Preface

The 9th workshop **ITAT'09 – Information Technology – Applications and Theory** (http://www.itat.cz) was held in Horský hotel Kráľova studňa (http://www.kralova-studna.sk/), located 1300 meters above the sea level in Low Tatras, Slovakia, from 25 to 29 September 2008.

ITAT workshop is a place of meeting of scientists and experts working in computer science mainly from the Czech Republic and Slovakia. Official languages for oral presentations are Czech and Slovak; proceedings papers are in English).

The emphasis of the workshop is on exchange of information between participants. Workshop offers a possibility for students to make a first public presentation and to discuss with more experienced scientists. Therefore a big space is devoted to informal discussions. The place is traditionally chosen at least 1000 meter above the sea level in a location not directly accessible by public transport.

Thematically workshop ranges from foundations of computer science, security, through data and semantic web to software engineering. There were 51 submissions; these proceedings consists of

- 11 original scientific papers

All papers were refereed by at least two independent referees.

The workshop was organized by

- Institute of Informatics of University of P.J. Šafárik in Košice
- Faculty of Mathematics and Physics, Charles University in Prague
- Institute of Computer Science of Academy of Sciences of the Czech Republic, Prague

Partial support has to be acknowledged from projects of the Program Information Society of the Thematic Program II of the National Research Program of the Czech Republic 1ET100300517 "Methods for intelligent systems and their application in data mining and natural language processing" and Czech institutional project MSM-0021620838 "Modern methods, structures and systems of computer science".

Hereby we express sincere thanks to our sponsors:

Profinit (http://www.profinit.eu/) profinit. NEW FRONTIER GROUP

and CSKI (http://www.cski.cz/) ČSKI

Filip Zavoral, Peter Vojtáš

Program Committee

Filip Zavoral, (Chair), *Charles University, Prague, CZ*
Gabriela Andrejková, *University of P.J. Šafárik, Košice, SK*
Mária Bieliková, *Slovak University of Technology, Bratislava, SK*
Leo Galamboš, *Czech Technical University, Prague, CZ*
Ladislav Hluchý, *Slovak Academy of Sciences, Bratislava, SK*
Tomáš Horváth, *University of P.J. Šafárik, Košice, 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*
Stanislav Krajči, *University of P.J. Šafárik, Košice, SK*
Věra Kůrková, *Institute of Computer Science, AS CR, Prague, CZ*
Markéta Lopatková, *Charles University, Prague, CZ*
Ján Paralič, *Technical University, Košice, SK*
Dana Pardubská, *Comenius University, Bratislava, SK*
Martin Plátek, *Charles University, Prague, CZ*
Jaroslav Pokorný, *Charles University, Prague, CZ*
Karel Richta, *Charles University, Prague, CZ*
Gabriel Semanišin, *University of P.J. Šafárik, Košice, SK*
Václav Snášel, *Technical University VŠB, Ostrava, CZ*
Vojtěch Svátek, *University of Economics, Prague, CZ*
Jiří Šíma, *Institute of Computer Science, AS CR, Prague, CZ*
Július Štuller, *Institute of Computer Science, AS CR, Prague, CZ*
Peter Vojtáš, *Charles University, Prague, CZ*
Jakub Yaghob, *Charles University, Prague, CZ*
Stanislav Žák, *Institute of Computer Science, AS CR, Prague, CZ*
Filip Železný, *Czech Technical University, Prague, CZ*

Organizing Committee

Tomáš Horváth, (chair), *University of P.J. Šafárik, Košice, SK*
Hanka Bílková, *Institute of Computer Science, AS CR, Prague, CZ*
Peter Gurský, *University of P.J. Šafárik, Košice, SK*
Róbert Novotný, *University of P.J. Šafárik, Košice, SK*
Jana Pribolová, *University of P.J. Šafárik, Košice, SK*
Veronika Vaneková, *University of P.J. Šafárik, Košice, SK*

Organization

ITAT 2009 – 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
Slovak Society for Artificial Intelligence, SK

Table of Contents

Web Information Extraction systems for Web Semantization*

Jan Dedek

Department of Software Engineering, Faculty of Mathematics and Physics
Charles University in Prague, Czech Republic
dedek@ksi.mff.cuni.cz
Institute of Computer Science, Academy of Science of the Czech Republic
Prague, Czech Republic

Abstract. *In this paper we present a survey of web information extraction systems and semantic annotation platforms. The survey is concentrated on the problem of employment of these tools in the process of web semantization. We compare the approaches with our own solutions and propose some future directions in the development of the web semantization idea.*

1 Introduction

There exist many extraction tools that can process web pages and produce structured machine understandable data (or information) that corresponds with the content of a web page. This process is often called Web Information Extraction (WIE). In this paper we present a survey of web information extraction systems and we connect these systems with the problem of web semantization.

The paper is structured as follows. First we sketch the basic ideas of semantic web and web semantization. In the next two sections methods of web information extraction will presented. Then description of our solutions (work in progress) will continue. And finally just before the conclusion we will discuss the connection of WIE systems with the problem of web semantization.

1.1 The Semantic Web in use

The idea of the Semantic Web [4] (World Wide Web dedicated not only to human but also to machine – software agents) is very well known today. Let us just shortly demonstrate its use with respect to the idea of Web Semantization (see in next section).

The Fig. 1 shows a human user using the (Semantic) Web in three possible manners: a keyword query, a semantic query and by using a software agent. The difference between the first two manners (keyword and semantic query) can be illustrated with the question: "Give me a list of the names of E.U. heads of state."

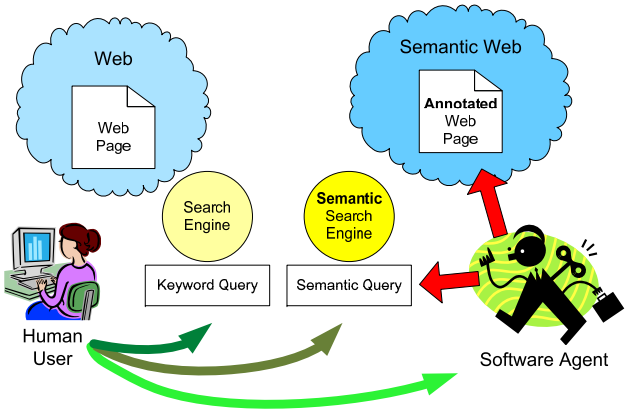


Fig. 1. The Semantic/Semantized Web in use.

This example from interesting article [16] by Ian Horrocks shows the big difference between use of a semantic query language instead of keywords. In the semantic case you should be given exactly the list of names you were requesting without having to pore through results of (probably more then one) keyword queries. Of course the user have to know the syntax of the semantic query language or have a special GUI[1] at hand.

The last and the most important possibility (in the semantic or semantized setting) is to use some (personalized) software agent that is specialized to tasks of some kind like planning a business trip or finding the most optimal choice from all the relevant job offers, flats for rent, cars for sale, etc.

Both the semantic querying and software agents engagement is actually impossible to realize without any kind of adaptation of the web of today in the semantic direction.

1.2 Web Semantization

The idea of Web Semantization [9] consist in gradual enrichment of the current web content as an automated process of third party annotation for mak-

* This work was partially supported by Czech projects: IS-1ET100300517, GACR-201/09/H057, GAUK 31009 and MSM-0021620838.

[1] Such handy GUI can be found for example in the KIM project [20].

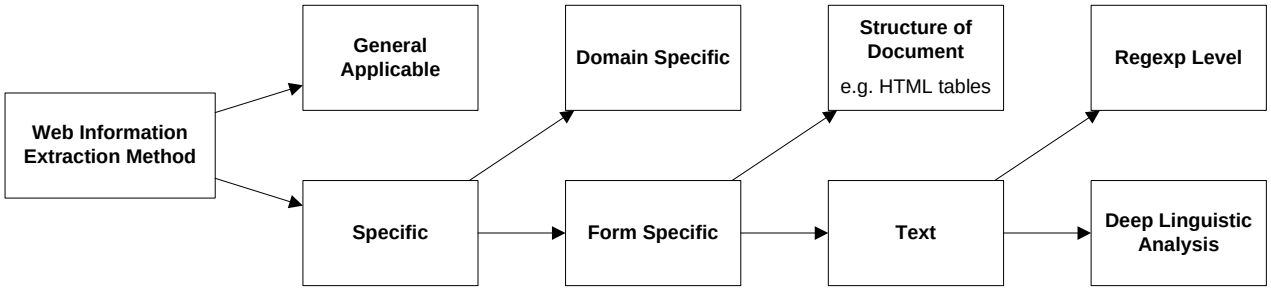


Fig. 2. Division of extraction methods.

ing at least a part of today's web more suitable for machine processing and hence enabling it intelligent tools for searching and recommending things on the web (see [3]).

The most strait forward idea is to fill a semantic repository with some information that is automatically extracted from the web and make it available to software agents so they could access to the web of today in semantic manner (e.g. through semantic search engine).

The idea of a semantic repository and a public service providing semantic annotations was experimentally realized in the very recognized work of IBM Almaden Research Center: the SemTag [13]. This work demonstrated that an automated semantic annotation can be applied in a large scale. In their experiment they annotated about 264 million web pages and generated about 434 millions of semantic tags. They also provided the annotations as a *Semantic Label Bureau* – a HTTP server providing annotations for web documents of 3rd parties.

2 Web information extraction

The task of a web information extraction system is to transform the web pages into program-friendly structures such as a relational database. There exists a rich variety of Web Information Extraction systems. The results generated by distinct tools usually can not be directly compared since the addressed extraction tasks are different. The extraction tasks can be distinguished according several dimensions: the task domain, the automation degree, the techniques used, etc. These dimensions are analyzed in detail in the recent publications [6] and [18]. Here we will concentrate on a little bit more specific division of WIE according to the needs of the Web Semantization (see in Sect. 5). The division is demonstrated on the Fig. 2 and should not be considered as disjoint division of the methods but rather as emphasization of different aspects of the methods. For example many extraction methods are domain and form specific at the same time.

The distinguishing between general applicable methods and the others that have meaningful application only in some specific setting (specific domain, specific form of input) is very important for Web Semantization because when we try to produce annotations in large scale, we have to control which web resource is suitable for which processing method (see in Sect. 5).

2.1 General applicable

The most significant (and probably the only one) generally applicable IE task is so called *Instance Resolution Task*. The task can be described as follows: Given a general ontology, find all the instances from the ontology that are present in the processed resource. This task is usually realized in two steps: (1) Named Entity Recognition (see in Sect. 3.1), (2) Disambiguation of ontology instances that can be connected with the found named entities. Success of the method can be strongly improved with coreference resolution (see in Sect. 3.1).

Let us mention several good representatives of this approach: the SemTag application [13], the KIM project [20] and the PANKOW annotation method [7] based on smart formulation of Google API queries.

2.2 Domain specific

Domain and from specific IE approaches are the typical cases. More specific information is more precise, more complex and so more useful and interesting. But the extraction method has to be trained to each new domain separately. This usually means indispensable effort.

A good example of domain specific information extraction system is SOBA [5]. This complex system is capable to integrate different IE approaches and extract information from heterogeneous data resources, including plain text, tables and image captions but the whole system is concentrated on the single domain

of football. Next similarly complex system is ArtEquAKT [1], which is entirely concentrated on the domain of art.

2.3 Form specific

Beyond general applicable extraction methods there exist many methods that exploit specific form of the input resource. The linguistic approaches usually process text consisting of natural language sentences. The structure-oriented approaches can be strictly oriented on tables [19] or exploit repetitions of structural patterns on the web page [21] (such algorithm can be only applicable to pages that contain more than one data record), and there are also approaches that use the structure of whole site (e.g. site of single web shop with summary pages with products connected with links to pages with details about single product) [17].

3 Information extraction from text-based resources

In this section we will discuss the information extraction from textual resources.

3.1 Tasks of information extraction

There are classical tasks of text preprocessing and linguistic analysis like

Text Extraction – e.g from HTML, PDF or DOC,
Tokenization – detection of words, spaces, punctuations, etc.,
Segmentation – sentence and paragraph detection,
POS Tagging – part of speech assignment, often including lemmatization and morphological analysis,
Syntactic Analysis (often called linguistic *parsing*) – assignment of the grammatical structure to given sentence with respect to given linguistic formalism (e.g. formal grammar),
Coreference Resolution (or *anaphora resolution*) – resolving what a pronoun, or a noun phrase refers to. These references often cross boundaries of a single sentence.

Besides these classical general applicable tasks, there are further well defined tasks, which are more closely related to the information extraction. These tasks are domain dependent. These tasks were widely developed in the MUC-6 conference 1995 [15] and considered as semantic evaluation in the first place. These information extraction tasks are:

Named Entity Recognition: This task recognizes and classifies named entities such as persons, locations, date or time expression, or measuring units. More complex patterns may also be recognized as structured entities such as addresses.
Template Element Construction: Populates templates describing entities with extracted roles (or attributes) about one single entity. This task is often performed stepwise sentence by sentence, which results in a huge set of partially filled templates.
Template Relation Construction: As each template describes information about one single entity, this tasks identifies semantic relations between entities.
Template Unification: Merges multiple elementary templates that are filled with information about identical entities.
Scenario Template Production: Fits the results of Template Element Construction and Template Relation Construction into templates describing pre-specified event scenarios (pre-specified "queries on the extracted data").

Appelt and Israel [2] wrote an excellent tutorial summarizing these traditional IE tasks and systems built on them.

3.2 Information extraction benchmarks

Contrary to the WIE methods based on the web page structure, where we (the authors) do not know about any well established benchmark for these methods[2], the situation in the domain of text based IE is fairly different. There are several conferences and events concentrated on the support of automatic machine processing and understanding of human language in text form. Different research topics as text (or information) retrieval[3], text summarization[4] are involved.

On the filed of information extraction, we have to mention the long tradition of the Message Understanding Conference[5] [15] starting in 1987. In 1999 the event of *Automatic Content Extraction (ACE) Evaluation*[6] started, which is becoming a track in the Text Analysis Conference (TAC)[7] this year (in 2009).

[2] It is probably at least partially caused by the vital development of the presentation techniques on the web that is still well in progress.
[3] e.g. Text REtrieval Conference (TREC) `http://trec.nist.gov/`
[4] e.g. Document Understanding Conferences `http://duc.nist.gov/`
[5] Briefly summarized in `http://en.wikipedia.org/wiki/Message_Understanding_Conference`.
[6] `http://www.itl.nist.gov/iad/mig/tests/ace/`
[7] `http://www.nist.gov/tac`

All these events prepare several specialized datasets together with information extraction tasks and play an important role as information extraction benchmarks.

4 Our solutions

4.1 Extraction based on structural similarity

Our first approach for the web information extraction is to use the structural similarity in web pages containing large number of table cells and for each cell a link to detailed pages. This is often presented in web shops and on pages that presents more than one object (product offer). Each object is presented in a similar way and this fact can be exploited.

As web pages of web shops are intended for human usage creators have to make their comprehension easier. Acquaintance with several years of web shops has converged to a more or less similar design fashion. There are often cumulative pages with many products in a form of a table with cells containing a brief description and a link to a page with details about each particular product.

Our main idea is to use a DOM tree representation of the summary web page and by breadth first search encounter similar subtrees. The similarity of these subtrees is used to determine the data region – a place where all the objects are stored. It is represented as a node in the DOM tree, underneath it there are the similar sub-trees, which are called data records.

We[8] have developed and implemented this idea [14] on the top of Mozilla Firefox API and experimentally tested on table pages from several domains (cars, notebooks, hotels). Similarity between subtrees was Levenshtein editing distance (for a subtree considered as a linear string), learning thresholds for decision were trained.

4.2 Linguistic information extraction

Our second approach [11, 12, 10] for the web information extraction is based on deep linguistic analysis. We have developed a rule-based method for extraction of information from text-based web resources in Czech and now we are working on its adaptation to English. The extraction rules correspond to tree queries on linguistic (syntactic) trees made form particular sentences. We have experimented with several linguistic tools for Czech, namely Tools for machine annotation – PDT 2.0 and the Czech WordNet.

Our present system captures text of web-pages, annotates it linguistically by PDT tools, extracts data and stores the data in an ontology. We have made initial experiments in the domain of reports of traffic accidents. The results showed that this method can e.g. aid summarization of the number of injured people.

To avoid the need of manual design of extraction rules we focused on the data extraction phase and made some promising experiments [8] with the machine learning procedure of Inductive Logic Programming for automated learning of the extraction rules.

This solution is directed to extraction of information which is closely connected with the meaning of text or meaning of a sentence.

5 The Web Semantization setting

In this section we will discuss possibilities and obstructions connected with the employment of web information extraction systems in the process of web semantization.

One aspect of the realization of the web semantization idea is the problem of integration of all the components and technologies starting with web crawling, going through numerous complex analyses (document preprocessing, document classification, different extraction procedures), output data integration and indexing, and finally implementation of query and presentation interface. This elaborate task is neither easy nor simple but today it is solved in all the extensive projects and systems mentioned above.

The novelty that web semantization brings into account is the cross domain aspect. If we do not want to stay with just general ontologies and general applicable extraction methods then we need a methodology how to deal with different domains. The system has to support extension to a new domain in generic way. So we need a methodology and software to support this action. This can for example mean: to add a new ontology for the new domain, to select and train proper extractors and classifiers for the suitable input pages.

5.1 User initiative and effort

An interesting point is the question: Whose effort will be used in the process of supporting new domain in the web semantization process? How skilled such user has to be? There are two possibilities (demonstrated on the Fig 3). The easier one is that we have to employ very experienced expert who will decide about the new domain and who will also realize the support needed for the new domain. In the Fig 3 this situation is labeled as *Provider Initiated* and *Provider Trained* because the expert works on the side of the system that provides the semantics.

[8] Thanks go mainly to Dušan Maruščák and Peter Vojtáš.

mentation of tables. In SIGMOD '04: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, New York, NY, USA, ACM, 2004, 119–130.
18. B. Liu: *Web Data Mining.* Springer-Verlag, 2007.
19. D. Pinto, A. Mccallum, X. Wei, and B.W. Croft: *Table extraction using conditional random fields.* In SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval, New York, NY, USA, ACM Press, 2003, 235–242.
20. B. Popov, A. Kiryakov, D. Ognyanoff, D. Manov, and A. Kirilov: *Kim – a semantic platform for information extraction and retrieval.* Nat. Lang. Eng., 10, 3-4, 2004, 375–392.
21. H. Zhao, W. Meng, Z. Wu, V. Raghavan, and C. Yu: *Fully automatic wrapper generation for search engines.* In WWW Conference, 2005, 66–75.

PrefWork - a framework for the user preference learning methods testing*

Alan Eckhardt[1,2]

[1] Department of Software Engineering, Charles University,
[2] Institute of Computer Science, Czech Academy of Science,
Prague, Czech Republic
eckhardt@ksi.mff.cuni.cz

Abstract. *PrefWork is a framework for testing of methods of induction of user preferences. PrefWork is thoroughly described in this paper. A reader willing to use PrefWork finds here all necessary information - sample code, configuration files and results of the testing are presented in the paper. Related approaches for data mining testing are compared to our approach. There is no software available specially for testing of methods for preference learning to our best knowledge.*

1 Introduction

User preference learning is a task that allows many different approaches. There are some specific issues that differentiate this task from a usual task of data mining. User preferences are different from measurements of a physical phenomenon or a demographic information about a country; they are much more focused on the objects of interest and involve psychology or economy.

When we want to choose the right method for user preference learning, e.g. for an e-shop, the best way is to evaluate all possible methods and to choose the best one. The problems with the testing of methods for preference learning are:

- how to evaluate these methods automatically,
- how to cope with different sources of data, with different types of attributes,
- how to measure the suitability of a method,
- to personalise the recommendation for every user individually.

2 Related work

The most popular tool related to PrefWork is the open source projec t Weka [1]. Weka is in development for many years and has achieved to become the most widely used tool for data mining. It offers many classificators, regression methods, clustering, data preprocessing, etc. However this variability is also its weakness - it can be used for any given task, but it has to be customised, the developer has to choose from a very wide range of possibilities. For our case, Weka is too strong.

RapidMiner [2] has a nice user interface and is in a way similar to Weka. It is also written in Java and has source codes available. However the ease of use is not better than that of Weka. The user interface is nicer than in Weka but the layout of Weka is more intuitive (allowing to connect various components that are represented on a plane).

R [3] is a statistical software that is based on its own programming language. This is the biggest inconvenience - a user willing to use R has to learn yet another programming language.

There are also commercial tools as SAS miner [4], SPSS Clementine [5], etc. We do not consider these, because of the need to buy a (very expensive) licence.

We must also mention the work of T. Horváth - Winston [6], which was developed recently. Winston may suit our needs, because it is light-weighted, has also a nice user interface, but in the current stage there are few methods and no support for the method testing. It is more a tool for the data mining lecturing than the real world method testing.

We are working with ratings the user has associated to some items. This use-case is well-known and used across the internet. An inspiration for extending our framework is many other approaches to user preference elicitation. An alternative to ratings has been proposed in [7, 8] - instead of ratings, the system requires direct feedback from the user about the attribute values. The user has to specify in which values the given recommendation can be improved. This approach is called critique based recommendations.

Among other approaches, we should mention also work of Kiessling [9], which uses the user behaviour as the source for the preference learning.

We also need some implementations of algorithms of the user preference learning that are publicly available for being able to compare various methods among themselves. This is a strength of PrefWork - any existing method, which works with ratings, can be

* The work on this paper was supported by Czech projects MSM 0021620838, 1ET 100300517 and GACR 201/09/H057.

integrated into PrefWork using a special adaptor for each tool (see Section 4.3). There is a little bit old implementation of collaborative filtering Cofi [10] and a brand new one (released 7.4.2009) Mahout [11], developed by Apache Lucene project. Cofi uses Taste framework [12], which became a part of Mahout. The expectations are that Taste in Mahout would perform better than Cofi, so we will try to migrate our PrefWork adaptor for Cofi to Mahout. Finally there is IGAP [13] - a tool for learning of fuzzy logic programs in form of rules, which correspond to user preferences. Unfortunately, IGAP is not yet available publicly for download.

We did not find any other mining algorithm specialised on user preferences available for free download, but we often use already mentioned Weka. It is a powerful tool that can be more or less easily integrated into our framework and provide a reasonable comparison of a non-specialised data mining algorithm to other methods that are specialised for preference learning.

3 User model

For making this article self-contained, we describe in brief our user model, as in [14]. In this section, we describe our user model. This model is based on a scoring function that assigns the score to every object. User rating of an object is a fuzzy subset of X(set of all objects), i.e. a function $R(o) : X \to [0, 1]$, where 0 means the least preferred and 1 means the most preferred object. Our scoring function is divided into two steps.

Local preferences In the first step, which we call local preferences, all attribute values of object o are normalised using fuzzy sets $f_i : D_{A_i} \to [0, 1]$. These fuzzy sets are also called objectives or preferences over attributes. With this transformation, the original space of objects' attributes $X = \prod_{i=1}^{N} D_{A_i}$ is transformed into $X' = [0, 1]^N$. Moreover, we know that the object $o \in X'$ with transformed attribute values equal to $[1, \ldots, 1]$ is the most preferred object. It probably does not exist in the real world, though. On the other side, the object with values $[0, \ldots, 0]$ is the least preferred, which is more probable to be found in reality.

Global preferences In the second step, called global preferences, the normalised attribute values are aggregated into the overall score of the object using an aggregation function $@ : [0, 1]^N \to [0, 1]$. Aggregation function is also often called utility function.

Aggregation function may have different forms; one of the most common is a weighted average, as in the following formula:

$$@(o) = (2 * f_{Price}(o) + 1 * f_{Display}(o) + 3 * f_{HDD}(o) + 1 * f_{RAM}(o))/7\,,$$

where f_A is the fuzzy set for the normalisation of attribute A.

Another totally different approach was proposed in [15]. It uses the training dataset as partitioning of normalised space X'. For example, if we have an object with normalised values $[0.4, 0.2, 0.5]$ with rating 3, any object with better attribute values (e.g. $[0.5, 0.4, 0.7]$) is supposed to have the rating at least 3. In this way, we can find the highest lower bound on any object with unknown rating. In [15] was also proposed a method for interpolation of ratings between the objects with known ratings and even using the ideal (non-existent) virtual object with normalised values $[1, ..., 1]$ with rating 6.

4 PrefWork

Our tool PrefWork was initially developed as a master thesis of Tomáš Dvořák [16], who has implemented it in Python. In this initial implementation, only Id3 decision trees and collaborative filtering was implemented. For better ease of use and also for the possibility of integrating other methods, PrefWork was later rewritten to Java by the author. Many more possibilities were added until the today state. In the following sections, components of PrefWork are described.

Most of the components can be configured by XML configurations. Samples of these configurations and Java interfaces will be provided for each component. We omit methods for configuration from Java interfaces such as `configTest(configuration,section)` which is configured using a configuration from a section in an XML file. Also data types of function arguments are omitted for brevity.

4.1 The workflow

In this section a sample of workflow with PrefWork is described.

The structure of PrefWork is in Figure 1. There are four different configuration files - one for database access configuration (confDbs), one for datasources (confDatasources), one for methods (confMethods) and finally one for PrefWork runs (confRuns). A run consists of three components - a set of methods, a set of datasets and a set of ways to test the method. Every method is tested on every dataset using every way to

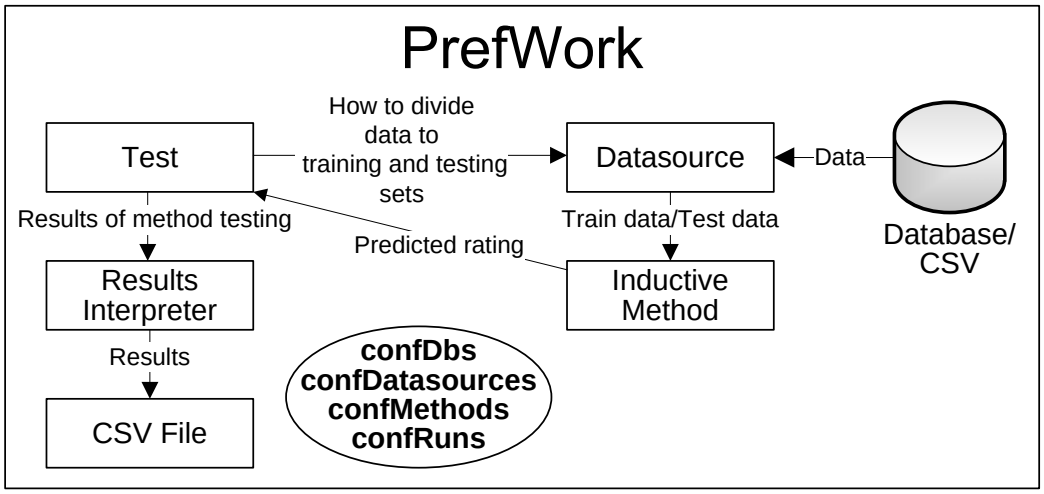


Fig. 1. PrefWork structure.

test. For each case, results of the testing are written into a csv file.

A typical situation a researcher working with PrefWork finds himself in is: "I have a new idea X. I am really interested, how it performs on that dataset Y."

The first thing is to create corresponding Java class X that implements interface InductiveMethod (see 4.3) and add a section X to confMethods.xml. Then copy an existing entry defining a run (e.g. IFSA, see 4.5) and add method X to section `methods`. Run ConfigurationParser and correct all errors in the new class (and there will be some, for sure). After the run has finished correctly, process the csv file with results to see how X performed in comparison with other methods.

A similar case is when introducing a new dataset into PrefWork - confDatasets.xml and confDBs.xml have to be edited if the data are in SQL database or in a csv file. Otherwise a new Java class (see 4.2) able to handle the new type of data has to be created. For example, we still have not implemented the class for handling of arff files - these files have the definition of attributes in themselves, so the configuration in confDatasets.xml would be much more simple (see Section 4.2 for an example of a configuration of a datasource with its attributes).

4.2 Datasource

Datasource is, as the name hints, the source of data for inductive methods. Currently, we are working only with ratings of objects. Data are vectors, where the first three attributes typically are: the user id, the object id and the rating of the object. The attributes of the object follow. There is a special column that contains a random number associated to each rating. Its purpose will described later.

Every datasource has to implement the following methods:

```
interface BasicDataSource{
  boolean hasNextRecord();
  void setFixedUserId(value);
  List<Object> getRecord();
  Attribute[] getAttributes();
  Integer getUserId();
  void setLimit(from, to,
        recordsFromRange);
  void restart();
  void restartUserId();
}
```

There are two main attributes of datasource - a list of all users and a list of ratings of the current user. `getUserId` returns the id of the current user. The most important function is `getRecord`, which returns a vector containing the rating of the object and its attributes. Following calls of `getRecords` return all objects rated by the current user. A typical sequence is:

```
int userId = data.getUserId();
data.setFixedUserId(userId);
data.restart();
while(data.hasNextRecord()){
  List<Object> record =
                  data.getRecord();

  // Work with the record
  ...
}
```

Another important function is `setLimit`, which limits the data using given boundaries `from` and `to`. The random number associated to each vector returned by `getRecord` has to fit into this interval. If `recordsFromRange` is false, then the random number should be outside of the given interval on the contrary. This method is used when dividing the data into training and testing sets. For example, let us divide the data to 80% training set and 20% testing set. First, we call `setLimit(0.0,0.8,true)` and let the method train on these data. Then, `setLimit( 0.0,0.8,false)` is executed and vectors returned by the datasource are used for the testing of the method.

Let us show a sample configuration of a datasource that returns data about notebooks:

```
<NotebooksIFSA>
  <attributes>
   <attribute><name>userid</name>
     <type>numerical</type>
   </attribute>
   <attribute><name>notebookid</name>
     <type>numerical</type>
   </attribute>
   <attribute><name>rating</name>
     <type>numerical</type>
    </attribute>
   <attribute><name>price</name>
     <type>numerical</type>
   </attribute>
   <attribute><name>producer</name>
     <type>nominal</type>
   </attribute>
   <attribute><name>ram</name>
     <type>numerical</type>
   </attribute>
   <attribute><name>hdd</name>
     <type>numerical</type>
   </attribute>
  </attributes>
  <recordsTable>
    note_ifsa
  </recordsTable>
  <randomColumn>
    randomize
  </randomColumn>
  <userID>userid</userID>
  <usersSelect>
  select distinct userid from note_ifsa
  </usersSelect>
</NotebooksIFSA>
```

First, a set of attributes is defined. Every attribute has a name and a type - numerical, nominal or list. An example of list attribute is actors in a film. This attribute can be found in the IMDb dataset [17].

Let us also note the select for obtaining the user ids (section `usersSelect`) and the name of the column that contains the random number used in setLimit (`randomColumn`).

Other types of user preferences. PrefWork as it is now supports only ratings of objects. There are many more types of data containing user preferences - user clickstream, user profile, filtering of the result set etc.

PrefWork does not work with any information about the user, either demographic like age, sex, place of birth, occupation etc. or his behaviour. These types of information may bring a large improvement in the prediction accuracy, but they are typically not present - users do not want to share any personal information for the sole purpose of a better recommendation. Another issue is the complexity of user information; a semantic processing would have to be used.

4.3 Inductive method

InductiveMethod is the most important interface - it is what we want to evaluate. Inductive method has two main methods:

```
interface InductiveMethod {
  int buildModel(trainingDataset,
        userId);
  Double classifyRecord(record,
        targetAttribute);
}
```

`buildModel` uses the training dataset and the userId for the construction of a user preference model. After having it constructed, the method is tested - it is being given records via method `classifyRecord` and is supposed to evaluate them.

Various inductive methods were implemented. Among the most interesting are our method Statistical ([18, 15]) and Instances ([15]), WekaBridge that allows to use any method from Weka (such as Support vector machine) and ILPBridge that transforms data to a prolog program and then uses Progol [19] to create the user model. CofiBridge allows to use Cofi as a PrefWork InductiveMethod.

A sample configuration of method Statistical is:

```
<Statistical>
  <class>Statistical</class>
  <rater>
   <class>WeightAverage</class>
   <weights>VARIANCE</weights>
  </rater>
  <representant>
   <class>AvgRepresentant</class>
  </representant>
```

```
  <numericalNormalizer>
       Linear
  </numericalNormalizer>
  <nominalNormalizer>
       RepresentantNormalizer
  </nominalNormalizer>
  <listNormalizer>
       ListNormalizer
  </listNormalizer>
</Statistical>
```

Every method requires a different configuration, only the name of the class is obligatory. Note that the methods based on our two-step user model (Statistical and Instances for now) can be easily configured to test different heuristics for the processing of different types of attributes. Configuration contains three sections: `numericalNormalizer`, `nominalNormalizer` and `listNormalizer` for the specification of the method for the particular type of attribute. Also see Section 4.5 for an example of this configuration.

4.4 Ways of the testing of the method

Several possible ways for the testing of methods can be defined, the division to training and testing sets is the most typically used. The method is trained on the training set (using buildModel) and then tested on the testing set (using classifyRecord). Another typical method is k-fold cross validation that divides data into k sets. In each of k runs, one set is used as the testing set and the rest as the training set.

```
interface Test {
  void test(method, trainDataSource,
         testDataource);
}
```

When the method is tested, the results in the form userid, objectid, predictedRating, realUserRating have to be processed. The interpretation is done by a TestResultsInterpreter. The most common is DataMiningStatistics, which computes such measures as correlation, RMSE, weighted RMSE, MAE, Kendall rank tau coefficient, etc. Others are still waiting to be implemented - ROC curves or precision-recall statistics.

```
abstract class TestInterpreter {
  abstract void writeTestResults(
                       testResults);
}
```

4.5 Configuration parser

The main class is called ConfigurationParser. The definition of one test follows:

```
<IFSA>
  <methods>
   <method>
     <name>Statistical</name>
     <numericalNormalizer>
       Standard2CPNormalizer
     </numericalNormalizer>
   </method>
   <method><name>Statistical</name>
   </method>
   <method><name>Mean</name></method>
   <method><name>SVM</name></method>
  </methods>
  <dbs>
   <db>
    <name>MySQL</name>
    <datasources>NotebooksIFSA
    </datasources>
   </db>
  </dbs>
  <tests>
   <test>
    <class>TestTrain</class>
    <ratio>0.05</ratio>
    <path>resultsIFSA</path>
    <testInterpreter>
      <class>DataMiningStatistics
      </class>
    </testInterpreter>
   </test>
   <test>
    <class>TestTrain</class>
    <ratio>0.1</ratio>
    <path>resultsIFSA</path>
    <testInterpreter>
      <class>DataMiningStatistics
      </class>
    </testInterpreter>
   </test>
  </tests>
</IFSA>
```

First, we have specified which methods are to be tested - in our case it is two variants of Statistical, then Mean and SVM. Note that some attributes of Statistical, which was defined in confMethods, can be "overridden" here. The basic configuration of Statistical is in Section 4.3. Then the datasource for testing of the methods is specified – we are using MySql database with datasource NotebooksIFSA. Several datasources or databases can be specified here. Finally, the ways of the testing and interpretation are given in section `tests`. TestTrain requires ratio of the training and the testing sets, the path where the results are to be written, and the interpretation of the test results.

date;Ratio;dataset;method;userId;mae;rmse;weightedRmse;monotonicity;tau;weightedTau;correlation;buildTime;
testTime;countTrain;countTest;countUnableToPredict
28.4.2009
12:18;0,05;NotebooksIFSA;Statistical,StandardNorm2CP;1;0,855;0,081;1,323;1,442;0,443;0,358;0,535;94;47;10;188;0;
28.4.2009
12:18;0,05;NotebooksIFSA;Statistical,StandardNorm2CP;1;0,868;0,078;1,216;1,456;0,323;0,138;0,501;32;0;13;185;0;
28.4.2009
12:18;0,05;NotebooksIFSA;Statistical,StandardNorm2CP;1;0,934;0,083;1,058;1,873;0,067;0,404;0,128;31;16;12;186;0;
28.4.2009 12:31;0,025;NotebooksIFSA;Statistical,Peak;1;0,946;0,081;1,161;1,750;0,124;0,016;0,074;15;16;4;194;0
28.4.2009 12:31;0,025;NotebooksIFSA;Statistical,Peak;1;0,844;0,076;1,218;1,591;0,224;0,215;0,433;0;16;6;192;0
28.4.2009 12:31;0,025;NotebooksIFSA;Statistical,Peak;1;1,426;0,123;1,407;1,886;0,024;0,208;-0,063;16;0;4;194;0

Fig. 2. A sample of results in a csv file.

The definitions of runs are in confRuns.xml in section `runs`. The specification of the run to be executed is in section `run` of the same file.

4.6 Results of testing

In Figure 2 is a sample of the resulting csv file. In our example, there are three runs with method Statistical with normaliser StandardNorm2CP and three runs with normaliser Peak. Runs were performed on different settings of the training and the testing sets, so the results are different even for the same method.

The results contain all necessary information required for generation of a graph or a table with the results. Csv format was chosen for its simplicity and wide acceptance, so any other possible software can handle it. We are currently using Microsoft Excel and its Pivot table that allows aggregation of results by different criteria. Among other possibilities is also the already mentioned R [3].

Example figures of the output of PrefWork are in Figures 3 and 4. The lines represent different methods, X axis represents the size of the training set and the Y axis the value of the error function. In Figure 3 the error function is Kendall rank tau coefficient (the higher it is the better) and in Figure 4 is RMSE weighted by the original rating (the lower the better). The error function can be chosen, as is described in Section 4.4.

It is impossible to compare PrefWork to another framework generally. A simple comparison to other such systems is in Section 2. This can be done only qualitatively; there is no attribute of frameworks that can be quantified. The user itself has to choose among them the one that suits his needs the most.

4.7 External dependencies

PrefWork is dependent on some external libraries. Two of them are sources for inductive methods - Weka [1] and Cofi [10]. Cofi also requires `taste.jar`.

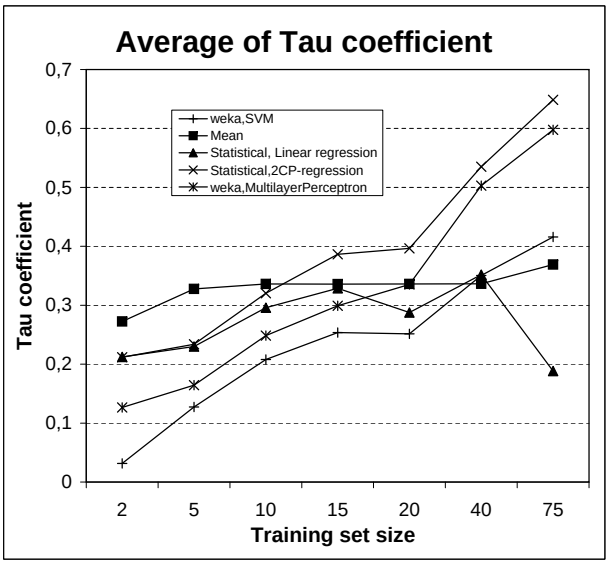


Fig. 3. Tau coefficient.

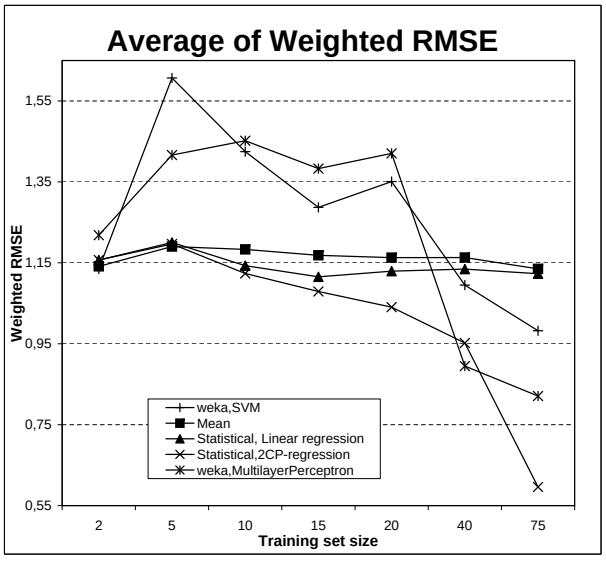


Fig. 4. Weighted RMSE.

PrefWork requires following jars to function correctly:

Weka	weka.jar
Cofi	cofi.jar
Cofi	taste.jar
Logging	log4j.jar
CSV parsing	opencsv-1.8.jar
Configuration	commons-configuration-1.5.jar
Configuration	commons-lang-2.4.jar
MySql	mysql-connector-java-5.1.5-bin.jar
Oracle	ojdbc1410.2.0.3.jar

Tab. 1. Libraries required by PrefWork.

5 Conclusion

PrefWork has been presented in this paper with a thorough explanation and description of every component. Interested reader should be now able to install PrefWork, run it, and implement a new inductive method or a new datasource.

The software can be downloaded at `http://www.ksi.mff.cuni.cz/~eckhardt/PrefWork.zip` as an Eclipse project containing all java sources and all required libraries or can be downloaded as SVN checkout at [20]. The SVN archive contains Java sources and sample configuration files.

5.1 Future work

We plan to introduce time dimension to PrefWork. Netflix [21] datasets uses a timestamp for each rating. This will enable to study the evolution of the preferences in time, which is a challenging problem. However, the integration of the time dimension into PrefWork can be done in several ways and the right one is yet to be chosen.

Allowing other sources of data apart from the ratings is a major issue. The clickthrough data can be collected without any effort of the user and can be substantially larger than the number of ratings. But its integration into PrefWork would require a large reorganisation of existing methods.

References

1. I.H. Witten, E. Frank: *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd Edition. Morgan Kaufmann, San Francisco (2005).
2. I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, T. Euler: *Yale: Rapid prototyping for complex data mining tasks.* In Ungar, L., Craven, M., Gunopulos, D., Eliassi-Rad, T., eds.: KDD'06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, New York, NY, USA, ACM (August 2006), 935–940.
3. R-project. `http://www.r-project.org/`.
4. SAS enterprise miner. `http://www.sas.com/`.
5. SPSS Clementine. `http://www.spss.com/software/modeling/modeler/`.
6. Š. Pero, T. Horváth: *Winston: A data mining assistant.* In: To appear in proceedings of RDM 2009, 2009.
7. P. Viappiani, B. Faltings: *Implementing example-based tools for preference-based search.* In: ICWE'06: Proceedings of the 6th international conference on Web engineering, New York, NY, USA, ACM, 2006, 89–90.
8. P. Viappiani, P. Pu, B. Faltings: *Preference-based search with adaptive recommendations.* AI Commun. **21**, 2-3, 2008, 155–175.
9. S. Holland, M. Ester, W. Kiessling: *Preference mining: A novel approach on mining user preferences for personalized applications.* In: Knowledge Discovery in Databases: PKDD 2003, Springer Berlin / Heidelberg, 2003, 204–216.
10. Cofi: A Java-Based Collaborative Filtering Library. `http://www.nongnu.org/cofi/`.
11. Apache Mahout project. `http://lucene.apache.org/mahout/`.
12. Taste project. `http://taste.sourceforge.net/old.html`.
13. T. Horváth, P. Vojtáš: *Induction of fuzzy and annotated logic programs.* In Muggleton, S., Tamaddoni-Nezhad, A., Otero, R., eds.: ILP06 - Revised Selected papers on Inductive Logic Programming. Number 4455 in Lecture Notes In Computer Science, Springer Verlag, 2007, 260–274.
14. A. Eckhardt: *Various aspects of user preference learning and recommender systems.* In Richta, K., Pokorný, J., Snášel, V., eds.: DATESO 2009. CEUR Workshop Proceedings, Česká technika - nakladatelství ČVUT, 2009, 56–67.
15. A. Eckhardt, P. Vojtáš: *Considering data-mining techniques in user preference learning.* In: 2008 International Workshop on Web Information Retrieval Support Systems, 2008, 33–36.
16. T. Dvořák: *Induction of user preferences in semantic web*, in Czech. Master Thesis, Charles University, Czech Republic, 2008.
17. The Internet Movie Database. `http://www.imdb.com/`.
18. A. Eckhardt: *Inductive models of user preferences for semantic web.* In Pokorný, J., Snášel, V., Richta, K., eds.: DATESO 2007. Volume 235 of CEUR Workshop Proceedings., Matfyz Press, Praha, 2007, 108–119.
19. S. Muggleton: *Learning from positive data.* 1997, 358–376
20. PrefWork - a framework for testing methods for user preference learning. `http://code.google.com/p/prefwork/`.
21. Netflix dataset, `http://www.netflixprize.com`.

Boosted surrogate models in evolutionary optimization*

Martin Holeňa

Institute of Computer Science, Academy of Sciences of the Czech Republic,
Pod Vodárenskou věží 2, 18207 Praha 8, Czech Republic, martin@cs.cas.cz, web: cs.cas.cz/~martin

Abstract. *The paper deals with surrogate modelling, a modern approach to the optimization of empirical objective functions. The approach leads to a substantial decrease of time and costs of evaluation of the objective function, a property that is particularly attractive in evolutionary optimization. In the paper, an extension of surrogate modelling with regression boosting is proposed. Such an extension increases the accuracy of surrogate models, thus also the agreement between results of surrogate modelling and results of the intended optimization of the original objective function. The proposed extension is illustrated on a case study in the area of searching catalytic materials optimal with respect to their behaviour in a particular chemical reaction. A genetic algorithm developed specifically for this application area is employed for optimization, multilayer perceptrons serve as surrogate models, and a method called AdaBoost.R2 is used for boosting. Results of the case study clearly confirm the usefulness of boosting for surrogate modelling.*

1 Introduction

For more than two decades, *evolutionary algorithms*, especially their most frequently encountered representative – *genetic algorithms*, belong to the most successful methods for solving difficult optimization tasks [3, 11, 31, 32, 42]. The popularity of evolutionary algorithms is to some extent due to their biological inspiration, which increases their comprehensibility outside computer science. Nevertheless, they share several purely mathematical properties of all stochastic optimization methods, most importantly, the valuable ability of to escape a local optimum and continue the search for a global one, and the restriction of the information on which they rely to function values only. Consequently, they do not need information about gradients or second-order partial derivatives, differently to smooth optimization methods (such as steepest descent, conjugate gradient methods, the popular Levenberg-Marquardt method, etc.). This makes them particularly attractive for the optimization of *empirical objective functions*, the values of which cannot be analytically computed, but have to be obtained experimentally, through some measurement or testing. Indeed, the impossibility to compute analytically the function values of such a function makes also an analytical computation of its gradient and second-order derivatives impossible, whereas measurement errors usually hinder obtaining sufficiently accurate estimates of the derivatives.

Like other methods relying solely on function values, evolutionary algorithms need the objective function to be evaluated in quite a large number of points. In the context of optimization of empirical objective functions, this can be quite disadvantageous because the evaluation of such a function in the points forming one generation of an evolutionary algorithm is often costly and time-consuming. Hence, the above mentioned advantages of using evolutionary algorithms for the optimization of empirical objective functions are frequently counterbalanced by considerably high costs and time needed for the evaluation of such functions. An area, where the trade-off between successful optimization and costly objective function evaluations plays a crucial role, is the computer-aided search for new materials and chemicals optimal with respect to certain properties [2]. Here, evolutionary algorithms are used in more than 90 % of optimization tasks, and the rarely encountered alternatives are simulated annealing [9, 22, 23], simplex method [17], and holographic search strategy [37, 38, 41], which also use solely function values, therefore needing a similarly high number of objective function evaluations as evolutionary algorithms. Testing a generation of materials or chemicals typically needs hours to days of time and costs hundreds to thousands euros. Therefore, the evolutionary optimization rarely runs for more than ten generations.

The usual approach to decreasing the cost and time of optimization of empirical objective functions is to evaluate the objective function only sometimes and to evaluate a suitable regression model of that function otherwise. The employed model is termed *surrogate model* of the empirical objective function, and the approach is referred to as *surrogate modelling*. Needless to say, the time and costs needed to evaluate a regression model are negligible compared to an empirical objective function. However, it must not be forgotten that the final optimized function coincides with the original empirical objective function only in some points, whereas in the remaining points it coin-

* The research reported in this paper has been supported by the grant No. 201/08/1744 of the Grant Agency of the Czech Republic and partially supported by the Institutional Research Plan AV0Z10300504.

cides only with its surrogate model. Consequently, the agreement between the results of surrogate modelling and the results of the intended optimization of the original objective function depends on the accuracy of the approximation of the original objective function by the surrogate model.

This paper suggests to increase the accuracy of surrogate models by means of boosting. Boosting is a popular approach to increasing the accuracy of classification, and due to the success of classification boosting, also several methods of regression boosting have already been proposed. However, so far no attempt has been reported to combine regression boosting with surrogate modelling. Hence, the purpose of the research reported in the paper is basically a proof of concept: to extend surrogate modelling through the incorporation of regression boosting, and to validate that extension on several sufficiently complex case studies. One of those case studies is described in the paper.

In the following section, basic principles of surrogate modelling and its strategies in evolutionary optimization are recalled, and important surrogate models are listed. Section 3 recalls the principles of boosting and explains a particular method of regression boosting that will be employed later in a case study in materials science. That case study is sketched and its main results are presented in Section 4.

2 Surrogate modelling

Surrogate modelling is a general approach to the optimization of costly objective functions in which the evaluation of the objective function is restricted to points that are considered to be most important for the progress of the employed optimization method [5, 25, 27, 30, 39, 40]. It is most frequently encountered in connection with the optimization of empirical objective functions, but has been equally successfully applied also to expensive optimization tasks in engineering design in which the objective function is not empirical, but its evaluation is connected with intensive computations [25]. In the context of computer-aided search for new materials and chemicals optimal with respect to certain properties, surrogate modelling can be viewed as replacing real experiments with simulated virtual experiments in a computer: such virtual experiments are sometimes referred to as *virtual screening* [2].

Although surrogate modelling is a general optimization approach (cf. its application in the context of conventional optimization in [5]), it is most frequently encountered in connection with evolutionary algorithms. The reason is that in evolutionary optimization, the approach leads to the approximation of the landscape of the fitness function, i.e., to a method that is known to be useful in general [19, 20, 29]. In evolutionary algorithms, most important for the progress of the method are on the one hand points that best indicate the global optimum (typically through highest values of the fitness function), on the other hand points that most contribute to the diversity of the population.

In the context of evolutionary optimization, surrogate modelling has the following main steps:

(i) Collecting an initial set of points in which the objective function has already been empirically evaluated. This can be the first generation or several first generations of the evolutionary algorithm, but such points are frequently available in advance.

(ii) Approximating the objective function by a surrogate model, with the use of the set of all points in which it has been empirically evaluated.

(iii) Running the evolutionary algorithm for a population considerably larger than is the desired population size, with the empirical objective function replaced by the surrogate model.

(iv) Forming the next generation of the desired size as a subset of the large population obtained in the preceding step that includes points most important according to considered criteria for the progress of optimization (such as indication of global optimum, diversity).

(v) Empirically evaluating the objective function in all points that belong to the next generation of the desired size, and returning to step (ii).

Actually, the above steps (ii)–(v) correspond to only one possible strategy of surrogate modelling in evolutionary optimization: the *individual-based control*, sometimes also referred to as *pre-selection* [40]. An alternative strategy to the steps (ii)–(v) is to run the algorithm for only the desired population size, interleaving one generation/several generations in which the original objective function is empirically evaluated with a certain number of generations in which the surrogate model is evaluated. This is the *generation-based control* of surrogate modelling in evolutionary optimization.

For empirical objective functions, it is typical to be highly nonlinear. Therefore, nonlinear regression models should be used as surrogate models. They can be basically divided into two large groups according to whether the set of functions among which the surrogate model has to be chosen has an explicit finite parametrization.

1. So far, mostly *parametric models* have been used for surrogate modelling. From the point of view of their role in this context and/or their overall importance, the following kinds of parametric nonlinear regression models are most worth mentioning:
   (i) *Multilayer feed-forward neural networks*, more precisely, the nonlinear mappings computed

by such networks. Their attractiveness for nonlinear regression in general and for surrogate modelling in particular [20] is due to their *universal approximation capability*, which actually means that linear spaces of functions computed by certain families of multilayer feed-forward neural networks are dense in some general function spaces [18, 21, 26]. For example, considering the most common representative of such networks – multilayer perceptrons, the linear space formed by all functions computed by the family of perceptrons with one hidden layer and infinitely smooth activation functions is dense in the space $L_p(\mu)$ of functions with the p-th power of absolute value finitely integrable with respect to a finite measure μ, in the space $C(X)$ of functions continuous on a compact X, and in Sobolev spaces generalizing $L_p(\mu)$ to functions that are differentiable up to a given order. In the application domain of catalytic materials, from which the case study presented in Section 4 is taken, nearly all examples of regression analysis published since mid 1990s rely on multilayer feed-forward neural networks, typically on multilayer perceptrons (Figure 1). In the last edition of "Handbook of heterogeneous catalysis", more than 20 such examples are listed, as well as several additional, based on other kinds of such networks – radial basis function networks and piecewise-linear neural networks [16]. Therefore, these three kinds of neural networks are now briefly recalled:

- *Multilayer perceptrons (MLPs)* can have an arbitrary number of hidden layers, and the basis functions of their linear space of computed functions are constructed by means of sigmoidal activation functions, such as logistic sigmoid, hyperbolic tangent, or arctangent [13, 43].
- *Radial basis function (RBF) networks* always have only one hidden layer, and the basis functions of their space of computed functions are radial, i.e., the function value depends only on the distance of the vector of input values from some centre, specific to the function [7].
- *Piecewise-linear neural networks* are simply MLPs with piecewise-linear activation functions. Their linear space of computed functions is dense only in $C(X)$, but on the other hand, they allow a straightforward extraction of logical rules describing the relationships between input and output values of the network [15].

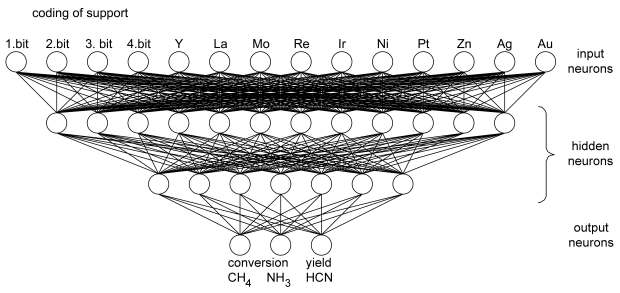


Fig. 1. Example MLP architecture with two hidden layers, used in the case study presented in Section 4.

  - **(ii)** *Support vector regression* based on positive semi-definite kernels [34, 36]. It is worth mentioning that they generalize the above recalled RBF networks, and also the historically first kind of nonlinear regression – polynomial regression.
  - **(iii)** *Gaussian process regression* [28] is listed here also due to a relationship to radial basis function networks, but most importantly due to the fact that it has already been successfully employed in surrogate modelling [6].
2. *Nonparametric regression models* are, in general, more flexible than parametric models, but the flexibility is typically paid for by more extensive computations. Therefore, their importance has been increasing only during the last two decades, following the increasing power of available computers [12, 14]. Nevertheless, there is one noteworthy exception:
  - **(v)** *Regression trees* have been successfully used already since the early 1980s [4]. They are actually a modification of a classification method, therefore the regression function is piecewise-constant. That property accounts for relatively low computational requirements of regression trees, but also decreases their flexibility, otherwise the main advantage of nonparametric methods.

3 Boosting regression models

Boosting is a method of improving classification accuracy that consists in developing the classifier iteratively, and increasing the relative influence of the training data that most contributed to errors in the previous iterations on its development in the subsequent iterations [33]. The usefulness of boosting for classification has incited its extension to regression [8]. Both for classification and for regression, the basic approach to increasing the relative influence of particular

training data is re-sampling the training data according to a distribution that gives them a higher probability of occurrence. This is equivalent to re-weighting the contributions of the individual training pairs (x_j, y_j), with higher weights corresponding to higher values of the error measure.

Since surrogate models are regression models, any method for regression boosting (such as [8, 10, 35]) is suitable for them. In the following, the method *AdaBoost.R2* will be explained in detail, proposed in [8].

Similarly to other adaptive boosting methods, each of the available pairs $(x_1, y_1), \ldots \ldots, (x_p, y_p)$ of input and output data is in the first iteration of AdaBoost.R2 used exactly once. This corresponds to re-sampling them according to the uniform probability distribution P_1 with $P_1(x_1) = \frac{1}{p}$ for $j = 1, \ldots, p$. In addition, the weighted average error of the 1st iteration is set to zero, $\bar{E}_1 = 0$.

In the subsequent iterations ($i \geq 2$), the following sequence of steps is performed:

1. A sample $(\xi_1, \eta_1), \ldots, (\xi_p, \eta_p)$ is obtained through re-sampling $(x_1, y_1), \ldots, (x_p, y_p)$ according to the distribution P_{i-1}.
2. Using $(\xi_1, \eta_1), \ldots, (\xi_p, \eta_p)$ as training data, a regression model F_i is constructed.
3. A [0,1]-valued squared error vector E_i of F_i with respect to $(x_1, y_1), \ldots, (x_p, y_p)$ is calculated as

$$E_i = (E_i(1), \ldots, E_i(p)) = \\ = \frac{((F_i(x_1) - y_1)^2, \ldots, (F_i(x_p) - y_p)^2)}{\max_{k=1,\ldots,p}(F_i(x_k) - y_k)^2}. \quad (1)$$

4. The weighted average error of the i-th iteration is calculated as

$$\bar{E}_i = \frac{1}{p} \sum_{k=1}^{p} P_i(x_k, y_k) E_i(k). \quad (2)$$

5. Provided $\bar{E}_i < 0.5$, the probability distribution for re-sampling $(x_1, y_1), \ldots, (x_p, y_p)$ is for $k = 1, \ldots, p$ updated according to

$$P_i(x_k, y_k) = \\ = \frac{P_{i-1}(x_k, y_k) \left(\frac{\bar{E}_i}{1-\bar{E}_i}\right)^{(1-E_i(k))}}{\sum_{i=1}^{p} P_{i-1}(x_k, y_k) \left(\frac{\bar{E}_i}{1-\bar{E}_i}\right)^{(1-E_i(k))}}. \quad (3)$$

6. The *boosting approximation* in the i-th iteration is set to the median of the approximations $F_1, \ldots, F_i$ with respect to the probability distribution

$$\left(\frac{\bar{E}_1}{1-\bar{E}_1}, \ldots, \frac{\bar{E}_i}{1-\bar{E}_i}\right). \quad (4)$$

The errors used to asses the quality of the boosting approximation are then called *boosting errors*, e.g., *boosting MSE*, or *boosting MAE*, where MSE refers to the mean squared error between the computed and measured values, whereas MAE refers to the mean absolute error, i.e., to the mean Euclidean distance between them. For simplicity, also the approximation in the first iteration, F_1, is called boosting approximation if boosting is performed, and the respective errors are then called boosting errors, although boosting actually does not introduce any modifications in the first iteration.

The above formulation of the method deals only with the case $\bar{E}_i < 0.5$. For $\bar{E}_i \geq 0.5$, the original formulation of the method in [8] proposes to stop the boosting. However, that is not allowed if the stopping criterion should be based on an independent set of validation data. Indeed, the calculation of $\bar{E}_i$ does not rely on any such independent data set, but it relies solely on the data employed to construct the regression model. A possible alternative for the case $\bar{E}_i \geq 0.5$ is reinitialization, i.e., proceeding as in the 1st iteration [1].

In connection with using feed-forward neural networks as surrogate models, it is important to be aware of the difference between the iterations of boosting and the iterations of neural network training. Boosting iterates on a higher level, one iteration of boosting includes a complete training of an ANN, which can proceed for many hundreds of iterations. Nevertheless, both kinds of iterations are similar in the sense that starting with some iteration, over-training is present. Therefore, also over-training due to boosting can be reduced through *stopping* in the iteration after which the *error for an independent set of data first time increases*. Moreover, *cross-validation* can be used to find the iteration most appropriate for stopping.

4 Case study in materials science

The extension of surrogate modelling with boosting will now be illustrated on a case study using data from the investigation of *catalytic materials for the high-temperature synthesis of hydrocyanic acid*. That investigation and its results have been recently described in [24]. It has been performed through *high-throughput experiments* in a circular 48-channel reactor. In most of those experiments, the composition of the materials was designed by means of a *genetic algorithm* developed *specifically for heterogeneous catalysis* [44]. More precisely, the algorithm was running for 7 generations of population size 92, and in addition 52 other catalysts with manually designed composition were investigated. Consequently, data about altogether 696 catalytic materials were gathered.

The composition and preparation of the investigated catalytic materials and the conditions in which they had been tested have been in detail described in [24]. Here, only the independent and dependent variables are recalled, the latter corresponding to the considered possible objective functions:

- *independent variables*: material used as support, and proportions of the 10 metal additives Y, La, Mo, Re, Ir, Ni, Pt, Zn, Ag, Au (an 11th metal, Zr, was left out due to the fact that the proportions of all active compounds sum up to 100 %);
- *dependent variables*, i.e., objective functions: conversions of CH_4 and NH_3, and yield of HCN.

As the surrogate model, MLPs were employed, in accordance with their leading role among nonlinear regression models in the area of catalytic materials [2, 16]. Each considered neural network had *14 input neurons*: 4 of them coding the material used as support, the other 10 corresponding to the proportions of the 10 metal additives belonging to independent variables; output neurons were 3, corresponding to the possible objective functions (Figure 1).

The most appropriate MLP architectures were searched by means of cross-validation, using only data about catalysts from the 1.–6. generation of the genetic algorithm and about the 52 catalysts with manually designed composition, thus altogether data about 604 catalytic materials. Data about catalysts from the 7. generation were completely excluded and left out for validating the search results. To use as much information as possible from the available data, cross-validation was applied as the extreme 604-fold variant, i.e., leave-1-out validation. The set of architectures within which the search was performed was delimited by means of the heuristic *pyramidal condition*: the number of neurons in a subsequent layer must not increase the number of neurons in a previous layer. Denote n_I, n_h and n_O the numbers of input, hidden and output neurons, respectively, and n_{H1} and n_{H2} the numbers of neurons in the first and second hidden layer, respectively. Then the pyramidal condition reads:

(i) for MLPs with 1 hidden layer: $n_I \geq n_H \geq n_O$, in our case $14 \geq n_H \geq 3$ (12 architectures);

(ii) for MLPs with 2 hidden layers: $n_I \geq n_{H1} \geq n_{H2} \geq n_O$, in our case $14 \geq n_{H1} \geq n_{H2} \geq 3$ (78 architectures).

To investigate the usefulness of boosting in our case study, the same data were used and the same set of architectures was considered as for architecture search. In each iteration, a leave-1-out validation was performed, in the way briefly outlined in the preceding section: The mean squared error of the performance of the catalytic materials serving in the individual folds as test data was calculated, and averaged over all the 604 folds. The *criterion* according to which *boosting is considered useful* to an architecture was: the average boosting MSE in the 2nd iteration has to be lower than in the 1st iteration. The iteration till which the average boosting MSE continuously decreased was then taken as the *final iteration of boosting*.

According to that criterion, boosting was useful to 9 from the 12 considered architectures with one hidden layer and to 65 from the 78 considered architectures with two hidden layers. To validate the most promising results of the investigation of the usefulness of boosting in our case study, the data from the 7th generation of the genetic algorithm were used. The validation included the *5 architectures* that were most promising for boosting from the point of view of the *lowest boosting MSE on test data in the final iteration*. These were the architectures (14,10,6,3), (14,14,8,3), (14,13,5,3), (14,10,4,3) and (14,11,3), for which the final iterations of boosting were 32, 29, 31, 19 and 3, respectively. For each of them, the validation proceeded as follows:

1. In each iteration up to the final boosting iteration corresponding to the respective architecture, a single MLP was trained with data about the 604 catalytic materials considered during the architecture search.
2. Each of those MLPs was employed to approximate the conversions of CH_4 and NH_3 and the yield of HCN for the 92 materials from the 7. generation of the genetic algorithm.
3. In each iteration, the medians with respect to the probability distribution (4) of the approximations of the two conversions and of the HCN yield obtained up to that iteration were used as the boosting approximations.
4. From the conversions and the yield predicted by the boosting approximations, and from the measured values, the boosting MSE and MAE were calculated for each MLP.

The boosting errors (MSE and MAE) are summarized in Figure 2, whereas Figure 3 compares the boosting approximations of the conversions of CH_4 and NH_3 and of the yield of HCN in the 1st and final iteration with their measured values. The presented results clearly confirm the usefulness of boosting for the five considered architectures. For each of them, boosting led to an overall decrease of both considered error measures, the MSE and MAE, on new data from the 7th generation of the genetic algorithm. Moreover, the decrease of the MSE (which is the measure employed during the investigation of the usefulness of boosting) is uninterrupted or nearly uninterrupted till the final boosting iteration. On the other hand, the

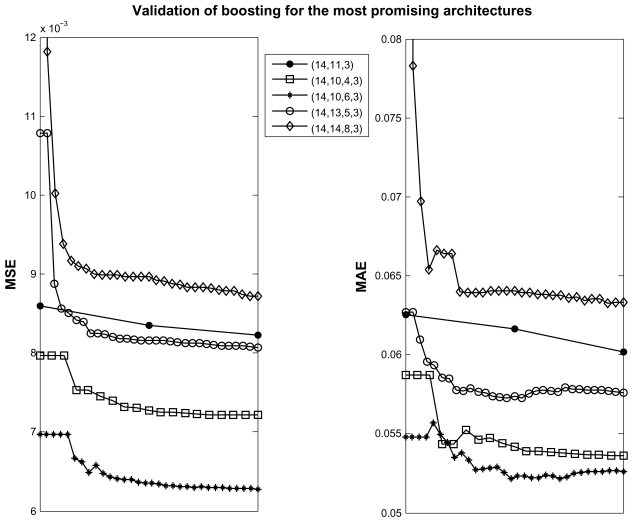


Fig. 2. History of the boosting MSE and MAE on the data from the 7th generation of the genetic algorithm for MLPs with the five architectures included in the validation of boosting.

scatter plots in Figure 3 do not indicate any apparent difference between the effect of boosting on the three properties employed as catalyst performance measures in our case study – conversion of CH_4, conversion of NH_3, and yield of HCN. Hence, the performed validation confirms the usefulness of boosting irrespectively of which of those performance measures is considered.

5 Conclusions

The paper dealt with surrogate modelling, a modern approach to the optimization of empirical objective functions, which is particularly attractive in evolutionary optimization. It proposed to extend surrogate modelling with regression boosting, to increase the accuracy of surrogate models, thus also the agreement between results of surrogate modelling and results of the intended optimization of the original objective function. Needless to say, regression boosting is not new, though it is less common than the popular classification boosting. However, novel is its combination with surrogate models, which adds the advantage of increased accuracy to the main advantage of surrogate modelling – decreasing the time and costs of optimization of empirical objective functions.

Theoretical principles of both surrogate modelling and boosting are known, therefore the main purpose of the reported research was to validate the feasibility of the proposed extension of surrogate modelling on several sufficiently complex case studies, one of which was sketched in this paper. The presented case study results clearly confirm the usefulness of boosting. For the five most promising architectures, boosting leads to an overall decrease of both considered error measures, MSE and MAE, on new data from the 7th generation of the genetic algorithm. Moreover, the decrease of the MSE (which is the boosting error employed during the investigation of the usefulness of boosting) is uninterrupted or nearly uninterrupted till the final boosting iteration. On the other hand, the scatter plots in Figure 3 do not indicate any apparent difference between the effect of boosting on the three catalyst properties considered as possible objective functions in our case study – conversion of CH_4, conversion of NH_3, and yield of HCN. Hence, the performed validation confirms the usefulness of boosting irrespectively of which of these objective functions is selected.

References

1. H. Altinçay: *Optimal resampling and classifier prototype selection in classifier ensembles using genetic algorithms.* Pattern Analysis and Applications 7, 2004, 285–295.
2. M. Baerns and M. Holeňa: *Combinatorial Development of Solid Catalytic Materials. Design of High-Throughput Experiments, Data Analysis, Data Mining.* World Scientific, Singapore, 2009.
3. T. Bartz-Beielstein: *Experimental Research in Evolutionary Computation.* Springer Verlag, Berlin, 2006.
4. L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone: *Classification and Regression Trees.* Wadsworth, Belmont, 1984.
5. A.J. Brooker, J. Dennis, P.D. Frank, D.B. Serafini, V. Torczon, and M. Trosset: *A rigorous framework for optimization by surrogates.* Structural and Multidisciplinary Optimization, 17, 1998, 1–13.
6. D. Büche, N.N. Schraudolph, and P. Koumoutsakos: *Accelerating evolutionary algorithms with gaussian process fitness function models.* IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 35, 2005, 183–194.
7. M.D. Buhmann: *Radial Basis Functions: Theory and Implementations.* Cambridge University Press, Cambridge, 2003.
8. H. Drucker: *Improving regression using boosting techniques.* In A.J.C. Sharkey, editor, Proceedings of the 14th International Conference on Machine Learning, Springer Verlag, London, 1997, 107–115.
9. A. Eftaxias, J. Font, A. Fortuny, J. Giralt, A. Fabregat, and F. Stber: *Kinetic modelling of catalytic wet air oxidation of phenol by simulated annealing.* Applied Catalysis B: Environmental 33, 2001, 175–190.
10. J. Friedman: *Greedy function approximation: A gradient boosting machine.* Annals of Statistics 29, 2001, 1189–1232.
11. D. Goldberg: *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, 1989.

12. L. Györfi, M. Kohler, A. Krzyzak, and H. Walk: *A Distribution-Free Theory of Nonparametric Regression.* Springer Verlag, Berlin, 2002.
13. M.T. Hagan, H.B. Demuth, and M.H. Beale: *Neural Network Design.* PWS Publishing, Boston, 1996.
14. T.J. Hastie and R.J. Tibshirani: *Generalized Additive Models.* Chapman & Hall, Boca Raton, 1990.
15. M. Holeňa: *Piecewise-linear neural networks and their relationship to rule extraction from data.* Neural Computation 18, 2006, 2813–2853.
16. M. Holeňa and M. Baerns: *Computer-aided strategies for catalyst development.* In G. Ertl, H. Knözinger, F. Schüth, and J. Eitkamp, editors, Handbook of Heterogeneous Catalysis, Wiley-VCH, Weinheim, 2008.
17. A. Holzwarth, P. Denton, H. Zanthoff, and C. Mirodatos: *Combinatorial approaches to heterogeneous catalysis: Strategies and perspectives for academic research.* Catalysis Today 67, 2001, 309–318.
18. K. Hornik: *Approximation capabilities of multilayer neural networks.* Neural Networks 4, 1991, 251–257.
19. Y. Jin: *A comprehensive survery of fitness approximation in evolutionary computation.* Soft Computing 9, 2005, 3–12.
20. Y. Jin, M. Hüsken, M. Olhofer, and B. Sendhoff: *Neural networks for fitness approximation in evolutionary optimization.* In Y. Jin, editor, Knowledge Incorporation in Evolutionary Computation, Springer Verlag, Berlin, 2005, 281–306.
21. P.C. Kainen, V. Kůrková, and M. Sanguineti: *Estimates of approximation rates by gaussian radial-basis functions.* In Adaptive and Natural Computing Algorithms, Springer Verlag, Berlin, 2007, 11–18.
22. B. Li, P. Sun, Q. Jin, J. Wang, and D. Ding: *A simulated annealing study of Si, Al distribution in the omega framework.* Journal of Molecular Catalysis A: Chemical 148, 1999, 189–195.
23. A.S. McLeod and L.F. Gladden: *Heterogeneous catalyst design using stochastic optimization algorithms.* m Journal of Chemical Information and Computer Science 40, 2000, 981–987.
24. S. Möhmel, N. Steinfeldt, S. Endgelschalt, M. Holeňa, S. Kolf, U. Dingerdissen, D. Wolf, R. Weber, and M. Bewersdorf: *New catalytic materials for the high-temperature synthesis of hydrocyanic acid from methane and ammonia by high-throughput approach.* Applied Catalysis A: General 334, 2008, 73–83.
25. Y.S. Ong, P.B. Nair, A.J. Keane, and K.W. Wong: *Surrogate-assisted evolutionary optimization frameworks for high-fidelity engineering design problems.* In Y. Jin, editor, Knowledge Incorporation in Evolutionary Computation, Springer Verlag, Berlin, 2005, 307–331.
26. A. Pinkus: *Approximation theory of the MPL model in neural networks.* Acta Numerica 8, 1998, 277–283.
27. K. Rasheed, X. Ni, and S. Vattam: *Methods for using surrogate modesl to speed up genetic algorithm oprimization: Informed operators and genetic engineering.* In Y. Jin, editor, Knowledge Incorporation in Evolutionary Computation, Springer Verlag, Berlin, 2005, 103–123.
28. E. Rasmussen and C. Williams: *Gaussian Process for Machine Learning.* MIT Press, Cambridge, 2006.
29. A. Ratle: *Accelerating the convergence of evolutionary algorithms by fitness landscape approximation.* In A.E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, Parallel Problem Solving from Nature, Springer Verlag, Berlin, 1998, 87–96.
30. A. Ratle: *Kriging as a surrogate fitness landscape in evolutionary optimization.* Artificial Intelligence for Engineering Design, Analysis and Manufacturing 15, 2001, 37–49.
31. C.R. Reeves and J.E. Rowe: *Genetic Algorithms: Principles and Perspectives.* Kluwer Academic Publishers, Boston, 2003.
32. R. Schaefer: *Foundation of Global Genetic Optimization.* Springer Verlag, Berlin, 2007.
33. R. Schapire: *The strength of weak learnability.* Machine Learning 5, 1990, 197–227.
34. B. Schölkopf and A.J. Smola: *Learning with Kernels.* MIT Press, Cambridge, 2002.
35. D.L. Shrestha: *Experiments with AdaBoost.RT, an improved boosting scheme for regression.* Neural Computation 18, 2006, 1678–1710.
36. I. Steinwart and A. Christmann: *Support Vector Machines.* Springer Verlag, New York, 2008.
37. A. Tompos, J.L. Margitfalvi, E. Tfirst, L. Végvári, M.A. Jaloull, H.A. Khalfalla, and M.M. Elgarni: *Development of catalyst libraries for total oxidation of methane: A case study for combined application of "holographic research strategy and artificial neural networks" in catalyst library design.* Applied Catalysis A: General 285, 2005, 65–78.
38. A. Tompos, L. Vgvári, E. Tfirst, and J.L. Margitfalvi: *Assessment of predictive ability of artificial neural networks using holographic mapping.* Combinatorial Chemistry and High Throughput Screening 10, 2007, 121–134.
39. H. Ulmer, F. Streichert, and A. Zell: *Model-assisted steady state evolution strategies.* In GECCO 2003: Genetic and Evolutionary Computation, Springer Verlag, Berlin, 2003, 610–621.
40. H. Ulmer, F. Streichert, and A. Zell: *Model assisted evolution strategies.* In Y. Jin, editor, Knowledge Incorporation in Evolutionary Computation, Springer Verlag, Berlin, 2005, 333–355.
41. L. Végvári, A. Tompos, S. Göbölös, and J.F. Margitfalvi: *Holographic research strategy for catalyst library design: Description of a new powerful optimisation method.* Catalysis Today 81, 2003, 517–527.
42. M.D. Vose: *The Simple Genetic Algorithm. Foundations and Theory.* MIT Press, Cambridge, 1999.
43. H. White: *Artificial Neural Networks: Approximation and Learning Theory.* Blackwell Publishers, Cambridge, 1992.
44. D. Wolf, O.V. Buyevskaya, and M. Baerns: *An evolutionary approach in the combinatorial selection and optimization of catalytic materials.* Applied Catalyst A: General 200, 2000, 63–77.

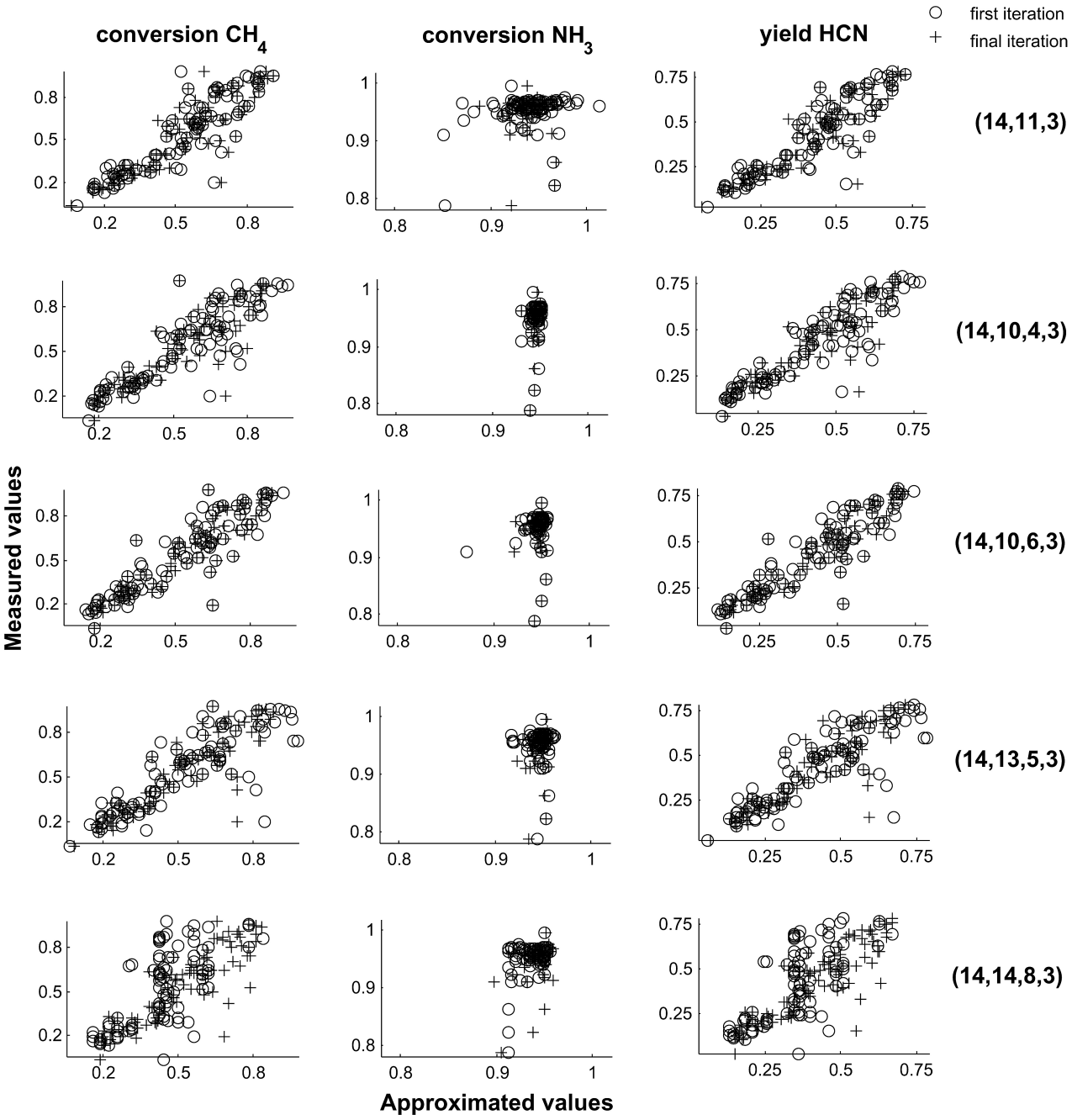


Fig. 3. Comparison of the boosting approximations of the conversions of CH_4 and NH_3 and of the yield of HCN in the 1st and final iteration with their measured values for the 92 catalytic materials from the 7th generation of the genetic algorithm.

Local safety of an ontology*

Lukáš Homoľa[1] and Július Štuller[2]

[1] Faculty of Nuclear Sciences and Physical Engineering, Czech Technical University
lukashomola@hotmail.com
[2] Institute of Computer Science, Academy of Sciences of the Czech Republic
stuller@cs.cas.cz

Abstract. *The ability to import ontologies safely, that is, without changing the original meaning of their terms, has been identified as crucial for the collaborative development and the reuse of (OWL) ontologies.*
In this paper, we propose the notion of local safety of an ontology and we identify scenarios in which this notion may be useful in guiding the development of an ontology that is to import other ontologies safely.

1 Introduction

Defined as explicit specifications of conceptualizations of a domain of knowledge (or of a discourse) [1], ontologies are (virtually) always manifestations of a shared understanding of a domain. They typically take the form of a formal (e.g., logical) theory that fixes the vocabulary of a domain and, through constraining possible interpretations and well-formed use of the vocabulary terms, provides meaning for the vocabulary.

Ontologies have been advocated as a tool to support human communication, knowledge sharing and reuse, and interoperability between distributed systems. As such, ontologies have a range of applications in fields like knowledge management, information retrieval and integration, cooperative information systems, bioinformatics, medicine, linguistics, e-commerce, etc. Today, they are perhaps best known as the key technology of the Semantic Web vision.

The construction of a typical ontology is a collaborative process that involves *direct cooperation* among multiple individuals or groups of ontology engineers and domain experts (sometimes from different domains of expertise and different organizations) and/or *indirect cooperation* through the *reuse* of previously published, autonomously developed ontologies.

Most often, each team participating in the development of an ontology focuses on a part of it (a "component ontology") that pertains to the team's domain of expertise/authority and cooperates with other teams to relate the part it is working on with other parts. Performing an upgrade of even only one such a component ontology may require the participation of all the teams as different component ontologies are, when combined together, interrelated, depend on and affect one another (changing one component ontology may thus necessitate changes to the others and might require teams to reconcile their changes).

By reusing an ontology we mean using it as an input to develop a new ontology. In such a process, significant parts of the reused ontology are often extracted, refined, extended or otherwise adapted and then combined with other ontologies to form the final assembly.

One of the prerequisites for efficient collaborative ontology construction and maintenance is the ability to combine ontologies in a controlled way. The interaction among component ontologies should be controlled and well-understood in order to reduce the communication that is needed among different teams and to avoid expensive reconciliation processes. Ideally, controlled interaction should allow different teams to develop, test and upgrade their ontologies independently, to replace a component ontology or extend an ontology with minimal side effects. The issue is also vital for ontology reuse, especially in the case when the reused ontology, rather than being adapted and used as a draft to develop an ontology component, is linked to and remains under the control of its original developers, who may perform changes to it autonomously.

2 Problem definition

The Web Ontology Language (OWL) [2], a widely accepted W3C recommendation for creating and sharing ontologies on the Web, provides only very limited support for combining ontologies.

OWL adopts an importing mechanism, implemented by the `owl:imports`[3] construct, which allows one to include in an OWL ontology all the statements

* The work was supported by the project No. 1M0554 "Advanced Remedial Processes and Technologies" of the Ministry of Education, Youth and Sports of the Czech Republic and partly by the Institutional Research Plan AV0Z10300504 "Computer Science for the Information Society: Models, Algorithms, Applications". The first author also acknowledges the financial support of the Department of Mathematics at his institution.

[3] `http://www.w3.org/2002/07/owl#imports`, to be precise

contained in some other OWL ontology. In the importing ontology, there is no logical difference between the statements that are imported and the proper ones.

A number of recent papers by Grau et al. [3–7] stressed the particular relevance of the ability to import OWL ontologies "safely", that is, in such a way that the imported terms (the terms of the imported ontologies) preserve their original meaning (the meaning these terms have in the imported ontologies) in the importing ontology. This ability is applicable in typical scenarios of OWL ontology development such as in the following one (see the above-mentioned works by Grau et al. for a motivating example):

- an ontology engineer distinguishes between the so-called *external terms* and the so-called *local terms* of the ontology $\mathcal{O}$ he or she is developing;
- the local terms are those whose meaning is assumed to be fully described in the ontology $\mathcal{O}$ itself, possibly with the help of the remaining, external terms;
- the meaning of the external terms is assumed to be only partially described in the ontology $\mathcal{O}$ – in terms of their use in the description of the local terms – and to be further described in some other ontologies (preexistent or concurrently developed) that are to be imported into $\mathcal{O}$;
- the use of the external terms in the statements of the ontology $\mathcal{O}$ is expected not to alter the original meaning these terms have in the ontologies to be imported.

In the paper, we continue in the study, initiated by Grau at al., of the methodology for OWL ontology development in the scenario given above. We propose the notion of *local safety of an ontology* and discuss under which conditions and how this notion can be used to guide the development of OWL ontologies.

3 Preliminaries

In this section we introduce description logics (DLs) [8], a family of logic-based knowledge representation formalisms, which underly modern ontology languages such as OWL. OWL consists of three (sub)languages of increasing expressive power, two of which, namely OWL Lite and OWL DL, roughly correspond to the DLs $\mathcal{SHIF}$ and $\mathcal{SHOIN}$, respectively.

DLs view the world as being populated by objects and allow one to represent the relevant notions of the domain of interest in terms of *concepts*, *roles* and (possibly) *individuals*, representing sets of elements, binary relationships between elements and single elements, respectively. Starting from atomic concepts, atomic roles and individuals, which are denoted simply by a name, complex concepts and complex roles are built using concept and role *constructors*. We assume the sets $\mathbf{C}$, $\mathbf{R}$ and $\mathbf{I}$ of (respectively) *atomic concepts*, *atomic roles* and *individuals* to be countably infinite and mutually disjoint and to be fixed for every DL. An ontology $\mathcal{O}$ formalized in a DL takes the form of a finite set of terminological and role *axioms*, which are used to suitably organize and interrelate multiple concept and role descriptions. DLs are distinguished by constructors and/or types of axioms they provide. We will use the term $\mathcal{L}$-axiom ($\mathcal{L}$-ontology) to emphasize we are talking about an axiom (an ontology) in the DL $\mathcal{L}$.

In the abstract notation we will use the letters A, B to denote atomic concepts, r, s to denote atomic roles, and a, b for individuals (all the letters possibly with a subscript). The letters C, D will be used to denote a concept (atomic or complex), R, S to denote a role, and α, β to denote an axiom.

As the *minimal DLs* of practical interest are usually considered the DLs $\mathcal{EL}$ and $\mathcal{AL}$, which both are fragments of the smallest propositionally closed DL $\mathcal{ALC}$. In $\mathcal{ALC}$, concepts are composed inductively according to the following syntax rule:

$$
\begin{aligned}
C, D \rightarrow\ & A \,(\text{atomic concept}) \mid \\
& \bot \,(\text{bottom concept}) \mid \top \,(\text{top concept}) \mid \\
& \neg C \,(\text{concept negation}) \mid \\
& C \sqcap D \,(\text{conjunction}) \mid C \sqcup D \,(\text{disjunction}) \mid \\
& \exists R.C \,(\text{existential restriction}) \mid \\
& \forall R.C \,(\text{value restriction}).
\end{aligned}
$$

Valid constructs for $\mathcal{EL}$ are: $\bot$, $C \sqcap D$ and $\exists R.C$. In $\mathcal{AL}$, the syntax of complex concepts is the following: $\bot$, $\top$, $\neg A$ (atomic concept negation), $C \sqcap D$, $\exists R.\top$ (limited existential restriction), and $\forall R.C$.

DLs $\mathcal{EL}$, $\mathcal{AL}$ and $\mathcal{ALC}$ provide no role constructors.

The listed $\mathcal{ALC}$ constructors are not all independent ($\top = \neg\bot$, $C \sqcup D = \neg(\neg C \sqcap \neg D)$, $\forall R.C = \neg(\exists R.\neg C)$). In fact, $\mathcal{ALC}$ can be obtained from both $\mathcal{EL}$ and $\mathcal{AL}$ by adding the concept negation constructor.

A *terminological axiom* in $\mathcal{EL}$, $\mathcal{AL}$ and $\mathcal{ALC}$ is an expression of the following forms: $A \equiv C$ (*concept definition*), $A \sqsubseteq C$ (*concept specialization*) or $C \sqsubseteq D$ (*general concept inclusion*, GCI). The abbreviation of the form $C \equiv D$ (*concepts equality*) stands for the two GCIs $C \sqsubseteq D$ and $D \sqsubseteq C$. $\mathcal{EL}$, $\mathcal{AL}$ and $\mathcal{ALC}$ provide no role axioms.

$\mathcal{S}$ is an extension of $\mathcal{ALC}$ in which an atomic role can be declared transitive using the role axiom of the form $\mathsf{Trans}(r)$.

Further extensions of DLs are indicated by appending letters to the DL's name. Advanced concept constructors include *number restrictions* of the form

$\geq nR$ (indicated by appending the letter $\mathcal{N}$), *qualified number restrictions* $\geq nR.C$ (appending $\mathcal{Q}$) and *nominals* $\{a\}$ (appending $\mathcal{O}$). In the case of number restrictions and qualified number restrictions, the dual constructors $\leq nR$ and $\leq nR.C$ are introduced as abbreviations for $\neg(\geq n+1R)$ and $\neg(\geq n+1R.C)$, respectively. Nominals allows to construct a concept representing a singleton set containing one individual. Enumeration $\{a_1,\ldots,a_n\}$ is an abbreviation for $\{a_1\}\sqcup\ldots\sqcup\{a_n\}$.

Yet other extensions include role constructors, of which the *inverse role* constructor r^- (appending $\mathcal{I}$) is the most prominent one. Another important type of role axioms is the *role inclusion* $R\sqsubseteq S$ (appending $\mathcal{H}$).

These extensions can be used in different combinations, for example $\mathcal{ALN}$ is an extension of $\mathcal{AL}$ with number restrictions, and $\mathcal{SHOIN}$ is the DL that uses 5 of the constructors we have presented.

The semantics of DLs is defined via interpretations. An *interpretation* $\mathcal{I}$ is a pair $\mathcal{I}=(\Delta^\mathcal{I},.^\mathcal{I})$, where $\Delta^\mathcal{I}$ is a non-empty set, called the *domain* of the interpretation, and $.^\mathcal{I}$ is the *interpretation function*, which maps atomic concepts to subsets of $\Delta^\mathcal{I}$, atomic roles to binary relations over $\Delta^\mathcal{I}$ and individuals to elements of $\Delta^\mathcal{I}$. The interpretation function extends to complex concepts as follows:

$$\begin{aligned}
\bot^\mathcal{I} &= \emptyset, \top^\mathcal{I} = \Delta^\mathcal{I},\\
(C\sqcap D)^\mathcal{I} &= C^\mathcal{I}\cap D^\mathcal{I}, (C\sqcup D)^\mathcal{I} = C^\mathcal{I}\cup D^\mathcal{I},\\
(\neg C)^\mathcal{I} &= \Delta^\mathcal{I} - C^\mathcal{I},\\
(\forall R.C)^\mathcal{I} &= \{x\in\Delta^\mathcal{I};\ \forall y\,((x,y)\in R^\mathcal{I}\rightarrow y\in C^\mathcal{I})\},\\
(\exists R.C)^\mathcal{I} &= \{x\in\Delta^\mathcal{I};\ \exists y\,((x,y)\in R^\mathcal{I}\wedge y\in C^\mathcal{I})\},\\
(\geq nR)^\mathcal{I} &= \{x\in\Delta^\mathcal{I};\ |\{y\in\Delta^\mathcal{I};\ (x,y)\in R^\mathcal{I}\}|\geq n\},\\
(\geq nR.C)^\mathcal{I} &= \{x\in\Delta^\mathcal{I};\\
&\qquad |\{y\in\Delta^\mathcal{I};\ (x,y)\in R^\mathcal{I}\wedge y\in C^\mathcal{I}\}|\geq n\},\\
\{a\}^\mathcal{I} &= \{a^\mathcal{I}\},\\
(r^-)^\mathcal{I} &= \{(x,y);\ (y,x)\in r^\mathcal{I}\}.
\end{aligned}$$

The semantics of terminological axioms is defined in terms of a satisfaction relation $\models$, which relates interpretations to the terminological axioms they satisfy. It is defined as follows: $\mathcal{I}\models C\sqsubseteq D$ iff $C^\mathcal{I}\subseteq D^\mathcal{I}$, $\mathcal{I}\models R\sqsubseteq S$ iff $R^\mathcal{I}\subseteq S^\mathcal{I}$, $\mathcal{I}\models C(a)$ iff $a^\mathcal{I}\in C^\mathcal{I}$, $\mathcal{I}\models R(a,b)$ iff $(a^\mathcal{I},b^\mathcal{I})\in R^\mathcal{I}$, $\mathcal{I}\models\mathsf{Trans}(r)$ iff the relation $r^\mathcal{I}$ is transitive. Interpretations satisfying an axiom are said to be its models.

An interpretation $\mathcal{I}$ is a *model* of an ontology $\mathcal{O}$ (written $\mathcal{I}\models\mathcal{O}$) iff $\mathcal{I}\models\alpha$ for all $\alpha\in\mathcal{O}$. An ontology is said to be *consistent* if it has at least one model and is said to be *inconsistent* otherwise.

An ontology $\mathcal{O}$ *entails* an axiom α (written $\mathcal{O}\models\alpha$) iff all models of $\mathcal{O}$ satisfy α, especially we will speak about *subsumption* between C and D in the case of $\mathcal{O}\models C\sqsubseteq D$, and *satisfiability of concept* C in the case $\mathcal{O}\not\models C\sqsubseteq\bot$.

Interpretations $\mathcal{I}$ and $\mathcal{J}$ are *isomorphic* (written $\mathcal{I}\cong\mathcal{J}$) iff there is a bijection $\mu:\Delta^\mathcal{I}\rightarrow\Delta^\mathcal{J}$ such that for every $x,y\in\Delta^\mathcal{I}, A\in\mathbf{C}, r\in\mathbf{R}, a\in\mathbf{I}$ the following holds: $x\in A^\mathcal{I}$ iff $\mu(x)\in A^\mathcal{J}$, $(x,y)\in r^\mathcal{I}$ iff $(\mu(x),\mu(y))\in r^\mathcal{J}$, $x=a^\mathcal{I}$ iff $\mu(x)=a^\mathcal{J}$. Isomorphic interpretations are semantically indistinguishable (in particular, they satisfy the same axioms).

A *signature* $\mathbf{S}$ is a finite subset of $\mathbf{C}\cup\mathbf{R}\cup\mathbf{I}$. Two interpretations $\mathcal{I}$ and $\mathcal{J}$ *coincide* on a signature $\mathbf{S}$ (written $\mathcal{I}|_\mathbf{S}=\mathcal{J}|_\mathbf{S}$) iff $\Delta^\mathcal{I}=\Delta^\mathcal{J}$ and $X^\mathcal{I}=X^\mathcal{J}$ holds for all $X\in\mathbf{S}$.

We say that $\mathcal{I}$ has been obtained from $\mathcal{J}$ through a *domain expansion* with the set Δ (such $\mathcal{I}$ will by denoted by $\mathcal{J}_{\cup\Delta}$) iff Δ is a non-empty set disjoint with $\Delta^\mathcal{J}$, $\Delta^\mathcal{I}=\Delta^\mathcal{J}\cup\Delta$, and $X^\mathcal{I}=X^\mathcal{J}$ holds for all $X\in\mathbf{C}\cup\mathbf{R}\cup\mathbf{I}$. Note that $\mathcal{J}_{\cup\Delta}$ and $\mathcal{J}$ only differ in that the domain of $\mathcal{J}$ is a proper subset of the domain of $\mathcal{J}_{\cup\Delta}$ (with Δ being the set of additional domain elements).

A DL $\mathcal{L}$ is said to have the *finite model property* (FMP) iff every consistent $\mathcal{L}$-ontology admits a model that is finite (i.e., with a finite domain). One of the most prominent DLs that exhibit the FMP is $\mathcal{SHOQ}$, while $\mathcal{SHIN}$ is an example of a DL that lacks the FMP. For a DL $\mathcal{L}$ with the FMP, $\mathcal{L}$-ontology $\mathcal{O}$ and $\mathcal{L}$-axiom α the following holds: $\mathcal{O}\models\alpha$ iff $\mathcal{I}\models\alpha$ for all finite models $\mathcal{I}\models\mathcal{O}$.

We say that an interpretation $\mathcal{K}$ is a *disjoint union* of interpretations $\mathcal{I}$ and $\mathcal{J}$ (written $\mathcal{K}=\mathcal{I}\uplus\mathcal{J}$) iff there exist some interpretations $\tilde{\mathcal{I}}$ and $\tilde{\mathcal{J}}$ satisfying $\tilde{\mathcal{I}}\cong\mathcal{I}$, $\tilde{\mathcal{J}}\cong\mathcal{J}$ and $\Delta^{\tilde{\mathcal{I}}}\cap\Delta^{\tilde{\mathcal{J}}}=\emptyset$ for which the following holds: $\Delta^\mathcal{K}=\Delta^{\tilde{\mathcal{I}}}\cup\Delta^{\tilde{\mathcal{J}}}$, $X^\mathcal{K}=X^{\tilde{\mathcal{I}}}\cup X^{\tilde{\mathcal{J}}}$ for all $X\in\mathbf{C}\cup\mathbf{R}$ and $a^\mathcal{K}=a^{\tilde{\mathcal{I}}}$ for all $a\in\mathbf{I}$. Intuitively, the interpretation $\mathcal{K}$ for which $\mathcal{K}=\mathcal{I}\uplus\mathcal{J}$ holds is composed of two unrelated parts one being isomorphic to $\mathcal{I}$ and the other to $\mathcal{J}$. Disjoint union of a set of interpretations is defined analogously.

A DL $\mathcal{L}$ is said to have the *disjoint union model property* (DUMP) iff the set of models of arbitrary $\mathcal{L}$-ontology is closed under disjoint unions.

A prominent example of a DL that enjoys the DUMP is $\mathcal{SHIQ}$. DLs that support nominals do not have this property.

In the subsequent sections, will use $\mathbf{C}(\alpha)$ to denote the set of all atomic concepts that occur in the axiom α (the sets $\mathbf{R}(\alpha)$ and $\mathbf{I}(\alpha)$ are defined analogously). We will use $\mathbf{Sig}(\alpha)$ as a shorthand for $\mathbf{C}(\alpha)\cup\mathbf{R}(\alpha)\cup\mathbf{I}(\alpha)$. $\mathbf{C}(\mathcal{O})$ will stand for $\bigcup_{\alpha\in\mathcal{O}}\mathbf{C}(\alpha)$ (the sets $\mathbf{R}(\mathcal{O})$, $\mathbf{I}(\mathcal{O})$ and $\mathbf{Sig}(\mathcal{O})$ are defined analogously).

4 Related work

In the papers by Grau et al. [3–7], safety of ontology import is formulated using the notion of conservative extension, in the context of ontologies first used in [9] and recently further studied in [10, 11].

Definition 1 (Conservative extension). *Let $\mathcal{L}$ be a DL, $\mathcal{O}_1$ and $\mathcal{O}_2$ two ontologies such that $\mathcal{O}_1 \subseteq \mathcal{O}_2$.*

We say that $\mathcal{O}_2$ is a deductive conservative extension of $\mathcal{O}_1$ w.r.t. $\mathcal{L}$, if for every $\mathcal{L}$-axiom α with $\mathbf{Sig}(\alpha) \subseteq \mathbf{Sig}(\mathcal{O}_1)$, we have $\mathcal{O}_2 \models \alpha$ iff $\mathcal{O}_1 \models \alpha$.

An ontology $\mathcal{O}$ into which an ontology $\mathcal{O}'$ can be safely imported is said to be *safe for* $\mathcal{O}'$.

Definition 2 (Safety for an ontology). *Let $\mathcal{L}$ be a DL, $\mathcal{O}$ and $\mathcal{O}'$ two ontologies.*

We say that $\mathcal{O}$ is safe for $\mathcal{O}'$ w.r.t. $\mathcal{L}$, if $\mathcal{O} \cup \mathcal{O}'$ is a conservative extension of $\mathcal{O}'$ w.r.t. $\mathcal{L}$.

Ghilardi at al. [12] studied novel DL reasoning services aimed at supporting developers in customizing their ontology to be safe for a particular ontology.

As regards the scenario we are concerned with, Grau et al. [3–7] argues that in practice it is often convenient, or even necessary, for the developers of an ontology $\mathcal{O}$ to abstract from particular ontologies that are to be imported into it and focus instead only on $\mathcal{O}$ and on its external terms:

- ontologies to be imported might not be available during the development of $\mathcal{O}$ (as it is in the case when these ontologies are developed concurrently with $\mathcal{O}$);
- the developers of $\mathcal{O}$ are usually not willing to commit to particular versions of the ontologies they intend to import (the development of a typical ontology is a never-finished process);
- at a later time, the developers might find ontologies other than those initially considered more suitable for providing the meaning of the external terms of $\mathcal{O}$.

Grau et al. proposed the following condition to be used to guide the development of an ontology $\mathcal{O}$ in such cases.

Definition 3 (Safety for a signature). *Let $\mathcal{L}$ be a DL, $\mathcal{O}$ an ontology and $\mathbf{S}$ a signature.*

We say that $\mathcal{O}$ is safe for $\mathbf{S}$ w.r.t. $\mathcal{L}$, if for every $\mathcal{L}$-ontology $\mathcal{O}'$ such that $\mathbf{Sig}(\mathcal{O}) \cap \mathbf{Sig}(\mathcal{O}') \subseteq \mathbf{S}$, $\mathcal{O}$ is safe for $\mathcal{O}'$ w.r.t. $\mathcal{L}$.

Once an ontology $\mathcal{O}$ is safe for the signature $\mathbf{S}$ (which is presumably the set of its external terms) w.r.t. $\mathcal{L}$, one can safely import into $\mathcal{O}$ any ontology $\mathcal{O}'$ written in $\mathcal{L}$ and sharing only terms from $\mathbf{S}$ with $\mathcal{O}$.

As Grau et al. showed, even the problem of checking whether an ontology consisting of a single $\mathcal{ALC}$ axiom is safe for a signature w.r.t. $\mathcal{ALCO}$ is undecidable. It is not yet known whether the safety for a signature is decidable for weaker DLs, such as $\mathcal{EL}$, or for more expressive DLs. Grau et al. proposed several safety classes of ontologies, parametrized by a signature $\mathbf{S}$ and representing sufficient conditions for safety for $\mathbf{S}$, that are decidable and can be checked syntactically in polynomial time.

Several extensions to OWL have been proposed to better support collaborative ontology development and ontology reuse, including P-OWL [13], C-OWL [14], the extension based on $\mathcal{E}$-connections [15] and the extension based on the so-called semantic import [16]. All such extension are, however, still subjects of research and are not included in the current candidate recommendation for OWL 2 [17], an ongoing extension to and revision of OWL.

5 Local safety of an ontology

The notion of safety for a signature, along with the corresponding safety classes, facilitates the construction of an ontology that is safe for any ontology (in a given DL) with which it shares only some prearranged set of terms.

In the scenario we are interested in here, however, an ontology engineer does not always need to have the ontology $\mathcal{O}$ safe for every possible ontology (every possible set of axioms in a certain DL), but often only needs to have it safe for a certain, conveniently chosen class of *candidate* ontologies. This is the case, for instance, when the scenario applies to collaborative ontology development and $\mathcal{O}$ is considered as a component ontology for a larger ontology developed distributively as a set of ontologies importing one another. The development of component ontologies in such a case is typically coordinated to some extent (e.g., some principles on which individual component ontologies will be build are resolved beforehand and made explicit) and the developers can make assumptions about some qualities and characteristics of the ontologies they import (as well as about the way these ontologies may further evolve).

5.1 Local ontologies

Ontologies, like other engineering artifacts, are designed. When we choose how to represent something in an ontology[4], we are making design decisions. The

[4] "There is no one correct way to model a domain – there are always viable alternatives." [18]

best solution to ontology design depends on a number of factors, of which the most important include the intended use of an ontology, and the anticipated extensions and refinements to it.

Generally accepted and widely cited are the five design criteria Gruber [19] proposed for ontologies whose purpose is knowledge sharing and interoperation among programs. They include the following criterion:

> *An ontology should offer a conceptual foundation for a range of anticipated tasks, and the representation should be crafted so that one can extend and specialize the ontology monotonically. Especially, one should be able to define new terms for special uses based on the existing vocabulary, in a way that does not require the revision of the existing definitions.*

To facilitate the design, deployment and reuse of ontologies, Guarino [20] suggested the development of different kinds of ontologies with different levels of generality and dependence on a particular domain, task or point of view, namely top-level ontologies, domain and task ontologies and application ontologies. Terms of ontologies on a lower level are, in some sense, held to be specializations of terms of ontologies on a level above. Top-level ontologies, which describe concepts independent of a particular problem or domain (such as space, time, object, event, action, etc.) are meant to be unifying for a large group of ontologies on lower levels.

Swartout et al. [21] proposed a number of desiderata aimed primarily at domain, task and application ontologies. They include the two following:

> *An ontology should be extensible. [. . .] Extension should be possible both at a low level, by adding domain-specific subconcepts, or at high level by adding intermediate or upper level concepts that cover new areas.*
>
> *Ontologies should not be "stovepipes." The derisive term "stovepipe system" is used to describe a system that may be vertically integrated but cannot be integrated horizontally with other systems.*

Here we propose a notion of restricting the meaning of the top concept, which, as we believe, provide a partial characterization of ontologies violating the aforementioned criteria.

Definition 4 (Restricting the meaning of the top concept). *We say that the ontology $\mathcal{O}$ restricts the meaning of the top concept, if there are atomic concepts $A_1, \ldots, A_n$, atomic roles $r_1, \ldots, r_m, s_1, \ldots, s_k$ and individuals $a_1, \ldots, a_l$ in* $\mathbf{Sig}(\mathcal{O})$ *such that:*

$$\mathcal{O} \models \top \sqsubseteq A_1 \sqcup \ldots \sqcup A_n \sqcup \exists r_1.\top \sqcup \ldots \sqcup \exists r_m.\top \sqcup \sqcup \exists s_1^-.\top \sqcup \ldots \sqcup \exists s_k^-.\top \sqcup \{a_1, \ldots, a_l\}.$$

Intuitively, an ontology restricts the meaning of the top concept if it introduces its vocabulary in such a way that the vocabulary can only be further specialized but not otherwise monotonically extended. In the case of domain, task and application ontologies at least, such an ontology can be considered badly-designed:

- it can not be, without previous modification, extended to cover a broader subject area than it already does,
- it is unsuitable for importing into any ontology that touches, even marginally, a subject area disjoint with that already covered by it.

As regards top-level ontologies, we studied the Basic Formal Ontology[5] and also the design of several other top-level ontologies [22], and came to the conclusion that even in this case the developers prefer not to restrict the meaning of the top concept. The only exception we found is the top-level ontology[6] proposed by John Sowa.

The notion of restricting the meaning of the top concept is closely related to the notion of localness of an ontology studied (also under the name safety of an ontology) by Grau et al. [23–26].

Definition 5 (Localness). *An ontology $\mathcal{O}$ is local if the set of its models is closed under domain expansion (i.e., if $\mathcal{I} \models \mathcal{O}$ implies $\mathcal{I}_{\cup\Delta} \models \mathcal{O}$ for every interpretation $\mathcal{I}$ and every non-empty set Δ disjoint with $\Delta^{\mathcal{I}}$).*

In [23], a syntactic characterization of localness for $\mathcal{SHOIQ}$ ontologies is given, which allows one to check localness of an $\mathcal{SHOIQ}$ ontology in polynomial time. We used this characterization[7] to show that $\mathcal{SHOIQ}$ ontologies restricting the meaning of the top concept are exactly those that are not local.

Proposition 1. *A $\mathcal{SHOIQ}$ ontology restricts the meaning of the top concept iff it is not local.*

Grau et al. [25] also reported testing over 700 ontologies available on the Web for localness and finding more than 99% of them local. However, we could not trace any further details on their experimental results.

5.2 Local safety

Regarding the development of an ontology in our scenario, the considerations above suggest that it is often possible to consider only local ontologies as the candidates for importing.

[5] `http://www.ifomis.org/bfo`

[6] `http://www.jfsowa.com/ontology/toplevel.htm`

[7] Relevant points of the characterization are reproduced at the beginning of the Appendix.

Definition 6 (Local safety for a signature). *Let $\mathcal{L}$ be a DL, $\mathbf{S}$ a signature and $\mathcal{O}$ an ontology.*

We say that $\mathcal{O}$ is locally safe for $\mathbf{S}$ w.r.t. $\mathcal{L}$, if for every local $\mathcal{L}$-ontology $\mathcal{O}'$ with $\mathbf{Sig}(\mathcal{O}) \cap \mathbf{Sig}(\mathcal{O}') \subseteq \mathbf{S}$, $\mathcal{O}$ is safe for $\mathcal{O}'$ w.r.t. $\mathcal{L}$.

The following proposition provides sufficient condition for local safety w.r.t. $\mathcal{SHOIQ}$.

Proposition 2. *Let $\mathbf{S}$ be a signature and $\mathcal{O}$ an ontology such that for every interpretation $\mathcal{J}$ there exists a model $\mathcal{I}$ of $\mathcal{O}$ such that*

- *$\Delta^{\mathcal{I}} = \Delta^{\mathcal{J}} \cup \Delta$ for some Δ, $\Delta \cap \Delta^{\mathcal{J}} = \emptyset$,*
- *$X^{\mathcal{I}} = X^{\mathcal{J}}$ for all $X \in \mathbf{S}$.*

Then $\mathcal{O}$ is locally safe for $\mathbf{S}$ w.r.t. $\mathcal{SHOIQ}$.

The following example demonstrates that, in comparison with safety condition proposed by Grau et al., the condition of local safety is less restrictive and may allow for a more convenient use of the external terms.

Example 1. Let us consider building an OWL ontology intended to provide a reference terminology for the annotation of films. Let us assume we intend to safely import into our ontology $\mathcal{O}$ some well-designed (and thus local) ontology that defines the categorization of films by genre. Suppose that the atomic concept Film is expected to be the only term our ontology will share with the imported ontology. Whereas the ontology

$$\mathcal{O} = \{\mathsf{Director} \sqsubseteq \neg\mathsf{Film}, \mathsf{Film} \sqsubseteq \exists\mathsf{hasDirector}.\mathsf{Director}\}$$

is not safe for $\{\mathsf{Film}\}$ even w.r.t. $\mathcal{AL}$ (take the non-local ontology $\mathcal{O}' = \{\top \sqsubseteq \mathsf{Film}\}$ as a counterexample), it is, according to Proposition 2, locally safe for $\{\mathsf{Film}\}$ w.r.t. $\mathcal{SHOIQ}$.

When we are concerned with local safety w.r.t. $\mathcal{SHOQ}$ (which has the FMP) we can use the following sufficient condition.

Proposition 3. *Let $\mathbf{S}$ be a signature and $\mathcal{O}$ an ontology such that for every finite interpretation $\mathcal{J}$ there exists a model $\mathcal{I}$ of $\mathcal{O}$ such that*

- *$\Delta^{\mathcal{I}} = \Delta^{\mathcal{J}} \cup \Delta$ for some Δ, $\Delta \cap \Delta^{\mathcal{J}} = \emptyset$,*
- *$X^{\mathcal{I}} = X^{\mathcal{J}}$ for all $X \in \mathbf{S}$.*

Then $\mathcal{O}$ is locally safe for $\mathbf{S}$ w.r.t. $\mathcal{SHOQ}$.

The two following propositions give a recipe for deciding local safety of an ontology written in $\mathcal{SHIQ}$ (which has the DUMP) w.r.t. $\mathcal{SHOQ}$ in the case when all external terms are atomic concepts.

Proposition 4. *Let $\mathbf{S}$ be a signature such that $\mathbf{S} \subseteq \mathbf{C}$, $\mathcal{O}$ a $\mathcal{SHIQ}$ ontology and a an individual. Assume that for every subset $\tilde{\mathbf{S}} \subseteq \mathbf{S}$ the ontology*

$$\mathcal{O} \cup \{A \equiv \{a\}\}_{A \in \tilde{\mathbf{S}}} \cup \{A \equiv \bot\}_{A \in \mathbf{S} - \tilde{\mathbf{S}}}$$

is consistent.

Then $\mathcal{O}$ is locally safe for $\mathbf{S}$ w.r.t. $\mathcal{SHOQ}$.

Proposition 5. *Let $\mathbf{S}$ be a signature such that $\mathbf{S} \subseteq \mathbf{C}$, $\mathcal{O}$ a $\mathcal{SHIQ}$ ontology and a an individual. Assume that there exists a subset $\tilde{\mathbf{S}} \subseteq \mathbf{S}$ such that the ontology*

$$\mathcal{O} \cup \{A \equiv \{a\}\}_{A \in \tilde{\mathbf{S}}} \cup \{A \equiv \bot\}_{A \in \mathbf{S} - \tilde{\mathbf{S}}}$$

is inconsistent.

Then $\mathcal{O}$ is not locally safe for $\mathbf{S}$ w.r.t. $\mathcal{ALO}$ ($\mathcal{ELO}$).

Corollary 1. *Let $\mathbf{S}$ be a signature such that $\mathbf{S} \subseteq \mathbf{C}$, $\mathcal{O}$ a $\mathcal{SHIQ}$ ontology.*

Then $\mathcal{O}$ is locally safe for $\mathbf{S}$ w.r.t. $\mathcal{SHOQ}$ iff $\mathcal{O}$ is locally safe for $\mathbf{S}$ w.r.t. $\mathcal{ALO}$ ($\mathcal{ELO}$).

Corollary 2. *Let $\mathcal{L}$ be a DL that is in between $\mathcal{ALO}$ ($\mathcal{ELO}$) and $\mathcal{SHOQ}$.*

The problem of deciding whether a $\mathcal{SHIQ}$ ontology is locally safe for a signature $\mathbf{S}$, $\mathbf{S} \subseteq \mathbf{C}$, w.r.t. $\mathcal{L}$ is reducible to the problem of checking consistency of a finite set of $\mathcal{SHOIQ}$ ontologies, and thus decidable.

Corollary 3. *Let $\mathcal{O}$ be a $\mathcal{SHIQ}$ ontology locally safe for $\mathbf{S} \subseteq \mathbf{C}$ w.r.t. $\mathcal{SHOQ}$, $\mathcal{O}'$ a local $\mathcal{SHOQ}$ ontology such that $\mathbf{Sig}(\mathcal{O}) \cap \mathbf{Sig}(\mathcal{O}') \subseteq \mathbf{S}$. Then $\mathcal{O} \cup \mathcal{O}'$ is locally safe for $\mathbf{S} \setminus \mathbf{Sig}(\mathcal{O}')$ w.r.t. $\mathcal{SHOQ}$.*

6 Conclusion and outlook

This paper contributes to the framework for ontology development presented by Grau et al. We proposed the notion of local safety of an ontology and showed its applicability in the development of real-world ontologies. We showed that local safety for a signature consisting solely of atomic concepts is decidable for an interesting group of description logics.

For the future work, we would like to study decidability and computational properties of (sufficient conditions for) local safety for a signature that contains atomic roles as well. The results obtained in the paper are also directly applicable to the problem of extracting reusable ontology parts, or ontology modules, as conceived by Grau et al. in the cited works.

References

1. T.R. Gruber: *A translation approach to portable ontology specifications.* Knowledge Acquisition, 5, 2, 1993, 199–220.
2. P. Patel-Schneider, P. Hayes, and I. Horrocks: *OWL Web Ontology Language: Semantics and Abstract Syntax.* W3C recommendation, W3C, 2004. Available at `http://www.w3.org/TR/owl-semantics/`.
3. B.C. Grau, I. Horrocks, Y. Kazakov, and U. Sattler: *Ontology Reuse: Better Safe than Sorry.* In Description Logics, Bozen/Bolzano University Press, 2007, 41–52.

4. B.C. Grau, I. Horrocks, Y. Kazakov, and U. Sattler: *Just the right amount: extracting modules from ontologies.* In Proc. of the 16th Int. World Wide Web Conf., 2007, 717–727.
5. B.C. Grau, I. Horrocks, Y. Kazakov, and U. Sattler: *A logical framework for modularity of ontologies.* In Proc. of the 20th Int. Joint Conf. on Artificial Intelligence, 2007.
6. B.C. Grau, I. Horrocks, Y. Kazakov, and U. Sattler: *Modular reuse of ontologies: theory and practice.* Journal of Artificial Intelligence Research, 31, 2008, 273–318.
7. E. Jimenez-Ruiz, B.C. Grau, T. Schneider, U. Sattler, and R. Berlanga: *Safe and economic re-use of ontologies: a logic-based methodology and tool support.* In Proc. of the 5th European Semantic Web Conf., Springer LNCS, 2008.
8. F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, and P.F. Patel-Schneider, editors: *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2003.
9. G. Antoniou and A. Kehagas: *A note on the the refinement of ontologies. Int. Journal of Intelligent Systems*, 15, 7, 2000, 623–632.
10. C. Lutz and F. Wolter: *Conservative extensions in the lightweight description logic EL.* In Proc. of the 21th Conf. on Automated Deduction, Springer-Verlag, 2007, volume 4603, 84–99.
11. C. Lutz, D. Walther, and F. Wolter: *Conservative extensions in expressive description logics.* In M. Veloso, editor, Proc. of the 20th Int. Joint Conf. on Artificial Intelligence, AAAI Press, 2007, 453–458.
12. S. Ghilardi, C. Lutz, and F. Wolter: *Did I damage my ontology? A case for conservative extensions in description logics.* In P. Doherty, J. Mylopoulos, and C. Welty, editors, Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning, AAAI Press, 2006, 187–197.
13. J. Bao and V. Honavar: *Ontology language extensions to support localized semantics, modular reasoning, and collaborative ontology design and ontology reuse.* Technical report, Iowa State University, 2004.
14. P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, and H. Stuckenschmidt: *C-OWL: Contextualizing Ontologies.* In Proc. of the Second Int. Semantic Web Conf., Springer, 2003, 164–179.
15. B.C. Grau, B. Parsia, and E. Sirin: *Combining OWL ontologies using $\mathcal{E}$-Connections.* Web Semantics: Science, Services and Agents on the World Wide Web, 4, 1, 2006, 40–59.
16. J.Z. Pan, L. Serafini, and Y. Zhao: *Semantic import: an approach for partial ontology reuse.* In P. Haase, V. Honavar, O. Kutz, Y. Sure, and A. Tamilin, editors, Proc. of the 1st Int. Workshop on Modular Ontologies, volume 232, CEUR-WS.org, 2006.
17. M. Boris, B.C. Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz: *OWL 2 Web Ontology Language: Profiles.* W3C draft, W3C, 2009. Available at `http://www.w3.org/TR/2009/CR-owl2-profiles-20090611/`.
18. N.F. Noy and D.L. Mcguinness: *Ontology development 101: a guide to creating your first ontology.* Technical report, Stanford Knowledge Systems Laboratory, 2001. Available at `http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html`.
19. T.R. Gruber: *Toward principles for the design of ontologies used for knowledge sharing* . Int. Journal of Human-Computer Studies, 43, 5-6, 1995, 907–928.
20. N. Guarino: *Formal ontology and information systems.* In N. Guarino, editor, Proc. of the 1st Int. Conf. on Formal Ontologies in Information Systems, IOS Press, 1998, 3–15.
21. B. Swartout, R. Patil, K. Knight, and T. Russ: Toward distributed use of large-scale ontologies. In Proc. of the 10th Knowledge Acquisition for Knowledge-Based Systems Workshop, 1996.
22. N.F. Noy and C.D. Hafner: *The state of the art in ontology design: A survey and comparative review. AI Magazine*, 18, 1997, 53–74.
23. B.C. Grau, B. Parsia, E. Sirin, and A. Kalyanpur: *Modularity and web ontologies.* In P. Doherty, J. Mylopoulos, and C. A. Welty, editors, Proc. the 20th Int. Conf. on Principles of Knowledge Representation and Reasoning, AAAI Press, 2006, 198–209.
24. B.C. Grau, B. Parsia, E. Sirin, and A. Kalyanpur: *Automatic partitioning of OWL ontologies using $\mathcal{E}$-connections.* In Proc. of the 18th Int. Workshop on Description Logics, 2005.
25. B.C. Grau, I. Horrocks, O. Kutz, and U. Sattler: *Will my ontologies fit together?* In Proc. of the 19th Int. Workshop on Description Logics, 2006.
26. B.C. Grau and O. Kutz: *Modular ontology languages revisited.* In Workshop on Semantic Web for Collaborative Knowledge Acquisition at the 20th Int. Joint Conf. on Artificial Intelligence, 2007.

7 Appendix

As shown in [23]:

- for each $\mathcal{SHOIQ}$ concept C, one of the following holds:
  - $C^{\mathcal{I}_{\cup\Delta}} = C^{\mathcal{I}}$ for all $\mathcal{I}$ and Δ, $\Delta \cap \Delta^{\mathcal{I}} = \emptyset$ (such C is said to be *local*);
  - $C^{\mathcal{I}_{\cup\Delta}} = C^{\mathcal{I}} \cup \Delta$ for all $\mathcal{I}$ and Δ, $\Delta \cap \Delta^{\mathcal{I}} = \emptyset$ (C is *non-local*).
- If R is a $\mathcal{SHOIQ}$ role, then $R^{\mathcal{I}_{\cup\Delta}} = R^{\mathcal{I}}$ for all $\mathcal{I}$ and Δ, $\Delta \cap \Delta^{\mathcal{I}} = \emptyset$.
- A $\mathcal{SHOIQ}$ ontology is not local iff it explicitly contain a GCI $D \sqsubseteq C$ such that C is local and D is non-local.

Lemma 1 (Auxiliary). *Let α be a $\mathcal{SHOIQ}$ axiom, $\mathcal{I}$ an interpretation and Δ a set disjoint with $\Delta^{\mathcal{I}}$. Then $\mathcal{I} \not\models \alpha$ implies $\mathcal{I}_{\cup\Delta} \not\models \alpha$.*

Proof. Suppose that $\mathcal{I} \not\models \alpha$.
If α is of the form $C \sqsubseteq D$ that means $C^{\mathcal{I}} \not\subseteq D^{\mathcal{I}}$ ($\star$). For C, D local, we have $C^{\mathcal{I}_{\cup\Delta}} = C^{\mathcal{I}}$ and $D^{\mathcal{I}_{\cup\Delta}} = D^{\mathcal{I}}$. By

$(\star)$, we get $C^{\mathcal{I}_{\cup\Delta}} \not\subseteq D^{\mathcal{I}_{\cup\Delta}}$. For C local, D non-local, we have $C^{\mathcal{I}_{\cup\Delta}} = C^{\mathcal{I}}$ and $D^{\mathcal{I}_{\cup\Delta}} = D^{\mathcal{I}} \cup \Delta$. By $(\star)$ and by the fact that $\Delta \cap C^{\mathcal{I}} = \emptyset$ (because $\Delta \cap \Delta^{\mathcal{I}} = \emptyset$), we get $C^{\mathcal{I}_{\cup\Delta}} \not\subseteq D^{\mathcal{I}_{\cup\Delta}}$. For C non-local, D local, we have $C^{\mathcal{I}_{\cup\Delta}} = C^{\mathcal{I}} \cup \Delta$ and $D^{\mathcal{I}_{\cup\Delta}} = D^{\mathcal{I}}$. By $(\star)$, we get $C^{\mathcal{I}_{\cup\Delta}} \not\subseteq D^{\mathcal{I}_{\cup\Delta}}$. For C, D are non-local, we have $C^{\mathcal{I}_{\cup\Delta}} = C^{\mathcal{I}} \cup \Delta$ and $D^{\mathcal{I}_{\cup\Delta}} = D^{\mathcal{I}} \cup \Delta$. By $(\star)$ and by the fact that $\Delta \cap C^{\mathcal{I}} = \emptyset$, $\Delta \cap D^{\mathcal{I}} = \emptyset$ (because $\Delta \cap \Delta^{\mathcal{I}} = \emptyset$), we get $C^{\mathcal{I}_{\cup\Delta}} \not\subseteq D^{\mathcal{I}_{\cup\Delta}}$. For each of the four possible cases we showed that $\mathcal{I}_{\cup\Delta} \not\models C \sqsubseteq D$.

The remaining types of $\mathcal{SHOIQ}$ axioms ($C \equiv D$, $\mathsf{Trans}(r)$, $R \sqsubseteq S$) can be treated in the same way. □

Proof (of Proposition 1.). The proposition is obviously true for inconsistent ontologies.
Assume that a consistent ontology $\mathcal{O}$ restricts the meaning of the top concept. Then, by Definition 4, $\mathcal{O} \models \top \sqsubseteq C$ for some C of the form $A_1 \sqcup \ldots \sqcup A_n \sqcup \exists r_1.\top \sqcup \ldots \sqcup \exists r_m.\top \sqcup \exists s_1^-.\top \sqcup \ldots \sqcup \exists s_k^-.\top \sqcup \{a_1, \ldots, a_l\}$. Take any model $\mathcal{I}$ of $\mathcal{O}$ and any $x \notin \Delta^{\mathcal{I}}$. As $\top^{\mathcal{I}_{\cup\{x\}}} = \Delta^{\mathcal{I}} \cup \{x\}$ and $C^{\mathcal{I}_{\cup\{x\}}} = \Delta^{\mathcal{I}}$, $\mathcal{I}_{\cup\{x\}} \not\models \top \sqsubseteq C$, and thus $\mathcal{I}$ is not a model of $\mathcal{O}$. This shows $\mathcal{O}$ is not local.

Assume a consistent $\mathcal{SHOIQ}$ ontology $\mathcal{O}$ does not restrict the meaning of the top concept. Then, by Definition 4, $\mathcal{O} \not\models \top \sqsubseteq C$ for C of the form $A_1 \sqcup \ldots \sqcup A_n \sqcup \exists r_1.\top \sqcup \ldots \sqcup \exists r_m.\top \sqcup \exists r_1^-.\top \sqcup \ldots \sqcup \exists r_m^-.\top \sqcup \{a_1, \ldots, a_l\}$, where $A_1, \ldots, A_n$, $r_1, \ldots, r_m$ and $a_1, \ldots, a_l$ are all atomic concepts, atomic roles and individuals in $\mathbf{Sig}(\mathcal{O})$. Since $\mathcal{O} \not\models \top \sqsubseteq C$, there exists a model $\mathcal{I}$ of $\mathcal{O}$ such that $\mathcal{I} \not\models \top \sqsubseteq C$. Pick any such model $\mathcal{I}$. Since $\mathcal{I} \not\models \top \sqsubseteq C$, there exists an object $x \in \Delta^{\mathcal{I}}$ such that x do not participate in the interpretation $X^{\mathcal{I}}$ of any atomic concept, atomic role and individual X in $\mathbf{Sig}(\mathcal{O})$. Observe that: for all local $\mathcal{SHOIQ}$ concepts C_1 composed of the symbols from $\mathbf{Sig}(\mathcal{O})$, $x \notin C_1^{\mathcal{I}}$ holds; for all non-local $\mathcal{SHOIQ}$ concepts C_2 composed of the symbols from $\mathbf{Sig}(\mathcal{O})$, $x \in C_2^{\mathcal{I}}$ holds. Therefore, $\mathcal{O}$ does not contain a GCI of the form $C_2 \sqsubseteq C_1$ (otherwise $\mathcal{I}$ were not its model) and thus is local. □

Proof (of Proposition 2.). Let $\mathcal{O}'$ be an arbitrary local $\mathcal{SHOIQ}$ ontology with $\mathbf{Sig}(\mathcal{O}) \cap \mathbf{Sig}(\mathcal{O}') \subseteq \mathbf{S}$. We need to show that $\mathcal{O} \cup \mathcal{O}'$ is a conservative extension of $\mathcal{O}'$ w.r.t. $\mathcal{SHOIQ}$.

Assume (for contradiction) that there exists a $\mathcal{SHOIQ}$ axiom α with $\mathbf{Sig}(\alpha) \subseteq \mathbf{Sig}(\mathcal{O}')$ for which both $\mathcal{O}' \not\models \alpha$ $(\star)$ and $\mathcal{O} \cup \mathcal{O}' \models \alpha$ $(*)$ hold.
By $(\star)$, there is a model $\mathcal{J}$ of $\mathcal{O}'$ such that $\mathcal{J} \not\models \alpha$. $(\diamond)$

The conditions of the proposition imply the existence of a model $\mathcal{I}$ of $\mathcal{O}$ such that $\Delta^{\mathcal{I}} = \Delta^{\mathcal{J}} \cup \Delta$, $\Delta \cap \Delta^{\mathcal{J}} = \emptyset$ and $X^{\mathcal{I}} = X^{\mathcal{J}}$ for all $X \in \mathbf{S}$. Pick any interpretation $\mathcal{K}$ satisfying: $\Delta^{\mathcal{K}} = \Delta^{\mathcal{I}}$, $X^{\mathcal{K}} = X^{\mathcal{I}}$ for all $X \in \mathbf{Sig}(\mathcal{O})$, $X^{\mathcal{K}} = X^{\mathcal{J}}$ for all $X \in \mathbf{Sig}(\mathcal{O}')$. Such an interpretation exists as $X^{\mathcal{I}} = X^{\mathcal{J}}$ for $X \in \mathbf{Sig}(\mathcal{O}) \cap \mathbf{Sig}(\mathcal{O}')$, and symbols not occurring in $\mathbf{Sig}(\mathcal{O}) \cap \mathbf{Sig}(\mathcal{O}')$ can be interpreted arbitrarily.

First, because $\mathcal{I} \models \mathcal{O}$ and $\mathcal{K}|_{\mathbf{Sig}(\mathcal{O})} = \mathcal{I}|_{\mathbf{Sig}(\mathcal{O})}$, $\mathcal{K} \models \mathcal{O}$ holds. Second, because $\mathcal{J} \models \mathcal{O}'$ and $\mathcal{O}'$ is local, we have $\mathcal{J}_{\cup\Delta} \models \mathcal{O}'$ and consequently, since $\mathcal{K}|_{\mathbf{Sig}(\mathcal{O}')} = \mathcal{J}_{\cup\Delta}|_{\mathbf{Sig}(\mathcal{O}')}$, $\mathcal{K} \models \mathcal{O}'$. Therefore $\mathcal{K} \models \mathcal{O} \cup \mathcal{O}'$.

Since $\mathcal{J} \not\models \alpha$, we have, using Lemma 1, that $\mathcal{J}_{\cup\Delta} \not\models \alpha$. Furthermore, since $\mathcal{K}|_{\mathbf{Sig}(\mathcal{O}')} = \mathcal{J}_{\cup\Delta}|_{\mathbf{Sig}(\mathcal{O}')}$ and $\mathbf{Sig}(\alpha) \subseteq \mathbf{Sig}(\mathcal{O}')$, $\mathcal{K} \not\models \alpha$.

We showed there exists a model of $\mathcal{O} \cup \mathcal{O}'$ that is not a model of α, which yields a contradiction with the assumption $(*)$. □

Proof (of Proposition 3.). Same proof as of Proposition 2 goes through - we only need to replace the sentence labeled with $(\diamond)$ with the following: Because $(\star)$ and because $\mathcal{SHOQ}$ has the FMP, there exists a finite model $\mathcal{J}$ of $\mathcal{O}'$ such that $\mathcal{J} \not\models \alpha$. □

Proof (of Proposition 4.). Let $\mathcal{J}$ be a finite interpretation.

Let us associate with every $x \in \Delta^{\mathcal{J}}$ ($\Delta^{\mathcal{J}}$ is finite) an unique $a_x \in \mathbf{I}$, $a_x \notin \mathbf{Sig}(\mathcal{O})$ (i.e., different elements are associated with different individuals). For every $x \in \Delta^{\mathcal{J}}$, let us set $\mathbf{S}_x = \{A \in \mathbf{S};\ x \in A^{\mathcal{J}}\}$.

The conditions of the proposition imply that for every $x \in \Delta^{\mathcal{J}}$ there exists a model $\mathcal{I}_x$ of

$$\mathcal{O} \cup \{A \equiv \{a_x\}\}_{A \in \mathbf{S}_x} \cup \{A \equiv \bot\}_{A \in \mathbf{S} - \mathbf{S}_x}.$$

Pick some interpretation $\tilde{\mathcal{I}}$ such that $\tilde{\mathcal{I}} = \biguplus_{x \in \Delta^{\mathcal{J}}} \mathcal{I}_x$ and some interpretation $\mathcal{I}$ isomorphic with $\tilde{\mathcal{I}}$ such that ${a_x}^{\mathcal{I}} = x$ for all $x \in \Delta^{\mathcal{J}}$ (to get such interpretation $\mathcal{I}$ it is enough to "rename" finitely many elements in $\Delta^{\tilde{\mathcal{I}}}$).

Since $\mathcal{I}_x \models \mathcal{O}$ for all $x \in \Delta^{\mathcal{J}}$, $\mathcal{O}$ is a $\mathcal{SHIQ}$ ontology, $\mathcal{SHIQ}$ has the DUMP, $\Delta^{\mathcal{J}}$ is finite, we have $\tilde{\mathcal{I}} \models \mathcal{O}$ and consequently, since $\mathcal{I} \cong \tilde{\mathcal{I}}$, $\mathcal{I} \models \mathcal{O}$ $(\star)$.

As ${a_x}^{\mathcal{I}} = x$ for all $x \in \Delta^{\mathcal{J}}$, we have $\Delta^{\mathcal{J}} \subseteq \Delta^{\mathcal{I}}$ $(*)$.

It is easy to see that for $A \in \mathbf{S}$ and $x \in \Delta^{\mathcal{J}}$ the following holds: $A^{\mathcal{I}_x} = \{{a_x}^{\mathcal{I}_x}\}$ if $x \in A^{\mathcal{J}}$; $A^{\mathcal{I}_x} = \emptyset$ if $x \notin A^{\mathcal{J}}$. Thus, for $A \in \mathbf{S}$ we have $A^{\tilde{\mathcal{I}}} = \bigcup_{x \in A^{\mathcal{J}}} \{{a_x}^{\tilde{\mathcal{I}}}\}$ and, consequently, $A^{\mathcal{I}} = \bigcup_{x \in A^{\mathcal{J}}} \{{a_x}^{\mathcal{I}}\} = \bigcup_{x \in A^{\mathcal{J}}} \{x\}$, and thus $A^{\mathcal{I}} = A^{\mathcal{J}}$ $(\diamond)$.

We showed that, for any finite interpretation $\mathcal{J}$, there exists an interpretation $\mathcal{I}$ satisfying $(\star, *, \diamond)$. Using Proposition 3 we have that $\mathcal{O}$ is locally safe for $\mathbf{S}$ w.r.t. $\mathcal{SHOQ}$. □

Proof (of Proposition 5.). Consider an $\mathcal{ALO}$ ($\mathcal{ELO}$) ontology $\mathcal{O}' = \{A \equiv \{a\}\}_{A \in \tilde{\mathbf{S}}} \cup \{A \equiv \bot\}_{A \in \mathbf{S} - \tilde{\mathbf{S}}}$, which evidently is local, satisfies $\mathbf{Sig}(\mathcal{O}) \cap \mathbf{Sig}(\mathcal{O}') \subseteq \mathbf{S}$ and is consistent ($\mathcal{O}' \not\models \top \sqsubseteq \bot$). The conditions of the proposition say that the ontology $\mathcal{O} \cup \mathcal{O}'$ is inconsistent ($\mathcal{O} \cup \mathcal{O}' \models \top \sqsubseteq \bot$). We showed that there exists a local $\mathcal{ALO}$ ($\mathcal{ELO}$) ontology $\mathcal{O}'$ satisfying $\mathbf{Sig}(\mathcal{O}) \cap \mathbf{Sig}(\mathcal{O}') \subseteq \mathbf{S}$ for which $\mathcal{O}$ is not safe. □

Statistical machine translation between related and unrelated languages*

David Kolovratník, Natalia Klyueva and Ondřej Bojar

Charles University in Prague, Faculty of Mathematics and Physics,
Institute of Formal and Applied Linguistics

Abstract. *In this paper we describe an attempt to compare how relatedness of languages can influence the performance of statistical machine translation (SMT). We apply the Moses toolkit on the Czech-English-Russian corpus UMC 0.1 in order to train two translation systems: Russian-Czech and English-Czech. The quality of the translation is evaluated on an independent test set of 1000 sentences parallel in all three languages using an automatic metric (BLEU score) as well as manual judgments. We examine whether the quality of Russian-Czech is better thanks to the relatedness of the languages and similar characteristics of word order and morphological richness. Additionally, we present and discuss the most frequent translation errors for both language pairs.*

1 Introduction

Statistical Machine Translation nowadays has become one of the easiest and cheapest paradigms of the MT systems. Researchers can now use various toolkits to experiment with different language pairs. We experiment with Moses [2], an open-source implementation of phrase-based statistical translation system.

For closely-related languages, statistical MT methods are sometimes believed to be unreasonably complicated. For example, in the project Česílko [3] – Machine Translation among Slavic languages – the main accent was put on the idea that the relatedness of the languages rather than statistics should be exploited. Česílko was initially a rule-based system, based on the direct word-for-word translation (for very closely related Czech and Slovak) and engaging a few syntactic transfer rules in case less related languages are concerned (Czech and Polish or Czech and Lithuanian).

In our experiments we try to compare if the relatedness has a positive effect when using phrase-based statistical models.

Our main hypothesis was that we should obtain better results in Russian-to-Czech translation than in English-to-Czech. We used the Moses toolkit in order to carry out the experiments and evaluation. Additionally, we applied factored models on the tagged version of the corpus and compared the outputs.

The paper is structured as follows. Section 2 and Section 3 provide a description of the data we used during the experiment and our tokenization and tagging tools. In Section 4 and Section 5 we briefly summarize the Moses toolkit and present our experiments with MT between English/Russian and Czech. In Section 6 we evaluate our MT output using an automatic and a few manual evaluation metrics. Finally, the paper is concluded by a discussion and plans of future work.

2 Data

Phrase-based SMT systems need huge amount of parallel data in order to extract dictionaries of phrases and their translations, so called phrase tables. The most reliable source of parallel data are books and their translations into different languages, still it seems to be very laborious to collect a big corpus based on books. Web pages can serve as a good and significantly cheaper source for parallel texts, although usually less reliable. Moreover, while for the wide-spread languages we can easily find them, for minority languages parallel texts may not be available on the web in sufficient quantities.

We carried our experiments using the Czech-English-Russian (cs-en-ru) corpus UMC 0.1 [1] with automatic pairwise sentence alignment containing texts from Project Syndicate[1]. Although we could have used additional data to train the translation model for Czech and English, we need English-Czech and Russian-Czech corpus to be comparable. Table 1 provides statistics of the data we used in our experiments.

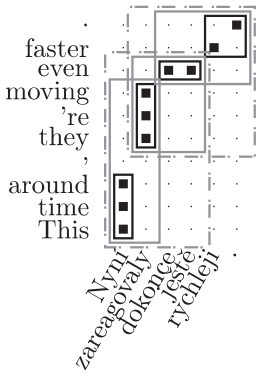
We had to collect the held-out and test set sentences ourselves for two reasons: first, we needed the sentences to be tri-parallel, that is parallel across the three languages, and second to be sure they do not overlap with the training data set. We also used Project Syndicate but extracted the test sets only from

* This work was supported by the Czech Science Foundation under the contract no. 201/09/H057, Grant Agency of Charles University under the contract no. 100008/2008, and the grants FP7-ICT-2007-3-231720 (EuroMatrix Plus), GAAV CR 1ET201120505 and MSM 0021620838.

[1] `http://www.project-syndicate.org/`

Cz: *prostě/prostě/Dg-------1A---- jsem/být/VB-S---1P-AA--- brala/brát/VpQW---XR-AA---*
Ru: *включая/включая/Sp-a президента/президент/Ncmsay мбеки/мбеки/Vmip3s-a-p*
En: *the/the/DT visionaries/visionary/NNS would/would/MD have/have/VH gotten/get/VVN nowhere/nowhere/RB*

Fig. 1. Example of a factored corpus. The sentences are not parallel.



This	=	nyní
time	=	nyní
around	=	nyní
they	=	zareagovaly
...	=	...
This time around	=	Nyní
they 're moving	=	zareagovaly
even	=	dokonce ještě
...	=	...
This time around, they 're moving	=	Nyní zareagovaly
even faster	=	dokonce ještě rychleji
...	=	...

Fig. 2. Simple phrase-based translation: Training sentences are automatically word-aligned and used to extract all phrases constistent with the word alignment (not all consistent phrases have been marked in the picture). The extracted dictionary of phrases is used in translation: the input sentence is segmented into known phrases, each phrase is translated and the output is constructed by concatenating translated phrases. Usually only little phrase-reordering is performed.

	Languages	Sentences
Language Model	cs	92,233
Translation Model	ru → cs	79,888
Translation Model	en → cs	76,588
Held-out	cs, en, ru	750
Test set	cs, en, ru	1,000

Table 1. Summary of corpus sizes.

newly published articles. The held-out and test set sentences have been added to the corpus UMC[2].

3 Data preprocessing

We used the tools developed under the UMC project, namely the trainable tokenizer for Czech, English and Russian languages. It was applied on the test and development set of data to make them consistent with training sets.

In order to train a factored model we tagged and lemmatized the UMC corpus with the help of TreeTagger [5] for English and Russian and Hajič's morphological tagger for Czech [8]. Figure 1 provides examples of the tagged and lemmatized parts of text in the format as suitable for the factored training.

4 Simple Moses

Moses[3] is a phrase based SMT system that is very much language independent since it implements a purely data driven method. In contrast to other methods of MT, phrase-based systems can perform translation directly between surface forms (thus often the name "direct translation"). The most important property of phrase-based systems is the ability to translate contiguous sequences of words (called "phrases") rather than merely single words. See Figure 2 for an illustration.

The Moses toolkit is a complex system which utilizes several other components. Let us mention at least GIZA++[4] involved in finding word alignment, the SRI Language Modeling Toolkit[5] and the built-in implementation of model optimization (Minimum Error Rate Training, MERT) on a given held-out set of sentences.

To establish a baseline, we trained translation models for direct translation from Russian to Czech (ru→cs simple) and English to Czech (en→cs simple), optimizing them on the 750 held-out sentences.

5 Moses factored

All knowledge used by Moses comes from the corpus. Moreover, direct phrase-based translation models have no generalizing capacity. Thus their performance strongly depends on whether particular words and word sequences were seen in the training sentences data. Phrase-based translation thus often faces a problem known as data sparseness, and the problem is more

[2] http://ufal.mff.cuni.cz/umc/
[3] http://www.statmt.org/moses/
[4] http://www.fjoch.com/GIZA++.html
[5] http://www.speech.sri.com/projects/srilm/

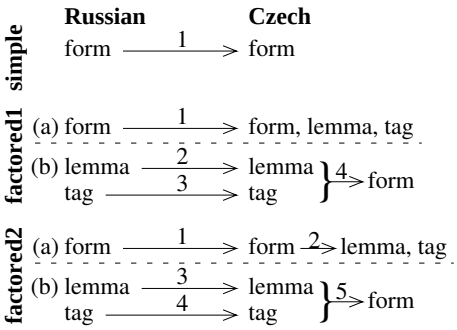


Fig. 3. Ilustration of all explored translation settings: (a) and (b) parts represent alternative decoding paths of a given factored setup.

pronounced for morphologically rich languages where all word forms have to be seen.

Factored translation [6] is an interesting extension of phrase-based models that aims i.a. to mitigate this issue. It allows us to replace an input word with a vector of features as exemplified in Figure 1 and configure the model to back-off to a more coarse-grained representation of input words if there are not enough training data. The features on the source side can also participate in translation. Features on the target side may be obtained by translation from the source side or by a generation step. The generation works with features already available on the target side and fills in the remaining ones.

The most common example of employing factored translation looks as follows. A surface word form is enriched with its base form (lemma) and morphological information (a tag for short), forming a three-compound features vector. Base forms and tags are translated independently without regard to surface forms. Then, on the basis of translated base form and tag the surface form is generated. The setup can use three language models ensuring coherence of the output sequence: one for base forms, one for tags and one for surface forms.

To summarize, there are two translation models (for base forms and for tags), one generation table to get surface form and three language models. This was the approach we first planned to exploit. Unfortunately, the setup has a subtle drawback: it does not work with input forms at all, so it applies the independent translation of base form and tag even in cases where there is enough data for direct translation. Moses allows to specify multiple decoding paths (decoding means finding the most probable translation of a given sentence according to the model), so it is possible to let compete the factored path with the direct transfer, exploiting mutual advantages of both approaches. That is the approach we used in our factored experiments.

Although in the direct translation path used as the back-off of the factored translation we are not interested in the target-side lemma and tag, we still have to supply them for the language models. We use two distinct setups for constructing the additional output factors for the direct translation: 1) translating the source form to all three target factors at once, and 2) translating the source form to target source form and using a generation step for "instant tagging" of the output to construct the target lemma and tag. We denote the combination of the main factored translation with one of the two back-off models *factored1* and *factored2*, resp. Both are ilustrated in Figure 3.

We are aware that there is relatively little possibility for an improvement with factorization in our language pairs and overall setting. For instance, let us point out that generation step for target-side factors is integrated into Moses unlike the preprocessing of input factors where external tools are used. Naturally, the generation capabilities of Moses are rather limited: it learns only from sentences supplied in training. Because we train the generation step only on the target side of the parallel sentences, we cannot expect to gain much coverage by translating lemmas and tags independently because the data will hardly ever provide the required form that should be generated from the target lemma and tag. A better approach would be to either use a larger monolingual corpus for training the generation step, or use an external morphological generator as e.g. [9]. With the current simple setting, we can expect improvement rather to come from the additional lemma- and tag-based language models that will be able to judge hypothesis coherence more robustly.

6 Evaluation

We tried to evaluate the output of our systems by several metrics: BLEU, flagging of errors and a simple hypothesis ranking (i.e. asking "which is the best output").

6.1 BLEU

BLEU score [4] is an established automatic metric used to evaluate MT systems. Thus, despite all known issues we also used it not only for completeness but also as an integral part of model optimization (see MERT in Section 4). Anyway, let us mention two major issues of the BLEU score.

BLEU, when applied to languages with free word order, cannot be reliable indeed. BLEU is based on counting occurrences of n-grams from reference translation in generated output. In many cases the

translator of reference texts will use a word order different from the source sentence, whereas the machine usually preserves the original word order whenever it is an acceptable variant. However, many n-grams do not match when words are swapped. Here are some examples of the problem from our test data:
(reference translation) *syrský postoj by dosah íránské strategie regionální destabilizace nemusel rozšiřovat, ale spíš omezovat.*
(ru→cs translation) *postoj sýrie může omezit, nikoliv rozšířit, sféru vlivu íránské strategie regionální destabilizace.*

Such shifts done by a translator lead to a lower (automatic) score while not necessarily impacting the comprehensibility of the output.

There is a similar problem with inflection. Word forms different from the reference translation are not approved by the BLEU score, so minor translation variations or errors can cause unfair loss in BLEU score. However, a partial remedy may be achieved by scoring lemmatized text:
(reference translation) *složitost hrozeb , jimž čelí izrael*
(ru→cs translation) *složitost hrozeb izraeli*
(en→cs translation) *složitostí hrozby pro izrael*

Table 2 summarizes BLEU scores obtained by our various translation setups. For English all scores are very close. In contrast, Russian is more sensitive to a method – factored translation performs slightly better than simple. Unfortunately, we were unable to compute *factored2* for Russian due to troubles with model optimization. A discussion of closeness of simple and factored results is to be found in the last paragraph of Section 5.

	BLEU score on forms		
pair	simple	factored1	factored2
en→cs	14.58±0.96	15.84±1.03	15.39±1.05
ru→cs	11.91±0.91	13.11±0.90	—

	BLEU score on lemmas		
pair	simple	factored1	factored2
en→cs	24.16±1.10	24.77±1.18	24.99±1.16
ru→cs	15.98±0.97	18.06±0.92	—

Table 2. Achieved BLEU scores in our experiments.

6.2 Flagging of errors

As shown in the previous section, the BLEU metric does not always reflect translation quality. A more reliable, though labour-intensive approach is to manually judge MT output. In one of such evaluations, inspired by [7], human annotators mark errors in MT output and classify them according to their nature. We used the following rough error classes: **Bad Punctuation**, **Unknown Word**, **Missing Word**, **Word Order**, **Incorrect Words**, with some classes further refined into several subtypes. As our annotation capabilities were limited to one person only, we present here the evaluation of the simple model (direct translation) only.

Table 3 documents that in the case of English-to-Czech translation, the most common errors concerned morphology, which matches our expectations as Czech is a inflective language and needs to express many features like case and gender, often not marked in English source. On the other hand, lots of words were not recognized in Russian-to-Czech translations. We have not been able to evaluate the factored translation according to the scheme, but a first few sentences show higher accuracy in morphological forms when factored models are used.

Error Class	en→cs	ru→cs
Disambiguation	9.3 %	8.8 %
Extra word	6.2 %	18.2 %
Word Form	49.0 %	22.0 %
Lexical Variant	5.4 %	5.7 %
Missed Auxilary	0.8 %	1.9 %
Missed Content	6.6 %	20.1 %
Word Order Long	0.8 %	0.6 %
Word Order Short	4.6 %	0.6 %
Punctuation	13.9 %	2.5 %
Unknown	3.5 %	19.5 %
Total	259 (100.0%)	159 (100.0%)

Table 3. Error types in simple moses model.

6.3 Ranking of translations

Finally, we carried out a ranking evaluation which is very similar to the human judgments in WMT Manual Evaluation[6]. For each of the translation schemes described in Section 4 and Section 5 we took 40 sentences and ranked them on the basis of the question "which translation is the best". So each MT output of the 40 test sentences translated to Czech from both languages and by all examined setups got a score from 1 (worst) to 5 (best). Table 4 summarizes the evaluation. For each translation setup, we compute the mean, median and count of how often the method got the best and the second best rank.

Almost a half of the sentences that got the highest score were factored translations from Russian into

[6] `http://www.statmt.org/wmt08/judge/`

En→Cz	simple	factored1	factored2
Median	3	3	2
Mean	2.487	3.051	2.718
Best/Second	2/8	9/6	4/6

Ru→Cz	simple	factored1	factored2
Median	4	4	—
Mean	3.436	3.923	—
Best/Second	10/12	19/9	—

Table 4. Manual ranking of MT output.

Czech, the second score was obtained by those translated using the simple model from Russian into Czech. Factored model (*factored1*) from English to Czech was the third one. This confirms our expectation that translating from a related language is easier also for phrase-based MT.

The evaluation allows us to make further conclusions. First, enriching the model with additional morphological information improves the translation quality both for related and unrelated languages. For Russian as the source, the improvement seems to be less apparent, because Russian itself marks most of the relevant morphological properties in its word forms. Second, BLEU score does not necessarily corresponds with manual judgments: while translating from Russian was better percieved by our human annotator, it obtained a lower BLEU score than translation from English[7]. We are aware that the evaluation should be repeated with more human annotators and on a larger set of sentences for a better confidence.

6.4 Observation of frequent errors

As it was shown in the previous section, there are lots of words unrecognized (not translated). This problem is not of a linguistic nature, it is caused simply by insufficient training data.

Here we will name some linguistically interpreted errors.

- Russian → Czech
  - Lost negation.
    (ru src) *без которого было невозможно создание*
    (cs ref) *bez něhož* **ne***bylo možné sestavit*
    (ru → cs) *bez něhož bylo možné vytvoření*
    Here we can observe that due to the difference in how negation is expressed in the two languages, the negative sense is translated as positive.
  - Lost reflexive particle.
    (ru src) *сумел уйти от*
    (cs ref) **se** *zdařilo vyjít z*
    (ru → cs) *podařilo odejít od*
    The mistake above—missing reflexive particle in Czech—is caused by the fact that some verbs can be reflexive in Czech and non-reflexive in Russian which is difficult for a phrase-based MT to learn because the reflexive particle is often far away from the verb in training sentences.
- English → Czech
  - Word order in possessive constructions.
    (en src) *mahmoud abbas 's palestinian authority*
    (cs ref) *palestinskou samosprávou prezidenta mahmúda abbáse*
    (en → cs) *prezidenta mahmúda abbáse palestinské samosprávy*
- Both source languages → cs
  - Bad case after a preposition.
    (cs ref) *podle indických vyšetřovatelů*
    (en src) *according to indian investigators*
    (en → cs) *podle* **indické** *řešitelů*
    (ru src) *согласно индийским экспертам*
    (ru → cs) *podle* **indickým** *experti*

7 Conclusion

We have succeeded in our goal to compare the performance of phrase-based and factored phrased-based statistical machine translation when translating between related and unrelated languages. So far we have failed in taking advantage of language relatedness explicitly in the model, but a preliminary manual ranking of system outputs confirms that translation between related languages delivers better results. This observation contradicts to the automatic MT quality score using the BLEU metric.

We are aware of the remaining data sparseness issue (there are many times more tags for Russian than for English), so while the language relatedness makes the Czech and Russian tagsets similar, many tags needed in the translation of unseen sentences are not in our training data. Also we suspect the training corpus to be better parallel for English-Czech pair than for Russian-Czech, because Czech is the direct translation of English original while Russian is the translation of English, not Czech.

Our second conclusion is that enriching SMT with morphological features improves the translation quality especially for the closely-related morphologically rich Czech and Russian.

[7] While BLEU scores are not comparable across language, they *are* comparable in our setup: we test BLEU scores on a single test set in Czech only, it is the source language that differs, not the target one.

We hope that our results will serve as a good basis for a future comparison of SMT with rule-based approach used in Česílko, which intends to include Russian-Czech translation pair soon. Our experiments are also a good start for further improvements in MT quality when translating to Czech. For instance, we plan to improve the morphological generation step by using larger target-side monolingual training data.

References

1. N. Klyueva and O. Bojar: *UMC 0.1: Czech-Russian-English multilingual corpus.* Proc. of International Conference Corpus Linguistics., Saint-Petersburg, 2008, 188–195.
2. P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. Dyer, O. Bojar, A. Constantin, and E. Herbst: *Moses: open source Toolkit for statistical machine translation.* ACL 2007, Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions, Prague, Czech Republic, 2007, 177–180.
3. P. Homola and V. Kuboň: *A hybrid machine translation system for typologically related languages.* Proceedings of the 21st International Florida-Artificial-Intelligence-Research-Society Conference, FLAIRS, 2008, 227–228.
4. K. Papineni, S. Roukos, T. Ward: *BLEU: a method for automatic evaluation of machine translation.* IBM Research Report RC22176(W0109-022), 2001.
5. H. Schmid: *Probabilistic part-of-speech tagging using decision trees.* Proceedings of International Conference on New Methods in Language Processing, 1994.
6. P. Koehn and H. Hoang: *Factored translation models.* Conference on Empirical Methods in Natural Language Processing (EMNLP), 2007, 868–876.
7. D. Vilar, J. Xu, L. Fernando D'Haro, and H. Ney: *Error analysis of statistical machine translation output.* LREC-2006: Fifth International Conference on Language Resources and Evaluation. Proceedings, Genoa, Italy, 22-28 May 2006, 697–702.
8. J. Hajič: *Disambiguation of rich inflection.* (Computational Morphology of Czech). Nakladatelství Karolinum, ISBN 80-246-0282-2, Prague, 2004.
9. A. de Gispert, J.B. Mariño and J.M. Crego: *Improving statistical machine translation by classifying and generalizing inflected verb forms.* Eurospeech 2005, Lisbon, Portugal, 2005, 3185–3188.

Benchmarking a B-tree compression method*

Filip Křižka, Michal Krátký, and Radim Bača

Department of Computer Science, Technical University of Ostrava, Czech Republic
{filip.krizka,michal.kratky,radim.baca}@vsb.cz

Abstract. *The B-tree and its variants have been widely applied in many data management fields. When a compression of these data structures is considered, we follow two objectives. The first objective is a smaller index file, the second one is a reduction of the query processing time. In this paper, we apply a compression scheme to fit these objectives. The utilized compression scheme handles compressed nodes in a secondary storage. If a page must be retrieved then this page is decompressed into the tree cache. Since this compression scheme is transparent from the tree operation's point of view, we can apply various compression algorithms to pages of a tree. Obviously, there are compression algorithms suitable for various data collections, and so, this issue is very important. In our paper, we compare the B-tree and compressed B-tree where the Fast Fibonacci and invariable coding compression methods are applied.*

Key words: *B-tree and its variants, B-tree compression, compression scheme, fast decompression algorithm*

1 Introduction

The B-tree represents an efficient structure for the finding of an ordered set [6]. The B-tree has been often used as the backbone data structure for the physical implementation of RDBMS or file systems. Its most important characteristic is that keys in a node have very small differences to each others. We utilize this feature in the B-tree compression. In this case, nodes are compressed in the secondary storage and they are decompressed during their reading into the cache. Due to the fact that the random access in the secondary storage is a rather expensive operation, we save time when reading the nodes.

In work [11], authors summarize some methods for organizing of B-trees. A prefix B-tree, introduced in [7], provides the head and tail compression. In the case of the head compression, one chooses a common prefix for all keys that the page can store, not just the current keys. Tail compression selects a short index term for the nodes above the data pages. This index needs merely to separate the keys of one data node from those of its sibling and is chosen during a node split. Tail compression produces variable length index entries, and [7] describes a binary search that copes with variable length entries.

Work [9] describes a split technique for data. Rows are assigned tag values in the order in which they are added to the table. Note that tag values identify rows in the table, not records in an individual partition or in an individual index. Each tag value appears only once in each index. All vertical partitions are stored in the B-tree with the tag value as the key. The novel aspect is that the storage of the leading key is reduced to a minimal value.

Unlike these works, in our work we suppose the B-tree compression without changes of the B-tree structure. We mainly utilize the fast decompression algorithm. In the case of the previously depicted papers, B-tree compression is possible using a modification of the B-tree structure. In work [7], B-tree is presented by B*-index and B*-file. The keys stored in the B*-index are only used to searching and determining in which subtree of a given branch node a key and its associated record will be found. The B*-index itself is a conventional B-tree including prefixes of the keys in the B*-file. This prefix B-tree combines some of the advantages of B-trees, digital search trees, and key compression without sacrificing the basic simplicity of B-trees and the associated algorithms and without inheriting some of the disadvantages of digital search trees and key compression techniques. Work [9] describes an efficient columnar storage in B-trees. Column-oriented storage formats have been proposed for query processing in relational data warehouses, specifically for fast scans over non-indexed columns. This data compression method reuses traditional on-disk B-tree structures with only minor changes yet achieves storage density and scan performance comparable to specialized columnar designs. In work [1], B-tree compression is used for minimizing the amount of space used by certain types of B-tree indexes. When a B-tree is compressed, duplicate occurrences of the indexed column values are eliminated. It is compressed by clustering the same keys and their unindexed attributes.

This paper is organized as follows. In Section 2, we briefly summarize basic knowledge about the B-tree. Section 3 shows a compression scheme used [3]. Section 4 describes two compression methods. Section 5 shows results of the compression methods. The compressed B-tree is compared with a proper B-tree. In

* Work is partially supported by Grants of GACR No. 201/09/0990 and IGA, FEECS, Technical University of Ostrava, No. BI 4569951, Czech Republic.

Section 6, we summarize the paper content and outline possibilities of our future work.

2 B-tree and its variants

The B-tree is a tree structure published by Rudolf Bayer and Edward M. McCreight in 1972 [6]. The B-tree keeps data sorted and allows searches, insertions, and deletions in logarithmic amortized time. It is optimized for systems that read and write large blocks of data. A B-tree is kept balanced by requiring that all leaf nodes are at the same depth. This depth will increase slowly as elements are added to the tree, but an increase in the overall depth is infrequent, and results in all leaf nodes being one more node further away from the root.

B-trees have substantial advantages over alternative implementations when node access times far exceed access times within nodes. This usually occurs when most nodes are in secondary storage such as hard drives. By maximizing the number of child nodes within each internal node, the height of the tree decreases, balancing occurs less often, and efficiency increases. Usually this value is set such that each node takes up a full disk block or an analogous size in secondary storage.

A B-tree of order m (the maximum number of children for each node) is a tree which satisfies the following properties:

- Every node has at most m children.
- Every node (except root and leaves) has at least $\frac{m}{2}$ children.
- The root has at least two children if it is not a leaf node.
- All leave nodes are in the same level.
- All inner nodes with k children contain k–1 links to children.

Each internal node's elements act as separation values which divide its subtrees. For example, if an internal node has three child nodes (or subtrees) then it must have two separation values or elements a_1 and a_2. All values in the leftmost subtree will be less than a_1, all values in the middle subtree will be between a_1 and a_2, and all values in the rightmost subtree will be greater than a_2.

Internal nodes in a B-tree – nodes which are not leaf nodes – are usually represented as an ordered set of elements and child pointers. Every internal node contains a maximum of U children and – other than the root – a minimum of L children. For all internal nodes other than the root, the number of elements is one less than the number of child pointers; the number of elements is between $L-1$ and $U-1$. The number U must be either $2 \cdot L$ or $2 \cdot L-1$; thus each internal node is at least half full. This relationship between U and L implies that two half-full nodes can be joined to make a legal node, and one full node can be split into two legal nodes (if there is an empty space to push one element up into the parent). These properties make it possible to delete and insert new values into a B-tree and adjust the tree to preserve the B-tree properties.

Leaf nodes have the same restriction on the number of elements, but have no children, and no child pointers. The root node still has the upper limit on the number of children, but has no lower limit. For example, when there are fewer than L-1 elements in the entire tree, the root will be the only node in the tree, and it will have no children at all.

A B-tree of depth n+1 can hold about U times as many items as a B-tree of depth n, but the cost of search, insert, and delete operations grows with the depth of the tree. As with any balanced tree, the cost increases much more slowly than the number of elements.

Some balanced trees store values only at the leaf nodes, and so have different kinds of nodes for leaf nodes and internal nodes. B-trees keep values in every node in the tree, and may use the same structure for all nodes. However, since leaf nodes never have children, a specialized structure for leaf nodes in B-trees will improve performance. The best case height of a B-tree is: $\log_M n$. The worst case height of a B-tree is: $\log_{\frac{M}{2}} n$. Where M is the maximum number of children a node can have.

There exists many variants of the B-tree: B*-tree [13], B**-tree [15], B$^+$-tree [17]. In the case of the B$^+$-tree, data is only stored in leaf nodes and inner nodes include keys. Leaf nodes hold links to the previous and next nodes. Moreover, many paged data structures like UB-tree [5, 12], BUB-tree [8], and R-tree [10] are based on the B-tree.

3 A compression scheme for tree-like data structures

In this section, we describe a basic scheme which can be utilized for most paged tree data structures [3]. Pages are stored in a secondary storage and retrieved when the tree requires a page. This basic strategy is widely used by many indexing data structures such as B-trees, R-trees, and many others. They utilize cache for fast access to pages as well, since access to the secondary storage can be more than 20 times slower compared to access to the main memory. We try to decrease the amount of disc access cost (DAC) to a secondary storage while significantly decreasing the size of a tree file in the secondary storage.

Let us consider a common cache schema of persistent data structures in Figure 1(a). When a tree requires a node, the node is read from the main memory cache. If the node is not in the cache, the node page is retrieved from the secondary storage.

An important issue of the compression schema is that tree pages are only compressed in the secondary storage. In Figure 1(b), we can observe the basic idea of the scheme. If a tree data structure wants to retrieve a page, the compressed page is transfered from the secondary storage to the tree's cache and it is decompressed there. Function `TreeNode:: Decompress()` performs the decompression. Afterwards, the decompressed page is stored in the cache. Therefore, the tree's algorithms only work with decompressed pages. Obviously, the tree is preserved as a dynamic data structure and in our experiments we show the page decompression does not significantly affect query performance because we save time with the lower DAC.

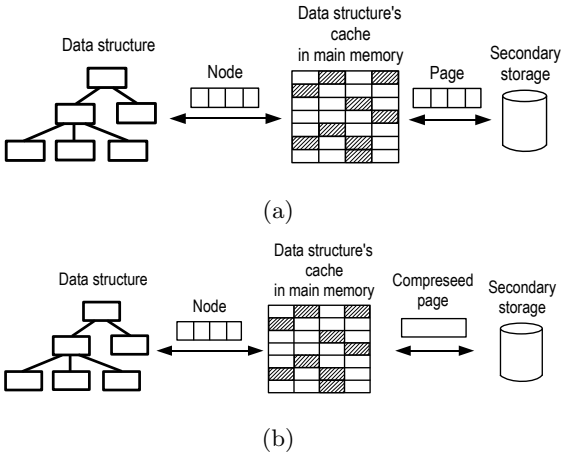


Fig. 1. (a) Transfer of tree's pages between the secondary storage and tree's cache. (b) Transfer of compressed pages between the secondary storage and tree's cache.

3.1 How the compression scheme affects tree algorithms

When the compression scheme is taken into consideration, the tree insert algorithm only needs to be slightly modified. When we insert or modify a record in a page, we have to perform the function `TreeNode::Compress Test()` which tests whether the node fits into the page. If not, the node needs to be split. Also, during the split, we have to make sure that the final nodes fit into the page. This means that the maximum capacity of a page can vary depending on the redundancy of the data. The maximum capacity of each tree's page must be determined by a heuristic:

$$C_c = \frac{C_u}{CR_A},$$

where CR_A is the assumed maximum compression ratio, C_u is the capacity of the uncompressed page. For example, the size of the page is 2,048 B, $C_u = 100$, $CR_A = 1/5$, then $C_c = 500$. The size of the page for the capacity is 10,240 B. This means that all pages in the tree's cache have $C_c = 500$, although their S size in the secondary storage is less than or equal to 2,048 B. Let us note that

$$CR = \frac{compressed\ size}{original\ size}.$$

The `TreeNode::Compress()` function is called when the page must be stored in the secondary storage.

Every page in the tree has its minimal page utilization $C_c/2$. Let S_l denote the byte size of a compressed page. After deleting one item in the page, the byte size of the page is denoted by S_c. Without loss of generality, we can assume that $S_c \leq S_l$. If items are deleted from the page, we must check whether capacity is less than or equal to $C_c/2$. If so, the page is stretched into other pages according to the tree deleting algorithm.

Query algorithms are not affected at all because page decompression is processed only between cache and secondary storage and the tree can utilize decompressed pages for searching without knowing that they have been previously compressed.

This basic idea of the compression scheme can be applied to any paged tree data structure. It is not dependent upon an applied split algorithm, nor on the key type stored in the tree's pages. We test this scheme on B^+-tree data structure because this data structure has remained very popular in recent years and it is suitable for further page compressing.

4 B-tree compression methods

In this article, we have applied two compression methods: Fast Fibonacci (FF) and Invariable Coding (IC). Since keys in a node are close to each another, we use the well-known difference compression method [14]. Similar algorithms were used in the case of the R-tree compression [3, 16].

4.1 Fast Fibonacci compression

In this method, we apply the Fibonacci coding [2] which uses the Fibonacci numbers; 1, 2, 3, 5, 8, 13, ... A value is coded as the sum of the Fibonacci numbers that are represented by the 1-bit in a binary buffer. Special 1-bit is added as the lowest bit in this binary buffer after the coding is finished. For example, the

Algorithm 1: Fast Fibonacci Compression Algorithm

```
   function : CompressionLeafNode(node)
 1 Write item1
 2 for i in 2 .. node.count do
 3     num ←
       FibonacciCodder(node.key(i)-node.key(i-1))
 4     Write num
 5     num ← FibonacciCodder(node.nonatrr(i))
 6     Write num
 7 end
 8 Write links
   function : CompressionNode(node)
 9 Write item1
10 for i in 2 .. node.count do
11     num ←
       FibonacciCodder(node.key(i)-node.key(i-1))
12     Write num
13 end
14 Write links
   function : Compression(node)
15 if node.isLeaf is leaf then
16     CompressionNode(node)
17 end
18 else
19     CompressionLeafNode(node)
20 end
   function : CompressionTest(node)
21 tmp ← Compression(node)
22 if tmp.size > page.size then
23     return false
24 end
25 else
26     return true
27 end
```

value 20 is coded as follows: 0101011 (13 + 5 + 2). Due to the fact that we need the compression algorithm as fast as possible, we use the Fast Fibonacci decompression introduced in [4].

Algorithm of the Fast Fibonacci compression is shown in Algorithm 1. We explain the algorithm in the following paragraphs.

Compression of inner nodes:

- Keys are compressed by the Fibonacci coding. The first value, obviously minimal, is stored. We compute the difference of each neighboring value and the difference is coded by Fibonacci coding. (see Lines 10-13)
- Child and parent links are not compressed. (Line 14)

Compression of leaf nodes:

- Keys are compressed in the same way as the keys in an inner node. (Lines 3-4)
- Unindexed attribute values are compressed by Fibonacci coding. (Lines 5-6)
- Parent, previous, and next nodes links are not compressed. (Line 8)

4.2 Invariable coding compression

As in the previous method, we work with the difference of each neighboring value. Since, we use invariable coding, we must first compute the difference of the last, maximal value, and the first value. The result of this computation is the number of bits necessary for a storage of the maximal value and, obviously, each value of

Algorithm 2: Invariable Coding Compression Method

```
   function : CompressionLeafNode(node)
 1 Write maxBits(node.key(1),node.key(n))
 2 Write node.key(1)
 3 for i in 2 .. node.count do
 4     num ← ICCodder(node.key(i)-node.key(i-1))
 5     Write num
 6 end
 7 Write maxBits(node.nonatrr(1),node.nonatrr(n))
 8 Write node.nonatrr(min)
 9 for i in 2 .. node.count do
10     num ←
       ICCodder(node.nonatrr(i)-node.nonatrr(min))
11     Write num
12 end
13 Write links
   function : CompressionNode(node)
14 Write maxBits(node.key(1),node.key(n))
15 Write node.key(1)
16 for i in 2 .. node.count do
17     num ← ICCodder(node.key(i)-node.key(i-1))
18     Write num
19 end
20 Write links
   function : Compression(node)
21 if node.isLeaf is leaf then
22     CompressionNode(node)
23 end
24 else
25     CompressionLeafNode(node)
26 end
   function : CompressionTest(node)
27 tmp ← Compression(node)
28 if tmp.size > page.size then
29     return false
30 end
31 else
32     return true
33 end
```

	RND_8			RND_16		
	B$^+$-tree	**FF**	**IC**	**B$^+$-tree**	**FF**	**IC**
Height	2	2	2	2	2	2
Domain	8b			16b		
DAC Read	20,858,473	12,115,647	10,010,790	5,996,536	5,872,078	5,739,912
Creating time [s]	15,288	65,618	44,201	6,360	11,897	10,605
Cache Time [s]	7,357	5,398	1,565	6,020	1,495	1,181
Compress time [s]	0	867	652	0	883	636
Decompress time [s]	0	53,524	35,908	0	9,276	8,567
# Inner nodes	35	17	9	33	17	9
# Leaf nodes	7,746	3,350	2,431	5,695	2,596	2,114
# Avg. node items	222.3	198	271	172.58	153.65	235.8
# Avg. leaf node items	129.1	298.5	411.4	176.58	385.21	473
Index size [kB]	15,564	6,736	4,882	11,394	5,228	4,248
Compression ratio	1	0.56	0.69	1	0.54	0.63

Table 1. Building B$^+$-tree index, result for RND_8 and RND_16.

	RND_24			RND_32		
	B$^+$tree	**FF**	**IC**	**B$^+$tree**	**FF**	**IC**
Height	2	2	2	2	2	2
Max item value	24b			32b		
DAC Read [all]	5,996,536	5,907,459	5,830,822	5,996,536	5,931,627	5,889,079
Creating time [s]	7,377	12,098	12,435	7,629	13,686	13,154
Cache Time [s]	7,001	1,556	1,690	7,203	2,935	2,267
Compress time [s]	0	882	664	0	885	597
Decompress time [s]	0	9,419	9,828	0	9,595	10,003
# Inner nodes	33	17	17	33	26	17
# Leaf nodes	5,663	2,717	3,099	5,663	2,800	3,756
# Avg node items	172.58	160.76	183.24	172.58	108.65	221.88
# Avg leaf node items	176.58	368.05	322.68	176.58	357.14	266.24
Tree size [kB]	11,394	5,470	6,234	11,394	5,654	5,272
Compression ratio	1	0.52	0.45	1	0.50	0.54

Table 2. Building B$^+$-tree index, results for RND_24 and RND_32.

	RND_8			RND_16			RND_24			RND_32		
	B$^+$tree	**FF**	**IC**	**B$^+$tree**	**FF**	**IC**	**B$^+$tree**	**FF**	**IC**	**B$^+$tree**	**FF**	**IC**
Query time [s]	182.4	37.6	33.5	38.2	34.2	31.7	38.1	34.2	35.2	38.2	45.3	37.2
Decompress time [s]	0	6.8	5.7	0	6.7	5.0	0	6.6	6.1	0	6.8	6.8
Cache Time [s]	149.1	3.8	1.8	5.0	2.0	1.3	5.6	2.4	2.5	7.2	12.7	3.0
DAC Read	64,813	28,123	20,516	47,068	21,897	17,610	47,068	22,469	25,700	47,068	23,200	31,296

Table 3. B-tree querying results for RND_8, RND_16, RND_24 and RND_32.

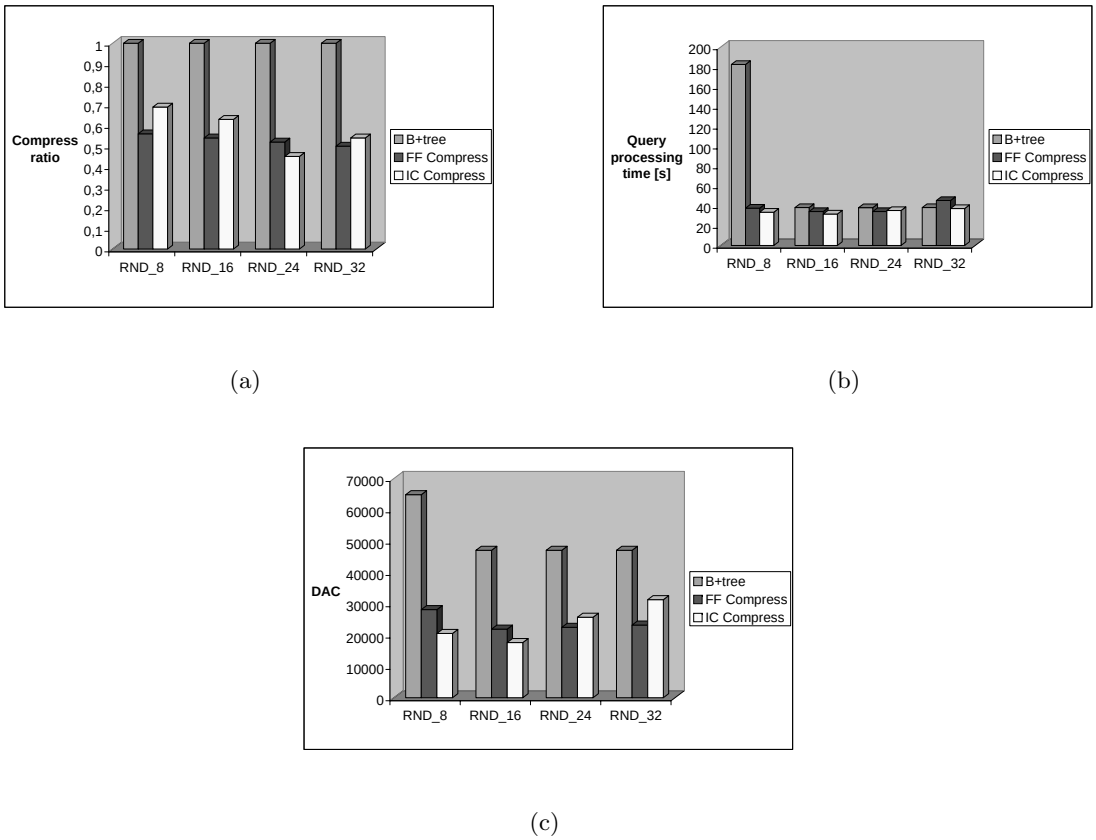


Fig. 2. Experiment results: (a) Compression ratio (b) Query processing time (c) DAC.

the node. For example, if the difference of the maximal and minimal value is 20, all values are stored in 5bits.

Algorithm of the IC compression is shown in Algorithm 1. We explain the algorithm in the following paragraphs.

Compression of inner nodes:

- Keys are compressed by the above proposed method. We store the first value and the number of bits necessary for a storage of the maximal value. After, all difference values are stored. (Lines 14-19)
- Child and parent links are not compressed. (Line 20)

Compression of leaf nodes:

- Keys are compressed in the same way as the keys in an inner node. (Lines 1-6)
- Unindexed attribute values are similarly compressed as keys, however the maximal value is not the last value – it must be found by a sequence scan.
- Parent, previous, and next nodes links are not compressed. (Line 13)

5 Experimental results

In our experiments[1], we test previously described compression methods. These approaches are implemented in C++. We use four synthetic collections which differ in values included. Collection RND_8 includes values in $\langle 0; 255 \rangle$, RND_16 includes values in $\langle 0; 65,535 \rangle$. In this way, we create collections RND_24 and RND_32 as well. Each collection contains 1,000,000 items.

For each collection, we test index building and querying by processing time and DAC. In all tests, the page size is 2,048B and cache size is 1,024 nodes. The cache of the OS was turned off.

Results of index building are depicted in Table 1 and 2. We see that the compression ratio decreases for increasing size of domains. FF compression is more efficient for lower values; on the other hand, the IC compression is more efficient for higher values. Obviously, due to features of the compressed scheme used, we obtain the high compress time. Consequently, the

[1] The experiments were executed on an AMD Opteron 865 1.8Ghz, 2.0 MB L2 cache; 2GB of DDR333; Windows 2003 Server.

time of creating of B^+-tree with the FF compression is $1.6 - 4.3\times$ higher then the time of creating for the B^+-tree. In the case of the IC compression, the creating time is $1.7 - 2.9\times$ higher. The compression ratio is shown is Figure 4.2(a) as well.

In our experiments, we test 50 random queries and the results are then averaged. The results are shown in Table 3. The number of DAC is $2.1 - 3.5\times$ lower for the FF compression when compared to the B^+-tree and $1.5 - 3.6\times$ for the IC compression. This result influences the query processing time. The query processing times is $0.84 - 4.85\times$ more efficient in the case of the FF compression when compared to the B^+-tree and the time is $1.03 - 5.4\times$ more efficient for the IC compression. Obviously, if the compression ratio is over a threshold then the B^+-tree overcomes the compressed indices. In Figure 4.2(b) and (c), we see the query processing time and DAC.

6 Conclusion

In this article, we propose two methods for B-tree compression. If the compression ratio is below a threshold then the query processing performance of the compressed index overcomes the B-tree. However, there are still some open issues. The first one is the high creating time. In this case, we must develop a more efficient method or we must use and test the bulkloading (see [3, 16]). Additionally, we must test our method for a real data collection. Finally, we should test different compression and coding methods (i.e. Elias-delta code, Elias-gamma code, Golomb code [14]).

References

1. C. Antognini: *Troubleshooting Oracle Performance*. Apress, 2008.
2. A. Apostolico and A. Fraenkel: *Robust transmission of unbounded strings using Fibonacci representations*. IEEE Trans. Inform., 33, 2, 1987, 238–245.
3. R. Bača, M. Krátký, and V. Snášel: *A compression scheme for the R-tree data structure*. In Submitted in Information Systems, 2009.
4. R. Bača, V. Snášel, J. Platoš, M. Krátký, and E. El-Qawasmeh: *The fast Fibonacci decompression algorithm*. In arXiv:0712.0811v2, `http://arxiv.org/abs/0712.0811`, 2007.
5. R. Bayer: *The universal B-tree for multidimensional indexing: general concepts*. In Proceedings of World-Wide Computing and Its Applications (WWCA 1997), Tsukuba, Japan, Lecture Notes in Computer Science, Springer–Verlag, 1997, 198–209.
6. R. Bayer and E. M. McCreight: *Organization and maintenance of large ordered indices*. Acta Informatica, 1972, 173–189.
7. R. Bayer and K. Unterauer: *Prefix B-trees*. *ACM Trans. on Database Systems*, 2, 1, 1977, 11–26.
8. R. Fenk: *The BUB-tree*. In Proceedings of 28rd VLDB International Conference on Very Large Data Bases (VLDB'02), Hongkong, China, Morgan Kaufmann, 2002.
9. G. Goetz: *Efficient columnar storage in B-trees*. In Proceedings of SIGMOD Conference, 2007.
10. A. Guttman: *R-Trees: a dynamic index structure for spatial searching*. In Proceedings of ACM International Conference on Management of Data (SIGMOD 1984), ACM Press, June 1984, 47–57.
11. D. Lomet: *The evolution of effective B-tree page organization and techniques: a personal account*. In Proceedings of SIGMOD Conference, Sep. 2001.
12. V. Markl: *Mistral: Processing relational queries using a multidimensional access technique*. Ph.D. thesis, Technical University München, Germany, 1999, `http://mistral.in.tum.de/results/publications/Mar99.pdf`.
13. Y. Sagiv: *Concurrent operations on B*-trees with overtaking*. In Journal of Computer and System Sciences, 1986.
14. D. Salomon: *Data Compression The Complete Reference*. Third Edition, Springer–Verlag, New York, 2004.
15. A.A. Toptsis: *B**-tree: a data organization method for high storage utilization*. In Computing and Information, 1993.
16. J. Walder, M. Krátký, and R. Bača: *Benchmarking coding algorithms for the R-tree compression*. In Proceedings of DATESO 2009, Czech Republic, 2009.
17. N. Wirth: *Algorithms and Data Structures*. Prentice Hall, 1984.

Input combination for Monte Carlo Localization

David Obdržálek

Charles University in Prague, Faculty of Mathematics and Physics
Malostranské náměstí 25, 118 00 Praha 1, Czech Republic
david.obdrzalek@mff.cuni.cz

Abstract. *One of the basic skills for an autonomous robot is the ability to determine its own position. There are numerous high-level systems which provide precise position information, but the same task may be also solved using less advanced and less accurate sensors. In our paper, we show how a good output may be acquired from not so good inputs if it is combined using modified Monte Carlo Localization (MCL) system. The combination of more inputs also helps to acquire plausible results even in situations, where adding new sensor(s) to an existing system raises doubts about the position calculation, because the newly added sensor may provide different position information (or data from which the position is calculated) than what is provided by the already used sensors. We will show that using such "incorrect" data may be beneficial for determining the position with a reasonable probability.*

1 Introduction

Good localization is for an autonomous robot one of the keys to success. A robot which does not know its position, or which gets lost while performing its task, is not something what could be used in real life well. For some tasks, localization can be quite simple, but in general, the better autonomous robot we want, the better (and usually more complicated) localization it needs. Some systems use single input for the localization task and do not need any complex mechanisms to accomplish all needed goals. Other systems combine more inputs (and more input types) to acquire data for the localization process; it may be because of different nature of the data which is available to be measured for the localization as well as because different sensing methods may help to overcome problems with erroneous data coming from individual sensors. However, combining more inputs may at the same time rise questions about the preciseness of the localization process: What went wrong if two or more inputs showed different positions? Is it because of faulty sensor, inexact measurement, cumulative errors or improper data handling?

Recently, the research in the robot localization started to bring very good results using probabilistic methods. In this paper, we discuss this problem in general, and we present one particular implementation which uses Monte Carlo Localization (MCL).

The following text is organized as follows: Section 2 gives characterization of the task and presents the specifics of our selected problem. Section 3 gives brief outline of existing localization techniques. Section 4 presents Monte Carlo Localization in general, Section 5 shows our modification to MCL by differentiating between sensor classes and Section 6 shows some implementation aspects. Sections 7 and 8 summarize the results and conclude the paper.

2 Motivation and characterization of the task

The goal of robot localization is to determine the position of the robot which moves through its working environment. It may use data about the environment and data about the robot, both using measured data as well as data known in advance. To a great extent, the localization algorithm itself can be application independent and the usage for specific purposes can vary just by choosing different inputs.

Inputs for the localization system may come from different sources: data may be acquired for example using different sensors mounted on the robot (measuring either internal or external properties), by receiving signals sent from external sources, or even created as virtual data which does not represent any real measurements (e.g. expected position change based on commands issued by the control system). Because the different sensors provide data with different level of accuracy and trustworthiness, data should be also handled with different weights.

The localization algorithm processes the selected inputs to calculate the robot position. If the algorithm itself does not depend on a specific type(s) of input data, then it is possible to create a generic solution which works with different (and configurable) sets of sensors.

In this paper we present one possible solution of the localization problem which has been tested on a real robot. This robot was used to play in the Eurobot autonomous robot contest in 2009 (see [1]). Although the system was created for the 2009 edition of the contest, it was designed so that it could be used without reprogramming for future editions too. Moreover, it was

designed to be independent on this particular task at all and it may be reused in other projects with different inputs as well.

The Eurobot contest rules change every year, but they always share the same core of basic characteristics (for more details, see the Eurobot Association web pages at [1]):

- Known indoor environment with highlighted important landmarks
- Small working area (2.1 × 3 meters)
- Possibility to place localization beacons at predefined places around the working area
- Target objects placed semi-randomly on the working area
- Predefined starting position of the robot
- Limited height and circumference of the robot
- Two robots moving in the area at the same time with the necessity to avoid collisions

This list obviously affects the development of robot hardware and software. However, our aim was to create a localization algorithm which is not that much application dependent and can be used for other applications in different conditions too.

The exactly needed precision level is application dependent and varies a lot from one application to another. For the specific conditions it is however important to maintain the precision in an acceptable range. Therefore, our aim was to create a localization algorithm which would be able to reach the required precision level even using less precise inputs. The precision should not be predefined nor implied by the algorithm.

3 Localization algorithms

The area of autonomous robot localization is well researched (see e.g. [2]), and several ways can be used to solve the localization problem. Therefore we do not try to invent a new algorithm. Instead, we will outline some existing localization algorithms and discuss some of their implementation details, together with technical problems we have met.

For localization based on various input values one can choose from many algorithms; the most know are:

Kalman filter [3, 4] generalizes the floating mean. It can handle noisy data so it is suitable for processing the data from less precise sensors. However, the model must be described with the expected value and variability which is often too difficult constraint.

Markov localization based on grid [5] resolves one of the problems of Kalman filter, which needs to know the expected value and the variance of input data. This algorithm splits the area to the grid of proper size and tries to determine the one the robot is in. Unfortunately, this requires large operational memory to store the data and computing power to handle it.

Monte-Carlo localization [6] can represent multimodal distribution and thus localize the robot globally. It can process inputs from many sensors with different accuracies. Moreover, it can be easily implemented.

For our given task, it is not possible to use standard Kalman filter, because its basic requirements are not met: in our case, the expected value and variance of the measured values are not known.

The second mentioned localization algorithm, Markov grid-based localization, would overcome this problem, but would impose another problem at the same time – the need to handle lot of data. The position of a robot is specified as one cell in a grid covering the whole working area. It is needed to store some data for each grid cell, and to reach good precision level, the grid must be fine. Our hardware platform provided sufficient storage with reasonable power for processing the navigation task and for controlling the hardware, but including Markov localization would cause overloading of the system and the goal to create a successful autonomous robot could not be reached: neither the memory volume nor the computational power of our hardware were strong enough to handle Markov grid-based localization alone, not talking about combining it with all the other needed tasks.

Therefore, we have decided to implement Monte-Carlo localization and let it use the remaining resources in the system as long as it does not affect its functionality. This also directly implied the selection of the MCL parameters in the tuning phase – "eat as much as you like as long as supply lasts". At the same time, we gained the possibility to use more different sensors for the localization task.

The Monte-Carlo localization will be further discussed in Section 4 and our implementation in Sections 5 and 6.

4 General description of MCL

In this section we will briefly outline Monte Carlo Localization (MCL), introduced by Dieter Fox in the late 1990s [6]. This algorithm meets all the requirements mentioned in problem statement section earlier in this paper. It is a well defined and researched algorithm and it is also well established in many applications (see e.g. [7–10]).

Monte Carlo Localization maintains a list of possible states of the state space (representing the positions of the robot). Each state is weighted by its probability

of correspondence with the actual state of the robot. In the most common implementation, the state represents the coordinates in 2D Cartesian space and the heading direction of the robot. It may be of course easily extended to 3D space and/or contain more information depicting the robot state. All these possible states compose the so called probability cloud.

The Monte Carlo Localization algorithm consists of three phases: Prediction, Measurement, and Resampling.

During the Prediction phase, a new value for each item of the cloud is computed, resulting in a new probability cloud. To simulate various inaccuracies that appear in a real hardware, random noise is added to each position in the prediction phase. This is very useful. For example: If the wheels were slipping and no random noise was added, the probability cloud would travel faster than the real hardware.

During the measurement phase, data from real sensors are processed to adjust probability of the positions in the cloud. The probability of samples with lesser likelihood (according to sensors) is lowered and vice versa. For example, when the sensors show the robot orientation is northwards, weight for samples representing other orientations is lowered.

The last phase - resampling - manages size and correctness of the cloud. Samples with probability lower than a given threshold are removed from the cloud. To keep the number of positions constant, new positions are added. These new positions are placed around existing positions with high probability.

Formally, the goal is to determine robot's state in step k, presuming the original state and all the measurements $M^k = \{m_i, i = 1..k\}$ are known. Robot's state is given by a vector $x = \langle x, y, \alpha \rangle$, where x and y is the robot position and α is its heading.

During the prediction phase, the probability density $p\left(x_k \mid M^k\right)$ for step k is enumerated. It is based only on presumed movement of the robot without any input from real sensors. Therefore, for any known command u_{k-1} given to the robot, we have

$$p\left(x_k \mid M^{k-1}\right) = \\ = \int p\left(x_k \mid x_{k-1}, u_{k-1}\right) p\left(x_{k-1} \mid M^{k-1}\right) dx_{k-1}$$

In the measurement phase, we will compute the final value of probability density for actual step k. To do so, data from sensors is used. It implies the probability of $p\left(m_k \mid x_k\right)$, where m_k is the actual state and x_k is the assumed position. The probability density in step k is then described by the following equation:

$$p\left(x_k \mid M^k\right) = \frac{p\left(m_k \mid x_k\right) p\left(x_k | M^{k-1}\right)}{p\left(m_k | M^{k-1}\right)}$$

During the initialization of MCL, it is needed to set the probability cloud. If the robot's position is known, then for its (known) state x the probability $p\left(x \mid M^0\right) = 1$, and for all other states $y \neq x$ the probability $p\left(y \mid M^0\right) = 0$.

If the robot's position is not known, the probability of all positions is the same and $p\left(x \mid M^0\right)$ must be set for all x so that

$$\int p\left(x \mid M^0\right) dx = 1$$

One of the most important features of this method is its ability to process data from more than one source. Every sensor participates on computing the probability for the given state. For example, if we have a compass sensor and it reads that the robot is heading to the north, we can lower the probability of differently oriented samples. If robot's bumper signalizes collision, there is a high probability for the robot to be near a wall or another obstacle. It is therefore possible to discard the part of the probability cloud which lies in an open space.

The Monte Carlo Localization can be implemented easily, yet it provides very good results especially in cases, where the sensors do not provide exactly correct data (e.g. the distance measurement is subject to errors). Our implementation of the MCL algorithm, which shows its great usability, is described in more detail in the following section.

5 Sensor classes in modified MCL

We have decided to modify the original MCL algorithm and use sensor input also for the prediction phase. We allow selected reliable sensors to change the position of MCL samples in addition to changing the weights of samples based on the sensor readouts. This allows to maintain the probability cloud more in shape with the actual robot movement, yet we keep the core MCL idea of adding random noise to handle unexpected inputs or inputs which are not in accordance to actual robot movement.

In our implementation, we divide the inputs coming from sensors in two categories, which will be further discussed in following paragraphs:

- Advancing inputs
- Checking inputs

Our system contains two interfaces for these two types of inputs. The device or its abstraction in Hardware Abstraction Layer implements the corresponding interface based on its type, so the MCL core can use it as its input. The MCL core calls each device when it has new data, and the work with the samples is done

by each device separately. This keeps the main code easier to read, simpler, and input independent. Also, the device itself knows the best how to interpret the raw data it measures.

The level of reliability can be specified for each input device. Then, the samples are adjusted by the devices with respect to their configured 'credibilities'. For example: two sets of odometry encoders, one pair on driven wheels and one pair on dedicated wheels, have different accuracy because the driven wheels may slip on the surface when too much power is used. Then, the credibility of driven wheels encoders will be set lower than the credibility of the sensors mounted on undriven wheels. In addition, setting the reliability level helps to deal with different frequencies of data sampling.

5.1 Advancing inputs

This input type is used for changing the samples which form the probability cloud. Such input could be for example odometry, from which relative movement since last iteration is calculated. This difference is then used to change the samples properties. i.e. to move them. The information provided by these kind of inputs applied to samples is 'blurred' by randomly generated noise as described earlier in the general MCL description. After moving the samples, boundary conditions are checked (i.e. to exclude samples which fell out of the physical working area). As a result the probability of samples representing impossible positions is decreased.

These advancing inputs are added to the original MCL algorithm, which deals only with theoretical movement based on movement commands. It is not necessary to use it so, but it brings much better precision for little cost.

Our robot currently uses only one advancing input: the odometry information from encoders mounted on dedicated sensor wheels.

5.2 Checking inputs

Checking inputs do not affect the position of the samples. Instead, they are just adjusting their probability (also called sample weights). The reason for this is that inputs of this type provide absolute position information and not relative difference from the last measurement. This also does not need to be one exact point, but an area or position probability distribution, which fits perfectly to the Monte Carlo Localization algorithm.

All checking inputs may be processed separately; we regulate them only by setting their reliability levels.

Our robot uses these checking inputs:

- Compass - checks the direction of samples
- Beacons - check the distance from stationary beacons
- Bumpers - check collisions with playing field borders and other objects
- IR distance sensors - check distance to borders and obstacles

6 Implementation aspects

Our implementation of MCL is based on previous work on a robot which participated in Eurobot 2008 contest (see [11]). That first "pilot" MCL implementation in 2008 was not complete; it was rather proof-of-concept than a reliable software unit, and we also knew the computational power will be different if 2009 so performance was not considered at all. However, the results seemed very promising, so it has not been dropped but has been further developed and extended to use it in 2009 for real. In the following paragraphs, we emphasize several implementation aspects we consider as important for the successful result.

6.1 Position estimation

It is expected that the MCL outputs the estimation of robot position. Because of its nature, the resulting position ("most probable position") can be computed from the samples at any time. This estimation is very simple, just computing the weighted average of all samples. In addition we can determine the overall reliability of this estimation. Therefore, we have made the interface to the MCL asynchronous to the inputs, and the MCL core can be called at any time whenever needed. This approach obviously dramatically saves the computational power in comparison to incremental localization methods which might need periodical updates or re-calculations.

6.2 Localization without initial knowledge

Monte Carlo Localization can also determine the robot position from scratch. At the beginning of localization (when the robot is lost) samples are spread uniformly all over the playing field as described in Section 4. The sensors providing absolute positioning information lower the weight of misplaced samples and new samples are placed in regions with higher probability (see Figure 2). This is repeated until sufficiently reliable position estimation of the robot is reached.

At the same time, it is possible to reach a result even without absolute sensors – as the robot moves, the sensors which provide relative information (advancing inputs) will move the samples as usually and

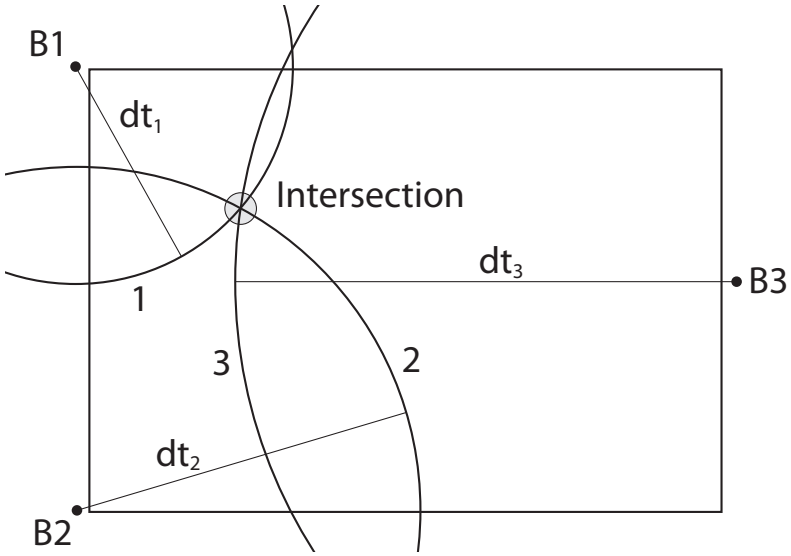


Fig. 1. Beaconing system; $B1$, $B2$, $B3$ are the beacons, and the robot (marked by grey circle) measured distances dt_1, dt_2, dt_3 from individual beacons.

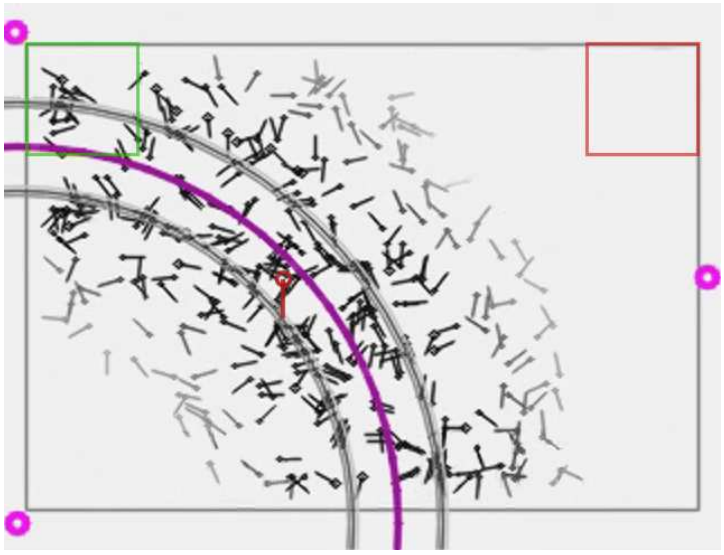


Fig. 2. MCL after processing one beacon input: The circular belt marks the input from the bottom left beacon. The "pins" represent oriented MCL samples; sample probability is proportional to their darkness.

the boundary checks gradually cut the impossible configurations until the required precision is reached (e.g. only one and small cloud remains).

6.3 Adding / removing sensors

When new sensors are added to a system, the information they provide may affect the position the robot "thinks" it is in. It may be because the new sensor gives better data (in which case we certainly appreciate the change). On contrary, the new sensor could provide data with lesser quality then the already existing sensors. This is not a big problem in MCL, because such low quality data may change the samples properties (position) or weight, but because of the nature how MCL works, such change may be perceived as adding the random noise which is part of MCL anyway. So, it may even help the algorithm to work well. It could be said in general – the more different inputs, the better.

If we remove a sensor from the robot or if it stops providing data and there are other sensors available, it does not imply the MCL results will be worse. It means there are fewer inputs which adjust the samples or their weights but the remaining sensors will adapt the probability cloud in accordance to their inputs and so sufficient level of result preciseness can be maintained. Therefore, the system is less vulnerable than a system which relies during the localization on one sensor or sensor set.

6.4 Beacons

In the following paragraphs, we present our design of a sensor set which provides relatively good information about the robot position and in cooperation with other sensors it helps to create a robust localization system – the beacon system.

The main idea of this beacon system is to mount several beacons around the working area of the robot and let the robot measure the distance to these beacons. Then, the robot will be able to estimate its position because the beacons position is known. This sensor set provides absolute position information, but its correctness may be attacked by the environment features, as for example the signal may get reflected by close obstacles like walls or the robot could not be able to measure the distance to one beacon because the signal may get lost (or get blocked by an obstacle).

The principle In our system, we measure the time the signal travels from the transmitter at the beacon to the receiver mounted on the robot (TOF – Time of Flight). Of course this works only if the speed is constant. This condition is met as we are using ultrasonic waves in a small environment with more or less constant conditions and the robot speed is negligible.

Since the speed of the sound waves is known, we can measure the time difference, from which the distance may be easily calculated. It is possible to use two beacons for good position calculation (provided the measurements are precise and we know at which halfplane the robot is). If there 3 or more beacons located around the working area, the trilateration will theoretically give a single solution. Practically, the measurements may not be precise and so the calculation may not lead to any intersection of the circles. In our system, this is not a problem, because the measurements are not used for trilateration but handled separately to adapt the probability cloud used in MCL.

To correctly measure the traveling time, we synchronize the transmitter-receiver system by using infrared light (see below).

Many other systems based on the TOF principle have been developed; for examples see e.g. [12, 13].

Hardware The transmitting system consists of three stationary interconnected beacons located at specified places around the working area of the robot. When the system receives signals from the three beacons and calculates the three distances, it can theoretically determine its position by using trilateration. In praxis, such simplest form does not fully work. The robot may move between receiving signals from all the beacons, some signals may not be received or they may provide incorrect information because of reflections. Even that, good estimation may be acquired as was discussed in Section 6.

The signal is sent one-way only, the receivers do not communicate with the transmitters. Therefore, there can be multiple independent identical receivers, which are able to determine its individual positions. These receivers may be then used for localizing more objects and if the information is passed to a single center, it may be of substantial help (for example to create opponent avoidance system by placing one receiver on the opponent and reading it by wireless transfer).

Transmitters at each beacon work in the following way:

1. Send timing information (infrared)
2. Wait for a defined period of time
3. Send distance measuring information (ultrasonic)
4. Wait for a defined period of time
   (this is sequentially repeated for all beacons)

As we want to measure time difference between the signal being sent and received, we need to have syn-

chronized clocks. This is done in step 1 by using infrared modulated signals. Besides that, the transmitted information contains additional information about the beacon which is going to transmit sound waves.

Sound waves are transmitted as ultrasonic waves, and are always transmitted only from one beacon at a time. The transmitted signal contains also the identification of the source beacon.

Receiver waits for the infrared timing information. When it is received, the receiver resynchronizes its internal timer and generates a message. These messages are transported to the localization unit. Every message contains time stamp, information that synchronization occurred, and the information about beacon which is going to transmit ultrasonic information in this time step. Upon reception of the timing information, the receiver switches its state to wait for ultrasonic signal. When correct ultrasonic information arrives, the receiver generates similar message as is the message after IR reception, but containing time stamp for ultrasonic reception and beacon identification transmitted in the ultrasonic data. The difference in these two timestamps is linearly dependent to distance with a constant offset (the two signals are not transmitted exactly at the same time). Since each beacon identifies itself in both infrared and ultrasonic transmissions, the probability of mismatch is reduced.

When the infrared information is not received, a message is generated saying the synchronization did not occur and the timestamp is generated from previously synchronized internal clock. When the ultrasonic information is not received, localization unit is notified that nothing was received.

The situation after three successfully received ultrasonic signals with synchronized clock can be seen in Figure 1.

6.5 Beacons and MCL

As described earlier, our beacon system consists of three transmitting and one receiving beacons. The information is passed from the beacon system to the main computing unit via messages containing beacon id (i.e. transmitter identification) and time difference between the infrared and ultrasonic transmissions.

There are two reasons why each message contains the time difference (delta) instead of the calculated distance: computational power of the microcontroller and the degree of robustness. The main computing unit is more powerful than the receiving beacon, so we let the beacon do less work and we even benefit from this decision. We considered deltas to be the perfect raw data for our purpose - distance measurement. The computation is done in the main computing unit which controls all the other devices and is highly configurable. It means that all the parameters of the equation for distance calculation can be changed easily without the need of changing the beacons hardware or device firmware. It even allows us to calculate or adjust the parameters on the flight if distance information is provided based on external measurement.

The configuration of the main computing unit contains not only the important constants for the equation, but also the positions of the transmitting beacons. As we know the distance and the beacon id, we can increase the weights of the MCL samples in the circular belt formed by these two values and a range constant. MCL samples far from the belt are penalized (see Figure 2).

This approach is much better and more robust than just waiting for intersections and then computing the robot position using simple trilateration. These intersections may not happen very often because of the time gap between individual beacon transmissions (especially when the robot is moving fast). At the same time, it is good to implement different weighting for the samples on a belt, near an intersection of two belts and near the intersection of all three belts.

6.6 Camera

The idea of using camera for absolute robot positioning seemed very hard at the first time. Later, when we had the modular MCL implementation finished, we realized there is a great opportunity to use the information we get from the camera while looking for the playing elements positioned at predefined places of the playing area. Now, we can compare the playing element positions (acquired from the camera) with their fixed positions (defined by the Eurobot contest rules) and adjust the weight of the MCL samples to merge the two positions. For more details, see [14].

6.7 Gyroscope

In the early stages of robot design, we proposed to use a compass as one of the input sensors. However, using a compass in a small indoor competition is not a very good idea, because its precision can be degraded by influence of many factors (e.g. huge metallic cupboard, electromagnetism, steel concrete walls or metal structure building). Using a gyroscope instead of a compass would be much more efficient for our purposes, because gyroscope works completely independently and the influence of the environment is minimal. The only problem is the placement of the gyroscope itself, because it should be placed in the rotational axis of the robot.

7 The results and performance

Apparently, processing of a large number of MCL samples may have impact on performance of the whole localization system. In general, the more samples are taken into computation, the more precise the localization is, but the slower the computation is.

In our project, we have achieved acceptable speed using 400 samples. The acquired precision lies within a margin of single millimeters which is sufficient for the current task; should higher precision be needed, it could be easily reached by increasing the samples count and fine-tuning the weighting functions. Also, this number of samples and the resulting precision are independent on the working area size.

On contrary, the Markov grid-based localization requires to handle a grid with the size of the robot working area and the number of cells proportional to required precision. In the case of Eurobot contest, to reach the same precision we would need to handle a grid of 2100 x 3000 samples which is magnitudes higher than the 400 samples needed when using MCL.

The depicted modification of using sensors for the prediction phase instead of using just the information of expected robot movement has increased the precision too, as it takes into account not only where the robot was supposed to move, but where it actually moved. To be able to use sensors for this modification, it is needed to assure their good credibility. For our real robot, we have used odometry sensors mounted on dedicated wheels, which are not subject to skids and slides of the powered wheels. It has proven this is sufficient to reach a good level of precision.

8 Conclusion and future work

In our paper, we have described the advantages of the Monte Carlo Localization compared to other methods of position estimation and how we benefit of it in our implementation. Based on available sensor types, we have decided to adapt the MCL algorithm to use one sensor class to change the samples instead of using all sensors to change the sample weights only.

As a practical result, we have developed a modular system for robot localization which allows easy extension by different kinds of modules. Our implementation allows us to add more facilities with almost no or just minimal work effort and with no changes to the core localization itself at all, while increasing the precision of the resulting position. The created system was successfully used for Eurobot 2009 contest edition, and its design allows using it for other purposes too.

Because the system has been created for 2009 edition of Eurobot contest, we want to continue gathering testing data throughout the whole year of 2009 in the contest and all connected events when the working conditions for the robot remain unchanged. After finishing the year, we want to evaluate the performance of this MCL implementation to be able to judge its usage in other environments and with other hardware.

References

1. Eurobot Association: Eurobot autonomous robot contest: `http://www.eurobot.org`, 2009.
2. S. Thrun: *Robotic mapping: a survey.* In: Exploring artificial intelligence in the new millennium. Morgan Kaufmann Publishers Inc., 2003, 1–35.
3. R. Negenborn: *Robot localisation and kalman filters: On finding your position in a noisy world.* Master's thesis, Utrecht University, 2003.
4. G. Welch, G. Bishop: *An introduction to the Kalman filter.* Technical Report TR 95-041, University of North Carolina at Chapel Hill, 2004.
5. W. Burgard, A. Derr, D. Fox, A.B. Cremers: *Integrating global position estimation and position tracking for mobile robots: The dynamic Markov localization approach.* In: Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)., 1998.
6. F. Dellaert, D. Fox, W. Burgard, S. Thrun: *Monte Carlo localization for mobile robots.* In: Proc. of the IEEE International Conference on Robotics & Automation (ICRA99), 1998.
7. E. Menegatti, M. Zoccarato, E. Pagello, H. Ishiguro: *Image-based Monte-Carlo localisation with omnidirectional images.* Robotics and Autonomous Systems **48**, 2004, 17–30.
8. D. Hähnel, W. Burgard: *Mapping and localization with rfid technology.* In: Proc. of the IEEE International Conference on Robotics & Automation (ICRA05), 2004, 1015–1020.
9. O. Wulf, M. Khalaf-Allah, B. Wagner: *Using 3D data for Monte Carlo localization in complex indoor environments.* In: 2nd Bi-Annual European Conference on Mobile Robots (ECMR05), 2005, 170–175.
10. S. Lenser, M. Veloso: *Sensor resetting localization for poorly modelled mobile robots.* In: Proc. of the IEEE International Conference on Robotics & Automation (ICRA00), 2000.
11. A. Mikulik, D. Obdrzalek, T. Petrusek, S. Basovnik, M. Dekar, P. Jusko, R. Pechal, R. Pitak: *Logion – a robot which collects rocks.* In: Proceedings of the EUROBOT Conference 2008, 276–287.
12. S.Y. Yi: *Global ultrasonic system with selective activation for autonomous navigation of an indoor mobile robot.* Robotica **26**, 3, 2008, 277–283.
13. L. Dazhai, F.H. Fu, W. Wei: *Ultrasonic based autonomous docking on plane for mobile robot.* In: IEEE International Conference on Automation and Logistics (ICAL 2008), 2008, 1396–1401.
14. S. Basovnik, L. Mach, A. Mikulik, D. Obdrzalek: *Detecting scene elements using maximally stable colour regions.* In: Proceedings of the EUROBOT Conference 2009.

Improved rate upper bound of collision resistant compression functions

Richard Ostertág*

Department of Computer Science, Faculty of Mathematics, Physics and Informatics,
Comenius University, Mlynská dolina, 842 48 Bratislava, Slovak Republic
ostertag@dcs.fmph.uniba.sk
http://www.dcs.fmph.uniba.sk

Abstract. *Based on Stanek's results [1] we know that in model with integer rate PGV like compression functions no high speed collision resistant compression functions exist. Thus we try to study more general multiple block ciphers based model of compression functions with rational rate, like 6/5. We show a new upper bound of the rate of collision resistant compression functions in this model.*

1 Motivation and goals

The cryptographic hash functions are a basic building block of many other cryptographic constructions (such as digital signature schemes, message authentication code, ...). For more complete overview see e.g. [2, 3].

Majority of modern hash functions is based on Merkle-Damgård paradigm [4, 5]. Many compression functions are explicitly based on block cipher. Even some of "dedicated" hash functions (which were not constructed in this way) have this structure. For example, it is possible to extract 160 bits block cipher with 512 bits key (called SHACAL-1) from compression function implemented in SHA-1 hash function [6].

The idea of hash function construction by iterating block cipher is at least 30 years old [7]. Nevertheless no systematic analysis of this idea was done until 1994. In this year Preneel, Govaerts and Vandewalle done the first systematic study of 64 hash functions based on block cipher [8]. Thereafter Black, Rogaway and Shrimpton [9] analyzed these constructions in black-box model and showed that 20 of them are collision resistant up to birthday-attack bound.

At least from the usability point of view, speed is important property of hash function. So it is only natural to attempt to speedups it. One of possible speedups of iterated hash functions based on block ciphers is increasing the number of input message blocks processed by one use of block cipher. Another possibility of speedup is a restriction of keys used in all block ciphers to a small fixed set of keys. Then it is possible to pre-schedule subkeys for each round of used block ciphers, whereby saving a big amount of work.

* Supported by VEGA grant No. 1/0266/09.

Traditional constructions [8] of hash functions require one block cipher transformation per input message block (so called rate-1 hash functions) and they require rekeying for every input message block. Black, Cochran and Shrimpton [10] showed in year 2005 that it is not possible to construct a provably secure rate-1 iterated hash function based on block cipher, which uses only small fixed set of keys.

For these reasons our goal is to maximize rate of iterated hash function based on block cipher. In other words, we attempt to maximize the number of input message blocks processed by a single block cipher invocation.

2 Notation

We now briefly introduce basic definitions and notations, following closely [10, 9].

Let V_m be set of all m-ary binary vectors, i.e. $V_m = \{0,1\}^m$. Let $V_m^* = (V_m)^*$ be set of all binary strings that we get by 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 $E : V_k \times V_n \to V_n$. Let denote E^{-1} the inverse of block cipher E.

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 will write the first argument (the block cipher) as superscript of the compression function, i.e. $f^E(\cdot,\cdot) = f(E,\cdot,\cdot)$. An iterated hash function based on 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 fixed element from V_a (so called initialization vector). Let $H^E(\varepsilon) = h_0$ for empty string ε. We often omit superscript E of functions f and H when it is apparent from the context which block cipher is used.

If the computation of $f^E(h,m)$ uses t queries on E, then compression function f (and its iterated hash function H) is rate-r, where $r = (b/n)/t$. Often b is

divisible by n. The rate r represents average number of input message blocks processed by a single enciphering transformation E. For example, if $b/n = 3$ and $t = 2$ then we get rate-$\frac{3}{2}$ compression function.

2.1 Black-box model

Black-box model (see e.g. [9]) is also known as ideal-cipher model. In this model, an adversary A is given access to oracles E and E^{-1}, where E is a block cipher. We write the oracles as superscripts, i.e. $A^{E,E^{-1}}$. Where used oracles are clear from the context, the superscript of A will be omitted.

Adversary A tries to find collisions in the compression function. Other cryptographic properties of compression functions are also important, but we focus exclusively on collision resistance, as on the most "problematic" property of compression functions. We will see that our results are negative, so it is not necessary to analyze other properties.

In the black-box model the adversary's collision finding effort is measured by the number of queries made to oracles E and E^{-1}. Computational power of the adversary is not limited in any way — i.e. we assume information-theoretic adversary.

Attacks in this model treat the block cipher as a black-box. The only modeled structural property of the block cipher is its invertibility. This model cannot guarantee security of compression functions based on weak block ciphers with inappropriate properties (such as weak keys). On the other hand, black-box model is stronger than model in which block cipher is assumed to be random function, because adversary can compute E^{-1}.

We say that inputs (h, m) and (h', m') of compression function f collide, if they are distinct and $f^E(h, m) = f^E(h', m')$. We say that (h, m) collides with empty string, if $f^E(h, m) = h_0$, where h_0 is initialization vector.

We write random draw of element x from finite set S as $x \xleftarrow{\$} S$. We will use notation $(x, y) \leftarrow A^{E,E^{-1}}$ for computation of two colliding inputs x and y by adversary A (represented by probabilistic algorithm) with knowledge of oracles E and E^{-1}.

Definition 1 (Coll. res. of comp. function [9]). *Let f be 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 adversary A (denoted by $\mathbf{Adv}_f^{\mathrm{comp}}(A)$) in finding collisions in compression function f is the following probability:*

$$\Pr\Big[E \xleftarrow{\$} \mathrm{Bloc}(k, n);\ \big((h, m), (h', m')\big) \leftarrow A^{E,E^{-1}} : \big((h, m) \neq (h', m') \wedge f^E(h, m) = f^E(h', m')\big) \vee f^E(h, m) = h_0\Big].$$

For any $q \geq 0$ 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 oracles (E or E^{-1}) at most q queries.

Definition 2 (Collision resistance of hash function [9]). *Let H be hash function based on block cipher. Let A be an adversary. Then the advantage of the adversary A in finding collisions in hash function H is the following 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 any $q \geq 0$ 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 oracles (E or E^{-1}) at most q queries.

The Merkle-Damgård construction of iterated hash functions is based on the following theorem. It states that iterated hash function is collision resistant if underlying compression function is collision resistant.

Theorem 1 (Merkle-Damgård [4, 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 function of f. Then $\mathbf{Adv}_H^{\mathrm{coll}}(q) \leq \mathbf{Adv}_f^{\mathrm{comp}}(q)$ for any $q \geq 1$.*

Birth-day attack is generic way of attacking collision resistance of any compression or hash function. The advantage of finding collision by applying birth-day attack is $\Theta(q^2/2^n)$, where q is number of evaluation of the function and n is output length.

If q depends on n, then we assume that $q(n) = o(2^{n/2})$, because greater $q(n)$ does not make sense, as we can still use generic birth-day attack with lower $q(n) = 2^{n/2}$ with unacceptably high probability ($\approx 1/2$) of finding collision.

Compression function f (or hash function H) is usually called collision resistant up to birthday attack bound or simply collision resistant if $\mathbf{Adv}_f^{\mathrm{comp}}(q) = O(q^2/2^n)$ (or $\mathbf{Adv}_H^{\mathrm{coll}}(q) = O(q^2/2^n)$). Since birth-day attack is always possible, we can rewrite these equations into equivalent form $\mathbf{Adv}_f^{\mathrm{comp}}(q) = \Theta(q^2/2^n)$ (or $\mathbf{Adv}_H^{\mathrm{coll}}(q) = \Theta(q^2/2^n)$).

3 Known results

In [11] we have proposed a model of rate-r compression functions that cover all compression functions that

process r input message blocks of length n per block cipher invocation with a key of length k. In that paper we have showed that $1 + k/n$ is the upper bound of rate of any collision resistant compression function in such a model.

For typical constructions, when $k = n$, we get that if any high-rate collision resistant function in our model exists, then it is rate-2 compression function.

Consequently we have analyzed in [11] all rate-2 generalizations of compression functions from [8] (all of them are covered by our model). We have proved that none of them is collision resistant in the black-box model. Staneková and Stanek showed in [12] that either hash functions constructed from them are not collision resistant.

But these functions does not cover whole set of rate-2 compression functions from our model. Hence the question, if there exist any rate-2 collision resistant compression function still remains open.

This question is answered by Stanek in [1], where he improves our upper bound by utilizing the possibility of asking q queries during the attack (before the adversary ask only one query).

Theorem 2 (Stanek [1]). *Let $E \in \mathrm{Bloc}(k, n)$. Let $f^X : V_a \times V_{rn} \to V_n$, $f^K : V_a \times V_{rn} \to V_k$ and $f^C : V_a \times V_{rn} \times V_n \to V_a$ be arbitrary functions. Let $f : V_a \times V_{rn} \to V_a$ be compression function defined by $f(h, m) = f^C\big(h, m, E_{f^K(h,m)}(f^X(h, m))\big)$. Let $q \geq 1$ denote maximum number of queries on E and E^{-1}. Let $r > 1 + \frac{k - \log_2 q}{n}$. Then $\mathbf{Adv}_f^{\mathrm{comp}}(q) = 1$.*

By substituting $q = n$, $a = n$ and $k = n$ into theorem 2 we get upper bound for rate r in the following form $r > 2 - \frac{\log_2 n}{n}$. If we take into account that in our model rate r is always an integer, then we get following corollary of previous theorem.

Corollary 1 (Stanek [1]). *Let $E \in \mathrm{Bloc}(n, n)$. Let $f^X : V_n \times V_{rn} \to V_n$, $f^K : V_n \times V_{rn} \to V_n$ and $f^C : V_n \times V_{rn} \times V_n \to V_n$ be arbitrary functions. Let function $f : V_n \times V_{rn} \to V_n$ be a compression function defined by $f(h, m) = f^C\big(h, m, E_{f^K(h,m)}(f^X(h, m))\big)$. Let $r > 1$. Then f is not collision resistant in black-box model.*

Now our result about nonexistence of rate-2 collision resistant PGV-like compression functions from [11] follows from corollary 1. But the attack based on theorem 2 has exponential time complexity (and asks n oracle queries, even if it is not necessary). Therefore our attacks from [11] constructed specifically for rate-2 PGV-like compression and hash functions are still justified as they use only polynomial time and ask at most two queries.

Until now we have not modeled any compression function which uses more block ciphers per one compression function computation. For example:

$$f(h, m) = f^C\Big(h, m, E_1\big(f_1^K(h, m), f_1^X(h, m)\big), E_2\big(f_2^K(h, m), f_2^X(h, m)\big)\Big) \ .$$

If m is created from four input message blocks, then this will be rate-2 compression function but is not covered by model from [11]. Also using of multiple block ciphers allows compression functions with rational rate. For example, if m is created from three input message blocks, then we get rate-$\frac{3}{2}$ compression function. Therefore we have concentrated on creation of new more general model.

4 The generalized model of compression function

A compression function f based on t block ciphers[1] is function defined by $f : \mathrm{Bloc}(k, n)^t \times V_a \times V_b \to V_c$, where a, b and c are positive integers such that $a + b \geq c$. When we will need to emphasize number of used block ciphers t, then we will write t as superscript of compression function, i.e. f^t. Iterated hash function based on compression function $f : \mathrm{Bloc}(k, n)^t \times V_a \times V_b \to V_a$ is function $H : \mathrm{Bloc}(k, n)^t \times V_b^* \to V_a$ defined by $H((E_1, \ldots, E_t), m_1 \ldots m_l) = h_l$, where $h_i = f((E_1, \ldots, E_t), h_{i-1}, m_i)$ and h_0 is fixed element from V_a. We define $H((E_1, \ldots, E_t), \varepsilon)$ to be equal to h_0. If block ciphers used in functions f and H are clear from the context, then we will omit them as arguments of these functions.

Now we will start to define the general model of compression function $f^t : \mathrm{Bloc}(k, n)^t \times V_a \times V_b \to V_a$ based on t block ciphers. Model is based on following assumptions:

- Computation of compression function f^t asks exactly one query on each oracle E_i for the purpose of evaluation of $f^t(h, m)$.
  This assumption is without loss of the generality. We do not assume that in practice all $E_1, \ldots, E_t$ are distinct, but the model allows it. If for computation of function f^t we need to evaluate E_i, e.g. two times, then we can set $E_{t+1} = E_i$ and use function f^{t+1} defined analogically as f^t but with the only exception, that in place of second evaluation of E_i evaluation of E_{t+1} will be used.

[1] As we will clarify in following paragraph, it is important that t queries on oracles are made during each evaluation of compression function f. It does not matter, if the same block cipher is invoked t times, or if t different block ciphers are invoked exactly once. Hence some of t block ciphers can be equal.

Analogically, it does not make sense to specify block cipher E_i if it is not used during any calculation of compression function f.
This assumption about f^t guarantees that every computation of f^t always asks exactly t queries on oracles.
- Computation of compression function f^t asks oracles E_i in order of their indexes[2]. Thus we can assume that evaluation of block cipher E_i had to occur before evaluation of E_{i+1}.
- The length of input message block m_i of compression function does not have to be divisible by block cipher E plain text block length n.

In following text we will often work with sequences, therefore we now clarify some necessary notation.

Definition 3. *Empty sequence will be denoted by* $()$. *We will write* $(a_1, a_2, \ldots, a_n)$ *for a sequence with* n *elements* a_1, a_2, …, a_n. *Sequences will be denoted by upper case letters with a overscore, for example* $\overline{Y}$. *For addition of element* a_{n+1} *at the end of a sequence* $(a_1, a_2, \ldots, a_n)$ *we will use operation "* $\cdot$ *" in the following way:* $(a_1, a_2, \ldots, a_n) \cdot a_{n+1} = (a_1, a_2, \ldots, a_n, a_{n+1})$.

Let for all $i \in \{1, 2, \ldots, t\}$ $f_i^X : V_a \times V_b \times V_n^{i-1} \to V_n$ and $f_i^K : V_a \times V_b \times V_n^{i-1} \to V_k$ be arbitrary functions. Let $f^C : V_a \times V_b \times V_n^t \to V_a$ be arbitrary function. Computation of compression function $f^t : \mathrm{Bloc}(k,n)^t \times V_a \times V_b \to V_a$ in generalized model is defined by the following algorithm 1.

Algorithm 1 The gen. model of compression function

```
1: function f((E_1, ..., E_t); h; m)
2:     Ȳ_0 = ()
3:     for i = 1 to t do
4:         X_i ← f_i^X(h, m, Ȳ_{i-1})
5:         K_i ← f_i^K(h, m, Ȳ_{i-1})
6:         Y_i ← E_i(K_i, X_i)
7:         Ȳ_i ← Ȳ_{i-1} · Y_i
8:     end for
9:     return f^C(h, m, Ȳ_t)
10: end function
```

Remark 1. Function f_i^X, respective function f_i^K prepares the plain text, respective the key for the block cipher E_i. Both inputs h and m are arguments of these functions together with all already computed cipher texts $Y_1, Y_2, \ldots, Y_{i-1}$. At the end of the algorithm, function f^C processes both inputs h and m with all intermediate results $Y_1, Y_2, \ldots, Y_t$ into final result. The algorithm uses t functions f_i^X and t functions f_i^K. But function f^C is just one. Introduction of analogous "postprocessing" for every block cipher (i.e. for each round) is needless. Calculation of local postprocessing at the end of i-th round can be incorporated into functions f_j^X and f_j^K of following rounds (i.e. for all $j > i$) and into function f^C.

The compression function f^t (and its iterated hash function H) have rate $r = (b/n)/t$.

This generalized model of compression functions covers all compression functions, which takes messages of length a and b and process them using exactly t block ciphers $E_1, E_2, \ldots, E_t$ from $\mathrm{Bloc}(k,n)$ in this specified order, into message of length a. All rate-1 schemes from [8] and their rate-2 generalizations fall into this model.

5 Upper bound of rate of collision resistant compression functions

Before proof of the upper bound we first define some auxiliary notions and prove some lemmas.

Definition 4. *Let* $i \in \{0, 1, \ldots, t\}$ *and* $(h, m) \in V_a \times V_b$. *If* $i = 0$ *then* $\overline{Y}_{i,(h,m)} = ()$. *If* $i > 0$ *then we define* $\overline{Y}_{i,(h,m)}$ *recursively as follows:*

$$\overline{Y}_{i-1,(h,m)} \cdot E_i\Big(f_i^K\big(h, m, \overline{Y}_{i-1,(h,m)}\big), f_i^X\big(h, m, \overline{Y}_{i-1,(h,m)}\big)\Big) \ .$$

Sequence $\overline{Y}_{i,(h,m)}$ represents individual Y_i calculated during individual rounds of $f^t(h,m)$ evaluation. It can easily be seen that $\overline{Y}_{i-1,(h,m)}$ is prefix of $\overline{Y}_{i,(h,m)}$ and that $\overline{Y}_{t,(h,m)}$ is equal to $\overline{Y}_t$, which is created during evaluation of compression function $f^t(h,m)$.

Definition 5. *Let* $i \in \{1, 2, \ldots, t\}$, $X \in V_n$, $K \in V_k$ *and let* $2^{n+k} > \alpha > 0$ *be an integer. Let* $S \subseteq V_a$, *where* $|S| = s > 0$. *Then* $D_\alpha^0 = S \times V_b$ *and* D_α^i *is union of* α *largest sets* $D_{X,K}^i$ *taken through all* X *and* K *(let denote them* $D^i_{X_1^i,K_1^i}, \ldots, D^i_{X_\alpha^i,K_\alpha^i}$*), where* $D_{X,K}^i$ *is defined as follows:*

$$D_{X,K}^i = \Big\{(h,m) \in D_\alpha^{i-1} \,\Big|\, f_i^X\big(h, m, \overline{Y}_{i-1,(h,m)}\big) = X \wedge f_i^K\big(h, m, \overline{Y}_{i-1,(h,m)}\big) = K\Big\} \ .$$

Set $D_{X,K}^i$ is subset of D_α^{i-1}. It consists of those elements, which in next (i-th) round will lead to the same query $E_i(X,K)$ on oracle E_i. That means that to compute next round for all elements from $D_{X,K}^i$ one oracle

[2] Requirement of fixed evaluation order of block ciphers is not so restrictive as it can seem. We can simulate compression function with variable evaluation order of t block ciphers by compression function with fixed evaluation order of t^2 block ciphers. See e.g. discussion at the end of section 2 in [13].

query is sufficient. Construction of sets $D^i_{X,K}$ have of course exponential complexity, but does not require any oracle queries. Since we use black-box model, adversary have computationally unlimited power and is limited only by number of oracle queries.

Set D^1_α is the largest set of tuples $(h,m) \in S \times V_b$, for which we can made first round of compression function f^t with spending exactly α queries on oracle E_1. By definition D^1_α is union of α largest sets $D^1_{X^1_1,K^1_1}, \ldots, D^1_{X^1_\alpha,K^1_\alpha}$. For the calculation of the first round for elements from every set $D^1_{X^1_j,K^1_j}$ we need one query $E_1(X^1_j, K^1_j)$ on oracle E_1. Since all tuples (X^1_j, K^1_j) are distinct, we need exactly α queries for selected α sets.

We do not know how to estimate cardinality of set, which is the largest set of tuples $(h,m) \in S \times V_b$, for which we can do first two rounds of compression function f with at most 2α queries on oracles E_1 and E_2. However we know how to estimate cardinality of set D^2_α, which is such largest set of tuples $(h,m) \in D^1_\alpha$. Therefore we have constructed set D^i_α as subset of D^{i-1}_α. Then we are able to lower bound cardinality of set D^i_α in following way.

Lemma 1. *Let $1 \le \alpha \le 2^{n+k}$ and let $0 \le i \le t$ be integers. Then $|D^i_\alpha| \ge \alpha^i 2^{b-i(n+k)} s$.*

Proof. (Using mathematical induction over i.)
IND. BASIS: $|D^0_\alpha| = |S \times V_b| = s2^b \ge \alpha^0 2^{b-0(n+k)} s$.
IND. HYPOTHESIS: Let $|D^i_\alpha| \ge \alpha^i 2^{b-i(n+k)} s$.
IND. STEP: Then $|D^{i+1}_\alpha| \ge \alpha^{i+1} 2^{b-(i+1)(n+k)} s$.

Set $|D^{i+1}_\alpha|$ is by definition 5 union of $\alpha < 2^{n+k}$ largest sets $D^{i+1}_{X,K}$. Nonempty sets $D^{i+1}_{X,K}$ are all distinct and their union is equal to D^i_α. In other words, elements of the set D^i_α are divided into 2^{n+k} shelves. Then using pigeonhole principle we can estimate cardinality of α largest of them in the following way:

$$|D^{i+1}_\alpha| \ge \alpha \frac{|D^i_\alpha|}{2^{n+k}} \ge \\ \ge \alpha \frac{\alpha^i 2^{b-i(n+k)} s}{2^{n+k}} = \alpha^{i+1} 2^{b-(i+1)(n+k)} s \ .$$

□

Lemma 2. *At most $t\alpha$ queries on oracles $E_1, \ldots, E_t$ are sufficient for computation of set D^t_α among with values of compression function $f^t(h,m)$ for all tuples (h,m) from the set D^t_α.*

Proof. We construct matrix M, which has on i-th row tuples $(X^i_1, K^i_1) \ldots (X^i_\alpha, K^i_\alpha)$ used during the construction of set D^i_α by taking the union of α largest sets $D^i_{X^i_1,K^i_1}, \ldots, D^i_{X^i_\alpha,K^i_\alpha}$. M has t rows and α columns, so matrix M have totally $t\alpha$ elements.

$$M = \begin{pmatrix} (X^1_1,K^1_1) & \ldots & (X^1_j,K^1_j) & \ldots & (X^1_\alpha,K^1_\alpha) \\ \vdots & & \vdots & & \vdots \\ (X^i_1,K^i_1) & \ldots & (X^i_j,K^i_j) & \ldots & (X^i_\alpha,K^i_\alpha) \\ \vdots & & \vdots & & \vdots \\ (X^t_1,K^t_1) & \ldots & (X^t_j,K^t_j) & \ldots & (X^t_\alpha,K^t_\alpha) \end{pmatrix}$$

The only place where queries are made during the computation of sets D^i_α is the computation of $\overline{Y}_{i,(h,m)}$. During the construction of set D^0_α no queries on oracles are necessary as it is $S \times V_b$ by definition. Similarly during the construction of set D^1_α no queries on oracles are necessary as $\overline{Y}_{0,(h,m)}$ is by definition empty.

During the construction of D^i_α for $i \in \{2,3,\ldots,t\}$ all queries will be on oracles $E_1, \ldots, E_{i-1}$. Queries on oracle E_1 will be only from the first row of matrix M, queries on oracle E_2 will be only from the second row, and so on, ending with queries on oracle E_{i-1}, which are only from $(i-1)$-th row of matrix M. Last row of matrix M (together with all others) is used during the computation of values $f^t(h,m) = f^C(h,m,\overline{Y}_{t,(h,m)})$ for all $(h,m) \in D^t_\alpha$.

During the computation of D^i_α a new i-th row is created in the matrix M. Tuples (X^{i-1}_j, K^{i-1}_j) from $(i-1)$-th row are for the first time evaluated by oracle E_{i-1}. Queries on oracle E_l for $l < i-1$ will be only from already evaluated row l of matrix M. That follows from the fact that $D^i_\alpha \subseteq D^{i-1}_\alpha$. Therefore we will need at most $t\alpha$ queries on oracles $E_1, \ldots, E_t$ during the computation of D^t_α together with values of compression function $f^t(h,m)$ for all $(h,m) \in D^t_\alpha$ if we remember already asked queries together with corresponding answer. □

Theorem 3. *Let $f : \mathrm{Bloc}(k,n)^t \times V_a \times V_b \to V_a$ be arbitrary rate-r compression function defined by algorithm 1, while $r = \frac{b/n}{t}$. Let $q \ge 1$ be maximum allowed number of queries on oracles E_i and E_i^{-1}. Let q be an integer of the form $q = t\alpha$, where $\alpha \ge 1$ is also an integer*[3]*. Let $r > 1 + \frac{k}{n} - \frac{\log_2 \alpha}{n}$. Then $\mathbf{Adv}^{\mathrm{comp}}_f(q) = 1$.*

Proof. By asking at most q queries we are according to lemma 2 able to compute values of $f^t(h,m) \in V_a$ for all $(h,m) \in D^t_\alpha$. Let $S = V_a$, thus $s = |S| = 2^a$. According to lemma 1 we know that $|D^t_\alpha| \ge \alpha^t 2^{b-t(n+k)} s = \alpha^t 2^{a+b-t(n+k)}$. We can guarantee that between computed values there are at least two identical values if:

$$\alpha^t 2^{a+b-t(n+k)} > 2^a$$
$$t \log_2 \alpha + a + b - t(n+k) > a$$
$$b > t(n+k) - t \log_2 \alpha \ .$$

[3] This requirement is natural. For computation of f^t we need t oracle queries. Hence if we set $q = t\alpha$, then as if we allow α complete computations of f^t.

Now we rewrite this inequality into required form by using following equality $r = \frac{b/n}{t}$:

$$\begin{aligned} b &> t(n+k) - t\log_2 \alpha \\ \frac{b/n}{t} &> \frac{n+k}{n} - \frac{\log_2 \alpha}{n} \\ r &> 1 + \frac{k}{n} - \frac{\log_2 \alpha}{n} \quad . \end{aligned}$$

This means that with probability 1 we can find (and so the adversary) collision in the compression function f, while asking at most q queries on oracles. Hence $\mathbf{Adv}_f^{\mathrm{comp}}(q) = 1$ holds. □

Computation of the particular D_α^i has exponential complexity. Also finding the collision between values $f^t(h,m)$ for all $(h,m) \in D_\alpha^t$ has exponential complexity. But computationally unlimited adversary of black-box model can do all this unless he does not ask more than q queries on oracles.

Theorem 3 gives upper bound depending on number of oracle queries. The following corollary adapts the previous theorem in such a way, that instead of number of queries q, the number of output bits a of compression function is used in the inequality for r.

Corollary 2. *Let $f^t : \mathrm{Bloc}(k,n)^t \times V_a \times V_b \to V_a$ be arbitrary rate-r compression function defined by algorithm 1, where $r = \frac{b/n}{t}$. Let $0 \le \varepsilon < \frac{1}{2}$ be arbitrary constant. Let $r > 1 + \frac{k}{n} - \varepsilon\frac{a}{n}$. Then f^t is not collision resistant.*

Proof. Let $0 \le \lambda < 1$ be arbitrary constant. Then we set $q = t2^{\lambda\frac{a}{2}}$, i.e. $\alpha = 2^{\lambda\frac{a}{2}}$. Then by substituting into theorem 3 we get that $\mathbf{Adv}_f^{\mathrm{comp}}(q) = 1$ (that means according to size of q that f^t is not collision resistant) if:

$$\begin{aligned} r &> 1 + \frac{k}{n} - \frac{\log_2 \alpha}{n} \\ r &> 1 + \frac{k}{n} - \frac{\log_2 2^{\lambda\frac{a}{2}}}{n} \\ r &> 1 + \frac{k}{n} - (\lambda/2)\frac{a}{n} \quad . \end{aligned}$$

Now we make a substitution $\varepsilon = \lambda/2$ and required inequality follows:

$$r > 1 + \frac{k}{n} - \varepsilon\frac{a}{n}, \text{ where } 0 \le \varepsilon < \frac{1}{2} \quad .$$

□

As we have already mentioned, constructions of compression function based on block cipher, often have the same size of the key and the plain-text input of block cipher, i.e. $k = n$. Similarly, the output of compression function have usually the same size, i.e. $a = n$. For this typical situation we can simplify corollary 2.

Corollary 3. *Let $f^t : \mathrm{Bloc}(n,n)^t \times V_n \times V_b \to V_n$ be arbitrary rate-r compression function defined by algorithm 1, where $r = \frac{b/n}{t}$. Let $r > 3/2$. Then compression function f^t is not collision resistant.*

Proof. After substituting n for a and k into corollary 2 we get that compression function f^t is not collision resistant if $r > 1 + \frac{n}{n} - \varepsilon\frac{n}{n} = 2 - \varepsilon$ for an arbitrary constant $0 \le \varepsilon < \frac{1}{2}$.

That implies that compression function f^t is not collision resistant if $r > 2 - \frac{1}{2} = 3/2$. □

In generalized model rate r of compression function can be rational number and not only integer as in [11]. Therefore based on our results we cannot conclude that no high rate compression function exists in the generalized model. Still it is possible that e.g. rate-$\frac{6}{5}$ collision resistant compression function exists.

6 Conclusion

In our effort to find high speed collision resistant compression function we have introduced and studied new generalized model of compression function since in all previous models it was proved that no such functions exists. This model introduces rational rates, so we can study more precisely the rate upper bound of collision resistant compression functions. Based on previous results, it seems to be less than or equal to 2. We have improved this bound to be less than or equal to $\frac{3}{2}$.

References

1. M. Stanek: *Analysis of fast blockcipher-based hash functions.* In: Computational Science and Its Applications – ICCSA 2006, Springer, 2006, 426–435.
2. D.R. Stinson: *Cryptography: Theory and Practice*, Third Edition. Chapman & Hall/CRC, Boston, MA, USA, 2005.
3. A.J. Menezes, P.C. van Oorschot, S.A. Vanstone: *Handbook of Applied Cryptography.* CRC-Press, Boca Raton, FL, USA, 1996.
4. R.C. Merkle: *One way hash functions and DES.* Volume 435 of Lecture Notes in Computer Science, Springer Berlin, Heidelberg, 1990, 428–446.
5. I.B. Damgård: *A design principle for hash functions.* Volume 435 of Lecture Notes in Computer Science, Springer Berlin, Heidelberg, 1990, 416–427.
6. H. Handschuh, L.R. Knudsen, M.J. Robshaw: *Analysis of SHA-1 in encryption mode.* Volume 2020 of Lecture Notes in Computer Science, Springer Berlin, Heidelberg, 2001, 70–83.
7. M.O. Rabin: *Digitalized signatures.* In Millo R.D., Dobkin D., Jones A., Lipton R., eds.: Foundations of Secure Computations, New York, Academic Press, 1978, 155–166.

8. B. Preneel, R. Govaerts, J. Vandewalle: *Hash functions based on block ciphers: A synthetic approach.* Volume 773 of Lecture Notes in Computer Science, Springer Berlin, Heidelberg, 1994, 368–378.
9. J. Black, P. Rogaway, T. Shrimpton: *Black-box analysis of the block-cipher-based hash-function constructions from PGV.* Volume 2442 of Lecture Notes in Computer Science, Springer Berlin, Heidelberg, 2002, 103–118.
10. J. Black, M. Cochran, T. Shrimpton: *On the impossibility of highly-efficient blockcipher-based hash functions.* Volume 3494 of Lecture Notes in Computer Science, Springer Berlin, Heidelberg, 2005, 526–541.
11. R. Ostertág, M. Stanek: *On high-rate cryptographic compression functions.* Computing and Informatics **26**, 2007, 77–87.
12. L. Staneková, M. Stanek: *Generalized PGV hash functions are not collision resistant.* In: ITAT: Information Technologies – Applications and Theory, Seòa: PONT, 2006, 139–143.
13. P. Rogaway, J. Steinberger: *Security/efficiency tradeoffs for permutation-based hashing.* Volume 4965 of Lecture Notes in Computer Science, Springer Berlin, Heidelberg, 2008, 220–236.

Encoding monadic computations in C# using iterators

Tomas Petricek

Charles University in Prague, Faculty of Mathematics and Physics
tomas@tomasp.net

Abstract. *Many programming problems can be easily solved if we express them as computations with some non-standard aspect. This is a very important problem, because today we're struggling for example to efficiently program multi-core processors and to write asynchronous code. Unfortunately main-stream languages such as Java or C# don't support any direct way for encoding unrestricted non-standard computations. In languages like Haskell and F#, this can be done using monads with syntactic extensions they provide and it has been successfully applied to a wide range of real-world problems. In this paper, we present a general way for encoding monadic computations in the C# 2.0 language with a convenient syntax using an existing language feature called iterators. This gives us a way to use well-known non-standard computations enabling easy asynchronous programming or for example the use of software transactional memory in plain C#. Moreover, it also opens monads in general to a wider audience which can help in the search for other useful and previously unknown kinds of computations.*

1 Introduction

In functional programming languages such as Haskell and F#, monadic computations are used to solve wide range of problems. In Haskell [5], they are frequently used to deal with state or I/O, which is otherwise difficult in a purely functional language. F# uses monadic computations to add non-standard aspects such as asynchronous evaluation, laziness or implicit error handling to an existing piece of code written in F#. In this article, we'll prefer the F# point of view, meaning that we want to be able to adjust C# code to execute differently, using additional aspects provided by the monadic computation.

The primary motivation for this work is that monadic computations are very powerful technique for dealing with many modern computing challenges caused by the rise of multi-core processors and distributed web based applications. The standard F# library uses monadic computations to implement asynchronous workflows [16] which make it easy to write communication with the web and other I/O operations in the natural sequential style, but without blocking threads while waiting. In Haskell, monadic computations are used for example to implement software transactional memory, which is a concurrent programming mechanism based on shared memory, which avoids the need for explicit locking [3].

The motivation for this article is that we want to be able to use the techniques just described in a main-stream and widely used C# language. The main contributions of this paper are following:

- As far as we're aware, we show for the first time that monadic computations can be encoded in C# in a syntactically convenient way without placing any restrictions on the C# statements that can be used inside the computation. This can be done purely as a library without changing the language using widely adopted C# 2.0 features (Section 3).
- We use the presented technique to implement a library that makes it easier to write scalable multi-threaded applications that perform long running I/O operations. We demonstrate it using several case study examples (Section 4).
- Finally, we describe a method for systematical encoding of arbitrary monadic computation in C# (Section 5). This technique can be used for implementing other useful computations such as software transactional memory and others.

There are several related projects, mostly concerned with asynchronous programming (Section 6), but our aim is wider and focuses on monadic computations in general. However, asynchronous computations can nicely demonstrate the problem.

1.1 Asynchronous computations in C# today

Since we're using asynchronous computations as the primary real-world motivation for this paper, we should first clarify what problem we want to solve is. Let's start by looking at naive synchronous code that downloads the first kilobyte of web site content:

```
1: var req = HttpWebRequest.Create(url);
2: var rsp = req.GetResponse();
3: var strm = rsp.GetResponseStream();
4: var read = strm.Read(buffer, 0, 1024);
```

On lines 2 and 4 we're performing I/O operations that can take a long time, but that aren't CPU bounded. When running the operation, the executing thread will be blocked, but it cannot perform any other

work in the meantime. If we wanted to run hundreds of downloads in parallel, we could create hundreds of threads, but that introduces significant overheads (such as allocation of kernel objects and thread stack) and also increases context switching. The right way to solve the problem on the .NET platform is to use the *Asynchronous Programming Model* (APM):

```
1: var req = HttpWebRequest.Create(url);
2: req.BeginGetResponse(a1 => {
3:   var rsp = req.EndGetResponse(a1);
4:   var strm = rsp.GetResponseStream();
5:   strm.BeginRead(buffer, 0, 1024, a2 => {
6:     int read = strm.EndRead(a2);
7:     // ...
8:   }, null);
9: }, null);
```

In this context "asynchronous" means that the program invokes start of the operation, registers a callback, transfers the control to the system and releases the current thread, so that it can perform other work. In the snippet above, we're starting two operations on lines 2 and 5 and we're using the C# 3.0 lambda function notation "=>" to specify the callback function that will be eventually invoked.

The code above is far less readable than the first synchronous version, but that's not the only problem. To download the whole page, we'd need to call the `BeginRead` method in a loop until we fetched the whole page, but that's ridiculously difficult, because we can't use any higher level constructs such as the `while` loop when writing code using nested callbacks. For every simple problem, the programmer has to explicitly write a state machine using mutable state.

It is worth pointing out that using asynchronous model does not in principle increase the CPU parallelism in the application, but it still significantly improves the performance and makes the application more scalable because it considerably reduces the number of (expensive) threads the application creates.

2 Background

To write non-blocking asynchronous code, we can use *continuation passing style* where the next piece of code to execute after an operation completes is given as a function as the last argument to the operation. In the snippet above we've written the code in this style explicitly, but as we've seen this isn't a satisfying solution.

2.1 F# asynchronous workflows

In F#, we can use *asynchronous workflows*, which is one particularly useful implementation of monadic computations that is already defined in F# libraries. This feature hasn't been described in the literature before, so we quickly review it here.

When we wrap code inside an `async` block, the compiler automatically uses continuation passing style for specially marked operations. Moreover, we can use all standard language constructs inside the block including for example the `while` loop:

```
1: let downloadUrl(url:string) = async {
2:  let req = HttpWebRequest.Create(url)
3:  let! rsp = req.AsyncGetResponse()
4:  let strm = rsp.GetResponseStream()
5:  let buffer = Array.zeroCreate(8192)
6:  let state = ref 1
7:  while !state > 0 do
8:   let! read = strm.AsyncRead(buffer,0, 8192)
9:   Console.WriteLine("got {0}b", read);
10:  state := read }
```

This function downloads the entire content of a web page in a loop. Although it doesn't use the data in any way and only reports the progress, it nicely demonstrates the principle. Its body is an `async` block, which specifies that the function doesn't actually run the code, but instead returns a value representing computation that can be executed later.

In the places where the original C# code executed asynchronous operations, we're now using the `let!` Keyword (lines 3 and 8), which represents monadic value binding. This means that instead of simply assigning value to a symbol, the computation invokes `Bind` operation that is provided by the async value (called *computation builder*) giving it the rest of the code wrapped inside a function as an argument. The `Bind` member specifies non-standard behavior of the operation. In this case the behavior is that the operation is executed asynchronously.

The computation builder (in this case `async`) also defines the meaning of other primitive constructs such as the while loop or returning the result from a function. These primitive operations are exposed as standard methods with well-defined type signatures:

```
Bind   : Async<'a> * ('a -> Async<'b>) ->
         Async<'b>
Return : 'a -> Async<'a>
While  : (unti -> bool) * Async<unit> ->
         Async<unit>
```

The first two functions are standard operations that define the abstract monadic type as first described in [18]. These operations are also called *bind* and *unit*. The `Bind` member takes an existing computation and a function that specifies how to produce subsequent computation when the first one completes and composes these into a single one. The `Return`

member builds a computation that returns the given value. The additional `While` member takes a predicate and a computation and returns result that executes the computation repeatedly while the predicate holds.

When compiling code that uses computation expressions, the F# compiler syntactically transforms the code into code composed from the calls to these primitive operations. The translated version of the previous example can be found in the online supplementary material for the article [12].

2.2 C# Iterators

One of the non-standard computations that is very often used in practice is a computation that generates a sequence of values instead of yielding just a single result. This aspect is directly implemented by C# iterators [2], but without any aim to be more generally useful. In this article, we show that it can be used in a more general fashion. However, we start by briefly introducing iterators. The following example uses iterators to generate a sequence of all integers:

```
1: IEnumerator<int> GetNumbers() {
2:   int num = 0;
3:   while (true) {
4:     Console.WriteLine("generating {0}", num);
5:     yield return num++;
6:   }
7: }
```

The code looks just like ordinary method with the exception that it uses the `yield return` keyword to generate elements a sequence. The `while` loop may look like an infinite loop, but due to the way iterators work, the code is actually perfectly valid and useful. The compiler translates the code into a state machine that generates the elements of the sequence lazily one by one. The returned object of type `IEnumerator<int>` can be used in the following way:

```
1: var en = GetNumbers();
2: en.MoveNext();
3: Console.WriteLine("got {0}", en.Current);
4: en.MoveNext();
5: Console.WriteLine("got {0}", en.Current);
```

The call to the `GetNumbers` method (line 1) returns an object that represents the state machine generated by the compiler. The variables used inside the method are transformed into a local state of that object. Each call to the `MoveNext` method (lines 2 and 4) runs one step of the state machine until it reaches the next yield return statement (line 5 in the earlier snippet) updating the state of the state machine. This also executes all side-effects of the iterator such as printing to the console, so the program above shows the "generating" message directly followed by "got" for numbers 0 and 1. There are two key aspects of iterators that are important for this paper:

- The iterator body can contain usual control structures such as loops or exception handlers and the compiler automatically turns them into a state machine.
- The state machine can be executed only to a certain point (explicitly specified by the user using `yield return`), then paused and later resumed again by invoking the `MoveNext` method again.

In many ways this resembles the continuation passing style from functional languages, which is essential for monadic computations and F# asynchronous workflows.

3 Monadic computations in C#

Now that we've introduced asynchronous workflows in F# (as an example of monadic computations) and C# iterators, we can ask ourselves the question whether iterators could be used for encoding other non-standard computations then code that generates a sequence.

The key idea of this article is that it is indeed possible to do that and that we can write standard C# library to support any monadic computation. In this section, we'll briefly introduce how the library looks using the simplest possible example and in section 5 we'll in detail explain how the encoding works.

3.1 Using option computations

As the first example, we'll use computations that produce value of type `Option<'a>` [1], which can either contain no value or a value of type `'a`. The type can be declared using F#/ML notation like this:

```
type Option<'a> = Some of 'a | None
```

Code that is composed from simple computations that return this type can return `None` value at any point, which bypasses the entire rest of the computation. In practice this is useful for example when performing series of data lookup that may not contain the value we're looking for. The usual way for writing the code would check whether the returned value is `None`

[1] In Haskell, this type is called `Maybe` and the corresponding computation is known as Maybe monad.

after performing every single step of the computation, which significantly complicates the code [2].

To show how the code looks when we apply our encoding of the option computation using iterators, we'll use method of the following signature:

```
ParseInt : string -> Option<int>
```

The method returns `Some(n)` when the parameter is a valid number and otherwise it returns `None`. Now we can write code that reads a string, tries to parse it and returns 10 times the number if it succeeds. The result of the computation will again be the option type.

```
1: IEnumerator<IOption> ReadInt() {
2:   Console.Write("Enter a number: ");
3:   var optNum = ParseInt(Console.ReadLine());
4:   var m = optNum.AsStep();
5:   yield return m;
6:   Console.WriteLine("Got a valid number!");
7:   var res = m.Value * 10;
8:   yield return OptionResult.Create(res);
9: }
```

The code reads a string from the console and calls the `ParseInt` method to get `optNum` value, which has a type `Option<int>` (line 3). Next, we need to perform non-standard value binding to access the value and to continue running the rest of the computation only when the value is present. Otherwise the method can return `None` as the overall result straight ahead.

To perform the value binding, we use the `AsStep` method that generates a helper object (line 4) and then return this object using yield return (line 5). This creates a "hole" in the code, because the rest of the code may or may not be executed, depending on whether the `MoveNext` method of the returned state machine is called again or not. When `optNum` contains a value, the rest of the code will be called and we can access the value using the `Value` property (line 7)[3].

Finally, the method calculates the result (line 7) and returns it. To return from a non-standard computation written using our encoding, we create another helper object, this time using `OptionResult.Create` method. These helper objects are processed when executing the method.

To summarize, there are two helper objects. Both of them implement the `IOption` interface, which means that they can both be generated using yield return. The methods that create these two objects have the following signatures:

```
AsStep : Option<'a> -> OptionStep<'a>
OptionResult.Create : 'a -> OptionResult<'a>
```

The first method creates an object that corresponds to the monadic bind operation. It takes the option value and composes it with the computation that follows the `yield return` call. The second method builds a helper that represents monadic unit operation. That means that the computation should end returning the specified value as the result.

3.2 Executing option calculation

When we write code in the style described in the previous section, methods like `ReadInt` only return a state machine that generates a sequence of helper objects representing bind and unit. This alone wouldn't be at all useful, because we want to execute the computation and get a value of the monadic type (in this case `Option<'a>`) as the result. How to do this in terms of standard monadic operations is described in section 5, but from the end user perspective, this simply means invoking the `Apply` method:

```
Option<int> v = ReadInt().Apply<int>();
Console.WriteLine(v);
```

This is a simple piece of standard C# code that runs the state machine returned by the `ReadInt` method. `Apply<'a>` is an extension method [4] defined for the `IEnumerator<IOption>` type. Its type signature is:

```
Apply : IEnumerable<IOption> -> Option<'a>
```

The type parameter (in the case above `int`) specifies what the expected return type of the computation is, because this unfortunately cannot be safely tracked in the type system. Running the code with different inputs gives the following console output:

```
Enter a number: 42      Enter a number: $%?!
Got a valid number!     None
Some(420)
```

Strictly speaking, `Apply` doesn't necessarily have to execute the code, because its behavior depends on the monadic type. The `Option<'a>` type represents a value, so the computation that produces it isn't delayed. On the other hand the `Async<'a>` type, which represents asynchronous computations is delayed meaning that the Apply method will only build a computation from the C# compiler generated state machine.

[2] We could as well use exceptions, but it is generally accepted that using exceptions for control flow is a wrong practice. In this case, the missing value is an expected option, so we're not handling exceptional condition.

[3] The F# code corresponding to these two lines is: `let! value = optNum`

[4] Extension methods are new feature in C# 3.0. They are standard static methods that can be accessed using dot-notation as if they were instance methods [1].

The encoding of non-standard computations wouldn't be practically useful if it didn't allow us to compose code from primitive functions and as we'll see in the next section, this is indeed possible.

3.3 Composing option computations

When encoding monadic operations, we're working with two different types. The methods we write using the iterator syntax return `IEnumerator<IOption>`, but the actual monadic type is `Option<'a>`. When writing code that is divided into multiple methods, we need to invoke method returning `IEnumerator<IOption>` from another method written using iterators. The following example uses the `ReadInt` method from the previous page to read two integer values (already multiplied by 10) and add them.

```
1: IEnumerator<IOption> TryCalculate() {
2:   var n = ReadInt().Apply<int>().AsStep();
3:   yield return n;
4:   var m = ReadInt().Apply<int>().AsStep();
5:   yield return m;
6:   var res = m.Value + n.Value;
7:   yield return OptionResult.Create(res);
8: }
```

When the method needs to read an integer, it calls the `ReadInt` to build a C# state machine. To make the result useable, it converts it into a value of type `Option<int>` (using the `Apply` method) and finally uses the `AsStep` method to get a helper object that can be used to bind the value using `yield return`.

We could of course provide a method composed from `Apply` and `AsStep` to make the syntax more convenient, but this paper is focused on explaining the principles, so we write this composition explicitly.

The previous example also nicely demonstrates the non-standard behavior of the computation. When it calls the `RadInt` method for the second time (line 4) it does that after using non-standard value binding (using `yield return` on line 3). This means that the user will be asked for the second number only if the first input was a valid number. Otherwise the result of the overall computation will immediately be `None`.

Calculating with options nicely demonstrates the principles of writing non- standard computations. We can use non-standard bindings to mark places where the code can abandon the rest of the code if it already knows the overall result. Even though this is already useful, we can make even stronger point to support the idea by looking at asynchronous computations.

4 Case study: asynchronous C#

As discussed in the introduction, writing non-blocking code in C# is painful even when we use latest C# features such as lambda expression. In fact, we haven't even implemented a simple loop, because that would make the code too lengthy. We've seen that monadic computations provide an excellent solution [5] and we've seen that these can be encoded in C# using iterators.

As next, we'll explore one larger example that follows the same encoding of monadic computations as the one in the previous section, but uses a different *monad* to write asynchronous code that doesn't block the tread when performing long-running I/O. The following method reads the entire content of a stream in a buffered way (using 1kb buffer) performing each read asynchronously.

```
1: IEnumerator<IAsync> ReadToEndAsync(Stream s)
   {
2:   var ms = new MemoryStream();
3:   byte[] bf = new byte[1024];
4:   int read = -1;
5:   while (read != 0) {
6:     var op = s.ReadAsync(bf, 0, 1024).
       AsStep();
8:     yield return op;
9:     ms.Write(bf, 0, op.Value);
10:    read = op.Value;
11:  }
12:  ms.Seek(0, SeekOrigin.Begin);
13:  string s = new StreamReader(ms).
     ReadToEnd();
14:  yield return AsyncResult.Create(s);
15: }
```

The code uses standard `while` loop which would be previous impossible. Inside the body of the loop, the method creates an asynchronous operation that reads 1kb of data from the stream into the specified buffer (line 6) and runs the operation by yielding it as a value from the iterator (line 7). The operation is then executed by the system and when it completes the iterator is resumed. It stores the bytes to a temporary storage and continues looping until the input stream is fully processed. Finally, the method reads the data using `StreamReader` to get a string value and returns this value using `AsyncResult.Create` method (line 14).

The encoding of asynchronous computations is essentially the same as the encoding of computations with option values. The only difference is that the method now generates a sequence of `IAsync` values. Also, the `AsStep` method now returns an object of type `AsyncStep<'a>` and similarly, the helper object used

[5] To justify this, we can say that asynchronous workflows are one of the important features that contributed to the recent success of the F# language.

for returning the result is now `AsyncResult<'a>`. Thanks to the systematic encoding described in section 5, it is very easy to use another non-standard computation once the user understands one example.

The method implemented in the previous listing is very useful and surprisingly, it isn't available in the standard .NET libraries. We'll use it to asynchronously download the entire web page. However, we first need to asynchronously get the HTTP response, so we'll write the code as another asynchronous method using iterator syntax. In section 3.3, we've seen how to compose *option computations* and since the principle is the same, it isn't surprising that composing *asynchonous computations* is also straightforward:

```
1: var stream = resp.Value.GetResponseStream();
2: var html = ReadToEndAsync(stream).
3:            Execute<string>().AsStep();
4: yield return html;
5: Console.WriteLine(html.Value);
```

The listing assumes that we already have a response object (`resp`). So far we haven't seen how to actually start the download, because *asynchronous computations* are delayed, meaning that we're just constructing a function that can be executed later. This makes it possible to compose large number of computations and spawn them in parallel. The runtime then uses only a few threads, which makes it very efficient and scalable. You can find full example that shows how to start the download in the online supplementary material for the article [12]

Implementing the functionality we presented in this section asynchronously using the usual style would be far more complicated. For example, to implement the `ReadToEndAsync` method we need two times more lines of very dense C# code. However, the code also becomes hard to read because it cannot use many high-level language features (e.g. `while` loop), so it would in addition also require a decent amount of comments[6].

5 Encoding arbitrary monads

As we've seen in the previous two sections, writing monadic computations in C# using iterators requires several helper objects and methods. In this section, we'll look how these helpers can be defined.

Unfortunately, C# doesn't support higher-kinded polymorphism, which is used when defining monads in Haskell and is available in some object-oriented language such as Scala [10]. This means that we can't write code that is generic over the monadic type (e.g. `Option<'a>` and `Async<'a>`). As a result, we need to define a new set of helper objects for each type of computation. To make this task easier, we provide two base classes that encapsulate functionality that can be reused. The code that we need to write is the same for every computation, so writing it is a straightforward task that could be even performed by a very simple code-generator tool.

In this section, we'll look at the code that needs to be written to support calculations working with option values that we were using in section 3. The code uses only two computation-specific operations. Indeed, these are the two operations bind and unit that are used to define monadic computations in functional programming ("O" is a shortcut for "Option"):

```
Bind : O<'a> -> ('a -> O<'b>) -> O<'b>
Return : 'a -> O<'a>
```

The bind operation uses the function provided as the second parameter to calculate the result when the first parameter contains a value. The unit operation wraps an ordinary value into an option value. The implementation of these operations is described elsewhere, so we won't discuss it in detail. You can for example refer to [11]. We'll just assume that we already have `OptionM` type with the two operations exposed as static methods.

5.1 Defining iterator helpers

As a first thing, we'll implement helper objects that are returned from the iterator. We've seen that we need two helper objects - one that corresponds to bind and one that corresponds to unit. These two objects share common interface (called `IOption` in case of option computations) so that we can generate a single sequence containing both of them. Let's start by looking at the interface type:

```
interface IOption {
  Option<R> BindStep<R>(Func<Option<R>> k);
}
```

The `BindStep` method is invoked by the extension that executes the non-standard computation (we'll discuss it later in section 5.2). The parameter `k` specifies a continuation, that is, a function that can be executed to run the rest of the iterator. The continuation doesn't take any parameters and returns an option value generated by the rest of the computation. The implementation of the helper objects that implement the interface looks like this:

[6] The source code implementing the same functionality in the usual style can be found in [12]

```
class OptionStep<T> : MonadStep<T>, IOption {
 internal Option<T> Input { get; set; }
 public Option<R> BindStep<R>(Func<Option<R>> k)
 {
  return OptionM.Bind(Input,
  MakeContinuation(k));
 }
}
class OptionResult<T> : MonadReturn<T>, IOption
{
 internal OptionResult(T value) : base(value)
 { }
 public Option<R> BindStep<R> (Func<Option<R>>
 k) {
  return OptionM.Return(GetResult<R>());
 }
}
```

The `OptionStep<'a>` type has a property named `Input` that's used to store the option value from which the step was constructed using the `AsStep` method. When the `BindStep` method is executed the object uses the monadic bind operation and gives it the input as the first argument. The second argument is more interesting. It should be a function that takes the actual value extracted from the input option value as an argument and returns a new option value. The extracted value can be used to calculate the result, but there is no way to pass a value as an argument back to the iterator in the middle of its evaluation, which is why the function given as the parameter to `BindStep` method doesn't take any parameters.

As we've seen in the examples, the `OptionStep<'a>` helper exposes this value as the `Value` property. This property is inherited from the `MonadStep<'a>` type. The `MakeContinuation` which we use to build a parameter for monadic bind operation is also inherited and it simply stores the input obtained from bind into `Value`, so that it can be used in the iterator and then runs the parameter-less continuation `k`.

The `OptionResult<'a>` type is a bit simpler. It has a constructor that creates the object with some value as the result. Inside the `BindStep` method, it uses the monadic unit operation and gives it that value as the parameter. This cannot be done in a statically type-checked way, so we use the inherited `GetResult` method that performs dynamic type conversion. Finally, the `OptionResult.Create` and `AsStep` methods are just simple wrappers that construct these two objects in the syntactically most pleasant way.

5.2 Implementing iterator evaluation

Once we have an iterator written using the helpers described in the previous section, we need some way for executing it using the non-standard behavior of the monadic computation. The purpose of the iterator isn't to create a sequence of values, so we need to execute it in some special way. The following example shows an extension method `Apply` that turns the iterator into a monadic value. In case of options the type of the value is `Option<'a>` but note that the code will be exactly the same for all computation types.

```
static class OptionExtensions {
  public static Option<R> Apply<R>
      (this IEnumerator<IOption> en) {
    if (!en.MoveNext())
      throw new InvalidOperationException
        ("Enumerator ended without a result!");
    return en.Current.BindStep<R>(() =>
      en.Apply<R>());
  }
}
```

The method starts by invoking the `MoveNext` method of the generated iterator to move the iterator to the next occurrence of the `yield return` statement. If the return value is `false`, the iterator ended without returning any result, which is invalid, so the method throws an exception.

If the iterator performs the step, we can access the next generated helper object using the `en.Current` property. The code simply invokes `BindStep` of the helper and gives it a function that recursively calls the `Apply` method on the same iterator as the argument. Note that when the helper is `OptionResult<'a>`, the continuation isn't used, so the recursion terminates.

It is worth noting that for monadic computations with zero operation we can also write a variant of the `Apply` method that doesn't require the iterator to complete by returning a result. In that case, we'd modify the method to return a value constructed by the monadic *zero* operation instead of throwing an exception in the case when the iterator ends.

Finally, there are also some problems with using possibly deeply nested recursive calls in a language that doesn't guarantee the use of tail-recursion. We can overcome this problem by using some technique for tail-call elimination. Schinz [15] gives a good overview in the context of the Scala language. Perhaps the easiest option to implement is to use a trampoline [17].

5.3 Translation semantics

To formalize the translation in a more detail, we've also developed a translation semantics for the library. We first define an abstract language extension for C# that adds monadic compuations as a language feature to C# in a way similar to F#. Then we show that any code written in the extended C# can be translated to the standard C# 2.0 by using the iterator encoding

presented in this article. The grammar of the language extension as well as the semantic rules are available in the online supplementary material [12].

6 Related work and conclusions

There is actually one more way for writing some monadic computations in C# using the recently added query syntax. The syntax is very limited, but may be suitable for some computations. We'll briefly review this option and then discuss other relevant related work and conclusions of this paper.

6.1 LINQ queries

As many people already noted [8], the LINQ query syntax available in C# 3.0 is also based on the idea of monad and can be used more broadly than just for encoding computations that work with lists. The following example shows how we could write the computation with option values using LINQ syntax:

```
1: Option<int> opt =
2:   from n in ReadInt()
3:   from m in ReadInt()
4:   let res = m + n
5:   select res;
```

The implementation of library that allows this kind of syntax is relatively easy and is described for example in [11]. This syntax is very restricted. In it allows non-standard value bindings corresponding to the bind operation using the from keyword (lines 2 and 3), standard value bindings using the let construct (line 4) and returning of the result using select keyword (line 5). However, there are no high-level imperative constructs such as loops which were essential for the asynchronous example in section 4. With some care, it is possible to define several mutually recursive queries, but that still makes it hard to write complex computations such as the one in section 4.

On the other hand, query syntax is suitable for some monadic computations where we're using only a limited language. Parser combinators as described for example in [6] can be defined using the query syntax [4]. In general, C# queries are a bit closer to writing monads using the Haskell's list comprehension notation, while using iterators as described in this article is closer to the Haskell's do-notation.

6.2 Related work

The principle of using a main-stream language for encoding constructs from the research world has been used with many interesting features including for example Joins [14]. FC++ [8] is a library that brings many functional features to C++, including monads, which means it should be possible to use it for re-implementing some examples from this paper.

There are also several libraries that use C# iterators for encoding asynchronous computations. CCR [7] is a more sophisticated library that combines join patterns with concurrent and asynchronous programming, which makes it more powerful than our encoding. On the other hand it is somewhat harder to use for simple scenarios such as those presented in this paper.

Richter's library [13] is also focused primarily on asynchronous execution. It uses yield return primitive slightly differently - to specify the number of operations that should be completed before continuing the execution of the iterator. The user can then pop the results from a stack.

7 Conclusions

In this paper, we have presented a way for encoding monadic computations in the C# language using iterators. We've demonstrated the encoding with two examples - computations that work with option values and computations that allow writing of non-blocking asynchronous code.

The asynchronous library we presented is useful in practice and would alone be an interesting result. However, we described a general mechanism that can be useful for other computations as well. We believe that using it to implement for example a prototype of software transactional memory support for C# can bring many other interesting results.

References

1. G.M. Bierman, E. Meijer, M. Torgersen: *Lost in translation: formalizing proposed extensions to C#.* In Proceedings of OOPSLA 2007.
2. ECMA International.: C# Language Specification.
3. T. Harris, S. Marlow, S. Peyton-Jones, M. Herlihy: *Composable memory transactions.* In Proceedings of PPoPP 2005.
4. L. Hoban: *Monadic parser combinators using C# 3.0.* Retrieved May 2009, from `http://blogs.msdn.com/lukeh/archive/2007/08/19/monadic-parser-combinators-using-c-3-0.aspx`
5. P. Hudak, P. Wadler, A. Brian, B.J. Fairbairn, J. Fasel, K. Hammond et al.: *Report on the programming language Haskell: A non-strict, purely functional language.* ACM SIGPLAN Notices.
6. G. Hutton, E. Meijer: *Monadic parser combinators.* Technical Report. Department of Computer Science, University of Nottingham.
7. G. Chrysanthakopoulos, S. Singh: *An asynchronous messaging library for C#.* In proceedings of SCOOL Workshop, OOPSLA, 2005.

8. B. McNamara, Y. Smaragdakis: *Syntax sugar for FC++: lambda, infix, monads, and more.* In Proceedings of DPCOOL 2003.
9. E. Meijer: *There is no impedance mismatch (Language integrated query in Visual Basic 9).* In Dynamic Languages Symposium, Companion to OOPSLA 2006.
10. A. Moors, F. Piessens, M. Odersky: *Generics of a higher kind.* In Proceedings of OOPSLA 2008.
11. T. Petricek, J. Skeet: *Functional Programming for the Real World.* Manning, 2009.
12. T. Petricek: *Encoding monadic computations using iterators in C# 2.0 (Supplementary material).* Available at `http://tomasp.net/academic/monads-iterators.aspx`
13. J. Richter: *Power threading library.* Retrieved May 2009, from `http://www.wintellect.com/PowerThreading.aspx`
14. C.V. Russo: *The Joins concurrency library.* In Proceedings of PADL 2007.
15. M. Schinz, M. Odersky: *Tail call elimination on the Java Virtual Machine.* In Proceedings of BABEL 2001 Workshop on Multi-Language Infrastructure and Interoperability.
16. D. Syme, A. Granicz, A. Cisternino: *Expert F#*, Apress, 2007.
17. D. Tarditi, A. Acharya, P. Lee: *No assembly required: Compiling standard ML to C.* School of Computer Science, Carnegie Mellon University.
18. P. Wadler: *Comprehending monads.* In Proceedings of ACM Symposium on Lisp and Functional Programming, 1990.

On existence of robust combiners for cryptographic hash functions*

Michal Rjaško

Department of Computer Science, Faculty of Mathematics, Physics and Informatics,
Comenius University, Bratislava
rjasko@dcs.fmph.uniba.sk

Abstract. *A (k, l)-robust combiner for collision resistant hash functions is a construction, which takes l hash functions and combines them so that if at least k of the components are collision resistant, then so is the resulting combination. A black-box (k, l)-robust combiner is robust combiner, which takes its components as black-boxes. A trivial black-box combiner is concatenation of any $(l-k+1)$ of the hash functions. Boneh and Boyen [1] followed by Pietrzak [3] proved, that for collision resistance we cannot do much better that concatenation, i.e. there does not exist black box (k, l)-robust combiner for collision resistance, whose output is significantly shorter that the output of the trivial combiner. In this paper we analyze whether robust combiners for other hash function properties (e.g. preimage resistance and second preimage resistance) exist.*

Key words: *Cryptographic hash function, robust combiner, preimage resistance, second preimage resistance*

1 Introduction

Cryptographic hash functions play important role in the current cryptography. In the last few years, many attacks on popular hash functions believed to be secure (e.g. SHA1, MD5) have been proposed. Within these attacks arises a question, whether we are able to construct a secure hash function. A hash function is a function $H : \{0,1\}^* \to \{0,1\}^v$, which maps messages (strings of 0 and 1) of arbitrary length to strings of fixed length – called images. Practically useful hash functions must guarantee several security properties: they should be collision resistant, what means that it is hard to find two different messages which map to the same image. Other important properties of hash functions are preimage resistance (for given image, it is hard to find its preimage, i.e. a message which maps to that image) and second-preimage resistance (for given message, it is hard to find another message, which maps to the same image). For formal definitions of the properties mentioned above or other notions of hash function security we refer to the works [4], [5].

Natural way how to construct secure hash function is to combine several known (regarded to be secure) hash functions in such a way, that if one of the combined functions appears to be insecure, the combination remains secure. For collision resistance we can achieve this by concatenation. Let $H_1, H_2 : \{0,1\}^* \to \{0,1\}^v$ be some hash functions. We can construct a hash function H, where

$$H(M) = H_1(M)||H_2(M).$$

Note, that if (M, M') is a pair of colliding messages for H, then this pair collides also for H_1 and H_2. Therefore if at least one of the H_1 and H_2 is collision resistant, then H is collision resistant too. However this approach has one important disadvantage – the output length of H is twice as large as the output of underlying hash functions H_1 and H_2, what can lead to problems with practical implementation, mainly on devices with small amount of memory as smart-cards.

Thus the question is, whether one can construct a secure combiner with output shorter than the concatenation. Boneh and Boyen in [1] proved the first negative result in this direction, in particular that there does not exist secure combiner for collision resistant hash functions with output shorter than concatenation, with an assumption, that the combiner queries each hash function exactly once. This result was generalized by Pietrzak [3], where the author proved that the secure combiners for collision resistance with significantly shorter output than concatenation do not exist. The later work consider a (k, l)-robust combiners for collision resistant, which are secure if at least k of the l components are secure.

In this paper we follow the work of Pietrzak [3] and prove the similar results for other important properties of hash functions – second preimage resistance and preimage resistance. We define a (k, l) combiner for preimage resistance and second preimage resistance and for these definitions we prove the impossibility results similar to one from [3].

Organization In the section 2 we start by some useful notation and continue with the formal definitions of (k, l) combiners for collision resistance, preimage resistance and second-preimage resistance. In the section 3 we prove the negative results, namely in the Theorem 1 we prove that secure combiner for preimage resistance with output significantly shorter than concatenation does not exists and in the Theorem 2 we prove the similar result for second-preimage resistance.

* This paper was supported by VEGA grant number 1/0266/09 and by Comenius University grant number UK/365/2009.

2 Preliminaries

In this section we formally define a combiner of l hash functions for three notions of hash function security – collision resistance, preimage resistance and second preimage resistance. The definition of the combiner for collision resistance is from [1], the other definitions (i.e. for preimage and second-preimage resistance) are slight modification of the former one.

We start with some basic notation. We write $M \xleftarrow{\$} \mathcal{S}$ for the experiment of choosing random element from the distribution $\mathcal{S}$. If $\mathcal{S}$ is a finite set, then M is chosen uniformly from $\mathcal{S}$. Concatenation of finite strings M_1 and M_2 we denote by $M_1||M_2$ or simply M_1M_2.

An oracle turing machine T with oracle access to turing machines $T_1, \ldots, T_l$ is a turing machine, which accepts inputs via input tape, performs some computation and replies via output tape. During the computation it can write on some additional "oracle" input tapes $t_1, \ldots, t_l$ and recieves responses via oracle output tapes $t'_1, \ldots, t'_l$ – connections to the turing machines $T_1, \ldots, T_l$. Whenever T writes some input on tape t_i, the turing machine T_i is run on that input and T recieves the output on tape t'_i. We call such a operation a query to oracle T_i. All queries are performed in unit time (i.e. computation of T_i is not counted into the running time of T). By $T^{T_1,\ldots,T_l}$ we denote that the oracle turing machine T has oracle access to the turing machines $T_1, \ldots, T_l$.

Let λ be some parameter. In this section, and later, by H_i we denote a function mapping from $\{0,1\}^*$ to $\{0,1\}^v$, where $i = 1 \ldots l$ and $v = p(\lambda)$ for some polynomial p. In general, a combiner of l hash functions H_i, $i = 1, \ldots, l$ for some notion of hash function security is a pair (C, P), where

- $C : \{0,1\}^m \to \{0,1\}^n$ does the "combination" ($m = p_m(\lambda)$ and $n = p_n(\lambda)$ for some polynomials p_m, p_n), i.e. it is an oracle turing machine with oracle access to $H_1, \ldots, H_l$. It behaves as a standard hash function (i.e. it takes some message on input and outputs a hash of the message – string of fixed length), but during the computation it can query any of its oracles.
- P provides a proof of the security for C. It is an algorithm, which transforms the "ability" of breaking the C (with respect to the particular security property) to the ability of breaking the candidates.

We note that both C and P should be efficient – they run in a time that is polynomial in the security parameter λ (and thus it is polynomial also in m, n or v). The oracle turing machine C is the same for combiners of all security notions. On the other hand, P needs to be modified when going from one security notion to another.

The security of combiner (with respect to some security notion) is determined by the number how many of the candidate hash functions need to be secure in order to guarantee that the resulting combiner is secure. By (k,l)-combiner (C,P) we denote the combiner (C,P), which is secure, if at least k of the l candidate hash functions are secure.

We expect from all candidate hash functions H_i to have the same output length v. We assume this just for simplicity, our results can be easily extended for variable output length of these candidate hash functions.

In this paper we deal only with black box combiners, what means that the combination algorithm (C) has only black box (i.e. oracle) access to the candidate hash functions. It does not know the structure of underlying primitives $H_1, \ldots, H_l$.

We start by formal definition of the combiner for collision resistance from [3].

2.1 Combiner for collision resistance

A collision resistant (k,l)-combiner for l hash functions $H_1, \ldots, H_l$ is a pair (C,P) of oracle turing machines C and P, where

- $C : \{0,1\}^m \to \{0,1\}^n$ has oracle access to hash functions $H_1, \ldots, H_i$ and performs the "combination" of hash functions. On input it takes a message of fixed length m and outputs an image of fixed length n. The output of C on an input M and with oracle access to $H_1, \ldots, H_l$ is denoted as $C^{H_1,\ldots,H_l}(M)$.
- P is an oracle machine, which provides a "proof" of security for C. It takes as input a pair of messages (M, M') and outputs two vectors

$$\mathbf{W} = (w_1, \ldots, w_l) \text{ and } \mathbf{W}' = (w'_1, \ldots, w'_l).$$

The role of the algorithm P is to transform the collision in C to collisions in at least $l - k + 1$ of the underlying hash functions $H_1, \ldots, H_l$. If such a P exists, which transforms collisions in C to collisions in H_i for *all* compatible $H_1, \ldots, H_l$, then we consider C as a collision resistant (k,l) combiner. Now, we formally define what was discussed above.

We say that P k succeeds on $H_1, \ldots, H_l$, M and M' if:

$$\exists J \subseteq \{1, \ldots, l\}, |J| \geq l - k + 1 :$$

$$(\forall j \in J) : \ (w_j, w'_j) \text{ is a collision for } H_j$$

$$(\text{i.e. } H_j(w_j) = H_j(w'_j))$$

Let

$$\mathbf{Adv}_P^{\text{Coll}[k]}[(H_1, \ldots, H_l), M, M']$$

denote the probability that P k-succeeds.

We say that (C, P) is ε-secure (k, l)-Coll-combiner, if for all $H_1, \ldots, H_l$ and all collisions (M, M') in C we have:

$$\mathbf{Adv}_P^{\mathrm{Coll}[k]}[(H_1, \ldots, H_l), M, M'] \geq 1 - \varepsilon$$

We consider (C, P) to be secure (k, l)-Coll-combiner, if ε is negligible in the security parameter λ.

For example, a secure $(1, 2)$-combiner for collision resistance can look like follows:

- $C^{H_1,H_2}(M) = H_1(M)||H_2(M)$
- $P^{H_1,H_2}(M, M') = (M, M), (M, M')$

The turing machine C just passes its input to the candidates and returns the concatenation of their output. Note that collision in C implies collision in both H_1 and H_2. In more detail, if (M, M') is a collision for such a C, then (M, M') is also a collision in both H_1 and H_2. Thus P has easy work – it copies its input to the output. It is easy to see, that for all H_1, H_2 and all collisions (M, M') in C is

$$\mathbf{Adv}_P^{\mathrm{Coll}[1]}[(H_1, H_2), M, M'] = 1.$$

Therefore (C, P) is 0-secure $(1, 2)$-Coll-combiner.

2.2 Combiner for preimage resistance

The combiner for preimage resistance is similar to one for collision resistance. Only difference is in the algorithm P, which provides "a proof of the security". Algorithm P is given on its input challenge images $y_1, \ldots, y_l$ of $H_1, \ldots, H_l$, for which it has to find preimages. It plays a game, in which it chooses an image of C for which it gets a preimage.

A preimage resistant combiner for l hash functions $H_1, \ldots, H_l$ is a pair (C, P) of oracle turing machines C and P, where

- $C : \{0,1\}^m \to \{0,1\}^n$ is the same as in the collision resistant combiner.
- P is an oracle turing machine, which provides a "proof" of security for C. It plays the following game. Let $f : \{0,1\}^n \to \{0,1\}^m \cup \{\perp\}$ be a function, such that for all $Y \in \{0,1\}^n$ is $f(Y) = M$, for which $C^{H_1,\ldots,H_l}(M) = Y$. If such a M does not exists, then $f(Y) = \perp$.

  Pre-Comb^f game:
  1. Messages $w_1, \ldots, w_l \in \{0,1\}^m$ are chosen at random, then images $y_1 = H_1(w_1), \ldots, y_l = H_l(w_l)$ are computed and given to P on its input.
  2. P with oracle access to $H_1, \ldots, H_l$ outputs some image $Y \in \{0,1\}^n$.
  3. A message $M = f(Y)$, is given to P (note that $C^{H_1,\ldots,H_l}(M) = Y$), if $f(Y) = \perp$, then the game ends and P fails.
  4. P continues (still can query $H_1, \ldots, H_l$) and outputs a vector $\mathbf{W} = (w'_1, \ldots, w'_l)$.

We note, that the function f, which parametrizes the Pre-Comb game can be understood as a deterministic "device", which finds preimages in C (it need not to be efficient). An algorithm P from a secure preimage resistant combiner should win the Pre-Comb game for all possible functions f (or at least for non-negligible part of such functions, however in this paper we consider combiners to be secure if they win for all possible fs).

We say that P k succeeds on $H_1, \ldots, H_l$, $y_1, \ldots, y_l$ and f if:

$$\exists J \subseteq \{1, \ldots, l\}, |J| \geq l - k + 1 :$$

$$(\forall j \in J) : \; w'_j \text{ is preimage of } y_j \text{ on } H_j$$

$$(\text{i.e. } H_j(w'_j) = y_j)$$

Let

$$\mathbf{Adv}_P^{\mathrm{Pre}[k]}[(H_1, \ldots, H_l), (y_1, \ldots, y_l), f]$$

denote the probability that P k-succeeds.

Finally, (C, P) is ε-secure (k, l)-Pre-combiner, if for all $H_1, \ldots, H_l$, all images $y_1, \ldots, y_l$ and all possible f we have:

$$\mathbf{Adv}_P^{\mathrm{Pre}[k]}[(H_1, \ldots, H_l), (y_1, \ldots, y_l), f] \geq 1 - \varepsilon$$

We say (C, P) is secure (k, l)-Coll-combiner, if ε is negligible in the security parameter λ.

For example consider the following $(1, 2)$-combiner for preimage resistance:

- $C^{H_1,H_2}(M_1||M_2) = H_1(M_1) \;||\; H_2(M_2)$ – C gets a message on its input, divides it into two parts of roughly equal length and passes these two parts to H_1 and H_2.
- P – given images y_1, y_2 on input, P computes the image $Y = y_1||y_2$ and returns it in the second step of the Pre-Comb game. In turn P receives a message M, where $C^{H_1,H_2}(M) = Y$. Finally P divides the message M into two parts w_1 and w_2 (exactly as C divides its input) and returns (w_1, w_2).

Since $C^{H_1,H_2}(w_1||w_2) = H_1(w_1)||H_2(w_2)$, we can see that $H_1(w_1) = y_1$ and $H_2(w_2) = y_2$. Thus (C, P) is 0-secure $(1, 2)$-Pre-combiner.

2.3 Combiner for second-preimage resistance

Here we define a combiner for second-preimage resistance. Again, only difference from combiners for preimage or collision resistance is in the algorithm P.

A second-preimage resistant combiner for l hash functions $H_1, \ldots, H_l$ is a pair (C, P) of oracle turing machines C and P, where

- $C : \{0,1\}^m \to \{0,1\}^n$ is the same as in the case of preimage resistant or collision resistant combiner.
- P is an oracle turing machine, which provides a "proof" of security for C. It plays the following game. Let $f : \{0,1\}^m \to \{0,1\}^m \cup \{\perp\}$ be a function, such that for all $M \in \{0,1\}^m$ is $f(M) = M'$, for $M' \neq M$ and $C^{H_1,\ldots,H_l}(M) = C^{H_1,\ldots,H_l}(M')$. If such a M' does not exist, then $f(M) = \perp$.

  Sec-Comb^f game:
  1. Message $w \in \{0,1\}^m$ is chosen at random and given to P on its input.
  2. P with oracle access to $H_1, \ldots, H_l$ outputs some message $M \in \{0,1\}^m$.
  3. P is given a message $M' = f(M)$ ($M' \neq M$ and $C^{H_1,\ldots,H_l}(M) = C^{H_1,\ldots,H_l}(M')$).
  4. P continues (still can query $H_1, \ldots, H_l$) and outputs a vector $(w'_1, \ldots, w'_l)$.

We say that P k succeeds on $H_1, \ldots, H_l$, w and f if:

$$\exists J \subseteq \{1, \ldots, l\}, |J| \geq l - k + 1 :$$

$$(\forall j \in J) : \; w'_j \text{ is second-preimage of } w \text{ on } H_j$$

$$\text{(i.e. } H_j(w'_j) = H_j(w) \text{ and } w'_j \neq w_j)$$

Now, let

$$\mathbf{Adv}_P^{\mathrm{Sec}[k]}[(H_1, \ldots, H_l), w, f]$$

denote the probability that P k-succeeds.

Finally, (C, P) is ε-secure (k, l)-Sec-combiner, if for all $H_1, \ldots, H_l$, all messages w and all possible f we have:

$$\mathbf{Adv}_P^{\mathrm{Sec}[k]}[(H_1, \ldots, H_l), w, f] \geq 1 - \varepsilon$$

And we say (C, P) is secure (k, l)-Coll-combiner, if ε is negligible in the security parameter λ.

Consider the following example of secure $(1, 2)$ combiner for second-preimage resistance:

- $C^{H_1,H_2}(M) = H_1(M)||H_2(M)$
- P – on input w in the second step of the Sec-Comb game P returns a message $M = w$. In turn P receives a message $M' \neq M$, where $C^{H_1,H_2}(M') = C^{H_1,H_2}(M)$. Finally P returns a vector (M', M').

It is easy to see,

$$\begin{aligned} C^{H_1,H_2}(M') &= H_1(M')||H_2(M') \\ &= H_1(M)||H_2(M) \\ &= C^{H_1,H_2}(M), \end{aligned}$$

thus $H_1(M') = H_1(M)$, $H_2(M') = H_2(M)$.

3 Impossibility proofs

Boneh and Boyen [1] followed by Pietrzak [3] showed, that there does not exist a collision resistant (k, l) combiner with short output. In this section we prove the similar results for preimage resistance and second-preimage resistance.

The impossibility result for preimage resistant combiners is given in the Theorem 1, and in the Theorem 2 is given the impossibility result for second-preimage resistance.

We start by notation used in the rest of this paper. Let (C, P) be some combiner and let:

- $\mathbf{W}_i(M)$ be the set of oracle queries to H_i made while evaluating $C^{H_1,\ldots,H_l}$ on the message M
- $\mathbf{V}_i(M) = \{H_i(M); M \in \mathbf{W}_i(M)\}$ be the set of corresponding answers.
- $w_{i,j}(M)$ be the j-th query to H_i made while evaluating $C^{H_1,\ldots,H_l}(M)$ and $v_{i,j}(M)$ be the corresponding answer.

To simplify the presentation we will assume that P can output only messages that it has queried during the game. Note that we can assume this without loss of generality.

Theorem 1. *Let (C, P) be a (k, l)-combiner for preimage resistance, where C can make at most q_C oracle queries. Suppose, that*

$$n < (v - \lg(q_C))(l - k + 1) - l$$

Then there exist $y_1, \ldots, y_l$, $H_1, \ldots, H_l$ and f, such that

$$\mathbf{Adv}_P^{Pre[k]}[(H_1, \ldots, H_l), (y_1, \ldots, y_l), f] \text{ is negl. in } \lambda$$

Proof. Let $H_1, \ldots, H_l : \{0,1\}^* \to \{0,1\}^v$ be all uniformly random hash functions. Let q_P be the total number of queries that P makes in the Pre-Comb game and let $y_1, \ldots, y_l$ be the challenge images that P gets in the first step of the game. From the fact, that P runs in a polynomial time we have that $q_P = p(\lambda)$ for some polynomial p. Therefore the probability that P queries any H_i with some message w, such that $H_i(w) = y_i$, is negligible in λ. To see this only a simple idea is needed. All of the H_is are random thus the probability that P gets output y_i for some $i = 1, \ldots, l$ in one

query is $l/2^v$. The probability that P queries y_i for some i in q_P queries is therefore

$$q_P \frac{l}{2^v} = p(\lambda)\frac{l}{2^{p'(\lambda)}},$$

what is negligible in λ.

Thus P's only chance to win is in the message M it gets as a preimage for the image Y chosen in the second step of the Pre-Comb game. We show, that if $n < (v - \lg(q_C))(l-k+1) - l$, then there exist images $y_1, \ldots, y_l$, and a function f such that for all images Y which P can output in the second step, evaluating of $C(f(Y))$ does not present a preimage for at least k of the $y_1, \ldots, y_l$. This means, that for such $H_1, \ldots, H_l$, $y_1, \ldots, y_l$ and f is

$$\mathbf{Adv}_P^{\mathrm{Pre}[k]}[(H_1, \ldots, H_l), (y_1, \ldots, y_l), f]$$

negligible in λ, what we want to prove.

Let $H_1, \ldots, H_l$ be as defined above (independent random hash functions) and let $Y \in \{0,1\}^n$ be some image of $C^{H_1,\ldots,H_l}$. Consider the following random experiment. First, images $y_1, \ldots, y_l \in \{0,1\}^v$ are chosen at random and then a message $M \in \{0,1\}^m$ is randomly chosen. Now consider the following events:

$$\begin{aligned}
\mathcal{E}_1 &\iff C^{H_1,\ldots,H_l}(M) = Y \\
\mathcal{E}_2 &\iff \exists J \subseteq \{1,\ldots,l\}, |J| > l-k : \\
&\qquad \forall j \in J : y_j \in \mathbf{V}_j(M)
\end{aligned}$$

Note that if $\Pr[\mathcal{E}_1] > \Pr[\mathcal{E}_2]$. then $\Pr[\mathcal{E}_1 \wedge \neg\mathcal{E}_2] > 0$, what means, that there exist images $y_1, \ldots, y_l \in \{0,1\}^v$ such that for each $Y \in \{0,1\}^n$ there exists a message $M \in \{0,1\}^m$ for which $C^{H_1,\ldots,H_l}(M) = Y$ and the evaluation of $C^{H_1,\ldots,H_l}(M)$ does not present preimages for $y_1, \ldots, y_l$. In other words, it means that there exist images $y_1, \ldots, y_l$ and a function f[1] for which the theorem holds.

We know that for particular Y and randomly chosen M is

$$\Pr[C^{H_1,\ldots,H_l}(M) = Y] \geq 2^{-n}.$$

Thus

$$\Pr[\mathcal{E}_1] \geq 2^{-n}.$$

To find $\Pr[\mathcal{E}_2]$, let q_i be the number of queries to H_i made by C (note that $\sum_{i=1}^{l} q_i = q_C$). The $\Pr[\mathcal{E}_2]$ can be upper bounded by the probability that the best oracle algorithm $A^{H_1,\ldots,H_l}$, which is allowed to query H_i at most q_i times finds a preimage for at least $l-k+1$ of y_i (A can evaluate $C^{H_1,\ldots,H_l}$ and then A's success probability is equal to $\Pr[\mathcal{E}_2]$). Since H_i are all independent random functions, the best A can do is to query each H_i with q_i distinct inputs. Now we follow the same steps as Pietrzak [3] did in the proof of the similar theorem for collision resistant combiner.

$$\begin{aligned}
\Pr[\mathcal{E}_2] &\leq \Pr[A^{H_1,\ldots,H_l} \text{ finds } l-k+1 \text{ preimages}] \\
&\leq \sum_{\substack{J\subseteq\{1,\ldots,l\} \\ |J|=l-k+1}} \Pr[\forall i \in J : A \text{ finds preimage for } y_i] \\
&\leq \sum_{\substack{J\subseteq\{1,\ldots,l\} \\ |J|=l-k+1}} \prod_{i\in J} \frac{q_i}{2^v} \\
&< \sum_{\substack{J\subseteq\{1,\ldots,l\} \\ |J|=l-k+1}} \frac{q_C^{l-k+1}}{2^{v(l-k+1)}} \\
&\leq \binom{l-k+1}{l} \frac{q_C^{l-k+1}}{2^{v(l-k+1)}} < \frac{2^l q_C^{l-k+1}}{2^{v(l-k+1)}}
\end{aligned}$$

When we put everything together:

$$\lg(\Pr[\mathcal{E}_1]) \geq \lg(2^{-n}) = -n$$

and

$$\begin{aligned}
\lg(\Pr[\mathcal{E}_2]) &< \lg\left(\frac{2^l q_C^{l-k+1}}{2^{v(l-k+1)}}\right) \\
&= (-(v - \lg(q_C))(l-k+1) + l).
\end{aligned}$$

Thus if $n < (v-\lg(q_C))(l-k+1) - l$ we have $\Pr[\mathcal{E}_1] > \Pr[\mathcal{E}_2]$, what we wanted to prove.

□

The condition $n < (v-\lg(q_C))(l-k+1)-l$ gives the lower bound how short can be the output of a secure (k,l)-Pre-combiner. It looks rather unnatural, however if C is allowed to query each H_i exactly once and we consider only $(1,l)$-combiners (what means, that a combiner is secure if at least one of the candidates is secure) then the condition can be rewritten to $n < (v-1)l$.

We note, that this lower bound need not to be optimal – maybe there exists a higher lower bound. We leave such an analysis for future work. Similar analysis for collision resistant combiners can be found in the work [2].

Theorem 2. *Let (C,P) be a (k,l)-combiner for 2nd-preimage resistance, where C makes at most q_C oracle queries. Suppose, that*

$$n < (v - \lg(q_C))(l-k+1) - l + 1$$

Then there exist w, $H_1, \ldots, H_l$ and f, for which

$$\mathbf{Adv}_P^{Sec[k]}[(H_1, \ldots, H_l), w, f] \text{ is negligible in } \lambda$$

[1] for each Y we set $f(Y)$ to the corresponding message M

Proof. The proof is very similar to one in the Theorem 1, therefore we provide just a sketch of the proof. Let $H_1, \ldots, H_l$ be all independent random hash functions. We claim, that the probability where P queries a second-preimage of w for at least one of the H_i is negligible. This is due the fact, that P runs in a polynomial time and therefore it can make at most polynomial number of queries, what is not enough for winning against random functions (for more formal discussion see the similar part in the proof of the Theorem 1).

Therefore we only need to prove that if $n < (v - \lg(q_C))(l-k+1) - l + 1$, then there exist a message w and a function f, where for all messages M that P can output in the second step of the Sec-Comb game evaluation of $C(f(M))$ does not present a second preimage of w for at least k of the $H_1, \ldots, H_l$.

Thus let $H_1, \ldots, H_l$ be as defined above and let $M \in \{0,1\}^m$ be some message. Consider the random experiment, where $w \in \{0,1\}^m$ and $M' \in \{0,1\}^m$ are chosen uniformly at random. We define the following events $\mathcal{E}_1$ and $\mathcal{E}_2$:

$$\begin{aligned}\mathcal{E}_1 &\iff M' \neq M \wedge C^{H_1,\ldots,H_l}(M) = C^{H_1,\ldots,H_l}(M') \\ \mathcal{E}_2 &\iff \exists J \subseteq \{1,\ldots,l\}, |J| > l-k : (\forall j \in J)(\exists i) : \\ &\qquad v_{j,i}(M') = H_j(w) \wedge w_{j,i}(M') \neq w\end{aligned}$$

In other words, $\mathcal{E}_1$ is the event when M' is the second-preimage of M in the sense of $C^{H_1,\ldots,H_l}$ and $\mathcal{E}_2$ means that the evaluation of $C^{H_1,\ldots,H_l}(M')$ does presents at least $l-k+1$ second-preimages for w. Again, if we prove that $\Pr[\mathcal{E}_1] > \Pr[\mathcal{E}_2]$, then $\Pr[\mathcal{E}_1 \wedge \neg\mathcal{E}_2] > 0$, what means that for each M the above $w_1, \ldots, w_l$ and M' exist and therefore there exists a function f (we set $f(M) := M'$) for which the theorem holds (together with $w_1, \ldots, w_l$).

Now, since $m > n$

$$\Pr[\mathcal{E}_1] = 2^{-n} - 2^{-m} \geq 2^{-n-1}$$

Now, let q_i be the maximum number of queries to H_i made by C. We can upper bound $\Pr[\mathcal{E}_2]$ by the probability that the best oracle algorithm A which can query each H_i at most q_i times finds second-preimage of w for at least $l-k+1$ of the H_is. However $H_1, \ldots, H_l$ are all independent random functions, thus the best A can do is to query each H_i on q_i distinct inputs different from w. If we follow the steps as in the corresponding part of the proof of the Theorem 1 we get

$$\Pr[\mathcal{E}_2] < \frac{2^l q_C^{l-k+1}}{2^{v(l-k+1)}}$$

When we put everything together:

$$\lg(\Pr[\mathcal{E}_1]) \geq \lg(2^{-n-1}) = -n - 1$$

and

$$\begin{aligned}\lg(\Pr[\mathcal{E}_2]) &< \lg\left(\frac{2^l q_C^{l-k+1}}{2^{v(l-k+1)}}\right) \\ &= -(v - \lg(q_C))(l-k+1) + l.\end{aligned}$$

Thus if $n < (v - \lg(q_C))(l-k+1) - l + 1$ we have $\Pr[\mathcal{E}_1] > \Pr[\mathcal{E}_2]$, what we wanted to prove.

□

4 Conclusion

We proved that combiners for preimage resistance and second-preimage resistance with output (significantly) shorter than concatenation do not exist. Our results are similar to ones for collision resistance from [1] and [3]. In particular, we showed that one can not create a secure (k,l)-combiner (for some mentioned security notion) of l *arbitrary* hash functions, with output shorter than concatenation of $l-k+1$ candidates.

These results are, however, too strict in a point, that we want from a combiner to work for *all* l-tuples of (compatible) hash functions and that the algorithm P providing the proof of combiner's security must succeed with non-negligible probability on all possible inputs. Such a condition is not very practically relevant – one can think of creating a combiner, which combines only hash functions from some subset of all hash functions (e.g. set of functions computable in polynomial time). Another way how to weaken the restrictions on combiners is to allow the algorithm P to fail on negligible part of inputs. We leave the analysis of such "weaker" combiners for a future work.

References

1. D. Boneh and X. Boyen: *On the impossibility of efficiently combining collision resistant hash functions.* LNCS, 4117, 2006.
2. R. Canetti, R. Rivest, M. Sudan, L. Trevisan, S. Vadhan and H. Wee: *Amplifying collision resistance: a complexity-theoretic treatment.* LNCS, 4622, 2007.
3. K. Pietrzak: *Non-trivial black-box combiners for collision-resistant hash-functions don't exist.* LNCS, 4515, 2007.
4. P. Rogaway and T. Shrimpton: *Cryptographic hash-function basics: definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance.* In Fast Software Encryption, LNCS, Springer, 3017, 2004, 371–388.
5. M. Rjaško: *Properties of cryptographic hash functions.* Mikulášska Kryptobesídka, 2008.

Localization with a low-cost robot*

Stanislav Slušný, Roman Neruda, and Petra Vidnerová

Institute of Computer Science, Academy of Sciences, Czech Republic
slusny@cs.cas.cz

Abstract. *The robot localization problem is a fundamental and well studied problem in robotics research. Algorithms used to estimate pose on the map are usually based on Kalman or particle filters. These algorithms are able to cope with errors, that arise due to inaccuracy of robot sensors and effectors. The performance of the localization algorithm depends heavily on their quality.*
This work shows performance of localization algorithm based on particle filter with small miniature low-cost E-puck robot. Information from VGA camera and eight infrared sensors are used to correct estimation of the robot's pose.

1 Introduction

The robot localization problem is a fundamental and well studied problem in robotics research. Several algorithms are used to estimate pose on the known map and cope with errors, that arise due to inaccuracy of robot sensors and effectors. Their performance depends heavily on quality of robot's equipment: the more precise (and usually more expensive) sensors, the better results of localization procedure.

This work deals with localization algorithm based on particle filter with small miniature low-cost E-puck robot. Information from cheap VGA camera and eight infrared sensors are used to correct estimation of the robot's pose. To achieve better results, several landmarks are put into the environment. We assume, that robot knows the map of the environment in advance (distribution of obstacles and walls in the environment and position of the landmarks). We do not consider the more difficult simultaneous localization and mapping (SLAM) problem in this work (the case, when robot does not know it's own position in advance and does not have the map of the environment available).

E-puck is a widely used robot for scientific and educational purposes - it is open-source and low-cost. Despite its cheapness and limited sensor system, localization can be successfully implemented, as will be shown in this article. We are not aware of any published results of localization algorithms with E-puck robot. The survey of localization methods can be found in [1].

* This work was supported by GA ČR grant 201/08/1744, Institutional Research Plan AV0Z10300504 and by the Ministry of Education of Czech Republic under the project Center of Applied Cybernetics No. 1M684077004 (1M0567).

Fig. 1. Miniature e-puck robot has eight infrared sensors and two motors.

2 Introducing E-puck robot

E-puck ([2, 3], Figure 1) is a mobile robot with a diameter of 70 mm and a weight of 50 g. The robot is supported by two lateral wheels that can rotate in both directions and two rigid pivots in the front and in the back. The sensory system employs eight "active infrared light" sensors distributed around the body, six on one side and two on other side. In "passive mode", they measure the amount of infrared light in the environment, which is roughly proportional to the amount of visible light. In "active mode" these sensors emit a ray of infrared light and measure the amount of reflected light. The closer they are to a surface (the e-puck sensors can detect a white paper at a maximum distance of approximately 8 cm), the higher is the amount of infrared light measured. Unfortunately, because of their imprecision and characteristics (see Figure 2), they can be used as bumpers only. As can be seen, they provide high resolution only within few millimeters. They are very sensitive to the obstacle surface, as well. Besides infrared sensors, robot is equipped with low-cost VGA camera. The camera and image processing will be described in the following section.

Two stepper motors support the movement of the robot. A stepper motor is an electromechanical device which converts electrical pulses into discrete mechanical movements. It can divide a full rotation into a 1000 steps, the maximum speed corresponds to about a rotation every second.

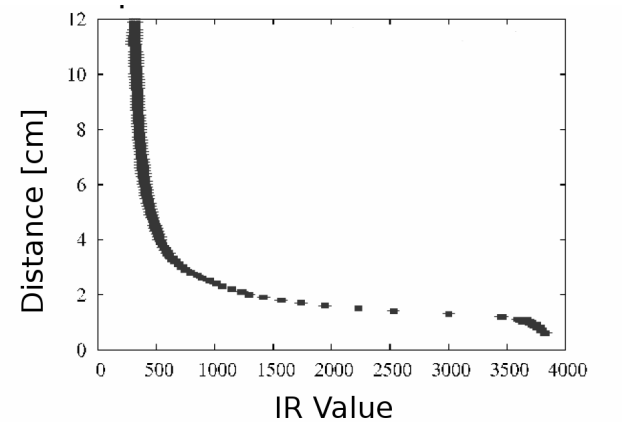


Fig. 2. Multiple measurements of front sensor. E-puck was placed in front of the wall at a given distance and average IR sensor value from 10 measurements was drown into the graph.

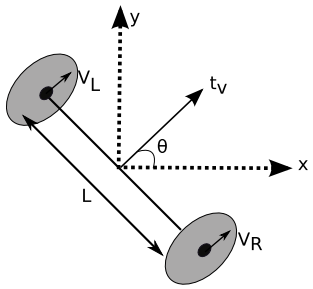


Fig. 3. Differential drive robot schema.

3 Dead reckoning

Dead reckoning ([4], derived originally from deduced reckoning) is the process of estimating robot's current position based upon a previously determined position. For shorter trajectories, position can be estimated using shaft encoders and precise stepper motors.

E-puck is equipped with a differential drive (Figure 3) - a simplest method to control robot. For a differential drive robot the position of the robot can be estimated by looking at the difference in the encoder values Δs_R and Δs_L. By estimating the position of the robot, we mean the computation of tuple x, y, Θ as a function of previous position $(x_{OLD}, y_{OLD}, \Theta_{OLD})$ and encoder values (Δs_R and Δs_L).

$$\begin{pmatrix} x \\ y \\ \theta \end{pmatrix} = \begin{pmatrix} x_{OLD} \\ y_{OLD} \\ \theta_{OLD} \end{pmatrix} + \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{pmatrix} \quad (1)$$

$$\Delta\theta = \frac{\Delta s_R - \Delta s_L}{L} \quad (2)$$

$$\Delta s = \frac{\Delta s_R + \Delta s_L}{2} \quad (3)$$

Parameters	Value
Maximum translational velocity	12.8 cm / sec
Maximum rotational velocity	4.86 rad / sec
Stepper motor maximum speed	+- 1000 steps / sec
Distance between tires	5.3 cm

Table 1. Velocity parameters of E-puck mobile robot.

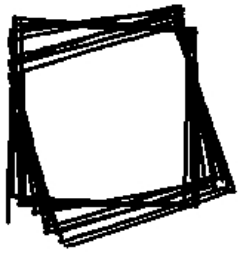


Fig. 4. Illustration of error accumulation. Robot was ordered to make 10 squares of size 30 cm. Odometry errors are caused mostly by rotation movement.

$$\Delta x = \Delta s.cos(\theta + \frac{\Delta\theta}{2}) \quad (4)$$

$$\Delta y = \Delta s.sin(\theta + \frac{\Delta\theta}{2}) \quad (5)$$

The major drawback of this procedure is error accumulation. At each step (each time you take an encoder measurement), the position update will involve some error. This error accumulates over time and therefore renders accurate tracking over large distances impossible (see Figure 4). Tiny differences in wheel diameter will result in important errors after a few meters, if they are not properly taken into account.

4 Image processing

The robot has a low-cost VGA camera with resolution of 480x640 pixels. Unfortunately, the Bluetooth connection supports only a transmission of 2028 colored pixel. For this reason a resolution of 52x39 pixels maximizes the Bluetooth connection and keeps a 4:3 ratio. This is the resolution we have used in our experiments (see Figure 4). Another drawback of the camera is that it is very sensitive to the light conditions.

Despite these limitations, camera can be used to detect objects or landmarks. However, the information about distance to the landmark extracted from the

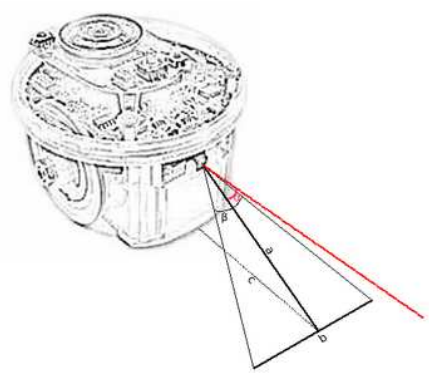


Fig. 5. The physical parameters of the real camera (picture taken from [2]). Camera settings used in experiments corresponds to parameters $a = 6$ cm, $b = 4.5$ cm, $c = 5.5$ cm, $\alpha = 0.47$ rad, $\beta = 0.7$ rad.

camera is not reliable (due to the noise), and we do not use it in following section.

Landmarks are objects of rectangular shape of size 5x5 cm and three different colors - red, green and blue. We implemented image processing subsystem, that detects relative position of the landmark from the robot. Following steps are included:

- Gaussian filter is used to reduce camera noise ([5])
- Color segmentation into the red, blue and green color. ([6])
- Blob detection is used to detect position and size of the objects on the image. ([7])
- Object detection is used to remove objects from image, that have non-rectangular shape.

Output from the image processing is the relative position and color of the detected landmarks (for example - I see red landmark by angle 15 degrees).

5 Particle filter localization

As shown previously (Figure 4), pose estimation based on dead reckoning is possible for short distances only. For longer trajectories, more clever methods are needed. These methods are based either on Kalman filter [8] (or some of its variants) or particle filter (PF) [9].

The PF possesses three basic steps - state prediction, observation integration and resampling. It works with quantity $p(x_t)$ - the probability, that robots is located at the position x_t in time t. In the case of PF, the probability distribution is represented by the set of particles. Such a representation is approximate, but can represent much broader space of distributions that, for example, Gaussians, as it is nonparametric.

Each particle $x_t^{[m]}$ is a hypothesis, where the robot can be at time t. We have used M particles in our

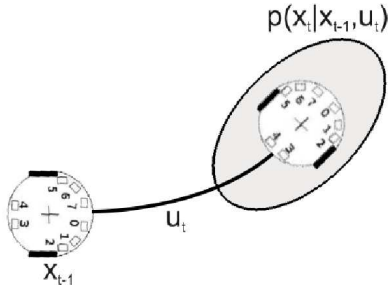


Fig. 6. First step in PF algorithm - to each position hypothesis x_{t-1} is applied odometry model based on movement u_{t-1} and new hypothesis x_t is sampled from distribution $p(x_t|x_{t-1}, u_{t-1})$.

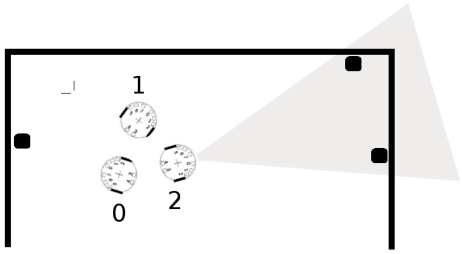


Fig. 7. Second step in PF algorithm - each particle is assigned a importance factor, corresponding to the probability of observation z_t. If image processing detects two landmarks on the actual camera image, particles 0 and 1 will be assigned small weight.

experiment. The input of the algorithm is the set of particles X_t, most recent control command u_t and the most recent sensor measurements z_t.

1. State prediction based on odometry.
   The first step is the computation of temporary particle set X from X_t. It is created by applying odometry model $p(x_t|u_t, x_{t-1})$ to each particle $x_t^{[m]}$ from X_t.
2. Correction step - Observation integration
   The next step is the computation of *importance factor* $w_t^{[m]}$. It is the probability of the measurement z_t under particle $x_t^{[m]}$, given by $w_t^{[m]} = p(z_t|x_t^{[m]})$.
   Two types of measurements were considered:
   - Measurement coming from distance sensors
     Distance sensor (one averaged value for front, left, right and back direction) were used as bumpers only. In case of any contradiction between real state and hypothesis, importance factor was decreased correspondingly.
   - Measurement obtained from image processing

Fig. 8. Placement of red, green and blue landmarks in rectangular arena of size 1x0.75m.

Output from image processing was compared with expected position of the landmarks. In case of any contradiction (colors and relative angle of landmarks were checked), importance factor was decreased. The bigger mismatch, the smaller importance factor was assigned to the hypothesis.

3. Re-sampling
   The last step incorporates so-called *importance sampling*. The algorithm draws with replacement M particles from temporary set X and creates new particle set X_{t+1}. The probability of drawing each particles is given by its importance weight. This principle is called *survival of the fittest* in AI ([10]).

6 Experiments

Experiments were carried out in an arena of size 1x0.75 meters. Three landmarks (red, blue and green, one of each color) were placed into the arena, as shown on the figure 8. Robot was controlled by commands sent from computer, values from sensors were sent back to computer by using Bluetooth. Execution of each command took 64 milliseconds.

The experiment started by putting robot into the arena and randomly distributing 2000 particles. After several steps, the PF algorithm relocated the particles into real location of the robot. The robot was able to localize itself. The convergence of the algorithm depends on the fact, if robot is moving near the wall or in the middle of the arena. The impact of infrared sensors was obvious. Algorithms were verified in the simulator([11]) and in reality, as well. The video demonstration can be found at ([12]).

The localization algorithm was able to cope with even bigger areas, up to the size of three meters. However, we had to add more landmarks to simplify the localization process. Localization algorithm showed satisfiable performance, relocating hypothesis near real robot pose.

7 Conclusions

Localization and pose estimation is an opening gate towards more sophisticated robotics experiments. As we have shown, the localization process can be carried out even with low-cost robot. Experiments were executed both in simulation and real environment.

A lot of work remains to be done. The experiments in this work considered static environment only. Addition of another robot will make the problem much more difficult.

As we have mentioned already, there are certain areas in the environment, where convergence of the localization algorithm is very fast - in corners or near walls. *Sensor fusion* is the process of combining sensory data from disparate sources such that the resulting information is in some sense better than would be possible when these sources were used individually. We are dealing with sensor fusion of infrared sensors and input from camera.

As a future work, we would like to implement path planning, that takes into account performance of the localization algorithm. Suggested path (generated by path planning algorithm) should be safe (the chance to get lost should be small) and short. Multi-criterial path planning will be based on dynamic programming ([13]). The idea is to learn areas with high loss probability from experience.

References

1. S. Thrun, W. Burgard, and D. Fox: *Probabilistic Robotics.* Cambridge, MA: MIT Press, 2005.
2. `http://en.wikibooks.org/wiki/Cyberbotics_Robot_Curriculum/`
3. E-puck, online documentation. `http://www.e-puck.org`
4. R.C. Arking: *Behavior-Based Robotics.* The MIT Press, 1998.
5. L.G. Shapiro, G.C. Stockman: *Computer Vision.* Prentence Hall, 150, 2001, p. 137.
6. J. Bruce, T. Balch and M. Veloso: *Fast and Inexpensive Color Image Segmentation for Interactive Robots.* In Proceedings of IROS-2000, 2000, 2061–2066.
7. `http://www.v3ga.net/processing/BlobDetection/index-page-home.html`
8. R.E. Kalman: *A new approach to linear filtering and prediction problems.* Trans. ASME, Journal of Basic Engineering 82, 35–45.
9. R.Y. Rubinstein: *Simulation and the Monte Carlo Method..* John Wiley and Sons, Inc.
10. K. Kanazawa, D. Koller, and S.J. Russel: *Stochastic simulation algorithms for dynamic probabilistic networks.* In Proceedings of the 11th Annual Conference on Uncertainty in AI, Montreal, Canada.
11. Webots simulator. `http://www.cyberbotics.com`.
12. Video demonstration. `http://www.cs.cas.cz/slusny`.
13. R.S. Sutto, and A. Barto: *Reinforcement Learning: An Introduction.* The MIT Press, 1998.