěšně použity při psaní dalších součástí projektu. Všechny nově vytvořené modifikace by měly být díky vytvořené dokumentaci snadno udržované. Budoucí vývoj by měl směřovat k většímu propojení nových stylů se standardními prostředky DocBooku, jako je například překladová služba nebo některé standardní parametry, které jsou prozatím v nových stylech ignorovány. Dále budou zřejmě připravené opory vkládány do LMS umožňujícího adaptaci.

Reference

1. Duda M., Komplexní podpora pro přípravu multimediálních studijních opor: diplomová práce. Ostrava: VŠB-Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky, 2005
2. Ptáček M., Výuková podpora pro elektronické publikování: diplomová práce. Ostrava: VŠB-Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky, 2005
3. Daniel A., Výuková podpora pro přípravu animací a multimédií: diplomová práce. Ostrava: VŠB-Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky, 2005
4. Sovadina J., Multimediální doplňky a cvičení ke studijní opoře C# : diplomová práce. Ostrava: VŠB-Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky, 2005
5. Rajský P., eLearning a jazyk C : diplomová práce. Ostrava: VŠB-Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky, 2002
6. Walsh N., Extreme DocBook [online]. 2004 [cit. 10. 4. 2005]. Dostupné na WWW: `http://nwalsh.com/docs/presentations/extreme2004/slides/index.html`
7. XSL Transformations (XSLT) 1.0, `http://www.w3.org/TR/xslt`
8. XML Inclusions (XInclude) 1.0, `http://www.w3.org/TR/xinclude/`
9. XML Pointer Language (XPointer) 1.0, `http://www.w3.org/TR/WD-xptr`
10. Saxon [online]. [cit. 10. 4. 2005]. Dostupné na WWW: `http://saxon.sourceforge.net/`
11. Xalan-Java 2.6.0 [online]. [cit. 10. 4. 2005]. Dostupné na WWW: `http://xml.apache.org/xalan-j/`
12. The xsltproc tool [online]. [cit. 10. 4. 2005]. Dostupné na WWW: `http://xmlsoft.org/XSLT/xsltproc2.html`
13. RenderX [online]. [cit. 10. 4. 2005]. Dostupné na WWW: `http://renderx.com/`
14. Inkscape [online]. [cit. 18. 4. 2005]. Dostupné na WWW: `http://www.inkscape.org/index.php`

ROZŠÍRENÉ ABSTRAKTY

Aproximace vysokodimenzionálních dat pomocí perceptronových sítí*

Věra Kůrková

Ústav informatiky, Akademie věd ČR
Pod Vodárenskou věží 2, 182 07 Praha 8
vera@cs.cas.cz

Pro libovolný konečný vzorek dat vstup-výstup existuje perceptronová síť, která tato data přesně interpoluje [2]. Taková síť ale vyžaduje stejný počet jednotek jako je velikost vzorku dat a tedy nemusí být implementovatelná. Můžeme ji však aproximovat sítěmi s menším počtem prvků a odhadnout, jak rychle klesají se zvyšujícím se počtem prvků minima součtu čtverců chyb na vzorku dat k nule. Na rychlosti této konvergence záleží, zda se daná data dají dobře aproximovat sítí přijatelné velikosti. Pomocí výsledků z teorie nelineární aproximace se dá rychlost konvergence odhadnout v závislosti na velikosti variačních norem funkcí, které vzorek dat interpolují [1], [5].

Variační normy hrají v teorii nelineární aproximace podobnou roli jako Sobolevovy normy (které měří velikost parciálních derivací) v teorii lineární aproximace. Z této teorie je známo, že pro zajištění rychlosti konvergergence $\mathcal{O}(\frac{1}{n})$ je třeba omezit součet všech iterovaných parciálních derivací až do stupně d rovnajícího se počtu proměnných [7]. Vzhledem k tomu, že se jedná o 2^d sčítanců, s rostoucím počtem proměnných prudce rostou omezení na data, která lze efektivně aproximovat pomocí lineárních metod.

V případě perceptronových sítí jsou podmínky na vysokodimenzionální data mnohem mírnější. Pro rychlou konvergenci stačí, když data lze interpolovat funkcí, pro jejíž parciální derivace až do stupně d platí velice mírné omezení: stačí když jsou tyto derivace menší než exponencionálně rostoucí funkce d tvaru $(\frac{d}{4\pi})^{1/2}(\frac{2\pi}{e})^{d/2}$. Tento výsledek plyne ze vztahu mezi variační normou odpovídající perceptronům a Sobolevovou seminormou, která je na rozdíl od Sobolevovy normy používané v lineární aproximaci definovaná jako maximum absolutních hodnot parciálních derivací. Vztah mezi těmito variační a Sobolevovou seminormou byl odvozen v [4] na základě integrální reprezentace ve tvaru neuronové sítě s kontinuem jednotek.

V případě neuronových sítí se tedy dokonce s rostoucí dimenzí dat zvyšuje tolerance na velikost parciálních derivací funcí, které tato data interpolují. Perceptronové sítě jsou tedy vhodnějším nástrojem pro zpracování vysokodimenzionálních dat než lineární metody.

Je možno ale nalézt příklady dat, kdy ani tato velká tolerance nestačí, každá funkce, která je interpoluje má větší parciální derivace než $(\frac{d}{4\pi})^{1/2}(\frac{2\pi}{e})^{d/2}$. Existence těchto funkcí plyne z porovnání velikosti kvaziortogonální dimenze [3]

* Tato práce byla částečně podporována grantem 1ET100300517 programu Informační společnost.

a počtu Boolovských funkcí, které lze počítat perceptrony [8]. Konstrukce takových funkcí byla popsána v [6].

Reference

1. Barron A.R., Neural Net Approximation. In: Proceedings of the 7th Yale Workshop on Adaptive and Learning Systems, 1992, 69–72
2. Ito Y., Finite Mapping by Neural Networks and Truth Functions. Math. Scientist 17, 1992, 69–77
3. Kainen P.C., Kůrková V., Quasiorthogonal Dimension of Euclidean Spaces. Applied Math. Letters 6, 1993, 7–10
4. Kainen P.C., Kůrková V., Vogt A., A Sobolev-Type Upper Bound for Rates of Approximation by Linear Combinations of Plane Waves. Research Report ICS-2003-900, Institute of Computer Science, Prague, 2004
5. Kůrková V., (2003). High-Dimensional Approximation and Optimization by Neural Networks. Chapter 4 In: Advances in Learning Theory: Methods, Models and Applications, Eds. J. Suykens et al., IOS Press, Amsterdam, 2003, 69–88
6. Kůrková V., Savický P., Hlaváčková K., Representations and Rates of Approximation of Real-Valued Boolean Functions by Neural Networks. Neural Networks 11, 1998, 651–659.
7. Pinkus A., n-Width in Approximation Theory. Berlin: Springer, 1986
8. Shläfli L., Gesamelte Mathematische Abhandlungen. Band 1. Basel, Verlag Birkhäuzer, 1950

Semantic annotation based on regular expressions

M. Laclavík, E. Gatial, Z. Balogh, O. Habala, G. Nguyen, and L. Hluchý

Institute of Informatics, Slovak Academy of Sciences
Dúbravská cesta 9, 845 07 Bratislava, Slovakia
laclavik.ui@savba.sk

Abstract. In this paper we describe a solution for creation of ontological metadata from text documents using a semantic annotation. A lot of solutions are known for creation of ontological metadata from text documents. Our solution is based on a regular expression. The solution uses a regular expression - ontology pairs to detect ontology concepts and annotate the document with them. The solution requires having ontology describing the problem domain. The paper explains two possible applications for using such technique.

1 Pattern ontology model and annotation algorithm

The instances of the Pattern class are used to define and identify relations between text and domain ontology, where the pattern property contains the regular expression which describes textual representation of the ontology element. The examined text is processed with the regular expression for every pattern and when it is found: the detected ontology element of hasClass or hasInstance represents text in the chosen problem domain. Moreover, when the hasClass property exists in the Pattern, the RDQL query is constructed and processed to find the individuals that match the condition:

- individual is the class of hasClass
- a property of individual contains the matched word

Figure 1 shows results of the annotation based on patterns from Figure 2. We will describe an algorithm by an illustration example from the Znalosti project where a job offer text is analyzed and annotated by regular expressions patterns (Figure 2). System will find related ontology elements from domain ontology. In this example the job offer location - New York and USA are identified by a regular expression ($[A-Za-z]+$) a ($[-A-Za-z0-9]+[]+[-A-Za-z0-9]+$), because individual $locNY$ has the property title $NewYork$, $locUS$ has the property title USA. Similarly other ontology elements are detected. Some regular expressions search for ontology individuals, other ontology classes and others such as $pattFullTime$ (Figure 2) annotate a job offer by a concrete individual $jtPermanent$ if expression ($Full[-]Time$) is found. Systems detect ontology elements based on domain ontology. In this example it is ontology of job offers.

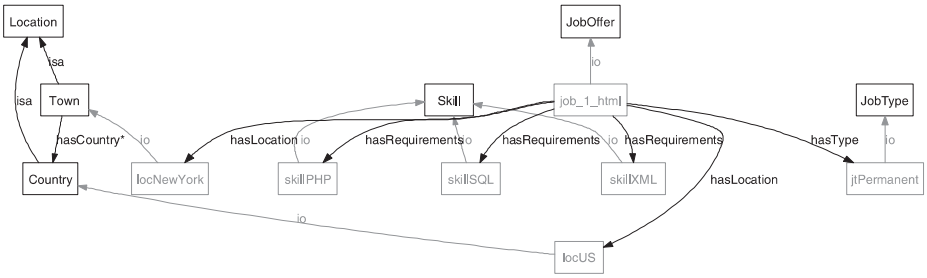


Fig. 1. Job Offer Individual with its properties detected by regular expressions showed in Figure 2 from Znalosti job offer application.

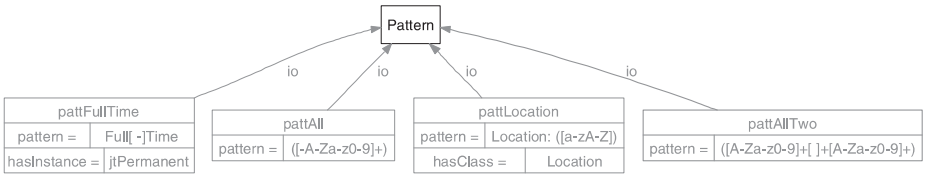


Fig. 2. Fragment of pattern ontology from Znalosti job offer application.

2 Conclusion

The described solution is used in K-Wf Grid [2] and the Znalosti project to detect relevant structured knowledge described by a domain specific ontology model in the unstructured text. The main difference between existing annotation solutions such as Anotea [1] is detection of ontology elements from existing domain ontology, while other annotation solutions try to create such ontology. In the Znalosti project our solution is used to detect structured information about job offers. In the K-Wf Grid project our solution is used to detect a user context/problem from the text description as well as annotate user knowledge entered in a form of text notes[3]. The main disadvantage is that regular expression patterns need to be manually created. In our future work we will focus on semi-automatic pattern creation.

Acknowledgments
This work is supported by the project K-Wf Grid EU RTD IST FP6-511385, VEGA No. 2/3132/23, APVT-51-024604 and SPVV 1025/2004.

References

1. Annotea Project, 2001, `http://www.w3.org/2001/Annotea/`
2. K-Wf Grid Consortium, K-Wf Grid Project Website, 2005, `http://www.kwfgrid.net/`
3. Laclavik M., Gatial E., Balogh Z., Habala O., Nguyen G., Hluchy L., Experience Management Based on Text Notes (EMBET), Challenges Conference, 19 - 21 October 2005, Ljubljana, Slovenia , 2005

Query optimization by genetic programming

Suhail S. J. Owais, Pavel Krömer, and Václav Snášel

Department od Computer Science,
VŠB - Technical univerzity of Ostrava Ostrava,
17. listopadu 15, 708 33 Ostrava-Poruba, Česká republika
{Suhail.Owais, Pavel.Kromer, Vaclav.Snasel}@vsb.cz

Abstract. When implementing an IR system that can support comprehensive searches from a wide variety of documents, it is crutial to have mechanism for selecting appropriative query formulations. For this purpose, we use Genetic Programming. Experiment with optimizing of a Boolean query will be presented in this paper.

1 Introduction

There are two major user model for evaluating IR systems for WWW documents. In the first, a user attempts to find a comprehensive set of documents relevant for his/her information needs. In other model, the user requeries just one or a few queries. Our research aims to support model for build new queries from a few queries (for example, from log file).

Evaluation of the information retrieval system, measured by three statistics measures precision and recall and the harmonic mean of precision and recall (F-score) [3]. The F-score evaluation measure evaluate as the average for precision and recall measures.

$$Recall = \frac{RelevantRetrieved}{Relevant} \qquad Precision = \frac{RelevantRetrieved}{Retrieved}$$

$$F - score = \frac{2 * Precision * Recall}{Precision + Recall}$$

2 Evolutionary algorithms

Evolutionary algorithms are stochastic search methods that mimic the metaphor of natural biological evolution, which applies the principles of evolution found in nature to the problem of finding an optimal solution to a solver problem [1], [4].

In this work see [3], we develop genetic programming for implementing GA operators with variable length of chromosomes and mixture symbolic of information, like the five Boolean operators **and**, **or**, **xor**, **of** and **not**.

3 Conclusions

In this paper, a genetic programming were implemented to optimize a Boolean query. We used Precision; Recall; and Harmonic mean of precision and recall 'F-score' as an objective function to measure the performance of the system, also different types of mutations were used too.

We used most of the Boolean operators "**and**, **or**, **xor**, **of** and **not**" operators. And because of implemented each Boolean operator in a separate class, simple we can adding any other Boolean operators.

From our system tests experiments we obtained the following results:

- *First*, when applying mutation operator on terms, the list of terms from whole collection was used; it give us best results with less number of generations than if terms used from user query terms list only or from initial population terms list. Only then there can come into existence new queries, describing the same documents as user query request, but it may contains terms not included in user query or in initial population.
- *Second*,when using precision as a fitness function we produce best results for optimized Boolean queries with highest performance within less number of generations than when using recall or F-score.
- *Third*, in some cases, especially when we used for mutation over leaves the terms from collection or from initial population on two different types of fitness functions, so an optimized queries were reached within few number of generations.

We will focus in our *future works* on weighted terms and weighted Boolean operators for implementing the fuzzy logic operations [2].

References

1. Cordon O., Herrera-Viedma E., Luque M., Evolutionary Learning of Boolean Queries by Multiobjective Genetic Programming. J.J. Merelo Guervos et al. (Eds.): PPSN VII, LNCS 2439, Springer-Verlag Berlin Heidelberg, 2002, 710-719
2. Kraft D.H., Bordogna G., Pasi G., Fuzzy Set Techniques in Information Retrieval, in Bezdek J.C., Didier D. and Prade H. (eds.), Fuzzy Sets in Approximate Reasoning and Information Systems, vol. 3, The Handbook of Fuzzy Sets Series, Norwell, MA: Kluwer Academic Publishers, 1999
3. Owais S.S., Krömer P., Snášel, V., Implementation of Evolutionary Algorithms-Genetics Programmning for Optimization of Boolean Queries in Information Retrieval System. In GESTS International Transaction on Computer Science and Engineering. Editor: J.M. Jun, B.M. Bae, K.Y. Lee, Global Engineering, Science, and Technology Society (GESTS). Volume 4, Number 1. April 30, 2005, 47–66
4. Smith M.P., Smith M., The Use of Genetic Programming to Build Boolean Queries for Text Retrieval through Relevance Feedback, Journal of Information Science 23:6, 1997, 423-431
5. Yoshioka M., Haraguchi M., An Appropriate Boolean Query Reformulation Interface for Information Retrieval Based on Adaptive Generalization. WIRI in conjunction with IEEE ICDE 2005, Tokyo, Japan 2005, 148–153

TUTORIÁLY

Získávání logických tvrzení z dat jako významný směr dobývání znalostí z dat

Martin Holeňa

Ústav informatiky AV ČR, Pod Vodárenskou věží 2, 182 07 Praha 8
martin@cs.cas.cz

Abstrakt Příspěvek se zabývá problematikou získávání logických tvrzení z dat, tedy těmi metodami dobývání znalostí z dat (data mining), jejichž výsledky lze vyjádřit v jazyce nějaké formální logiky. Je podán velmi stručný přehled širokého spektra rozmanitých metod tohoto typu, jak metod vycházejících ze statistických přístupů, tak i metod spočívajících na principech strojového učení, a je poukázáno na specifický charakter metod založených na umělých neuronových sítích. Pro ilustraci jsou podrobněji popsány dvě konkrétní metody získávání logických tvrzení z dat. Jednou z nich je metoda Guha, která vychází z observační logiky a je pravděpodobně nejstarší metodou získávání pravidel z dat vůbec. Druhou je metoda založená na po částech lineárních vícevrstvých perceptronech.

1 Úvod

Široké spektrum metod pro dobývání znalostí z dat (data mining) s sebou přináší velkou rozmanitost různých formálních reprezentací, v nichž mohou být získané znalosti zachyceny, např. rozhodovací a asociační pravidla, klasifikační hierarchie, shluky, regresní funkce, pravděpodobnostní sítě či další grafické struktury. Většina těchto reprezentací je specifická pro určité třídy metod, např. klasifikační hierarchie pro klasifikaci, shluky pro shlukovou analýzu, regresní funkce pro lineární a nelineární regresní analýzu. Důsledkem toho je, že metody zachycující znalosti získané z dat pomocí téže formální reprezentace bývají typicky založené na podobných principech, a to i v případech, kdy jde o metody zcela rozdílného původu. Tak například se shluky se setkáváme v metodách vyvinutých ve statistické shlukové analýze, v metodách vyvinutých v teorii fuzzy množin i v metodách pocházející z umělých neuronových sítí – při studiu všech konkrétních metod, v nichž se používají, však zjistíme, že principy, na nichž tyto metody spočívají, jsou si ve skutečnosti velmi blízké.

Z této obecné charakteristiky však existuje jedna velmi významná výjimka – reprezentace ve formě tvrzení nějaké formální logiky. Taková tvrzení, obvykle nazývaná *logická pravidla* nebo prostě jen *pravidla*, se používají k zachycení znalostí získaných z dat ve velmi mnoha metodách a tyto metody jsou založeny na značně rozmanitých principech. Cílem tohoto příspěvku je naznačit význam získávání pravidel z dat prostřednictvím stručného přehledu hlavních tříd metod, v nichž se s ním lze setkat. Tento přehled se pro stručnost omezuje pouze na

oblasti a technologie, které již v dobývání znalostí z dat mají relativně pevné místo (Obr. 1). Pozornost proto není věnována získávání pravidel z dat v těch oblastech dobývání znalostí, které jsou teprve ve stádiu výzkumu, jako např. získávání pravidel z dat pomocí teorie neostrých množin [1], [2]. Naproti tomu příspěvek upozorňuje na specifický charakter získávání pravidel z dat pomocí umělých neuronových sítí. Přehled metod získávání pravidel z dat je podán velmi zhuštěně, pro ilustraci jsou však popsány dva konkrétní příklady těchto metod, včetně jedné metody založené na umělých neuronových sítích.

Lapidárně lze metody získávání pravidel z dat charakterizovat jako metody, jejichž vstupem je množina dat a výstupem množina pravidel. První z těchto množin lze pokládat za *extenzionálně vyjádřenou relaci* na kartézském součinu oborů hodnot jednotlivých atributů, zatímco druhou lze naopak interpretovat pomocí *intenzionálně vyjádřené relace*, jež by v nějakém smyslu měla odpovídat extenzionální relaci na vstupu. Vstupní relace se dnes obvykle získává z relační databáze, kde je buď přímo uložena jako tabulka, nebo ji lze odvodit z uložených tabulek pomocí operací relační algebry. V minulosti byla typicky získávána jako nestrukturovaný soubor dat nebo konstruována pomocí navigačních operací při procházení hierarchických nebo síťových (codasylovských) databází. Výstupní množina pravidel může být vyjádřena v jazyce jakékoliv formální logiky. V existujících metodách získávání pravidel z dat je touto logikou vždy buď nějaká booleovská logika nebo některá z fuzzy logik.

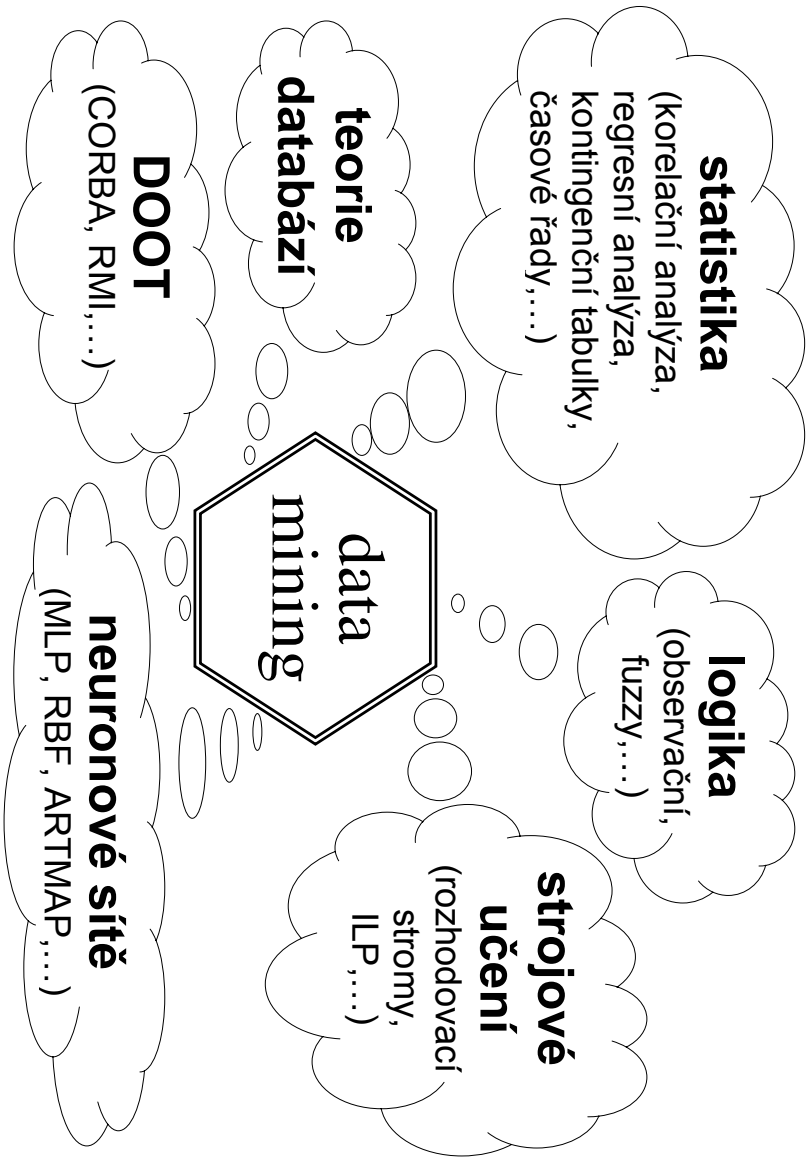
2 První příklad – metoda Guha

Pravděpodobně nejstarší metodou získávání pravidel z dat je *metoda Guha* (General unary hypotheses automaton), rozpracovaná v 60. – 70. letech českými matematiky [3]. Pravidla, která Guha získává z dat, jsou tvrzení observační logiky, což je booleovská predikátová logika se zobecněnými kvantifikátory. Máme-li soubor dat o n objektech, je pravdivostním ohodnocením booleovského predikátu φ na těchto objektech vektor $\|\varphi\| \in \{0,1\}^n$, zatímco pravdivostním ohodnocením tvrzení $(Qx)(\varphi_1(x),\ldots,\varphi_m(x))$ sestávajícího z m booleovských predikátů $\varphi_1,\ldots,\varphi_m$ a m-árního zobecněného kvantifikátoru Q je hodnota

$$\|(Qx)(\varphi_1(x),\ldots,\varphi_m(x))\| = \mathrm{Tf}_Q(\|\varphi_1\|,\ldots,\|\varphi_m\|), \tag{1}$$

kde Tf_Q označuje $\{0,1\}$-hodnotovou funkci na množině m-sloupcových binárních matic, tedy $\mathrm{Tf}_Q : \bigcup_{k=1}^{\infty}\{0,1\}^{k,m} \to \{0,1\}$, nazývanou *pravdivostní funkce* kvantifikátoru Q. V metodě Guha jsou přitom získávána z dat pouze tvrzení velmi speciálního typu:

- vždy mají tvar $(\sim x)(\varphi(x),\psi(x))$, zjednodušeně zapisovaný $\varphi \sim \psi$, s booleovskými predikáty φ a ψ a binárním zobecněným kvantifikátorem $\sim$;
- pravdivostní funkce kvantifikátoru $\sim$ je vždy konstruována jako funkce čtyřpolní tabulky pravdivostních ohodnocení predikátů φ a ψ:



Obr. 1. Hlavní oblasti a technologie používané v dobývání znalostí z dat (data mining).

$$\|\varphi \sim \psi\| = \mathrm{Tf}(a_{\varphi,\psi}, b_{\varphi,\psi}, c_{\varphi,\psi}, d_{\varphi,\psi}), \tag{2}$$

kde

$$\mathrm{Tf} : \mathcal{N}_0^4 \to \{0,1\}, \tag{3}$$

$$a_{\varphi,\psi} = \#\{i : \|\varphi\|_i = \|\psi\|_i = 1\}, \tag{4}$$
$$b_{\varphi,\psi} = \#\{i : \|\varphi\|_i = 1 \;\&\; \|\psi\|_i = 0\}, \tag{5}$$
$$c_{\varphi,\psi} = \#\{i : \|\varphi\|_i = 0 \;\&\; \|\psi\|_i = 1\}, \tag{6}$$
$$d_{\varphi,\psi} = \#\{i : \|\varphi\|_i = \|\psi\|_i = 0\}. \tag{7}$$

S použitím značení $\#S$ pro počet prvků množiny S. Ačkoliv pravdivostní funkci Tf splňující (3) – (7) lze definovat zcela abstraktně, definice zobecněných kvantifikátorů nejčastěji používaných v aplikacích byly ve skutečnosti inspirovány statistickou analýzou dat. Konkrétně jde o následující kvantifikátory:

1. Kvantifikátor $\to_\theta$, označovaný jako *fundovaná implikace* s prahem $\theta \in (0,1]$ se používá k získání odpovídajícího observačního tvrzení (tj., $\varphi \to_\theta \psi$) v závislosti na hodnotě nestranného odhadu podmíněné pravděpodobnosti $p_{\psi|\varphi}$ že pravdivostní ohodnocení predikátu ψ je "1" za podmínky že pravdivostní ohodnocení predikátu φ na témže objektu je "1". Pravdivostní funkce tohoto kvantifikátoru je definována:

$$\mathrm{Tf}_{\to_\theta}(a,b,c,d) = \begin{cases} 1 & \text{při } \frac{a}{a+b} \geq \theta, \\ 0 & \text{jinak.} \end{cases} \tag{8}$$

2. Kvantifikátor $\to^!_{\theta,\alpha}$, označovaný jako *dolní kritická implikace* s prahem $\theta \in (0,1]$ se používá k získání observačního tvrzení $\varphi \to^!_{\theta,\alpha} \psi$ na základě testu hypotézy $p_{\psi|\varphi} \leq \theta$ proti alternativě $p_{\psi|\varphi} > \theta$ pomocí binomiálního testu na hladině významnosti α. Jeho pravdivostní funkce je definována:

$$\mathrm{Tf}_{\to^!_{\theta,\alpha}}(a,b,c,d) = \begin{cases} 1 & \text{při } \sum\limits_{i=a}^{a+b} \binom{a+b}{i} \theta^i (1-\theta)^{a+b-i} \leq \alpha, \\ 0 & \text{jinak.} \end{cases} \tag{9}$$

3. Konečně *Fisherův kvantifikátor* $\sim^F_\alpha$ a χ^2*-qkvantifikátor* $\sim^{\chi^2}_\alpha$ se používají k získání odpovídajících observačních tvrzení $\varphi \sim^F_\alpha \psi$ a $\varphi \sim^{\chi^2}_\alpha \psi$ na základě testování nezávislosti ohodnocení predikátů φ a ψ proti alternativě pozitivní závislosti těchto ohodnocení. Přitom Fisherův kvantifikátor odpovídá testování pomocí Fisherova jednostranného faktoriálového testu na hladině významnosti α, zatímco χ^2-qkvantifikátor odpovídá testování pomocí χ^2 testu na asymptotické hladině významnosti α. Tyto kvantifikátory mají následující pravdivostní funkce:

$$
\begin{aligned}
1 \leq \texttt{petal length} &\leq 3 \rightarrow^{!}_{0.8,0.05} \texttt{class = Setosa}\\
0.1 \leq \texttt{petal width} &\leq 0.6 \rightarrow^{!}_{0.8,0.05} \texttt{class = Setosa}\\
0.6 \leq \texttt{petal width} &\leq 1.6 \rightarrow^{!}_{0.8,0.05} \texttt{class = Versicolor}\\
5 \leq \texttt{petal length} &\leq 7 \rightarrow^{!}_{0.8,0.05} \texttt{class = Virginica}\\
1.6 \leq \texttt{petal width} &\leq 2.6 \rightarrow^{!}_{0.8,0.05} \texttt{class = Virginica}
\end{aligned}
$$

$$
\begin{aligned}
1 \leq \texttt{petal length} &\leq 3 \;\&\; 0.1 \leq \texttt{petal width} \leq 0.6 \sim^{F}_{0.01} \texttt{class = Setosa}\\
3 \leq \texttt{petal length} &\leq 5 \;\&\; 0.6 \leq \texttt{petal width} \leq 1.6 \sim^{F}_{0.01} \texttt{class = Versicolor}\\
5 \leq \texttt{petal length} &\leq 7 \;\&\; 1.6 \leq \texttt{petal width} \leq 2.6 \sim^{F}_{0.01} \texttt{class = Virginica}
\end{aligned}
$$

Obr. 2. Ukázka tvrzení získaných metodou Guha ze známých Fisherovýchd dat o kosatcích.

$$
\mathrm{Tf}_{\sim^{F}_{\alpha}}(a,b,c,d) = \begin{cases} 1 & \text{při } ad > bc \wedge \displaystyle\sum_{i=a}^{\min(a+b,a+c)} \frac{\binom{a+c}{i}\binom{b+d}{a+b-i}}{\binom{a+b+c+d}{a+b}} \leq \alpha, \\ 0 & \text{jinak,} \end{cases} \tag{10}
$$

$$
\mathrm{Tf}_{\sim^{\chi^2}_{\alpha}}(a,b,c,d) = \begin{cases} 1 & \text{při } ad > bc \wedge \frac{(a+b+c+d)(ad-bc)^2}{(a+b)(c+d)(a+c)(b+d)} \geq \chi^2_1(1-2\alpha), \\ 0 & \text{jinak.} \end{cases} \tag{11}
$$

Obrázek 2 ukazuje typický příklad tvrzení získaných metodou Guha, v tomto případě z jednoho z nejznámějších testovacích souborů dat, Fisherových dat o kosatcích.

Statistickou analýzou dat jsou inspirovány rovněž nedávná zobecnění metody Guha pro získávání tvrzení fuzzy logiky z dat [4], [5].

3 Další metody získávání pravidel z dat

Přes svůj historický význam je Guha dnes již pouze jednou z širokého spektra metod dobývání znalostí z dat, jejichž výstup lze formálně reprezentovat jako pravidla. Připomínku si rozhodně zaslouží přinejmenším následující další zástupci těchto metod:

Několik často používaných metod získává z dat *asociační pravidla*, tj. tvrzení booleovské logiky, která platí v datech s alespoň předepsanou spolehlivostí a jsou podporována alespoň předepsaným procentem případů [6], [7], [8], [9]. Právě tyto metody představují nejbližší příbuzné metody Guha mezi moderními metodami dobývání znalostí z dat. Pravda, dvě desetiletí, která je od vzniku metody Guha oddělují, přinesla řadu terminologických posunů. Tak např. již

sám termín „asociační pravidla" navozuje parlelu s třídou zobecněných kvantifikátorů používaných v metodě Guha a označovaných jako *asociační kvantifikátory*. Ve skutečnosti však asociační pravidla získávána z dat metodami požitými v citovaných pracech jsou vždy booleovskými implikacemi, díky čemuž tyto metody odpovídají mnohem užší třídě *implikačních kvantifikátorů*. Terminologické odlišnosti však pouze znesnadňují rozpoznání překvapivě silné podobnosti mezi moderními metodami pro získávání asociačních pravidel a metodou Guha. Ukázala to nedávná srovnávací analýza obou přístupů v [10].

Při získávání asociačních pravidel není nijak na překážku, jestliže se předpoklady (levé strany implikace) dvou či více pravidel překrývají. Naopak, nemalá část dat, z nichž pravidla získáváme, bývá typicky popsána pomocí předpokladů několika pravidel současně. Soubory pravidel s překrývajícími se předpoklady jsou však nevýhodné, pokud je chceme použít ke klasifikaci dat, a zejména pokud na jejich základě chceme činit nějaká rozhodnutí. V takových situacích se používají metody, jejichž výsledkem jsou soubory booleovských implikací s nepřekrývajícími se předpoklady. Pravidla uvedeného typu nazýváme *rozhodovací pravidla*. K nejvýznamějším zástupcům metod pro získávání rozhodovacích pravidel z dat patří metody AQ [11], [12],CN2 [13], a zejména velká skupina metod označovaných jako *rozhodovací stromy* [14], [15], [16]. Soubory pravidel získaných pomocí této skupiny mohou být díky své hierarchické struktuře snadno vizualizovány jako grafy stromového typu, čemuž rozhodovací stromy vděčí za své jméno. Ještě důležitější je, že právě snadná vizualizovatelnost získaných pravidel je příčinou značné popularity této třídy metod – pokud je výška stromu dostatečně nízká, jsou získaná pravidla velmi dobře srozumitelná. Navíc jsou rozhodovací stromy velmi robustní vůči chybám v datech, neboť hranice oblastí odpovídajících platnosti antecedentů jednotlivých pravidel jsou v jejich případě po částech konstantní a zpravidla nijak nezávisí na datech ve větší vzdálenosti.

Na přelomu mezi asociačními a rozhodovacími pravidly jsou *fuzzy rozhodovací pravidla*. I v jejich případě jde o implikace, avšak o implikace nějaké fuzzy logiky. I tato pravidla se používají k rozhodování a klasifikaci, avšak fuzzy logika obecně neumožňuje dosáhnout toho, aby se předpoklady různých pravidel nepřekrývaly, takže výsledek aplikace všech získaných pravidel je nikoliv konkrétní rozhodnutí či třída, ale fuzzy množina na množině všech rozhodnutí či tříd. Dobrý teoretický popis fuzzy rozhodovacích pravidel lze najít v [17], k nejznámějším metodám pro získávání takových pravidel patří ANFIS [18], [19] and NEFCLASS [20], [21].

Induktivní logické programování (ILP) spočívá vpodstatě v indukci intenzionální definice relace (která má právě formu logických pravidel) na základě extenzionálního popisu té její části a té části jejího doplňku, které jsou k dispozici v datech, tj. na základě pozitivních a negativních příkladů této relace v datech [22], [23]. Přitom se může využívat znalostí zachycených v již existujících intenzionálních definicích jiných relací. Dnes již existuje celá řada implementovaných ILP systémů, pro indukci relace odpovídající jedinému konceptu i pro současnou indukci více relací odpovídajících různým konceptům, pro indukci dávkovým i inkrementálním způsobem, pro interaktivní i neinteraktivní indukci.

Získávání pravidel z dat pomocí genetických algoritmů je dnes pravděpodobně nejrozpracovanější aplikací genetických algoritmů v dobývání znalostí z dat [24]. Použití genetických algoritmů k získávání pravidel z dat je možné díky tomu, že genetické algoritmy jsou optimalizační metodou, která pracuje pouze s funkčními hodnotami optimalizované funkce. Tou je v případě získávání pravidel z dat obvykle nějaká charakteristika výsledného souboru získaných pravidel, např. jeho celková správnost, přesnost či úplnost, ale také různé charakteristiky zajímavosti získaných pravidel, nebo kombinace více charakteristik. Jako každá jiná metoda pro optimalizaci obecných funkcí i genetické algoritmy musí startovat z nějakého předem dodaného počátečního členu iterační posloupnosti, která v ideálním případě konverguje k hledanému optimu funkce. Na rozdíl od jiných optimalizačních metod však genetické algoritmy hledají optimum pomocí celé populace iteračních posloupností, proto potřebují i celou populaci počátečních členů, tvořících dohromady výchozí generaci optimalizovaných jedinců. Získání této výchozí generace představuje samostatný problém, intenzivně studovaný jak v rámci genetických algoritmů obecně, tak v souvislosti s jejich použitím k získávání pravidel z dat [24], [25]. Pokud jde o vlastní optimalizaci, tj. aplikaci genetických operátorů selekce, křížení a mutace na jedince tvořící populaci, používají se při získávání pravidel z dat dva principiálně odlišné přístupy. Tzv. *Pittsburský přístup* vychází z toho, že cílem není získání jednotlivých pravidel, ale celého souboru pravidel co nejlépe popisujícího data. Jedinci, na které se aplikují genetické operátory, jsou proto celé soubory pravidel, a v každé generaci se získá celá populace takových souborů. Naproti tomu v tzv. *Michiganském přístupu* jsou jedinci jednotlivá pravidla a v každé generaci se získá populace pravidel. Michiganský přístup je tudíž výpočetně jednodušší, musí však externě řešit problémy s nekonzistencí a redundancí pravidel, které v případě Pittsburského přístupu lze snadno zabudovat již přímo do genetických operátorů.

4 Specificita metod založených na neuronových sítích

Všechny dosud zmíněné metody získávání pravidel z dat mají jeden důležitý společný rys – pravidla jsou získávána přímo z extenzionálně vyjádřené vstupní datové relace, mezi touto relací a pravidly se znalosti neuchovávají v žádné další formální reprezentaci. Tento rys však přesto není univerzálním rysem všech metod získávání pravidel z dat. Přesněji řečeno, nemá jej jedna důležitá třída metod, která zde dosud zmíněna nebyla – *metody založené na umělých neuronových sítích*. Všimněme si na závěr právě této třídy metod poněkud podrobněji.

Ve skutečnosti již zobrazení, které síť realizuje, reprezentuje znalosti přenesené do ní v průběhu učení z dat, znalosti o tom, jak jsou určité hodnoty vstupních proměnných korelovány s určitými hodnotami výstupních proměných. Tyto znalosti jsou reprezentovány částečně pomocí *architektury sítě*, především ale pomocí mnohorozměrného *vektoru parametrů*, který společně s architekturou zobrazení realizované sítí jednoznačně určuje. Např. V případě vícevrstvých perceptronů jsou takovými parametry *váhy* všech spojů mezi sousedními vrstvami sítě a *aktivační prahy* všech skrytých a výstupních neuronů. Je to právě

tato „rozprostřená" reprezentace pomocí architektury a parametrů realizovaného zobrazení, které vícevrstvé perceptrony vděčí za své vynikající aproximační vlastnosti (viz např. [26], [27], [28], [29]). Člověku je však tato reprezentace mnohem méně srozumitelná než logická pravidla (v terminologii, kterou oblast dobývání znalostí z dat převzala z kognitivních věd, má reprezentace znalostí V umělých neuronových sítích vysoký „datový fit", ale nízký „mentální fit"). Proto se již od konce osmdesátých let studují metody extrakce pravidel z umělých neuronových sítí ([30], [31], [32], [33], [34], [35], [36], [37], [38], odkazy na metody publikované před rokem 1999 lze nalézt v přehledových článcích [39], [40], [41]). Dnes již těchto metod existuje několik desítek a navzájem se liší v celé řadě aspektů, z nichž nejdůležitější jsou expresivnost metody (booleovské a fuzzy metody), její transparentnost, složitost a univerzálnost, přípustné typy proměnných odpovídajících vstupům a výstupům sítě, jakož i korektnost, úplnost, přesnost a věrnost získávaných pravidel. Jejich společným rysem však je, že pracují nejenom s informacemi o dvojicích vektorů ze vstupního a výstupního prostoru sítě, které byly použity již k jejímu trénování, ale v menší či větší míře i s informacemi o dvojicích získaných s využitím zobrazení realizovaného sítí. Některé z metod pracují dokonce pouze s dvojicemi získanými s využitím realizovaného zobrazení, trénovací dvojice vůbec nepotřebují. V metodách získávání pravidel z dat založených na umělých neuronových sítích je proto zobrazení realizované sítí vždy vloženo mezi data a výsledná pravidla, a jemu odpovídající reprezentace pomocí architektury a parametrů představuje přechodnou reprezentaci znalostí, z níž teprve metoda vychází pro získání reprezentace ve formě logických pravidel. Tato vložená přechodná reprezentace znalostí představuje důležitý specifický rys metod založených na umělých neuronových sítích, odlišující je od ostatních typů metod získávání pravidel z dat.

5 Druhý příklad – po částech lineární vícevrstvé perceptrony

Většina dosud navržených metod pro získávání pravidel z umělých neuronových sítí spočívá do značné míry na heuristikách. Metod, jejichž vlastnosti lze studovat matematicky, existuje pouze několik. Pro ilustraci metod získávání pravidel z dat založených na umělých neuronových sítí bude nyní nastíněna jedna z nich.

Je to metoda pro získávání pravidel z vícevrstvých perceptronů se spojitými sigmoidními aktivačními funkcemi, založená na přechodu od původního perceptronu k perceptronu s po částech lineárními aktivačními funkcemi, tzv. po částech lineárních vícevrstvých perceptronů [42]. Takový přechod je možný díky hustotě prostoru po částech lineárních sigmoidních funkcí v prostoru spojitých sigmoidních funkcí a díky tomu, že všechny algoritmy pro trénování vícevrstvých perceptronů se spojitými aktivačními funkcemi jsou iterační. Soubor vah a prahů, který algoritmus nalezne v konečném počtu kroků, a tomuto souboru odpovídající zobrazení F, které síť realizuje, tudíž obecně nejsou vzhledem k použité učicí posloupnosti $(x_1, y_1), \ldots, (x_p, y_p)$ dvojic vstupních a výstupních vektorů

optimální, tj. neminimalizují nesoulad mezi $(y_1, \ldots, y_p)$ a $(F(x_1), \ldots, F(x_p))$, měřený např. čtvercem jejich euklidovské vzdálenosti. Lze pouze pro libovolné dané $\varepsilon > 0$ nalézt ε-suboptimální zobrazení, pro něž je nesoulad mezi vektory $(y_1, \ldots, y_p)$ a $(F(x_1), \ldots, F(x_p))$ nejvýše ε nad minimem. A suboptimalita je invariantní vzhledem k po částech lineární aproximaci aktivační funkce v tom smyslu, že při každých konstantách $\varepsilon > \delta > 0$ se ke každému δ-suboptimálnímu zobrazení F, které je realizováno nějakým vícevrstvým perceptronem se spojitou sigmoidní aktivační funkcí dá sestrojit taková po částech lineární aproximace této aktivační funkce, že zobrazení G, které při ní síť realizuje, je pořád ještě ε-suboptimální (přesná formulace této vlastnosti je uvedena v [42]).

Z vícevrstvého perceptronu s n_I vstupními neurony které odpovídají proměnným $X_1, \ldots, X_{n_I}$ a n_O výstupními neurony, které odpovídají proměnným $Y_1, \ldots, Y_{n_O}$ lze potom již snadno získat disjunkce booleovských pravidel tvaru

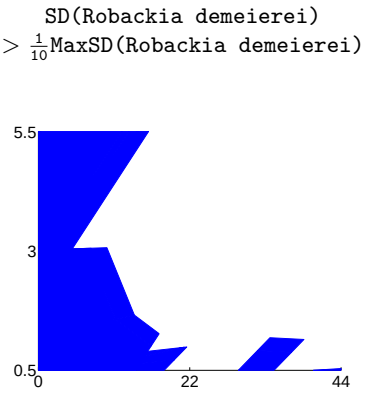
$$(X_1, \ldots, X_{n_I}) \in P \rightarrow \bigwedge_{j=1}^{n_O} Y_j \in V_j, \tag{12}$$

kde P je nějaký mnohostěn ve vstupním prostoru sítě a $V_1, \ldots, V_{n_O}$ jsou podmnožiny oborů hodnot proměnných $Y_1, \ldots, Y_{n_O}$ (typicky nějaké intervaly). Pravidla tvaru (12) se díky linearitě mnohostěnu snadno ukládají v paměti a dále matematicky zpracovávají, jejich srozumitelnosti je však na překážku obtížná interpretace formule $(X_1, \ldots, X_{n_I}) \in P$ v případě obecného mnohostěnu P (Obr. 3), zvláště je-li dimenze n_I vysoká. Proto se obvykle po získání pravidla tvaru (12) snažíme přidat další krok, totiž náhradu mnohostěnu P n_I-rozměrným kvádrem H (Obr. 4). Tím přejde (12) v pravidlo

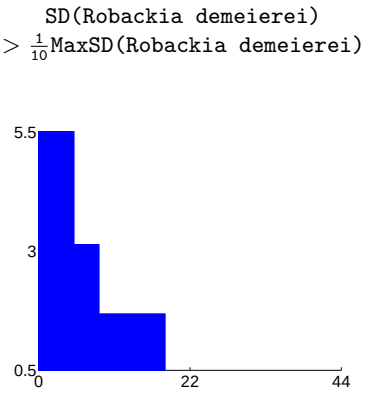
$$\bigwedge_{i=1}^{n_O} X_i \in V_i' \rightarrow \bigwedge_{j=1}^{n_O} Y_j \in V_j, \tag{13}$$

kde $V_1', \ldots, V_{n_I}'$ jsou podmnožiny oborů hodnot proměnných $X_1, \ldots, X_{n_I}$. Pravidlo tvaru (13) je již snadno srozumitelné, a nadto systém pravidel, který takto získáme, je v disjunktivní normální formě. Zda si můžeme dovolit mnohostěn P kvádrem H opravdu nahradit, závisí na naší nespokojenosti s body, které buď leží v P ale neleží v H, nebo naopak. Za značně obecných předpokladů lze tuto nespokojenost vyjádřit jako $\mu_P(P \Delta H)$, kde μ_P je monotonní nezáporná míra na $\Re^{n_I}$, která může záviset na P, a $P \Delta H$ označuje symetrický rozdíl množin P a H. V implementaci popsané v [42]) se jako míry μ_P používá empirické podmíněné pravděpodobnosti dané daty a podmíněné mnohostěnem P, a pro nahražení P kvádrem H je třeba splnění následujících tří podmínek:
(i) $\mu_P(P \Delta H)$ nepřekračuje předem danou toleranci ε,
(ii) $\mu_P(P \Delta H)$ má v H minimum na množině $\{\mu_P(P \Delta H') : H'$ je kvádr v $\Re^{n_i}\}$,
(iii) P má neprázdný průnik s daty.

V souvislosti s právě zmíněnou metodou stojí za zmínku, že pomocí po částech lineárních perceptronů lze získávat z dat rovněž tvrzení jedné z hlavních fuzzy logik, Łukasiewiczovy logiky, byť dosud navržené algoritmy mají dvojitě exponenciální složitost, a tudíž nejsou prakticky použitelné [43].



Obr. 3. Dvojrozměrný řez sjednocením mnohostěnů ve vstupním prostoru sítě odpovídajících antecedentům (levým stranám) pravidel nalezených k zadanému konsekventu (pravé straně).



Obr. 4. Dvojrozměrná projekce kvádrů ve vstupním prostoru sítě, kterými bylo možno nahradit na základě výše uvedených podmínek (i) – (iii) některé z mnohostěnů na Obr. 3.

6 Závěrečné shrnutí

Tento příspěvek se snažil naznačit význam metod získávání pravidel z dat jakožto velmi rozšířeného typu metod dobývání znalostí z dat. Byla podána obecná charakteristika metod získávání pravidel z dat, která byla ilustrována na příkladu klasické metody Guha. Ve druhé části příspěvku bylo poukázáno na specifický charakter metod získávání pravidel z dat založených na umělých neuronových sítích. Rovněž tento typ metod byl ilustrován na konkrétním příkladu.

Příspěvek byl připraven v rámci projektu 1ET100366419 "Intelligent Models, Algorithms, Methods and Tools for the Semantic Web Realization" programu Informační Společnost, a za podpory ústavního výzkumného záměru AV0Z10300504.

Reference

1. Khoo L., S.B.T., Zhai L., A Rough-Set Approach for Classification and Rule Induction. Internationa Journal of Advanced Manufacturing Technology 15, 1999, 438–444
2. Mak B., Munakata T., Rule Extraction from Expert Heuristics: A Comparative Study of Rough Sets with Neural Networks and id3. European Journal of Operational Research 136, 2002, 212–229
3. Hájek P., Havránek T., Mechanizing Hypothesis Formation. Springer Verlag, Berlin, 1978
4. Holeňa M.: Fuzzy Hypotheses for GUHA Implications. Fuzzy Sets and Systems 98, 1998, 101–125

5. Holeňa M., Fuzzy Hypotheses Testing in the Framework of Fuzzy Logic. Fuzzy Sets and Systems 145, 2004, 229–252
6. Agrawal R., Mannila H., Srikant R., Toivonen H., Verkamo A.: Fast Discovery of Association Rules. In Fayyad U., Piatetsky-Shapiro G., Smyth P., Uthurusamy R., eds.: Advances in Knowledge Discovery and Data Mining. AAAI Press, Menlo Park, 1996, 307–328
7. Korn F., Labrinidis A., Kotidis Y., Faloutsos C., Quantifiable Data Mining Using Ration Rules. VLDB Journal 8, 2000, 254–266
8. Srikant R., Vu Q., Agrawal R., Mining Association Rules with Item Constraints. In: Proceedings of the Third International Conference on Knowledge Discovery and Data Mining KDD-97, 1997
9. Zaki M., Parathasarathy S., Ogihara M., Li W., New Parallel Algorithms for Fast Discovery of Association Rules. Data Mining and Knowledge Discovery **1**, 1997, 343–373
10. Hájek P., Holeňa M., Formal Logics of Discovery and Hypothesis Formation by Machine. Theoretical Computer Science 292, 2003, 345–357
11. Michalski R.: Knowledge Acquisition through Conceptual Clustering: A Theoretical Framework and Algorithm for Partitioning Data into Cojunctive Concepts. International Journal of Policy Analysis and Information Systems 4, 1980, 219–243
12. Michalski R., Kaufman K., Learning Patterns in Noisy Data. In Paliouras G., Karkaletsis V., Spyropoulos C., eds.: Machine Learning and Its Applications. Lecture Notes in Computer Science 2049, Springer Verlag, New York, 2001, 22–38
13. Clark P., Boswell R., Rule Induction with cn2: Some Recent Improvements. In Kodratoff Y., ed.: Machine Learning – EWSL-91. Lecture Notes in Computer Science 482, Springer Verlag, New York, 1991, 151–163
14. Breiman L., Friedman J., Olshen R., Stone C., Classification and Regression Trees. Wadsworth, Belmont, 1984
15. Hasti T., Tibshirani R., Friedman J., The Elements of Statistical Learning. Springer Verlag, New York, 2001
16. Quinlan J.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, San Francisco, 1992
17. Hájek P., Metamathematics of Fuzzy Logic. Kluwer Academic Publishers, Dordrecht, 1998
18. Jang J., ANFIS: Adaptive-Network-Based Fuzzy Inference System. IEEE Transactions on Systems, Man, and Cybernetics 23, 1993, 665–685
19. Jang J., Sun C., Neuro-Fuzzy Modeling and Control. The Proceedings of the IEEE 83, 1995, 378–406
20. Nauck D., Kruse R., NEFCLASS-X: A Neuro-Fuzzy Tool to Build Readable Fuzzy Classifiers. BT Technology Journal 3, 1998, 180–192
21. Nauck D., Fuzzy Data Analysis with NEFCLASS. International Journal of Approximate Reasoning 32, 2002, 103–130
22. De Raedt L., Interactive Theory Revision: An Inductive Logic Programming Approach. Academic Press, London, 1992.
23. Muggleton S., Inductive Logic Programming. Academic Press, London, 1992
24. Freitas A., Data Mining and Knowledge Discovery with Evolutionary Algorithms. Springer Verlag, Berlin, 2002
25. Wong M., Leung K., Data Mining Using Grammar Based Genetic Programming and Applications. Kluwer Academic Publishers, Dordrecht, 2000
26. Hornik K., Approximation Capabilities of Multilayer Neural Networks. Neural Networks 4, 1991, 251–257

27. Hornik K., Stinchcombe M., White H., Auer P., Degree of Approximation Results for Feedforward Networks Approximating Unknown Mappings and Their Derivatives. Neural Computation 6, 1994, 1262–1275
28. Kůrková V., Kolmogorov's Theorem and Multilayer Neural Networks. Neural Networks 5, 1992, 501–506
29. Kůrková V., Rates of Approximation by Neural Networks. In Sinčák P., Vasčák J., eds.: Quo Vadis Computational Intelligence? Springer Verlag, Berlin, 2000, 23–26
30. Alexander J., Mozer M., Template-Based Procedures for Neural Network Interpretation. Neural Networks 12, 1999, 479–498
31. Bologna G., Symbolic Rule Extraction from the DIMPL Neural Network. In Wermter S., Sun R., eds.: Hybrid Neural Systems. Springer Verlag, Heidelberg, 2000, 241–255
32. Chen J., Liu J., Using Mixture Principal Component Analysis Networks to Extract Fuzzy Rules from Data. Industrial and Engineering Chemistry Research 39, 2000, 2355–2367
33. d'Avila Garcez A., Broda K., Gabbay D., Symbolic Knowledge Extraction from Artificial Neural Networks: A Sound Approach. Artificial Intelligence 125, 2001, 155–207
34. Duch W., Adamczak R., Grabczewski K., A New Methodology of Extraction, Optimization and Application of Crisp and Fuzzy Logical Rules. IEEE Transactions on Neural Networks 11, 2000, 277–306
35. Finn G.: Learning Fuzzy Rules from Data. Neural Computing & Applications 8, 1999, 9–24
36. Ishikawa M., Rule Extraction by Successive Regularization. Neural Networks 13, 2000, 1171–1183
37. Mitra S., De R., Pal S., Knowledge-Based Fuzzy MLP for Classification and Rule Generation. IEEE Transactions on Neural Networks 8, 1997, 1338–1350
38. Tsukimoto H., Extracting Rules from Trained Neural Networks. IEEE Transactions on Neural Networks 11, 2000, 333–389
39. Andrews R., Diederich J., Tickle A., Survey and Critique of Techniques for Extracting Rules from Trained Artificical Neural Networks. Knowledge Based Systems 8, 1995, 378–389
40. Mitra S., Hayashi Y., Neuro-Fuzzy Rule Generation: Survey in Soft Computing Framework. IEEE Transactions on Neural Networks, 11, 2000, 748–768
41. Tickle A., Andrews R., Golea M., Diederich J., The Truth Will Come to Light: Directions and Challenges in Extracting Rules from Trained Artificial Neural Networks. IEEE Transactions on Neural Networks 9, 1998, 1057–1068
42. Holeňa M., Extraction of Logical Rules from Data by Means of Piecewise-Linear Neural Networks. In: Proceedings of the 5th International Conference on Discovery Science. Springer Verlag, Berlin, 2002, 192–205
43. Holeňa M., Extraction of Fuzzy Logic Rules from Data by Means of Artificial Neural Networks. Kybernetika 41, 2005, 297–314

WORK IN PROGRESS

Rámec pre prezentáciu informácií reprezentovaných ontológiou*

Mária Bieliková, Vladimír Grlický, and Jaroslav Kuruc

Ústav informatiky a softvérového inžinierstva
Fakulta informatiky a informačných technológií
Slovenská technická univerzita v Bratislave
Ilkovičova 3, 842 16 Bratislava
{bielik,grlicky,kuruc}@fiit.stuba.sk

Abstrakt V príspevku opisujeme rámec pre prezentáciu informácií a znalostí reprezentovaných ontológiou, ktorého cieľom je poskytnutie prostriedkov pre definovanie spôsobu prezentácie spomínaných informácií. Opisovaný rámec je založený na architektúre dátovodov a filtrov, ktorá poskytuje flexibilný spôsob definovania transformácie dát medzi ich rôznymi reprezentáciami. Rámec uvažuje dva typy transformácií. Prvým typom sú tranformácie z jedného RDF modelu do iného (RDF → RDF), druhým typom sú transformácie z RDF modelu do výslednej podoby určenej na prezentáciu (RDF → XML), založené na použití ontológie Fresnel. Ontológia Fresnel predstavuje univerzálny slovník na špecifikovanie častí RDF modelu určených na prezentáciu a tiež na definovanie štýlu prezentovaných častí RDF modelu pomocou existujúcich formátovacích jazykov.

1 Úvod

Hlavným cieľom vytvorenia webu so sémantikou je, aby web obsahoval takú sieť informácií, ktorá by bola spracovateľná strojmi, na rozdiel od súčasného stavu, v ktorom je obsah webu zrozumiteľný v zásade len pre človeka. Doplnením metadát sa však tieto informácie stávajú ťažšie zrozumiteľné pre človeka, a preto sú potrebné mechanizmy efektívnej prezentácie informácií a znalostí reprezentovaných technológiami webu so sémantikou.

RDF, RDFS a OWL ako základné elementy webu so sémantikou umožňujú ľuďom vytvárať ontológie s cieľom opisu špeciálnych vlastností informačných zdrojov metadátami. Metadáta sú dôležité pre strojové spracovanie dát dostupných v rôznych formách na webe. Tým, že vznikajú dva prepojené svety: dáta, informácie a znalosti, ktoré sú určené na prezentáciu koncovému používateľovi a dáta o týchto dátach, informáciách a znalostiach určené najmä pre spracovanie a ďalšie automatizované odvodzovanie, vzniká potreba primerane tieto svety prezentovať používateľovi. Prezentácia informačných zdrojov je známa

* Táto práca bola podporovaná Vedeckou grantovou agentúrou, grant č. VG1/ 0162/03 a Agentúrou na podporu vedy a techniky na základe Zmluvy č. APVT-20-007104.

požiadavka, ktorá sa v súčasnosti rozširuje o možnosti adaptívnej prezentácie, t.j. prezentácie závislej od kontextu (napr. ciele, preferencie používateľa alebo vlastnosti prostredia, v ktorom sa používateľ nachádza) [2]. Prezentácia metadát je požiadavkou novou. Ukazuje sa, že je dôležité, aby metadátam "rozumeli" nielen stroje, ale aj ľudia, keďže práve ľudia definujú spôsob ich vytvárania, podieľajú sa na udržiavaní a práve ľudia sú tí, ktorí metadátam dodávajú sémantiku, napr. tým, že sa dohodnú na význame pojmov a takto sa v aplikáciach, ktoré človek vytvoril používajú.

Súčasné možnosti interakcie vo webových prehliadačoch sú značne obmedzené. Každá sofistikovanejšia prezentácia musí byť vytvorená explicitne. Našim cieľom je využitie metadát tak, aby sa dali jednoducho definovať rôzne pohľady na prezentované dáta, informácie a znalosti spolu s efektívnou navigáciou v nich. V príspevku uvádzame návrh prezentačného rámca, ktorý umožňuje vytvorenie efektívnej prezentácie informačných zdrojov opísaných prostriedkami webu so sémantikou. Efektívnosť prezentácie spočíva jednak vo využití poznatkov z oblasti adaptívnych webových hypermediálnych systémov a tiež v umožnení definovania rôznych pohľadov s využitím toho, že informačné zdroje sú opísané ontológiou. Opis ontológiou nám umožňuje s využitím jedného opisu vytvárať rôzne prezentácie, ktoré definujú rôzne prepojenia medzi informačnými zdrojmi. Tým, že navrhnutý rámec je založený na architektúre dátovodov a filtrov, zároveň poskytujeme flexibilný spôsob definovania transformácie dát medzi ich rôznymi reprezentáciami.

2 Architektúra rámca pre prezentáciu informácií

Architektúra rámca pre prezentáciu informácií vychádza z architektonického štýlu dátovody a filtre, ktorý je založený na sekvenčnom spracovaní dát pomocou filtrov, pričom prúdenie dát medzi nimi zabezpečujú dátovody [1]. Dáta vstupujúce do filtra sa spracujú a následne poskytnú ďalšiemu filtru. Výstup jedného filtra je tak vstupom pre iný filter. Takto sa vytvorí zreťazenie filtrov, ktoré realizuje požadovanú transformáciu od vstupných dát až po dáta v požadovanom výstupnom formáte.

Aj keď názov "filter" evokuje, že pri spracovaní dát ide o ich filtrovanie (redukciu), v skutočnosti ide o transformáciu týchto dát. Okrem transformácií, kedy dochádza k redukcii vstupných dát (ich filtrovaniu) sú možné aj transformácie, pri ktorých dochádza k ich doplneniu. Môže ísť dokonca o produkciu úplne nových dát, pri ktorej sa využijú vstupné dáta. V najjednoduchšom prípade ide o filtre, ktoré majú práve jeden vstupný dátovod a práve jeden výstupný dátovod. Možné sú však aj prípady, kedy filter agreguje vstupné dáta z viacerých dátovodov, prípadne produkuje dáta pre viacero dátovodov.

Výhodou architektúry dátovodov a filtrov je aj flexibilita definície zreťazení. Pri zachovaní vstupného/výstupného formátu dát možno jednoducho presmerovať dátovod z/do iného filtra, prípadne vložiť do zreťazenia medzi dva filtre ďalšie. Okrem primárneho vstupu z dátovodu môžu mať filtre prístup aj k iným dátam, ktoré môžu byť statické (zapísané v súbore na disku, v pamäti a pod.),

prípadne dynamické (kedy sú výsledkom iného zret'azenia filtrov). Možnosť nahradenia statických zdrojov dát zdrojmi, ktoré ich produkujú dynamicky, dáva široké možnosti d'alšieho rozširovania existujúcich definícií zret'azení.

Vzhľadom na spomenuté vlastnosti sme zvolili túto architektúru ako základ rámca pre prezentovanie informácií reprezentovaných ontológiou. Navrhnutý rámec poskytuje prostriedky pre tvorbu webových aplikácií, ktorých cieľom je prezentovanie informačných zdrojov opísaných ontológiou. Ide o sadu filtrov umožňujúcich transformáciu dát z ontologickej reprezentácie (RDF/OWL) do požadovanej cieľovej reprezentácie (napr. XHTML[1], SVG[2], FO[3] či PDF).

Počiatočnými dátami zret'azenia sú dáta v požiadavke protokolu HTTP. Tieto slúžia na špecifikovanie časti RDF modelu určenej na prezentáciu. Vybraná časť sa následne transformuje sériou filtrov a nakoniec vo zvolenom formáte sa zašle ako odpoveď protokolu HTTP. Filtre zaradené do zret'azenia na začiatok, ktorých úlohou je produkovať dáta pre d'alšie filtre v zret'azení, sa nazývajú *generátory*. Filtre, ktoré vstupné dáta transformujú a poskytujú d'alším filtrom, sú zaradené uprostred zret'azenia. Nazývajú sa *transformátory*. Filtre, ktoré poskytujú výstupné dáta, sú zaradené na koniec zret'azenia a nazývajú sa *serializátory*. Ukážka definície zret'azenia je naznačená na obr. 1.

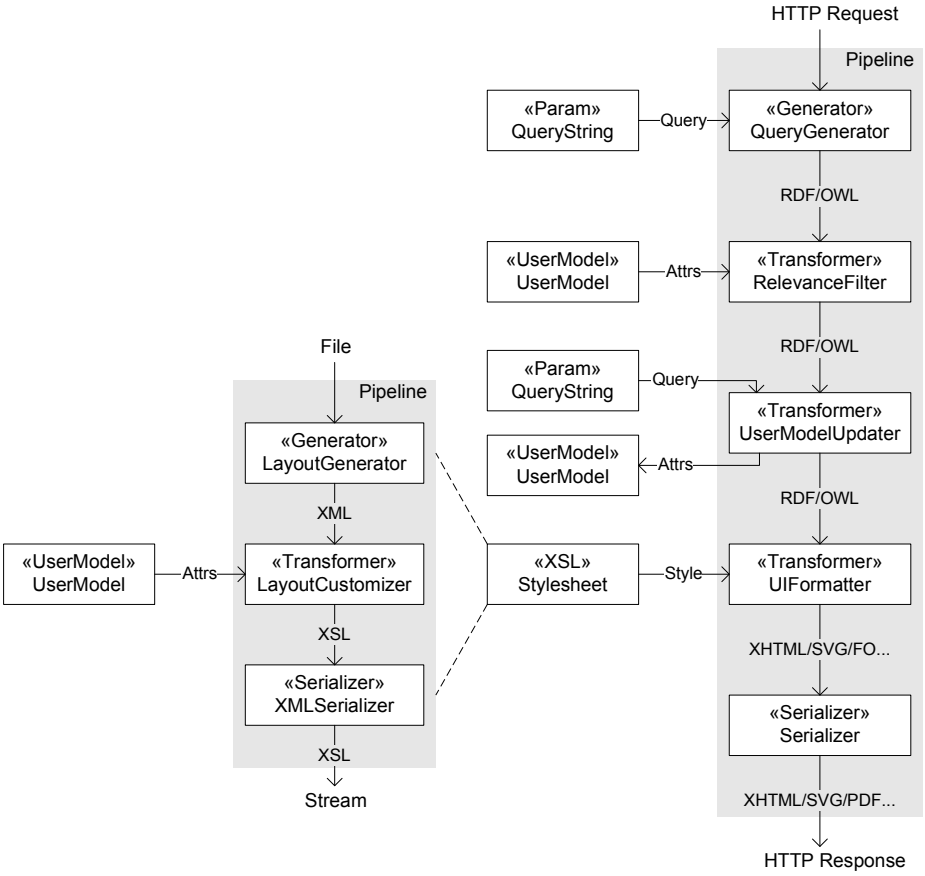
Navrhnutý rámec pre prezentovanie informačných zdrojov poskytuje dva typy transformácií:

- transformácie z jedného RDF modelu do iného (RDF → RDF)
- transformácie z RDF modelu do cieľovej reprezentácie určenej na prezentáciu, najčastejšie formátu založeného na jazyku XML (RDF → XML).

Transformácie typu RDF → RDF predpripravujú vybranú časť RDF modelu určenú na prezentáciu tým, že prezentované zdroje zoraďujú, vyberajú vlastnosti zdrojov, ktoré budú prezentované a pod. Špecifikácia výberu môže byť generická pre všetky požiadavky. Možná je aj personalizácia, kedy napr. výber a zoradenie môže ovplyvňovať model používateľa, ktorý pracuje so systémom. Do tejto skupiny patria aj filtre, ktoré vstupné dáta nijako nemodifikujú a len ich posúvajú d'alej, pričom ich cieľom je modifikácia stavu systému, napr. aktualizácia modelu používateľa. Transformácie typu RDF → XML určujú spôsob prezentácie vybraného RDF modelu tým, že určujú napr. spôsob rozmiestnenia prvkov alebo štýly použité pri prezentácii.

Obr. 1 naznačuje príklad spracovania požiadavky protokolu HTTP (`HTTP Request`). Táto požiadavka je vstupom pre generátor (`QueryGenerator`), ktorý na základe nej parametrizuje výraz určujúci, ktorá časť RDF modelu bude prezentovaná, a na základe nej produkuje dáta v ontologickom tvare (RDF/OWL). Tieto sú vstupom pre transformátor (`RelevanceFilter`), ktorého úlohou je odfiltrovať iba také zdroje RDF modelu, ktoré sú vzhľadom na model používateľa, ktorý so systémom pracuje, relevantné. Následne sa dáta posunú transformátoru (`UserModelUpdater`), ktorého úlohou je aktualizácia modelu používateľa, napr. zaznamenaním informácie o tom, ktoré zdroje sa používateľovi

[1] Extensible HyperText Markup Language, http://www.w3.org/TR/xhtml1/
[2] Scalable Vector Graphics, http://www.w3.org/Graphics/SVG/
[3] Extensible Stylesheet Language, http://www.w3.org/TR/xsl/



Obr. 1. Ukážka definície zret'azenia.

zobrazujú. V tomto prípade nie sú vstupné dáta nijako modifikované. Spomínané transformátory sú typu RDF → RDF, teda aj vstupné aj výstupné dáta sú v reprezentácii RDF/OWL, čo znamená, že je určený obsah, nie však forma prezentácie. Ďalším transformátorom v poradí je transformátor (`UIFormatter`), ktorého úlohou je pretransformovat' dáta do výsledného prezentovateľného formátu, napr. XHTML, SVG, či FO. Je to transformátor typu RDF → XML, ktorý využíva transformačnú šablónu vo formáte XSL, ktorá je dynamicky generovaná iným zret'azením. Toto pri jej generovaní využíva opät' model používateľa. Štandardné rozmiestnenie, ktoré sa generuje zo súboru uloženého na disku (`LayoutGenerator`) upraví transformátor (`LayoutCustomizer`) a následne poskytne už spomínanému transformátoru, ktorý zabezpečuje rozmiestnenie dát v produkovanom dokumente (výslednej prezentácii). Posledným článkom v zret'azení je serializátor, ktorého úlohou je nastavenie príslušných hlavičiek v odpovedi protokolu HTTP (`HTTP Response`), prípadne prevod z textovej podoby dokumentu do jej binárnej podoby, ako je to naznačené na príklade konverzie reprezentácie FO do reprezentácie PDF.

Uvedený príklad naznačuje vo všeobecnosti možnosti definovania zreťazení, a teda transformácie dát z ontologickej reprezentácie do reprezentácie určenej na prezentáciu. Pri špecifikovaní výberu RDF modelu, ako aj štýlov použitých pri prezentácii je možné použitie univerzálnej prezentačnej ontológie, opísanej v nasledujúcej časti.

3 Prezentačná ontológia

Pri prezentácii obsahu webu so sémantikou spôsobom, ktorý by bol zrozumiteľný aj pre človeka treba riešiť dva základné problémy a to špecifikovať

- *ktoré* informácie nachádzajúce sa v RDF grafe majú byť prezentované
- *ako* majú byť tieto informácie zobrazené.

V súčasnosti však každý prehliadač alebo nástroj na vizualizáciu RDF modelu spolieha na vlastný (ad hoc) mechanizmus, ktorým sa snaží uvedené problémy riešiť. Problém s takýmto prístupom je nemožnosť zdieľať informácie a znalosti o prezentácii RDF modelu medzi rôznymi aplikáciami. Riešenie tohto problému spočíva v definovaní spôsobu prezentácie ako aj výberu položiek na prezentáciu, tak že ho bude možné zdieľať. Túto požiadavku spĺňa prezentačná ontológia Fresnel [3]. Predstavuje univerzálny slovník, ktorý možno použiť na špecifikovanie častí RDF modelu určených na prezentáciu, a tiež na definovanie štýlu prezentovaných častí RDF modelu pomocou existujúcich formátovacích jazykov.

3.1 Základné koncepty

Základnými konceptmi ontológie Fresnel[4] sú šošovky a štýly. *Šošovky* (angl. lenses) definujú, ktoré vlastnosti RDF zdroja alebo množiny zdrojov majú byť zobrazené a ako budú tieto vlastnosti usporiadané. *Štýly* (angl. styles) určujú ako sa majú zdroje a vlastnosti zobraziť špecifikovaním atribútov, ktoré definujú vzhľad týchto vlastností, pri použití existujúcich formátovacích jazykov (napr. CSS, SVG).

Ontológia Fresnel bola navrhnutá ako čisto deklaratívny jazyk. Ďalšími cieľmi pri jej návrhu bolo vytvoriť ontológiu, ktorá bude:

- aplikovateľná na rôzne paradigmy zobrazovania RDF modelu (napr. grafovo orientovaná, textovo orientovaná),

[4] Ako názov prezentačnej ontológie jej autori zvolili priezvisko francúzskeho fyzika Augustina Jeana Fresnela (čítať Fre'-nel alebo Fray'nel), ktorý sa zaoberal odrazom a lomom svetla, a okrem iného sa preslávil vynájdením sústavy špeciálne zostavených optických šošoviek, ktoré koncentrujú intenzívne svetlo do jedného smeru (tzv. Fresnelove šošovky). Jedným z krokov generovania prezentácie pomocou opisovanej prezentačnej ontológie je definovanie pohľadov na časti RDF grafu a ich vzájomné skladanie. A práve analógia pohľadov a Fresnelových šošoviek inšpirovala autorov ontológie k jej pomenovaniu podľa Fresnela (a premenovaniu pohľadov na šošovky).

– použiteľná pre generovanie výstupu do rôznych formátov (XHTML, SVG, PDF, obyčajný text a iné),
– postavená na existujúcich webových technológiách,
– rozšíriteľná o špecifické potreby konkrétnych aplikácií,
– ľahko pochopiteľná a použiteľná.

Ontológia Fresnel striktne dodržiava oddelenie výberu dát od definovania ich vzhľadu čo priamo ovplyvňuje proces generovania prezentácie. Proces generovania prezentácie pozostáva z troch krokov (podľa [3]):

1. *Výber:* časti RDF grafu, ktoré chceme zobraziť, sa vyberú a usporiadajú pomocou šošovky alebo množiny šošoviek ontológie Fresnel. Výsledkom tohto kroku je usporiadaný strom RDF uzlov, ktorý zatiaľ neobsahuje žiadne informácie o definovaní vzhľadu týchto uzlov.
2. *Definovanie vzhľadu:* na strom RDF uzlov (výsledok kroku 1) sa aplikuje množina štýlov ontológie Fresnel, a to pridaním množiny formátovacích inštrukcií pre každý RDF uzol.
3. *Generovanie výstupu:* výsledok kroku 2 (strom RDF uzlov obohatených o formátovacie inštrukcie) je transformovaný do príslušného výstupného formátu.

3.2 Definovanie vzhľadu

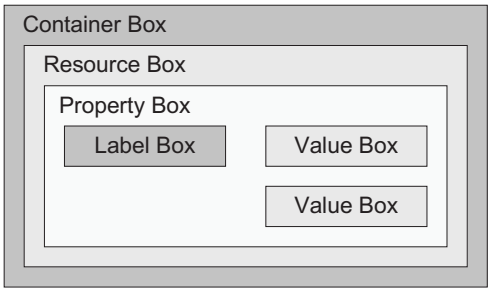
Na definovanie vzhľadu RDF uzlov v strome ontológia Fresnel používa formátovacie jazyky CSS[5] a SVG[2]. Formátovacie inštrukcie CSS slúžia na definovanie farieb, okrajov, fontov a iných dekoratívnych prvkov. Formátovacie inštrukcie SVG sa uplatnia najmä v grafovo orientovanej paradigme zobrazovania RDF modelu, kde pomocou nich možno určovať vzhľad štetca.

Oba typy inštrukcií možno aplikovať na ľubovoľný z piatich elementov modelu vzhľadu ontológie Fresnel (označovaného ako *box model*). Model vzhľadu (pozri obr. 2) pozostáva z piatich elementov:

– *kontajner* (angl. container) – vzhľad oblasti, ktorá uzatvára všetky aktuálne zobrazované inštancie,
– *zdroj* (resource) – vzhľad oblasti pre všetky vlastnosti jedného zdroja,
– *vlastnosť* (property) – vzhľad oblasti pre jednu vlastnosť určitého zdroja,
– *menovka* (label) – vzhľad menovky nejakej vlastnosti,
– *hodnota* (value) – vzhľad hodnoty nejakej vlastnosti.

Konkrétny prehliadač RDF modelu interpretuje vrstvový model ontológie Fresnel v závislosti od paradigmy zobrazovania. Ak napríklad uvažujeme prezentáciu RDF modelu pomocou grafových diagramov (ako v grafickom editore RDF modelu IsaViz[6]), potom sa element *vlastnosť* interpretuje ako oblúk (orientovaná čiara), ktorého vzhľad môže byť určený pomocou inštrukcií na definovanie štýlu štetca, ktoré poskytuje SVG. V textovo orientovaných prehliadačoch RDF modelu (napr. Longwell[7]) sa tento element interpretuje ako obdĺžnikový výsek obrazovky.

[5] Cascading Style Sheets, http://www.w3.org/Style/CSS/
[6] IsaViz: A Visual Authoring Tool for RDF, http://www.w3.org/2001/11/IsaViz/
[7] Longwell, http://simile.mit.edu/longwell/



Obr. 2. Model vzhľadu v ontológii Fresnel [3].

3.3 Príklad použitia prezentačnej ontológie

Na ukážku použitia prezentačnej ontológie Fresnel sme vybrali zobrazenie časti jednej inštancie pracovnej ponuky (`ioJobOffer545058`) triedy `pp:JobOffer` z ontológie pracovných príležitostí vytváranej v rámci projektu štátneho programu výskumu a vývoja NAZOU (Nástroje pre získavanie, organizovanie a udržovanie znalostí v prostredí heterogénnych informačných zdrojov). Neúplné definovanie tejto inštancie v notácii N3 je uvedené v nasledujúcom príklade:

```
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#>
@prefix fresnel: <http://www.w3.org/2004/09/fresnel#>
@prefix pp:      <http://www.fiit.stuba.sk/ontology/nazou/pp/0.1/>
@prefix :        <#>

:ioJobOffer545058
     rdf:type pp:JobOffer ;
     pp:id 545058 ;
     pp:title "Director of Business Development"@en ;
     pp:summary "Develop, plan and be responsible for Cast Irons channel
                 and business development strategy in the US. Acquire
                 and develop new partnerships with Resellers, System
                 Integrators and ISVs to deliver customer leads,
                 brand/benefit awareness, budgeted IT projects and
                 programs to drive either direct or indirect revenue
                 for Cast Iron..."@en ;
     pp:type pp:permanent ;
     pp:status pp:fullTime ;
     pp:hoursPerWeek 40 ;
     pp:hasMinSalary pp:negotiable ;
     pp:hasDutyLocation [ rdf:type Location ;
                          pp:text "Mountain View, CA" ] .
```

Po vytvorení ontológie pracovných príležitostí (doménovej ontológie) a definovaní jej inštancií treba v d'alšom kroku špecifikovat', ktoré inštancie sa vyberú na prezentáciu a ako budú tieto inštancie zobrazené. Špecifikácia zobrazenia inštancií doménovej ontológie sa uskutočňuje pomocou pojmov prezentačnej ontológie Fresnel. Šošovky zabezpečia výber tých vlastností tried, ktoré chceme zobrazit', tj. vlastnosti: názov pracovnej ponuky (`pp:title`), miesto vykonávania práce (`pp:hasDutyLocation`), typ pracovného pomeru (`pp:type`), úväzok (`pp:status`), ponúkaný plat (`pp:hasMinSalary`) a stručný opis ponuky (`pp:summary`). Tieto vlastnosti by sa mali v zozname pracovných ponúk (`:JobOfferInListLens`) zobrazit' v uvedenom poradí. Pomocou štýlov sa následne určí vzhľad pre vybrané triedy a vlastnosti.

V nasledujúcom príklade sú v notácii N3 uvedené definície šošoviek a ich štýlov pre pracovnú ponuku reprezentovanú triedou `pp:JobOffer`:

```
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#>
@prefix fresnel: <http://www.w3.org/2004/09/fresnel#>
@prefix pp:      <http://www.fiit.stuba.sk/ontology/nazou/pp/0.1/>
@prefix :        <#>
# ------------------------------------------------------------------------
# Šošovky ontológie Fresnel pre ontológiu pracovných príležitostí
# ------------------------------------------------------------------------
:JobOfferInListLens
     rdf:type fresnel:Lens ;
     fresnel:lensDomain pp:JobOffer ;
     fresnel:group :JobOfferInListGroup ;
     fresnel:showProperties ( pp:title
                              [ rdf:type fresnel:PropertyDetails ;
                                fresnel:property pp:hasDutyLocation ;
                                fresnel:sublens :LocationInListLens ] ;
                              pp:type
                              pp:status
                              pp:hasMinSalary
                              pp:summary ) .
:LocationInListLens
     rdf:type fresnel:Lens ;
     fresnel:group :JobOfferInListGroup ;
     fresnel:showProperties pp:text .
# ------------------------------------------------------------------------
# Štýly ontológie Fresnel pre ontológiu pracovných príležitostí
# ------------------------------------------------------------------------
:JobOfferInListGroup
     rdf:type fresnel:Group ;
     fresnel:stylesheetLink <http://www.fiit.stuba.sk/master.css> ;
     fresnel:containerStyle "offers" .
```

Nasledujúci príklad je ukážkou možného výstupu generovania prezentácie vybranej pracovnej ponuky s využitím prezentačnej ontológie Fresnel. Výstupným formátom je XHTML, na definovanie vzhľadu sú použité triedy štýlov CSS z externého zoznamu štýlov.

```
<li>
  <div class="title">
    <a href="#">Director of Business Development</a>
  </div>
  <div class="info">
    <em class="pic organization"
        title="Duty Location">Mark Logic Corporation, Mountain View, CA
    </em>
    <em class="pic jobType"
        title="Job Type">Permanent</em>
    <em class="pic jobStatus"
        title="Job Status">Full-Time (40 hrs per week)</em>
    <em class="pic salary"
        title="Salary">Negotiable</em>
  </div>
  <div class="desc">
    Develop, plan and be responsible for Cast Irons channel and business
    development strategy in the US. Acquire and develop new partnerships
    with Resellers, System Integrators and ISVs to deliver customer
    leads, brand/benefit awareness, budgeted IT projects and programs to
    drive either direct or indirect revenue for Cast Iron...
    <a href="#">&raquo; more</a>
  </div>
</li>
```

Konečný výsledok generovania prezentácie vybranej pracovnej ponuky tak, ako sa zobrazí v niektorom zo súčasných webových prehliadačov, je znázornený na obr. 3.

Obr. 3. Výsledok generovania prezentácie s využitím ontológie Fresnel tak, ako sa zobrazí vo webovom prehliadači.

4 Záver

V príspevku sme opísali návrh rámca pre prezentovanie informácií reprezentovaných ontológiou. Ide o univerzálny prostriedok pre vytváranie webových systémov s cieľom prezentovať informačné zdroje opísané pomocou metadát, ktorý je založený na architektúre dátovodov a filtrov. Výhodou tohto riešenia je jeho univerzálnosť, pretože ho môžeme použiť na prezentovanie ľubovoľnej doménovej ontológie (pri existencii jej mapovania na prezentačnú ontológiu). Výhodou je tiež znovupoužiteľnosť už vytvorených filtrov, ako aj vysoká flexibilita pri definícii zreťazení definujúcich transformácie medzi rôznymi reprezentáciami dát.

Medzi možné použitie tohto riešenia patrí vytvorenie univerzálneho prehliadača založeného na pohľadoch (angl. faceted browser), ktorý by pomocou klasifikačnej ontológie umožňoval výber dát z doménovej ontológie pomocou množiny reštrikcií. Príkladom takéhoto prístupu je projekt Swed[8] alebo tiež projekt Longwell[6]. Druhý menovaný projekt je tiež implementáciou založenou na ontológii Fresnel. Medzi ďalšie implementácie postavené na tejto ontológii patrí aj systém Arago [4].

V súčasnosti pracujeme na overení navrhnutého rámca vytvorením portálu pre ponuku pracovných príležitostí v rámci projektu štátneho programu výskumu a vývoja NAZOU. Jednotlivé navrhnuté filtre špecializujeme pre uvedenú doménu pracovných príležitostí s využitím metód klasifikácie a určovania relevantnej informácie [5], [7].

Referencie

1. Bass L., Clements P., Kazman R., Software Architecture in Practice. Second Edition, Addison-Wesley, 2003
2. Bieliková M., Presentation of Adaptive Hypermedia on the Web. In: Proc. of DATAKON, L. Popelínský (ed.), Brno, 2003, 72–91
3. Bizer Ch., Lee R., Pietriga E. (eds.), Fresnel – Display Vocabulary for RDF: User Manual, 2005, [online]. Dostupné na: <`http://simile.mit.edu/repository/fresnel/trunk/docs/manual/FresnelVocabulary.htm`>
4. Gassert H., Harth A., From Graph to GUI: Displaying RDF Data from the Web with Arago. Workshop on Scripting for the Semantic Web, Colocated with ESWC 2005, Heraklion, Greece, May 30, 2005
5. Horváth T., Krajči S., Lencses R., Vojtáš P., An ILP Model for a Graded Classification Problem. J. Kybernetika, 40, 2004, 317–332
6. Pietriga E. (eds.), Fresnel Selector Language for RDF (FSL), 2005. [online]. Dostupné na: <`http://simile.mit.edu/repository/fresnel/trunk/docs/manual/fsl.html`>
7. Polčicová G., Návrat P., Semantic Similarity in Content-based Filtering. In Proc. of ADBIS 2002 - Advances in Databases and Information Systems, Manolopoulos Y. and Návrat P. (eds.), Springer LNCS 2435, 2002, 80-85

[8] Semantic Web Enviromental Directory, http://www.swed.org.uk

Problémy recyklování systému automatického překladu*

Ondřej Bojar, Petr Homola, and Vladislav Kuboň

Ústav formální a aplikované lingvistiky, ÚFAL MFF UK
Malostranské náměstí 25, Praha 1, CZ-11800 Česká republika
{bojar,homola,vk}@ufal.mff.cuni.cz

Abstract. Tento článek se zabývá problémy, spojenými se snahou využít jednotlivých modulů systému automatického překladu mezi češtinou a ruštinou RUSLAN pro překlad do typově odlišného jazyka – do angličtiny. Tato snaha je motivována hypotézou, že přes veškeré problémy spojené s oživováním původního systému, je použití jeho klíčových modulů jednodušší než vývoj zcela nových modulů. Původní idea projektu počítala s použitím dvou klíčových částí sytému, syntaktické analýzy češtiny a jeho slovníku. V první části článek představuje systém RUSLAN, vyvíjený v letech 1985-90 pro překlad manuálů operačních systémů sálových počítačů z češtiny do ruštiny. V následující kapitole je rozebrána současná situace v oblasti automatického překladu z češtiny do angličtiny a je vysvětlena motivace pro recyklaci částí systému. Klíčovou část článku tvoří popis jednotlivých součástí systému, včetně nových modulů, kterými reagujeme na problémy vzniklé při recyklaci původního systému.

1 Úvod

V minulosti již bylo do nejrůznějších systémů automatického překladu investováno obrovské množství úsilí a prostředků. Výsledky však většinou ani zdaleka nebyly uspokojivé. Samozřejmě, existují výjimky, historie automatického překladu zná mnoho příkladů úspěšných systémů, ať již komerčních nebo experimentálních, ale počet těch systémů, které spolykaly obrovské množství lidské práce a finančních prostředků a které byly potichu opuštěny, když se ukázalo, že jejich výsledky nesplňují očekávání zadavatelů, je mnohem vyšší. Toto tvrzení se vztahuje zvláště na klasické překladové systémy založené na ručně vytvářených pravidlech, které obvykle vyžadují značné investice do vývoje gramatik a slovníků ještě předtím, než začnou produkovat výsledky v přijatelné kvalitě. Není náhodou, že vývoj takových systémů, jako je např. Systran či moskevský Etap, trval několik desetiletí - zkvalitňování takového systému je dlouhý proces, zahrnující zvětšování slovní zásoby, ladění gramatiky a testování celého systému.

* Výzkum popsaný v tomto článku je podporován grantem GAČR 405/03/0914 a zčásti také grantem GAUK 207-10/203330

Poslední desetiletí bylo svědkem několika pokusů o zvýšení kvality systémů automatického překladu prostřednictvím nových metod. Silný důraz na pravděpodobnostní a statistické ve zpracování přirozeného jazyka obecně se projevil i v oblasti automatického překladu. Začaly se objevovat hybridní systémy, překladatelskou veřejností začaly být široce přijímány systémy využívající tzv. překladovou paměť, objevily se úspěšné prototypy systémů na překlad mluvené řeči (i když zatím pouze s velmi omezenou tématickou doménou). Všechny tyto (a také mnoho dalších) trendy v posledních letech dosáhly celé řady povzbuzujících výsledků. Hledání nových metod a jejich využívání v praxi rozhodně posouvá celé odvětví automatického překladu kupředu, neměli bychom ale zapomínat na úsilí, investované do starších systémů. Opětovné použití, recyklování, celých systémů nebo jejich částí by mohlo pomoci snížit náklady na vývoj nových systémů, zvláště pokud se jedná o tzv. malé jazyky, pro které neexistuje takové množství nejrůznějších nástrojů, gramatik a slovníků jako např. pro angličtinu, japonštinu, němčinu nebo španělštinu. V tomto článku bychom rádi popsali jeden takový pokus o recyklování částí existujícího systému pro nový jazykový pár a problémy, na které jsme při práci narazili.

2 RUSLAN

Jeden ze systémů, které byly v tichosti opuštěny na počátku devadesátých let byl systém automatického překladu manuálů k sálovým počítačům, nazývaný RUSLAN [1]. Tento systém byl vyvíjen ve druhé polovině osmdesátých let jako společný projekt mezi MFF UK Praha a Výzkumným ústavem matematických strojů v Praze.

Tento systém využíval architekturu založenou na existenci fáze transferu. Jeho autoři zpočátku tvrdili, že pro tak příbuzné jazyky, jako je čeština a ruština, bude z důvodu syntaktické podobnosti obou jazyků fáze transferu nepatrná. Tento předpoklad se ukázal jako mylný, testování a ladění systému jednoznačně ukázalo, že množství rozdílů mezi oběma jazyky vede k celé řadě speciálních transferových pravidel.

Systém byl prakticky kompletně implementován v Q-systémech, formalismu vytvořeném Alainem Colmerauerem [2]. Q-systémy jsou v podstatě grafovým analyzátorem (chart parserem). Poprvé byly úspěšně použity v systému automatického překladu TAUM-METEO na montrealské univerzitě. Dovolují rozdělit gramatiku do modulů, které na sebe navazují tak, že výstupní graf předchozího modulu slouží jako vstup do modulu následujícího. Každý modul se skládá z množiny pravidel, která v podstatě popisují transformace stromových struktur. Ačkoli byl tento formalismus mnoha odborníky považován za zastaralý, Q-systémy prokázaly, že jsou spolehlivým nástrojem pro prakticky orientované projekty - interpret gramatiky napsané v Q-systémech byl dostatečně rychlý a spolehlivý i pro gramatiky pokrývající velkou část syntaxe obou jazyků.

Kromě gramatiky systém také spoléhal na sadu slovníků, které obsahovaly veškerá data využívaná jednotlivými moduly systému. Každá lexikální jednotka v hlavním (dvojjazyčném) slovníku neobsahovala pouze lexikálně-syntaktická

data (valenční rámce apod.), ale také množinu sémantických rysů, které se primárně používaly k řešení syntaktických víceznačností v průběhu syntaktické analýzy zdrojového jazyka (češtiny). Protože cílem celého systému byl překlad relativně omezené tématické domény, dosahovala velikost slovníku přibližně 8000 položek. Kromě nich však byl systém schopen překládat prostřednictvím speciálního modulu zvaného transdukční slovník dalších 2000 převážně technických a odborných termínů řecko-latinského původu pomocí algoritmu na přímou transkripci českých slovních forem do ruských [3].

Práce na systému RUSLAN byla ukončena v roce 1990 ve fázi konečného ladění a testování. Důvod ukončení prací na systému byl velmi jednoduchý – po roce 1989 nebyla po automatickém překladu z češtiny do ruštiny žádná poptávka.

3 Motivace

Poptávka po překladu z angličtiny nebo do angličtiny v letech po ukončení prací na systému RUSLAN dramaticky vzrostla. Na druhé straně se také podstatně zvětšilo množství metod, nástrojů a jazykových zdrojů pro automatický překlad. Pro češtinu byla vytvořena celá řada korpusů, z nichž nejznámějšími jsou morfologicky anotovaný Český národní korpus a syntakticky označkovaný Pražský závislostní korpus. V roce 2002 jsme zahájili práci na paralelním dvojjazyčném Pražském závislostním česko-anglickém korpusu (PCEDT) [4], který obsahuje asi polovinu textů z amerického korpusu PennTreebank 3, přeloženou do češtiny rodilými mluvčími. Dalším důležitým zdrojem dat, který byl pro češtinu také vyvinut v uplynulém období byl rozsáhlý morfologický slovník češtiny [5], který umožnil vysoce kvalitní morfologickou analýzu češtiny, která byla od té doby ověřena v bezpočtu komerčních aplikací a vědeckých projektů.

Uplynulé desetiletí bylo také svědkem vzniku několika stochastických analyzátorů češtiny, které by mohly být použity pro analýzu zdrojového jazyka v česko-anglických projektech. Ačkoli je kvalita výsledků těchto systémů o mnoho nižší než výsledky zveřejněné pro angličtinu – nejlepší výsledky, publikované Charniakem a Collinsem [6] se pohybují okolo 85% – je možné tyto analyzátory použít pro statistický překlad z češtiny.

Jedním z posledních stimulů pro naši snahu recyklovat části systému RUSLAN byly experimenty zahájené v roce 2001 M. Čmejrkem a J. Curínem. Jejich projekt (popsaný například v [7] se zaměřil na plně automatický statistický překlad, zahrnující stochastickou analýzu zdrojového textu na tektogramatickou (hloubkově syntaktickou) rovinu. Ačkoli tento projekt původně také měl využívat statistický transfer, novější experimenty jej nahradily modulem založeným z větší části na ručně vytvářených pravidlech. Tento posun od stochastického k ručně vytvořenému transferu otevřel otázku, zda by také nebylo lepší využít ručně psané gramatiky pro syntaktickou analýzu češtinu spíše než statistický analyzátor. Vyšší počet nesprávně přiřazených hran v závislostním stromě nemusí nutně znamenat horší analýzu. Pro systém automatického překladu je možná důležitější schopnost správně zanalyzovat souvislé části vstupních vět a tak umožnit správný překlad jednotlivých klauzí. Z našeho pohledu by systém

automatického překladu mohl sloužit jako jistý testovací nástroj, umožňující porovnání výsledků ručně vytvořených gramatik se statistickými analyzátory. Takové srovnání je obtížné, pokud chceme dostat opravdu objektivní výsledky. Důvodem je to, že standardní míry kvality analýzy jsou příliš spojeny s určitým typem dat a s určitým typem syntaktického značkování, použitého při budování syntaktických korpusů.

Hlavní motivací našeho česko-anglického experimentu bylo otestování několika hypotéz. Nejvýznamnější je hypotéza týkající se úrovně, na které má probíhat transfer. Vzhledem k odlišnostem mezi oběma jazyky není možné provádět transfer bezprostředně po morfologické analýze nebo po fázi mělké (částečné) syntaktické analýzy, jak to provádí systém Česílko, zaměřující se na překlad mezi blízce příbuznými (a podobnými) jazyky [viz [8]]. Na druhé straně je otázkou, zda typologické odlišnosti mezi češtinou a angličtinou opravňují použití transferu na tektogramatické (hloubkově syntaktické) úrovni, jako to bylo realizováno Cuřínem a Čmejrkem v jejich systému. Podle našeho názoru by transfer provedený na úrovni povrchové syntaxe mohl být vhodnější. Hlavním problémem transferu na tektogramatické (hloubkově syntaktické) úrovni je nízká kvalita výsledků analýzy na této úrovni, které jsou mnohem horší než výsledky stochastické analýzy na analytickou (povrchově syntaktickou) úroveň. Všechny výše uvedené výsledky patří analyzátorům zaměřujícím se na analytickou (povrchově syntaktickou) úroveň.

V neposlední řadě bylo naším záměrem vyvinout na pravidlech založený systém s minimálními možnými náklady, ať již metodou recyklace použitelných existujících modulů či snahou o využití (polo)automatických metod tam, kde to jen bude možné, se zaměřením na oblasti, kde by využití lidské práce bylo příliš drahé (například při budování rozsáhlého dvojjazyčného slovníku, viz níže.)

4 Jednotlivé moduly nového systému

Hlavním cílem našeho projektu bylo vyvinout experimentální systém automatického překladu pro překlad textů z korpusu PCEDT do angličtiny. Tento systém zkoumá, zda je možné prostřednictvím znovupoužití existujících zdrojů (gramatiky, slovníku) zkrátit čas potřebný k vývoji systému. Projekt také využívá dvojjazyčný paralelní korpus syntakticky označkovaných textů, i když nikoli jako přímý zdroj trénovacích dat, spíše jako dodatečný zdroj lingvistických dat užitečný zejména pro vývoj slovníku a pro testování celého systému.

4.1 Morfologická analýza

Kromě původního překladového systému RUSLAN a dvojjazyčného korpusu PCEDT můžeme také využít modul morfologické analýzy češtiny [5]. Tento modul pokrývá téměř celou českou slovní zásobu, s několika málo výjimkami (odhaduje se, že tato morfologická analýza je schopná rozeznat 800 000 lemat). Jediným problémem bylo jeho zapojení do systému – původní modul morfologické analýzy v systému RUSLAN byl velmi úzce svázán se slovníkem a s mor-

fologickým modulem. Nová morfologická analýza také používá odlišnou sadu značek než ta původní.

4.2 Dvojjazyčný slovník

Původně jsme předpokládali, že slovník použitý v systému RUSLAN budeme převádět a budeme se snažit využít velkého množství pečlivě zpracovaných lexikálně syntaktických údajů v něm obsažených. Tento předpoklad se ale ukázal jako příliš optimistický. Ačkoli jsou informace z původního slovníku pro modul syntaktické analýzy češtiny nesmírně cenné, rozhodli jsme se tyto informace obětovat. 8000 hesel původního slovníku tvoří příliš malou část slovní zásoby potřebné pro nový systém, než aby stálo zato kvůli těmto informacím obětovat konzistenci údajů v novém slovníku. Proto jsme se rozhodli tento slovník vytvořit znovu.

Budování česko-anglických překladových slovníků stálo již mnoho úsilí a je jistě vhodné raději znovu použít dostupné slovníky než budovat slovníky zcela nové. Pro češtinu a angličtinu jsou však dostupné pouze slovníky určené pro lidské uživatele (např. WinGED[1] nebo [9]), které trpí mnohými problémy:

- jedno překladové heslo[2] v některých případech kóduje více překladových variant současně
- cenné morfologické nebo i syntaktické informace jsou ve slovníku dostupné jen zřídka a navíc způsobem, který není nijak standardizován (Některá slovesná hesla např. obsahují nejen sloveso, ale též formy doplnění slovesa. Bohužel se způsob zápisu formy liší heslo od hesla, někde je uvedena jen předložka, někde předložka a klíčové slovo *(ně)koho* ap. Klíčová slova se navíc nijak formálně neliší od běžných slov, která jsou povinnou součástí hesla.)
- většina hesel neobsahuje vůbec morfologickou a syntaktickou informaci

Pro použití slovníku v systému strojového překladu je třeba tyto informace doplnit, jinak systém např. nebude vědět, zda pro překlad podstatného jména *stát* ve vstupní větě smí použít heslo *stát = stand.* Podobně důležité je, aby víceslovná hesla obsahovala popis syntaktické struktury zdrojového výrazu, protože bez ní je velmi obtížné ve fázi transferu ve větě, která již byla syntakticky analyzována, spolehlivě poznat, kde je které překladové heslo.

Čištění a doplňování anotace do slovníku probíhá v několika oddělených fázích, částečně manuálně a částečně automaticky za použití dostupných paralelních nebo jednojazyčných korpusů:

Identifikace metainformací Všechna slovníková hesla, která obsahovala slova typická pro označení nějaké speciální informace (např. *někoho*, *sg.* ap.), jsme

[1] `http://www.rewin.cz/`
[2] dvojice český výraz–anglický výraz

ručně prošli po blocích podle nalezeného podezřelého slova. Tento způsob zpracování nám umožnil soustředit se vždy na jeden konkrétní jev a zpracovat velké množství hesel současně.

Během této fáze jsme tedy formalizovali všechny metainformace a rovněž jsme roznásobili jednotlivé varianty hesel, které byly dosud sloučeny do jednoho hesla.

Identifikace slovních druhů Po automatické morfologické analýze hesel pomocí analyzátoru [5] bylo nutné provést zjednoznačnění slovního druhu u sporných slov. Toto zjednoznačnění bylo nutné provést opět ručně, po blocích se stejným typem nejednoznačnosti, neboť řada spojení umožňuje více různých interpretací, z nichž však jen jedna odpovídá anglickému překladu. (Např. spojení *kniha účetní* je buď *account book*, nebo prostě kniha, která patří nějaké účetní.)

Přidání morfologických omezení Zejména pro víceslovná hesla je nutné doplnit, které morfologické rysy jsou ve výrazu povinné a které je možné libovolně měnit. Např. hesla typu adjektivum a substantivum typicky vyžadují shodu pádu, čísla a rodu. U bohatších konstrukcí může být seznam omezení složitější.

Morfologická omezení doplňujeme automaticky. Předem jsme definovali repertoár typických omezení (např. pevně daný pád jména nebo zmiňovaná shoda v některé morfologické kategorii) a poté využili Českého národního korpusu (ČNK[3]) k tomu, abychom pro každé heslo na základě pozorovaných (pravděpodobných) výskytů identifikovali, jaká omezení heslo vždy splňuje. Tento postup naráží na problémy jednak u velmi řídkých hesel (kde ČNK neposkytuje dostatek výskytů na to, aby bylo možné spolehlivě rozhodnout, zda dané omezení platí) a jednak u hesel složených z velmi četných slov. V takovém případě totiž najdeme příliš velké množství náhodných kolokací všech slov hesla, nicméně tato slova navzájem nesplňují morfologická omezení (protože koneckonců nejde o výskyt daného hesla), takže algoritmus morfologické omezení neodhalí.

Doplnění syntaktické informace Vzájemné závislostní vztahy mezi slovy hesla přidáváme opět ručně, po blocích se všech hesel se stejnými slovními druhy jednotlivých slov. Bylo by rovněž možné postupovat podobně jako při přidávání morfologických omezení, bylo by však nutné použít syntakticky analyzovaný korpus nebo korpus automaticky syntakticky analyzovat.

4.3 Modul rozpoznávající pojmenované entity

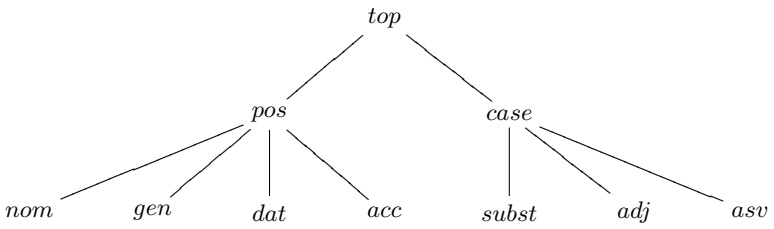
Tento relativně nezávislý modul se stará o zpracování idiomů, terminologických jednotek a ostatních typů pojmenovaných entit.

[3] `http://ucnk.ff.cuni.cz/`

Pojmenované entity (*named entities*, NE) jsou atomické jednotky, např. vlastní jména, časové výrazy či množství. Vyskytují se často v různých typech textů a nesou důležitou informaci. Proto má odpovídající analýza pojmenovaných entit a jejich správný překlad podstatný dopad na kvalitu strojového překladu [10].

Překlad NE zahrnuje jak sémantický překlad, tak fonetickou transliteraci. Různé typy NE se překládají různě. Například osobní jména se nepřekládají sémanticky (nutná je v některých případech pouze transliterace), zatímco některé tituly či označení profese se překládají (např. *první dáma Laura Bushová → first lady Laura Bush*). V případě organizací obvykle stačí regulární pravidla pro analýzu jmenných frází (NP) (např. *Ústav formální a aplikované lingvistiky → Institute of formal and applied linguistics*), ačkoliv někdy je vhodnější idiomatický překlad. Co se týče geografických názvů, používáme zvláštní glosář.

Pro rozpoznávání NE jsme vyvinuli gramatiku založenou na regulárních výrazech, jež pracuje s typovanými strukturami rysů. Interpretr této gramatiky, podobný formálně poněkud slabší platformě SProUT [11], používá řetězcové grafy a unifikaci, gramatika tedy sestává z pravidel, kde levá strana je regulární výraz nad typovanými strukturami rysů s proměnnými, jenž reprezentuje rozpoznaný podřetětec vstupu, a pravá strana definuje výstupní strukturu rysů po aplikaci pravidla. Hierarchie typů je definována globálně, jednoduchý příklad je na obr. 1.



Obr. 1. Příklad jednoduché hierarchie typů.

Gramatika rozpoznávající NE je inspirována pravidly popsanými v [12]. Příkladem jednoduchého pravidla je:

```
#subst[LEMMA: ministerstvo]$s1
+ #top[CASE: gen, PHRASE: $phr]$s2
== $s1#ministry[ATTR: $s2,
PHRASE: &('ministerstvo ' + phr)]
```
(1)

První struktura rysů reprezentuje libovolnou morfologickou variantu slova *ministerstvo* následovanou jmennou frází v genitivu. Proměnné $s1, $s2 a $phr zajišťují dynamické přiřazení hodnot a umožňují přenést tyto hodnoty na příslušné místo ve výstupní struktuře rysů typu *ministry*. Tato struktura obsahuje nový atribut PHRASE s lematizovanou hodnotou celé fráze.

Je-li vstupem například

informace ministerstva zahraničí
o cestování do ohrožených oblastí (2)

potom fráze "ministerstva zahraničí" bude rozpoznána jako NE a v dalším procesu překladu s ní bude zacházeno jako s atomickou (nedělitelnou) jednotkou:

$$\begin{bmatrix} \textit{ministry} & \\ \text{LEMMA} & \text{ministerstvo} \\ \text{FORM} & \text{ministerstva} \\ \text{PHRASE} & \text{ministerstvo zahraničí} \\ \text{ATTR} & \begin{bmatrix} \textit{subst} & \\ \text{LEMMA} & \text{zahraničí} \\ \text{PHRASE} & \text{zahraničí} \\ \text{FORM} & \text{zahraničí} \\ \text{CASE} & \text{gen} \\ \text{NUMBER} & \text{sg} \\ \text{GENDER} & \text{n} \end{bmatrix} \\ \text{CASE} & \text{gen} \\ \text{NUMBER} & \text{sg} \\ \text{GENDER} & \text{n} \end{bmatrix} \quad (3)$$

Lematizace NE je klíčovým předpokladem pro strojový překlad. Zejména u jazyků s bohatou flexí může být při analýze problémem strukturní víceznačnost, např. vnitřní uzávorkování komplexních jmenných frází. Jádro gramatiky použité v tomto projektu vychází z gramatiky systému strojového překladu Česílko [8].

4.4 Syntaktická analýza češtiny

Ačkoli jsme původně předpokládali, že modul syntaktické analýzy češtiny bude vyžadovat pouze velmi malé úpravy, ukázalo se, že bude nutné jeho gramatiku doplnit zejména o taková pravidla, která se uplatní na novém vstupním textu. Jedná se například o pravidla zpracovávající obrovské množství číselných výrazů, kterými jsou texty z Wall Street Journalu doslova protkány. Přes tyto potíže však modul syntaktické analýzy zůstal jako jediný, který bylo možno plně uplatnit v novém systému.

4.5 Transfer

Hlavním úkolem tohoto modulu je převod syntaktických struktur (závislostních stromů) popisujících české věty ze vstupního textu do syntaktických struktur (stromů) odpovídajících anglických vět. Transfer se nezabývá překladem jednoduchých lexikálních jednotek (jednotlivých slov), tento překlad obstarává hlavní dvojjazyčný slovník systému v dřívějších fázích zpracování. Transfer se soustředí na tři hlavní úkoly:

- Převod českých závislostních stromů na anglické, zejména s ohledem na slovosledné změny, vyžadované cílovým jazykem.
- Odhalování a překlad těch českých konstrukcí, které vyžadují speciální překlad do angličtiny.
- Vkládání určitých a neurčitých členů do vět cílového jazyka.

Vývoj tohoto modulu stále pokračuje, počáteční testy ukázaly, že se v něm skrývají velké rezervy, které v budoucnu mohou pomoci zlepšit kvalitu výsledného překladu.

4.6 Syntaktická syntéza angličtiny

Syntaktická syntéza ruštiny je v systému RUSLAN velmi úzce svázána s transferem, proto se snažíme využít co největší část gramatiky, kdekoli je to možné, a doplnit nová pravidla podobně, jako je tomu u modulu syntaktické analýzy. Práce na tomto modulu stále pokračují.

4.7 Morfologická syntéza angličtiny

Vzhledem k jednoduchosti anglické morfologie má tento modul v našem systému pouze velmi omezenou roli. V podstatě se jeho úloha v této fázi projektu redukuje pouze na generování správných tvarů plurálu, třetí osoby a nepravidelných slovesných tvarů.

5 Výsledky překladu

Pro vyhodnocení výsledků jsme se rozhodli alespoň zpočátku používat metodu, popsanou v [8]. Tato metoda měří podobnost původního a přeloženého textu (původním textem zde označujeme opravdu původní text z Wall Street Journalu, který máme k dispozici v korpusu PCEDT). Tato podobnost se měří pomocí aplikace TRADOS Translation Workbench. Systém TRADOS dokáže totiž vyhodnotit procentuální shodu mezi dvěma větami, v našem případě tedy mezi původní větou z Wall Street Journalu a automatickým překladem jejího českého překladu zpět do angličtiny. Výhodou této metody je její snadná použitelnost pro monitorování pokroku při vývoji systému, v budoucnu ale plánujeme její nahrazení standardními a přesnějšími metodami, např. použitím tzv. BLEU skóre.

Článek [8] uvádí, že komerční systém automatického překladu PC Translator testovaný na 256 větách z korpusu PCEDT dosáhl váženého průměru shody ve výši 30%. Naše testy na odlišné (a menší) sadě vět ze stejného zdroje naznačují nepatrně lepší výsledky (okolo 35%). Toto měření však bylo pouze orientační a nelze z něj zatím odvozovat žádné závěry, systém stále prochází dalším vývojem a testováním.

6 Závěr

Ačkoli první testy ukázaly povzbuzující výsledky, stále existuje celá řada možností, jak systém dále vylepšit. Kromě práce na současných modulech se nabízejí nejméně tři další směry pro další výzkum. Jedním z velkých problémů je morfologická víceznačnost jednotlivých českých slovních forem. Cesta, kterou jsme se rozhodli v budoucnu vyzkoušet, nevede přes přes stochastické značkování, ale přes částečnou (ale bezchybnou) desambiguaci výsledků

morfologické analýzy češtiny. Ta by měla odstranit značné množství nesprávných značek a tím velmi ulehčit roli modulu syntaktické analýzy češtiny.

Dalším způsobem, jak bude možné snížit víceznačnost slov ze vstupního textu, je zapojení speciálního modulu, který bude schopen vyřešit lexikální víceznačnost v těch případech, kdy slovník nabídne více lexikálních ekvivalentů ke vstupním slovům. Tento stochastický model se bude rozhodovat na základě kontextu a pokusí se navrhnout nejlepší překlad.

Třetím směrem možného vývoje prací na systému je opuštění dědictví systému RUSLAN a nahrazení jeho ručně vytvořené syntaktické analýzy některým z dostupných stochastických analyzátorů češtiny. Takový experiment by mohl dát odpověď na klíčovou otázku celého projektu – opravdu se vyplatí recyklovat staré systémy nebo ne?

References

1. Oliva K., A Parser for Czech Implemented in Systems Q. Explizite Beschreibung der Sprache und automatische Textbearbeitung, 1989
2. Colmerauer A., Les Systemes Q ou un formalisme pour analyser et synthetiser des phrases sur ordinateur, 1969
3. Bémová A., Kuboň V., Czech-to-Russian Transducing Dictionary. In: Proceedings of the 13th International Conference COLING 3, 1990, 314–316
4. Cuřín J., Čmejrek M., Havelka J., Kuboň V., Building a Parallel Bilingual Syntactically Annotated Corpus. In: Proceedings of the 1st International Joint Conference on NLP, 2004
5. Hajič J., Disambiguation of Rich Inflection – Computational Morphology of Czech. Volume I. Prague Karolinum, Charles University Press, 2001, 334 pp.
6. Collins M., Hajič J., Brill E., Ramshaw L., Tillmann C., A Statistical Parser of Czech. In: Proceedings of 37th ACL Conference, University of Maryland, College Park, USA, 1999, 505–512
7. Čmejrek M., Cuřín J., Havelka J., Czech-English Dependency-based Machine Translation. In: Proceedings of the 10th Conference of The European Chapter of the Association for Computational Linguistics, Budapest, Hungary, 2003, 83–90
8. Hajič J., Homola P., Kuboň V., A Simple Multilingual Machine Translation System. In: In: Proceedings of the MT Summit IX, New Orleans, 2003
9. Svoboda M., GNU/FDL English-Czech Dictionary, 2001, `http://slovnik.zcu.cz/`
10. Babych B., Hartley A., Selecting Translation Strategies in MT Using Automatic Named Entity Recognition. In: Proceedings of the Ninth EAMT Workshop, Valetta, Malta, 2004
11. Bering C., Drożdżyński W., Erbach G., Guasch C., Homola P., Lehmann S., Li H., Krieger H.U., Piskorski J., Schaefer U., Shimada A., Siegel M., Xu F., Ziegler-Eisele D., Corpora and Evaluation Tools for Multilingual Named Entity Grammar Development, 2003
12. Piskorski J., Homola P., Marciniak M., Mykowiecka A., Przepiórkowski A., Woliński M., Information Extraction for Polish Using the SProUT Platform. In: Proceedings of the International IIS:IIP WM'04 Conference, Zakopane, Poland, 2004

Design of ontology used for knowledge sharing

Ivana Budinská, Balatazár Frankovič, Giang Nguyen, and Ladislav Hluchý

Institute of Informatics, Slovak Academy of Sciences
Dúbravská cesta 9, 845 07 Bratislava, Slovakia

Abstract. The paper deals with aproblem of ontology design for knowledge intensive applications. Ontology present an important background of each experience and knowledge management system. It defines structure and relationships among experience and knowledge entities. The ontology is the main mechanism used for the representation of information and knowledge, definition of the meaning of the terms used in the content language and the relation among these terms. Components of the system communicate using ontology structure, store and retrieve data organized and described by ontology to/from the system memory. The ontology model is modelled using the Protg. The output of the Protg is portable RDF/OWL file, which is further mapped into Organizational Memory (OM) through OM interface at the first running (initial) time as ontology class definition with initial instances and property values. Ontology-based information and data in OM are also further capitalized to invent new knowledge (also ontology-based) that will be stored in OM for the future reuse.
The research in this area has been initially developed and evaluated in the project Pellucid and it is further improved in the Slovak national NAZOU (SPVV 1025/2004), RAPORT (APVT-51-024604), VEGA No. 2/6103/6, as well as towards the Grid environment in the K-Wf Grid project (EU RTD FP6-511385).

1 Introduction

Ontology has become a very important aspect in many applications to provide a semantic framework for knowledge management. Ontology is a set of definitions of content-specific knowledge representation primitives (classes, relations, functions and constants). The advantages of using ontology are modularity and reusability. For that reason, standardized languages with rich description and terminological logics are needed. Agent-based software integration involves designing ontology for a specific domain and integration of different tools to allow agents communication using that ontology. Ontology presents a shared understanding about a certain specific domain. Comparing to database, ontology allows handling not only numerical transactional data but also it is suited to model unstructured and informal knowledge. However the advent of object oriented databases, improved logics and faster inference is making the distinction between DBs and ontologies fuzzier. There is also (multiple) inheritance, strong encapsulation, fuzzy set algorithms, meta-data standards, neural networks

to train, discover and disambiguate meaning, and increased computing power. Although ontology enables processing knowledge and data, the most important role of ontology is in defining sharing meaning, emergence and discovery of gaps and for improving tacit knowledge transfer. Besides ontology, a knowledge base may contain information specified in a declarative language such as logic or expert-system rules, but it may also include unstructured or unformalized information expressed in natural language or procedural code.

Ontology based knowledge management includes activities like knowledge acquisition, creation, accumulation, sharing, reuse and capitalization. Knowledge items are abstracted to a characterization by metadata descriptions, which are used for further processing.

2 Methodology for ontology development

In present time many publications deal with ontology development, ontological methodology and with the creation of ontological tools for the solution of different tasks in different areas [2], [3], [4], [5]. Domain ontology is constructed as a conceptual model for knowledge items. The ontology is represented using some logic formalism (F-Logic) that specifies relations within domain concepts.

Ontological engineering or methodological ontology denoted as METHONTOLOGY (MO) [6,7,8,9] belongs to the young area of artificial intelligence (AI). The important element of Methontology may be considered as follows:

- Specification is the determination of classes or concepts
- Conceptualization is the modeling at the knowledge level by, for example, tables an graphs
- Formalization uses, for example, a formal language or some frames
- Implementation uses, for example, a special language of ontology
- maintainance.

There exist two types of knowledge in each domain:

- basic knowledge of the system introduced implicitly (modelling)
- knowledge of the system introduced into system explicitly (decision actions).

Their regularity may be confirmed on the meta level. Then the proposed tables and graphs allow modelling concepts, attributes, logic formulas and other rules used, for example, in Multi-Agent-System (MAS). They are handled by experts in the domain to be modeled.

A graph is formulated as a concept classification tree and a table is an example of concept dictionary [1]. Methontology defined using graphs and tables formulated as above do not give a sufficient answer to the questions

- how to elaborate a completely new modelling way with completely new graphs and tables,
- how to build an ontology collaboratively,
- how to agree and exchange the characteristics of the tables and graphs to be used.

This problem appears mainly in the case of flexible systems, hierarchically ordered with different types of classes as for example flexible manufacturing systems and supply chains are. To allow more flexible modelling of ontologies and to ease the exchange of characteristics of tables and graphs in [2] a bi-phase method is proposed for the process how to model the ontology.

First phase consists of:

- meta specification
- meta conceptualization
- meta formalization
- meta implementation.

Second phase consists of:

- specification, conceptualization using the meta model (meta tables, meta graphs) obtained in the first phase
- formalization and implementation of the ontology.

For the facilitation and implementation of the building of meta models in our case a reference meta model may be proposed. Further we suppose that it is possible to modify the reference meta model according to modeling requirements of each ontology. Such meta model is expressed by means of meta tables and meta graphs, and it is also formally expressed. Then the reference meta model allows building ontologies with:

- concepts
- class and instance attributes
- facet of such attributes
- relations
- first order logic formulas or other logical formulations
- instances.

These components appear in the reference meta model because each one of them have been in some ontologies developed during the experimentation.

Many current applications, such as commerce, production, delivering, sale etc., with different types of resources or agents to interoperate with each other may be solved on the basis of the ontological methodology. In some cases, interoperation becomes more complex, because agents may have been developed independently; therefore the assumption that agents use the same communication language and the same terminology in a consistent way cannot be made. When dealing with independently developed agents, their interoperability with humans and others depends on the agent ability to understand them, which leads us directly to ontology. Ontology are an explicit formal specification of a shared conceptualization, where a conceptualization refers to an abstract of some phenomenon by having identified the relevant concepts of that phenomenon, explicit means that the type of concepts used, and the constraints on their use are explicitly defined, formally refers to the fact that the ontology should be

machine readable. The ontologies provide a formally defined specification of the meaning of those terms that are used by agents during the interoperation.

One of the possible ways for the solution of the tasks introduced above might be the meta level approach. The meta properties we propose, namely Mutability, Mutability Frequency, Reversible Mutability, Event Mutability, Modality, Prototypicality, Exceptionality, Inheritance, Inheritance and Distinction, encompass semantic information aiming at characterizing the behavior of properties in the concept description. We claim that this characterization is very important in order a precise specification of the semantics of the concepts. Such characterization is essential if we want to perform a formal ontological analysis, in which knowledge engineers can precisely determine which formal tools can be used in order to build an ontology which has a taxonomy that is clean and not very tangled. The novelty of this characterization is that it explicitly represents the behavior of attributes over time by describing the permitted changes in a property that describes the concept. It also explicitly represents the class membership mechanism by associating with each attribute a qualitative quantifier representing how properties are inherited by sub-concepts. Finally, the model does not only describe the prototype properties holding for a concept, but also the exceptional ones. By providing this explicit characterization, we are asking knowledge engineers to make more hidden assumptions explicit, thus providing a better understanding not only of the domain in general, but also of the role a concept plays in describing a specific domain [5].

3 Formalization of the ontology

Ontology can be represented by UML, any object oriented language, RDF, DAML+OIL, OWL or any other representation which can define objects, properties and its relations. Each domain concept has assigned a representation. The representation can have a form of a textual representation and can be realized for example in a form of the regular expression.

Meta – specification of the conceptualization process consists of three steps:

- specification of tables and graphs forms which may be used in the conceptualization of ontology,
- recommended order to fill in the tables and to create the graphs,
- the consistency verification rules between tables and graphs.

For example, the first step, the form of a table, can contain the following elements: Classes, Symbols, Number of subclasses, Description, Input/Output edge Input Multiplication, Outpur Multiplication.

The graph to be used during the conceptualization is the concept classification tree and can be created on the basis of the following procedure:

We assume that ontology is represented as a graph along with a set of logical rules. Formally, an ontology:

$G = (V, E)$

comprises a finite set of nodes V and a finite set of edges E. The label of a node is given by a noun phrase that represents a concept name. The label of an edge is

the semantic relationship among the concepts and can be null if the relationship is not known or not realizable. The label of an edge can be any user defined relationship. The set of relationship the pre-defined semantics is for example (class, subclass, Part, Attribute, Instance, Value). In general, other relationships may be interpreted too. It depends on the complexity of the system.

The solution way of the second step depends on the concrete type of application. The third step supposes the definition of concepts. It contains the possible fields of concepts name, its classes and corresponding attributes.

4 Case study - Supply chain

A supply chain is considered as a loop from customers demand to customers satisfaction by final product or service, Fig. 1. It is made through a chain of producers, suppliers, distributors, and transporters. A supply chain is a complex and dynamic system, which has a character of hybrid-distributed system. In MAS language, a supply chain can be modeled as a system of intelligent agents, which agree to cooperate to reach the final goal. A new, modern, and cost-effective implementation of supply chain management (SCM) is enabled by rapidly developing information and communication technologies. SCM is a process of creating and configuring a supply chain, identifying measuring metrics in the chain, determining weak points in the chain and work to achieve the best results to meet customer demands.

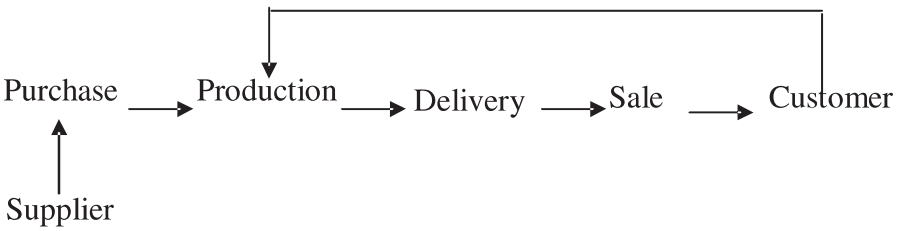


Fig. 1. A basic scheme for SC.

From this point of view the SC may be considered as a workflow in different areas of application. The workflow provides the connectivity tools to manage, organize and distribute specific tasks and the associated, required information needed to complete those tasks successfully. Generally this flow of tasks and information might include images of documents, any number of information media, including multimedia. Workflow as a SC is a modern method for passing information from source to user (customer) at one site and the technological flow at other site (such flow is considered in MO as well).

The SC with workflow properties is characterized by the following items that should be evaluated independently:

- information capturing, which is a function of making the necessary data available to each part of SC workflow,
- work routing is a common function of workflow,
- application integration, where the workflow software integrates the parts of SC into the production process.

Let us consider a special SC where the chain consists of suppliers, producers, distributors, and transporters. There are the main classes defined in ontology. In MAS language, SC can be modeled as a system of intelligent agents, which agree to cooperate to reach the final goal [10]. This approach can be used to create supply chain management (SCM), a new, modern, and cost-effective implementation, which is enabled by rapidly developing information and communication technologies. In SC one element can be either supplyer or customer at the same time.

Supply chain management aims to develop strategies for managing all resources (raw materials, services, etc.) and balance cumulative demands and supplies. SCM includes the following five steps:

- developing a set of metrics to monitor the SC efficiency
- creating network of suppliers and producers, warehouses and distribution and transportation services
- coordination of orders from customers, developing invoicing policy
- control of manufacturing - scheduling, testing, packaging, preparation for delivery
- development of return policy - deals with a questionable or defective products

Each basic SC is a chain of Source, Make and Deliver processes. A SCM is different for various systems, depending on specific requirements, which must be analysed in a process of establishing a SC.

5 Conclusion

Building ontology is considered as a crucial part of building knowledge systems. The cooperation of many domain and IT experts is required to build ontology that really is tailored for the designed knowledge system. Creation of ontologies represents an open environment for many researchers and developers. The paper is intended to present a general problems and methods of creation ontologies rather than present a special ontology (e.g. scheduling ontology, SCM ontology, etc.). Methodology of creating ontologies is very important in order to develop recently existing ontologies and establish new ontologies into more specific and generic systems, which will be used widely in the defined area. The basic steps in creation of ontologies (specification, conceptualization, formalization, and implementation) were defined.

References

1. Fernandez-Lopez M., Meta-Modelling for Ontology Development and Knowledge Exchange. In: 15th ECAI Conference Workshop 1 "Ontologies and Semantic Interoperability" Lyon 2002, 6–15
2. Frankovic B., Budinska I., Dang T.T.: Creation of Ontology for Planning and Modeling. In. Proc. 3th Int. Symposium of Computational Intelligence Budapest 2002, 73–84
3. Gangemi A., et all, A Formal Ontological Framework for Semantic Interoperability in the Fishery Domain. In: 15th ECAI Conference Workshop 1 "Ontologies and Semantic Interoperability", Lyon 2002, 16–30
4. Menzel Ch., Ontology Theory In: 15th ECAI Conference Workshop 1: "Ontologies and Semantic Interoperability", Lyon 2002, 38–44
5. Mitra P., Wiederhold G., Resolving Terminological Heterogeneity in Ontologies. In: 15th ECAI Conference Worshop 1: "Ontologies and semantic Interoperability", Lyon 2002, 45–50
6. Uschold M., Gruninger M., Ontologies: Principles, Methods and Applications. The Knowledge Engineering Review 11(2), 1996, 93–136
7. Bernaras A., Laresgoiti I., Corera J., Building and Reusing Ontologies for Electrical Network Applications. In Proceedings of the European Conference on Artificial Intelligence (ECAI 96), 1996
8. Gmez-Prez A., Fernndez-Lpez M., de Vicente A., Towards a Method to Conceptualize Domain Ontologies. In Proceedings of ECAI96 Workshop on Ontological Engineering, Budapest, 1996, 41–51
9. Swartout B., Ramesh P., Knight K., Russ T., Toward Distributed Use of Large-Scale Ontologies. AAAI Symposium on Ontological Engineering, Stanford, 1997
10. Dang T.-Tung, Improving Plan Quality through Agent Coalitions. In Proc. of the 2nd IEEE International Conference on Computational Cybernetics, ICCC 2004, Viena University of Technology, Viena, August 30 - September 1, 2004, Eds: W. Elmenreich et all, 171–176, ISBN 3-902463- 02-3

Skrývání informace kompresní metodou kvantováním vektorů

Jiří Dvorský, Eliška Ochodková, and Václav Snášel

Katedra informatiky, VŠB – Technická univerzita Ostrava,
tř. 17. listopadu 15, 708 33 Ostrava-Poruba, Česká republika
{Jiri.Dvorsky, Eliska.Ochodkova, Vaclav.Snasel}@vsb.cz

Abstrakt Skrývání informace (steganografie) patří mezi intenzivně zkoumané oblasti CS. V článku popíšeme nový přístup ke kompresi kvantováním vektorů. Dále popíšeme jaké je jeho využití v oblasti steganografie.

Klíčová slova: Steganografie, komprese, kvantování vektorů, SVD.

1 Úvod

V současnosti, kdy už jsou téměř všechna místa na Zemi propojena celosvětovou sítí, se dostává do popředí otázka bezpečnosti a s ní spojená nutnost ochrany informací či duševního vlastnictví. Jedním z možných způsobů, jak tyto problémy řešit, je využití steganografie, vědy, která se zabývá skrýváním informací. Steganografie se, na rozdíl od kryptografie, snaží utajit samotnou existenci tajné zprávy. Kryptografie existenci zprávy netají, pouze mění tvar zprávy m tak, aby byla pro potenciálního útočníka nečitelná. K tomu používá šifrování, obecně kryptografické algoritmy, např. RSA [12]. Samotný fakt, že takováto tajná (šifrová) zpráva c existuje, ji však činí podezřelou, a může tak snadno upoutat útočníkovu pozornost.

Základním principem staganografie [10] je vkládání tajné zprávy m do tzv. krycího objektu c, což je obecně neškodná, ale smysluplná zpráva. Tak vznikne tzv. stego-objekt s. Hlavním cílem steganografie je, aby rozdíl mezi krycím objektem c a stego-objektem s nebyl zjistitelný ani lidským okem ani např. počítačem zkoumajícím statistické vlastnosti objektu, obecně aby jej potenciální útočník nepostřehl.

Existuje mnoho různých způsobů jak a kam nějakou zprávu ukrýt. Všechny poskytují určitou míru utajení, ale ne všechny jsou zároveň odolné vůči odhalení a případným útokům. Některé lze kombinovat s jinými, a zvýšit tak celkovou bezpečnost systému. Všechny steganografické metody využívají jistou míru náhodnosti, neurčitosti nebo přítomnosti redundantních dat v souboru [10]. Způsobem, jak např. ochránit své elektronické dílo, může být použití "digitálního vodoznaku", jenž je založený právě na steganografických technikách a nachází uplatnění především v digitálních multimédiích.

Pokud je na krycí objekt c aplikována ztrátová komprese, hrozí samozřejmě ztráta informací z tajné zprávy m. To platí hlavně u obrázků, zvuků či videa, kde

se tato komprese využívá nejčastěji. V případě zvuku jsou hlavním nepřítelem steganografie komprese typu MP3, OGG či AAC, které využívají nedokonalosti lidského sluchu a ořezávají zvukové soubory o vysoké frekvence. U obrázků a videa hrozí např. komprese JPEG a MPEG, které převádějí digitální obraz do frekvenčního spektra, a tím jej ochuzují o jemné detaily.

V našem příspěvku se věnujeme skrývání informace kompresí kvantováním vektorů. Tato problematika je v současnosti intenzivně zkoumána. Existuje celá řada přístupů, viz kapitola 3, jak ke kvantování vektorů, tak k vlastnímu skrývání tajné zprávy m (informace) v takto komprimované zprávě, [5], [16], [9], [11]. Náš přístup, viz kap. 3 spočívá v nahrazení části algoritmu GLA pro vytvoření slovníku algoritmem SVD, tyto metody jsou popsány v kapitole 2.

2 Použité metody

2.1 Kvantování vektorů

Kvantování [18] je často základem ztrátových kompresních metod. V principu se jedná o diskretizaci oboru hodnot. Obor hodnot se rozdělí na intervaly, jimž je pak přidělena jediná zástupná hodnota. Jinak řečeno, vezme se "spojitá" množina (rozsah) hodnot a reprezentuje se (nepřesně, se ztrátou informace) podmnožinou, která je tvořena hodnotami souvisejícími s původní množinou. Jako zástupná (kvantizační) hodnota se obvykle používá:

- průměr hodnot z daného intervalu (někdy vážený průměr),
- medián,
- průměr z hodnot na okraji intervalu.

Podle způsobu rozdělení kvantované veličiny se rozlišuje:

- uniformní (lineární) kvantování - používá konstantní délku intervalu,
- neuniformní (nelineární) kvantování - používá proměnnou délku intervalu.

Při kvantování dochází ke kvantizační chybě, která se v počítačové grafice projevuje například na plochách s malou změnou gradientu jako náhlý skok barev. Původně hladký přechod je nahrazen skokovou změnou a v obraze vznikají hrany. Neuniformní kvantování může kvantizační chybu částečně eliminovat.

Podle typu kvantované veličiny se rozeznává:

- kvantování skalárů (Scalar Quantization, SQ) - vezme se jedna vstupní hodnota a výsledkem je jedna kvantovaná (kvantizační) výstupní hodnota,
- kvantování vektorů (Vector Quantization, VQ) - vezme se skupina vstupních hodnot (vektor) a výsledkem je skupina kvantovaných (kvantizačních) výstupních hodnot (vektor).

Kvantování vektorů, na rozdíl od kvantování skalárů, dokáže využít a z velké části zachovat vzájemný vztah mezi daty (pixely). Místo jednotlivých pixelů kvantuje vektory (bloky) pixelů. Pro danou míru komprese (rate) dosahuje menší

distorze, a naopak při dané distorzi dosahuje lepší míry komprese. Kvantování vektorů může být použito jako standardní kompresní metoda pracující s původními daty (obrázky, zvuk) nebo může být použito jako kvantizační část některé obecné ztrátové komprese.

Základní princip

- Vytvoření slovníku (codebook). Slovník se skládá z N_c vektorů (kódových) o n prvcích (z bloků o n pixelech).
- Kódování
  1. Rozdělení vstupního obrazu na bloky (vektory) o n pixelech.
  2. Pro každý takto vzniklý vektor x se najde ve slovníku $C = \{c_0, c_1, \ldots, c_{N_c-1}\}$ nejbližší kódový vektor c_i. Kóduje se vektor x indexem i kódového vektoru c_i ze slovníku C.
  3. Bezeztrátová komprese získaných indexů.
- Dekódování (vyhledávací tabulka)
  1. Bezeztrátová dekomprese indexů.
  2. Nahrazení každého indexu i příslušným kódovým vektorem c_i ze slovníku C, rekonstrukce vstupního vektoru x.
- Slovník je uložen nebo přenesen.
- Slovník může být vytvořen pro konkrétní obrázek nebo pro skupinu obrázků.

Konstrukce slovníku. Základním nástrojem pro konstrukci slovníku je LBG algoritmus (Linde-Buzo-Gray Algorithm), který je též někdy označován jako GLA (Generalized Lloyd Algorithm). Podrobný popis algoritmu GLA přesahuje rámec tohoto článku, lze jej nalézt např. v [15]. Ve stručnosti si popišme jednotlivé kroky algoritmu:

1. Začíná se s počátečním slovníkem, který má N_c kódových vektorů.
2. Vytvoří se N_c tříd z trénovacích vektorů - každý trénovací vektor x se přiřadí do třídy i, jestliže i-tý kódový vektor c_i je trénovacímu vektoru x nejbližší. (Trénovací množina je množina všech bloků získaných z obrázku určeného ke kompresi. V případě globálního slovníku je trénovací množina množinou všech bloků získaných ze zastupujících obrázků vybraných z dané třídy obrázků.)
3. Opakovaně se přetváří jednotlivé třídy tak, že se pro každou třídu spočítá nový centroid (střed) c_i. Potom se pro každý trénovací vektor x najde nejbližší centroid c_i a trénovací vektor se přiřadí do třídy, kterou tento centroid c_i reprezentuje.
4. Tento proces se ukončí v okamžiku, když se celková odchylka (rozdíly mezi trénovacími vektory a jejich centroidy) již příliš nemění.
5. Jako slovník se vezmou nejnovější centroidy.

2.2 SVD rozklad

Nechť A je matice typu $m \times n, m \geq n$. Pak existuje ortogonální matice U řádu $m \times m$, diagonální matice W typu $m \times n$ s kladnými prvky na diagonále a ortogonální matice V řádu $n \times n$ taková, že platí

$$A = UWV^T. \tag{1}$$

Tomuto rozkladu se říká singulární rozklad (Singular Value Decomposition, SVD) matice A. Diagonální prvky $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > \sigma_{r+1} = \ldots = \sigma_p = 0$ (kde p je menší z dimenzí m, n) matice W se nazývají singulární čísla (singular values) matice A (hodnost matice $A = r$). Sloupce matic U a V jsou singulární vektory matice A. Výpočet vlastních čísel i singulární rozklad se skládá ze dvou kroků:

- Aplikováním ortogonální transformace se převede matice do jednoduššího tvaru. Ortogonální transformace zachovává vlastní i singulární čísla. Nejčastěji se používá transformace na třidiagonální matici.
- Na zjednodušenou matici se aplikuje iterační algoritmus výpočtu vlastních čísel nebo singulárního rozkladu, viz [13].

Základním přínosem této metody je (i přes jisté zvýšení výpočetní náročnosti) vynikající numerická stabilita inverze. Díky tomu vede postup k použitelným výsledkům i v případě velmi špatně podmíněné soustavy.

3 Skrývání informace

3.1 Náš přístup

Jedno z využití algoritmu SVD se nabízí právě v oblasti komprese obrázků, viz [6]. Je-li původní obrázek reprezentován např. čtvercovou maticí A, je paměťová složitost m^2. Použijeme-li na původní matici SVD a pouze k singulárních čísel, bude potřeba $2mk + k$ paměti. Místo komprimování celého obrázku najednou se však používá dělení obrázků na bloky a jejich následná komprese, což vede k možnosti využít nestejné složitosti původního obrázku tak, že se podle složitosti jednotlivých jeho částí používá různý počet singulárních čísel k dosažení jeho uspokojivé aproximace. Tento přístup vyplývá z pozorování jak jsou distribuována singulární čísla jednotlivých bloků obrázku. Jednoduchý postup je popsán např. v [6]. Další modifikace [1] využívají dynamickou volbu velikosti bloků obrázku. Hlavní nevýhodou SVD-komprese je to, že spolu se singulárními čísly je třeba ukládat matice U a V.

Tuto nevýhodu lze ale zužitkovat tak, že se SVD-komprese použije v kombinaci s jinými kompresními technikami, např. v kombinaci s kvantováním vektorů, viz [17]. Autoři zde používají následující přístup:

- Obrázek je rozdělen na bloky 8×8.
- Je vytvořen VQ-slovník.

– Každé slovo v slovníku je rozloženo pomocí SVD, díky čemuž se vytvoří u-podslovník, resp. v-podslovník (důsledek SVD rozkladu je vznik dvou množin singulárních vektorů, tedy podslovníků). Tyto podslovníky obsahují jeden nebo více singulárních vektorů matice U, resp. V, které jsou dále používány pro vlastní kódování.

V každé iteraci algoritmu GLA, viz kap. 2 se pro trénovací vektor x hledá nejbližší kódový vektor c_i. Pro nalezení nejbližšího kódového vektoru se používá Euklidovská metrika. Námi navrhovaný přístup nahrazuje výpočet podobnosti vzdáleností v prostoru do kterého jsou kódové vektory promítnuty pomocí metody SVD [2]. Tato vzdálenost podstatně lépe zachovává podobnost obrazové informace [14], a nejen informace obrazové, viz [3], [7]. Předpokládáme, že tato modifikace povede k dosažení lepších výsledků než kterých dosahuje samostatné kvantování vektorů. Tento postup použijeme dále i pro vkládání tajné informace do obrázku pomocí dělení slovníku na více částí. Využijeme výsledek SVD rozkladu podobně jako v [17].

3.2 Příklad techniky skrývání informace

Existuje celá řada přístupů jak ke kvantování vektorů, tak k vlastnímu skrývání tajné zprávy m (informace) v takto komprimované zprávě.

Obecně se dá říci, že tyto přístupy používají buď jen kvantování vektorů [9] nebo navíc tzv. Side Match Vector Quantization (SMVQ). Prosté kvantování vektorů může ve výsledném obrázku způsobit viditelné hranice mezi jednotlivými bloky (do kterých je obrázek rozdělen), což je způsobeno tím, že aktuální blok je kódován nezávisle na sousedních. Proto někteří autoři, např. [5], používají VQ pouze pro zakódování prvního řádku a prvního sloupce. Pro kódování dalších bloků používají kvantizaci SMVQ, která používá předešlé kódové bloky k zakódování bloku aktuálního. K tomu však je potřeba mít k dispozici kromě "hlavního" slovníku ještě další podslovníky. Existují různé způsoby jak tyto podslovníky vytvořit [11].

Jeden z nich, včetně způsobu vkládání tajné informace m do obrázku komprimovaného SMQV je uveden v [4], zde uvedenou fázi skrývání lze popsat takto:

1. Zakódujeme první řadu a první sloupec pomocí VQ.
2. Pro ostatní bloky použijeme kódové bloky vlevo a nad aktuálním blokem. Tyto se použijí k vytvoření podslovníku $K = \{k_0, k_1, \ldots, k_{N_c-1}\}$, kde $k_i, i = 0, 1, \ldots, N_c - 1$ je i-tý kódový vektor.
3. Přeskupíme podslovník K tak, aby k_{2p} a $k_{2p+1}, p = 0, 1, \ldots, N_c - 1/2$ byly co nejblíže. Poté se přeskupený slovník K rozdělí na dva podslovníky K_0 a K_1, kde $K_0 = \{k_0, k_2, \ldots, k_{N_c-2}\}$ a $K_1 = \{k_1, k_3, \ldots, k_{N_c-1}\}$.
4. Nechť r-bitový proud $M = \{m_1, m_2, \ldots, m_r\}$ je tajná informace, kterou chceme skrýt. Pro každý její bit provedeme: jestliže $m_j = 0, 1 \leq j \leq r$, aktuální blok je kódován nejbližším kódovým vektorem k_i z podslovníku K_0 a na výstupu dostaneme index i. Jinak se provede 5. krok.
5. Jestliže $m_j = 1, 1 \leq j \leq r$, aktuální blok je kódován nejbližším kódovým vektorem k_i z podslovníku K_1 a na výstupu dostaneme index i.

4 Závěr a budoucí práce

V tomto článku jsme se věnovali problematice skrývání informace kvantováním vektorů. Tato problematika je velice široká, existuje celá řada přístupů jak ke kvantování, tak ke skrývání informace. Jeden z nich je v příspěvku stručně popsán. My jsme naznačili způsob, jak se dá vylepšit hledání nejbližšího kódového vektoru c_i pro vstupní vektor x, a to pomocí algoritmu SVD. Naše další práce bude spočívat ve vykonání experimentů pro takto pozměněné kvantování vektorů a experimentů s vkládáním tajné informace do obrázku.

Reference

1. Arnold B., An Investigation into Using Singular Value Decomposition as a Method of Image Compression, 2000, http://www.cs.bgu.ac.il/ña031/Arnold.pdf
2. Berry M.W., Browne M., Understanding Search Engines: Mathematical Modeling and Text Retrieval. SIAM Book Series: Software, Environments, and Tools, 1999
3. Cooper M., Foote J., Summarizing Polpuar Music via Structural Similarity Analaysis. IEEE Workshop on Applications of Signal Processing to Audio and Acoustics 2003, NY, USA, 2003
4. Chang Ch., Tai W.L., Lin Ch., Secret Hiding Using Side Matching Vector Quantization. Workshop on Soft Computing as Transdisciplinary Science and Technology 2005, Muroran, Japan, 2005
5. Chang C.C., Wu W.C., A Steganographic Scheme for Hiding Secret Data Using Side Match Vector Quantization. IEICE Transactions on Information and Systems, 2005
6. Chen J., Image Compression with SVD, 2000, http://graphics.cs.ucdavis.edu/j̃chen007/UCD/ECS289K/Project.html
7. Dvořák P., Střižík M., Praks P., Pudil P., Šumpíková M., Lešetický O., The Feasibility of Using Special Quantitative Methods for Prediction of Currency Crises. International Scientific Conference European Finance – Theory, Politics And Practice; Section C: Information Technology and Quantitative Methods in Finance, Banska Bystrica, Slovakia, 2004, ISBN 80-8055-968-6
8. Gorodetski V.I., Popyack L.J., Samoilov V., Skormin V.A., SVD-Based Approach to Transparent Embedding Data into Digital Images. International Workshop on Mathematical Methods, Models and Architectures for Computer Network Security MMM-ACNS 2001, St. Petersbug, Russia, 2001 http://varuna.ece.ucsb.edu/ece178W02/20520263.pdf
9. Jo M., Kim H.D., A Digital Image Watermarking Scheme Based on Vector Quantization. IEICE Transactions on Information and Systems, Vol. E85-D, No. 6, 2002, 1054–1056
10. Katzenbeisser S., Petitcolas F.A.P., Information Hiding Techniques for Steganography and Digital Watermarking. Artech House, Norwood, 2000, ISBN 1-58053-035-4
11. Lu Z.M., Xing W., Xu D.G., Sun S.H., Digital Image Watermarking Method Based on Vector Quantization with Labeled Codewords. IEICE Transactions on Information and Systems, Vol. E86-D, No 12, 2003
12. Menezes A., van Oorschot P., Vanstone S., Handbook of Applied Cryptography, 2001, http://cacr.math.uwaterloo.ca/hac/
13. Numerical Recipes. Cambridge University Press, http://www.library.cornell.edu/nr/

14. Praks P., Machala L., Snášel V., Iris Recognition Using the SVD-Free Latent Semantic Indexing. MDM/KDD2004 - Fifth International Workshop on Multi-media Data Mining "Mining Integrated Media and Complex Data" in Conjunction with KDD'2004 - The 10th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining; Section 2. Multimedia Data Mining: Techniques and Applications, Seattle, USA, 2004
15. Snášel V., Húsek D., Vyhledávání v kolekci obrázků s pomocí kvantování vektorů. Datasem 2000
16. Tsai C.S., Chang C.C., Chen T.S., Image Steganography Using Vector Quantization and Discrete Cosine Transformation. Imaging Science Journal 48, 2001, 205–215
17. Yang J.F., Lu C.L., Combined Techniques of Singular Value Decomposition and Vector Quantization for Image Coding. IEEE Transactions on Image Processing 4, No. 8, 1995, 1141–1146
18. Žára J., Beneš B., Felkel P., Sochor J., Moderní počítačová grafika. Computer Press, 2005, ISBN 80-251-0454-0

Representing content structure for e-learning systems*

Jana Dvořáková, Jana Katreniaková, and Branislav Rovan

Department of Computer Science,
Faculty of Mathematics, Physics and Informatics, Comenius University
Mlynská dolina, 842 48 Bratislava, Slovakia
{dvorakova,katreniakova,rovan}@dcs.fmph.uniba.sk

Abstract. In this paper we stress the importance of structuring content in e-learning systems. Following a brief discussion of some general issues in e-learning we concentrate on analysis of requirements for storing the structure of learning content. We decided to model this structure by a graph. We then analyse and compare suitability of various existing technologies (databases, directory structure, web space, XML) for storing the structure. We conclude that none of the existing approaches fulfills all the requirements. It thus follows that their combination and/or completely new approach has to be developed.

Keywords: e-learning, learning content, content structure

1 Introduction

E-learning (or computer enhanced learning) has been a very frequently used term in recent years. It denotes a type of learning, where knowledge is mediated to students through electronic media, especially internet. Initially, e-learning materials were created by simple rewriting of texts into an electronic form. Recently they became much more dynamic and interactive as learning materials may contain text combined with multimedia, 3D graphics or even communication, e.g., with experts through videoconferences.

All digital resources intended to be used as part of e-learning material must be stored in some locations. Obviously it is not suitable to use them directly for e-learning purposes. First we need to describe these resources (e.g., learning objectives to which the resource is to contribute) and relationships between them (e.g, prerequisits) so that we can present any given information to students in a well-arranged and comprehensible way.

Early approaches to e-learning were concentrating mainly on technology aspects. We believe, that leaving out the pedagogical and methodological aspects inhibited the development of the area. In this paper we intend to contribute to a stream of thought in which the choice of learning material and its structuring

* Supported in part by VEGA 1/0131/03.

is stressed. In particular, we shall consider the structure of content. We are convinced that a well structured content and a possibility to exploit this structure will enhance the usability of e-learning systems. We shall present a model of the structure and basic ways of using this structure. We shall then analyze suitability of various technological approaches for the "structured" view of content.

In Section 2 we specify basic e-learning concepts used within this paper, we describe how e-learning content is created and we list requirements for this process. Then in Section 3 we introduce our graph model of content structure and compare several approaches to its storage. Section 4 contains conclusion and proposals for future work.

2 E-learning

In comparison to the traditional learning, e-learning has some significant advantages. The learning content is complete and abundant as it is selected from many sources by experts. It is also kept up-to-date and accesible at any time. E-learner takes active role in the learning process, since e-learning provides easy way of interactive communication with other e-learners as well as with experts.

There is no single generally accepted definition of the term e-learning. Besides many definitions of e-learning there are also many, usually ad hoc, approaches to developing e-learning systems. Still, the area would benefit from a more unified and systematic approach. In our work we concentrate mainly on content of e-learning system, its structure, and possibilities of storing and representing the learning content exploiting this structure.

2.1 Process of the learning content development

First of all, we define the objects used in e-learning systems. In general, *an information object* (or a concept object) is defined as any single digital resource. *A learning object* consist of one or more information objects with metadata (*learning object metadata - LOM*). There is no standard for the size (or granularity) of learning objects. Within the framework of this paper we will consider it as an object holding some properties. Learning objects together with relations defined among them create *a learning content*. In the rest of this paper we will use the term *content* for learning content.

There exist several standards that specify learning objects' metadata. The most important are SCORM [4] and IEEE 1484.12.1 [2].

We shall assume, that at the beginning we have information that we want to use for building learning content. This information consists of information objects. These may be documents, database tables, multimedia, web pages, etc[1]. First of all we have to join these information objects into larger groupings, to describe them (add metadata) and to define relations among these groupings.

[1] We consider information objects as the smallest indivisible units. If it is necessary to break some digital resource into smaller parts, we suppose it was done by preprocessing.

Each grouping represents a learning object and it can be either a set of several information objects or a particular information object itself. After adding relations to learning objects we obtain learning content ready for presentation.

Creating relations between the learning objects can be made manually or automatically. The automatical solutions are based on:

- data mining or text mining (The practice of an automated search for patterns in large stores of data. Data mining uses statistical computational techniques, pattern recognition, etc.)
- building ontologies

Various criteria for connecting learning objects using both manual and automatic methods may be used (e.g., similar words or keywords in connected learning objects). In some cases the relations are obvious from the structure of the document (e.g. links connecting webpages).

The overall schema of content structure development is shown in Fig 1. The resulting structured information (learning objects and the relations between them) must be stored in some structure. The main part of this paper discusses this part of producing e-learning system. We do not consider the way the information objects (digital resources) are stored and referenced. However, we assume that there exist some methods for accessing and presenting them.

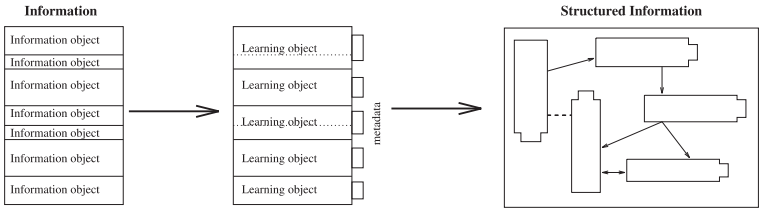


Fig. 1. Structuring the information.

The stored content has to be visualized in some way. The visualization should enable different views on learning content as well as different levels of detail. Furthermore, it must be possible to change, to add and to delete relations and learning objects. Thus we are interested mainly in two aspects of learning content development:

- structuring the information (adding metadata and relations)
- visualization

Both of them are very important and so is the connection between them. There are several approaches to storing the content structure and they offer different possibilities for each part. Some of them have better support for structuring information while others for visualization. We intend to examine these approaches, to describe their advantages and drawbacks and to compare them.

2.2 Requirements for the storage of content structure

As mentioned previously, a content structure consists of learning objects and relations between them. Further we deal only with learning object metadata, as the important part of learning objects. There are some aspects that we have to take into account when we are deciding how to store the content structure.

Storing LOMs: We have to find a suitable format. It should allow to describe different properties of a learning object.

Storing relations: In general relations among objects can be represented by a graph. It is important to consider especially the following issues while choosing a format for storing relations:

- relation properties - Is there a support for differentiating among relations?
- topology - What topology of relations is allowed?
- structure location - Are relations stored together with the corresponding information objects or separately?

Support for visualization: Formats for storing LOMs and relations must support visualization of structured information. LOMs' format must enable easy and fast access to the value of a particular property and format for storing relations should support fast retrieval of relationships between particular learning objects. Thus, the following criteria should be examined:

- queries
- support for visual display of the structure
- support for browsing

3 Content structure representation

3.1 Model of the content structure

A content structure can be represented by a graph. We present a *content structure*, where nodes represent learning objects and edges represent relations between them. We started from the definition of the *Content graph* introduced in [10] where authors consider two sets of learning objects - abstract learning objects and displayable learning objects. Then relations are differentiated according to the type of connected nodes. This approach is useful, but it appears too complex for our present needs. Therefore we introduce a simpler definition as follows.

A *content structure* is a pair $CS = (NodeSet, EdgeSet)$, where $NodeSet$ contains a set of nodes $Node = (ID, resources, nodeProperties)$ and $EdgeSet$ contains a set of edges $Edge = (ID, sNode, tNode, edgeProperties)$. Nodes of the graph represent learning objects, edges are directed and represent semantic relationship between two learning objects. Both of them are identified by an ID and hold some properties - e.g., type of the node/edge, difficulty level of the

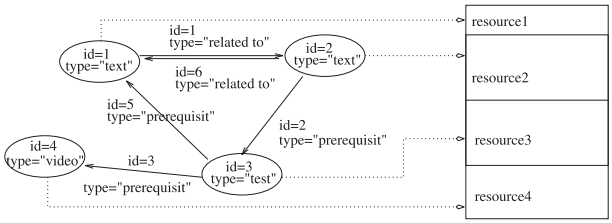


Fig. 2. Simple graph of content structure with property *type* for both nodes and edges.

node. Moreover, node contains references to included digital resources (information objects) and edge contains IDs of source and target node. An example of a content structure graph is shown in Fig. 2.

This simple model of the content structure can be developed further by allowing structure of the nodes. Some of the representations below allow this generalisation and we plan to consider it in the future.

In the rest of this section we shall discuss several approaches to storing content structure of the form defined above. At the end we present a brief summary.

3.2 Relational databases

Database systems are frequently used to store data in information systems. In general information systems can be seen as sets of elements with attributes and relations between them. In a relational database, element attributes are represented by table columns and relations between elements are modeled via references. This view corresponds to our model, where nodes of a content graph are elements, node properties are attributes and edges represent relations. Thus, it is straightforward to represent such a graph by tables in relational database. Relational database enables to store different types of node and edge properties. The topology of the content structure is not restricted, we can model any graph using references. Both ways of storing edges are possible. They can be located in a table together with an element (as its attribute) or in a separate table. Separate storage is necessary if we want to distinguish edges according to some criteria, i.e., if we want to add some properties. There are efficient query languages designed for relational databases (especially SQL) and many interfaces for handling them. Queries are performed fast and they usually return a table. Obviously this format cannot be visualized directly and it does not provide support for browsing[2].

Fig. 3 presents a mapping of the content structure shown in Fig. 2 into a relational database. The edges are stored separately from the nodes, thus we obtain two tables - a table of nodes and a table of edges.

[2] There is only possibility to move between neighbouring records.

id	type	···	resource
1	text		^resource1
2	text		^resource2
3	test		^resource3
4	video		^resource4

id	snode	tnode	type	···
1	1	2	related to	
2	2	3	prerequisit	
3	3	4	prerequisit	
5	3	1	prerequisit	
6	2	1	related to	

Fig. 3. Mapping the content structure into a relational database.

3.3 Directory structure

Files in a hierarchical directory structure represent a simple way of storing learning content. It is one of the oldest alternatives for data storage and all common data in computers are stored in this way. When we consider the content structure as defined above, the nodes of the graph are files. There is no restriction for the type of a file and a file represents a resource itself. The edges are specified by relation directory-subdirectory and create basic tree structure. Usually also cross-references (links) are possible, however this feature depends on the particular file system.

If we store content as documents, several problems arise. Basically, content graph is restricted to a tree. Using links (hard or symbolic) allows extension to any directed graph, but as mentioned above, links are file system-dependent and result in more difficult manipulation with content. Moreover, these two types of edges are the only ones allowed as there is no other way to differentiate among edges. Node properties are restricted to those provided by the file system (permissions, access times, etc.). It is possible to extend this set if we allow a file to contain the description of node properties in some format (e.g. XML), including reference to digital resources. In this way we can specify any node property.

Directory structure provides a possibility of global visualization through common file browser, however the links (cross-references) are not depicted. Browsing is direct within tree structure. The mapping of content structure from Fig. 2 is shown in Fig. 4.

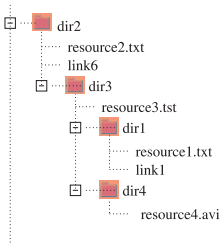


Fig. 4. Mapping the content structure into a directory structure.

Treepad [5] is a tool for managing information in a tree structure. The content is presented similarly to the visualization of directory structure in a file browser. However in this case the underlying format for storing content structure is a mark-up language similar to XML.

3.4 Web space

The web space (or hypermedia space) consists of web pages connected via hyperlinks. Web pages can be either static HTML pages (prepared and stored in advance) or created dynamically through the use of scripting language or created via transformation from other documents. Web space can be locally visualized through web browser and it provides support for local browsing as well.

Creation of static HTML pages is the simplest way of using web space for representing learning content. In this case, no further processing for presentation has to be done. Mapping content graph into web space is quite straightforward. Single web page consists of one or more nodes and hyperlinks represent edges. It is possible to create a hyperlink between two locations within one web page which then corresponds to an edge between two nodes located in this page. Another feature is overlapping of node resource, node properties and edge properties. I.e., simple text resources are usually included in HTML document, while other (pictures, movies) are accessed via links. Edges are stored together with a source node. An example of mapping content structure into web space is shown in Fig. 5.

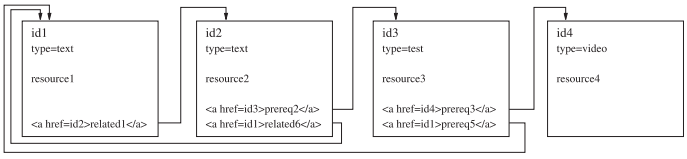


Fig. 5. Mapping content structure into web space.

Drawbacks of this approach result from the usage of HTML. It is a language with fixed constructs and therefore property types of both nodes and edges are restricted. Despite this HTML provides a mechanism to differentiate among edges - using *class*, *id* and *name* attributes for hyperlink. Another problem is that one can get lost in the amount of hyperlinks and web pages. We can avoid this by restricting the content graph into tree or DAG. However, this topology may not be sufficient for modeling all relationships among learning objects. Another solution is not to display web pages directly, but first to perform some transformation. In this case we store content structure in HTML with arbitrary topology of relations, but we can simplify navigation either by reducing the number of displayed hyperlinks or by adding some hierarchical structure (e.g. content of web pages).

Web space is often used for presentation of learning content while learning content structure is stored in another way (database, XML). A dynamical transformation is then performed in order to create displayable web pages[3]. In this case we are not restricted by HTML syntax, but we can take advantage of web space support for presentation. The well-known example of using web space for presenting information space (not specific for e-learning) is wikipedia [9] - web-based, free-content encyclopedia. Some existing e-learning systems (e.g. Moodle [3]) use web space too.

3.5 Structured documents

Structured documents store information decomposed into logical parts. Recently XML (Extensible Markup Language) [8], the standard specified by the W3C consortium, has become very popular as a format for creating structured documents. Logical parts of an XML document are called elements. An element delimits a part of a document and may contain other nested elements. Such a decomposition creates a tree structure of a document. Moreover we can assign attributes to an element, which represent its properties. Unlike HTML, XML is a meta-language and one can define his own class of XML documents, i.e., specify the set of elements and the way they can be nested.

We can represent content graph as an XML document in two basic ways. If a tree structure is sufficient, nodes of content graph are mapped to elements and edges are specified implicitly by element nesting. However, in most cases the tree structure is too restrictive. Another method can also be used - besides nodes also edges are specified explicitly as elements. Manipulating documents becomes more difficult in this case. XML enables to define arbitrary node and edge properties, using either child elements or attribute values. Another strength of XML is a wide selection of manipulation languages, interfaces and frameworks which make development of applications easier.

There exist several languages within the scope of XML which are suitable for representing content structure, e.g. XFML[6], XTM[7], GraphML[1]. In Fig. 6 we show an example of representing content structure from Fig. 2 in the XML format. Obviously there are more possibilities for realizing it. We represent both nodes and edges as elements and their properties as attributes. We omit node references since, as mentioned earlier, this is out of the scope of our interest in this paper.

The drawback of XML is slow access to data since XML documents are text-based. Most commonly XML documents are visualized as HTML pages. We can either use the presentation language CSS which allows XML documents to be directly displayed thorugh a web browser or the transformation language XSLT which enables to specify transformation of an XML document into an HTML document. Also other output formats are possible (e.g., using the language XSL-FO).

[3] Using relational databases and XML for storing content structure is discussed in other parts of this section.

```
<?xml version="1.0" encoding="UTF-8"?>

<content_structure>

    <node id="1" resource="^resource1" type="text">
    <node id="2" resource="^resource2" type="text"/>
    <node id="3" resource="^resource3" type="test"/>
    <node id="4" resource="^resource4" type="video"/>

    <edge id="1" source="1" target="2" type="related to"/>
    <edge id="2" source="2" target="3" type="prerequisit"/>
    <edge id="3" source="3" target="4" type="prerequisit"/>
    <edge id="5" source="3" target="1" type="prerequisit"/>
    <edge id="6" source="2" target="1" type="related to"/>

</content_structure>
```

Fig. 6. A mapping of a content structure into XML.

3.6 Comparison

In Table 1 we list all requirements as defined in Section 2 and we summarize how the described representations fulfill them.

In general, databases offer good support for storing the structure, but visualization cannot be done in a straightforward manner. Directory structure enables global visualization through file browser, however there are significant restrictions both for the topology of the structure and node/edge properties. Web space can be visualized locally through web browser, but only few node/edge properties can be specified. Using XML, any structure and properties can be stored. It is necessary to specify formatting rules explicitly in order to visualize the content structure through web browser[4]. We claim that access to data is slow in the case of directory structure, XML and web space, it is important to mention that this can be improved using indexing techniques.

4 Conclusion

In this work our main concern was the comparison of methods for storing content for e-learning systems. We defined basic requirements divided into three groups - requirements posed on storing learning object metadata, on storing relations and on support for visualization. Four representations suitable for content storage were described and for each of them we outlined both its advantages and disadvantages. At last we summarized our findings in a table and we pointed out how the representations meet defined requirements.

It is easy to see that none of the mentioned approaches fulfills all requirements. Thus it is reasonable to suppose that the optimal solution consists in

[4] Obviously it is possible to create different kinds of displayable formats to any of the listed representations through the use of more or less complex transformation scripts. However we were mostly interested in direct support for visualization, or a visualization achieved by simple transformation.

	databases	**directory structure**	**web space**	**XML**
storing LOMs				
- property types	any	restricted (FS dependent)	restricted	any
storing relations				
- supported topology	any	tree or directed graph (FS dependent)	any	any
- relation types	any	restricted (FS dependent)	restricted	any
- location	included in LOM or separate	implicit or included in LOM	included in LOM	implicit or separate
visualization				
- queries	fast	slow	slow	slow
- visual display	no support	global - file browser	local - web browser	local - web browser
- browsing	no support	direct	moves via hyperlinks	moves via hyperlinks

Tab. 1. Comparison (FS = file system).

combining some of them. Few combinations have been already briefly mentioned (directory structure and XML, web space and XML, web space and databases). In the future we intend to analyse these combinations in more detail in order to achieve elimination of relevant drawbacks and to define a schema for storing content for e-learning meeting our requirements. We plan to specify a graph of the content structure with respect to chosen representation.

References

1. GraphML, `http://graphml.graphdrawing.org/`
2. IEEE, IEEE Standard for Learning Object Metadata – IEEE 1484.12.1-2002, 2002, `http://standards.ieee.org`
3. Moodle, `http://moodle.org`
4. Sharable Content Object Reference Model (SCORM) 2004, 2nd Edition, 2004, `http://www.adlnet.org`
5. Treepad, `http://www.treepad.com`
6. XFML Core - eXchangeable Faceted Metadata Language, 2002, `http://www.xfml.org/spec/1.0.html`
7. XML Topic Maps (XTM), 2001, `http://www.topicmaps.org/xtm/1.0`
8. W3C. Extensible Markup Language (XML) 1.0 (Third edition), W3C Recommendation, 2004, `http://www.w3.org/TR/REC-xml`
9. Wikipedia, `http://wikipedia.org`
10. Wittmann J., Moller D.P.F., The Content-Graph as a Basic Data Structure to Manage Authoring- and Learning Processes. International Conference on Advanced Learning Technologies ICALT'03

Podpora navigácie v otvorených informačných priestoroch*

Katarína Matušíková and Mária Bieliková

Ústav informatiky a softvérového inžinierstva
Fakulta informatiky a informačných technológií
Slovenská technická univerzita v Bratislave
Ilkovičova 3, 842 16 Bratislava, Slovensko
cakanka@gmail.com, bielik@fiit.stuba.sk

Abstrakt Personalizovaná navigácia zahŕňa skupinu techník, ktoré podporujú navigáciu používateľa vo virtuálnych priestoroch prispôsobenú jeho cieľom, preferenciám, znalostiam a d'alším charakteristikám. V príspevku diskutujeme možnosti realizácie personalizovanej navigácie v otvorenom informačnom priestore využitím prvkov sociálnej navigácie. Navigácia v tomto prípade prebieha na základe "kolektívnych znalostí" získaných sledovaním skupiny používateľov s podobnými cieľmi. Otvorený informačný priestor, na rozdiel od uzavretého, ktorého štruktúru tvorca systému pozná, poskytuje stále nové informačné zdroje, čo zvyšuje potrebu podpory efektívnej navigácie. Zároveň je však náročnejšia realizácia takejto navigácie. Hlavným prínosom je návrh metódy personalizovanej navigácie, ktorá vychádza zo sémantického opisu zdrojov ontológiou. Navrhnutú metódu prezentujeme v doméne informačného priestoru pracovných ponúk.

1 Úvod

Adaptívne hypermediálne systémy zachytávajú niektoré charakteristiky používateľa a s ich využitím zobrazujú najvhodnejšiu časť informačného obsahu. Charakteristiky používateľa (napr. preferencie, znalosti a záujmy) sa reprezentujú modelom používateľa. Prispôsobovanie sa realizuje poskytovaním adaptívnej podpory navigácie a adaptívnej prezentácie [1].

V príspevku sa venujeme možnostiam adaptívnej podpory navigácie pre individuálneho používateľa – personalizovanej navigácii s využitím opisu zdrojov ontológiou. Základná myšlienka personalizovanej navigácie spočíva v prispôsobení odkazov tak, aby viedli používateľa k cieľom jeho záujmov, relevantným informáciám alebo aby sa vyhol nepodstatným informáciám. V súčasnosti sa používa niekoľko techník personalizovanej navigácie (vytvorených pre aplikácie pracujúce v uzatvorených informačných priestoroch) medzi ktoré patria priame vedenie, adaptívne usporadúvanie odkazov, adaptívne skrývanie odkazov, adaptívna anotácia odkazov a adaptácia máp a generovanie odkazov [3], [11].

* Táto práca bola podporovaná Vedeckou grantovou agentúrou, grant č. VG1/ 0162/03 a Agentúrou na podporu vedy a techniky na základe Zmluvy č. APVT-20-007104.

Tieto techniky sú založené na znalostiach expertov, ktoré sa poskytujú v rámci obmedzenej množiny informačných zdrojov. Metóda personalizovanej navigácie, ktorú v príspevku prezentujeme, predstavuje alternatívny prístup k personalizovanej navigácii a je použiteľná v otvorenom informačnom priestore, ktorý podlieha častým zmenám obsahu. Vychádza z myšlienok sociálnej navigácie, ktoré spája s prístupom využívajúcim opis sémantiky zdrojov.

V príspevku analyzujeme možnosti personalizovanej navigácie v otvorenom informačnom priestore. Prezentujeme myšlienky sociálnej navigácie využívajúcej sémantiku a ukážeme aplikáciu tejto metódy v doméne informačného priestoru pracovných ponúk, ktorá slúžila ako doména pre experimentálne overenie navrhnutej metódy.

2 Personalizovaná navigácia v otvorenom informačnom priestore

Napriek tomu, že adaptívne hypermediálne systémy preukázali silu personalizovanej navigácie v množstve aplikačných oblastí, dodnes celkom neuspeli v úsilí vytvoriť personalizovanú navigáciu široko dostupnú v rámci Internetu. Je to dané najmä tým, že súčasné adaptívne hypermediálne aplikácie sú prevažne orientované na uzavretý informačný priestor. Priestor zdrojov týchto systémov predstavuje uzatvorenú množinu informačných položiek spravidla manuálne indexovaných expertmi v danej oblasti. Celý informačný priestor je dobre štruktúrovaný a často už pri tvorbe obsahu obohatený metadátami.

Metódy personalizácie v takýchto hypermediálnych aplikáciách vychádzajú zo znalostí o štruktúre priestoru a obsahu informačných zdrojov v tomto priestore a sú použiteľné iba v rámci obmedzenej množiny zdrojov. Preto klasické adaptívne hypermediálne systémy nie sú priamo použiteľné v otvorenom informačnom priestore. V otvorenom informačnom priestore sa personalizovaná navigácia musí vyrovnávať s viacerými problémami:

- *obrovský informačný potenciál* – určenie a výber zdrojov z takmer neobmedzeného informačného priestoru a získanie informácií z týchto zdrojov v potrebnej forme;
- *získanie metadát* – získanie ohodnotenia zdrojov a informácií z hľadiska kvality a vhodnosti pre používateľa;
- *často sa meniaci obsah* – zabezpečenie aktualizácie už získaných informácií, uchovávanie znalostí a metadát;
- *orientácia v informačnom priestore* – vytvorenie siete, na základe ktorej sa uskutočňuje navigácia.

K riešeniu uvedených problémov možno pristupovať tak, že adaptívne webové aplikácie, ktoré poskytujú podporu používateľovi v otvorenom priestore, budú schopné v určitom rozsahu "porozumieť" obsahu a odkazom bez toho, aby bol do procesu porozumenia zapojený človek (alebo bol zapojený len minimálne).

Ďalšie prístupy k riešeniu prezentujú webové odporúčacie systémy, ktoré sú schopné získať informácie z dokumentov v otvorenom priestore na základe syntaktickej analýzy zdrojov priestoru (obsahovo orientované filtrovanie, napr. [9]).

Iným mechanizmom personalizácie, ktoré webové odporúčacie systémy používajú je sociálna navigácia, ktorá vychádza zo získavania "kolektívnych znalostí" – metadáta o zdrojoch priestoru sa získavajú nepriamo sledovaním používateľov systému.

Existujúce metódy personalizovanej navigácie v otvorenom informačnom priestore sú založené na transformácii určitej časti otvoreného priestoru na uzavretý, pričom získavajú znalosti o štruktúre a obsahu zdrojov tejto časti [2].

Obsahovo orientovaná navigácia. Zdrojom prispôsobovania metódou obsahovo orientovanej navigácie sú pozorované aktivity používateľa, ktoré zachytávajú jeho záujmy (napr. označovanie odkazov alebo explicitné ohodnotenie dokumentu). Obsahovo orientovaná personalizácia vychádza z reprezentácie dokumentov na úrovni slov a profilu používateľa z informácií získaných prieskumom. Personalizácia sa zvyčajne realizuje zapojením techník strojového učenia.

Sociálna navigácia. Sociálna navigácia vychádza z interakcií predchádzajúcich používateľov so systémom, ktoré sa zaznamenávajú pomocou rôznych foriem spätnej väzby a tak sa vytvárajú "kolektívne znalosti". Navigácia prebieha na základe kolektívnej znalosti veľkej skupiny používateľov. Pri tomto prístupe sa môžu vytvárat' skupiny používateľov s podobnými záujmami. Medzi sociálne navigačné techniky sa zaraďuje napr. kolaboratívne filtrovanie alebo SANS (angl. social adaptive navigation support [4]), ktoré sa líšia najmä mierou vplyvu modelu používateľa (individuálneho používateľa) na personalizáciu.

Navigácia využívajúca opis sémantiky zdrojov. Výzvou pre oblasť otvorených informačných priestorov je výskum a vývoj v oblasti webu so sémantikou. V adaptívnych hypermediálnych systémoch, ktoré využívajú web so sémantikou je personalizovaná navigácia realizovaná na základe opisu zdrojov pomocou metadát [10]. Predpokladá, že zdroje v rámci otvoreného informačného priestoru obsahujú sémantickú anotáciu v rámci súčasných technológií webu so sémantikou.

Obsahovo orientovaná navigácia a sociálna navigácia majú pri odvodzovaní najviac relevantných stránok d'aleko od presnosti klasických adaptívnych hypermédií. Napriek tomu, že umožňujú prácu s obrovským množstvom informácií, ktoré sa nachádzajú v otvorenom informačnom priestore [7] stále neposkytujú používateľom webu plnú podporu personalizovanej navigácie tak, ako ju doteraz poskytovali klasické adaptívne hypermediálne systémy.

Generovanie sémantických anotácií je možné aj štatistickými a lingvistickými metódami analýzy textu a hypertextových dokumentov. Môžeme ich úspešne aplikovat' na súbory dobre štruktúrovaných dokumentov ako sú napríklad Java API alebo Wiki [6]. Práve nutnosť štruktúrovanosti a sémantickej anotácie dokumentov je najväčšou slabinou prístupov využívajúcich opis sémantiky zdrojov. Na druhej strane technológie webu so sémantikou umožňujú podporovat' personalizovanú navigáciu oveľa presnejším spôsobom.

3 Sociálna navigácia využívajúca sémantiku

Sociálna navigácia je založená na známom sociánom fenoméne, že ľudia majú tendenciu nasledovať "stopy" iných. V otvorenom informačnom priestore môžeme týmto princípom nahradiť znalosti expertov, ktoré sa používajú pre navigáciu v uzatvorenom informačnom priestore. Na základe uvedenej myšlienky sme navrhli metódu personalizovanej navigácie. Metóda vychádza z princípov sociálnej navigácie, teda znalosti potrebné pre efektívnu podporu navigácie získavame na základe sledovania skupín používateľov s podobnými záujmami.

Otvorený informačný priestor je rozsiahly. Vhodným spôsobom podpory používateľa v tomto priestore sa javí použitie máp a orientačných bodov [5], ktoré svoju praktickosť stále dokazujú v reálnom svete. V kontexte mapy sa blízkosť (v našom prípade sémantická podobnosť informácií) premieta do ich vzájomnej polohy na mape. Mapa zlučuje podobné informačné fragmenty a vytvára tak podpriestory. Jednotlivé časti, zobrazené v mape musia byť opísané tak, aby sa podľa nich vedel používateľ zorientovať. Názov musí vhodne opisovať podpriestor v danej časti mapy. Skutočné mapy majú rôzne mierky, rôzna mierka mapy informačného priestoru si vyžaduje hierarchickú štruktúru zdrojov, aby zobrazovali podpriestory na rôznej úrovni abstrakcie. Hierarchická štruktúra umožňuje jednoduché zobrazenie priestoru a pohyb v tomto priestore. Umožňuje upresniť časť priestoru, v ktorej sa nachádza hľadaná informácia, ale aj získať informácie z príbuzných oblastí priestoru.

Vytvorenie hierarchickej štruktúry informačného priestoru znamená definovanie množiny informačných fragmentov a hierarchie týchto množín. Množina tu predstavuje podpriestor celého informačného priestoru. Sociálna navigácia – zaznamenávanie a zobrazovanie "stôp skupiny" – sa realizuje nielen nad informačnými fragmentami, ale aj nad množinami týchto fragmentov. Premieta sa do mapy priestoru označením podpriestorov orientačnými značkami.

Hierarchickú štruktúru informačného priestoru dosiahneme využitím opisu informačných zdrojov pomocou ontológie. Ontológia umožňuje definovať hierarchiu medzi jednotlivými objektmi (objektom môže byť používateľ, informácia), ich vlastnosti a vzájomné vzťahy. V súlade s modelmi, ktoré sa vytvárajú pre adaptívne hypermédiá, definujeme pre každý model samostatnú ontológiu – model domény, model používateľa, model prispôsobenia. Podobný prístup je použitý v [8], kde sa definuje ešte aj ontológia pozorovaní, ktorá opisuje pozorovania používateľa počas používania systému. My sme tieto vlastnosti zahrnuli priamo do ontológie používateľa, keďže modelujeme iba pozorovania vzťahujúce sa k jednotlivým používateľom.

Opis vlastností a vzájomných vzťahov ontológiou zabezpečí oddelenie charakteristík informačných fragmentov od samotného obsahu informačných fragmentov. Môžeme tak získané znalosti o vhodnosti informačných fragmentov uchovávať aj keď pôvodná informácia, na základe ktorej bola znalosť získaná, už nie je aktuálna alebo zanikla. Znalosti o vhodnosti viažeme na informačné fragmenty, aj na charakteristiky informačných fragmentov a používame ich na ohodnotenie nových informačných fragmentov s podobnými charakteristikami. Takto sa vyrovnáme aj s častou zmenou obsahu informačného priestoru.

Aby sme mohli používateľovi poskytnúť čo najlepšiu podporu navigácie, systém zobrazuje pohľad na informačný priestor z rôznych dimenzií. *Dimenzia priestoru* je charakteristika informačných fragmentov, v rámci ktorej systém naviguje používateľa. Metóda personalizovanej navigácie, ktorú sme navrhli:

- *vychádza zo sémantického opisu ontológiou* – vďaka čomu získame hierarchickú štruktúru informačných fragmentov a dokážeme oddelit' samotný informačný fragment od jeho charakteristík a metadáta (ohodnotenie) pre personalizáciu viazat' nielen na informáciu, ale aj na jej charakteristiky;
- *využíva princípy sociálnej navigácie* – navigácia sa uskutočňuje na základe "kolektívnych znalostí" skupiny používateľov s podobnými záujmami a cieľmi;
- *kombinuje viaceré techniky navigácie* – využíva d'alšie techniky navigácie používané v uzatvorenom informačnom priestore ako adaptívna anotácia, generovanie odkazov a adaptívne usporadúvanie.

Predpokladom personalizácie je vytvorenie skupín používateľov na základe zistenej vzájomnej podobnosti ich cieľov a záujmov. Personalizácia sa vykonáva v niekoľkých krokoch:

1. Zaznamenanie prístupu používateľa – v ontológii sa vytvorí nový objekt, ktorý opisuje prístup používateľa do systému.
2. Odvodenie a zaznamenanie dôsledkov pre skupinu – obnovenie "kolektívnych znalostí" skupiny pre objekty, ku ktorým používateľ pristúpil (teda nielen vhodnosti informačného fragmentu, ale aj jeho vlastností, príp. súvisiacich informačných fragmentov). Tým sa získaná znalost' naviaže nielen na informáciu, o ktorú prejavil používateľ záujem (napr. pracovnú ponuku), ale aj na jej vlastnosti. V prípade neaktuálnosti informácie (napr. zániku pracovnej ponuky v systéme) môžeme súvisiace a odvodené znalosti d'alej používat'.
3. Nájdenie všetkých podpriestorov v danej dimenzii v prezeranej časti informačného priestoru na danej úrovni abstrakcie.
4. Získanie skupinových odporúčaní pre tieto podpriestory.
5. Získanie odporúčaní týchto podpriestorov pre aktuálneho jednotlivca.
6. Zobrazenie výsledkov obohatených o personalizované odporúčania mapou priestoru.

Je dôležité používateľa nezahltit' príliš podrobným opisom jednotlivých častí priestoru. Preto treba zvolit' pre používateľa relevantné dimenzie priestoru. Navigácia sa vo veľkej miere realizuje vizualizáciou pomocou farieb a ikôn (záujem skupiny, záujem používateľa, veľkost' množiny).

4 Doména informačného priestoru pracovných ponúk

Experimentálne overenie myšlienok sociálnej navigácie využívajúcej sémantiku sme realizovali v doméne pracovných ponúk. Informačný priestor pracovných ponúk je charakteristický častou zmenou priestoru a relatívne krátkej dobe platnosti informačného fragmentu, ktorým je pracovná ponuka. Kladie veľký dôraz

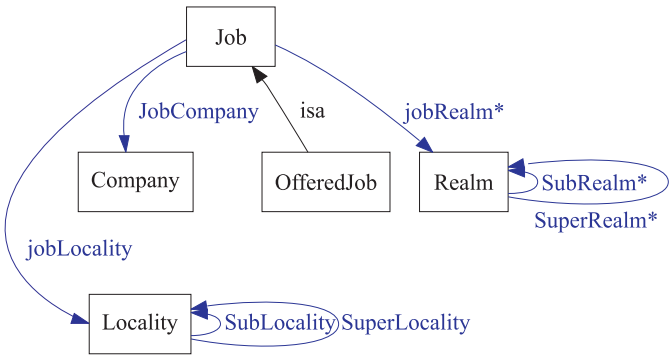
na schopnosť navigácie vyrovnávať sa so zmenami, preto je vhodnou doménou na overenie navrhnutej metódy.

Používatelia v tomto priestore sú charakterizovaní "hľadanými" pracovnými ponukami a podľa týchto charakteristík sa zarad'ujú do skupín, v ktorých sa realizuje sociálna navigácia. Používateľ vidí priestor zobrazený mapou priestoru zloženou z buniek, ktoré sú definované hodnotami v rôznych dimenziách. Každá bunka predstavuje množinu pracovných ponúk, ktoré spĺňajú hodnoty dimenzií. Používateľ sa v priestore pohybuje klikaním na navigačné značky pri názve dimenzie, čím spôsobuje zmenu mierky mapy v danej dimenzii. Pohyb používateľa sa sleduje zaznamenávaním prístupov. Používateľ má v každom kroku možnosť nechať si zobrazit' konkrétne pracovné ponuky, ktoré sa nachádzajú v danej množine ponúk (bunke mapy).

4.1 Modely pre realizáciu adaptívnej navigácie

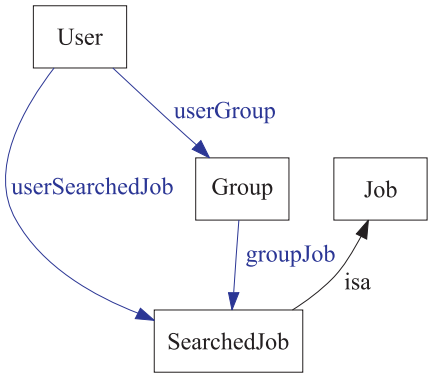
Model domény. Každá pracovná ponuka je v ontológii opísaná vlastnosťami. Tieto sú rozdelené na vlastnosti, ktoré nadobúdajú konkrétnu hodnotu z príslušného oboru hodnôt (dátové vlastnosti, ako napr. mzda) a vlastnosti, ktoré sa viažu na d'alší objekt (objektové vlastnosti, ako napr. lokalita, profesná oblasť). Schematický model domény (pozri obr. 1) zobrazuje pracovnú ponuku (*OfferedJob*), súvisiace triedy a ich vzájomné vzťahy – dedenie (čierna čiara reprezentujúca vzťah *isa*) a objektové vlastnosti (modré čiary).

Objektové vlastnosti vytvárajú hierarchickú štruktúru svojou definíciou (relácie s prefixom *sub* a *super*). Zmena mierky je v tomto prípade explicitne zadaná.



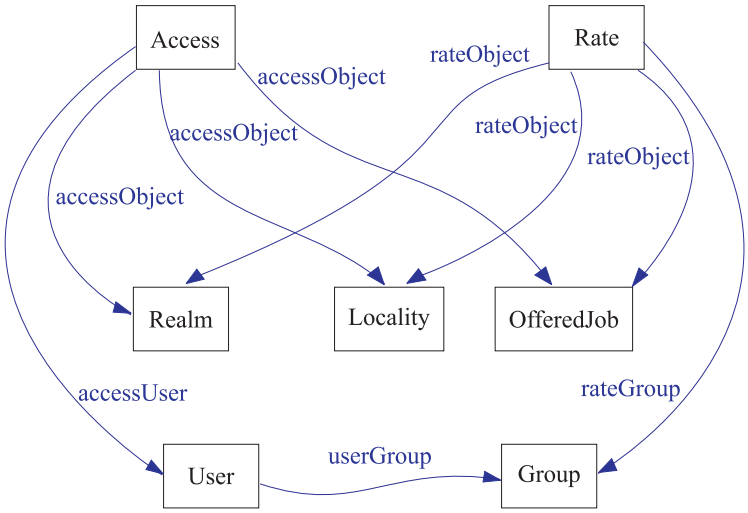
Obr. 1. Schematický model domény.

Model používateľa. Dáta o používateľovi, jeho ciele, preferencie a d'alšie charakteristiky sú opísané triedou *SearchedJob* (pozri obr. 2), ktorá ich zaznamenáva podobným spôsobom ako ponúkané pracovné ponuky. Na základe týchto informácií je používateľ (*User*) zaradený do skupiny (*Group*), ktorá je tiež opísaná triedou *SearchedJob* – teda spoločnými preferenciami a vlastnosťami používateľov v tejto skupine.



Obr. 2. Schematický model používateľa.

Model prispôsobenia. V procese prispôsobovania navigácie používateľovi sa zaznamenávajú informácie o prístupoch (*Access*) používateľa do informačného priestoru, na základe ktorých sa vytvára ohodnotenie (*Rate*) jednotlivých častí priestoru (pozri obr. 3). Podľa ohodnotenia sa realizuje sociálna navigácia.



Obr. 3. Schematický model prispôsobenia.

V súvislosti s prístupom používateľa sa udržuje informácia o trvaní prístupu (*accessDuration*), pričom je stanovený limit maximálneho času. Ak prístup trval dlhšie ako stanovený limit, zaznačí sa hodnota limitu a tiež dátum prístupu (*accessDate*).

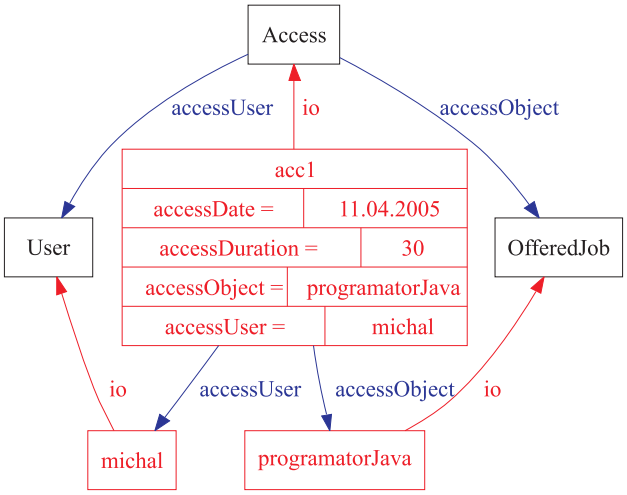
Ohodnotenie *Rate* okrem skupiny a objektu obsahuje informácie o počte prístupov všetkých používateľov skupiny k danému objektu (*rateAccesses*), sumárny čas trvania prístupov používateľov skupiny k objektu v sekundách (*rateDuration*), priemerný "vek" prístupov v dňoch (*rateAgeAverage*), pričom sa berie do

úvahy dĺžka prístupu (teda prístup, ktorý trval krátko ovplyvní priemerný "vek" prístupov menej) a samotné ohodnotenie (*rateValue*), ktoré je úmerné dĺžke trvania prístupov a nepriamo úmerné priemernému "veku"prístupov.

Skupina (*Group*) uchováva informácie o počte prístupov používateľov skupiny do systému (*groupAccesses*) a trvanie všetkých prístupov používateľov skupiny spolu (*groupAccessDuration*).

4.2 Realizácia sociálnej navigácie využívajúcej sémantiku

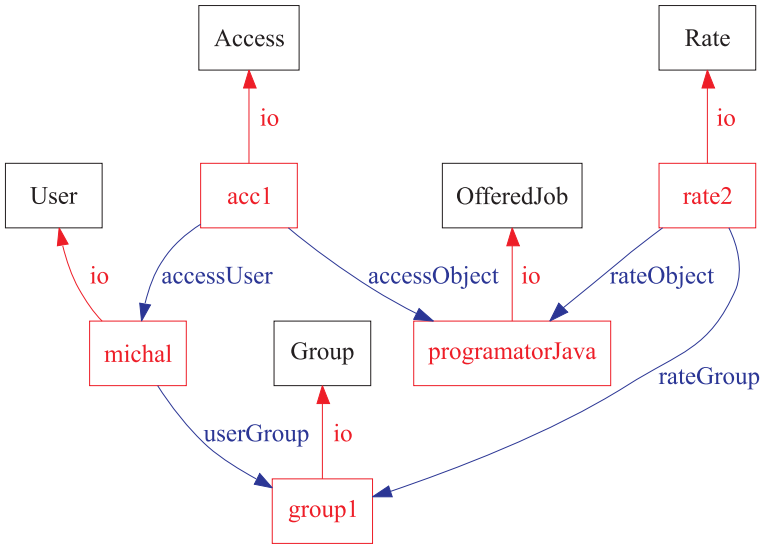
Prístup používateľa k nejakej časti priestoru sa zaznamená v ontológii tým, sa vytvorí nový objekt triedy *Access* a nastavia sa jeho vlastnosti (pozri obr. 4).



Obr. 4. Príklad zaznamenania prístupu používateľa 'michal' k pracovnej ponuke 'programator Java'.

Ďalším krokom je odvodenie dôsledkov pre skupinu (pozri obr. 5), pričom sa obnoví ohodnotenie skupiny k danému objektu. Posledným krokom zaznamenania prístupu je získanie *Rate* všetkých objektov, ktoré sú opisom (priamo alebo nepriamo) pristúpeného objektu. Ohodnotenia týchto objektov sa tiež zvýšia. Tým sa naviazala získanú znalosť nielen na pracovnú ponuku, o ktorú prejavil používateľ záujem, ale aj na jej vlastnosti. V prípade zániku pracovnej ponuky tieto znalosti môžeme ďalej používať. Po zaznamenaní prístupu nasleduje nájdenie všetkých podpriestorov prezeranej časti informačného priestoru vo zvolenej dimenzii (teda napr. nájdenie všetkých sublokalít európy). K nájdeným podpriestorom získame ich ohodnotenia pre skupinu (objekty triedy *Rate*) a pre aktuálneho používateľa. Výsledky sú zobrazené mapou priestoru.

Navrhnutú metóda personalizovanej navigácie sme overili vytvorením prototypu systému, ktorý túto metódu aplikuje nad navrhnutou ontológiou. Ontológiu



Obr. 5. Zvýšenie hodnotenia 'rate2' k objektu 'programatorJava' pre skupinu 'group1', do ktorej patrí používateľ 'michal'.

sme testovali na umelo vytvorených testovacích dátach. Prototyp je implementovaný ako webová aplikácia tenký klient. Používateľovi po prihlásení zobrazí informačný priestor mapou priestoru najvyššej úrovne a umožní mu pohybovať sa priestorom. Mapa priestoru je vizualizovaná tabuľkou, ktorej bunky sú označené podľa množín pracovných ponúk. Personalizovaná navigácia sa realizuje informáciami o vhodnosti buniek, zatiaľ vypísaných textovou formou. V prototype sú zobrazované dimenzie lokalita a profesná oblasť. Používateľ sa v priestore pohybuje šípkami, ktoré sú zobrazené pri názve hodnoty dimenzie. V najpodrobnejšom priblížení systém vypíše zoznam pracovných ponúk a informácie o nich spolu s informáciami o ich vhodnosti.

Prototyp overuje, že navrhnutou metódou je možné realizovať personalizovanú navigáciu, založenú na "kolektívnych znalostiach" nad ontológiou. Navrhnutá metóda personalizovanej navigácie je použiteľná aj na iné domény s podobnými vlastnosťami. Pri jej aplikácii v podobnej oblasti je potrebné prispôsobiť doménový model.

5 Zhrnutie a ďalšia práca

Navrhnutá metóda podporuje navigáciu v otvorenom informačnom priestore. Vysporiadava sa aj častou zmenou v tomto priestore. Navrhnuté štruktúrovanie informačného priestoru umožňuje efektívnu navigáciu a tiež zachovanie znalostí a ich opätovné použitie v meniacom sa informačnom priestore.

Získavanie metadát navrhnutá metóda rieši formou sociálnej navigácie. Používatelia sú zadelení do skupín, v rámci ktorých sa podieľajú na vytváraní "kolektívnych znalostí". Tie sa využívajú pri navigácii používateľa ku skupinám pra-

covných ponúk, ktoré sú pre neho vhodné a o ktoré má záujem. Uchovávanie metadát v informačnom priestore s častými zmenami obsahu sme riešili oddelením informačných fragmentov a ich charakteristík, pričom získané metadáta (ohodnotenie) sme viazali na tieto fragmenty aj na ich charakteristiky.

Hlavnými oblast'ami, ktorými sa chceme zaoberat' v d'alšej práci je automatizované zist'ovanie preferencií používateľov a automatizované vytváranie skupín. Môžeme sledovat' pohyb používateľa po systéme, prejavený záujem analyzovat' a tak určovat' preferencie používateľa, na základe ktorých je zaradený do skupiny. Takto by bolo možné preradit' používateľa do inej skupiny v prípade, že sa jeho záujmy zmenia. Ďalej sa plánuejme zaoberat' možnost'ami navigácie v rámci rôznych dimenzií. Chceme analyzovat', ktoré dimenzie sú pre používateľa rozhodujúce a tieto pri zobrazovaní uprednostňovat'.

Referencie

1. M. Bieliková. Adaptívna prezentácia hypermédií na webe. In *Proc. of DATAKON 2003*, L. Popelínský (Ed.), Brno, October 18-21, pp.72-91.
2. P. Brusilovsky. Adaptive Navigation Support: From Adaptive Hypermedia to the Adaptive Web and Beyond. *PsychNology Journal*, 2(1):7–23, 2004.
3. P. Brusilovsky. Methods and techniques of adaptive hypermedia. *User Modeling and User Adapted Interaction*, 6(2-3):87–129, 1996.
4. P. Brusilovsky, G. Chavan, and R. Farzan. Social Adaptive Navigation Support for Open Corpus Electronic Textbooks. In *Proc. of 3rd Int. Conf. on Adaptive Hypermedia and Adaptive Web-Based Systems*. pages 24–33, Springer-Verlag, LNCS 3137. Eindhoven, The Netherlands, August 2004.
5. P. Brusilovsky and R. Rizzo. Using maps and landmarks for navigation between closed and open corpus hyperspace in Web-based education. *The New Review of Hypermedia and Multimedia*, 9:59–82, 2002.
6. P. Dolog, N. Henze, W. Nejdl, and M. Sintek. The Personal Reader: Personalizing and Enriching Learning Resources using Semantic Web Technologies. In *Proc. of 3rd Int. Conf. on Adaptive Hypermedia and Adaptive Web-Based Systems*. pages 85–94, Springer-Verlag, LNCS 3137. Eindhoven, The Netherlands, August 2004.
7. R. Farzan. *Social Adaptive Navigation Support for Open-Corpus Elearning.* Independent study report. Univ. of Pittsburgh. 2004. [online]. [cit. 2005-06-06]. <http://www2.sis.pitt.edu/~peterb/indepstudies/farzan042.pdf>
8. N. Henze and M. Kriesell. Personalization Functionality for the Semantic Web: Architectural Outline and First Sample Implementations. In *First Int. Workshop on Engineering the Adaptive Web (EAW 2004)*, Eindhoven, The Netherlands, August 2004. [online]. [cit. 2005-06-06]. <http://www.kbs.uni-hannover.de/Arbeiten/Publikationen/2004/eaw2004.pdf>
9. G. Polčicová, P. Návrat. Semantic Similarity in Content-based Filtering. In *Proc. of ADBIS 2002 - Advances in Databases and Information Systems*, pages 80-85, Springer Verlag, LNCS 2435, Bratislava, Slovakia, September 2002.
10. V. Svátek. Ontologie a WWW. In *Proc. of DATAKON 2002*, pages 27–55, Brno, October 19 - 22. pp. 27-55.
11. H. Wu and P. De Bra. Link-Independent Navigation Support in Web-Based Adaptive Hypermedia. In *Proc. of the 11th Int. World Wide Web Conf.*, Hawai, USA, May 2002. [online]. [cit. 2005-06-06]. <http://www2002.org/CDROM/alternate/684/>

Identifikácia zmien v ontológiách*

Michal Tury and Mária Bieliková

Ústav informatiky a softvérového inžinierstva
Fakulta informatiky a informačných technológií
Slovenská technická univerzita v Bratislave
Ilkovičova 3, 842 16 Bratislava, Slovensko
michal@tury.sk, bielik@fiit.stuba.sk

Abstrakt Podobne ako obsah súčasného webu, aj ontológie ako súčasť webu so sémantikou podliehajú v čase zmenám. Zmeny zahŕňajú pridávanie, rušenie a modifikáciu elementov v ontológii na sémantickej aj na štrukturálnej úrovni. Takéto zmeny môžu viesť k nesprávnym uzáverom, ale aj k prípadnému znefunkčneniu systémov, ktoré využívajú ontológie na odvodzovanie. V príspevku opisujeme návrh metódy automatizovanej kontroly aktuálnosti prezentovaných informácií vrátane metadát opísaných ontológiami. Metóda vychádza z identifikácie zmien medzi verziami ontológií. Jej úlohou je nielen identifikovať zmeny, ale aj ekvivalentné elementy ontológií, čím sa dá dosiahnuť napr. možnosť sledovania informačných zdrojov v čase a aj znovupoužiteľnosť softvérových komponentov v rôznych aplikáciách. V príspevku opisujeme možnosti automatizovanej identifikácie zmien medzi verziami ontológií pomocou heuristík a ich overenie softvérovým nástrojom pre porovnávanie ontológií.

1 Úvod

Existuje množstvo štúdií, ktoré sa zaoberajú skúmaním zastarávania a evolúcie webu [1], [2]. Podľa výsledkov Online Computer Library Center [3] sa približne 50% webových stránok v rozmedzí rokov 1998 až 2002 stane každoročne nedostupnými. Je možné predpokladať, že v prípade rozšírenia webu so sémantikou, by jeho evolúcia mohla prebiehať podobne rýchlo ako v prípade súčasného webu (ak nie rýchlejšie). To je aj dôvodom, prečo je vhodné premýšľať o spôsobe, ako zabezpečiť, aby boli metadáta v každom okamihu konzistentné s obsahom, ktorý opisujú. Ontológie, ktoré sú súčasťou webu so sémantikou, pozostávajú z tried, vlastností, indivíduí a vzťahmi medzi nimi, ktoré modelujú nejakú časť sveta. Samotnú ontológiu môžu vytvárať viacerí ľudia, pričom každý z nich môže rovnakú časť sveta modelovať (a aj vnímať) iným spôsobom. Dokonca aj v prípade, že ontológie vytvára jeden človek, môže vytvoriť v čase viacero verzií jednej ontológie.

V tomto príspevku opisujeme návrh metódy automatizovanej kontroly aktuálnosti prezentovaných informácií vrátane metadát vyjadrených ontológiami. Metóda vychádza z identifikácie zmien medzi verziami ontológií. Jej úlohou nie je len

* Táto práca bola podporovaná Vedeckou grantovou agentúrou, grant č. VG1/ 0162/03 a Agentúrou na podporu vedy a techniky na základe Zmluvy č. APVT-20-007104.

identifikovať zmeny, ale aj ekvivalentné elementy ontológií, čím sa dá dosiahnuť napr. znovupoužiteľnosť softvérových komponentov a aj možnosť sledovania informačných zdrojov v čase. V príspevku opisujeme možnosti automatizovanej identifikácie zmien medzi verziami ontológií pomocou heuristík a ich overenie softvérovým nástrojom pre porovnávanie ontológií.

2 Identifikácia zmien

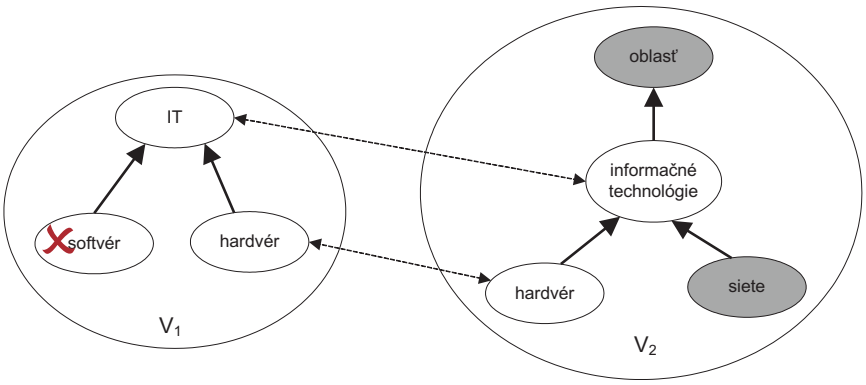
Pri kontrole aktuálnosti informácií sa porovnávajú dve verzie. Medzi týmito dvoma verziami treba identifikovať zmeny a identifikovať časti verzií, ktoré sú identické, t.j. určiť mapovanie týchto častí. Identifikáciu zmien medzi verziami možno charakterizovať aj tak, že treba nájsť časti, ktoré boli upravené, časti, ktoré boli pridané a časti, ktoré boli odstránené.

V prípade ontológií sa dá úloha identifikácie zmien rozdeliť na sémantickú a štrukturálnu úroveň. V prípade *sémantickej úrovne* sa kontroluje tá časť ontológie, ktorá definuje triedy, ich sloty, definičné obory, obory hodnôt, vzťahy medzi triedami, obmedzenia a pod. *Štrukturálnu úroveň* predstavujú inštancie/indivíduá tried definovaných na sémantickej úrovni. Kontrola na štrukturálnej úrovni zahŕňa nielen kontrolu konzistencie indivíduí s triedami, ktorých sú inštanciami, ale tiež hodnoty indivíduí v kontexte tried, hodnoty ich slotov a tiež porovnanie indivíduí medzi dvoma verziami a identifikovanie rovnakých, upravených, pridaných a odstránených indivíduí.

Pri identifikácii zmien je možné využiť niekoľko techník, ktorými sa táto identifikácia dá realizovať:

- *hľadanie identických prvkov ontológií* – najjednoduchší spôsob založený na porovnávaní štruktúr, objektov, reťazcov a vyhľadávaní zhôd,
- *heuristické metódy* – dopĺňa predchádzajúcu metódu o prvky, pravidlá a algoritmy, ktoré na základe nedokonalej či neúplnej doplnkovej informácie vyvodzujú isté závery,
- *metódy výpočtovej inteligencie*, ako napr. neurónové siete, genetické algoritmy.

Uvedené metódy umožňujú identifikovať rozdiely medzi dvoma verziami, pričom tieto rozdiely môžu mať rôznu povahu. Príklad identifikovaných zmien s využitím uvedených operácií je znázornený na obr. 1. Obrázok znázorňuje dve verzie ontológie na sémantickej úrovni. Plnými šípkami je znázornený vzťah medzi triedou a podtriedou, napr. vo verzii V_1 je trieda *hardvér* podtriedou triedy *IT*. Čiarkované šípky znázorňujú triedy, ktoré sú navzájom mapované, t.j. ich sémantika vo verziách je ekvivalentná. Napr. trieda *hardvér* vo verzii V_1 je mapovaná na triedu *hardvér* vo verzii V_2. Môžeme si taktiež všimnúť, že mapovanie nemusí existovať len medzi triedami, ktoré sú identické. Príkladom takéhoto mapovania je mapovanie medzi triedami *IT* vo verzii V_1 a *informačné technológie* vo verzii V_2. Tieto triedy majú rovnakú sémantiku, ale ich názov je rozdielny. Trieda *softvér* vo verzii V_1 je označená krížikom, pretože je vzhľadom



Obr. 1. Príklad identifikovaných zmien na sémantickej úrovni medzi dvoma verziami ontológie.

na verziu V_2 odstránená. Analogicky sú triedy *oblasť* a *siete* vo verzii V_2 vyznačené šedou farbou, pretože sú pridané vo verzii V_2 vzhľadom na verziu V_1.

Okrem už známych pojmov ako trieda, slot, indivíduum budeme používať tieto pojmy:

- *element* – akákoľvek časť ontológie, ktorej zvyčajne hľadáme ekvivalent,
- *typ elementu* – bližšie určuje, o aký element sa jedná, t.j. určuje, či je element triedou, slotom, indivíduom a pod.,
- *ekvivalent* – element E_2 vo verzii V_2, ktorý má rovnakú sémantiku ako element E_1 vo verzii V_1.

Uvedený príklad nám umožňuje povedať, že identifikované zmeny je možné definovať troma operáciami:

- *pridanie* – element sa nachádza vo verzii V_2 a nenachádza sa vo verzii V_1,
- *mapovanie* – určuje, ktorý element z verzie V_1 je sémanticky ekvivalentný elementu vo verzii V_2,
- *odobratie* – element sa nachádza vo verzii V_1 a súčasne sa nenachádza vo verzii V_2.

Tieto operácie môžu slúžiť na uchovávanie rozdielov medzi dvoma verziami ontológie. Samozrejmosťou je, že tieto operácie musia obsahovať údaje o tom, medzi ktorými dvoma verziami je operácia definovaná a taktiež aj údaje, na ktorú časť, resp. elementy verzie, sa vzťahuje.

V prípade operácií mapovania a odobratia je táto úloha jednoduchá, pretože stačí identifikovať konkrétnu triedu, slot alebo indivíduum. V prípade operácie pridania, musí táto obsahovať aj všetky potrebné informácie, ktorými sa daný pridaný element dá definovať. Naviac môžu byť opisy operácií doplnené aj o informácie o dôvodoch, autoroch, časovej platnosti a ďalšími prípadnými doplnkovými informáciami, ktorých hodnoty však bude musieť zadefinovať samotný používateľ. Na tomto mieste ešte treba pripomenúť, že napr. jazyk OWL, ktorý slúži na reprezentáciu ontológií, má podporu pre uchovávanie mapovania tried.

Proces identifikácie zmien môže prebiehať vo forme, že všetky elementy verzie V_1 (staršej verzie) prehlásime za odobraté a všetky elementy verzie V_2 (novšej verzie) za pridané. Následne sa snažíme nájsť mapovania medzi elementmi verzií, či už automatizovane alebo ručne. Po procese identifikácie týchto mapovaní dospejeme do stavu, kedy v starej verzii budeme mať elementy odobraté a mapované a v novej verzii elementy mapované a pridané.

V nasledujúcich častiach opíšeme techniky a základné algoritmy, ktoré sme navrhli na identifikáciu zmien medzi dvoma verziami ontológie ako na sémantickej, tak aj na štrukturálnej úrovni. Pri tomto návrhu sme sa zamerali najmä na heuristické metódy.

3 Zmeny na sémantickej úrovni

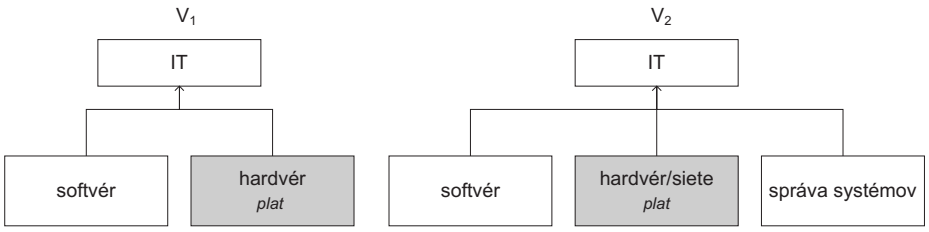
Úlohou kontroly na sémantickej úrovni je porovnať dve verzie ontológie, identifikovať jej rovnaké časti, napr. rovnaké triedy, rovnaké vlastnosti tried a nájsť rozdiely, resp. zmeny. Ako sme už uviedli v predchádzajúcej časti, tieto zmeny musia byť podrobené ďalšej analýze, pri ktorej sa musia nájsť prípadné vzťahy medzi oboma verziami, t.j. či ide o pridanie, odobratie, zmenu nejakej časti. Podobnosť medzi dvoma verziami sa dá vyjadriť ekvivalenciou elementov medzi nimi tak, že sa vytvorí mapovanie medzi elementmi dvoch verzií ontológie a určí sa vzťah medzi týmito elementmi. V prípade pridaného alebo odstráneného elementu sa vytvorí opis tejto transformácie.

Za účelom identifikácie zmien sme navrhli 6 heuristík podľa [4]. Ako vyplýva zo samotnej povahy heuristík, ich aplikáciou nedosiahneme vždy úspech, t.j. identifikácia môže byť chybná. Môže sa napr. stať, že heuristika označí nejaké dve triedy ako ekvivalentné, ale ich vnútorná štruktúra nie je rovnaká. To by nebol ani tak veľký problém, pretože sa musia identifikovať aj zmeny vnútri týchto tried a tak zistíme tieto vnútorné rozdiely. Aj napriek tomu môže byť celková sémantika tried rozdielna. Ak sa už stane takýto prípad, bude musieť samotný používateľ rozhodnúť o tom, či dané triedy sú alebo nie sú ekvivalentné.

V nasledujúcich opisoch heuristík používame pre elementy názov E, pre verzie názov V a pre triedy názov C.

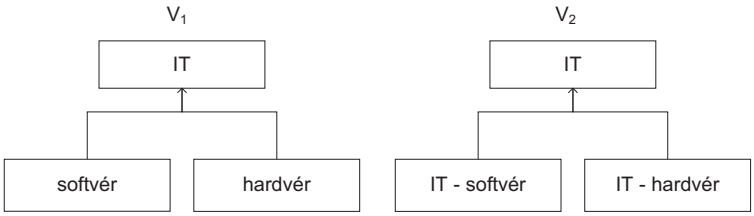
1. *Elementy s rovnakým názvom* – dva elementy $E_1 \in V_1$ a $E_2 \in V_2$ sú ekvivalentné, ak majú rovnaké meno a typ. Ide o štandardnú situáciu, kedy hľadáme základné mapovanie medzi dvoma verziami. Príklad takéhoto mapovania bol uvedený na obr. 1 medzi triedami hardvér.
2. *Jedna neekvivalentná podtrieda* – ak dve triedy $C_1 \in V_1$ a $C_2 \in V_2$, ktoré sú ekvivalentné, majú práve jednu podtriedu, ktorá nemá ekvivalent, potom tieto dve podtriedy označíme ako ekvivalentné. Táto situácia nastáva napr. v prípadoch, kedy sa zmení názov podtriedy.
3. *Viacero neekvivalentných podtried* – nech $C_1 \in V_1$ a $C_2 \in V_2$, C_1 a C_2 sú ekvivalentné, $subC_1$ je podtriedou C_1 a $subC_2$ je podtriedou C_2, $subC_1$ má všetky sloty ekvivalentné so $subC_2$, pričom tieto podtriedy majú sloty iné od ostatných podtried ich rodičovskej triedy, potom $subC_1$ a $subC_2$ sú ekvivalentné.

Príklad tejto situácie znázorňuje obr. 2, kde sme vo verzii V_2 pridali triede *IT* novú podtriedu *správa systémov* a zároveň sme zmenili názov podtriedy z *hardvér* vo verzii V_1 na *hardvér/siete* vo verzii V_2. V takomto prípade má trieda *IT* dve neekvivalentné podtriedy. Keďže triedy *hardvér* vo verzii V_1 a *hardvér/siete* vo verzii V_2 majú sloty s rovnakými názvami, triedy majú vlastne len iné meno a teda sú ekvivalentné. Z tejto úvahy vyplýva, že trieda *správa systémov* je novou triedou a teda pridaným elementom.



Obr. 2. Príklad viacerých neekvivalentných elementov.

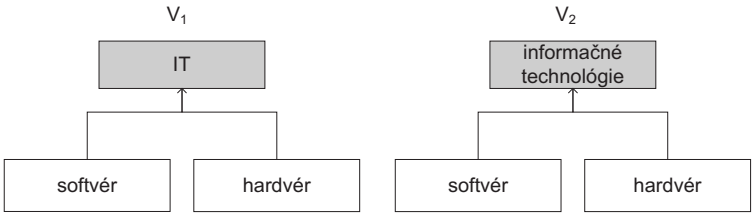
4. *Zmena názvov s rovnakým prefixom alebo sufixom* – ak $C_1 \in V_1$ a $C_2 \in V_2$ sú ekvivalentné a všetky podtriedy C_1 majú rovnaké názvy s podtriedami C_2 okrem konštantného prefixu alebo sufixu, potom podtriedy C_1 sú ekvivalentné podtriedam C_2. Na obr. 3 je znázornený prípad použitia uvedenej heuristiky, kde je obom podtriedam triedy *IT* vo verzii V_2 v porovnaní s verziou V_1 pridaný prefix *'IT -'*.



Obr. 3. Príklad zmeny názvu pridaním rovnakého prefixu.

5. *Neekvivalentná rodičovská trieda* – ak $C_1 \in V_1$ a $C_2 \in V_2$ majú ekvivalentné všetky podtriedy, potom C_1 a C_2 sú ekvivalentné. Na obr. 4 je znázornený príklad, v ktorom sa názov rodičovskej triedy zmenil zo skratky (*IT*) na celý názov (*informačné technológie*). Všetky jej podtriedy však zostali nezmenené a je možné medzi nimi nájsť ekvivalenciu, takže aj ich rodičovské triedy možno prehlásit' za ekvivalentné.
6. *Jeden nezhodný slot* – ak $C_1 \in V_1$ a $C_2 \in V_2$ sú ekvivalentné a obe triedy majú práve jeden neekvivalentný slot, tieto sloty sú ekvivalentné. Takáto situácia nastáva v prípadoch, kedy sa zmení názov slotu v triede.

Podľa [4] sa uplatňuje najjednoduchšia heuristika "elementy s rovnakým názvom" v 97,9 % prípadov. Na základe tohto čísla sa dá usúdit', že ontológie sa



Obr. 4. Príklad neekvivalentnej rodičovskej triedy.

vjednotlivých po sebe nasledujúcich verziách veľmi nemenia. Je to aj pozitívny fakt z pohľadu efektívnosti identidikácie zmien, pretože sa bude pri vykonávaní ostatných uvedených heuristík prehľadávať omnoho menšia časť verzie ontológie. Neznamená to však, že do ostatných heuristík bude vstupovať len zvyšná 2,1 % časť ontológie. Dôvodom je fakt, že v niektorých heuristikách (napr. pri neekvivalentnej rodičovskej triede) treba prehľadávať už mapované elementy, napr. za účelom zistenia všetkých mapovaných podtried. Dá sa však domnievať, že sa aj napriek tomuto faktu ostatné heuristiky radikálne zrýchlia po vykonaní vyhľadania elementov s rovnakým názvom, pretože budú stále pracovať s relatívne malou množinou elementov.

Po identifikácii zmien na sémantickej úrovni sa dá pre potreby identifikácie zmien na úrovni štrukturálnej určiť aj úroveň identifikovaných zmien. V prípade, že je element pridaný alebo odobratý, vieme presne určiť, čo na štrukturálnej úrovni má byť a čo nie. Ak napr. pridáme slot do triedy, vieme, že v novej verzii ontológie môžu mať indivíduá tejto triedy definované hodnoty nového slotu. Aby sme však vedeli ľahšie a efektívnejšie zmeny identifikovať, môžeme si určiť aj úroveň mapovania elementov. Táto úroveň predstavuje atribút operácie mapovania a určuje, či má daný element nejaké zmeny vnútornej štruktúry alebo nie. Rozpoznávať sa tak dajú tri úrovne mapovania:

- *bez zmeny* – štruktúra elementu aj jeho meno je rovnaké v oboch verziách,
- *izomorfná zmena* – štruktúra elementu je v oboch verziách rovnaká, ale meno je v oboch verziách rôzne,
- *sémantická zmena* – je zmenená vnútorná štruktúra elementu.

4 Zmeny na štrukturálnej úrovni

Úlohou kontroly na štrukturálnej úrovni je porovnať dve verzie ontológie, identifikovať jej rovnaké časti, napr. rovnaké indivíduá, rovnaké hodnoty slotov v indivíduách a nájsť rozdiely, resp. zmeny medzi nimi. Analogicky ako na sémantickej úrovni, treba tieto zmeny podrobiť ďalšej analýze, pri ktorej sa musia nájsť prípadné vzťahy medzi oboma verziami, t.j. či ide o pridanie, odobratie, zmenu nejakej časti. Identifikácia zmien na štrukturálnej úrovni sa odráža od identifikovaných zmien na sémantickej úrovni. Základným elementom ontológie, ktorý vstupuje na tejto úrovni do procesu identifikácie zmien, je indivíduum triedy. Keďže vnútorná štruktúra indivídua vychádza z vnútornej štruktúry triedy,

ktorej inštanciou je, musí táto identifikácia zmien odrážať identifikované zmeny na úrovni sémantickej. Takto sa dá dosiahnuť omnoho vyššia úspešnosť a aj efektivita identifikácie, pretože pri porovnávaní dvoch verzií ontológie vieme, ktoré triedy sú nové, ktoré sú odobraté a aj to, ktoré sú ekvivalentné.

Identifikácia zmien na štrukturálnej úrovni využíva niektoré heuristiky použité na sémantickej úrovni, aj keď mierne modifikované. Tieto modifikácie sa týkajú práve už spomínaných zmien na sémantickej úrovni. Keďže indivíduá definujú hodnoty slotov tried, môžu tieto hodnoty obsahovať ako literál, tak aj URI na nejaký zdroj. Tu sa vynára jedna zo základných otázok pri porovnávaní takýchto hodnôt a vyhodnocovaní týchto porovnávaní. Je známe, že v prípade aj najmenšej chyby (preklepu) v zápise URI, sa zdroj stáva úplne nedostupný, prípadne je to odkaz na iný zdroj, ktorý nebol vôbec zamýšľaný. V prípade literálov a najmä textových literálov je situácia opačná. V takomto prípade je jednoduchý preklep pre používateľa takmer transparentný. Všimnúť si takúto chybu síce môže, ale v drvivej väčšine danému textu pochopí a porozumie. Z uvedeného dôvodu je dobré rozlišovať typ a význam porovnávaných údajov.

Vytvoriť mechanizmus na jednoduché porovnanie dvoch literálov je triviálna záležitosť a takéto prostriedky obsahuje prakticky každý dnes používaný programovací jazyk. Pri takomto porovnaní jednoducho vezmeme dva reťazce a zistíme, či sú rovnaké. Výsledkom je informácia, či sú reťazce *úplne zhodné*.

Iným mechanizmom môže byť spôsob porovnania, pri ktorom berieme do úvahy určitú prahovú hodnotu množstva rozdielov, ktoré môžu dva reťazce obsahovať. Pokiaľ množstvo rozdielov nestúpne nad túto definovanú hodnotu, môžeme oba reťazce prehlásiť za *relatívne zhodné*. Je vhodné, aby prahovú hodnotu mohol určiť používateľ. Taktiež je vhodné pre možnosť vyhodnotenia porovnania vyznačiť, na základe čoho systém usúdil, že dva reťazce sú alebo nie sú relatívne zhodné. Otázkou naďalej zostáva, aká má byť granularita rozdielov, na základe ktorých sa bude rozhodovať o relatívnej zhodnosti. Sú to znaky, slová, odseky? Je značný rozdiel, keď porovnávame slová s prahovou hodnotou dva znaky a celé odseky s rovnakou prahovou hodnotou. Jednoduchým riešením by mohlo byť určenie relatívneho počtu znakov v percentách.

Aj na základe uvedených úvah môžeme dospieť k záveru, že úlohou identifikácie zmien na štrukturálnej úrovni verzie ontológie je nájsť indivíduá, ktoré boli pridané, odstránené a zmenené. Pritom treba zohľadniť aj zmeny sémantickej úrovne ako sú napr. zmeny názvov slotov, tried, pridanie slotu. Jednou z najdôležitejších úloh je identifikácia správneho mapovania ekvivalentných indivíduí medzi dvoma verziami. Pokiaľ dosiahneme takéto mapovanie, vieme určiť, ktoré sloty boli do akej miery zmenené, ktoré údaje sú aktuálne a vieme taktiež túto aktuálnosť aj overiť.

V predchádzajúcej časti sme uviedli, že identifikované zmeny možno definovať troma operáciami - pridanie, mapovanie a odobratie. Proces identifikácie prebieha podobne ako na sémantickej úrovni, t.j. všetky indivíduá verzie V_1 (staršej verzie) prehlásime za odobraté a všetky indivíduá verzie V_2 (novšej verzie) za pridané. Následne sa snažíme nájsť mapovania medzi indivíduami verzií, či už automatizovane alebo ručne. Po procese identifikácie týchto mapovaní dospe-

jeme do stavu, kedy v starej verzii budeme mať indivíduá odobraté a mapované a v novej verzii indivíduá mapované a pridané. Pri tomto procese je pre lepšiu efektivitu užitočné prehľadávať len indivíduá tých tried, pre ktoré existuje mapovanie medzi verziami.

Pre identifikáciu mapovania indivíduí medzi dvoma ontológiami sme navrhli tri heuristiky. V nasledujúcich opisoch heuristík používame pre indivíduum označenie I, pre verzie označenie V a pre triedy označenie C.

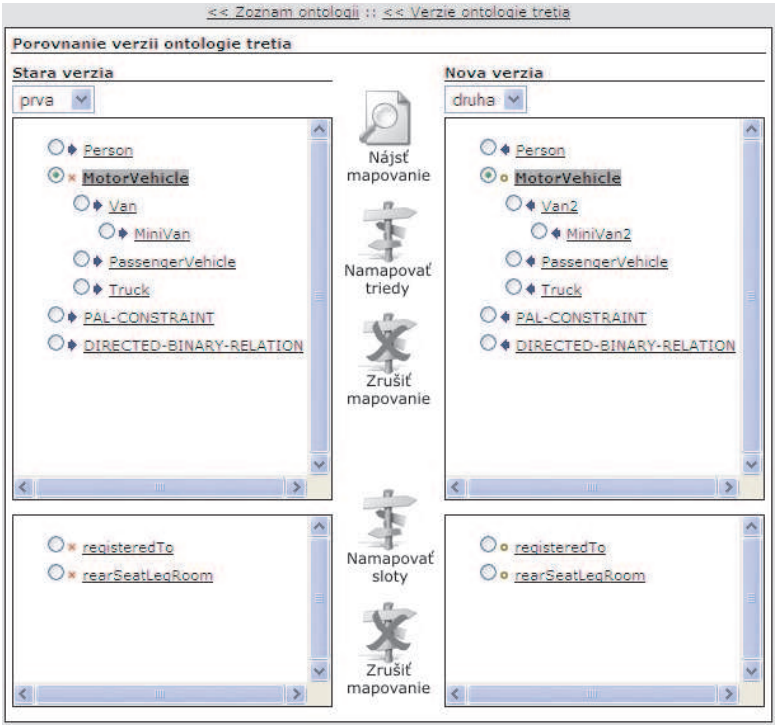
1. *Indivíduá s rovnakým názvom* – nech dve indivíduá I_1 je triedy C_1 a $C_1 \in V_1$ a I_2 je triedy C_2 a $C_2 \in V_2$ a C_1 a C_2 sú ekvivalentné. Potom I_1 a I_2 sú ekvivalentné, ak majú zhodné meno. Ide o štandardnú situáciu, kedy hľadáme mapovanie indivíduí medzi dvoma verziami, ale táto heuristika sa nedá aplikovať v prípade anonymných indivíduí, pretože tie neobsahujú meno. Na takéto indivíduá musí byť použitá iná heuristika.
2. *Zhodné hodnoty slotov* – nech dve indivíduá I_1 je triedy C_1 a $C_1 \in V_1$ a I_2 je triedy C_2 a $C_2 \in V_2$ a C_1 a C_2 sú ekvivalentné. Potom I_1 a I_2 sú ekvivalentné, ak majú relatívne zhodné hodnoty slotov, ktoré majú mapovanie medzi triedami C_1 a C_2. Táto heuristika je vhodná na identifikovanie ekvivalentných anonymných indivíduí. Môže sa však stať, že triedy obsahujú príliš málo ekvivalentných slotov na to, aby sa dali indivíduá pomocou tejto heuristiky s dostatočnou pravdepodobnosťou určiť. Tento problém sa dá čiastočne riešiť tým, že sa určí minimálny počet slotov, ktoré triedy C_1 a C_2 musia mať namapované alebo minimálny počet definovaných hodnôt v indivíduách I_1 a I_2, aby sa táto heuristika aplikovala.
3. *Jedna nezhodná hodnota slotu* – nech dve indivíduá I_1 je triedy C_1 a $C_1 \in V_1$ a I_2 je triedy C_2 a $C_2 \in V_2$ a C_1 a C_2 sú ekvivalentné. Potom I_1 a I_2 sú ekvivalentné, ak majú absolútne zhodné hodnoty slotov, ktoré majú mapovanie medzi triedami C_1 a C_2 okrem jedného. Táto heuristika je vhodná na identifikovanie ekvivalentných anonymných indivíduí, ktoré majú zmenenú hodnotu práve jedného slotu. Podobne ako pri predchádzajúcej heuristike je vhodné definovať minimálny počet slotov, ktoré sa budú analyzovať.

Uvedené heuristiky slúžia na identifikáciu ekvivalencie indivíduí. Všetky mapované indivíduá môžu byť podrobené ďalšej analýze, kedy sa identifikujú zmeny na úrovní slotov, resp. ich hodnôt. Opäť sa analyzujú len sloty, ktoré majú identifikované mapovanie na sémantickej úrovni, lebo len pre takéto sloty vieme pri indivíduách získať hodnoty, ktoré môžeme porovnávať. Toto porovnávanie už neslúži na to, aby sme zistili rozdiely medzi ontológiami pre účely systému, ale pre účely používateľa, t.j. aby samotný používateľ vedel a prípadne aj videl, ktoré konkrétne údaje sú zmenené a čo je na nich zmenené. Pre toto porovnávanie sa aplikujú už spomínané techniky na zistenie absolútnej a relatívnej zhodnosti.

5 Overenie návrhu

Za účelom overenia realizovateľnosti navrhnutých heuristík sme vytvorili prototyp, ktorého cieľom bolo implementovať kritické časti systému, overiť ich

funkčnosť a úspešnosť na niekoľkých vzorkách ontológií. Prototyp (pozri obr. 5) umožňuje ukladať verzie ontológií a identifikované zmeny v úložisku, obsahuje modul na správu verzií, modul na správu zmien medzi verziami, umožňuje automatizovane a manuálne identifikovať zmeny na sémantickej úrovni a identifikované zmeny dokáže vizuálne vyznačiť. Prototyp sme implementovali ako webovú aplikáciu na platforme J2EE s využitím rámcov Struts [6] a iBatis [7]. Na prácu s ontológiami sme použili rámec Jena [5].



Obr. 5. Ukážka používateľského rozhrania prototypu.

Pri testovaní prototypu sme použili tri ontológie, pričom každá z nich obsahovala 10 až 15 verzií s rôznymi typmi zmien a v rôznom rozsahu. Vo všeobecnosti sa dajú potvrdiť očakávané výsledky, pretože systém s mimoriadne veľkou spoľahlivosťou identifikoval menšie a stredne veľké zmeny ontológie, kedy nenastali nejaké zásadné presuny a súčasne aj premenovania celých častí či vetiev v štruktúre tried ontológie. V prípadoch, kedy bola systému poskytnutá verzia s množstvom koncepčných zmien, úspešnosť identifikácie mapovaní medzi triedami sa znížila a systém bol prakticky schopný identifikovať len triedy a sloty s rovnakými názvami. V takýchto prípadoch však identifikácia zmien môže napomôcť vďaka možnosti ručného mapovania. Systému totižto stačí, aby našiel "záchytné body", t.j. aby boli splnené podmienky pre tú ktorú heuristiku. Po splnení takýchto podmienok bol systém opäť schopný identifikovať ďalšie zmeny. Toto bolo testované na verziách dvoch rôznych ontológií.

6 Záver

Predložený príspevok sa zaoberá riešením problémov kontroly aktuálnosti prezentovaných údajov v kontexte webu so sémantikou. Základ riešenia spočíva vo využívaní heuristických metód na vyhľadávanie ekvivalentných prvkov verzií ontológií. Niektoré z metód, ktoré sa používajú na sémantickej úrovni boli prevzaté, aby bolo možné identifikovať zmeny na úrovni štrukturálnej. Hlavným prínosom je návrh metód identifikácie rozdielov na štrukturálnej úrovni spolu s identifikáciou ekvivalencie anonymných indivíduí. Všetky takto identifikované zmeny v konečnom dôsledku možno použiť na kontrolu platnosti samotných hodnôt slotov indivíduí, t.j. obsahu, ktorý sa v konečnom dôsledku dostane k používateľovi. Pri poznaní mapovania medzi verziami ontológie môžu napr. aj agenty, ktoré sú navrhnuté pre jednu verziu ontológie, pracovať či spolupracovať s inou verziou ontológie, pre ktorú pôvodne navrhnutí neboli.

Prototyp, ktorý sme vytvorili, poskytuje dobrý základ pre ďalšie rozšírenie na štrukturálnej úrovni ontológií. Je možné ho použiť aj pri vybudovaní riešenia na distribúciu rozdielov medzi dvoma ontológiami alebo na online identifikáciu, ktorá môže byť jednoducho a automatizovane použitá inými systémami. To všetko na báze napr. dnes veľmi populárnej technológie webových služieb.

Azda najzaujímavejším rozšírením systému je rozšírenie systému tak, aby bol viac orientovaný na konečného konzumenta obsahu. Takýto systém si uchováva v definovaných časoch verzie ontológie, či už ručným zadaním súboru alebo automatickým získaním z internetu. Následne analyzuje sémantické a štrukturálne zmeny. Na základe týchto zmien vie efektívnym spôsobom podať používateľovi zmeny v sledovaných informáciách z istej domény a tieto zmeny aj vyznačiť. V spojení s adaptívnymi hypermediálnymi systémami môžu byť takéto informácie "servírované" používateľovi presne podľa jeho záujmu a potrieb, čo môže byť mimoriadne zaujímavé v mnohých aplikačných oblastiach.

Referencie

1. D. Fatterly, M. Manase, M. Najork, J. Wiener, A Large-Scale Study of the Evolution of Web Pages. In Software—Practice and Experience 34(2), 2004, 213–237
2. M. Klein, D. Fensel, Dieter, A. Kiryakov, D. Ognyanov, Ontology Versioning and Change Detection on the Web. In 13th Int. Conf. on Knowledge Engineering and Knowledge Management (EKAW02), LNCS 2473, Sigüenza, Spain, 2002, page 197
3. OCLC - Online Computer Library Center. Web Characterization. [online]. [cit. 2005-06-05]. Dostupné na webe: `<http://wcp.oclc.org>`
4. N.F. Noy, M. A. Musen, Ontology Versioning as an Element of an Ontology-Management Framework. In IEEE Intelligent Systems, 2003
5. Jena - A Semantic Web Framework For Java [online]. [cit. 2005-06-05]. Dostupné na webe: `<http://jena.sourceforge.net>`
6. The Apache Software Foundation. Struts [online]. [cit. 2005-06-05] Dostupné na webe: `<http://struts.apache.org>`
7. iBatis Data Mapper Framework. [online]. [cit. 2005-06-05]. Dostupné na webe: `<http://www.ibatis.com>`

PÔVODNÉ VEDECKÉ PRÁCE II

Evoluce replikačních algoritmů v stohově orientovaných systémech*

Jiří Dokulil, Jakub Yaghob, Filip Zavoral

Katedra softwarového inženýrství MFF UK
Dokulil@gmail.com, Jakub.Yaghob@mff.cuni.cz, Filip.Zavoral@mff.cuni.cz

Abstrakt Mechanismus datového stohu byl publikován včetně popisu základních replikačních algoritmů. Při implementaci do reálného prostředí se ukázalo, že navržené postupy nedostačují, a to jak z hlediska vyjadřovací síly tak výkonu. Systém implementovaný přesně podle teoretického návrhu by nesplňoval zcela uživatelské požadavky na logiku replikací a nebyl by výkonnostně vyhovující. Proto jsme postupně navrhovali, implementovali a na reálných datech (co do velikosti i nečistoty) testovali nové replikační postupy a algoritmy replikací a detekcí identit, které v běžném provozu vykazují v mnoha směrech vylepšené vlastnosti oproti původně navrženým. Systém s předkládanými algoritmy běží v pilotním provozu na reálných datech.

1 Úvod

Při implementaci stohového systému do reálného prostředí se ukázalo, že teoreticky navržené postupy [1] nedostačují, a to jak z hlediska vyjadřovací síly tak výkonu. Proto jsme postupně navrhovali, implementovali a na reálných datech (co do velikosti i nečistoty) testovali nové unifikační algoritmy, které v běžném provozu vykazují v mnoha směrech vylepšené vlastnosti oproti původně navrženým.

V kap. 2 jsou stručně zrekapitolovány základní vlastnosti stohových systémů, v kap. 3 je popsán teoretický návrh unifikačních algoritmů, kap. 4 obsahuje popis skutečně implementovaných algoritmů, v kap. 5 jsou uvedena další rozšíření a změny unifikačních algoritmů, kap. 6 obsahuje jejich vyhodnocení.

2 Stohový systém a jeho replikace

2.1 Vertikalizace dat

Základní ideou stohových systémů je vertikalizace dat, tj. nevyužívá se tradiční horizontální pojetí databázové tabulky, kdy jedna řádka představuje nějakou množinu spolu souvisejících atributů nějaké entity [2], [3], [4]. Místo toho je každý

* The work was partly supported by the project 1ET100300419 of the Program Information Society of the Thematic Program II of the National Research Program of the Czech Republic

atribut 'tradiční' řádky představován jedním řádkem datového stohu a odpovídající si atributy jsou pak spojeny identifikací entity, které tyto atributy náleží. Celé datové schéma všech zúčastněných aplikací je nahrazeno dvěma tabulkami - hodnoty atributů a strukturou entit.

2.2 Unifikace

Unifikace entit nastává v okamžiku, kdy je při exportu dat z provozního systému do stohu detekována nová entita. Úkolem unifikace je buď najít již existující entitu ve stohu nebo vytvořit entitu novou.

Unifikace entit je založena na klasifikaci relevantnosti atributů entit do tří kategorií:

1. Určující – se značnou pravděpodobností shoda těchto atributů znamená stejnou entitu.
2. Relevantní – shoda těchto atributů zvyšuje pravděpodobnost shody entit.
3. Ostatní – nemají vliv na shodu entit.

Bohužel unifikace na základě atributů entity není vždy dostatečná. V některých případech (např. entita Osoba má závislou entitu Identifikace, kde jsou uvedeny např. RČ, ČOP, apod., tedy vše určující ,,atributy“) je nutné jako určující nebo relevantní vzít v potaz i obsah závislých entit. Nelze však uvažovat celé obsahy závislých entit, ale pouze atributy dle jejich klasifikace relevantnosti. Výslednou klasifikaci atributů v závislých entitách lze určit podle tab. 1.

	Určující	**Relevantní**	**Ostatní**
Určující	Určující	Relevantní	Ostatní
Relevantní	Relevantní	Ostatní	Ignoruj
Ostatní	Ostatní	Ignoruj	Ignoruj

Tab. 1. Klasifikace atributů.

Sloupce tabulky znamenají klasifikaci atributu v závislé entitě, řádky znamenají klasifikaci závislé entity jako celku.

V současné době unifikační algoritmy považují klasifikace ,,ostatní“ a ,,ignoruj“ za shodné, je však pravděpodobné, že v budoucnosti budou mít odlišnou sémantiku.

Entity se dají dle vlivu na unifikaci rozdělit na dvě kategorie:

1. Kořenová – hlavní entita, u které je důležité rozpoznat stejné entity.
2. Vedlejší – pomocné entity, které zaznamenávají vedlejší údaje a vztahy.

Pro každou kořenovou entitu se pak následujícím algoritmem vytvoří množiny určujících a relevantních atributů:

1. Nastav množiny relevantních a určujících atributů na prázdno, nastav si aktuální klasifikátor na ,,Určující“, aktuální typ entity na typ kořenové entity.
2. Vyber pro aktuální typ entity atributy s klasifikací ,,Určující“ a ,,Relevantní“, podle tabulky a aktuálního klasifikátoru zjisti skutečnou klasifikaci těchto atributů. Pokud nějaká taková klasifikace je ,,Určující“ nebo ,,Relevantní“, zařaď je do příslušné množiny.
3. Pokud je nějaký z takto přidávaných atributů odkaz na jinou entitu, pak jdi rekurzivně do bodu 2, nastav aktuální typ entity na odkazovaný typ a nastav aktuální klasifikátor podle klasifikátoru daného atributu.
4. Pro všechny typy entit, které referencují aktuální typ entity vypočti klasifikátor podle aktuálního klasifikátoru a klasifikátoru referencujícího atributu. Pro každý typ entity, který má takto vypočtenou klasifikaci ,,Určující“ nebo ,,Relevantní“ jdi rekurzivně do bodu 2, nastav aktuální typ entity na referencující ty entity a nastav aktuální klasifikátor podle vypočtené klasifikace.
5. Návrat o jeden stupeň rekurze, pokud je to nejvyšší úroveň rekurze, ukonči algoritmus.

Tyto množiny atributů je možné si předpočítat pro zvýšení rychlosti před zahájením každého exportu.

2.3 Stohové cache

Cache slouží především k snadnějšímu přístupu k datům ve stohu, protože stohový formát dat není příliš vhodný pro přímé databázové dotazy. Tato úroveň cachí znamená převod aktuálních platných verzí dat do horizontální ‘relační‘ podoby. Nad těmito cachemi lze vytvořit další úrovně cachí pro zajištění přístupových práv pro jednotlivé aplikace pracující nad centrální databází, to však není předmětem tohoto článku.

Jednomu typu entity odpovídá v cachi jedna tabulka, jedné entitě odpovídá jedna řádka v této tabulce. Atributu entity pak odpovídá jeden sloupec. Aktualizací cachí a udržování koherence se stohem provádí aplikační server během replikací.

2.4 Roviny evoluce unifikačních algoritmů

Unifikační algoritmy je třeba posuzovat ve dvou rovinách – výkonnosti a kvalitě. Zatímco měřítko výkonnosti je zřejmé (počet unifikovaných entit za jednotku času), měřítko kvality může být v různých systémech vztaženo na různé aspekty. Ve stohových systémech je jedním z nejdůležitějších kritérií minimalizace unifikačních chyb.

Při unifikaci může dojít k dvěma druhům chyb, viz tab. 2, a to unifikace entit, které neměly být unifikovány (NCh) a neprovedení unifikace entit, které měly být unifikovány (PCh).

Všechny unifikační algoritmy byly navrženy defenzivně tak, aby v maximální možné míře eliminovaly unifikaci entit, které v reálném světě nejsou stejnými entitami. Kvalita unifikačního algoritmu je pak měřena jako

Entity	Unifikovány	Neunifikovány
Měly se unifikovat	OKuni	Přípustná chyba (PCh)
Neměly se unifikovat	Nežádoucí chyba (NCh)	OKneuni

Tab. 2. Kvalifikace unifikací.

$$Qu = 1 - PCh/(OKuni + OKneuni)$$

Cílem kvalitativního vylepšování unifikačních algoritmů je tedy minimalizace množiny PCh, resp. přesun entit z množiny PCh do množiny OKuni.

3 Původní teoretický návrh

Unifikace entity podle původního teoretického návrhu vypadá následovně:

1. Nalezení shodných hodnot určujících atributů ve stohu a v unifikované entitě.
2. Nalezení kořenové entity dle shodných určujících atributů.
3. Porovnání skutečných určujících atributů u entit – kandidátů z bodu 2 s hodnotami unifikované entity.
4. Porovnání hodnot množiny relevantních atributů s hodnotami unifikované entity.

Po porovnání těchto množin atributů může nastat několik případů:

1. Shodují se určující i relevantní atributy – entita unifikována.
2. Shodují se určující atributy, ale některý z relevantních se neshoduje (např. souhlasí RČ, ale nesouhlasí příjmení) – je vytvořena nová entita, přičemž do unifikačního logu se zaznamená informace o možném kandidátu unifikace.
3. Shodují se všechny relevantní a některé určující (např. překlep v RČ) – je vytvořena nová entita, přičemž do unifikačního logu se zaznamená informace o možném kandidátu unifikace.
4. Ostatní případy znamenají různé entity.

Pokud byla nalezena odpovídající kořenová entita, je ještě třeba unifikovat odpovídající vedlejší entity (např. adresu osoby).

4 Implementovaný algoritmus

V dalším textu jsou pod pojmem atributy entit myšleny jen atributy, které nejsou typu Id. Atributy typu Id jsou označovány jako vztahy. Navíc vztahy jsou interpretovány jako obousměrné, tedy pokud entita A má atribut, jehož hodnota je id entity B, pak tento vztah náleží entitě B zcela stejně jako entitě A.

4.1 Pseudokód algoritmu

```
match(entita E)
{
  Dotaz Q;
  TE=typ entity E;
  označ TE jako zpracovaný;
  do Q přidej test na určující atributy E
  match_recursion(E,Q);

  spusť Q nad databází;
  R=kolekce řádků s možnými kombinacemi unifikací
  if (žádná kombinace v R)
  {
    // všechny testované entity nejsou unifikovány
    jsou vytvořeny nové entity;
    odznačit typy entit označené jako použité;
  }
  if (jediná kombinace v R)
  {
    ROW=ten jediný řádek z R;
    pro každou testovanou entit
      nastala unifikace na odpovídající entitu z ROW;
    odznačit typy entit označené jako použité;
  }
  else  // existuje více možných kombinací
  {
    // není možné určit, která z nich je správná
    všechny testované entity jsou nové entity;
    odznačit typy entit označené jako použité;
  }
}

match_recursion(entita E, dotaz Q)
{
  TE=typ entity E;
  for each V in {vztahy v nichž vystupuje TE
      a jsou klasifikovány jako určující} do
  {
    if (druhý konec V už je označen jako zpracovaný)
      continue;
    IV=instance vztahu V v nichž vystupuje E;
    for each P in {druhé konce IV než je E} do
    {
      TP=typ entity P;
```

```
      do Q přidej test vztahu E a P;
      if (kategorie typu entity je kořenová)
      {
        match(P);
        do Q ulož výsledek match(P)
          jako jediného kandidáta pro P;
      } else {
        do Q přidej test na určující atributy P;
        match_recursion(P,Q);
      }
    }
  }
}
```

Jakmile je entita jednou zpracována unifikačním algoritmem, není zpracovávána znovu. Protože k unifikaci entity může dojít i kvůli jejímu vztahu s jinou entitou, není možné z entity samotné určit, kdy se bude unifikovat.

Algoritmus vychází z jedné entity, ale zpracovává zároveň nejen tuto entitu, ale i entity, které jsou na ni vázány. To, jestli je entita vázána na jinou, závisí na více okolnostech. Pokud algoritmus nezačal výpočet na kořenové entitě, pak veškeré vztahy mezi entitami jsou považovány za určující. Pokud výpočet začal na kořenové entitě, algoritmus se řídí klasifikací atributů vztahů mezi entitami.

Algoritmus postupně prochází všechny typy určujících vztahů, tedy všechny atributy pro typ testované entity a všechny atributy ostatních typů entit, které referencují typ testované entity. Pokud algoritmus během svého chodu doposud nezpracoval typ entit, který je referencován právě zpracovávanou entitou nebo tuto referencuje, pak všechny tyto entity jsou označeny jako vázané na původní unifikovanou entitu. Pro každou z těchto entit je rekurzivně spuštěn algoritmus.

Speciálním případem je situace, kdy referencovanou nebo referencující entitou je kořenová entita. Ta je zpracována samostatným spuštěním celého unifikačního algoritmu s tím, že sdílí množinu již zpracovaných typů entit.

4.2 Omezení algoritmu

Algoritmus neumožňuje cykly. Přesněji určeno, algoritmus při vyvolání rekurze (ať již pomocné rekurze testující potomky, nebo hlavní rekurze spuštěné na potomky, které jsou kořenovými entitami) na typ entity označí tento typ jako již zpracovaný. Pokud narazí na potomka, jehož typ je již označený jako zpracovaný, chová se, jako kdyby tento potomek neexistoval. Toto označení je zrušeno na konci unifikačního algoritmu.

Nemožnost cyklů výrazně posiluje vliv pořadí zpracovávání typů entit na výsledek unifikace.

Toto omezení vzniklo kvůli efektivitě unifikací, protože bez něj by se algoritmus mohl pokusit unifikovat v jeden okamžik velké množství entit, jak původně předpokládal teoretický návrh. Vyhodnotit takový dotaz však není reálné.

Dalším omezením tohoto algoritmu je, že nepodporuje relevantní atributy. Pro toto omezení existují dva důvody. Implementace teoretického návrhu je příliš náročná a v aktuálně používaném schématu dat se žádné určující atributy s výjimkou osob nevyskytují. Toto omezení je řešeno dalšími úpravami.

4.3 Vyhodnocení výsledků unifikace

Výsledkem dotazu Q je kolekce přípustných unifikací. Pokud je tato možnost jediná, je situace jasná a unifikace aplikována. V případě, kdy není ani jedna možnost, jsou pro všechny testované entity vytvořeny nové entity.

V případě, kdy možností je více, není možné algoritmicky rozhodnout, která z nich je správná. Navíc určující atributy by měly být zvoleny tak, aby tato situace za normálního provozu nenastala. Pokud tato situace nastane, je zvolen defenzivní přístup a jsou vytvořeny nové entity. Správci dat je poskytnuta formou unifikačního logu informace o nejednoznačném výsledku spolu s možnými kandidáty na unifikaci.

5 Další úpravy a rozšíření

5.1 Použití cachí

V reálném provozu se ukázalo, že provádět unifikace přímo nad stohem je neefektivní, a to především kvůli velkému množství spojení (join), která skládají vertikalizovaná data do relačního formátu. Algoritmus byl upraven tak, aby pracoval nad cachemi. Nad těmito tabulkami je navíc možné vytvořit indexy, které algoritmus dále urychlují.

Unifikace je implementována jedním velmi netriviálním SQL dotazem, který konstruovanými poddotazy testuje existenci odpovídající skupiny v cache pro celou skupinu vázaných entit. Odpovídající skupina znamená, že hodnoty určujících atributů jsou identické a vztahy mezi vázanými entitami jsou zachovány.

5.2 Částečná identifikace osob

Největším problémem při unifikaci reálných dat se ukázaly být osoby a jejich identifikace. Zde se jak teoreticky navrhnutý tak prvotně implementovaný algoritmus ukázal jako neodpovídající očekávání uživatelů.

Jedním ze zdrojů problémů je požadavek na to, aby pro všechny právě unifikované entity algoritmus našel ve stohu jim odpovídající entity.

Pokud nastane situace, že osoba právě vkládaná do stohu v něm již existuje, ale tato nově vkládaná osoba má nějakou identifikaci navíc, pak se jistě neunifikuje, i kdyby všechny identifikace ve stohu měly stejné hodnoty jako odpovídající identifikace vkládané entity. Např. osoba ve stohu má evidováno pouze rodné číslo a nově unifikovaná tatáž osoba obsahuje jak rodné číslo (shodné s rodným číslem ve stohu) tak i číslo občanského průkazu.

Unifikační algoritmus byl upraven tak, že při unifikaci osob stačí, aby jediná identifikace vkládané osoby měla odpovídající identifikaci ve stohu. Tato verze vykazuje z pohledu uživatele lepší kvalitu než původní verze.

5.3 Druhy identifikací

Po několika měsících provozu se však i verze algoritmu uvedená v předchozí kapitole ukázala jako nedostatečně kvalitní. Požadavkem bylo, aby se vkládaná osoba A unifikovala s osobou B ze stohu pouze v situaci, kdy se shodují ve všech na nich závislých identifikacích, u kterých platí, že daný druh identifikace mají osoby A i B. Tedy pokud osoba A má uvedeno RČ, číslo OP a číslo pasu a osoba B má RČ, číslo pasu a identifikační číslo přidělené fakultou, pak se musí shodovat hodnoty RČ a čísla pasu.

Problémem je, že nalezení společných identifikací není možné provést předem, a pak unifikovat pouze ty společné, protože každá osoba ve stohu může mít u sebe různou sadu identifikací. Přesto je i v této verzi možné provést matchování celé osoby jediným dotazem nad cachemi za použití levého vnějšího spojení. Výsledný dotaz se podle předběžných testů ukazuje jako dostatečně efektivní.

6 Vyhodnocení

Vyhodnocování algoritmů probíhalo na mnoha vzorcích dat pocházejících z reálných provozních systémů. Databázový stroj byl použit Oracle 9.2, unifikační algoritmy byly implementovány v C++ v rámci vyvíjeného aplikačního serveru. Vzhledem k velké heterogenitě a míře znečištění dat (překlepy, absence relevantních údajů, neaktualizovaná data apod.) jsme vyhodnocování algoritmů prováděli nad různými datovými soubory, a to jak velikostí tak kvalitou. Celkově bylo unifikováno (zaokrouhleně) 600 tisíc entit a 11 miliónů atributů.

Algoritmus	**Doba (100 000 entit)**	**Qu**
Teoretický	N/A	N/A
První implementace	15 hod.	0.9880
Cache	49 min.	0.9880
Částečná identifikace osob	28 min.	0.9931
Druhy identifikací	31 min.	0.9987

Tab. 3. Vyhodnocení algoritmů.

Původní teoretický návrh při testovacím spuštění byl po několika dnech výpočtu ukončen pro nedostatek místa v temporálním segmentu databázového stroje (10 GB). První reálně fungující implementace dokončovala výpočet v řádech desítek hodin, přičemž kvalita unifikace byla pro další zpracování správcem dat příliš nízká. Aby bylo možno algoritmy optimalizovat na kvalitu v reálném čase, bylo nutno nejprve vyladit výkonnost unifikací. Výrazným zlepšením se ukázalo použití cachí, které zrychlily výpočty cca 15 až 20-násobně (v závislosti na struktuře dat). V další fázi jsme se zaměřili na vylepšování Qu. Další úpravy pak vylepšily Qu (tj.minimalizovaly PCh) tak, aby na reálných datech přicházejících z provozních systémů byl správce dat schopen efektivní kontroly.

Některé úpravy mohou zlepšit Qu na úkor rychlosti unifikace. Pokud však toto zpomalení není výrazné, byl kladen větší důraz na Qu. Příkladem takové úpravy je úprava pro druhy identifikací.

7 Závěr

Při nasazení stohového systému do reálného prostředí a následných úpravách unifikačních algoritmů jsme dosáhli výrazného zlepšení jak ve výkonnostní tak kvalitativní rovině unifikačních algoritmů. V aktuální verzi došlo oproti první implementaci o 25 až 30-násobnému zrychlení a cca 9-násobnému zlepšení kvality unifikací.

Systém je v současné podobě nasazen v pilotním provozu, vzhledem k heterogenitě postupně připojovovaných provozních systémů a jimi generovaných dat očekáváme potřebu dalších úprav a optimalizací algoritmů, které povedou k dalšímu zlepšení obou sledovaných rovin.

Reference

1. D. Bednarek, D. Obdrzalek, J. Yaghob, F. Zavoral: Data Integration Using DataPile Structure. In proc. of the 9th East-European Conference on Advances in Database and Information Systems, Tallinn, Estonia, 2005
2. M. Lenzerini: Data Integration: A Theoretical Perspective. In Proc. of the 21st ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2002), 2002, pages 233–246
3. E. Schallehn, K. Sattler, and G. Saake: Extensible and Similarity-Based Grouping for Data Integration. In 8th Int. Conf. on Data Engineering (ICDE), San Jose, CA, 2002
4. T.W. Yan and H. Garcia-Molina: Duplicate Removal in Information Dissemination. In Proceedings of VLDB-95, September 1995. Information Systems, Irvine, CA, 2002

Názov:	ITAT 2005 Information Technologies-Applications and Theory
Editori:	prof. RNDr. P. Vojtáš, DrSc. (Ed.)
Vydavateľ:	Prírodovedecká fakulta University Pavla Jozefa Šafárika v Košiciach
Rok vydania:	2005
Náklad:	200 ks
Rozsah:	412 strán
Tlač:	Vydavateľstvo UPJŠ v Košiciach

ISBN 80-7097-609-8