Department of Computer Science
Faculty of Science
Pavol Jozef Šafárik University

Peter Vojtáš (Ed.)

ITAT 2006
Information Technologies - Applications and Theory

Workshop on Theory and Practice
of Information Technologies, Proceedings
Slovakia, September 2006

Department of Computer Science
Faculty of Science
Pavol Jozef Šafárik University

Peter Vojtáš (Ed.)

ITAT 2006
Information Technologies - Applications and Theory

Workshop on Theory and Practice
of Information Technologies, Proceedings
Slovakia, September 2006

Preface

The 6th workshop **ITAT'06 – Information Technology – Applications and Theory** (`http://ics.upjs.sk/itat`) was held in Chata Kosodrevina, Bystrá dolina (`http://www.nizketatry.sk/chaty/chkosodrevina/chkosodrevina.html/`) located 1510 meter above the sea level in Lower Tatra, Slovakia, in end of September 2006.

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

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

Thematically workshop ranges from foundations of informatics, security, through data and semantic web to software engineering. In this year the emphasis is on semantics of data, information and knowledge.
Papers are divided into following classes:

- original scientific papers;
- student papers
- tutorial and
- work in progress.

All papers were refereed by at least two independent referees. There were 39 abstracts submitted.

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

Affiliated to this event a workshop "NAZOU - Tools for knowledge acquisition and organization from heterogeneous information sources" with separate proceedings was organized.

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

Peter Vojtáš

Program Committee

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

Organization

ITAT 2006 Information Technologies – Applications and Theory was organized by
University of P.J. Šafárik, Košice, SK
Institute of Computer Science, AS CR, Prague, CZ
Faculty of Mathematics and Physics, Charles University, Prague, CZ
Faculty of Informatics and Information Technology of the Slovak Technical University, Bratislava, SK
Slovak Society for Artificial Intelligence, SK

Table of Contents

ITAT'06

Information Technology – Applications and Theory

TUTORIAL

Inverse problems in learning from data

Věra Kůrková

Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague,
vera@cs.cas.cz, http://www.cs.cas.cz/∼vera

Abstrakt *Theory of inverse problems has been developed to solve various tasks in applied science such as acoustics, geophysics and computerized tomography. It is shown that this theory can be also applied to learning from data. Minimization of expected and empirical error functionals modeling learning from examples can be formulated as inverse problems. This reformulation allows us to characterize optimal solutions of learning tasks and to model generalization in terms of stability imposed upon solutions by regularization.*

1 Inverse problems

Tasks of finding *unknown causes* (e.g., shapes of functions, forces or distributions) from *known consequences* (measured data) have been studied in applied science, such as acoustics, geophysics and computerized tomography (see, e.g., [16]), under the name *inverse problems*. To solve such a problem, one needs to know how unknown causes determine known consequences, which can often be described in terms of an *operator*. In problems originating from physics, dependence of consequences on causes is usually described by integral operators (such as those defining Radon or Laplace transforms [4], [11]).

For a *linear operator* $A : \mathcal{X} \to \mathcal{Y}$ between two Hilbert spaces $(\mathcal{X}, \|.\|_{\mathcal{X}})$, $(\mathcal{Y}, \|.\|_{\mathcal{Y}})$ (in finite-dimensional case, a matrix A) an *inverse problem* (see, e.g., [4]) determined by A is to find for $g \in \mathcal{Y}$ (called *data*) some $f \in \mathcal{X}$ (called *solution*) such that

$$A(f) = g \tag{1}$$

In 1902, Hadamard [15] introduced a concept of a *well-posed* problem in solving differential equations. Formally, well-posed inverse problems were defined by Courant [7] in 1962 as problems, where for all data there *exists a solution* which is *unique* and depends on the data *continuously*. So for a well-posed inverse problem, there exists a unique inverse operator $A^{-1} : \mathcal{Y} \to \mathcal{X}$. When A is continuous, then by the Banach open map theorem [12, p.141] A^{-1} is continuous, too, and so the operator A is a homeomorphism of $\mathcal{X}$ onto $\mathcal{Y}$. When for some data, either there is no solution or there are multiple solutions or solutions do not depend on data continuously, the problem is called *ill-posed*.

When A is continuous, then by the Banach open map theorem [12, p.141] A^{-1} is continuous, too. Continuous dependence of solutions on data cannot prevent small variations in data to have large effects on forms of solutions. *Stability* of solutions can be measured by the *condition number* of the operator A defined as

$$cond(A) = \|A\| \, \|A^{-1}\|.$$

When this number is large, solutions can be too sensitive to data errors. In such cases, the inverse problems are called *ill-conditioned*. Note that the concept of ill-conditioning is rather vague. More information about stability can be obtained from an analysis of behavior of the singular values of the operator A (see, e.g., [16]). Often, inverse problems are ill-posed or ill-conditioned.

2 Pseudosolutions and regularization

For finite-dimensional inverse problems, in 1920 Moore proposed a method of *generalized inversion* based on a search for *pseudosolutions*, also called *least-square solutions*, for data, for which no solutions exist. His idea, published as an abstract [22], has not received too much attention untill it was rediscovered by Penrose [24] in 1955. So it is called *Moore-Penrose pseudoinversion*. In the 1970s, it has been extended to the infinite-dimensional case, where similar properties as the ones of Moore-Penrose pseudoinverses of matrices hold for pseudoinverses of *continuous linear operators* between Hilbert spaces [14].

When for some $g \in \mathcal{Y}$ no solution exists, at least one can search for a *pseudosolution* f^o, for which $A(f^o)$ is a best approximation to g among elements of the range of A, i.e.,

$$\|A(f^o) - g\|_{\mathcal{Y}} = \min_{f \in \mathcal{X}} \|A(f) - g\|_{\mathcal{Y}}.$$

The set $S(g) = \mathrm{argmin}(\mathcal{X}, \|A(.) - g\|_{\mathcal{Y}})$ is convex and so if it is nonempty, there exists a unique pseudosolution of the minimal norm f^+, called the *normal pseudosolution* [14]. So

$$\|f^+\|_{\mathcal{X}} = \min\{\|f^o\|_{\mathcal{X}} \mid f^o \in \mathrm{argmin}(\mathcal{X}, \|A(.) - g\|_{\mathcal{Y}})\}.$$

For an operator $A : \mathcal{X} \to \mathcal{Y}$, let $R(A) = \{g \in \mathcal{Y} : (\exists f \in \mathcal{X})(A(f) = g)\}$ denotes its *range* and $\pi_{clR(A)} : \mathcal{Y} \to clR(A)$ the *projection* of $\mathcal{Y}$ onto the

closure of $R(A)$ in $(\mathcal{Y}, \|.\|_{\mathcal{Y}})$. Recall that every continuous operator A between two Hilbert spaces has an *adjoint* A^* satisfying for all $f \in \mathcal{X}$ and all $g \in \mathcal{Y}$,

$$\langle f, A^* g\rangle_{\mathcal{X}} = \langle Af, g\rangle_{\mathcal{Y}}.$$

We can summarize properties of the pseudoinverse operator as follows (see, e.g., [14, pp. 37–46] and [4, pp. 56–60]):

If the range of A is closed, then there exists a unique continuous linear pseudoinverse operator $A^+ : \mathcal{Y} \to \mathcal{X}$ such that for every $g \in \mathcal{Y}$,

$$A^+(g) \in S(g)$$

$$\|A^+(g)\|_{\mathcal{X}} = \min_{f^o \in S(g)} \|f^o\|_{\mathcal{X}}$$

and for every $g \in \mathcal{Y}$, $AA^+(g) = \pi_{clR}(g)$ and

$$A^+ = (A^*A)^+A^* = A^*(AA^*)^+. \tag{2}$$

If the range of A is not closed, then A^+ is only defined for those $g \in \mathcal{Y}$, for which $\pi_{clR(A)}(g) \in R(A)$.

Although the dependence of pseudosolutions on data is continuous, small errors in data may lead to rather different solutions. A method of improving stability of solutions of ill-posed inverse problems, called *regularization*, was developed in 1960th. Among several types of regularization, the most popular one penalizes undesired solutions by adding a term called stabilizer. It is called *Tikhonov's regularization* due to Tikhonov's unifying formulation [28]. Tikhonov's regularization replaces the problem of minimization of the functional

$$\|A(.) - g\|^2_{\mathcal{Y}}$$

with minimization of

$$\|A(.) - g\|^2_{\mathcal{Y}} + \gamma\Psi(.),$$

where Ψ is a functional called *stabilizer* and the *regularization parameter* γ plays the role of a trade-off between an emphasis on the least square solution and the penalization expressed by Ψ.

A typical choice of a stabilizer is the square of the norm on $\mathcal{X}$, for which Tikhonov regularization minimizes the functional

$$\|A(.) - g\|^2_{\mathcal{Y}} + \gamma\|.\|^2_{\mathcal{X}}. \tag{3}$$

In contrast to the case of pseudosolutions, which in the infinite dimensional case may not exist for some data, for this stabilizer regularized solutions do always exist. For every *continuous linear operator* $A : \mathcal{X} \to \mathcal{Y}$ between two Hilbert spaces and for every $\gamma > 0$, there exists a unique operator

$$A^\gamma : \mathcal{Y} \to \mathcal{X}$$

such that for every $g \in \mathcal{Y}$,

$$\{A^\gamma(g)\} = \operatorname{argmin}(\mathcal{X}, \|A(.) - g\|^2_{\mathcal{Y}} + \gamma\|.\|^2_{\mathcal{X}})$$

and

$$A^\gamma = (A^*A + \gamma I_{\mathcal{X}})^{-1}A^* = A^*(AA^* + \gamma I_{\mathcal{Y}})^{-1}, \tag{4}$$

where $I_{\mathcal{X}}, I_{\mathcal{Y}}$ denote the identity operators. Moreover for every $g \in \mathcal{Y}$, for which $A^+(g)$ exists,

$$\lim_{\gamma \to 0} A^\gamma(g) = A^+(g)$$

(see, e.g., [4, pp.68-70] and [14, pp.74-76].

So even if the original inverse problem does not have a unique solution (and so it is ill-posed), for every $\gamma > 0$ the regularized problem has a unique solution. This is due to the uniform convexity of the functional $\|.\|^2_{\mathcal{Y}}$ (see, e.g., [21]). With γ going to zero, the solutions of the regularized problem converge to the normal pseudosolution $A^+(g)$.

3 Minimization of error functionals

In learning theory, learning from data has been modeled as an optimization problem of *minimization of a functional* defined by the training data over a set of functions computable by a given computational model. The training data are described by a probability measure and a sample of input-output data.

Formally, for X a *compact* subset of $\mathbb{R}^d$ and Y a *bounded* subset of $\mathbb{R}$, and ρ a non degenerate (no non-empty open set has measure zero) *probability measure* on $Z = X \times Y$, the *expected error functional* (sometimes also called expected risk or theoretical error) $\mathcal{E}_\rho$ is defined for every f in the set $\mathcal{M}(X)$ of all bounded ρ-measurable functions on X as

$$\mathcal{E}_\rho(f) = \int_Z (f(x) - y)^2 \, d\rho\,.$$

For a *sample of input-output pairs of data* (called a training set) $z = \{(u_i, v_i) \in X \times Y, i = 1, \ldots, m\}$, a functional $\mathcal{E}_z$, called *empirical error*, is defined for a function $f : X \to \mathbb{R}$ as

$$\mathcal{E}_z(f) = \frac{1}{m}\sum_{i=1}^{m}(f(u_i) - v_i)^2.$$

To apply tools from approximation theory to investigation of optimization of these two functionals, we express them in terms of distances from certain "optimal" functions.

For the expected error functional $\mathcal{E}_\rho$, this function is the *regression function* f_ρ defined for $x \in X$ as

$$f_\rho(x) = \int_Y y \, d\rho(y|x),$$

where $\rho(y|x)$ is the *conditional (w.r.t. x) probability measure* on Y. Let ρ_X denote the *marginal probability measure* on X defined for every $S \subseteq X$ as $\rho_X(S) = \rho(\pi_X^{-1}(S))$, where $\pi_X : X \times Y \to X$ denotes the projection, and $\mathcal{L}^2_{\rho_X}(X)$ denotes the Lebesgue space of all functions satisfying $\int_X f^2 d\rho_X < \infty$ with the $\mathcal{L}^2_{\rho_X}$-norm denoted by $\|.\|_{\mathcal{L}^2}$. It is easy to see and well-known that the regression function f_ρ is the minimum point of $\mathcal{E}_\rho$ over $\mathcal{M}(X)$. Moreover, for every $f \in \mathcal{L}^2_{\rho_X}(X)$ [9, p.5]

$$\mathcal{E}_\rho(f) = \int_X (f(x) - f_\rho(x))^2 d\rho_X + \sigma_\rho^2 = \|f - f_\rho\|^2_{\mathcal{L}^2} + \sigma_\rho^2, \tag{5}$$

So minimization of the expected error functional $\mathcal{E}_\rho$ over some hypothesis subset of $\mathcal{L}^2_{\rho_X}(X)$ is a search for a function in such a set, which minimizes $\mathcal{L}^2_{\rho_X}$-distance from the regression function f_ρ.

Also the empirical error functional can be represented in terms of a distance from a certain function. Let $X_u = \{u_1, \ldots, u_m\}$ and $h_z : X_u \to Y$ be defined as

$$h_z(u_i) = v_i.$$

For $X \subseteq \mathbb{R}^d$ containing X_u and $f : X \to \mathbb{R}$, denote

$$f_u = f_{|X_u} : X_u \to \mathbb{R}.$$

Let $\|.\|_{2,m}$ denote the weighted l^2-norm on $\mathbb{R}^m$ defined as $\|x\|^2_{2,m} = \frac{1}{m}\sum_{i=1}^m x_i^2$. Then

$$\mathcal{E}_z(f) = \frac{1}{m}\|f_u - h_z\|_{2,m}. \tag{6}$$

So minimization of the empirical error $\mathcal{E}_z$ is a search for a function in a hypothesis set, which has a smallest l^2_m-distance from the function h_z defined by the sample z of input-output pairs of data.

4 Inverse problems in learning

We show that minimizations of the expected and the empirical error functionals can be studied as inverse problems. But the operators modeling dependence of data (the sample function h_z and the regression function f_ρ, resp.) on solutions (input-output functions) differ from operators describing inverse problems from physics. They are not defined as integrals, but they are defined as the *evaluation operator* at the sample of data z and as the *inclusion* of the hypothesis space into $\mathcal{L}^2_\rho(X)$.

To apply theory of inverse problems to learning, we have to assume that we search for solutions of a given learning task in some Hilbert space, denoted by $(\mathcal{H}, \|.\|_\mathcal{H})$, formed by functions defined on $X \subset \mathbb{R}^d$. For the input vector $u = \{u_1, \ldots, u_m\}$ from the sample of data $z = \{(u_i, v_i) \,|\, i - 1, \ldots, m\}$, let

$$L_u : (\mathcal{H}, \|.\|) \to (\mathbb{R}^m, \|.\|_{2,m})$$

denote the *evaluation operator* defined for all $f \in \mathcal{H}$ as

$$L_u(f) = (f(u_1), \ldots, f(u_m)).$$

It is easy to check that for every f on X,

$$\mathcal{E}_z(f) = \frac{1}{m}\sum_{i=1}^m (f(u_i) - v_i)^2 = \|L_u(f) - v\|^2_{2,m}. \tag{7}$$

So the empirical error $\mathcal{E}_z$ can be represented as

$$\mathcal{E}_z(.) = \|L_u(.) - v\|^2_{2,m}$$

and thus its minimization over $\mathcal{H}$ is equivalent to the inverse problem (1) defined by the *evaluation operator* L_u.

To express minimization of the expected error as an inverse problem, we use the *inclusion operator*

$$J : (\mathcal{H}, \|.\|_\mathcal{H}) \to (\mathcal{L}^2_{\rho_X}(X), \|.\|_{\mathcal{L}^2_{\rho_X}}).$$

By (5), we have

$$\mathcal{E}_\rho(f) = \|f - f_\rho\|^2_{\mathcal{L}^2} + \sigma_\rho^2 = \|J(f) - f_\rho\|^2_{\mathcal{L}^2} + \sigma_\rho^2. \tag{8}$$

So the expected error $\mathcal{E}_\rho$ can be represented as

$$\mathcal{E}_\rho(.) = \|J(.) - f_\rho\|^2_{\mathcal{L}^2} + \sigma_\rho^2$$

and its minimization is equivalent to the inverse problem (1) defined by the *inclusion operator* J.

To take advantage of the formulas (2) and (4) to characterize pseudosolutions and regularized solutions of these two inverse problems, we need to find some hypothesis Hilbert spaces, on which both the operators J and L_u are *continuous*. In the next section, we show that such spaces not only do exist, but in addition to continuity of both these operators, their norms can play roles of stabilizers penalizing various types of oscillations.

5 Hypothesis spaces defined by kernels

There exists a large class of Hilbert spaces, on which *all evaluation functionals are continuous*. Spaces from this class are called *reproducing kernel Hilbert spaces (RKHSs)*. They were formally defined by Aronszajn [2], but their theory includes many classical results on positive definite functions, matrices and integral operators with kernels. In data analysis, kernels were first used by Parzen [23] and Wahba [29], who applied them to data smoothing by splines. Girosi [13]

used squares of norms on RKHS as stabilizers penalizing high-frequency oscillations of the Fourier transform.

Aizerman, Braverman and Rozonoer [1] used kernels (under the name potential functions) to solve classification tasks by embedding input spaces into higher dimensional Hilbert spaces. Such transformation of geometry increase chances for linear separation of more types of data. Boser, Guyon and Vapnik [5] and Cortes and Vapnik [6] farther developed this classification method into the concept of the *support vector machine*, which is a one-hidden-layer network with kernel units in the hidden layer and one threshold output unit.

Kůrková [18], [19] and De Vito et al. [10] showed that RKHS are perfectly suited for applications of theory of inverse problems to minimization of empirical and expected error functionals.

So use of kernels in data analysis has three reasons: (1) norms on RKHSs can play roles of undesirable attributes of input-output functions, the penalization of which improves generalization, (2) kernels can define mappings of input spaces into feature spaces, where data to be classified can be separated linearly, and (3) as all evaluation functionals and inclusion to $\mathcal{L}^2_{\rho_X}(X)$ are continuous, theory of inverse problems can be applied to describe pseudosolutions and regularized solutions of learning tasks formulated as minimization of error functionals over RKHSs.

A variety of kernel methods and algorithms are of current interest, and their potential uses are wide ranging; see, e.g., the recent applications-oriented monographs by Schöllkopf and Smola [27] and by Cristianini and Shawe -Taylor [8], a theoretical article by Cucker and Smale's [9] or a brief survey by Poggio and Smale [26].

Aronszajn [2] defined a RKHS as a Hilbert space formed by functions on a nonempty set X such that for every $x \in X$, the evaluation functional $\mathcal{F}_x$, defined for any f in the Hilbert space as

$$\mathcal{F}_x(f) = f(x),$$

is bounded (continuous)(see also [3]).

RKHSs can be characterized in terms of *kernels*, which are *symmetric positive semidefinite functions* $K : X \times X \to \mathbb{R}$, i.e., functions satisfying for all m, all $(w_1, \ldots, w_m) \in \mathbb{R}^m$, and all $(x_1, \ldots, x_m) \in X^m$,

$$\sum_{i,j=1}^{m} w_i\, w_j\, K(x_i, x_j) \geq 0.$$

A kernel is *positive definite* if for any distinct $x_1, \ldots, x_m$

$$\sum_{i,j=1}^{m} w_i w_j K(x_i, x_j) = 0$$

implies that for all $i = 1, \ldots, m$, $w_i = 0$. In such a case $\{K_x : x \in X\}$ is a linearly independent set.

By the Riesz Representation Theorem [12, p. 200], for every $x \in X$ there exists a unique element $K_x \in X$, called the *representer* of x, such that

$$\mathcal{F}_x(f) = \langle f, K_x \rangle$$

for all $f \in X$ (this property is called the *reproducing property*). It is easy to check that the function $K : X \times X \to \mathbb{R}$ defined for all $x, y \in X$ as

$$K(x, y) = \langle K_x, K_y \rangle$$

is a kernel.

On the other hand, every kernel $K : X \times X \to \mathbb{R}$ generates an RKHS denoted by

$$\mathcal{H}_K(X).$$

It is formed by linear combinations of functions from $\{K_x : x \in X\}$ and pointwise limits of all Cauchy sequences of these linear combinations with respect to the norm $\|.\|_K$ induced by the inner product defined on generators as

$$\langle K_x, K_y \rangle_K = K(x, y).$$

A paradigmatic example of a kernel is the *Gaussian kernel*

$$K_b(x, y) = e^{-b\|x-y\|^2}$$

on $\mathbb{R}^d \times \mathbb{R}^d$. For this kernel, the space $\mathcal{H}_{K_b}(\mathbb{R}^d)$ contains all functions computable by radial-basis function networks with a fixed width equal to b.

Other examples of kernels are the *Laplace kernel*

$$K(x, y) = e^{-\|x-y\|},$$

homogeneous polynomial of degree p

$$K(x, y) = \langle x, y \rangle^p,$$

where $\langle \cdot, \cdot \rangle$ is any inner product on $\mathbb{R}^d$,
inhomogeneous polynomial of degree p

$$K(x, y) = (1 + \langle x, y \rangle)^p,$$

and

$$K(x, y) = (a^2 + \|x - y\|^2)^{-\alpha}$$

with $\alpha > 0$ [9, p. 38].

The role of $\|.\|^2_K$ as a stabilizer can be illustrated by two examples of classes of kernels. The first one is formed by *Mercer kernels*, i.e., *continuous, symmetric, positive semidefinite functions* $K : X \times X \to \mathbb{R}$, where $X \subset \mathbb{R}^d$ is *compact*.

For a Mercer kernel K, $\|f\|^2_K$ can be expressed using eigenvectors and eigenvalues of the compact linear operator

$$L_K : \mathcal{L}^2_{\rho_X}(X) \to \mathcal{L}^2_{\rho_X}(X)$$

defined for every $f \in \mathcal{L}^2_{\rho_X}(X)$ as

$$L_K(f)(x) = \int_X f(y)\, K(x,y)\, dy.$$

By the Mercer Theorem [9, p. 34]

$$\|f\|^2_K = \sum_{i=1}^{\infty} \frac{c_i^2}{\lambda_i},$$

where the λ_i's are the eigenvalues of L_K and the c_i's are the coefficients of the representation $f = \sum_{i=1}^{\infty} c_i \phi_i$, where $\{\phi_i\}$ is the orthonormal basis of $\mathcal{H}_K(X)$ formed by the eigenvectors of L_K. Note that the sequence $\{\lambda_i\}$ is either finite or convergent to zero. Thus, the stabilizer $\|.\|^2_K$ penalizes functions, for which the sequences of coefficients $\{c_i\}$ do not converge to zero sufficiently quickly, and so it plays a role of a high-frequency filter.

The second class of kernels illustrating the role of $\|.\|^2_K$ as a stabilizer consists of *convolution kernels*, i.e., kernels

$$K(x,y) = k(x-y)$$

defined as translations of a function $k : \mathbb{R}^d \to \mathbb{R}$, for which the Fourier transform $\tilde{k}$ is positive. For such kernels, the value of the stabilizer $\|.\|^2_K$ at any $f \in \mathcal{H}_K(\mathbb{R}^d)$ can be expressed as

$$\|f\|^2_K = \frac{1}{(2\,\pi)^{d/2}} \int_{\mathbb{R}^d} \frac{\tilde{f}(\omega)^2}{\tilde{k}(\omega)}\, d\omega \tag{9}$$

[13], [27, p. 97]. So the function $1/\tilde{k}$ plays a role analogous to that of the sequence $\{1/\lambda_i\}$ in the case of a Mercer kernel. An example of a convolution kernel with a positive Fourier transform is the Gaussian kernel.

6 Pseudosolutions and regularized solutions of learning tasks

As on every RKHS $\mathcal{H}_K(X)$, both the inclusion and evaluation operators describing learning from data as inverse problems are continuous, we can take advantage of the formulas (2) and (4) to describe their pseudosolutions and regularized solutions. The pseudosolution of the inverse problem described by the operator L_u can be expressed as

$$f^+ = L_u^+(v) = \sum_{i=1}^{m} c_i K_{u_i}, \tag{10}$$

where $c = (c_1, \ldots, c_m)$ satisfies

$$c = \mathcal{K}[u]^+ v, \tag{11}$$

with $\mathcal{K}[u]$ the *Gram matrix of the kernel K with respect to the vector u* defined as

$$\mathcal{K}[u]_{i,j} = K(u_i, u_j).$$

So the minimum of the empirical error $\mathcal{E}_z$ over $\mathcal{H}_K(X)$ is achieved at the function f^+, which is a linear combination of the representers $K_{u_1}, \ldots, K_{u_m}$ determined by the input data $u_1, \ldots, u_m$. Thus f^+ can be interpreted as an *input-output function of a neural network with one hidden layer of kernel units and a single linear output unit.*

For example, for the Gaussian kernel, f^+ is an input-output function of the Gaussian radial-basis function network with units centered at the input data $u_1, \ldots, u_m$. The coefficients $c = (c_1, \ldots, c_m)$ of the linear combination (corresponding to network output weights) can be computed by solving the system of linear equations (11).

Similarly as the function $f^+ = \sum_{i=1}^{m} c_i K_{u_i}$ minimizing the empirical error is a linear combination of the functions $K_{u_1}, \ldots, K_{u_m}$ defined by the input data $u_1, \ldots, u_m$, by (4) the regularized solution f^γ is of the form

$$f^\gamma = \sum_{i=1}^{m} c_i^\gamma K_{u_i}, \tag{12}$$

where $c^\gamma = (c_1^\gamma, \ldots, c_m^\gamma)$ satisfies

$$c^\gamma = (\mathcal{K}[u] + \gamma m \mathcal{I})^{-1} v. \tag{13}$$

So both the functions f^+ and f^γ minimizing $\mathcal{E}_z$ and $\mathcal{E}_z + \|.\|^2_K$, resp., are linear combinations of the representers $K_{u_1}, \ldots, K_{u_m}$ of input data $u_1, \ldots, u_m$, but the coefficients of the two linear combinations are different. For the pseudosolution, $c = (c_1, \ldots, c_m)$ is the image of the output vector $v = (v_1, \ldots, v_m)$ under Moore-Penrose pseudoinverse of the matrix $\mathcal{K}[u]$, while for the regularized solution, $c^\gamma = (c_1^\gamma, \ldots, c_m^\gamma)$ is the image of v under the inverse operator $(\mathcal{K}[u] + \gamma m \mathcal{I})^{-1}$.

This clearly shows the role of regularization – it makes *dependence of coefficients on output data more stable*. Growth of the regularization parameter γ leads from "under smoothing" to "over smoothing". However, the size of γ is constrained by the requirement of fitting f^γ to the sample of empirical data z, so γ cannot be too large.

The effect of regularization of the evaluation operator L_u depends on behavior of the eigenvalues of the Gram matrix $\mathcal{K}[u]$. Stability analysis of the regularized problem $\|L_u(.) - v\|_{2,m} + \gamma \|.\|^2_K$ can be investigated in terms of the finite dimensional problem of

regularization of the Gram matrix $\mathcal{K}[u]$ with the parameter $\gamma' = \gamma m$, because the coefficient vector c^γ satisfies $c^\gamma = (\mathcal{K}[u] + \gamma m \mathcal{I})^{-1}$. For K positive definite, the row vectors of the matrix $\mathcal{K}[u]$ are linearly independent. But when the distances between the data $u_1, \ldots, u_m$ are small, the row vectors might be nearly parallel and the small eigenvalues of $\mathcal{K}[u]$ might cluster near zero. In such a case, small changes of v can cause large changes of f^+. Various types of ill-posedness of $\mathcal{K}[u]$ can occur: the matrix can be *rank-deficient* when it has a cluster of small eigenvalues and a gap between large and small eigenvalues, or the matrix can represent a *discrete ill-posed problem*, when its eigenvalues gradually decay to zero without any gap in its spectrum [16].

Practical applications of learning algorithms based on theory of inverse problems are limited to such samples of data and kernels, for which iterative methods for solving systems of linear equations $c = \mathcal{K}[u]^+ v$ and $c^\gamma = (\mathcal{K}[u] + \gamma m \mathcal{I})^{-1} v$ are computationally efficient (see, e.g., [26] for references to such applications).

In typical neural-network algorithms, networks with the number of hidden units n much smaller than the size m of the training set are used [17]. In [20] and [21], estimates of the speed of convergence of infima of the empirical error over sets of functions computable by such networks to the global minimum $\mathcal{E}_z(f^\gamma)$ were derived. For reasonable data sets, such convergence is rather fast.

Acknowledgement

This work was partially supported by the project 1ET100300517 of the program Information Society of the National Research Program of the Czech Republic and the Institutional Research Plan AV0Z10300504.

Reference

1. Aizerman M.A., Braverman E.M., Rozonoer, L.I., Theoretical Foundations of Potential Function Method in Pattern Recognition Learning. Automation and Remote Control, f 28, 1964, 821–837
2. Aronszajn N., Theory of Reproducing Kernels. Transactions of AMS, 68, 1950, 33–404
3. Berg C., Christensen J.P.R., Ressel P., Harmonic Analysis on Semigroups. Springer-Verlag, New York, 1984
4. Bertero M., Linear Inverse and Ill-Posed problems. Advances in Electronics and Electron Physics, 75, 1989, 1–120
5. Boser B., Guyon I., Vapnik V.N., A Training Algorithm for Optimal Margin Clasifiers. In: Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory, D. Haussler (ed.), ACM Press, 1992, 144–152
6. Cortes C., Vapnik V.N., Support-Vector Networks. Machine Learning, 20, 1995, 273–297
7. Courant R., Hilbert D., Methods of Mathematical Physics. New York, Wiley, vol. 2., 1962
8. Cristianini N., Shawe-Taylor J., An Introduction to Support Vector Machines. Cambridge, Cambridge University Press, 2000
9. Cucker F., Smale S., On the Mathematical Foundations of Learning. Bulletin of the AMS, 39, 2002, 1–49
10. De Vito E., Rosasco L., Caponnetto A., De Giovannini U., Odone F., Learning from Examples as an Inverse Problem. Journal of Machine Learning Research, 6, 2005, 883–904
11. Engl H.W., Hanke M., Neubauer A., Regularization of Inverse Problems. Dordrecht, Kluwer, 2000
12. Friedman A., Modern Analysis. New York, Dover, 1982
13. Girosi F., An Equivalence between Sparse Approximation and Support Vector Machines. Neural Computation, 10, 1998, 1455–1480 (AI Memo No 1606, MIT)
14. Groetch C.W., Generalized Inverses of Linear Operators. Dekker, New York, 1977
15. Hadamard J.: Sur les problèmes aux dérivées partielles et leur signification physique. Bulletin of University of Princeton **13** (1902) 49
16. Hansen P.C., Rank-Deficient and Discrete Ill-Posed Problems. SIAM, Philadelphia, 1998
17. Kecman V., Learning and Soft Computing. Cambridge, MIT, Press, 2001
18. Kůrková V., Learning from Data as an Inverse Problem. In: COMPSTAT 2004 – Proceedings on Computational Statistics, J. Antoch (ed.), Heidelberg, Physica-Verlag/Springer, 2004, 1377–1384
19. Kůrková V.: Neural Network Learning as an Inverse Problem. Logic Journal of IGPL, 13, 2005, 551–559
20. Kůrková V., Sanguineti M., Error Estimates for Approximate Optimization by the Extended Ritz Method. SIAM Journal on Optimization, 15, 2005, 461–487
21. Kůrková V., Sanguineti M., Learning with Generalization Capability by Kernel Methods with Bounded Complexity. Journal of Complexity, 21, 2005, 350–367
22. Moore E.H., Abstract. Bull. Amer. Math. Soc., 26, 1920, 394–395
23. Parzen E., An Approach to Time Series Analysis. Annals of Math. Statistics, 32, 1966, 951–989
24. Penrose R., A Generalized Inverse for Matrices. Proc. Cambridge Philos. Soc., 51, 1955, 406–413
25. Poggio T., Girosi F., Networks for Approximation and Learning. Proceedings IEEE, 78, 1990, 1481–1497
26. Poggio T., Smale S., The Mathematics of Learning: Dealing with Data. Notices of the AMS, 50, 2003, 536–544
27. Schölkopf B., Smola A.J., Learning with Kernels – Support Vector Machines, Regularization, Optimization and Beyond. Cambridge, MIT Press, 2002
28. Tikhonov A.N., Arsenin V.Y., Solutions of Ill-Posed Problems. Washington, D.C., W.H. Winston, 1977
29. Wahba G., Splines Models for Observational Data. Philadelphia, SIAM, 1990

ITAT'06

Information Technology – Applications and Theory

ORIGINAL SCIENTIFIC PAPERS

Turingovské vzory v XSLT programech*

David Bednárek

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

Abstrakt *Již několik let je známo, že XSLT programy jsou Turingovsky úplné a že problém zastavení či jiná statická analýza XSLT programu je algoritmicky nerozhodnutelný problém. Důkazy těchto tvrzení ovšem využívají složité konstrukce, které se v běžných XSLT programech nevyskytují. Tento článek definuje pojem grafu závislostí pro XSLT program (v kombinaci s XML-schematem jeho vstupu) a dva vzory, jejichž přítomnost v grafu závislostí indikuje, že by z hlediska závislostí program mohl být simulátorem Turingova stroje. Podstatné je pak pozorování, že všechny složité konstrukce známé z důkazů nerozhodnutelnosti statické analýzy XSLT programů jsou speciálním případem těchto vzorů, zatímco v XSLT programech běžného určení se tyto vzory vyskytují naprosto výjimečně. To dává naději, že po odstínění případů, obsahujících tyto vzory, je řada problémů statické analýzy na běžných XSLT programech algoritmicky řešitelná.*

1 Úvod

Řada publikací se v poslední době zaměřuje na statickou analýzu XSLT programů jako jednoho z nejvýznamnějších programovacích jazyků rodiny XML. Kombinace funkcionálního jazyka se stromy a stromovými gramatikami je relativně nová a řada problémů statické analýzy je tedy otevřená, včetně otázky vhodného matematického modelu XSLT programu. Zatímco pro samotné X-Path výrazy existuje řada velmi odlišných modelů včetně atributových gramatik [2,1], dosud prezentované modely XSLT jsou buďto vázány na podstatně omezený fragment XSLT (nejčastěji pro XSLT bez proměnných), nebo jsou naopak obtížně analyzovatelné.

Tento článek předkládá nový, pravděpodobně dosud nepublikovaný, způsob modelování XSLT programu pomocí dvourozměrné atributové gramatiky, což je pojem poměrně přímočaře odvozený z přítomnosti dvojice stromů - stromu vstupního XML dokumentu a stromu reprezentujícího postup výpočtu na něm z hlediska (rekurzivního) volání XSLT šablon. Stejně jako u klasických atributových gramatik je zde oddělena definice relace závislostí atributů od funkcí počítajících skutečné hodnoty atributů, což dovoluje zkoumat dvourozměrné atributové gramatiky z hlediska grafů závislostí, vytvořených atributovou gramatikou pro konkrétní stromy.

Zkoumání grafů závislostí vedlo k identifikaci dvou vzorů, zvaných „páska“ a „mříž“, jejichž přítomnost v grafu závislostí indikuje, že analyzovaná atributová gramatika je z hlediska závislostí dostatečná na simulaci lineárně omezeného Turingova stroje (tedy, že existuje jiná gramatika se stejnými grafy závislostí, která je zmíněným simulátorem).

Zbytek článku je rozdělen takto: Sekce 2 definuje potřebný matematický aparát, zejména pak pojem kartézského součinu stromů, na něm pracující dvourozměrné atributové gramatiky a grafů závislostí indukovaných touto gramatikou. Následující kapitola popisuje postup konverze DTD nebo XML-schematu a XSLT programu na dvourozměrnou atributovou gramatiku. Kapitola 4 pak definuje výše uvedené Turingovské vzory na příkladech i na zpětně rekonstruovaných XSLT programech.

2 Dvourozměrné atributové gramatiky

Dvourozměrná atributová gramatika je mechanismus, doplňující hodnoty atributů ke kartézskému součinu dvou stromů. Při analýze XSLT bude jedním z těchto stromů vstupní XML dokument, druhý strom bude reprezentovat vzájemná volání XSLT šablon v průběhu zpracování daného vstupu. Z tohoto důvodu budeme tyto dva stromy, jejich gramatiky a další pojmy označovat prefixy či indexy D (data) a C (code).

2.1 Stromy a gramatiky

Přípustné tvary anotovaného stromu jsou definovány upravenou formou gramatiky $G = (\Pi, R, P, S)$. Konečná množina Π definuje neterminální symboly (terminální symboly pro určení tvaru stromů nejsou zapotřebí – listy stromů budou tvořeny pravidly s prázdnou pravou stranou). $S \in \Pi$ je počáteční neterminál.

Konečná množina R obsahuje jména pravidel, jejichž obsah je definován funkcí $P : R \to \Pi \times \Pi^\star$. Tato definice připouští existenci více pravidel se stejným obsahem, proto budeme ve stromech uvádět namísto neterminálů jména použitých pravidel. Neterminály

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

pak představují rozhraní, přes která je možno spojovat jednotlivá pravidla.

Anotovaný strom (vzhledem ke gramatice G) je struktura $T = (V, E, r, i, root)$ s vrcholy V, orientovanými hranami E a kořenem $root \in V$, doplněný o jména pravidel $r : V \to R$ a indexy hran $i : E \to \mathbb{N}$. Anotovaný strom je přípustný vzhledem ke gramatice G za těchto podmínek:

- Kořen je označen pravidlem $r = r(root)$ s počátečním neterminálem na levé straně: $X_{r,0} = S$
- Je-li vrchol $a \in V$ označen pravidlem $r = r(a)$ ve tvaru $X_{r,0} \to X_{r,1}X_{r,2}\dots X_{r,n(r)}$, pak z tohoto vrcholu vede právě $n(r)$ hran, jejichž indexy $i(e)$ tvoří množinu $1, 2, \dots n(r)$, a pro každou hranu $e = \langle a, b\rangle$ s indexem $i = i(e)$ platí $X_{r',0} = X_{r,i}$, kde $r' = r(b)$.

2.2 Kartézský součin stromů

Mějme dvojici D-stromu a C-stromu

$$T_D = (V_D, E_D, r_D, i_D, root_D)$$
$$T_C = (V_C, E_C, r_C, i_C, root_C)$$

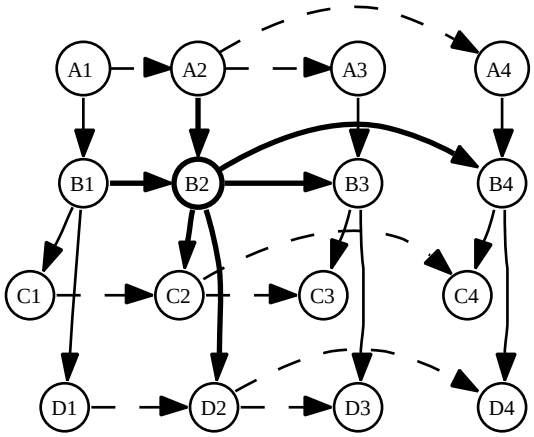
které jsou přípustné vzhledem ke dvojici D-gramatiky a C-gramatiky

$$G_D = (\Pi_D, R_D, P_D, S_D)$$
$$G_C = (\Pi_C, R_C, P_C, S_C)$$

Kartézský součin stromů T_D a T_C je definován jako orientovaný acyklický graf $TT = (V, E, r, i, root)$, kde

$$\begin{aligned}
V &= V_D \times V_C \\
E &= E^D \cup E^C \\
E^D &= \{\langle\langle v_1^D, v^C\rangle, \langle v_2^D, v^C\rangle\rangle \\
&\quad \mid \langle v_1^D, v_2^D\rangle \in E_D \wedge v^C \in V_C\} \\
E^C &= \{\langle\langle v^D, v_1^C\rangle, \langle v^D, v_2^C\rangle\rangle \\
&\quad \mid v^D \in V_D \wedge \langle v_1^C, v_2^C\rangle \in E_C\} \\
r &: V \to R_D \times R_C \\
&\quad r(\langle v^D, v^C\rangle) = \langle r_D(v^D), r_C(v^C)\rangle \\
i &: E \to \mathbb{N} \\
&\quad i(\langle\langle v_1^D, v^C\rangle, \langle v_2^D, v^C\rangle\rangle) = i_D(\langle v_1^D, v_2^D\rangle) \\
&\quad i(\langle\langle v^D, v_1^C\rangle, \langle v^D, v_2^C\rangle\rangle) = i_C(\langle v_1^C, v_2^C\rangle) \\
root &= \langle root_D, root_C\rangle
\end{aligned}$$

V takto definovaném kartézském součinu jsou hrany dvou druhů: D-hrany E^D a C-hrany E^C. Do každého vrcholu s výjimkou vrcholů tvaru $\langle v^D, root_C\rangle$, $\langle root_D, v^C\rangle$ a kořene vede jedna D-hrana a jedna C-hrana. Každý vrchol $v = \langle v^D, v^C\rangle$ má $n(r_D(v^D))$ odchozích D-hran a $n(r_C(v^C))$ odchozích C-hran.



Obrázek 1. Kartézský součin D-stromu a C-stromu.

Obrázek 1 ukazuje příklad kartézského součinu jednoduchých stromů. D-hrany jsou kresleny svisle, C-hrany čárkovaně vodorovně. Okolí uzlu $B2 = \langle B, 2\rangle$ je zvýrazněno.

2.3 Dvourozměrné atributové gramatiky

Dvourozměrná atributová gramatika představuje analogii klasické atributové gramatiky ve světě kartézských součinů stromů. Zatímco obyčejné atributové gramatiky dělí atributy na dědičné (Inh) a syntetizované (Syn) podle směru předávání informace ve stromě, v kartézském součinu dvou stromů přicházejí v úvahu čtyři směry předávání, a tedy čtyři skupiny atributů označované $DInh$, $DSyn$, $CInh$ a $CSyn$. Pro zkoumání XSLT programů nám však postačí pouze první tři skupiny, neboť XSLT šablony nemají výstupní parametry ani jiné možnosti předávání informací zdola nahoru ve stromě volání. Následující definice dvourozměrné atributové gramatiky tedy pro zjednodušení $CSyn$ neuvažuje (a měla by tedy možná nést název jeden-a-půl-rozměrná).

Dvourozměrná atributová gramatika

$$AG = (G_D, G_C, DInh, DSyn, CInh, Dep, W, f)$$

je především nadstavbou dvojice gramatik G_D a G_C, které společně určují kartézské součiny stromů, které budou opatřovány atributy.

Každý uzel $v = \langle v^D, v^C\rangle$ kartézského součinu bude opatřen třemi druhy atributů, přičemž konkrétní množina atributů závisí na neterminálu, ke kterému uzel patří. Pokud je tedy uzlu v přiřazena dvojice pravidel $r^D = r_D(v^D)$ a $r^C = r_C(v^C)$, pak je množina atributů určena na základě pravidel a jejich levostranných neterminálů $X^D = X_{r^D,0}$ a $X^C = X_{r^C,0}$, a to takto:

$DInh(X^D, r^C)$ - konečná množina atributů děděných ve směru D-hran

$DSyn(X^D, r^C)$ - konečná množina atributů syntetizovaných proti směru D-hran

$CInh(r^D, X^C)$ - konečná množina atributů děděných ve směru C-hran

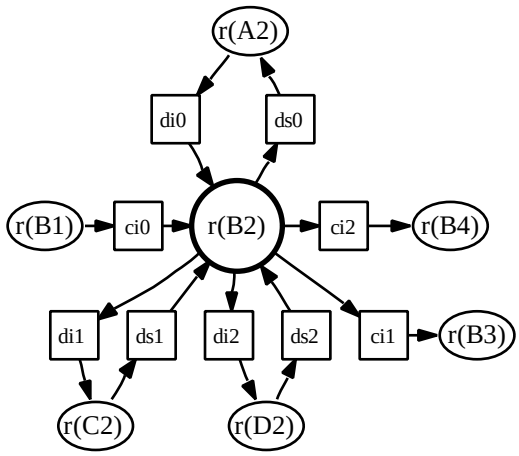
V okolí uzlu v jsou závislosti určeny atributovými pravidly spřaženými s dvojicí pravidel gramatik $r^D = r_D(v^D)$ a $r^C = r_C(v^C)$.

$$r^D : X_0^D \to X_1^D X_2^D \dots X_{n^D}^D$$
$$r^C : X_0^C \to X_1^C X_2^C \dots X_{n^C}^C$$

Podobně jako u klasických atributových gramatik definujeme vstupní a výstupní atributové výskyty v pravidle $r = \langle r^D, r^C \rangle$:

$$\begin{aligned} In(r) &= \{\langle a, 0\rangle \mid a \in DInh(X_0^D, r^C)\} \\ &\cup \{\langle a, 0\rangle \mid a \in CInh(r^D, X_0^C)\} \\ &\cup \{\langle a, i\rangle \mid 1 < i < n^D \wedge a \in DSyn(X_i^D, r^C)\} \end{aligned}$$

$$\begin{aligned} Out(r) &= \{\langle a, 0\rangle \mid a \in DSyn(X_0^D, r^C)\} \\ &\cup \{\langle a, i\rangle \mid 1 < i < n^D \wedge a \in DInh(X_i^D, r^C)\} \\ &\cup \{\langle a, i\rangle \mid 1 < i < n^C \wedge a \in CInh(r^D, X_i^C)\} \end{aligned}$$



Obrázek 2. Interakce atributů ve dvourozměrné atributové gramatice.

Obrázek 2 ukazuje atributové pravidlo pro uzel $B2$ z obrázku 1 a jeho interakci s pravidly sousedních uzlů prostřednictvím vstupních atributových výskytů $\{di0, ci0, ds1, ds2\}$ a výstupních $\{ds0, di1, di2, ci1, ci2\}$.

Závislosti atributů v pravidle r jsou definovány relací

$$Dep(r) \subseteq In(r) \times Out(r)$$

Atributy nabývají hodnot z univerza W a hodnota každého výstupního atributového výskytu je definována funkcemi $f_{r,\langle a,i\rangle}$. Pro analýzu XSLT budeme používat booleovské atributy. Pro posouzení z hlediska Turingovských vzorů pak jsou rozhodující závislosti atributů a konkrétní funkce, jimiž jsou tyto závislosti naplněny, nezkoumáme. Proto budeme z atributových gramatik extrahovat závislostní část ve tvaru

$$DEP(AG) = (G_D, G_C, DInh, DSyn, CInh, Dep)$$

Grafem závislostí pro konkrétní kartézský součin stromů pak rozumíme graf sestrojený sjednocením lokálních závislostí Dep na atributových výskytech všech uzlů kartézského součinu. Graf závislostí obecně může obsahovat cyklus, níže prezentovaný postup konverze XSLT programu však vede na dvourozměrnou gramatiku, která cykly negeneruje, čímž je zajištěna jednoznačnost výpočtu hodnot atributů. Vzhledem k neexistenci syntetizovaných atributů na C-hranách se jednotlivé větve C-stromu vzájemně neovlivňují a je tedy možné posuzovat je odděleně. Hledané Turingovské vzory je tedy možné zkoumat na kartézských součinech D-stromu a některé z větví C-stromu.

3 Abstrakce XML-schemat a XSLT

V předchozí kapitole definované pojmy nám umožní vyjádřit všechny potřebné struktury z XML světa, tedy vstupní dokumenty, jejich schemata a XSLT programy.

3.1 XML dokument

XML dokument je strom neomezeného stupně, v našem modelu je převeden na strom stupně nejvýše dva standardním trikem „first-child/next-sibling“ popsaným v řadě publikací (např. [6]). Původní anotace XML-elementy je nahrazena jmény pravidel D-gramatiky, která vyjadřuje schéma stromu. To znamená, že náš stromový model dokumentu je (z hlediska anotace) závislý na použitém schématu.

3.2 Schema dokumentu

Pro specifikaci formátu XML dokumentů jsou používány formáty DTD a XML-Schema, z teoretického pohledu se navíc rozlišuje deterministická a nedeterministická verze schematu (viz [4]). Naše D-gramatiky jsou ekvivalentem nedeterministických stromových automatů, které jsou nejčastěji používány jako nejobecnější abstrakce všech forem DTD a XML-Schemat.

V převáděném DTD či XML-schematu jsou nejprve pro každý element či složený typ abstrahovány přípustné posloupnosti synů pomocí nedeterministických konečných automatů. Stavy těchto automatů se

stanou neterminály D-gramatiky a pro každý přechod $q, e \to q'$ jsou vytvořena některá z těchto čtyř pravidel:

$$R^{11}_{q,e} : q \to q''q'$$
$$R^{10}_{q,e} : q \to q''$$
$$R^{01}_{q,e} : q \to q'$$
$$R^{00}_{q,e} : q \to$$

q'' je počáteční stav automatu pro syny elementu e v kontextu stavu q. Pravidla R^{1x} budou generována, pokud ze stavu q'' existuje přechod, pravidla R^{0x}, pokud je stav q'' koncový. Generování pravidel R^{y1} resp. R^{y0} je analogicky podmíněno vlastnostmi stavu q'.

Z názvu pravidla $R^{yx}_{q,e}$ je zpětně možno odvodit element e, čímž je zajištěna možnost bezztrátové reprezentace XML dokumentu anotovaným stromem odpovídajícím této gramatice. Zajímavá je skutečnost, že u nedeterministických verzí schemat může být jeden dokument reprezentován různými stromy lišícími se použitými názvy pravidel (a tedy i použitými neterminály). Jak však uvidíme v následujících odstavcích, reprezentace XSLT programu se chová stejně ke všem pravidlům pro stejné e.

3.3 XSLT program

XSLT program je nejprve zbaven složitých konstrukcí apply-templates, choose, key, globálních proměnných apod. převodem na call-template, if a dosazením do X-Path výrazů. Úpravy tohoto druhu byly již popsány v literatuře [5,3]. Pro přehlednost a úsporu budeme pro zápis XSLT programů používat snadno srozumitelnou syntaxi ve stylu jazyka C.

Implicitní proměnná „.“ představující kontext je nahrazena explicitní proměnnou. Každá konstrukce „for-each($expr$)$stmt$“ je přitom nahrazena voláním $f(/, expr)$, přičemž šablona f je vytvořena podle tohoto vzoru:

```
template f($c,$e)
  if ($c)
    if ($c intersection $e) stmt($c)
    f($c/first-child::*, $e)
    f($c/next-sibling::*, $e)
```

V tomto přepisu proměnná $\$e$ obsahuje původní node-set v hlavičce for-each, proměnná $\$c$ nahrazuje původní kontext. Iterativní charakter původní konstrukce je nahrazen rekurzivním průchodem vstupním stromem a testem příslušnosti každého navštíveného uzlu vůči proměnné $\$e$. Osy first-child a next-sibling jsou rozšířením X-Path, a budou, stejně jako standarní osy, v následujícím kroku převedeny na atributová pravidla dvourozměrné atributové gramatiky.

Tento přepis by byl samozřejmě značně neefektivní z hlediska provádění XSLT programu, pro účely statické analýzy je ovšem vyhovující.

Vlastní převod XSLT programu na dvourozměrnou atributovou gramatiku je založen na těchto principech:

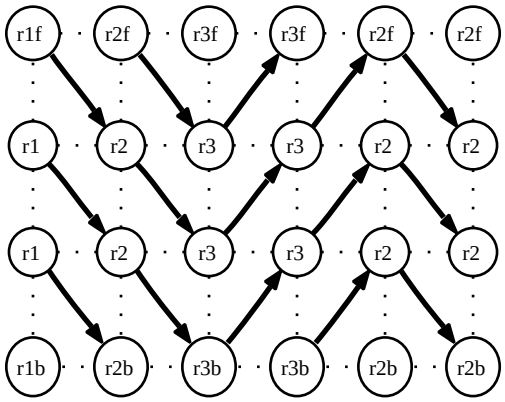
- Šablony jsou nejprve normalizovány (doplněním pomocných šablon) tak, aby každá obsahovala nejvýše jeden příkaz if.
- Normalizované šablony jsou nahrazeny neterminály C-gramatiky.
- Těla šablon neobsahujících větvení budou reprezentována jediným pravidlem C-gramatiky, šablonám s příkazem if budou odpovídat dvě pravidla C-gramatiky, reprezentující případy se splněnou a nesplněnou podmínkou.
- Posloupnost volání jiných šablon v těle šablony odpovídá pravé straně vygenerovaného pravidla C-gramatiky.
- Parametry šablon budou nahrazeny booleovskými atributy ze skupiny $CInh$. Hodnota atributu u uzlu $\langle v^D, v^C \rangle$ kartézského součinu D-stromu a C-stromu vyjadřuje přítomnost uzlu v^D vstupního dokumentu v příslušném parametru při volání procedury reprezentovaném uzlem v^C.
- Hodnoty X-Path výrazů ve výstupních XSLT příkazech value-of a copy-of budou nahrazeny booleovskými atributy ze skupiny $DSyn$ s obdobným významem jako u parametrů šablon. Tyto atributy lze chápat jako výstup zpracování vstupu dvourozměrnou atributovou gramatikou.
- Hodnoty booleovských X-Path výrazů v podmínkách příkazů if budou nahrazeny booleovským atributem ze skupiny $DSyn$ vyskytujícím se pouze u kořene D-stromu. Hodnota tohoto atributu bude při analýze chápána jako podmínka pro použitelnost příslušného C-pravidla.
- Mezivýsledky jednotlivých podvýrazů X-Path výrazů v těle šablony budou reprezentovány booleovskými atributy ze skupiny $DInh$ nebo $DSyn$ v závislosti na způsobu, jakým je podvýraz použit.
- Lokální proměnné budou vyřešeny podobným způsobem jako podvýrazy s tím rozdílem, že kvůli různosti způsobů použití může mít lokální proměnná dva reprezentanty v obou skupinách $DInh$ a $DSyn$.
- Booleovské podvýrazy testující neprázdnost množiny uzlů nezávislé na kontextu budou nahrazeny booleovským atributem ze skupiny $DSyn$, který postupně počítá disjunkci hodnot atributu reprezentujícího zkoumanou množinu. Výsledná hodnota tohoto atributu u kořene D-stromu odpovídá hodnotě podvýrazu.
- Booleovské podvýrazy testující neprázdnost množiny uzlů závislé na kontextu, tj. závislé podvýrazy

v X-path filtrech $a[b]$, budou reprezentovány booleovskými atributy u každého D-uzlu. Hodnota tohoto atributu odpovídá hodnotě podvýrazu v kontextu daného D-uzlu.

4 Turingovské vzory

4.1 Páska

Vzor nazvaný „páska“ odpovídá přímočaré simulaci Turingova stroje s páskou a hlavou. Páska je zde simulována vhodnou větví vstupního XML stromu, kroky stroje odpovídají rekurzivnímu volání XSLT procedur. Obsah pásky je zaznamenán pomocí XSLT proměnných a parametrů (typu node-set). V nejjednodušším případě stroje s abecedou $0, 1$ postačí jediná proměnná: Přítomnost uzlu v proměnné odpovídá jedničce v místě pásky příslušejícím k tomuto uzlu. Víceprvkové abecedy pak lze simulovat n-ticí proměnných.Vstupní data Turingova stroje jsou zaznamenána pomocí abecedy elementů na vybrané větvi vstupního XML, na základě níž se pak XSLT proměnné simulátoru inicializují.



Obrázek 3. Příklad vzoru „páska“.

Namísto posunu hlavy se v této simulaci rotuje páskou a hlava zůstává na stejném místě, a to rozdvojena na obou koncích vybrané větve. Pro simulaci Turingova stroje jsou nutné tyto schopnosti dvourozměrné atributové gramatiky:

- Schopnost přenést hodnotu všech atributů na dané větvi D-stromu oběma směry. Tento přenos nemusí být realizován jednou hranou C-stromu, ale posloupností hran. Posloupnosti přitom musejí být alespoň dvě - pro přenos ve směru D-hran a pro opačný směr.
- Schopnost přenést konečnou informaci mezi konci dané větve, tj. mezi polovinami rozdvojené hlavy. Tato schopnost je realizována volbou posloupnosti hran C-stromu z několika možných - tato volba tedy jednak ovlivňuje směr rotace pásky, jednak znak ukládaný při rotaci na vstupním konci pásky.

Obrázek 3 ilustruje chování vzoru „páska“. V některých sloupcích, tj. pro některá C-pravidla se hodnoty posouvají směrem nahoru, v jiných sloupcích, při jiné volbě C-pravidla se posouvají směrem dolů. Jeden z XSLT programů odpovídajících tomuto vzoru je tento:

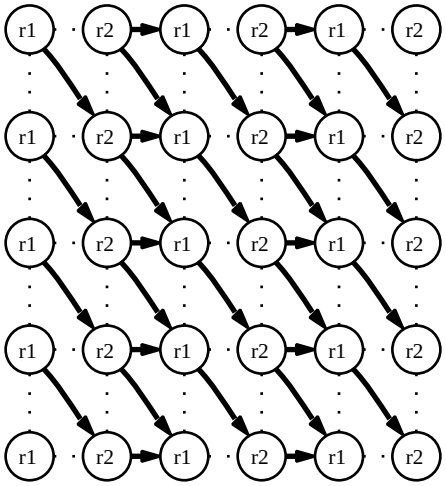
```
template r1($x)
  if($x/self::f and $x/self::b)
    r2(//f union $x/child::*)

template r2($x)
  if($x/self::f and $x/self::b)
    r2($x/child::*)
  if($x/self::f and not($x/self::b))
    r3(//f union $x/child::*)

template r3($x)
  if($x/self::f and $x/self::b)
    r3(//b union $x/parent::*)
  if(not($x/self::f) and $x/self::b)
    r2(//b union $x/parent::*)
```

Pro funkčnost vzoru je zapotřebí, aby schema vstupu dovolovalo neomezeně dlouhé větve tvaru $f \cdot a^\star \cdot b$ představující pásku. Abecedou pásky je množina 0,1 s jedničkami zaznamenanými přítomností v parametru $\$x$. Pro skutečnou simulaci lineárně omezeného Turingova stroje je samozřejmě nutné větši množství šablon a vhodná kombinace operátorů *not* v X-Path podmínkách.

4.2 Mříž

Vzor nazvaný „mříž“ je situace, kdy lze v grafech závislostí atributových výskytů nalézt čtverec neomezené velikosti, tvořený atributovými výskyty $a_{i,j}$ se závislostmi tvaru $a_{i,j} \to a_{i+1,j}$ a $a_{i,j} \to a_{i,j+1}$. Takový čtverec může být v grafech závislostí zmačkán do kosočtverce a otočen.

Výpočetní síly lineárně omezeného Turingova stroje lze u vzoru „mříž“ dosáhnout tehdy, pokud může atribut v mříži nést znak pásky i kód stavu, v případě booleovských atributů je tedy zapotřebí jejich znásobení. Pak je možno simulovat zachování obsahu na pásce ve směru diagonály původního čtverce, tedy $a_{i,j} \to a_{i+1,j+1}$ a zároveň posuny hlavy ve směrech $a_{i,j} \to a_{i,j+2}$ a $a_{i,j} \to a_{i+2,j}$.
Obrázek 4 ukazuje příklad vzoru „mříž“.Jeden z XSLT programů odpovídajících tomuto vzoru je tento:



Obrázek 4. Příklad vzoru „mříž“.

```
template r1($x)
  r2($x/child::*)

template r2($x)
  r1($x/self::* union $x/child::*)
```

Pro simulaci lineárně omezeného Turingova stroje je zapotřebí jednak větší počet stejně šířených parametrů šablon, jednak nahrazení operátoru *union* složitějšími výrazy s operátory *union* a *intersection* na kombinaci všech parametrů.

5 Závěr

Dvourozměrná atributová gramatika představuje formalismus, pomocí nějž lze modelovat XSLT program způsobem, který věrně odráží manipulaci s node-set parametry v původním programu. To je významný rozdíl proti většině dosud publikovaných prací z oboru statické analýzy XSLT, které buďto analýzu chování proměnných neumožňovaly, nebo omezovaly použité X-Path operace.

Za tuto schopnost samozřejmě platíme zvýšenou výpočetní silou modelu a tudíž algoritmickou nerozhodnutelností většiny problémů statické analýzy. Turingovské vzory „páska“ a „mříž“ uvedené v tomto článku ukazují dvě konkrétní příčiny této nerozhodnutelnosti.

Vzor „páska“ je možné detekovat algoritmem se složitostí $O(\exp(m^2 \cdot n^2))$ vzhledem k velikosti XSLT programu m a velikosti XML-schématu n. Pro detekci vzoru „mříž“ je znám algoritmus se složitostí $O(\exp(\exp(m^2 \cdot n^2)))$. Popis těchto algoritmů je bohužel za hranicemi prostorových možností tohoto článku.

Tato exponenciální a dvojitě exponenciální složitost je na první pohled odstrašující, v praxi se ovšem i takové algoritmy úspěšně používají - příkladem je již desítky let používaný algoritmus konstrukce LR(k) analyzátorů, v poslední době se pak objevuje řada algoritmů model-checkingu, tedy statické analýzy jednoduchých programů zapsaných různými procedurálními či neprocedurálními způsoby. Příčinou použitelnosti těchto exponenciálních algoritmů je na jedné straně skutečnost, že skutečná složitost na reálných datech je hluboko pod úrovní maximální složitosti, na druhé straně rostoucí schopnosti výpočetní techniky. Protože se jedná o problémy, jejichž vstupy jsou gramatiky či programy, je zde naděje, že i problémy statické analýzy XSLT včetně detekce Turingovských vzorů se budou chovat obdobně.

Výzkum Turingovských vzorů bude tedy pokračovat implementací algoritmů detekce a jejich aplikací na dostatečně reprezentativní množinu „reálných dat“. Výsledkem by mělo být jednak změření skutečné časové a prostorové náročnosti, jednak zjištění, jak častá je přítomnost těchto vzorů.

Hlavním důvodem zkoumání Turingovských vzorů je zatím neprokázaná hypotéza, že po odstranění XSLT programů s těmito vzory je na zbylém fragmentu XSLT statická analýza algoritmicky řešitelná. Jak ukazují výzkumy hranic polynomiální statické analýzy ([4]), nedá se očekávat lepší než exponenciální složitost, současný boom model-checkingu však ukazuje, že podobné metody mají šanci na praktické uplatnění.

Reference

1. Dong C. and Bailey J., Static Analysis of XSLT Programs. In Proceedings of the Fifteenth Conference on Australasian Database - Volume 27 (Dunedin, New Zealand). K. Schewe and H. Williams (eds.) ACM International Conference Proceeding Series, vol. 52, 2004
2. Gottlob G., Koch C., and Pichler R., Efficient Algorithms for Processing XPath Queries. In Proc. of the 28th International Conference on Very Large Data Bases (VLDB 2002)
3. Hidders J., Michiels P., Paredaens J., and Vercammen R. LiXQuery: a Formal Foundation for XQuery Research. SIGMOD Rec. 34, 4 Dec. 2005, 21–26
4. Martens W. and Neven F., Frontiers of Tractability for Typechecking Simple XML Transformations. In Proceedings of the Twenty-Third ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, Paris, France, June 14–16, 2004, PODS '04, ACM Press, New York, 2004
5. Mads Kristian Østerby Olesen, Static Validation of XSLT Transformations. Master's Thesis, University of Aarhus, 2004
6. Tozawa A., Towards Static Type Checking for XSLT. In Proceedings of the 2001 ACM Symposium on Document Engineering, Atlanta, Georgia, USA, November 09–10, 2001, DocEng '01. ACM Press, New York

Kombinácia algoritmu GHSOM a Sammonovho mapovania pre vizualizáciu množiny textových dokumentov

Peter Butka and Milan Števkov

Katedra kybernetiky a umelej inteligencie, Fakulta elektrotechniky a informatiky,
Technická univerzita v Košiciach, Letná 9, 04200 Košice, Slovensko
peter.butka@tuke.sk, mstevkov@gmail.com

Abstrakt *V tejto práci je prezentovaný prístup k vizualizácii množín textových dokumentov kombináciou hierarchických samoorganizujúcich sa máp (SOM-y, konkrétne algoritmus GHSOM) a Sammonovho mapovania. Algoritmy na báze SOM predstavujú robustnú zhlukovaciu metódu, vhodnú pre vizualizáciu väčšieho počtu dokumentov. Sammonovo mapovanie je nelineárna projekčná metóda, ktorá je vhodná najmä pre menšie množiny objektov. V rámci našej práce bola testovaná kombinácia týchto dvoch prístupov tak, že sa najprv množina textových dokumentov rozdelí použitím hierarchického SOM-u na podmnožiny súvisiacich dokumentov, potom pre zhluky s malým počtom dokumentov bude vytvorená Sammonova mapa. Pre popis zhlukov bola použitá metóda pre extrakciu charakteristických termov (slov) na základe informačného zisku. Pri implementácii bola pre potreby spracovania textových dokumentov využitá existujúca knižnica JBOWL, testované množiny dokumentov boli v anglickom jazyku.*

1 Úvod

V súčasnosti existuje mnoho systémov pre zhlukovanie a vizualizáciu textových dokumentov a mnohé z nich fungujú na základe princípu samoorganizujúcich sa máp (SOM, [1]). Základnou vlastnosťou modelov na báze architektúry SOM je ich schopnosť zachovávať topológiu mapovania vstupného priestoru vo výstupnom priestore (mape). Často je problémom klasickej architektúry SOM (Kohonenova mapa) jej pevná štruktúra (po začatí učenia). Model blízky klasickému modelu SOM, Growing Grid (popísaný Fritzkem – [2]), umožňuje zväčšovanie pôvodnej SOM dynamicky počas procesu učenia (pridávaním riadkov resp. stĺpcov nových neurónov). Druhá adaptívna metóda je založená na použití hierarchickej štruktúry nezávislých máp, kde pre každý prvok mapy (neurón) je na novej úrovni pridaná nová mapa, ktorá potom podrobnejšie rozdeľuje príklady nadradeného (rodičovského) neurónu. Táto architektúra sa nazýva Hierarchical Feature Map (HFM, [3]). Algoritmus GHSOM (Growing Hierarchial SOM, [4]) kombinuje postupy týchto dvoch modelov neurónových sietí. Znamená to, že každá vrstva hierarchickej štruktúry pozostáva z množiny nezávislých máp, ktoré adaptujú svoju veľkost s ohľadom na požiadavky vstupných dát (príkladov prislúchajúcich danej mape).

Sammonovo mapovanie [5] je nelineárna projekčná metóda, ktorá je vhodná najmä pre menšie množiny objektov. Snaží sa čo najvernejšie modelovať vzdialenosti objektov vo vysoko rozmernom priestore vzdialenosťami v projektovanom, zvyčajne dvojrozmernom priestore. Výhodou je vizualizačná stránka projekcie, pokiaľ objektov nie je príliš veľa, čím je modelovanie vzájomných vzdialenosti (najmä pri veľkom počte atribútov) značne problematické. Pri vizualizácii veľkého počtu dokumentov je Sammonovo mapovanie nepohodlné aj z hľadiska časovej náročnosti.

Táto práca sa venuje vizualizácii množiny textových dokumentov kombináciou hierarchických samoorganizujúcich sa máp (GHSOM) a Sammonovho mapovania. Kým samoorganizujúce sa mapy predstavujú robustnú zhlukovaciu metódu, vhodnú pre zhlukovanie väčšieho počtu dokumentov, Sammonovo mapovanie predstavuje metódu umožňujúcu zobraziť objekty ako samostatné body na ploche, pričom sa snaží (rovnako ako SOM-y) zachovať do určitej miery topológiu rozdelenia dokumentov na základe podobnosti. V rámci tejto práce sa testuje kombinácia týchto dvoch prístupov. Množina textových dokumentov je najprv rozdelená použitím hierarchického SOM-u na podmnožiny súvisiacich dokumentov. Potom je pre zhluky s malým počtom dokumentov vytvorená Sammonova mapa. Pre popis zhlukov bola použitá metóda pre extrakciu charakteristických termov na základe informačného zisku. Pri implementácii bola pre potreby spracovania textových dokumentov použitá existujúca knižnica JBOWL [6], testované množiny dokumentov boli v anglickom jazyku.

V nasledujúcej kapitole budú popísané metódy, ktoré boli použité v tejto práci. V ďalších kapitolách bude stručne popísaná ich spoločná integrácia a implementácia do jednotného systému, ako aj stručný popis experimentov.

2 Použité metódy a postupy

2.1 Predspracovanie textových dokumentov

Predspracovanie textových dokumentov v našom prípade pozostávalo z nasledujúcich krokov:

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

V prvom kroku (tokenizácia) je vstupný text transformovaný na tokeny. Tokenizácia je nevyhnutnou súčasťou predspracovania, ostatné kroky sú voliteľné (používajú sa kvôli redukcii priestoru termov). Takto získané tokeny predstavujú základný slovník spracovávanej množiny dokumentov. V nasledujúcom kroku sú z tejto množiny odstránené neplnovýznamové slová („stopwords“ - spojky, častice, zámená, ...) porovnaním so zoznamom slov v tzv. stop-liste. V ďalšom kroku sú zostávajúce tokeny transformované na ich základné tvary - stemy (ako napríklad „win“, „wins“ sú transformované na jeden koreň - „win“). Dosiahneme tak opätovné zmenšenie slovníka (pre rôzne morfologické tvary zostane len koreň slova - stem). Posledný krok na základe frekvencie výskytu slov takto upraveného slovníka odstráni tie slová, ktoré sa vyskytovali v texte príliš často (nie sú zaujímavé pre rozlíšenie dokumentov), resp. tie slová, ktoré sa v texte vyskytli v príliš malom počte dokumentov. Preto sa ponechajú len tie tokeny - termy, ktorých frekvencia výskytu je niekde medzi dvojicou nastaviteľných prahov pre minimálnu a maximálnu povolenú hodnotu frekvencie. Skúmaná kolekcia dokumentov je potom reprezentovaná pomocou dokument-term matice. Pre váhovanie termov sme použili *tfidf schému*, ktorá priradzuje každej dvojici term-dokument reálne číslo (príslušnú váhu) nasledovne:

$$w_{ij} = tfidf_{ij} = tf_{ij} \cdot \log\left(\frac{ndocs}{df_j}\right), \qquad (1)$$

kde tf_{ij} je frekvencia výskytu j-tého termu v i-tom dokumente, $ndocs$ je počet dokumentov v kolekcii a df_j je počet dokumentov obsahujúcich daný j-tý term (dokumentová frekvencia). Výsledkom celého procesu teda je dokument-term matica váh w_{ij} vybraných termov.

2.2 Algoritmus GHSOM

Na začiatku procesu učenia je potrebné inicializovať štartovaciu mapu najvyššej úrovne. Mapa, ktorá pozostáva z $m \times n$ neurónov (centier zhlukov), je inicializovaná náhodne spomedzi vstupných vektorov.

Po inicializácii sú náhodne vyberané vstupné vektory prezentované na vstup neurónovej siete. Pre každý neurón sa vypočíta aktivácia vzhľadom k danému dokumentu. Neurón s najvyššou aktiváciou (ak použijeme kosínusovú metriku, ako tomu bolo v našom prípade) sa označí ako víťaz. Tento vektor, ako aj vektory v okolí víťazného neurónu, sa adaptujú použitím nasledujúcich dvoch formúl:

$$INF = 1 - \frac{dist}{NGH + 0.5} \qquad (2)$$

$$c[i] = c[i] + LR \cdot INF \cdot (v[i] - c[i]), \qquad (3)$$

kde $dist$ je vzdialenosť na mape medzi víťazným neurónom a upravovaným neurónom c, NGH je parameter susednosti. Potom INF je vypočítaná miera vplyvu (influence) daného vstupu v na upravované váhy, LR je parameter učenia.

Počiatočný iteračný proces (jedna iterácia = každý dokument kolekcie je vybraný ako vstupný vektor) končí dosiahnutím nastaveného počtu iterácií. Potom sa vypočíta variabilita každého neurónu mapy (variabilita dokumentov priradených danému neurónu), ako aj variabilita celej mapy. Variabilita (stredná kvadratická odchýlka) neurónu sa vypočíta ako priemerná hodnota vzdialeností centra daného neurónu od vstupných vektorov priradených tomuto neurónu. Potom variabilita mapy je priemerná variabilita neurónov danej mapy.

Po týchto krokoch je testovaná stredná kvadratická odchýlka (MQE) danej mapy. Ak podmienka

$$MQE \geq \tau_1 \cdot mqe_0 \qquad (4)$$

je splnená (reprezentuje vzťah variability danej mapy MQE k variabilite celej množiny vstupných vektorov mqe_0), potom je potrebné pridať neuróny. Preto zvolená konštanta τ_1 reprezentuje prah pre vkladanie neurónov. Pred pridaním nájdeme najvariabilnejší neurón (tzv. chybový neurón) a jeho najvzdialenejšieho suseda (podľa metriky vo vstupnom priestore). Potom vložíme celý blok neurónov pridaním riadku resp. stĺpca neurónov medzi túto dvojicu. Ak variabilita neurónov mapy poklesne dostatočne, vkladanie neurónov sa skončí. Inicializácia váh nových neurónov sa získava spriemernením váh okolitých neurónov. Po každom vložení neurónov je potrebné mapu opäť preučiť.

Po skončení fázy rozširovania mapy sa testuje každý neurón mapy pre možnosť expanzie vo forme novej podmapy. Preto ak podmienka

$$mqe \geq \tau_2 \cdot mqe_0 \qquad (5)$$

je splnená (reprezentuje vzťah variability konkrétneho neurónu mqe k mqe_0), expanzia neurónu na novú úroveň hierarchie sa uskutoční. Parameter τ_2 predstavuje prah pre expanziu neurónu. Novovzniknutá mapa má veľkosť 2×2. Váhové vektory sú inicializované na základe neurónov v okolí expandovaného neurónu, a to spriemernením váh daného neurónu a troch jeho susedov v závislosti na pozícii nového neurónu v mape.

V našom prípade sme túto podmienku doplnili o minimálny potrebný počet dokumentov v zhluku, pre ktorý sa podmapa môže vytvárať (ďalší parameter v rámci implementácie algoritmu GHSOM).

Algoritmus teda postupuje zhora-nadol a rozkladá množinu vstupných vektorov na stále menšie skupiny. Dostávame tak celú hierarchiu máp. Koniec algoritmu je vtedy, ak už neexistuje neurón (na žiadnej mape), ktorý by bolo potrebné expandovať na novú úroveň.

2.3 Sammonovo mapovanie

Sammonovo mapovanie [5] je metóda nelineárnej projekcie vysoko-rozmerných dát do priestoru nižšej dimenzie, zvyčajne do dvojrozmerného priestoru. Cieľom Sammonovho mapovania je transformácia, ktorá minimalizuje definovanú chybovú funkciu. Sammonovo mapovanie sa pokúša minimalizovať túto chybu tým, že nastaví umiestnenie bodov v nižšej dimenzii tak, aby vzdialenosť medzi bodmi bola čo najbližšia k vzdialenosti medzi príslušnými bodmi vo vyššej dimenzii. Jednoduchým prístupom k minimalizácii môže byť napríklad nejaká gradientová iteračná metóda.

Predpokladajme, že máme množinu n objektov reprezentovaných bodmi v m - rozmernom (vysoko rozmernom) priestore. Cieľom Sammonovho mapovania je nájsť n bodov v d - dimenzionálnom priestore (kde $d < n$) takým spôsobom, aby korešpondujúce transformované vzdialenosti aproximovali originálne vzdialenosti čo najlepšie.

Nech pre ľubovoľné dva body vstupnej množiny s indexami i a j $(i, j = 1, \ldots, n)$ je d_{ij} ich vzdialenosť v originálnom m-rozmernom priestore a d^*_{ij} je ich vzdialenosť v projektovanom d-rozmernom priestore. Potom kritérium na určenie kvality mapovania v danom kroku je navrhnuté na základe nasledovnej chybovej funkcie, ktorá určuje rozdiel medzi súčasným zoskupením n bodov v d - dimenzionálnom priestore a usporiadaním n bodov v originálnom m - dimenzionálnom priestore (označuje sa aj ako „stress function“):

$$E = \frac{1}{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} d^*_{ij}} \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \frac{(d^*_{ij} - d_{ij})^2}{d^*_{ij}} \qquad (6)$$

Obor hodnôt E je interval $(0, 1)$, kde 0 indikuje bezstratové mapovanie. Ďalej sa budeme zaoberať len zobrazením v dvojrozmernom priestore ($d = 2$).

Problém nájdenia správneho zoskupenia v nízkorozmernom priestore je optimalizačný problém, zaujímame sa o získanie takého zoskupenia aby chybová funkcia E dosiahla minimum. Tento optimalizačný problém je zložitý, pretože priestor parametra je vysoko-rozmerný. Chybová funkcia je optimálna keď originálne vzdialenosti d^*_{ij} sú ekvivalentné so vzdialenosťami d_{ij}, ktoré sú projektované. Väčšinou však nájdené vzdialenosti budú skresľovať reprezentácie vzťahov medzi dátami. Čím väčšia bude hodnota funkcie E, tým väčšie bude skreslenie.

Na nájdenie projekčnej mapy začíname od inicializačného zoskupenia bodov, potom vypočítame chybovú funkciu E pomocou danej rovnice. Nasledovne je zoskupenie upravované tak, aby sme dosiahli čo najlepšie výsledky. Tento proces je opakovaný, pokiaľ nie je nájdená mapa korešpondujúca s lokálnym minimom chybovej funkcie E.

Nech $E(m)$ je chybová funkcia prislúchajúca m-tej iterácii, v ktorej

$$E(m) = \frac{1}{c}\sum_{i<j}^{n} \frac{\left[d^*_{ij} - d_{ij}(m)\right]^2}{d^*_{ij}}, \qquad (7)$$

kde

$$c = \sum_{i<j}^{n} d^*_{ij} \qquad (8)$$

a

$$d_{ij}(m) = \sqrt{\sum_{k=1}^{d} \big(y_{ik}(m) - y_{jk}(m)\big)^2}, \qquad (9)$$

kde d_{ij} je výpočet vzdialeností medzi vektormi matice pomocou euklidovskej vzdialenosti.

Nové súradnice bodov v iterácii $m + 1$ dostaneme pomocou vzťahu:

$$y_{pq}(m+1) = y_{pq}(m) - (MF) \cdot \Delta_{pq}(m), \qquad (10)$$

kde

$$\Delta_{pq}(m) = \frac{\partial E(m)}{\partial y_{pq}(m)} \Big/ \left|\frac{\partial^2 E(m)}{\partial y_{pq}(m)^2}\right| \qquad (11)$$

a pre korigujúci faktor MF (tzv. „magic factor“) sa zvyčajne používa hodnota z intervalu (0.3,0.4).

Parciálne derivácie vypočítame podľa vzťahov:

$$\frac{\partial E(m)}{\partial y_{pq}} = -\frac{2}{c}\sum_{\substack{j=1\\ j\neq p}}^{N} \frac{\big(d^*_{pj} - d_{pj}(m)\big)}{d_{pj}(m)\, d^*_{pj}} \cdot \big(y_{pq}(m) - y_{jq}(m)\big) \qquad (12)$$

a

$$\frac{\partial^2 E(m)}{\partial y^2_{pq}} = -\frac{2}{c}\sum_{\substack{j=1\\ j\neq p}}^{N} \frac{1}{d^*_{pj}(m)\, d_{pj}} \cdot \left[\big(d^*_{pj} - d_{pj}(m)\big) - \left(\frac{\big(y_{pq}(m) - y_{jq}(m)\big)^2}{d_{pj}(m)}\right)\left(1 + \frac{d^*_{pj} - d_{pj}(m)}{d_{pj}(m)}\right)\right] \qquad (13)$$

Podrobné odvodenia vzťahov parciálnych derivácií sa nachádzajú v [7].

2.4 Extrakcia charakteristických termov

Pre popis jednotlivých zhlukov dokumentov na mape je možné použiť rôzne metódy pre extrakciu charakteristických termov (kľúčových slov danej podmnožiny dokumentov). Jednou z možností je využiť metódu pracujúcu na princípe ohodnocovania dôležitosti termov na základe informačného zisku. Tento sa často používa ako kritérium určenia správnosti kľúčového slova. Určuje množstvo obsiahnutej informácie v terme vzhľadom na predikciu triedy zistením prítomnosti alebo absencie termu v dokumente.

Majme m kategórií v cieľovom priestore, označme i-tú kategóriu c_i. Informačný zisk termu t je definovaný ako:

$$\begin{aligned} G(t) = &- \sum_{i=1}^{m} \Pr(c_i) \log \Pr(c_i) + \\ &\Pr(t) \sum_{i=1}^{m} \Pr(c_i|t) \log \Pr(c_i|t) + \\ &\Pr(\bar{t}) \sum_{i=1}^{m} \Pr(c_i|\bar{t}) \log \Pr(c_i|\bar{t}), \end{aligned} \tag{14}$$

kde $\Pr(c) = \frac{N_c}{N}$ vyjadruje pravdepodobnosť výskytu kategórie c, $\Pr(c|t) = \frac{N_{tc}}{N_t}$ vyjadruje pravdepodobnosť výskytu kategórie podmienenú výskytom termu t a nakoniec $\Pr(c|\bar{t}) = \frac{N_{\bar{t}c}}{N_{\bar{t}}}$ vyjadruje pravdepodobnosť výskytu kategórie c podmienenú absenciou termu t. Číslo m udáva počet kategórií a číslo N zase počet dokumentov spĺňajúcich príslušnú podmienku.

Princíp aplikácie tejto metódy pre extrakciu termov zhluku spočíva v tom, že sa pre danú trénovaciu množinu dokumentov vypočíta informačný zisk pre každý jeden term voči zhlukom (ktoré teraz považujeme za priradenú triedu – kategóriu) a odstránia sa všetky tie termy z príznakového priestoru, ktorých informačný zisk je menší ako vopred zadefinovaný prah, prípadne sa zoberie zo zoradeného zoznamu len určitý počet termov pre každú kategóriu (zhluk).

3 Popis kombinácie prístupov

Kombinácia algoritmu GHSOM a Sammonovho mapovania bola implementovaná v prostredí knižnice JBOWL [6]. Postup spracovania je možné zhrnúť nasledovne:

1. Predspracovanie
2. Budovanie kombinovaného modelu
3. Vizualizácia modelu

Pri predspracovaní sa postupuje podľa krokov popísaných v kapitole 2.1, pričom pre stemming bol použitý jednoduchý Porterov stemmer [8]. Ide síce o jednoduchý algoritmus založený na orezávaní prípon, pre anglický text však dosahuje pomerne dobré výsledky.

Budovanie modelu realizuje integráciu Sammonovho mapovania do modelu GHSOM v rámci JBOWL. V knižnici JBOWL bol implementovaný algoritmus GSOM (jedna rozširujúca sa vrstva neurónov, GrowingGrid) a GHSOM. Algoritmus GHSOM rozširuje základnú mapu dokumentov a vytvára nové podmapy (GSOM) expanziou neurónov (uzlov).Na základe vzťahov slov v dokumente a na základe vzťahov medzi dokumentmi sa vytvorí určitá hierarchia týchto dokumentov. Na neuróny (uzly), ktoré už neexpandujú sa aplikuje Sammonov algoritmus (ďalej SammonAlgorithm). Každý uzol je reprezentovaný maticou vektorov (dokumentov) v mnohorozmernom priestore. V uzloch sa spúšťa tzv. LeafAlgorithm, ktorý zabezpečuje spustenie iného algoritmu v momente ukončenia expanzie tvorbou novej mapy pomocou GSOM. Neuróny, ktoré sa ďalej neexpandujú ošetruje ukončovacia podmienka (stop condition). Ak je táto podmienka splnená, spustí sa SammonAlgorithm.

SammonAlgorithm taktiež vychádza z veľmi podobného centroidného modelu, pričom tu sa to myslí tak, že centroidy sú súradnice odpovedajúcich dokumentov v dvojrozmernom priestore. Na začiatku algoritmus načíta rozmer matice, ktorou je zhluk reprezentovaný. Následne vytvorí vektory centroidov (dvojrozmerných reprezentácií dokumentov), ktoré náhodne nainicializuje. Počet týchto vektorov je ekvivalentný k rozmeru vstupnej matice. V ďalšom kroku vypočíta maticu vzdialeností vstupných vektorov dokumentov (distanceMatrix) a maticu vzdialeností náhodne nainicializovaných centroidov (distanceSammon) pomocou euklidovskej vzdialenosti. Nasleduje iteračný proces zmenšujúci postupne chybu metódou popísanou v kapitole 2.3, pričom počiatočné súradnice sú nainicializované náhodne.

Výstupom kroku budovania modelu je vytvorený hierarchický serializovaný model, ktorý obsahuje za sebou usporiadané modely máp vytvorené algoritmami GHSOM a Sammonovho mapovania. Tento model zároveň tvorí vstup do vizualizačného procesu.

V prípade, že by sme použili iba algoritmus GHSOM, výstupom celého procesu spracovania by bola množina HTML stránok, kde každá stránka zobrazuje jednu mapu hierarchie. Políčka mapy odpovedajú neurónom - centrám zhlukom. Každý zhluk je popísaný polohou na mape, počtom priradených vektorov a množinou charakteristických termov. Tieto sa získavajú na základe metódy popísanej v predchádzajúcej kapitole.

Doplňujúcim prvkom v našom prípade je existencia ďalších HTML stránok, ktoré zobrazujú Sammonove mapy zhlukov, ktoré sú na konci hierarchií, t.j. niečo ako listové uzly hierarchie (nemajú podmapu). Tu sa v našom prípade v HTML kóde dopĺňa linka na stránku zobrazujúcu Sammonovu mapu dokumentov tohto

listového zhluku. Úlohou vizualizačného kroku je zobraziť výslednú hierarchiu máp a extrahovať termy popisujúce jednotlivé zhluky, rovnako je potrebné vizualizovať mapy vytvorené pomocou SammonAlgorithm nachádzajúce sa vo vytvorenom hierarchickom modeli. Výstupom bloku (aj celého systému) je množina HTML stránok. Každá z nich je označená podľa toho, ktorú vrstvu výslednej hierarchie zobrazuje. Na začiatku je uvedený odkaz na rodičovskú („parent“) mapu, pričom najvrchnejšia mapa nemá rodiča. Mapa je potom tvorená množinou políčok reprezentujúcich zhluky. Každý obsahuje názov, údaj o počte pokrytých vektorov a priradené charakteristické termy (kľúčové slová). Na konci je uvedený odkaz na stránku s expandovanou mapou (ak existuje), alebo na stránku vizualizovaných dokumentov v dvojrozmernom priestore – ak bola splnená ukončovacia podmienka („stop condition“) a bol tak vytvorený model podľa Sammonovho algoritmu. Na stránke Sammonovho modelu je taktiež uvedený odkaz na rodičovskú mapu spolu s mapkou vizualizovaných dokumentov. Tie sú zobrazené na základe súradníc, ktoré sme dostali z výstupných hodnôt vektorov centroidov. Hodnoty boli normalizované na mierku (-1,1). Vytvorená množina stránok vytvára ľahko prehľadávateľnú štruktúru.

Cluster : 1 **Vectors** : 38 germani german germany' bonn ludwig coal miner (model-1.html)	**Cluster** : 2 **Vectors** : 40 french franc de europ pari charl gaull (model-2.html)	**Cluster** : 3 **Vectors** : 57 soviet russian moscow russia khrushchev nikita peke (model-3.html)
Cluster : 5 **Vectors** : 43 soviet russia reason moment court spend david (model-5.html)	**Cluster** : 6 **Vectors** : 20 station tourist hungari knock hungarian budapest cardin (model-6.html)	**Cluster** : 7 **Vectors** : 13 chines china peke china' shortag cambodia soft (model-7.html)

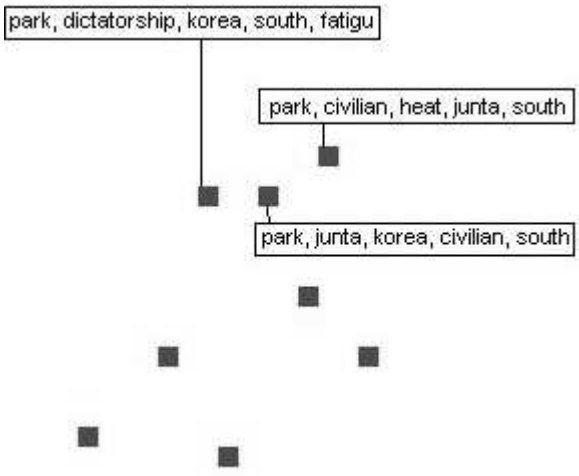
Obrázok 1. Časť mapy GHSOM-u pre články kolekcie Times60. Z popisov zhlukov môžeme vyčítať najčastejšie sa vyskytujúce témy v dokumentoch zobrazených zhlukov, ako napríklad témy týkajúce sa francúzskej histórie, vietnamskej vojny, Indonézie a okolitých štátov, Indie, Nemecka a Sovietskeho zväzu a podobne.

4 Príklad experimentu

Pri experimente bola použitá množina článkov denníka Times. Ide o 420 dokumentov zo šesťdesiatych rokov minulého storočia s rôznou tematikou. Obsahujú články o medzinárodných vzťahoch, ekonomickej a politickej situácii a histórii rôznych krajín a regiónov, obsahujú aj články o vietnamskej vojne a povojnové články po druhej svetovej vojne. Pre túto množinu budem používať označenie Times60. Niekoľko testov síce prebehlo aj na množine Reuters, avšak vzhľadom k tomu, že nemáme žiadne všeobecné kvantitatívne vyhodnotenie, nebudeme ich uvádzať. Podrobnejšie informácie k experimentom a implementácii je možné nájsť v [9].

Pri predspracovaní dát sme použili už uvedený postup,počet termov vstupujúcich do algoritmu je ovplyvnený dvomi parametrami minimálnej a maximálnej hodnoty frekvencie výskytu termu v dokumentoch, pre uvedené príklady experimentov bol počet termov 1925 (min.frek. 5, max.frek.100). Príklad časti mapy vrchnej vrstvy GHSOM-u je možné nájsť na obrázku 1. Príklad Sammonovej mapy je zase možné vidieť na obrázku 2.

Algoritmus GHSOM rozdelil množinu dokumentov na zhluky navzájom súvisiacich dokumentov. Inými slovami vytvoril kategórie dokumentov. O jednotlivých zhlukoch bolo známych niekoľko faktov, ako napríklad počet dokumentov, ktoré daný zhluk obsahoval, ako aj kľúčové slová prislúchajúce danému zhluku.



Obrázok 2. Príklad časti Sammonovej mapy. Pri prechode nad bodom prislúchajúceho dokumentu sa objaví ako alternatívny výpis zoznam termov, tu boli pre tri dokumenty termy vypísané a vlepené do obrázku. V dokumentoch v tejto časti sa najčastejšie vyskytujú termy ako korea, park, juta, dokumenty v ostatných častiach mapy obsahovali iné charakteristické termy.

Implementáciou Sammonovho mapovania do tohto algoritmu vznikol systém umožňujúci vizualizáciu dokumentov v 2D priestore v podobe máp. Zobrazovaný zhluk dokumentov bol popísaný kľúčovými slovami vo všeobecnosti pre celý zhluk. V mapách vytvorených Sammonovým algoritmom boli extrahované kľúčové slová pre každý dokument a tým bol umožnený podrobnejší náhľad na vizualizované dokumenty.

Sammonovo mapovanie zobrazovalo dokumenty pomerne presne. Výber kľúčových slov z dokumentov okrem samotného popisu poslúžil aj ako pomôcka na overenie správnosti projektovania. Zhluky obsahovali dokumenty s podobnými témami, dokumenty nachádzajúce sa na mape bližšie k sebe mali vo väčšine prípadov podobnejšie slová ako vzdialenejšie dokumenty.

5 Záver

V tejto práci bol prezentovaný jednoduchý prístup ku kombinácii dvoch algoritmov pre vizualizáciu množín textových dokumentov - algoritmu GHSOM a Sammonovho mapovania. Kým jedna je lepšia v robustnom rozdelení korpusu dokumentov na menšie podmnožiny príbuzných dokumentov, druhá je vhodnejšia pre rozdelenie menšej množiny dokumentov individuálne v dvojrozmernom priestore. Výhody oboch sa podarilo prepojiť v momente kedy GHSOM ukončil činnosť tvorby podmáp v danom neuróne. Potom dokumenty daného zhluku boli zobrazené pomocou Sammonovho mapovania, aby boli individuálne odlíšiteľné.

V budúcnosti by sa žiadalo najmä dôkladnejšie otestovanie navrhovanej metódy na väčších vstupných množinách dokumentov, najmä kvantitatívne vyhodnotenie výsledkov, ktoré práci určite chýba. Rovnako by bolo zaujímavé zlepšenie formy vizualizácie, prípadne zapojenie užívateľa do procesu, kde by užívateľ mohol na základe špecifického dotazu dostať ako odpoveď napríklad časť hierarchie vygenerovaných stránok a väzieb medzi nimi.

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

Referencie

1. Kohonen T., et al., Self Organization of a Massive Document Collection. In IEEE Transactions on Neural Networks 11(3), 2000, 574–585
2. Fritzke F., Growing Grid - a Self-Organizing Network with Constant Neighbourhood and Adaptive Strength. In Neural Processing Letters 2(5), 1995
3. Merkl D., Explorations of Text Collections with Hierarchical Feature Maps. In Proceedings of International ACM SIGIR Conference in Information Retrieval, Philadelphia, 1997
4. Dittenbach M., Merkl D., Rauber A.: Using Growing Hierarchical Self-Organizing Maps for Document Classification. In Proceedings of ESANN, Bruges, 2000, 7–12
5. Sammon J.R., A Nonlinear Mapping for Data Structure Analysis. In IEEE Transactions on Computers, C-18, 1969, 401–409
6. Bednar P., Butka P., Paralic J., Java Library for Support of Text Mining and Retrieval. In Proceedings of Znalosti 2005, Stará Lesná, 2005, 162–169
7. Tsai C.S., Visual Display Techniques, Dissertation Thesis. Department of Computer Science and Information Engineering, National Chi-Nan University, 2003
8. Porter M.F., An Algorithm for Suffix Stripping. In V Program, 14(3), 1980, 130–137
9. Števkov M., Vizualizácia množiny textových dokumentov, Diplomová práca, Katedra kybernetiky a umelej inteligencie, TU Košice, 2006

Použití relačních databází pro vyhodnocení SPARQL dotazů*

Jiří Dokulil

Katedra softwarového inženýrství, MFF UK Praha
dokulil@gmail.com

Abstrakt *Uložení RDF dat do relační databáze lze provést velmi přímočaře. Pokud však databázi obsahuje velmi složitá a rozsáhlá data, výsledky tohoto přístupu se dramaticky zhorší. Tento článek se pokouší analyzovat, z čeho tyto problémy pramení. Přináší i dva návrhy, jak těmto problémům čelit, včetně testů implementace jednoho z návrhů.*

1 Úvod

Jazyk RDF [4] je základním stavebním kamenem Sémantického webu [2]. Je to nástroj pro tvorbu popisu zdrojů, zvláště pak zdrojů na Internetu, jako například nadpis, autor a datum poslední změny webové stránky, autorská a licenční práva k dokumentu vystavenému na Internetu nebo třeba informace o časovém rozvrhu dostupnosti zdroje.

Zobecněním konceptu zdroje na Internetu je možné použít RDF pro popis objektů, které lze na Internetu pouze identifikovat, nikoliv však získat. Příkladem jsou třeba podrobnosti (velikost, cena, barva, ...) o zboží v elektronickém obchodě.

Jazyk RDF popisuje zdroje pomocí trojic, které je možné interpretovat jako orientovaný ohodnocený graf.

Samotné RDF je však pouze nástroj pro uložení dat. Definuje formát a sémantiku, ne však již způsob, jak se nad těmito daty dotazovat. V současné době existuje několik jazyků, které byly buď přímo vytvořeny pro dotazování nad RDF daty, nebo je možné je za tímto účelem upravit. Například RDQL [9], SeRQL [3], RQL [1] nebo SPARQL [8]. Velkou část posledně jmenovaného jazyka jsme implementovali s využitím relační databáze [7].

V tomto článku představíme naši implementaci dotazovacího jazyka SPARQL, výsledky měření rychlosti vyhodnocení dotazů spolu s problémy, které se při testech ukázaly. Zároveň navrhneme dva způsoby, jak tyto problémy řešit.

* Tato práce byla částečně podporována projektem 1ET100300419 Programu Informační společnost Tématického programu II Národního programu výzkumu České republiky.

1.1 Experimantální implementace jazyka SPARQL

Rozhodli jsme se ukládat RDF data v relační databázi Oracle Database 10g. Protože RDF data jsou trojice, je tabulka se třemi sloupci přirozený způsob, jak tato data uložit. Různé technické detaily si vynutily o něco složitější reprezentaci RDF dat, ale tato základní myšlenka zůstala.

Díky tomu je možné vyhodnocovat SPARQL dotazy jejich překladem na SQL dotazy. Systém z jednoho vstupního SPARQL dotazu vytvoří jeden SQL dotaz, který vrací stejný výsledek, jako by vracel původní SPARQL dotaz. Přeložené dotazy obsahují převážně spojení a jednoduché selekce, ale fakt, že SPARQL dovoluje i aritmetická porovnání, si vynutil i podporu několika PL/SQL funkcemi. Podrobně je předklad dotazů popsán v [7].

Zásadní důsledek tohoto přístupu je, že optimalizace vyhodnocení dotazu je ponechána na optimalizátoru relační databáze.

Tato metoda funguje dobře nad malými nebo jednoduchými RDF daty. Pokud jsou však data velká a složitá, vyhodnocení se dostane do problémů.

Konkrétní testovací data, výsledky a problémy ukážeme v následujících kapitolách. Napřed však trochu podrobněji popíšeme schéma databáze, ve které RDF data ukládáme.

1.2 Databázové schéma

Protože SPARQL dotazy mohou obsahovat proměnné i na místě predikátu, nebylo možné vytvořit pro každý predikát samostatnou tabulku. Proto jsou veškeré RDF trojice uloženy v jedné tabulce nazvané TRIPLES.

Tato tabulka se velmi často účastní spojení, proto neobsahuje přímo hodnoty subjektu, predikátu a objektu trojice, ale pouze zástupné identifikátory, které jsou mnohonásobně menší.

Identifikátory slouží jako cizí klíče do tabulky LITERALS, která uchovává jejich mapování na konkrétní hodnoty. Toto mapování musí být jednoznačné, tedy jedna textová hodnota odpovídá nejvýše jednomu identifikátoru. To výrazně zjednoduší a zrychlí vyhodnocení dotazů.

2 Data

Přesný postup získání dat popisuje [6]. Získaná data jsou složitá, nepravidelná a rozsáhlá.

Pokud bychom je uložili v relační databázi, pak by obsahovala 226 tabulek s celkem 1898 sloupci (bez započítání sloupců s primárními klíči). Jak RDF, tak systém, ze kterého jsme data získali, data neukládají jako n-tice, ale jako trojice, kde na prvním místě je identifikace objektu, který popisujeme, na druhém popisovaná vlastnost a na třetím místě hodnota vlastnosti. V relační databázi tomu odpovídá primární klíč, název sloupce a hodnota jedné buňky.

Získaná RDF data obsahují 226 tříd a 1898 predikátů. Schéma dat je tedy relativně složité. Navíc data pocházejí z různorodých aplikací nasazených v reálném provozu. To způsobilo, že data nejsou pravidelná, protože každý ze systémů bych schopen poskytnout pouze některé z hodnot definovaných v globálním schématu. Navíc data nebyla příliš čistá.

Celková velikost dat je 26 814 915 trojic. Hlavní částí dat jsou data o lidech, například jména a příjmení. Na tato data jsme se zaměřili v testech a zbytek dat používali hlavně jako "šum pozadí". Vliv těchto nevyužitých dat na vyhodnocení dotazu byl však zásadní, protože ovlivňoval četnosti a selektivity.

V datech platí, že počet trojic s jedním konkrétním predikátem tvoří pouze zlomek z celkového počtu trojic. Na druhou stranu, tento zlomek často znamená stovky tisíc trojic.

Počet trojic se shodným subjektem je malý (typicky 5-20, maximum je 58), počet trojic se shodným objektem se pohybuje od jedné trojice do dvou milionů trojic. Přibližně platí, že pro náhodně vybranou trojici je stejná pravděpodobnost, že existuje jedna trojice se stejným objektem nebo dva miliony trojic. Pro hodnoty mezi těmito okraji je pravděpodobnost menší, ale řádově podobná. Není tedy možné rozumně odhadnout počet trojic se zadaným objektem bez práce s databází.

3 Dotazy a rychlost jejich vyhodnocení

V této kapitole ukážeme několik dotazů spolu s rychlostí jejich vyhodnocení. Pro každý dotaz jsme otestovali čtyři metody uložení a indexace tabulky TRIPLES. Naším cílem bylo nalezení nejvhodnějšího způsobu indexace pro výběr trojic se zadaným predikátem.

V základní verzi byla tabulka uložena obvyklým způsobem a nad všemi sloupci byly indexy (B-stromy).

Ve verzi *B-strom* byla tabulka TRIPLES uložena jako B-strom, kde indexovaným sloupcem byl sloupec s predikáty. Díky tomu bylo možné výrazně zrychlit operaci prohledání všech trojic se zvoleným predikátem, protože tyto trojice byly ve výrazně menším počtů stránek na disku než v základní verzi. Nad subjektem a objektem byl vystavěn běžný index.

Paralelní verze byla totožná s předchozí, pouze bylo u tabulky triples navíc povoleno paralelní zpracování.

Poslední verze (*bitmapa*) byla stejná jako základní, avšak s bitmapovým indexem nad predikáty.

3.1 Testovací prostředí

Databáze běžela na stroji se dvěma procesory XEON 3.06GHz, 6GB RAM a SCSI diskovým polem se čtrnácti 144GB 10k RPM disky pro uložení dat a indexů. Pro RDF databázi bylo vyhrazeno 600MB RAM.

3.2 Měření dotazů

K dotazům uvádíme i tabulky s výsledky experimentálního měření rychlosti. Při měření jsme postupovali následujícím způsobem.

1. SPARQL dotaz převedeme do SQL.
2. SQL dotaz vložíme do měřicího systému. Ten automaticky provede kroky 3 až 8.
3. Proběhne restart databáze (vyprázdnění cache pamětí)
4. Pauza 10 vteřin, aby databáze dokončila start.
5. Spuštění SQL dotazu.
6. Změření času do vrácení první řádky.
7. Změření celkového času dotazu a celkového počtu vrácených řádek.
8. Druhé měření bez restartu databáze (body 4 až 7).

Výsledky ukázaly, že vyprázdnění cache v databázi a opakování měření má smysl, protože časy prvního a druhého pokusu se výrazně liší. Další iterace však již zrychlení nepřinášejí.

3.3 Jednoduché dotazy

Napřed se budeme zabývat dotazy, které jsou zaměřeny na jednotlivé konstrukce jazyka SPARQL.

Následující dotaz vrací seznam všech tříd v databázi.

```
select ?trida
where { ?trida a rdfs:Class }
```

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	91ms	90ms	20ms	19ms
B-strom	63ms	63ms	19ms	19ms
paralelní	38ms	37ms	21ms	20ms
bitmapa	73ms	72ms	21ms	20ms
	Počet řádků: 226			

Ukážeme si ještě dotaz, který vrací velký seznam, pro jehož získání je třeba spojenit velké množství trojic.

```
select ?jmeno ?prijmeni
where { ?osoba mt:ot_osoba__jmeno ?jmeno .
  ?osoba mt:ot_osoba__prijmeni ?prijmeni }
```

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	20s	2920ms	11s	914ms
B-strom	14s	878ms	13s	292ms
paralelní	13s	369ms	12s	315ms
bitmapa	19s	4634ms	11s	902ms
	Počet řádků: 91166			

Jako poslední ukážeme dotaz, kterým hledáme všechny objekty, jejichž hodnota je číslo 1.

```
select ?s ?p
where { ?s ?p 1 }
```

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	101s	274ms	83s	134ms
B-strom	87s	159ms	85s	196ms
paralelní	87s	177ms	84s	181ms
bitmapa	81s	234ms	80s	160ms
	Počet řádků: 1987905			

Celkově lze říct, že dosahované časy odpovídají očekávání.

3.4 Složitější dotazy

V této části se budeme zabývat dotazy, které mohou kombinovat i více základních konstrukcí SPARQL a měly by spíše odpovídat reálným dotazům, které by mohl uživatel pokládat nad uloženými daty.

Napřed uvažme dotaz, který na základě několika zadaných hodnot testuje, jestli taková osoba je nebo není v databázi.

```
select ?osoba
where {
  ?osoba mt:ot_osoba__jmeno 'Josef' .
  ?osoba mt:ot_osoba__prijmeni 'Dvořák' .
  ?osoba mt:ot_osoba__datum_narozeni
  '1968-04-06T00:00:00' .
  ?osoba mt:ot_osoba__rok_maturity '1987' .
  ?osoba mt:ot_osoba__trvaly_pobyt_v_cr 'A'.
  ?osoba mt:ot_osoba__pohlavi ?pohlavi .
  ?pohlavi mt:cht_ciselnik_plochy__kod '1'
}
```

V relační databázi s rozumně definovanými indexy by byl dotaz vyhodnocen téměř okamžitě. Protože některé z omezení (například na datum narození) mají velmi dobrou selektivitu (vyberou obvykle 0 až 1 trojici), bylo by možné vyhodnotit i tento dotaz v RDF velmi rychle díky tomu, že toto omezení nechá velmi málo možných ohodnocení proměnné osoba a ověřit existenci požadovaných hodnot pro tento subjekt lze díky indexu nad subjektem snadno.

Reálná měření však dopadla špatně. I v případě, že taková osoba v databázi neexistuje, trvá vyhodnocení dlouho.

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	8129ms		1154ms	
B-strom	826ms		117ms	
paralelní	4805ms		118ms	
bitmapa	5225ms		1147ms	
	Počet řádků: 0			

Následující dotaz vrací dvojice rodné číslo a email pro osoby, které mají oba údaje zadány.

```
select ?rc ?email
where {
  ?kontakt mt:ot_kontakt__id_osoba ?osoba .
  ?kontakt mt:ot_kontakt__druh_kontaktu
     ?dkon .
  ?kontakt mt:ot_kontakt__email ?email .
  ?dkon mt:cht_ciselnik_plochy__kod "2" .
  ?ident mt:ot_identifikace__id_osoba
     ?osoba .
  ?ident mt:ot_identifikace__druh ?rckod .
  ?rckod mt:cht_ciselnik_plochy__kod "2" .
  ?ident mt:ot_identifikace__identifikace
     ?rc
 }
```

Dosahované rychlosti ukazuje následující tabulka:

	První spuštění		Opakované spuštění	
	celkem	první řádek	celkem	první řádek
základní	54s	44s	54s	44s
B-strom	717s	716s	745s	743s
paralelní	16s	11s	14s	10s
bitmapa	275s	101s	286s	105s
	Počet řádků: 7861			

U tohoto dotazu se ukázala nevýhoda námi zvoleného přístupu, kdy veškerá data jsou uložena v jediné tabulce (to je však vynuceno obecností SPARQL dotazů) a optimalizace je ponechána na optimalizátoru relační databáze. Tento optimalizátor vychází ze statistik, které si o tabulkách udržuje.

Již jsme však zmínili, že naše data (ačkoliv mnoho pozorování by bylo možné zobecnit na většinu RDF dat) vykazují některé nezvyklé vlastnosti, co se týče celkových statistik. Důležité je například pozorování, že trojice s konkrétním predikátem představují pouze zlomek databáze a přesto jde až o stovky tisíc záznamů.

Tato vlastnost je pravděpodobná příčina toho, že optimalizátor dělá špatné odhady velikosti mezivýsledků při vyhodnocení dotazu. Po spojení několika tabulek TRIPLES dojde k odhadu, že výsledek spojení bude obsahovat pouze jeden řádek. Na základě tohoto předpokladu zvolí metodu spojení. Při skutečném vyhodnocení dotazu se často stane, že oproti jednomu předpokládanému řádku se do mezivýsledku dostanou stovky tisíc řádků, na což zvolená metoda spojení není vhodná.

Tím lze vysvětlit, proč vyhodnocení dotazu trvá nepřiměřeně dlouho.

Je také vidět, že mezi rychlostmi při různých metodách indexace jsou velké rozdíly. Obzvláště zajímavý je velký rozdíl mezi paralelní a neparalelní verzí B-stromu. Mnohonásobný nárůst rychlosti v paralelní verzi nemůže být způsoben jen větším dostupným výpočetním výkonem, protože databázový stroj měl k dispozici pouze dva procesory. Pravděpodobná příčina je, že různé plány vyhodnocení byly kvůli špatným odhadům optimalizátoru ohodnoceny podobnou celkovou cenou. Proto mohla drobná změna parametrů na vstupu optimalizátoru (například míra paralelizace) vést k výběru jiného plánu s podobnou odhadovanou cenou, který se při skutečném spuštění dotazu ukázal mnohem vhodnější.

Navíc dosahované časy se při opakovaném měření mění. Pravděpodobný zdroj tohoto chování je fakt, že Oracle při optimalizaci využívá i znalostí o předchozích vyhodnoceních stejného dotazu. Protože se všechny navržené plány vyhodnocení dotazu ukazují jako velmi neoptimální, snaží se optimalizátor (neúspěšně) o jejich zlepšení drobnou změnou plánu.

K řešení tohoto problému je možné přistupovat ze dvou směrů. Těmito směry se zabývají následující dvě kapitoly.

4 RDF indexy

Problémy s optimalizací jsou částečné způsobeny velkým množstvím spojení, které je potřeba pro vyhodnocení jednoho dotazu. Snížením tohoto počtu by se vyhodnocení mohlo urychlit.

Naše řešení (tzv. *RDF indexy*) spočívá v předvyhodnocení a uložení výsledků vhodně zvolených dotazů v databázi. Tyto dotazy musí v současné době definovat uživatel databáze, i když by jistě bylo možné je generovat automaticky na základě sledování dotazů pokládaných do databáze. Předvyhodnocená data jsou v databázi uložena jako B-strom (přesněji IOT - Index Organized Table) s pořadím sloupců tak, jak je definoval uživatel při vytváření indexu. Tím se chování RDF indexů podobá klasickým indexům, včetně toho, že jsou vhodné hlavně pro vyhledávání podle prvního indexovaného sloupce.

Pokud se takovýto předvyhodnocený dotaz objeví jako součást jiného dotazu, pak je možné jeho vyhodnocení spojením tabulek TRIPLES a LITERALS nahradit přímo tabulkou s výsledkem předvyhodnoceného dotazu.

Odstranění několika spojení znamená nejen eliminaci jejich výpočtu, ale také zjednodušení vyhodnocovaného SQL dotazu.

4.1 Implementace RDF indexů

Vytvořili jsme implementaci omezené verze navrhovaných RDF indexů. Tato implemetace podporuje jen indexy, které jsou definovány dotazem, který vyhovuje následujícím omezením:

- predikáty nejsou proměnné
- všechny predikáty jsou různé
- dotaz neobsahuje OPTIONAL, UNION a FILTER
- dotaz neobsahuje anonymní uzly

Pro vytváření indexů jsme rozšířili syntax jazyka SPARQL o klauzili CREATE INDEX.

```
CREATE INDEX jméno_indexu
AS ?proměnná1 ?proměnná2 ...
WHERE grafový_dotaz
```

Jako příklad ukážeme index nad jmény osob:

```
create index osobajmena as
?osoba ?jmeno ?prijmeni
where { ?osoba mt:ot_osoba__jmeno ?jmeno .
  ?osoba mt:ot_osoba__prijmeni ?prijmeni }
```

K výběru indexů, které budou použity pro vyhodnocení dotazu q, používáme jednoduchý hladový algoritmus, který postupně zkouší všechny indexy a pokud je možné některý použít, tak jej použije.

Tento cyklus se opakuje, dokud je v jeho průběhu nalezen alespoň jeden použitelný index. Pokud je takový index nalezen, pak jsou z q odstraněny trojice, které odpovídají indexu, a jejich použití je nahrazeno použitím vyhodnoceného indexu. Protože při každém použití indexu je z dotazu q odstraněna alespoň jedna trojice a nahrazena indexem, tento algoritmus vždy končí.

Snadno lze nalézt protipříklad, který ukazuje, že algoritmus není optimální vzhledem k počtu nahrazených trojic v dotazu q.

4.2 Rychlost RDF indexů

Vytvořili jsme index na základě tohoto dotazu:

```
select ?p ?q ?x ?y
where {
?x ns:p1 ?p . ?x ns:p2 ?y .
?y ns:p3 '2' . ?x ns:p4 ?q }
```

Tento index jsme použili pro vyhodnocení složitějšího dotazu, který vypadá takto:

```
select ?q ?r
where{
?u ns:p10 ?p . ?u ns:p11 ?v .
?u ns:p12 ?r . ?v ns:p13 '3' .
?x ns:p1 ?p . ?x ns:p2 ?y .
?y ns:p3 '2' . ?x ns:p4 ?q }
```

Bez použití indexu se dotaz do SQL přeloží takto (zkráceno):

```
select ... from
(select t1.object as V_p,
t17.object as V_q,
t5.object as V_r,
t1.subject as V_u,
t3.object as V_v,
t10.subject as V_x,
t12.object as V_y
from triples t1
inner join literals t2 on t2.id=t1.pred
inner join triples t3 on t1.subj=t3.subj
inner join literals t4 on t4.id=t3.pred
inner join triples t5 on t1.subj=t5.subj
inner join literals t6 on t6.id=t5.pred
inner join triples t7 on t3.obj=t7.subj
inner join literals t8 on t8.id=t7.pred
inner join literals t9 on t9.id=t7.obj
inner join triples t10 on t1.obj=t10.obj
inner join literals t11 on t11.id=t10.pred
inner join triples t12 on t10.subj=t12.subj
inner join literals t13 on t13.id=t12.pred
inner join triples t14 on t12.obj=t14.subj
inner join literals t15 on t15.id=t14.pred
inner join literals t16 on t16.id=t14.obj
inner join triples t17 on t10.subj=t17.subj
inner join literals t18 on t18.id=t17.pred
where t2.value='http://example.org/p10'
and t4.value='http://example.org/p11'
and t6.value='http://example.org/p12'
and t8.value='http://example.org/p13'
and t9.value='3'
and t11.value='http://example.org/p1'
and t13.value='http://example.org/p2'
and t15.value='http://example.org/p3'
and t16.value='2'
and t18.value='http://example.org/p4')
```

Druhá čtveřice trojic může být nahrazena indexem. Dotaz pak vypadá takto:

```
select ... from
(select t1.p as V_p,
t1.q as V_q,
t6.object as V_r,
t2.subject as V_u,
t4.object as V_v,
t1.x as V_x,
t1.y as V_y
from idx t1
inner join triples t2 on t1.p=t2.obj
inner join literals t3 on t3.id=t2.pred
inner join triples t4 on t2.subj=t4.subj
inner join literals t5 on t5.id=t4.pred
inner join triples t6 on t2.subj=t6.subj
inner join literals t7 on t7.id=t6.pred
inner join triples t8 on t4.obj=t8.subj
inner join literals t9 on t9.id=t8.pred
inner join literals t10 on t10.id=t8.obj
where t3.value='http://example.org/p10'
and t5.value='http://example.org/p11'
and t7.value='http://example.org/p12'
and t9.value='http://example.org/p13'
and t10.value='3')
```

Rychlost vyhodnocení se tím výrazně zlepší. Bez indexu trvalo vyhodnocení dotazu 92 vteřin, s indexem se čas zkrátil na 18 vteřin.

5 Přesné statistiky

RDF indexy představují možnost, jak čelit problémům s rychlostí vyhodnocení dotazu. K řešení problému je však možné přistupovat i jinak než eliminací spojení.

Problémem obvykle není velký počet spojení, jako spíš nevhodný způsob vyhodnocení, který zvolí optimalizátor. Protože však známe některé specifické charakteristiky RDF dat, jsme schopni si celkem snadno udělat mnohem lepší představu o průběhu vyhodnocení dotazu na tato data.

Protože četnosti subjektů jsou v reálných datech dosti omezené a existuje velké množství různých subjektů, nebylo by efektivní o nich uchovávat přesnější statistiky.

Na druhou stranu množství predikátů je relativně malé a ve většině SPARQL dotazů jsou předem zadány jejich konkrétní hodnoty. Proto, pokud by si systém pro každý predikát uchovával jeho četnost (s ohledem na jejich malý počet můžou být za běhu uloženy v paměti), byl by schopen dobře odhadnout velikost výsledků selekcí v listech stromu vyhodnocení dotazu.

Protože četnosti objektů jsou velmi různorodé, bylo by vhodné uchovávat i je. Například přidáním jednoho sloupce s touto četností do tabulky LITERALS. Při překladu dotazu by se systém mohl dotázat na tuto četnost do databáze a tím určit selektivitu trojice, ve které je daný objekt použit.

S pomocí těchto informací by bylo možné (například pomocí Oracle hints) ovlivnit plány vyhodnocení dotazů produkované optimalizátorem.

Tuto možnost jsme zatím neimplementovali.

6 Závěr

Rozsáhlá a složitá RDF data nejsou zatím běžně dostupná. Přestože jsou dostupná relativně velká data jako například WordNet a DBLP [11], jejich struktura je velmi jednoduchá. WordNet obsahuje šest tříd a pět predikátů. Přesto jsou tato data běžné používána pro testování RDF databázi, například v [5].

Podle našich experimentů není vhodné používat pouze takto jednoduchá data, protože dosažené výsledky nemusí vypovídat o výkonu v systémech, na které by RDF databáze měly být použity. Nejen velikost, ale i složitost dat, má vliv na vyhodnocení dotazů. A nelze očekávat, že by data skutečného Sémantického webu byla pravidelná a jednoduchá.

Proto jsme provedli testy nad rozsáhlými a hlavně složitými daty a narazili na problémy, které by nad jednoduchými daty nenastaly. Na základě těchto zkušeností jsme navrhli dva přístupy, jak s těmito problémy bojovat. Jeden z těchto přístupů jsme implementovali a měření potvrdila jeho pozitivní přínos.

Reference

1. Alexaki S., Christophides V., Karvounarakis G., Plexousakis D., Schol D., RQL: A Declarative Query Language for RDF. In Proceedings of the Eleventh International World Wide Web Conference, USA, 2002
2. Berners-Lee T., Hendler J., Lassila O., The Semantic Web. Scientific American, May 2001
3. Broekstra J., Kampman A., SeRQL: A Second Generation RDF Query Language. SWAD-Europe, Workshop on Semantic Web Storage and Retrieval, Netherlands, 2003
4. Carroll J.J., Klyne G., Resource Description Framework: Concepts and Abstract Syntax. W3C Recommendation, 10 February 2004
5. Chong E.I., Das S., Eadon G., Srinivasan J., An Efficient SQL-based RDF Querying Scheme. In Proc. of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005, 1216–1227
6. Dokulil J., (2006): Transforming Data from DataPile Structure into RDF. In Proceedings of the Dateso 2006 Workshop, Desna, Czech Republic, 2006, 54–62 `http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS//Vol-176/paper8.pdf`
7. Dokulil J., Dotazování nad RDF daty. Diplomová práce MFF UK Praha, 2006
8. Prud'hommeaux E., Seaborne A., SPARQL Query Language for RDF, W3C Working Draft, 23 November 2005
9. Seaborne A., RDQL – A Query Language for RDF. W3C Member Submission, 9 January 2004 `http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/`
10. RDFQL Scripting Reference RDFQL `http://www.intellidimension.com/default.rsp?topic=/pages/rdfgateway/reference/script/default.rsp`
11. `http://www.semanticweb.org/library/`

Softening splits in decision trees using simulated annealing

Jakub Dvořák and Petr Savický

Institute of Computer Science, Academy of Sciences of the Czech Republic
{dvorak,savicky}@cs.cas.cz

Abstract. *Predictions computed by a classification tree are usually constant on axis-parallel hyperrectangles corresponding to the leaves and have strict jumps on their boundaries. The density function of the underlying class distribution may be continuous and the gradient vector may not be parallel to any of the axes. In these cases a better approximation may be expected, if the prediction function of the original tree is replaced by a more complex continuous approximation. The approximation is constructed using the same training data on which the original tree was grown and the structure of the tree is preserved.*
The current paper uses the model of trees with soft splits suggested by Quinlan and implemented in C4.5, however, the training algorithm is substantially different. The method uses simulated annealing, so it is quite computationally expensive. However, this allows to adjust the soft thresholds in groups of the nodes simultaneously in a way that better captures interactions between several predictors than the original approach. Our numerical test with data derived from an experiment in particle physics shows that besides the expected better approximation of the training data, also smaller generalization error is achieved.

Keywords: Decision trees, soft splits, classification, simulated annealing.

1 Introduction

Classification trees are suitable for predicting the class in complex distributions, if a large sample from the distribution is available. The classical parametrical methods may not succeed in such situations, if they work with a closed formula describing the density in the whole predictor space. Decision trees and ensambles of trees are comparable to neural networks and SVM in classification accuracy. The predictor vector for a tree consists of a fixed number of numerical and categorical variables. In this paper, we consider single univariate decision trees with numerical predictors.

A trained classification tree usually does not only provide a discrete classification, but also an estimate of the confidence for it on a continuous scale. This confidence may be an estimate of the conditional probability of the classes, but this is not necessary. Even if it is not a good estimate of the probabilities, it may be a reasonable information. In the real world problems, it is frequently plausible to assume that the function which assigns such a confidence to each point of the predictor space is continuous. Even if the true distribution has a sharp boundary between the classes, a limited sample from the distribution does not provide enough information for a justified construction of a prediction function with a strict jump.

Classification trees constructed using the traditional methods like CART [2] or C4.5 [4] generate trees, whose internal nodes contain conditions of the form $x_{k_j} \leq c_j$, where x_{k_j} is one the predictors and c_j is a threshold value. The result of the use of such sharp threshold conditions is that the predictions computed by a classification tree are constant on axis-parallel hyperrectangles corresponding to the leaves and have strict jumps on their boundaries. Such functions may badly approximate boundaries of different type. This is the cost, which is paid for the simplicity of the classifier.

A soft threshold condition means that if the actual value of x_{k_j} is close to c_j, then both branches of the tree are evaluated and their results are combined using weights changing continuously with $x_{k_j} - c_j$. Soft splits were suggested first by Quinlan [4] including the implementation in C4.5. We use exactly the same extension of the decision tree classifiers, but use a substantially different technique for training.

Quinlan's technique determines the soft thresholds in each node separately using statistical estimation. In our approach, several thresholds corresponding to a group of nodes close to each other in the tree are adjusted simultaneously. Hence, the choice of the final thresholds is influenced also by the interactions between predictors. A nonsoft tree together with the training data used for its construction, are used as the input to an optimization phase, which tries to find the values of the parameters of the soft thresholds, which yield the best possible approximation of the training data. Since the structure of the tree, in particular, its number of nodes, is fixed during the optimization, overfitting was not observed in our experiments, although a computationally intensive optimization is used to tune the soft thresholds.

The goal of the postprocessing of the tree is to reach a smoother function that fits better the training data. Since the complexity of the classifier does not increase too much, one may expect to achieve also smaller generalization error. Besides better approximation in cases, where the unknown conditional prob-

ability function is continuous, we may obtain a better approximation even if the true value of the conditional probability makes a jump on a boundary between the regions of different classification, if the boundary is not in axis-parallel direction. Soft tree may represent a more complex function than a nonsoft tree. In particular, as a consequence of interactions of several predictors, the prediction function of a soft tree may have gradient vector in a general direction, while keeping the small number of nodes of the original tree. Approximating this using a nonsoft tree requires to use a stair like boundary with a large number of nodes.

A situation, where soft thresholds are well justified directly from an application domain, may be demonstrated for example on evaluating custumers asking for a credit card in a bank. The bank investigates several parameters of each customer, for example the current amount on his account, his regular monthly deposit amount, and some other criteria. Insufficiency in one of these criteria may be compensated by good values in some other. Instead of designing a tree with a large number of nodes, an approximating smaller tree with soft splits may be sufficient.

The current paper investigates classification accuracy on data from particle physics (MAGIC gamma telescope) considered already in [1]. In this comparison, ensambles of trees, in particular random forests, provided the best classifiers for these data. Our experimental results on these data demonstrate that the trees obtained by introducing soft splits may have substantially smaller generalization error than individual nonsoft trees.

We also compare the soft trees obtained by simulated annealing with soft trees obtained by C5.0[1]. There is no significant difference in test classification error between these two types of trees, see section Results, although the trees are obtained using substantially different principles. C5.0 finds the soft thresholds as bounds of confidence intervals based on a statistical model. It does not take into account whether the error on the training data changes or not. In fact, using thresholds constructed in this way frequently increases the training error. On the other hand, our approach optimizes the soft threshold purely by minimizing the training error. Our result shows that minimizing training error leads to similar results as the satistical estimation used in C5.0, at least on the MAGIC data.

2 Decision tree with soft splits

Let T be a decision tree with nodes v_j for $j = 1, \ldots, s$. We assume that if v_{j_1} and v_{j_2} are left and right successor of v_j, then $j < j_1$ and $j < j_2$. In particular, v_1 is the root. Let V be the set of indices of the internal nodes and U the set of indices of the leaves. The variable tested in node v_j is denoted x_{k_j} and the corresponding threshold value is denoted c_j. If C is the number of classes, then the label (response vector) $G(v)$ of a leaf v is a nonnegative real valued vector of dimension C, whose coordinates sum up to one. Let $T(x)$ be the function $\mathrm{R}^d \to \mathrm{R}^C$ computed by the tree, where d is the number of predictors. More generally, let $T_j(x)$ be the function computed by the subtree starting at v_j. In particular, $T_1(x) = T(x)$. Note that $T_j(x)$ is defined even in cases, when the computation of the whole tree T for x does not reach the node v_j.

If v_{j_1} and v_{j_2} are left and right successor of v_j, then we have $T_j(x) =$ if $x_{k_j} \le c_j$ then $T_{j_1}(x)$ else $T_{j_2}(x)$. If we define $I(\mathit{condition})$ equal to 1 if *condition* is true and equal to 0 if *condition* is false, then this is equivalent to

$$T_j(x) = I(x_{k_j} \le c_j) T_{j_1}(x) + I(x_{k_j} > c_j) T_{j_2}(x).$$

A tree with soft splits is obtained by replacing this by

$$T_j(x) = L_j(x_{k_j} - c_j) T_{j_1}(x) + R_j(x_{k_j} - c_j) T_{j_2}(x)$$

for appropriate continuous functions $L_j, R_j : \mathrm{R} \to [0, 1]$. It is required that if both subtrees of T_j return the same output vector, then T_j returns the same vector as well. Hence, we require $L_j(t) + R_j(t) = 1$ for all $t \in \mathrm{R}$. A natural further requirement is that L_j be nonincreasing with limits $L_j(-\infty) = 1$ and $L_j(\infty) = 0$. Hence, we also have that R_j is nondecreasing with limits $R_j(-\infty) = 0$ and $R_j(\infty) = 1$.

The functions L_j and R_j used in the current paper are piecewise linear functions interpolating the points in the table

t	$L_j(t)$	$R_j(t)$
$-\infty$	1	0
$-a_j$	1	0
0	1/2	1/2
b_j	0	1
∞	0	1

where the values $a_j \ge 0$ and $b_j \ge 0$ are parameters of the soft splits. If $a_j > 0$ and $b_j > 0$, then the functions L_j, R_j are uniquely determined by the above table. If some of the values a_j, b_j is zero, the corresponding function L_j, R_j is defined as a pointwise limit, which is noncontinuous. The limit function satisfies $L_j(0) = 1/2$ or $R_j(0) = 1/2$ as in any other case.

The tree with soft splits is obtained by replacing the evaluation function in each internal node by the above one. This requires to specify the parameters (a_j, b_j) in all internal nodes. Let $\theta = \{(a_j, b_j)\}_{j \in V}$. The functions L_j, R_j defined above depend on θ. So, the correct notation for them is $L_j(\theta, t), R_j(\theta, t)$. Moreover, let $T(\theta, x)$ and $T_j(\theta, x)$ denote the functions

[1] http://www.rulequest.com, commercial version of C4.5.

computed by the whole tree and the tree starting at node v_j, respectively, if the soft splits determined by θ are used. We again have $T(\theta, x) = T_1(\theta, x)$.

If $x_{k_j} = c_j$, then the soft split always evaluates both subtrees and returns their arithmetic mean. If $x_{k_j} \neq c_j$ the situation depends on x_{k_j} as follows. If $x_{k_j} \leq c_j - a_j$ or $x_{k_j} \geq c_j + b_j$, then the evaluation function in node v_j behaves in the same way as in the original tree and returns the value of one of the two subtrees. If $x_{k_j} \in (c_j - a_j, c_j + b_j)$, then evaluating $T_j(x, \theta)$ requires to compute both subtrees and the output is a combination of their values.

Note that $T(0, x)$ behaves similarly to $T(x)$, but there is a difference. If the computation of $T(x)$ never reaches a node, where $x_{k_j} = c_j$, then we have $T(0, x) = T(x)$. However, if $x_{k_j} = c_j$ is satisfied at some step of the computation, the results may differ, since evaluation of $T(0, x)$ combines both subtrees, while evaluation of $T(x)$ uses only one of them.

For every pair of nodes v_j and v_{j_1} such that v_{j_1} is one of the two successors of v_j, let $H(\theta, v_j, v_{j_1})(x)$ be the following function defined on the predictor vector x.

$$H(\theta, v_j, v_{j_1})(x) = \begin{cases} L_j(\theta, x_{k_j}) \text{ if } v_{j_1} \text{ is the left} \\ \qquad\qquad \text{successor of } v_j \\ R_j(\theta, x_{k_j}) \text{ if } v_{j_1} \text{ is the right} \\ \qquad\qquad \text{successor of } v_j \end{cases}$$

For every leaf v, let $Path(v)$ be the uniquely determined path from the root to v. Then an explicit formula for the function computed by a tree with soft splits is

$$T(\theta, x) = \sum_{\substack{j \in U \\ (u_1, \ldots, u_k) = Path(v_j)}} G(v_j) \prod_{i=2}^{k} H(\theta, u_{i-1}, u_i)(x).$$

The formula may be verified by induction starting at the leaves.

3 Optimization of the soft splits

The method described in this paper assumes that a nonsoft classification tree T with two classes 0 and 1 is available. Such a tree may be obtained using a method like CART or C4.5 without softening. We used the R implementation of CART. The response vector is two dimensional in this case and the two coordinates are assumed to sum up to one. Hence, each of the coordinates alone carries the full information on the prediction. Let us denote $T^*(x)$ the component of the response vector computed by tree T, which corresponds to class 1. The method of the current paper assumes that it is possible to find a threshold h such that the rule "predict 1 iff $T^*(x) \geq h$" provides a reasonable classification.

The goal is to find θ such that the error of classification using $T^*(\theta, x) \geq h$ on unseen cases is smaller than in the original tree. Since the algorithm has access only to the training data, the method tries to achieve the above goal by minimizing the error of $T^*(\theta, x)$ on the training data. This error may be measured in different ways. We tested first simply the classification error, but it appeared to be better to use a continuous error defined on the data (x_i, y_i), $i = 1, \ldots, m$, $y_i \in \{0, 1\}$, by the formula

$$f(\theta) = \sum_{i=1}^{m} \mathrm{e}^{\alpha(|T^*(\theta, x_i) - y_i| - 1)}, \tag{1}$$

where α was chosen to be 4. Experiments show that an improvement on unseen cases is indeed achieved.

Since the minimized function $f(\theta)$ is not a smooth function and has a large number of local minima, we used simulated annealing available in R Statistical Package [5] using method SANN of function "optim". Since this does not allow to restrict the range of θ to nonnegative values, we used a large penalty (number of errors larger than the number of training cases) for θ, which contain a negative value. The initial value of θ was chosen to be 0, i.e. the optimization starts approximately at the original nonsoft tree.

The dimension of the optimization problem (the number of parameters of the minimized function) is two times the number of internal nodes. In order to make the optimization process independent on the scaling of the data, the optimization function uses a normalized vector of parameters $\theta' = \{(a'_j, b'_j)\}_{j \in V}$, where $a'_j = a_j / a_{j,0}$, $b'_j = b_j / b_{j,0}$. The normalizing factors $a_{j,0}, b_{j,0}$ are defined using the original nonsoft tree as follows.

In each node of the tree, we find the hyperrectangle that is guaranteed to contain all training points that go through the node during classification. For the root, the hyperrectangle is the cartesian product of the smallest closed intervals, which contain all the values of the corresponding predictor in the training data. If v_{j_1}, v_{j_2} are the two successors of v_j, then the hyperrectangles assigned to them are obtained by splitting the hyperrectangle assigned to v_j by the hyperplane $x_{k_j} = c_j$. Then, the values $a_{j,0}$ and $b_{j,0}$ are chosen so that the interval $[c_j - a_{j,0}, c_j + b_{j,0}]$ is exactly the range of x_{k_j} within the hyperrectangle assigned to v_j.

4 Iteration of simulated annealing

In this section, we discuss the strategy to look for θ that minimizes the function (1). For the purpose of this section, we denote θ as $x \in \mathrm{R}^n$, where n is the

length of θ. It appeared to be better to split the minimization into phases, in each of which, only a small randomly chosen subset of arguments is modified. Let us introduce the following notation for this purpose. Let $S \subseteq \{1, \ldots, n\}$ and $z \in \mathrm{R}^n$. By R^S we mean the set of vectors $\{x_i\}_{i \in S}$, i.e. the vectors from $\mathrm{R}^{|S|}$ whose coordinates are indexed by elements of S instead of consecutive integers. Then, let $f[S, z] : \mathrm{R}^S \to \mathrm{R}$ be the function defined for every $x \in \mathrm{R}^S$ by $f[S, z](x) = f(y)$, where

$$y_i = \begin{cases} x_i & \text{if } i \in S \\ z_i & \text{otherwise} \end{cases}$$

Optimization is performed by a sequence of calls of method SANN of optim, which is an implementation of simulated annealing. The initial approximation of each call is the best solution found during the previous call. The initial temperature for all calls is $temp = 10$ and the bound on the number of iterations is $maxit = 101$ for all calls.

For each call of optim, a set S of k indices of variables is selected, see below. In the given call, $f(x)$ is minimized by modifying only variables with indices in S. Formally, the minimized function is the function $f[S, x_0](x)$ with $k = |S|$ arguments, where x_0 is the result of the previous call. The restriction of x_0 to the selected set S of indices is also the initial value for x in the current call.

One call of optim is successful, if it succeeds to find a better solution than the initial one. The whole process stops, when 50 consecutive calls are unsuccessful.

As mentioned above, for each call of optim, a set S of indices of variables is selected. Let s be a variable from the vector θ. This means that s is a_j or b_j for some j. Then, let T_s be the maximal subtree of the tree T, such that the root of T_s is the left son of the node v_j in the case that s is a_j and the right son of v_j if s is b_j. The selection of S starts with selecting randomly a variable s such that the root of T_s is not a leaf. Then S contains s and variables a_i, b_i where i traces through all indexes of nodes in the two top levels of T_s. Depending on the structure of T and selection of s the set S contains 3, 5 or 7 variables.

5 Experimental setup

In a single run of the experiment, the available data D were split at random in ratio 2:1 into a training set D_1 and a test set D_2 and the four classifiers obtained by the following methods were constructed:

1. CART.
2. Soft tree obtained by the method described in the prevoius section from CART trees.
3. C5.0 without softenning.
4. C5.0 with softenning (option "`-p`").

More detail is given in subsections below.

5.1 CART

The trainig set D_1 was further split in ratio 2:1 into D_{11} and D_{12}. The larger part D_{11} was used for growing the tree, the smaller D_{12} was used for pruning as the validation set (cost complexity pruning in CART method). The result of the pruning is a sequence of trees of different sizes. Accuracy of these trees is reported for comparison with the other methods. In order to show that the comparison result does not depend on the selection of the tree in the sequence, we select the best tree on the test set D_2 and report its accuracy. Even such trees are worse than the soft trees, whose error is measured using standard methodology.

5.2 Softening trees from CART

We use the sequence of pruned trees constructed by CART. Trees without split nodes are not considered. When interpreting the soft tree as a classifier, we used threshold $h = 0.5$. This means that the class with the larger confidence (we have two classes) in the response vector is predicted.

The error of the resulting soft tree is never worse than in the original tree, however, it is sometimes close to it. Such soft trees are discarded. The optimization is considered unsuccessful, if the ratio of the error of the original tree over the error of the soft tree is less than 1.01. The sequence of trees from CART contains trees of different sizes. Smaller trees have higher chance to be improved by one run of the softening procedure. On the other hand, if the softening procedure succeeds to improve a large tree, the result is usually better than for small trees. In order to balance between these two effects, we used a strategy which is splitted into steps numbered by $i = 1, 2, \ldots$. In step i, the softening procedure tries to improve all of the i largest trees in the sequence. The process terminates, when 10 trees successfuly improved by softening are collected. The resulting classifier is determined as the tree with the smallest classification error on D_1 among the 10 trees obtained by the above strategy. The error of this tree on D_2 is reported.

5.3 C5.0

We used C5.0 release 1.15. The confidence level, which determines the amount of pruning was chosen 0.1 (option "`-c 10`"), which appeared to be the best among several values that we tried. For each split of the data, C5.0 was run twice, with and without the option "`-p`", which forces that a softened tree is constructed.

The reason for choosing the confidence level equal to 0.1 was the following: We computed for each split of the data error rates of C5.0 softened trees for the

values of confidence level 0.01, 0.02, 0.05, 0.10, 0.15, 0.20, 0.25 and 0.30. Then for each of these values we compared the error rate of C5.0 and the error rate of the tree from CART softened. The value 0.10 is the one, for which the error rates of C5.0 softened trees are lower than the error rates of softened trees from CART in the highest number of data splits.

6 Results

We used data simulating registration of gamma and hadron particles in Cherenkov imaging gamma ray telescope MAGIC [1]. There are 10 numerical predictors and 2 classes. The predictors are numerical values that are produced by the registration device and characterize the registered particle. Class signal represents cases, where the registered particle is gamma. Class background corresponds to hadrons, mostly protons. The number of cases in the dataset is 19020.

The data were created by a complex Monte Carlo simulation [3] that approximates the development of a shower of particles generated by a high energy primary particle that reaches the atmosphere. The result of the simulation is an estimate of the number of Cherenkov ultraviolet photons that reach different pixels in the focus of an antenna at the ground and form a single registered event. The 10 predictors are numerical parameters of the geometric form of the obtained image. Generating each case in the dataset required several seconds of CPU time.

Creating the dataset was a part of the project of constructing the telescope and was used to support the decision, which classification technique to use in regular observations using the telescope. On the basis of [1], random forest was selected and is still used.

We used 7 random splits of this set in the ratio 2:1 into D_1 and D_2. For each of these splits, four classifiers were constructed using the methods described in the previous section. Besides the nonsoft CART trees, the classifier was constructed using only D_1 and its accuracy was estimated using D_2. Due to the large size of both training and testing set, there is a strong relationship between the training error and test error. The test errors on D_2 are presented in the following tables.

	CART non-soft	CART softened	ratio soft/non-soft
1	0.1677	0.1377	0.8213
2	0.1610	0.1371	0.8511
3	0.1598	0.1366	0.8549
4	0.1659	0.1393	0.8394
5	0.1591	0.1377	0.8652
6	0.1550	0.1380	0.8901
7	0.1533	0.1371	0.8940

The next table allows to compare the error rates of trees from CART softened and trees from C5.0 softened. The ratio of these two errors is included.

	CART softened	C5.0 non-soft	C5.0 softened	ratio CART s./C5.0 s.
1	0.1377	0.1473	0.1391	0.9898
2	0.1371	0.1535	0.1438	0.9529
3	0.1366	0.1456	0.1323	1.0322
4	0.1393	0.1508	0.1380	1.0091
5	0.1377	0.1478	0.1366	1.0081
6	0.1380	0.1516	0.1483	0.9309
7	0.1371	0.1479	0.1410	0.9720

In our experiments, softening using simulated annealing led to soft trees with similar accuracy on the MAGIC dataset compared to that of C5.0 softened tree. In order to formulate the result in terms of statistical significance, let p_1 be the probability of the event that the error rate of a tree softened using simulated annealing on the MAGIC dataset is lower than error rate of C5.0 softened tree. The two sided 0.95 confidence interval for p_1 obtained using the exact binomial test is approximately $[0.18, 0.9]$, since we have 4 successes out of 7 trials. This means that the result of the experiment does not imply any significant difference between the two methods.

The comparison of trees softened using simulated annealing with the non-soft trees from the CART method gives much better result. The error of the soft tree obtained by SANN was always by at least 10% better than that of the corresponding nonsoft tree. In order to formulate the result in terms of statistical significance, let p_2 be the probability of the event that the error rate of a tree softened using simulated annealing on the MAGIC dataset is lower than 90% of error rate of CART tree. The one-sided lower 0.95 confidence limit for p_2 obtained using the exact binomial test is approximately 0.65, since we have 7 successes out of 7 trials.

Acknowledgement. The first author was supported by Czech Science Foundation (GACR) under the grant No. GD201/05/H014. The second author was partially supported by the "Information Society" project 1ET100300517. Both authors were partially supported also by the Institutional Research Plan AV0Z10300504.

References

1. R.K. Bock, A. Chilingarian, M. Gaug, F. Hakl, T. Hengstebeck, M. Jiřina, J. Klaschka, E. Kotrč, P. Savický, S. Towers, A. Vaicilius, Methods for Multidimensional Event Classification: a Case Study Using Images from a Cherenkov Gamma-Ray Telescope. Nuclear Instruments and Methods in Physics Research, Section A, Volume 516, Issue 2-3, 2004, 511–528

2. L. Breiman, J.H. Friedman, R.A. Olshen, C.J. Stone, Classification and Regression Trees. Belmont CA: Wadsworth, 1993
3. D.Heck et al., CORSIKA, A Monte Carlo Code to Simulate Extensive Air Showers. Forschungszentrum Karlsruhe FZKA 6019, 1998
4. J.R. Quinlan, C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, San Mateo — California, 1993
5. R Development Core Team, R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, 2005 URL http://www.r-project.org

Geometry of probabilistic models

Zdeněk Fabián

Institute of Computer Science, Academy of Sciences of the Czech Republic
Pod Vodárenskou věží 2, Praha 8
zdenek@cs.cas.cz

Abstract. *In present paper, the main steps of the introduction of the core function [1] and Johnson score and new numerical characteristics of continuous distributions [2] are reviewed and elucidated. Johnson score is used for an introduction of a Johnson distance in the sample space.*

1 Introduction

Statistical parametric model is a set of distributions $\mathcal{F} = \{F_\theta, \theta \in \Theta\}$, $\Theta \subseteq \mathbb{R}^m$. Observed data $\mathbf{x} = (x_1, \ldots, x_n)$ are supposed to be realizations of independent identically distributed random variables distributed according to $F_\theta \in \mathcal{F}$ with unknown θ. Denoting by $f_\theta(x)$ the density of F_θ, the likelihood function is defined as function of θ

$$L(\theta) = f_\theta(\mathbf{x}).$$

It is easier to use the information contained in $L(\theta)$ in form of vector function $\mathbf{U}(\theta) = (U_{\theta_1}(\theta), \ldots, U_{\theta_m}(\theta))$, the components of which are the likelihood scores for θ_k,

$$U_{\theta_k}(\theta) = \frac{\partial}{\partial \theta_k} \log L(\theta).$$

The solution of the system of likelihood equations

$$\sum_{j=1}^{n} U_{\theta_k}(x_j; \theta) = 0, \quad k = 1, \ldots, m \tag{1}$$

is so called maximum likelihood estimate of θ, the best estimate if the data are actually distributed according to $F_\theta \in \mathcal{F}$. There are some other statistical tasks, however (for instance: estimation of simple characteristics, correlations), solution of which could be considerably simplified if we could work instead of $\mathbf{U}(\theta)$ with a scalar function in a form

$$S(\mathbf{x}; \theta) = \varphi\left(L(x; \theta); \frac{dL(x; \theta)}{dx}\right)|_{x=\mathbf{x}}.$$

Let us briefly describe main steps of the solution of the problem presented in [1] and discussed in [3]-[5].

Distribution F will be said to be supported by $(a, b) \subset \mathbb{R}$ if it has density

$$f(x) \begin{cases} > 0 \text{ for } x \in (a, b) \\ = 0 \text{ for } x \in \mathbb{R} - (a, b). \end{cases}$$

The natural inference function of distribution G with density g supported by $\mathbb{R}$ is its score function

$$Q(y) = -\frac{g'(y)}{g(y)}. \tag{2}$$

The reason for this assertion is as follows. Let $G_{\mu,s}$ be distribution with location parameter μ and density in the form

$$g_{\mu,s}(y) = \frac{1}{s} g\left(\frac{y - \mu}{s}\right). \tag{3}$$

Score function $Q_{\mu,s}(y)$ of $G_{\mu,s}$,

$$Q_{\mu,s} = -\frac{1}{g_{\mu,s}(y)} \frac{dg_{\mu,s}(y)}{dy} = \frac{\partial}{\partial \mu} \log g_{\mu,s}(y),$$

equals to the likelihood score $U_\mu(y; \mu, s)$ for the parameter expressing the central tendency of $G_{\mu,s}$.

A much more difficult task is to find a scalar inference function for distributions supported by $\mathcal{X} \neq \mathbb{R}$ (with 'partial support'). Score functions (2) of distributions with partial support are functions with odd behavior and a contingent choice of the likelihood score for certain parameter fails since it is unclear which of the parameters of distributions with partial support could represent a measure of their central tendency.

A procedure to gain a suitable scalar inference function for distributions with 'partial support' was suggested in [1]: to view any distribution F with interval support (a, b) as transformed '*prototype*' supported by $\mathbb{R}$. As a suitable transformation $\eta^{-1}: \mathbb{R} \to (a, b)$ was chosen the inverse of the Johnson transformation [6] adapted to arbitrary open interval, $\eta : (a, b) \to \mathbb{R}$, given by

$$\eta(x) = \begin{cases} x & \text{if } \quad (a, b) = \mathbb{R} \\ \log(x - a) & \text{if } -\infty < a < b = \infty \\ \log \dfrac{(x - a)}{(b - x)} & \text{if } -\infty < a < b < \infty \\ \log(b - x) & \text{if } -\infty = a < b < \infty. \end{cases} \tag{4}$$

Let G be prototype of F so that $F(x) = G(\eta(x))$. An interesting characteristic of F was shown to be the transformed score function of the prototype,

$$T(x) = Q(\eta(x)), \tag{5}$$

termed in [1]-[5] the *core function*. By using the well-known relation

$$f(x) = g(\eta(x))\eta'(x), \tag{6}$$

formula

$$T(x) = \frac{1}{f(x)}\frac{d}{dx}\left(-\frac{1}{\eta'(x)}f(x)\right) \tag{7}$$

was derived from (5) showing that the core function of a distribution with differentiable density can be determined without reference to its prototype by – somewhat sophisticated – differentiating of the density according to the variable.

For many distributions, mapping (4) represents the best choice: model distributions often have exponential forms of densities for which is the logarithmic transformation suitable and, moreover, by using (4):

i/ the prototype of the lognormal distribution is the normal distribution,

ii/ the core function of the rectangular distribution on (a, b) is linear.

Nevertheless, it is possible to find transformations for which explicit forms of (7) are for particular distributions more simple.

The most important property of Johnson core functions was formulated in Theorem 1 in [1], see also [3]-[4]. Theorem concerns of parametric distributions supported by interval $(a, b) \neq \mathbb{R}$ in form $F_{t,s} = G_{\mu,s}\eta$ where the prototype $G_{\mu,s}$ has density (3) and where

$$t = \eta^{-1}(\mu)$$

is the image of the location of the prototype, called a *Johnson parameter*. The density of $F_{t,s}$ is, by (3) and (6), in the form

$$f_{t,s}(x) = \frac{1}{s}g\left(\frac{\eta(x) - \eta(t)}{s}\right)\eta'(x).$$

Denoting by $T_{t,s}$ the core function (7) of distribution $F_{t,s}$, the theorem states that

$$U_t(x; t, s) = \frac{\partial}{\partial t}\log f_{t,s}(x) = \eta'(t)T_{t,s}(x), \tag{8}$$

i.e., that the core function of a distribution with prototype in location and scale form is proportional to the likelihood score for the Johnson parameter.

2 Johnson score and characteristics of continuous distributions

Result (8) was not general, however, since the density of distribution G with full support need not have the location parameter and, consequently, the transformed distribution $F = G\eta$ need not have the Johnson parameter. By realizing that t in $\eta'(t)$ on the r.h.s of equation (8) is the *value* of the Johnson parameter for which $T_{t,s}(t) = 0$, the concept of the core function was further generalized in [2] for any regular distribution with unimodal prototype .

Definition 1. *Let F be regular distribution with support $(a, b) \subseteq \mathbb{R}$. Let $\eta : (a, b) \to \mathbb{R}$ be given by (4), $T(x)$ be given by (7) and the solution x^* of equation $T(x) = 0$ be unique. A Johnson score of distribution F is defined by*

$$S(x) = \eta'(x^*)T(x).$$

Definition 1 adjoins to any distribution F with unimodal prototype a unique scalar function $S(x)$. With respect to the Johnson score, continuous probability distributions are of three types:

i/ If F has support $\mathbb{R}$, then $\eta'(x) = 1$ and $S(x)$ is the usual score function (2) (which is practically not used in probability theory and mathematical statistics, however, since it is considered as a concept which is not general). For standard normal distribution Johnson score $S(x) = x$ brings nothing new.

ii/ Johnson scores $S_{t,s}(x)$ of distributions supported by $(a, b) \neq \mathbb{R}$ with prototypes in the location and scale form (3) are equal to the likelihood scores $U_t(x; t, s)$ for *Johnson parameter* $t = \eta^{-1}(\mu)$. t is the image of the location (mode) of the prototype in mapping η. Johnson scores of distributions of this type are the well-known likelihood scores.

iii/ Johnson scores of other distributions with partial support and without the Johnson parameter are new functions. This is true even for distributions without parameters: for instance, the standard log-logistic distribution with support $(0, \infty)$ and density $f(x) = 1/(1+x)^2$ is characterized by the unknown (!) Johnson score $S(x) = (x - 1)/(x + 1)$.

Since $\eta'(x) > 0$, x^* appears to be the solution of equation

$$S(x) = 0.$$

This value can be taken as a measure of the central tendency of distribution F. It is easy to see that

$$ES = \int_a^b S(x)f(x)\,dx = 0. \tag{9}$$

This is the first Johnson score moment and x^* thus can be termed the *Johnson mean*.

Consider data sample $\mathbf{x}$ distributed according to F with unknown Johnson mean x^*. If F is a parametric distribution, x^* is a function of the parameters. Let us write Johnson score $S(x)$ of F in form $S(x; x^*)$. By (9),

the estimate $\hat{x}^*$ of x^*, the *sample Johnson mean*, can be obtained as a solution of an analogy of (1),

$$\frac{1}{n}\sum_{i=1}^{n} S(x_i; x^*) = 0. \tag{10}$$

For distributions of type ii/ it holds that $\hat{x}^* = \hat{t}$ where $\hat{t}$ the maximum likelihood estimate of the Johnson parameter, and Johnson score is the influence function (see [7]). Generalizing, we consider the Johnson score as the *influence function of the distribution*, describing the influence of value x_i for construction of the 'central point' of the distribution.

Function $S^2(x)$ attains its minimum at x^*, which is the least informative point of the distribution (cf. [5]), and value

$$ES^2 = \int_a^b S^2(x) f(x)\, dx$$

is the analogy of the Fisher information. Function $S^2(x)$ thus can be taken as an *information function* of the distribution, describing the information carried by x. Its reciprocal value

$$\omega^2 = (ES^2)^{-1}, \tag{11}$$

which we call a *Johnson variance*, was proposed in [5] and [2] as a measure of dispersion of values of distribution F around x^*.

The distance in the sample space of distribution F introduced in [3]-[5] is expressed by means of the Johnson score as follows.

Definition 2. *Johnson distance of points x_1 and x_2 taken from distribution F with Johnson score S is defined by*

$$d_{\mathcal{J}}(x_1, x_2) = \omega |S(x_2) - S(x_1)|, \tag{12}$$

where ω^2 is given by (11).

Another important function connected with Johnson score is its derivative

$$W(x) = \frac{dS(x)}{dx} \tag{13}$$

which can be called a *weight function* of F. By using (13), $d_{\mathcal{J}}(x_1, x_2) = \omega \int_{x_1}^{x_2} W(x)\ dx$. Let ds denotes the element of length. Function $W(x)$ defines Riemannian geometry in the sample space (a, b) of distribution F in the form $ds(x) = \omega W(x) dx$. The value ω can be interpreted as the length unit.

3 Geometry of distributions

For distributions with support $(0, \infty)$ it holds that $\eta'(x) = 1/x$ and the core functrion is

$$T(x) = -1 - xf'(x)/f(x). \tag{14}$$

The algorithm of determining geometric characteristics of an arbitrary distribution F with continuously differentiable density $f(x)$ supported by $(0, \infty)$ consists of four steps:

i/ find the (Johnson) core function $T(x)$ by means of (14)

ii/ find the Johnson point $x^* : T(x^*) = 0$ and Johnson score

$$S(x) = \frac{1}{x^*} T(x),$$

iii/ find ES^2 and Johnson variance $\omega^2 = 1/ES^2$,

iv/ $S^2(x)$ and (13 are the information and weight functions and (12) the distance in the sample space of distribution F.

In Table 1 are the densities of some commonly used distributions supported by $(0, \infty)$ and their usual characteristics, the mean and variance. Most of them are given by rather cumbersome expressions. Moreover, the mean and variance of the beta-prime distribution (a variant of the Fisher-Snedecor distribution) exist only in limited ranges of parameters ($q > 1$ and $q > 2$, respectively) and are practically of no use even if they exist: they cannot be estimated without approximate knowledge of their values.

$F(x)$	$f(x)$	m	σ^2
Weibull	$\frac{\beta}{x}(\frac{x}{t})^\beta e^{-(\frac{x}{t})^\beta}$	$t\Gamma(\frac{\beta+1}{\beta})$	$t^2(\Gamma(\frac{\beta+2}{\beta})$ $-\Gamma^2(\frac{\beta+1}{\beta}))$
lognormal	$\frac{\beta}{\sqrt{2\pi}x} e^{-\frac{1}{2}\log^2(\frac{x}{t})^\beta}$	te^{1/β^2}	$t^2 e^{1/\beta^2}(e^{1/\beta^2} - 1)$
gamma	$\frac{\gamma^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\gamma x}$	α/γ	α/γ^2
beta-prime	$\frac{1}{B(p,q)} \frac{x^{p-1}}{(x+1)^{p+q}}$	$\frac{p}{q-1}$	$\frac{p(p+q+1)}{(q-1)^2(q-2)}$

Table 1. Density, mean m and variance σ^2 of distributions with support $(0, \infty)$. Γ is the gamma function, B the beta function.

Johnson characteristics of distributions from Table 1 are given in Table 2.

Johnson mean and variance of the gamma distribution are (incidentally) identical with the usual mean and variance. Johnson characteristics of the Weibull and lognormal distribution (the distributions of type ii/ with Johnson parameter) are incredibly simple: their unit length ω is the Johnson mean $x^* = t$ multiplied by the scale parameter of the prototype. The gamma and beta-prime distributions are examples of distribution without Johnson location,for which the Johnson score is an unknown function, generating characteristics which exist for any value of parameters and, in the case of the beta-prime distribution, look like the usual mean and variance with 'corrected' denominators.

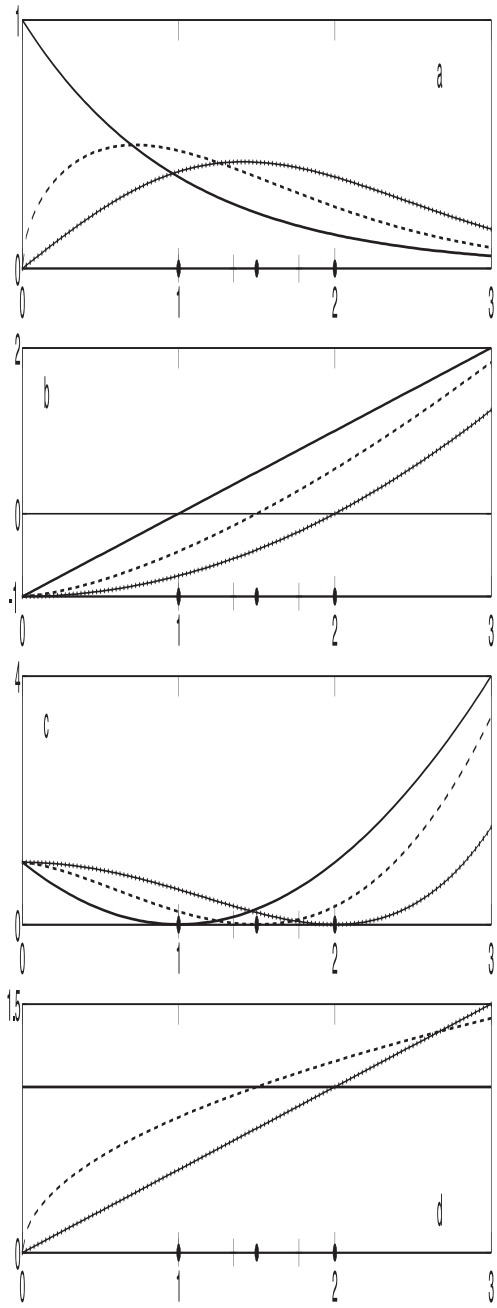


Fig. 1. Densities (a), Johnson scores (b), information functions (c) and weight functions (d) of Weibull distribution with $x^* = 1$ (full line), $x^* = 1.5$ (dashed) and $x^* = 2$ (dotted line). All distributions have Johnson variance $\omega^2 = 1$. $*$: Johnson mean, |: mean.

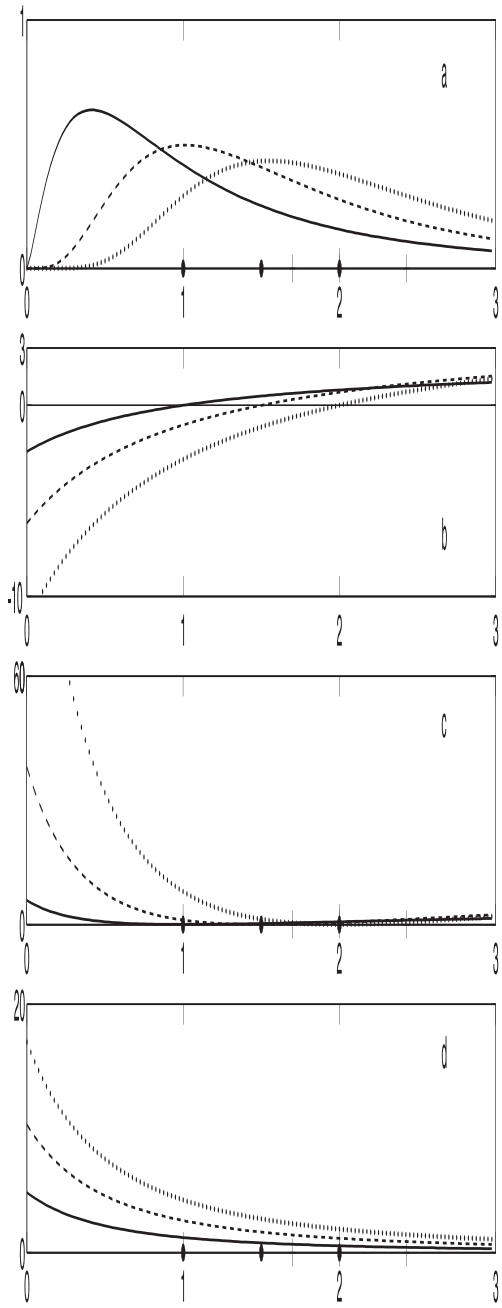


Fig. 2. Densities (a), Johnson scores (b), information functions (c) and weight functions (d) of beta-prime distributions with the same Johnson mean and Johnson variances and notation as in Fig.1.

$F(x)$	$S(x)$	x^*	ω^2
Weibull	$\frac{\beta}{t}((\frac{x}{t})^\beta - 1)$	t	t^2/β^2
lognormal	$\frac{\beta}{t}\log(\frac{x}{t})^\beta$	t	t^2/β^2
gamma	$\gamma(\frac{x}{\alpha/\gamma} - 1)$	α/γ	α/γ^2
beta-prime	$\frac{q}{p}\frac{qx-p}{x+1}$	p/q	$\frac{p(p+q+1)}{q^3}$

Table 2. Johnson score, Johnson mean and Johnson variance of distributions from Table 1.

An example of a prototype distribution without location and scale parameters is the prototype of the beta-prime distribution with density

$$f_{p,q}(x) = \frac{1}{B(p,q)}\frac{e^{px}}{(e^x+1)^{p+q}}.$$

Its score function is $S(x;p,q) = (qe^x - p)/(x+1)$ and $ES^2 = pq/(p+q+1)$. Let us find the symmetric prototype beta distribution with $\omega = 1$. The solution of equation $ES^2 = 1$ for $p = q$ is $p = 1+\sqrt{2}$. In Fig.3, a surprising coincidence of $f_{1+\sqrt{2},1+\sqrt{2}}(x)$ with the density of the standard normal is apparent.

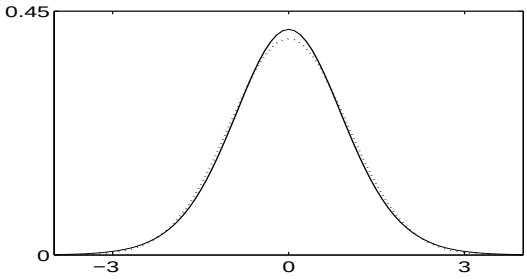


Fig. 3. Densities of prototype beta ($p = q = 1+\sqrt{2}$, full line) and standard normal (dotted line) with $\omega = 1$.

Fig.1 and Fig.2 show functions describing the Weibull and beta-prime distributions. Weibull with $x^* = t = 1$ (and $\omega^2 = 1$) is the exponential distribution. Its Johnson score is linear and weight constant, which reflects the fact that the exponential random variable is 'without memory'. The is from the group of so called heavy-tailed distributions with great density of probability of the occurrence of data far from the main bulk, so called outliers. Since its Johnson score is bounded and the weight of large data is small, the averages of the Johnson scores (10) of the beta-prime distribution are resistent to outliers in data.

Fig.4 shows distances from the Johnson mean in the sample space $(0,\infty)$ of some distributions from the tables above. The linear distance (dotted line) belongs to the gamma and to the Weibull distribution with $t = 1$. Other distances are non-linear. The distance of the lognormal distribution is $d_{\mathcal{J}}(x,1) = |\log x|$ and the distance of the heavy-tailed beta-prime distribution

$$d_{\mathcal{J}}(x,1) = q\left(\sqrt{ES^2}\right)^{-1}|x-1|/(x+1)$$

is bounded. These distances are to be used in the testing of hypotheses $H_0 : \hat{x}^* = x^*$ against alternatives $H_1 : \hat{x}^* \neq x^*$ for $\hat{x}^*$ estimated by (10).

Finally, let us remark that the Johnson characteristics do not depend on the speed with which the density approaches to zero. On the other hand, distributions with non-unimodal prototypes need further considerations.

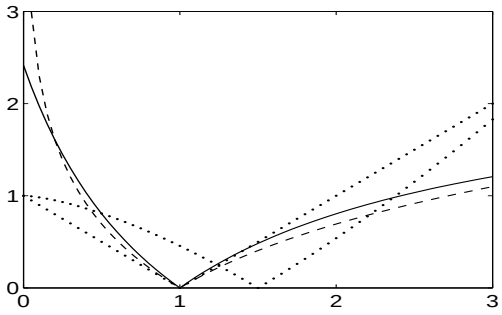


Fig. 4. Distance from the 'center of the distribution' (Johnson mean) in sample spaces of some distributions: beta-prime (full line), lognormal (dashed line), Weibull (dotted lines). Johnson variance of plotted distributions is $\omega^2 = 1$.

Aknowledgements. The work was supported by Grant Agency AS CR under grant number IAA1075403.

References

1. Fabián Z., Induced Cores and Their Use in Robust Parametric Estimation. Commun. Stat. Theory Methods 30, 2001, 537–556
2. Fabián Z., Johnson Score and Characteristics of Distributions. Statist. and Prob. Letters, 2006, (submitted)
3. Fabián Z., Relationship between Probability Measure and Metric in the Sample Space. ITAT'2001, PF Košice, 2001, 103–109
4. Fabián Z., Information Contained in an Observed Value. ITAT'2002, PF Košice, 2002, 95–102
5. Fabián Z., Core Functions and Statistical Inference. ITAT'2003, PF Košice, 2003, 11–20
6. Johnson N.L., Systems of Frequency Curves Generated by Methods of Translations. Biometrika, 36, 1949, 149–176
7. Antoch J., Vorlíčková D., Vybrané metody statistické analýzy dat. Academia Praha, 1992

Centralized broadcasting in radio networks with k-degenerate reachability graphs

František Galčík* and Gabriel Semanišin**

Institute of Computer Science, P.J. Šafárik University, Faculty of Science,
Jesenná 5, 041 54 Košice, Slovak Republic,
frantisek.galcik@upjs.sk, gabriel.semanisin@upjs.sk

Abstract. *We consider deterministic radio broadcasting in radio networks whose nodes have full topological information about network and the reachability graph of a network is k-degenerate. The goal is to design a polynomial algorithm which produces a fast radio broadcast schedule with respect to a reachability graph G and a source $s \in V(G)$. The length of produced schedule is considered as the measure of efficiency. For each k, $k \geq 2$, we show that there are k-degenerate graphs with n nodes for which every radio broadcast schedule has the length $\Omega(\log n)$. Finally, we design an algorithm producing a radio broadcast schedule of the length $O_k(D \log n/D)$ for each k-degenerate graph with n nodes and the eccentricity D of a source.*

1 Introduction

A *radio network* is a collection of autonomous stations that are referred as *nodes*. The nodes communicate via sending messages. Each node is able to receive and transmit messages, but it can transmit messages only to nodes, which are located within its transmission range. The network can be modeled by a directed graph called *reachability graph* $G = (V, E)$. The vertex set of G consists of the nodes of the network and two vertices $u, v \in V$ are connected by an edge $e = (u, v)$ if and only if the transmission of the node u can reach the node v. In such a case the node u is called a *neighbour* of the node v. If the transmission power of all nodes is the same, then the reachability graph is symmetric, i.e. a symmetric radio network can be modeled by an undirected graph.

Nodes of a network work in synchronised steps (time slots) called *rounds*. In every round, a node can act either as a *receiver* or as a *transmitter*. A node u acting as transmitter sends a message, which can be potentially received by each its neighbour. In the given round, a node, acting as a receiver, receives a message only if it has exactly one transmitting neighbour. The received message is the same as the message transmitted by the transmitting neighbour. If in the given round, a node u has at least two transmitting neighbours we say that a *collision* occurs at node u. In the case, when the nodes can distinguish collision from silence, we say that they have an *availability of collision detection*. It is also assumed that a node can determine its behavior in the following round within the actual round.

The goal of *broadcasting* is to distribute a message from one distinguished node, called a *source*, to all other nodes. Remote nodes of the network are informed via intermediate nodes. A *radio broadcast schedule* for a given network prescribes in which step which nodes transmit. Its length corresponds to the time required to complete the broadcast operation, i.e. to inform all nodes of the network. The time, required to complete an operation, is important and widely studied parameter of mostly every communication task. In this paper we consider the length of a broadcast schedule as a function of two parameters of radio network: number of nodes (denoted as n), and the largest distance from the source to any other node of the network (denoted as D).

According to different features of the stations forming a radio network, many models of radio networks have been developed and studied. They differ in used communications scenarios and initial knowledge assumed for nodes. The overview of the models of radio networks can be found e.g. in [13]. In this paper we focus on the deterministic broadcasting in radio networks whose nodes have full topological knowledge about the network. Each node has as an initial knowledge: its unique integer identifier, an identifier of the source node and a labeled copy of underlying reachability graph. Using the same algorithm to produce a broadcast schedule in each node with the same inputs (identifier of the source and underlying reachability graph), the obtained broadcast schedule can be considered as a broadcasting controlled from one central. Thus broadcasting can be performed by nodes according to produced broadcast schedule in a distributed way. The mentioned setting is refereed as *centralized radio broadcasting* or *broadcasting in known topology radio networks*. In this setting, the goal is to design

* Research of the author is supported in part by Slovak VEGA grant number 1/3129/06 and UPJŠ VVGS grant number 38/2006.

** Research supported in part by Slovak APVT grant number 20-004104 and Slovak VEGA grant number 1/3129/06.

a deterministic polynomial algorithm which produces a fast broadcast schedule for a given input.

In this paper we consider only the radio networks, whose underlying reachability graph has a special topology, namely it is k-degenerate for a fixed positive integer k.

1.1 Related work

The study of communication in known topology radio network was initiated in the context of broadcasting problem. In [8] the authors presented a deterministic polynomial algorithm producing a broadcast schedule of the length $O(D\log^2 n)$, for any graph with n nodes and diameter D. In [1] the lower bound $\Omega(\log^2 n)$ of the length of broadcasting schedule was proved for a family of graphs with diameter 2. Later in [5] the authors proposed a method improving the time of broadcasting in the case when reachability graph is undirected. The method is based on partitioning of the underlying reachability graph into clusters with smaller diameter and applying broadcast schedules produced by known algorithms in each cluster separately. This method was later improved in [4]. Applying the deterministic algorithm from [10], which produces a broadcast schedule of the length $O(D\log n+\log^2 n)$, method from [4] computes a broadcast schedule of the length $O(D+\log^4 n)$. Very recently these results have been further improved for undirected graphs in [7] to $O(D+\log^3 n)$. Finally in [11], the authors proposed an algorithm producing a radio broadcast schedule of the asymptotically optimal length $O(D+\log^2 n)$.

In the case when underlying reachability graph is undirected and planar, centralized broadcasting was firstly investigated in [4] where an algorithm producing schedules of the length $O(D+\log^4 n)$ was proposed. Recently, independently in [7] and [6], the algorithms producing a schedule of the length $3D$ were designed. It is remarked in [6] that a broadcast schedule of the length $3D$ can be produced also in the case of directed and planar reachability graph. This sharp gap between the time of broadcasting in general case and in the case of planar graphs was our main motivation to study centralized radio broadcasting in networks whose underlying reachability graph is k-degenerate (note that every planar graph is 5-degenerate).

In this paper we use also the notion of selective families, which has been introduced in [9]. More precisely, we use ad-hoc selective families whose computing has been studied in [3].

1.2 Terminology and preliminaries

Firstly we recapitulate some concepts of graph theory. We assume the standard graph terminology. Next we formulate the broadcasting problem more formally and we show its relationship to selective families. Finally we describe a class of k-degenerate graphs.

The set $N(u) = \{v \in V(G) : (v,u) \in E(G)\}$ we shall denote as *neighbourhood* of a node u. The *distance* of two nodes u, v (denoted by $dist(u,v)$) is the length of a shortest $u-v$-path in the underlying reachability graph. The *eccentricity* of a node v is defined as $ecc(v) = \max\{dist(v,u) : u \in V(G)\}$. We briefly denote the eccentricity of the source node s by $D = ecc(s)$. It is not difficult to see that all nodes of a reachability graph G can be partitioned into *layers* with respect to their distances from the source s. Hence, we can define the sets

$$L_i = \{v \in V(G) : dist(s,v) = i\},\ i = 0, 1, \ldots, ecc(s).$$

Definition 1. *Let $G = (V,E)$ be a directed graph and $R \subseteq V$ be a subset of nodes. The set of nodes informed by R, denoted by $I(R)$, is the set*

$$I(R) = \{v \in V : \textit{there exists the unique } x \in R \textit{ such that } v \in N(x)\}.$$

For a singleton set $R = \{x\}$, $I(R) = I(\{x\}) = N(x)$.

Definition 2. *Let $G = (V,E)$ be a directed graph. A sequence of sets $\Pi = (R_1, \ldots, R_q)$ is called a radio broadcast schedule with respect to the reachability graph G and a source $s \in V$ if and only if the following holds:*

1. *$R_i \subseteq V$, for every $i = 1, 2, \ldots, q$;*
2. *$R_1 = \{s\}$;*
3. *$R_{i+1} \subseteq \bigcup_{j=1}^{i} I(R_j)$, for every $i = 1, 2, \ldots, q-1$;*
4. *$V = \bigcup_{j=1}^{q} I(R_j)$.*

The length of the schedule Π is $q = |\Pi|$.

The property 2 of the previous definition guaranties that in the first round only the source transmits a message. From the property 3 it follows that only informed nodes can transmit. Finally, the property 4 implies that all nodes become informed after performing the broadcast schedule, i.e. every node receives a message in at least one round of the schedule. Note that it is supposed that in the graph G there is a path from the source s to any other node of the network.

For a given collection $\mathcal{F}$ of subsets of $[n] = \{1, \ldots, n\}$, a *selective family for $\mathcal{F}$* is a collection $\mathcal{S}$ of subsets of $[n]$ such that for any $F \in \mathcal{F}$ there exists $S \in \mathcal{S}$ such that $|F \cap S| = 1$. The relationship between selective families and broadcasting in radio networks can be expressed in the following way: suppose that the labels of nodes are pairwise distinct integers from the set $[n]$, where n is the number of nodes. Let I be a set of

informed nodes. Assume that there is a subset U of uninformed nodes such that each node $u \in U$ has at least one informed neighbour. Note that if there is at least one uninformed node then such set $U \neq \emptyset$ always exists. We can construct a collection $\mathcal{F}_U = \{N(u) \cap I : u \in U\}$, i.e. for each node the set of labels of its informed neighbours is a member of the collection $\mathcal{F}_{\mathcal{U}}$. Let $\mathcal{S}_U = \{S_1, \ldots, S_m\}$ be a selective family for $\mathcal{F}_U$. Clearly, a schedule of the length m, such that in the round i exactly the nodes with the labels in the set S_i transmit, ensures that all nodes belonging to the set U become informed. Intuitively, the smaller selective family for a given collection $\mathcal{F}$ we are able to compute, the shorter radio broadcast schedule we are able to produce.

In this paper we shall use the result from [3]:

Theorem 1. *[3] There exists an algorithm that for a given collection $\mathcal{F}$ of subsets of $[n]$, each of size in the range $[\Delta_{min}, \Delta_{max}]$, computes a selective family $\mathcal{S}$ for $\mathcal{F}$ of size $O((1 + \log(\Delta_{max}/\Delta_{min})).\log|\mathcal{F}|)$. The time complexity of the algorithm is*

$$O(n^2|\mathcal{F}|\log|\mathcal{F}|.(1 + \log(\Delta_{max}/\Delta_{min}))).$$

Now we define and describe a class of k-degenerate graphs.

Definition 3. *Let k be a non-negative integer. A graph G is called k-degenerate (we write $G \in \mathcal{D}_k$, if for each subgraph H of G, the minimum degree of H does not exceed k.*

The following value plays the fundamental role in the theory of k-degenerate graphs:

$$s(G) = \max_{H \subseteq G} \min_{v \in V(H)} \deg_H(v).$$

This number is called *Szekeres-Wilf number* and it is easy to see that G is k-degenerate if and only if $s(G) \leq k$. The definition implies that each subgraph of k-degenerate graph is k-degenerate as well (for more details see [2]) and moreover for each graph G there is a number k such that G is k-degenerate.

Proposition 1. *[12] A graph G of order $k + m$ is k-degenerate if and only if the vertex set $V(G)$ can be labeled $v_1, v_2, \ldots, v_{k+m}$ such that in the subgraph $\langle\{v_i, v_{i+1}, \ldots, v_{k+m}\}\rangle$ of G $\deg(v_i) \leq k$ for each $i = 1, 2, \ldots, m-1$.*

Note that the labeling of k-degenerate graph G satisfying the previous proposition can be computed in such a way, that in every step we take out one node of the lowest degree. Obviously this computation takes polynomial time. Also note that k-degenerate graphs have no general bound on the maximal degree of a node. On the other side it was shown in [12], that the number of edges of a k-degenerate graph is at most $kn - \binom{k+1}{2}$ where n is the number of nodes.

2 Lower bound

In this section we show a lower bound concerning the time of broadcasting in radio networks, whose reachability graph is k-degenerate for $k \geq 2$. In particular, we show that there is a subclass of 2-degenerate graphs, such that for each graph of this subclass every radio broadcast schedule has the length $\Omega(\log n)$.

At first we define a set of graphs $\mathcal{G} = \{G_m : m \geq 2\}$. For a fixed integer m, $m \geq 2$, the graph G_m is constructed from the graph K_m with vertex set $V(K_m) = \{v_1, \ldots, v_m\}$ (the complete graph on m vertices) as follows: we add a new node s to K_m and we join it to every node of K_m. Next we subdivide every edge $e_{i,j} = (v_i, v_j) \in E(K_m)$ by a new node $u_{i,j}$. Formally, $G_m = (V_m, E_m)$ is an undirected graph with the vertex set $V_m = \{s, v_1, \ldots, v_m\} \cup \{u_{i,j} : 1 \leq i < j \leq m\}$ and the edge set $E_m = \{(s, v_i) : 1 \leq i \leq m\} \cup \{(v_i, u_{i,j}), (v_j, u_{i,j}) : 1 \leq i < j \leq m\}$.

With respect to the source node s, the graph G_m can be partitioned into layers $L_0 = \{s\}$, $L_1 = \{v_i : 1 \leq i \leq m\}$ and $L_2 = \{u_{i,j} : 1 \leq i < j \leq m\}$. Each layer forms an independent set. Obviously, the radius of G_m is 2. Since every node, except the source s, has degree at most 2, the graph G_m is a 2-degenerate graph with $(m^2 + m + 2)/2$ nodes.

In the following lemma we show that it is not possible to complete radio broadcasting in the graph G_m with the source s in less than $\lfloor \log n \rfloor + 1$ rounds.

Lemma 1. *Any radio broadcast schedule for graph G_m with respect to the source $s \in V(G_m)$ has the length at least $\lfloor \log m \rfloor + 1$.*

Proof. We fix a radio broadcast schedule $\Pi = (R_1, \ldots, R_q)$. Since $R_1 = \{s\}$, only the source s transmits in the first round. This transmission informs all nodes belonging to the layer L_1. Hence the rest of the schedule informs only the nodes of the layer $L_2 = \{u_{i,j} : 1 \leq i < j \leq m\}$ by the transmissions of nodes of the layer L_1. According to the schedule Π we can associate a binary sequence $s_i = (s_i^1, \ldots, s_i^{q-1})$ of length $q-1$ with each node $v_i \in L_1$. We set s_i^r to 1 if and only if $v_i \in R_{r+1}$. Otherwise we set s_i^r to 0. It is easy to see that a node $u_{i,j} \in L_2$ receives a message exactly in each round r such that $s_i^{r-1} \neq s_j^{r-1}$. Since Π is a radio broadcast schedule, every node $u_{i,j} \in L_2$ is informed and it receives a message in at least one round. Thus for each i, j, $i \neq j$, the binary sequences s_i and s_j should differ in at least one position, i.e. $s_i \neq s_j$. It implies that there are exactly $m = |L_1|$ different sequences associated with nodes of the layer L_1.

Clearly, we can construct at most 2^{q-1} different binary sequences of the length $q-1$. Suppose now that $q - 1 < \lfloor \log m \rfloor$. It implies that $2^{q-1} < 2^{\lfloor \log m \rfloor} \leq 2^{\log m} = m$, i.e. $2^{q-1} < m$. The inequality contradicts the fact that we have m different binary sequences of the length $q-1$. □

Theorem 2. *There is a subclass $\mathcal{C}$ of 2-degenerate graphs with radius 2 such that:*

1. *for every integer n, $n \geq 9$, there is a graph $G \in \mathcal{C}$ such that $|V(G)| = n$;*
2. *for every graph $G \in \mathcal{C}$ there is a node $s \in V(G)$ such that every radio broadcast schedule with respect to the graph G and the source s has the length $\Omega(\log n)$, where $n = |V(G)|$ is the number of nodes.*

Proof. Fix an arbitrary real number $c \in (0, \frac{1}{4})$. For each $n \geq 9$, we show that there are 2-degenerate graphs on n nodes with the radius 2, for which every radio broadcast schedule has the length at least $\lfloor c \log n \rfloor$ rounds. It is easy to see that the chosen c, n and $m = \lceil n^c \rceil$ satisfy the inequality $n - (m^2+m+2)/2 \geq 0$.

Let $G = (V, E)$ be a graph with n nodes constructed from the graph G_m, where $m = \lceil n^c \rceil$, by adding $n - (m^2 + m + 2)/2$ new nodes and joining them to the node $s \in V(G_m)$. Then G has the vertex set $V(G) = V(G_m) \cup \{w_1, \ldots, w_{n-(m^2+m+2)/2}\}$ and the edge set $E(G) = E(G_m) \cup \{(s, w_i) : 1 \leq i \leq n - (m^2 + m + 2)/2\}$ and obviously graph G is 2-degenerate graph with radius 2. Let $s \in V(G)$ be the source. Since there are no edges between $V(G_m) \setminus \{s\}$ and $(V(G) \setminus V(G_m)) \setminus \{s\}$, the broadcast operation is performed in the subgraph G_m separately. Previous lemma implies that it is not possible to complete broadcasting in G_m (and also in G) in less than $\lfloor \log m \rfloor + 1 \geq \log n^c \geq \lfloor c \log n \rfloor$ rounds. □

Note that, in the previous proof there are more ways how to construct a graph G satisfying desired properties. In more general construction we add $n - (m^2 + m + 2)/2$ new nodes to the graph G_m. Next we add new edges to the graph G between the nodes of the set $W = (V(G) \setminus V(G_m)) \cup \{s\}$ (i.e. between newly created nodes and the source s) in such a way that the induced graph $H = \langle W \rangle_G$ is connected 2-degenerate graph satisfying $ecc_H(s) \leq 2$.

Since $\mathcal{D}_2 \subset \mathcal{D}_k \subset \mathcal{D}_{k+1}$ for each $k > 2$ (see [12]), the following holds:

Corollary 1. *Let k be a positive integer, $k \geq 2$. There is a subclass $\mathcal{C}$ of k-degenerate graphs with radius 2 such that:*

1. *for every integer n, $n \geq 9$, there is a graph $G \in \mathcal{C}$ such that $|V(G)| = n$;*
2. *for every graph $G \in \mathcal{C}$ there is a node $s \in V(G)$ such that every radio broadcast schedule for G with respect to the source s has the length $\Omega(\log n)$, where $n = |V(G)|$ is the number of nodes.*

3 Upper bound

In this section we focus on the upper bound of the length of a radio broadcast schedule. We present algorithms producing radio broadcast schedules for k-degenerate input reachability graph.

Consider a class of 1-degenerate graphs (remark that every connected 1-degenerate graph is a tree). In such a case we can construct a trivial radio broadcast schedule $\Pi = (R_1, \ldots, R_q)$ with respect to a graph (tree) G and a source $s \in V(G)$ as follows:

1. $R_1 := \{s\}$;
2. $R_{i+1} := I(R_i) \setminus \bigcup_{j=1}^{i-1} I(R_j)$, for $i \geq 1$.

The condition 2 yields that in the round $i+1$ a message is transmitted by all nodes which receive a message in the round i for the first time. It is easy to see that there is a round p such that $R_p = \emptyset$. Letting $q := p - 1$ one can prove that Π is a radio broadcast schedule of the optimal length.

In what follows we present algorithms producing a radio broadcast schedule for graphs which belong to $\mathcal{D}_k$ for a fixed integer k, $k \geq 2$.

Theorem 3. *Let $G = (A \cup B, E) \in \mathcal{D}_k$ be a bipartite k-degenerate graph ($k \geq 2$) such that $deg_G(v) \geq 1$ for all $v \in B$. Suppose that all nodes of the partition A are informed. There is a polynomial algorithm producing a schedule of the length at most $\lceil k^2/2 \rceil + k + O((1 + \log k) \log |B|)$ such that*

- *only the nodes of A transmit*
- *it ensures that all nodes of B become informed, i.e. every node of the partition B receives a message in at least one round*

Proof. The algorithm works in two phases. During each phase a part of the resulting schedule is produced. The goal of each part is to inform all nodes in the specific subset of B.

Phase 1: Let G be an input graph and denote $n = |V(G)|$. Since G is a k-degenerate graph, according to Proposition 1, in the polynomial time we can compute labeling $v_1, v_2, \ldots, v_n$ of the nodes of G such that for each $i = 1, 2, \ldots, n$ in the induced subgraph $G_i = \langle \{v_i, v_{i+1}, \ldots, v_n\} \rangle_G$ it holds $\deg_{G_i}(v_i) \leq k$. It means that the nodes of G can be ordered in such a way that there are at most k edges from the node v_i to the nodes of set $\{v_{i+1}, \ldots, v_n\}$.

For each i, $1 \leq i \leq n$, we define a set:

$$N_{deg}(v_i) = \{v_j \in V(G) : (v_i, v_j) \in E(G) \wedge j > i\}$$

Note that $N_{deg}(v_i) \subseteq N(v_i)$ and $|N_{deg}(v_i)| \leq k$ for each $v_i \in V(G)$. The goal of this computation phase is to produce a schedule, which ensures that each node $v_i \in B$, such that $|N(v_i) \setminus N_{deg}(v_i)| \geq 1$, becomes informed. During the computation every node $v_i \in V(G)$ has assigned one round, denoted as $round(v_i)$, such that $round(v_i) \in R \cup \{NIL\}$ where $R = \{1, \ldots, \lceil k^2/2 \rceil + k\}$. Symbol NIL is used for still undefined round. For a node $v_i \in A$, the value $round(v_i)$ denotes

a round in which the node v_i will transmit a message during the first part of schedule. For a node $v_i \in B$, it denotes a round (from the other admissible) in which the node v_i receives a message. Initially, we set $round(v_i) := NIL$ for all $v_i \in V(G)$. For each node $v_m \in B$ we shall maintain the following set during the computation:

$$Receive(v_m) = \{r : \text{there exists the unique } v_j \in N(v_m) \text{ such that } round(v_j) = r \neq NIL\}$$

The nodes are processed in the sequential order from v_n to v_1. After a node v_i is processed the following two invariants hold:

1. for each $v_j \in A$ such that $j \geq i$ we have:
   - $round(v_j) \neq NIL$
   - $round(v_m) \neq NIL$, for all $v_m \in N_{deg}(v_j)$.
2. for each $v_j \in B$ such that $j \geq i$ it holds:
   - $Receive(v_j) = \emptyset \Rightarrow round(v_j) = NIL$
   - $Receive(v_j) \neq \emptyset \Rightarrow round(v_j) \in Receive(v_j)$

Now we show, how a node $v_i \in V(G)$ is processed. We fix $v_i \in V(G)$ and suppose that all nodes in the set $\{v_{i+1}, \ldots, v_n\}$ have been already processed.

In case that $v_i \in B$ we process the node v_i as follows. If $Receive(v_i) \neq \emptyset$ then we set $round(v_i)$ to an arbitrary element of the set $Receive(v_i)$ Otherwise, $round(v_i)$ is unchanged, i.e. $round(v_i) = NIL$.

In case that $v_i \in A$ the processing of the node is more complex. For each $v_m \in B$ we compute the set:

$$Used(v_m) = \{round(v) : v \in N_{deg}(v_m)\}$$

Next we compute the following sets:

$$Unassigned = \{v_j \in N_{deg}(v_i) : round(v_j) = NIL\}$$
$$Assigned = N_{deg}(v_i) \setminus Unassigned$$
$$Used^* = \bigcup_{v_j \in Unassigned} Used(v_j)$$
$$Used = \{round(v_j) : v_j \in Assigned\} \cup Used^*$$

Finally, we set the value $round(v_i)$ to an arbitrary element of the set $R \setminus Used$. Afterwards we set the value $round(v_j) := round(v_i)$, for all $v_j \in Unassigned$.

The first part of the schedule, which corresponds to this computation phase, is produced as follows: a node $v_i \in A$ transmits a message exactly in the round $round(v_i)$, i.e. $R_j = \{v_i \in A : round(v_i) = j\}$ for each $j \in R$.

Correctness of phase 1: We use computational induction to show that during the first phase of the algorithm both mentioned invariants hold after processing of a node.

It is not difficult to see that the claim is true after processing of a node $v_i \in B$. In the case when $v_i \in A$, we show firstly the correctness of the assignment, i.e. that $R \setminus Used \neq \emptyset$ and we are able to choose a round-number.

Let $v_m \in Unassigned$. From the definition of the set $Unassigned$ it follows that $v_m \in N_{deg}(v_i) \subseteq B$, $m > i$ and $round(v_m) = NIL$. Since $m > i$ the second invariant implies that $Receive(v_m) = \emptyset$. Thus each round-number in the set $Used(v_m)$ is assigned to at least two nodes from the set $N_{deg}(v_m)$. Otherwise, there is a node $u \in N_{deg}(v_m)$ such the value $round(u) \neq NIL$ and moreover there is no node $w \in N_{deg}(v_m)$ such that $round(u) = round(w)$. But it contradicts to $Receive(v_m) = \emptyset$. Since $|N_{deg}(v_m)| \leq k$ and each round-number is used at least twice, it holds that $|Used(v_m)| \leq k/2$. Again using $|N_{deg}(v_i)| \leq k$ we obtain:

$$|Used| \leq |Assigned| + \frac{k}{2}|Unassigned| \leq \frac{k^2}{2} + k - 1.$$

Hence there is at least one free round-number and $R - Used \neq \emptyset$, i.e. the defined assignment is correct.

One can verify that according to the assignments which are created at the moment when the round-number $round(v_i)$ is determined, the first invariant holds.

Now we shall analyse the second invariant. Since during the processing of the node $v_i \in A$ we change only one round-number in the set A, it is sufficient to consider validity of conditions of the second invariant only for nodes of the set $N_{deg}(v_i) \subseteq B$. Since for each node $v_m \in Assigned$ it holds that $round(v_m) \notin R \setminus Used$, validity of the conditions remains unchanged for the nodes of the set $Assigned$. Consider a node $v_m \in Unassigned$. Since after processing of v_i it holds that $NIL \neq round(v_i) = round(v_m) \notin Used$ and $round(v_m) \notin Used \Rightarrow round(v_m) \notin Used(v_m) \subseteq Used$. Since $v_m \in Unassigned$, the first invariant implies that before processing of v_i there is no node $v_j \in N(v_m) \setminus N(v_m) \subseteq A$ such that $round(v_j) \neq NIL$. Both this facts we can summarise into the claim: There is no $v_j \in N_{deg}(v_m)$ such that $round(v_j) = round(v_m)$ and $round(v_j) = NIL$, for all $v_j \in (N(v_m) \setminus N_{deg}(v_m)) \setminus \{v_i\}$. This claim implies that $Receive(v_m) \neq \emptyset$ and $round(v_m) \in Receive(v_m)$.

Since after processing of all nodes both invariants hold, we show that each node $v_m \in B$, which satisfies $N(v_m) \setminus N_{deg}(v_m) \neq \emptyset$, become informed after execution of the first part of schedule. Let $v_m \in B$ be a node such that $N(v_m) \setminus N_{deg}(v_m) \neq \emptyset$ and let $v_i \in A$ be a node such that $v_i \in N(v_m) \setminus N_{deg}(v_m)$. The definition of $N_{deg}(v_m)$ implies that $i < m$ and thus $v_m \in N_{deg}(v_i)$. Finally, the first invariant guaranties that $round(v_m) \neq NIL$. Hence the second invariant implies that $round(v_m) \in Receive(v_m)$, i.e. v_m receives a message in at least the round $round(v_m)$.

It follows that only the nodes $v_i \in B$, for which $N(v_i) = N_{deg}(v_i)$, can be uninformed. For such uniformed nodes it holds $|N(v_i)| = |N_{deg}(v_i)| \leq k$.

Phase 2: The goal of this phase is to inform all remaining uninformed nodes. Since for every uninformed node $v_i \in B$ it holds that $|N(v_i)| \leq k$, we can use the algorithm from Theorem 1 to produce the second part of the schedule. Using the input collection $\mathcal{F} = \{\{j : v_j \in N(v_i)\} : v_i \in B \text{ is uninformed}\}$, the algorithm produces a collection $\mathcal{S} = \{S_1, \ldots, S_p\}$ as an output, where $p = O((1 + \log k) \log |B|)$. The second part of schedule is constructed using the following definition $R_{j+|R|} = \{v_i \in A : i \in S_j\}$ for each $j, 1 \leq j \leq p$, where R is the set of the size $\lceil k^2/2 \rceil + k$ which has been defined in the Phase 1. Correctness of the produced schedule follows from Theorem 1.

Complexity: The total length of produced schedule is $\lceil k^2/2 \rceil + k + O((1 + \log k) \log |B|)$ rounds. Since both phases take polynomial time, the designed algorithm is polynomial as well. □

Theorem 4. *Let $G = (V, E)$ be a directed connected k-degenerate graph ($k \geq 2$), i.e. $G \in \mathcal{D}_k$. Then there exists a polynomial algorithm producing a radio broadcast schedule of the length $D.(\lceil k^2/2 \rceil + k + O((1 + \log k) \log \frac{n}{D}))$.*

Proof. Since each subgraph of a k-degenerate graph is k-degenerate too, we use the algorithm from the proof of Theorem 3 to inform the nodes of every consecutive layer, i.e. broadcasting is scheduled layer by layer. The length of the schedule follows from the fact that $\sum_{i=1}^{D} \log |L_i|$ obtains maximal value for $|L_i| = n/D$. □

Since k is fixed constant, the previous theorem implies the following corollary:

Corollary 2. *Let $k \geq 2$ be an integer and D_k be a class of k-degenerate graphs. The there is a polynomial deterministic algorithm which produces a radio broadcast schedule of length $O_k(D \log n/D)$ with respect to a reachability graph $G \in \mathcal{D}_k$ and a source $s \in V(G)$.*

It is not difficult to see that for k-degenerate graphs with diameter $o(\log n)$ proposed algorithm produces radio broadcast schedules of shorter length than known algorithms for general case.

4 Conclusion

For a fixed positive integer k, we focused on deterministic centralized radio broadcasting in radio networks whose reachability graphs are k-degenerate. In view of the presented lower bound, the proposed algorithm produces asymptotically optimal radio broadcast schedules for k-degenerate graphs of constant diameter. The lower bound shows that planar graphs differ from 5-degenerate graphs from the viewpoint of broadcasting problem. The algorithm also gives a partial approximation to open problem (stated in [11]) whether there is a polynomial algorithm producing a broadcast schedule of the length $O(opt(G). \log n)$, where $opt(G)$ is the length of the shortest broadcast scheme on G. However the problem, whether it is possible to use an assumption about bounded average degree of arbitrary reachability graph in order to design algorithm producing shorter radio broadcast schedules, remains open.

References

1. N. Alon, A. Bar-Noy, N. Linial and D. Peleg, A Lower Bound for Radio Broadcasting. Journal of Computer and System Sciences 43, 1991, 290–298
2. M. Borowiecki, I. Broere, M. Frick, P. Mihók and G. Semanišin, Survey of Hereditary Properties of Graphs. Discuss. Math. Graph Theory 17, 1997, 5–50
3. A.E.F. Clementi, P. Crescenzi, A. Monti, P. Penna and R. Silvestri, On Computing Ad-Hoc Selective Families. 5th Int. Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM'01), LNCS, 2001, 211–222
4. M. Elkin and G. Kortsarz, Improved Schedule for Radio Broadcast. Proc. 16th ACM-SIAM Symposium on discrete Algorithms (SODA'2005), 2005, 222–231
5. I. Gaber and Y. Mansour, Centralized Broadcast in Multihop Radio Networks. Journal of Algorithms 46 (1), 2003, 1–20
6. F. Galčík, Complexity Aspects of Radio Networks. Master Thesis, Commenius University Bratislava, 2005
7. L. Gasieniec, D. Peleg and Q. Xin, Faster Communication in Known Topology Radio Networks. Proc. 24th Annual ACM Symposium on Principles of Distributed Computing (PODC'2005), 2005, 129–137
8. I. Chlamtac and S. Kutten, The Wave Expansion Approach to Broadcasting in Multihop Radio Networks. IEEE Transactions on Communications 39, 1991, 426–433
9. B. S. Chlebus, L. Gasieniec, A. Gibbons, A. Pelc and W. Rytter, Deterministic Broadcasting in Unknown Radio Networks. In Proceedings of 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'00), 2000, 861–870
10. D. Kowalski and A. Pelc, Centralized Deterministic Broadcasting in Undirected Multi-Hop Radio Networks. Proc. APPROX, LNCS 3122, 2004, 171–182
11. D. Kowalski and A. Pelc, Optimal Deterministic Broadcasting in Known Topology Radio Networks. To apper.
12. D.R.Lick and A.R.White, k-Degenerate Graphs. Canad. J. Math 22, 1970, 1082–1096
13. A. Pelc, Broadcasting in Radio Networks. Handbook of Wireless Networks and Mobile Computing, I. Stojmenovic (ed.) John Wiley and Sons, Inc., New York, 2002, 509–528

Modelovanie a spúšťanie aktivít na základe kontextu administratívneho procesu

Emil Gatial[1], Martin Šeleng[1], Igor Mokriš[2], and Radoslav Forgáč[2]

[1] Ústav informatiky, Slovenská akadémia vied
Dubravska cesta 9, 845 07 Bratislava, Slovensko
emil.gatial@savba.sk
[2] Centrum Simulačných Technológií, Národná akadémia Obrany maršala Andreja Hadika
Demänovská cesta 393, 031 01 Liptovský Mikuláš, Slovensko

Abstrakt *V tomto článku navrhujeme systém pre riadenie administratívnych procesov. Kľúčová myšlinka je založená na tom, že každá aktivita v rámci administratívneho procesu môže byť spustená iba ak spĺňa dopredu definované podmienky. Systém počas behu administratívneho procesu monitoruje a zbiera informácie o dokumentoch, účastníkoch a už ukončených aktivitách. Na základe toho sa mení kontext určitého používateľa a kontext pracovného procesu. Pokiaľ sú splnené podmienky pre vyvolanie nejakej aktivity, tak systém zmení stav kontextu pracovného procesu a kontextu uživateľa a informuje ho o novo spustenej aktivite. Každá aktivita má zároveň stanovený čas najneskoršieho ukončenia, ktorý je monitorovaný a sú podávané informácie o kritických a prioritných aktivitách, tak aby bol administratívny proces včas ukončený. Na zápis administratívneho procesu je použitý ontologický jazyk, ktorým sú popísane ontológie pre užívateľa, použíté dokumenty a administratívny process. Zároveň je oddelená generická úroveň ontológie pre zápis ľubovoľného pracovného procesu a špecifická úroveň ontológie, kde je popisaná štruktúra organizácie a pracovného procesu. Jednotlivé časti článku sa zaoberajú modelovaním administratívneho procesu, spôsobom spúšťania konkrétnych aktivít na základe kontextu pracovného procesu s použitím ontologického zápisu. Tento systém je navrhovaný a implementovaný v rámci projektu RAPORT.*

1 Úvod

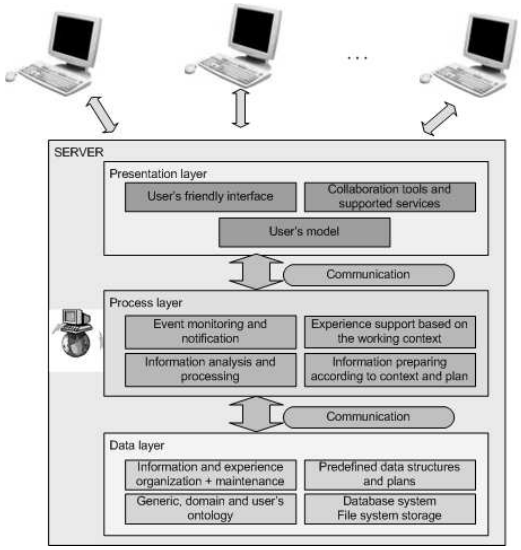
V súčasnej dobe, systémy riadiace tok práce (workflow management system - WfMS) sú používané na harmonizáciu a zvýšenie efektívnosti administratívnych procesov [1]. Obľúbená definícia termínu tok práce (workflow): Workflow doručí správne dáta správnym ľuďom pomocou správnych nástrojov v správnom čase [2]. Vďaka tomu workflow systémy sa javia ako ideálne riešenia pre automatizáciu denno dennej práce pomocou poskytnutia správnych materiálov ľuďom, ktorí ich potrebujú. Na dosiahnutie tohto cieľa používame kontext používateľa a kontext pracovného procesu. Tento článok sa snaží predstaviť pohľad na spúšťací mechanizmus pracovných aktivít založených na porovnaní požiadaviek na spustenie pracovnej aktivity vyjadrenej ako kontext aktivity a kontextu pracovného procesu. Navrhovaný systém využíva ontológie na popis pracovných prostriedkov, ktoré poskytujú zdieľané spoločné chápanie konceptov toku práce. Tento systém je testovaný na aplikácii prípravy vojenských cvičení v rámci projektu RAPORT.

2 Architektúra systému RAPORT

Architektúra je navrhnutá všeobecne s ohľadom na použitie v ľubovoľnom pracovnom procese. Takisto berie do úvahy použitie v pilotnej aplikácii administratívnych procesov v CST (Centre Simulačných Technológií). Systémová architektúra vychádza z nasledujúcich požiadaviek:

- zachytávanie skúseností od užívateľov [3] a prezentácia užitočných rád užívateľom, ktorí pracujú na rovnakej alebo podobnej aktivite identifikovanej pomocou podobnosti kontextu používateľa s kontextom, keď bola zachytená skúsenosť, dohliadanie nad práve vykonávanými tréningovými plánmi, ktoré obsahujú dôležité termíny, ktoré je nutné dodržať;
- informovanie užívateľa na o dôležitých termínoch,
- príprava dokumentov užívateľom podľa dôležitých termínov ako napríklad pred pripravené emailové správy, dokumenty a formuláre a informovanie užívateľov o nich,
- podpora výmeny skúseností medzi užívateľmi a podpora spolupráce.

Trojvrstvová architektúra je načrtnutá na obrázku 1, ktorá je tvorená tromi vrstvami: dátová vrstva (*data layer*), procesná vrstva (*process layer*) a prezentačná vrstva (*presentation layer*). Pri spúšťaní pracovných aktivít sú uplatnené algoritmy na spracovanie a analýzu informácií (procesná vrstva), ktoré pracujú nad informáciami uloženými v ontologickom úložisku (dátová vrstva). Komunikácia prebieha prostredníctvom aplikačného rozhrania použitého ontologického nástroja, v prípade projektu RAPORT je použitý nástroj Jena na prístup k ontologickému úložisku.



Obrázok 1. Architektúra systému RAPORT.

3 Ontologický model pracovného procesu

Tvorba ontológie je netriviálny problém, ktorý nielen znalosti v obore informačných technológií, ale aj značné znalosti v modelovaní danej domény. Na zjednodušenie procesu vytvárania ontológií bolo navrhnutých niekoľko metód. Základné princípy vytvárania ontológií sú odvodené od metodológie CommonKADS [4], ktorá sa zaoberá spoločnými princípmi vytvárania znalostných systémov. Pre vytvorenie všeobecného systému bol znalostný systém bol postavený na základe ontologického popisu. Boli navrhnuté dve úrovne ontológie: generická a špecifická. Generická ontológia popisuje základné koncepty použité v administratívnych procesoch ako sú Osoba, Proces alebo Aktivita.Zároveň sú modelované aj vzťahy medzi nimi. Vytváranie špecifickej ontológie vyžaduje analýzu konkrétnych typov procesov použitých pre navrhovanú aplikáciu. Pre pilotnú aplikáciu systému RAPORT bola analyzovaná príprava vojenského cvičenia v CST. Boli rozpracované funkčné a procesné modely na základe ktorých bola navrhnutá ontológia pre prípravu vojenského cvičenia.

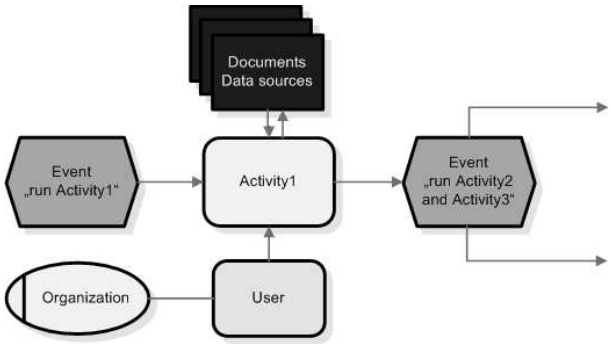
Hlavný ontologický model sa skladá z troch ontologických modelov, ktoré sú postavené na základe generickej ontológie a poskytujú tak všeobecný pohľad na prostriedky nevyhnutné pre modelovanie administratívneho procesu:

- Ontológia toku práce (*Workflow ontology*) je základná časť ontologického modelu, ktorá zahŕňa definíciu typov ako aj inštancií pracovného procesu.
- Ontológia dát (*Data ontology*) obsahuje špecifikácie vstupných a výstupných dokumentov a definíciu metadát, ktoré sú použité v administratívnom procese.
- Ontológia používateľa (*User ontology*) zahŕňa metadáta o účastníkoch administratívneho procesu.

Zmienené ontológie sú rozšírené špecifickou ontológiou, tak aby dostatočne popísali konkrétny typ administratívneho procesu. Na pochopenie navrhovaných princípov spúšťania aktivít v administratívnych procesoch nie je nutná znalosť špecifickej ontológie.

4 Model spúšťania aktivít administratívnych procesov

V nasledujúcej časti je popísaný princíp spúšťania aktivít a navrhovaný ontologický model. Proces je modelovaný použitím aktivít. Každá aktivita je modelovaná pomocou vstupného kontajnera (*input container*), výstupného kontajnera (*output container*) a zainteresovaných osôb. Vstupný (výstupný) kontajner popisuje prostriedky, ktoré vstupujú (vystupujú) do (z) popisovanej aktivity. Tieto prostriedky sú obyčajne odkazy na dokumenty použité alebo spracované v danej aktivite, ktoré sú (alebo budú) uložené v súborovom systéme. Aktivity sú spúšťané na základe udalostí (trieda *Event*) generovaných systémom v prípade, že sú splnené podmienky na spustenie aktivity. Teda aktivity sú spúšťané za behu systému, kde nie je presne určené, ktorá aktivita bude spustená. Spustenie aktivity závisí len od stavu (množiny ukončených aktivít, dostupných dokumentov a dostupnosti požadovaných osôb) pracovného procesu. Udalosti môžu byť použité na vyvolanie histórie spúšťania aktivít alebo potvrdzovanie spúšťania aktivity prostredníctvom osoby dohliadajúcej nad korektným spúšťaním aktivít v administratívnom procese. Obrázok 2 vyjadruje model pracovného procesu. Ontologický zápis abstraktného pra-



Obrázok 2. Model pracovného procesu založený na spúšťaní aktivít pomocou udalostí.

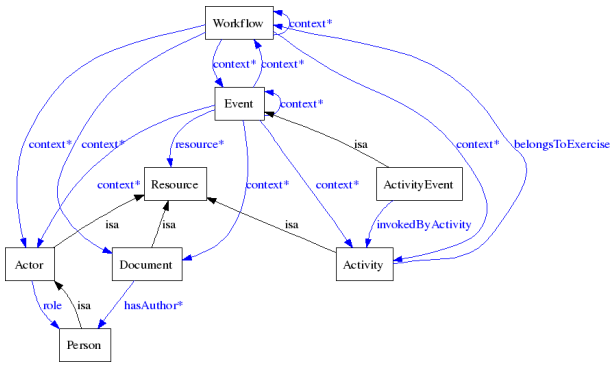
covného procesu je definovaný triedou *AbstractWorkflow*, ktorá obsahuje množinu udalostí (inštancie triedy *AbstractEvent*), ktoré môžu byť spustené v rámci daného pracovného procesu. Trieda *AbstractEvent* je kľúčovou pri spúšťaní konkrétnych aktivít, pretože obsahuje vlastnosť *contextRequirement*, kde je zapísaná podmienka na spustenie konkrétnej aktivity. Spúšťacím mechanizmom je platnosť tejto podmienky v rámci konkrétneho pracovného procesu. Rozhodovací algoritmus je založený na porovnaní vlastnosti *contextRequirement* a kontextu (*context*) triedy *Workflow*. Kde kontext pracovného procesu *context* je množina zahŕňajúca zainteresované osoby (*Actor*), dokumenty (*Document*) ako aj ukončené aktivity (*Activity*). Vlastnosť *context* je definovaná (podľa [5]) ako

$$Workflow.context \in Actor \cup Document \cup Activity \quad (1)$$

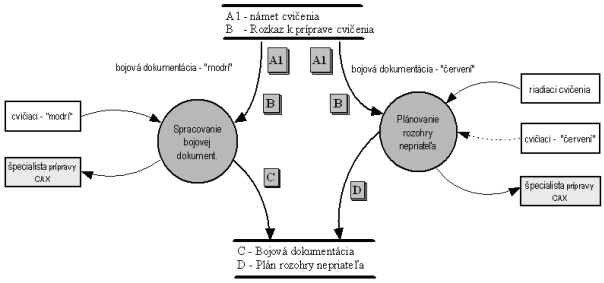
a požiadavky na kontext pre spustenie aktivity contextRequirement

$$AbstractEvent.contextRequirement \in owl : Class \quad (2)$$

Trieda *AbstractEvent* len definuje predpis akým spôsobom sa má vytvárať konkrétna inštancia triedy *Event*. Rozhodovací algoritmus potom porovnáva, či inštancie vlastnosti *context* sú požadovaného typu. To znamená, že je možné stanoviť nutné podmienky, kedy sa má vytvoriť inštancia typu *Event*, ktorá následne spustí požadovanú aktivitu. Na obrázku Obrázok 3. je znázornená časť generickej ontológie spolu s vzájomnými reláciami jednotlivých tried. Predopkladajme, že máme definovaný pracovný proces, v ktorom vstupy sú dokumenty spracovávané v pracovnej aktivite a výstupom je tiež množina dokumentov. Potom ontologicý zápis podmienok na spustenie pracovnej aktvity (pozri obrázok 4) „Spracovanie bojovej dokumentácie“ zahŕňa množinu typov dokumentov vstupujúcich do danej aktivity, teda dokumenty A1 a B.



Obrázok 3. Ontologický model použitý pri spúšťaní aktivít.



Obrázok 4. Schématické znázornenie pracovných aktivít.

$$AbstractEvent.contextRequirement = \{A1, B\} \quad (3)$$

Systém monitoruje novovytvorené dokumenty a vždy vykoná porovnanie aktuálneho kontextu pracovného procesu s požadovaným kontextom na spustenie pracovnej aktivity *AbstractEvent. contextRequirement.* Toto porovnanie prebehne pre všetky definované inštancie triedy *AbstractEvent.* Ak je splnená podmienka

$$AbstractEvent.contextRequirement \subset Workflow.context\,,$$

tak je spustená nová aktivita prostredníctvom vytvorenia inštancie pracovnej aktivity, ktorej trieda je zapísaná v *AbstractEvent.activityClass.* Tento postup je možné uplatniť na ľubovoľné elementy vstupujúce do pracovného procesu, ktoré môžu byť zachytené v systéme ako napríklad osoby, ukončené aktivity, termíny a podobne.

5 Zhrnutie

Definícia podmienok umožňujúcich testovať zaradenie prostriedkov dostupných v rámci pracovného procesu do určitej triedy poskytuje širšie možnosti pri spúšťaní aktivít. Jednou z hlavných výhod je to, že aktivita nemusí byť závislá len na ukončení nejakej inej aktivity, ale aj na dostupnosti dokumentov, prítomnosti zainteresovaných osôb alebo ich kombinácii. Ďalšou výhodou je možnosť pridávania, mazania alebo úpravy podmienok počas behu systému bez toho, aby to narušilo konzistenciu uložených dát.

Tento prístup však kladie zvýšené nároky na údržbu alebo spätnú rekonštrukciu pracovného procesu, pretože aktivity sú prepájané pomocou udalostí.

Možné rozšírenie popisovaného systému spočíva v použití časového rozvrhu pri testovaní kontextu pracovného procesu (vytvorenie triedy *Schedule*), čo by

umožnilo automaticky generovať udalosti, ktoré by mohli mať informatívny alebo varovný charakter pri nedodržaní stanovených termínov.

6 Záver

V tomto článku bol prezentovaný prístup spúšťacieho mechanizmu pracovných aktivít s použitím ontológií. Ontológie tu slúžia nielen ako dátové úložisko, ale zároveň sú využívané črty ako dedičnosť alebo zaradenie inštancií do tried. Popisovaný prístup je vo fáze implementácie v rámci projektu RAPORT, kde je modelovaný a testovaný na aplikácii prípravy vojenského cvičenia v CST Národnej Akadémie Obrany v Liptovskom Mikuláši.

Poďakovanie. Táto práca bola podporená APVT-51-024604 , NAZOU SPVV 1025/2004, EU RTD IST K-Wf Grid FP6-511385 and VEGA No. 2/6103/6.

Referencie

1. Gustavo Alonzo, Divyakant Agraval, Amr El Abbadi and C. Mohan, Functionality and Limitations of Current Workflow Management Systems. IEEE Expert, 1997
2. Heiko Maus, Workflow Context as a Means for Intelligent Information Support. Third International and Interdisciplinary Conference, CONTEXT 2001, Dundee, UK, 2001
3. Balogh Z., Laclavik M., Hluchy L., Budinska I., Krawczyk K., REMARK – Reusable Agent-Based Experience Management and Recommender Framework. In: ICCS'04 / Agent Day 2004; Proc.of International Conference on Computational Science, Part III, LNCS 3038, Springer-Verlag, 2004, 599–606, ISSN 0302-9743, ISBN 3-540-22116-6, June 6-9, Krakow, Poland
4. Schreiber August Th., et al., Knowledge Engineering and Management: The CommonKADS Methodology. ISBN 0-262-19300-0, The MIT Press, 2002
5. F. Baader, W. Nutt, Basic Description Logics. In the Description Logic Handbook, F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, P.F. Patel-Schneider (eds), Cambridge University Press, 2002, 47–100

A natural infinite hierarchy by free-order dependency grammars*

Radu Gramatovici[1] and Martin Plátek[2]

[1] Faculty of Mathematics, University of Bucharest
14, Academiei st., RO-010014, Bucharest, Romania
radu@funinf.cs.unibuc.ro
[2] Faculty of Mathematics and Physics, Charles University,
Malostranské nám. 25, CZ-11800, Prague, Czech Republic
martin.platek@mff.cuni.cz

Abstract. *We obtain a new infinite hierarchy of classes of semilinear languages by a gradual topological relaxation of word order in sentences generated by dependency grammars. This hierarchy starts by context-free languages. From some already known results it follows that the (classes of) languages forming the hierarchy are recognizable in polynomial time and space.*

1 Introduction

The notion of free-order dependency grammar (in the following, simply called *dependency grammar*) was introduced in [6] as a formal system suitable for dependency-based parsing of natural languages. In a certain way this notion enriches the interpretation of a type of dependency grammars described in [1].

The proposal of this system was based upon the experience acquired during the development of a grammar-checker for Czech and as a possible next step towards a complete syntactic analysis following the underlying ideas of the dependency-based framework of Functional Generative Description - FGD ([10]). Compared to FGD and other usual formal systems describing the syntax of natural languages, the framework introduced by dependency grammars takes a serious account for the freedom of word order in a sentence and assigns the same importance to linear precedence (LP) rules as to immediate dominance (ID) rules.

As the freedom of word order is not total, even in so-called *free-word-order languages*, one needs to constrain the formalism in order to not overgenerate the actual language.

In [9], a measure for the freedom of the word order was studied, based on the number of gaps issued in a sentence by the order of their words (*node-gaps-complexity*). Both *global* and *local constraints* on the maximal number of gaps at some node in the structure underlying the sentence were studied. In the view of node-gaps-complexity, word order relaxation means to stepwisely relax the constraints in order to obtain more complex language constructions. In this paper, we work only with global constraints, similarly to [4]. The main results achieved in [4] were incomparability results of classes of languages due to the subset relation for a special type of dependency grammars. Here we present a new hierarchy of naturally composed classes of languages, based on global constraints only. We use similar techniques for mutual separations of this composed classes as in [4] for the simple classes. The hierarchy illustrates the power of the dependency parser [5].

Two types of syntactic structures related to dependency grammars are used: **DR**-*trees* (**D**elete **R**ewrite trees) and **D**-*trees* (**D**ependency trees). If D-trees concern the dependency structure of the sentence, DR-trees rather concern the generation/parsing of the sentence. The two types of structures are related by the fact that any DR-tree can be transformed in an uniform way into a D-tree. The measure for the number of gaps in a sentence computed in the nodes of the structure was originally introduced for both DR-trees and D-trees. In this paper, we work mainly with the node-gap complexity for DR-trees.

In Section 4, we present the main result of this paper, i.e. the infinite hierarchy of classes of semilinear languages generated by dependency grammars. The first class in this hierarchy is the class of context-free languages without empty strings. Further, it follows from the results presented in [6, 9, 7] that the languages from this hierarchy can be recognized in polynomial time.

2 DR-trees and D-trees

Let M be a set. Let $Tr = (Nod, Ed, Rt, Ann)$ be a 4-tuple, where Nod is a set (*the set of nodes*), $Rt \in Nod$ is a special node (*the root*), $Ed : Nod \setminus \{Rt\} \to Nod$ is a function (*the set of edges*) and $Ann : Nod \to M$ is a function (*the annotation function*). We call **path** in Tr any sequence of nodes from Nod, $p = (n_1, n_2, \ldots, n_k)$, with $k \geq 1$, such that $Ed(n_i) = n_{i+1}$, for $i = 1, \ldots k-1$. We say that p is a *path of length* $k-1$

* The first author was supported by the grant ET75/2005 of the Romanian Ministry of Education and Research. The second author was supported by the program 'Information Society' under project 1ET100300517.

from n_1 *to* n_k. If $k > 1$, we denote p by $Path(n_1, n_k)$. We say that *Tr* is a M**-annotated tree** iff the graph (Nod, Ed) creates a tree such that from any of its nodes leads a path to *Rt*. Let $n \in Nod$ be a node in *Tr*. We say that n is a **leaf** iff $n \neq Ed(n')$, for any $n' \in Nod$.

We say that *Tr* is a **finite** M-annotated tree iff its set of nodes, Nod, is finite. In the sequel of this paper, we will work only with finite annotated trees, without clearly mention it.

If *Tr = (Nod, Ed, Rt, Ann)* is an annotated tree, and if $n_1, n_2 \in Nod$ are two nodes in Tr then there exists at least one node $n_3 \in Nod$ such that there exist a path from n_1 to n_3 and a path from n_2 to n_3. This last remark allows us that for any two nodes if $n_1, n_2 \in Nod$ to denote by $sup(n_1, n_2)$ the first node in *Tr* which connects n_1 and n_2, i.e. $(n_1, \ldots, Rt)$ and $(n_2, \ldots, Rt)$ are the paths from n_1 respectively n_2 to Rt, then $sup(n_1, n_2)$ is the first node, which belongs to both of these paths. From the definition of an annotated tree, $sup(n_1, n_2)$ is uniquely defined by this property.

Let $Tr_i = (Nod_i, Ed_i, Rt_i, Ann_i)$, for $i = 1, 2$, be two M-annotated trees. We say that Tr_1 and Tr_2 are **equivalent** iff there is a bijection $f : Nod_1 \to Nod_2$ such that:

i) $f(Ed_1(n)) = Ed_2(f(n))$, $\forall n \in Nod_1 \setminus \{Rt_1\}$;
ii) $f(Rt_1) = Rt_2$;
iii) $Ann_1(n) = Ann_2(f(n))$, $\forall n \in Nod_1$.

We call f an **isomorphism** between Tr_1 and Tr_2.

In [6], (free-order) dependency grammars were introduced, as a rewriting device over two alphabets, of non-terminals and, respectively, terminals. In its general form, a dependency grammar can rewrite both non-terminals and terminals, by a finite set of rules (productions). Throughout this paper, we will work with dependency grammars, which rewrite only non-terminals (in a similar way to context-free grammars), therefore, we will not use terminals on the lefthand-sides of the rules.

We call **dependency grammar** a structure $G = (N, T, S, P)$ such that N and T are non-empty, finite sets (the set of *nonterminals*, respectively, of *terminals*), $S \in N$ is the start symbol and P is a finite set, called the set of *productions* such that $P \subseteq (N \times VV \times \{L, R\}) \cup (N \times T)$, where $V = N \cup T$. Sometimes, we will write the productions in P as $A \to_L BC$, $A \to_R BC$, $A \to a$ instead of (A, BC, L), (A, BC, R), respectively (A, a).

Denote by Nat the set of natural numbers not equal to 0, by $[n]$ the set of the first n elements of Nat and by $Nat_0 = Nat \cup \{0\}$.

Let $G = (T, N, S, P)$ be a dependency grammar and denote $V = N \cup T$. A **DR-tree created by** G is a V-annotated tree *Tr = (Nod,Ed,Rt,Ann)* such that:

1. $Nod \subseteq Nat \times Nat$. If $Ed(i, j) = (k, l)$ then $j < l$.
2. A node $(i, j) \in Nod$ is a leaf if and only if $j = 1$ and $Ann(i, j) \in T$.
3. If $(i, j) \in Nod$, with $j \neq 1$ and $Ann(i, j) = A$, then one of the following cases necessarily occurs:
   a. $j = 2$ and there is exactly one node $n \in Nod$ such that $Ed(n) = (i, j)$; in this case $n = (i, 1)$ and if $Ann(n) = a$, the production $A \to a$ belongs to P.
   b. There are exactly two nodes $n_1, n_2 \in Nod$ such that $Ed(n_1) = Ed(n_2) = (i, j)$; in this case either:
      b1. $n_1 = (i, k)$ and $n_2 = (l, m)$ with $l > i$, $\max(k, m) = j-1$ and if $Ann(n_1) = B$ and $Ann(n_2) = C$, the production $A \to_L BC$ belongs to P, or
      b2. $n_1 = (l, k)$ and $n_2 = (i, m)$ with $l < i$, $\max(k, m) = j-1$ and if $Ann(n_1) = B$ and $Ann(n_2) = C$, the production $A \to_R BC$ belongs to P.

Let $n_o \in Nod, n_o = (i, j)$. We say that i is the **horizontal position** of n_o and j is the **vertical position** of n_o (see the example of a DR-tree in Figure 1). Let $Ed(n_1) = n_2$, i.e., $e_1 = (n_1, n_2) \in Ed$. Let i be the horizontal position of n_1 and j the horizontal position of n_2. If $i = j$ we say that e_1 is a **V-edge**, if $i > j$ we say that e_1 is an **L-edge**, if $i < j$ we say that e_1 is a **R-edge**.

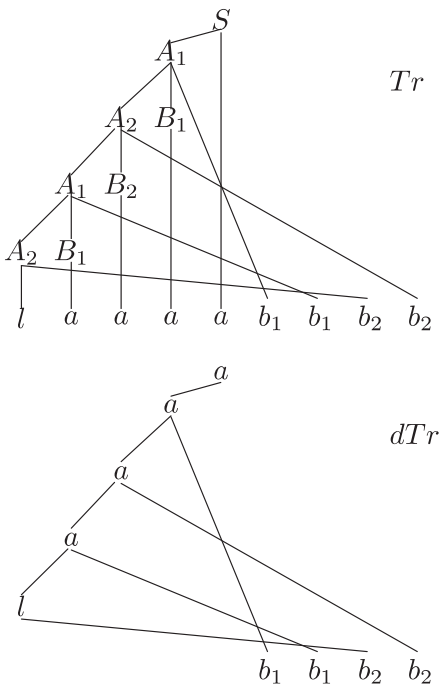


Fig. 1. An example of a DR-tree and its corresponding D-tree.

We say that a DR-tree *Tr = (Nod, Ed, Rt, Ann)* is **complete** iff for any leaf $(i,1) \in Nod$, if $i > 1$ then also $(i-1,1) \in Nod$. For any complete DR-tree *Tr = (Nod, Ed, Rt, Ann)* that is created by a dependency grammar $G = (N,T,S,P)$, we define the **sentence** associated with *Tr* by $s(Tr) = a_1 a_2 \dots a_n$, where $n = \max\{i \mid (i,1) \in Nod\}$ and $Ann(i,1) = a_i$, for any $i \in [n]$. Obviously, $s(Tr) \subseteq T^+$. In the sequel, we will consider that a dependency grammar creates only complete DR-trees.

Let $G = (N,T,S,P)$ be a dependency grammar. We denote by:

- $T(G)$ the set of DR-trees created by G and rooted by S;
- $DR\text{-}L(G) = \{s(Tr) \mid Tr \in T(G)\}$ the language generated by G, through the set of DR-trees.

Let $Tr_1 = (Nod_1, Ed_1, Rt_1, Ann_1)$ and $Tr_2 = (Nod_2, Ed_2, Rt_2, Ann_2)$ be two DR-trees created by the grammar $G = (N,T,S,P)$. We say that Tr_1 and Tr_2 are **DR-equivalent** iff there is an isomorphism $f: Nod_1 \to Nod_2$ between Tr_1 and Tr_2 as V-annotated trees and:

1. $f(i,j) = (s,j)$, for any node $(i,j) \in Nod_1$.
2. If $(i,j) \in Nod_1$, with $j \neq 1$ and $f(i,j) = (s,j)$, then:
   a. if there is exactly one node $n \in Nod_1$ such that $Ed(n) = (i,j)$ then $f(n) = (s, j-1)$.
   b. if there are exactly two nodes $n_1, n_2 \in Nod$ such that $Ed(n_1) = Ed(n_2) = (i,j)$ then either:
      b1. if $n_1 = (i,k)$ and $n_2 = (l,m)$ with $l > i$, then $f(n_1) = (s,k)$ and $f(n_2) = (t,m)$ with $t > s$ or
      b2. if $n_1 = (l,k)$ and $n_2 = (i,m)$ with $l < i$, then $f(n_1) = (t,k)$ and $f(n_2) = (s,m)$ with $t < s$.

We say that f is a **DR-isomorphism** between Tr_1 and Tr_2.

Let T be an alphabet and $Tr = (Nod, Ed, Rt, Ann)$ be a T-annotated tree such that $Nod \subseteq Nat$. We call *Tr* a **D-tree over** T (see the example of a D-tree in Figure 1). A DR-tree *Tr = (Nod, Ed, Rt, Ann)* can be transformed by contracting its vertical edges in a D-tree *dTr = (dNod, dEd, dRt, dAnn)* in the following way:

1. $dNod = \{i \mid (i,1) \in Nod\}$. $dRt = i$ iff $Rt = (i,j)$, for some $j \in Nat$.
2. Let $i \in dNod$ be a node in *dTr* and $(i,1) \in Nod$ the corresponding leaf in *Tr*. We consider the path $p = (n_1, n_2, \dots, n_k)$ in *Tr* from $n_1 = (i,1)$ to $n_k = Rt$. We also consider the natural number $r = \max\{l \mid n_l = (i,j), j \in Nat\}$. Then one of the following cases necessarily occurs:
   If $r = k$ then $dRt = i$. If $r < k$ and $n_{r+1} = (s,t)$ then $dEd(i) = s$.
3. $dAnn(i) = Ann(i,1)$, for any $i \in dNod$.

We say that *dTr* is the D-tree **corresponding** to *Tr*. We write $dTr = cr(Tr)$.

The D-tree *dTr* represented in Figure 1 is corresponding to the DR-tree *Tr* from the same figure. If *dTr* is a D-tree corresponding to a DR-tree created by a dependency grammar G, we say that *dTr* is created by G as well. A D-tree is **complete** if it corresponds to a complete DR-tree. Let $dEd(i) = j$, i.e., $e_1 = [i,j] \in dEd$. If $i > j$ we say that e_1 is an **L-edge**, if $i < j$ we say that e_1 is a **R-edge**.

3 Node gaps complexity

Let $Tr = (Nod, Ed, Rt, Ann)$ be an annotated tree, $n \in Nod$ be a node. We define the **covering subtree** of n in Tr by the following annotated tree, $Tr_n = (Nod_n, Ed_n, Rt_n, Ann_n)$ such that :

i) $Nod_n = \{n' \mid$ there is a path from n' to n in $Tr\}$;
ii) $Ed_n(n') = Ed(n')$, $\forall n' \in Nod_n \setminus \{n\}$;
iii) $Rt_n = n$;
iv) $Ann_n(n') = Ann(n')$, $\forall n' \in Nod_n$.

Let $Tr = (Nod, Ed, Rt, Ann)$ be a complete DR-tree and $n \in Nod$ be a node. Consider

$Tr_n = (Nod_n, Ed_n, Rt_n, Ann_n)$ the covering subtree of n in Tr. We define the **coverage** of n in Tr by the set $Cov(n,Tr) = \{i \in Nat \mid$ there is a node $(i,1) \in Nod_n\}$. Let $n \in Nod$ be a node in Tr such that $Cov(n,Tr) = \{i_1, i_2, \dots, i_m\}$, with $i_1 < i_2 < \dots < i_m$ and $i_{j+1} - i_j > 1$ for some $j \in Nat$, $j < m$. We say that the pair (i_j, i_{j+1}) is a **gap** in Tr at the node n. Let $Tr = (Nod, Ed, Rt, Ann)$ be a complete DR-tree, $n \in Nod$ be a node and $Cov(n,Tr)$ be its coverage. The symbol *DR-Ng(n,Tr)* represents the number of gaps in Tr at the node n. The symbol *DR-Ng(Tr)* is the maximum number of gaps in Tr at any node $n \in Nod$:

$$DR\text{-}Ng(Tr) = \max\{DR\text{-}Ng(n,Tr) \mid n \in Nod\}.$$

We say that *DR-Ng(Tr)* is the **DR-node-gaps complexity** of Tr. We say that Tr is **projective** iff *DR-Ng(Tr)=0*.

Let $G = (N,T,S,P)$ be a dependency grammar. We denote by:

- $T(G,i) \subseteq T(G)$ the set of complete and rooted by S DR-trees Tr created by G with at most i gaps, $DR\text{-}Ng(Tr) \leq i$;
- $DR\text{-}L(G,i) = \{s(Tr) \mid Tr \in T(G,i)\}$ the language generated by G, through DR-trees with at most i gaps.

– *DR-dT(G,i)* $= \{cr(Tr) \mid Tr \in T(G,i)\}$ the dT-language generated by G, through DR-trees with at most i gaps.

We mention the following obvious claims.

Claim. Let G be a dependency grammar. Then the following inclusions hold for any $i \in Nat_0$.
$T(G,i) \subseteq T(G,i+1) \subseteq T(G)$,
DR-L(G,i) $\subseteq$ *DR-L(G,i+1)* $\subseteq$ *DR-L(G)*,
DR-dT(G,i) $\subseteq$ *DR-dT(G,i+1)* $\subseteq$ *DR-dT(G)*.

Claim. For any complete DR-tree Tr_1 created by a dependency grammar G there exists a DR-equivalent projective complete DR-tree Tr_2 created by the same grammar G.

Claim. Let us suppose that Tr_3 is a DR-tree DR-equivalent to Tr_1. Then Tr_3 is also created by G.

Let us note that the same node-gaps-complexity measure can be considered for D-trees (see e.g. [6, 8]). The two complexity measures are quite different. We can easily define a non-projective DR-tree for which the corresponding D-tree is projective. It is not hard to see that for (our type of) dependency grammars is not easy to transfer the results obtained by node-gap comlexity based on DR-trees into results obtained by node-gap comlexity based on D-trees. In the following, we focus only on the node-gap complexity based on DR-trees.

4 Hierarchy

In the sequel of this paper, we will consider only dependency grammars in a normal form described by the following properties: the start symbol does not occur in the righthand side of any production; in any production of the form $A \to a$, the lefthand side nonterminal is the start symbol; any production is used in at least one complete DR-tree created by the grammar and rooted by the start symbol. It is not hard to see that the D-grammars with the previous restrictions keep the power of nonrestricted D-grammars due to the generation of the languages and of the sets of D-trees.

We denote by:

– DR-$\mathcal{L}(i)$ $=$ $\{$*DR-L(G,i)* $\mid$ G is a dependency grammar$\}$ – the classes of languages generated by dependency grammars through the set of DR-trees with at most i gaps;
– DR-$\mathcal{DT}(i)$ $=$ $\{$*DR-dT(G,i)* $\mid$ G is a dependency grammar$\}$ – the classes of dT-languages generated by dependency grammars through the set of DR-trees with at most i gaps.

Let $i \in Nat_0$, we denote

$$\text{DR-}\mathcal{LU}(i) := \bigcup_{j=0}^{i} \text{DR-}\mathcal{L}(j),$$

$$\text{DR-}\mathcal{DTU}(i) := \bigcup_{j=0}^{i} \text{DR-}\mathcal{DT}(j),$$

$$\text{DR-}\mathcal{LU}(\infty) := \bigcup_{j=0}^{\infty} \text{DR-}\mathcal{L}(j),$$

$$\text{DR-}\mathcal{DTU}(\infty) := \bigcup_{j=0}^{\infty} \text{DR-}\mathcal{DT}(j).$$

The next proposition follows from definitions.

Proposition 1 *For any $i \in Nat_0$ holds*

$$\text{DR-}\mathcal{LU}(i) \subseteq \text{DR-}\mathcal{LU}(i+1) \subseteq \text{DR-}\mathcal{LU}(\infty),$$

$$\text{DR-}\mathcal{DTU}(i) \subseteq \text{DR-}\mathcal{DTU}(i+1) \subseteq \text{DR-}\mathcal{DTU}(\infty).$$

To prove the main results of this paper we will define two types of languages of a particular kind. Let $n \in Nat$ be a natural number, $V = \{b_1, \ldots, b_n\}$ be an alphabet and $l, a \notin V$ be two distinct symbols. Denote by $L_{lab_1 \ldots b_n}$ and $L^{total}_{lab_1 \ldots b_n}$ two languages over $V \cup \{l, a\}$ such that $L_{lab_1 \ldots b_n} = \{\, la^{mn}(b_1)^m \ldots (b_n)^m \mid m \in Nat_0\}$ and $L^{total}_{lab_1 \ldots b_n} = \{\, lw \mid w \in (V \cup \{a\})^+, |w|_a = mn, |w|_{b_1} = \ldots = |w|_{b_n} = m, m \in Nat_0\}$, where $|w|_b$ denotes the number of occurrences of the symbol b in the string w.

In the sequel, we will consider languages bounded between $L_{lab_1 \ldots b_n}$ and $L^{total}_{lab_1 \ldots b_n}$, for appropriate natural numbers n and letters $b_1, \ldots, b_n$.

First, we prove that for the generation of a language of this kind by a dependency grammar G, the (necessary) node gaps complexity of the DR-trees created by G increases with n.

Theorem 1 (deleting theorem) *Let $L \in$ DR-$\mathcal{L}(i)$ be a language. Then, there exist two natural constants $p, r \in Nat$ such that for any sentence $w \in L$ with $|w| > p$, there exists a decomposition of w in $w = \alpha_1 a_1 \alpha_2 \ldots \alpha_r a_r \alpha_{r+1}$ such that the following conditions hold:*

i) $|a_h| > 0$, *for any* $h \in [r]$;
ii) $|a_1 \ldots a_r| < p$;
iii) $\alpha_1 \alpha_2 \ldots \alpha_r \alpha_{r+1} \in L$.
iv) If $r > i+1$ *there are at most* i *distinct indices* h *such that* $1 < h < r+1$ *and* $|\alpha_h| > p$.

Proof Let $G = (N, T, S, P)$ be a dependency grammar such that $L = \mathsf{DR\text{-}L(G,i)}$. Let no be the number of nonterminals in G. Denote $p = 2^{no+1}$. Let us consider a sentence $w \in L$ with $|w| > p$ and a DR-tree $Tr = (Nod, Ed, Rt, Ann)$ created by G such that $s(Tr) = w$. Since Tr is a binary tree with $|w|$ leaves, it results that there exists at least a (bottom-up) path $(n_0, n_1, \ldots, n_{no+1})$ in Tr. The nodes on this path are annotated by more than no nonterminals, thus in the sequence $Ann(n_1), \ldots, Ann(n_{no+1})$ there exist at least two equal nonterminals. Let us consider the first two equal nonterminals in this sequence, i.e. let us consider two indices $1 \leq s < t \leq no + 1$ such that $Ann(n_s) = Ann(n_t)$.

We consider the set $Cov(n_t, Tr) \backslash Cov(n_s, Tr)$, than we take the maximal sequences of consecutive indexes in this set and we denote by a_h, with h from 1 to a certain r the (non-empty) strings of terminals that correspond in w to these sequences of indexes. Let $w = \alpha_1 a_1 \alpha_2 \ldots \alpha_r a_r \alpha_{r+1}$. Obviously, we have $|a_h| > 0$, for any $h \in [r]$ and $|a_1 \ldots a_r| < p$ (conditions i) and ii) of the theorem).

Further, we may replace the covering subtree Tr_{n_t} with the covering subtree Tr_{n_s} (preserving the completeness under the transformation) and the resulted DR-tree Tr' is still a complete DR-tree created by G. Obviously the transformation will not increase the number of gaps. It results $s(Tr') = \alpha_1 \alpha_2 \ldots \alpha_{r+1} \in L$, which proves condition iii) of the theorem.

Finally, if $r > i + 1$, suppose towards a contradiction that there exist $i+1$ distinct indices h_j such that $1 < h_j < r + 1$ and $|\alpha_{h_j}| > p$, for all $j \in [i + 1]$. We observe that none of the gaps induced by α_{h_j}, with $j \in [i + 1]$ in the coverage of n_t in Tr can be fulfilled by the coverage of n_s in Tr, since the latest one has at most p elements. It follows that $DR\text{-}Ng(n_t, Tr) \geq i+1$ which is a contradiction with the assumption that $Tr \in T(G, i)$. This means that also the condition iv) of the theorem is true. □

Theorem 2 (second deleting theorem) *Let $L \in \mathsf{DR\text{-}}\mathcal{LU}(i)$ be a language. Then, there exist two natural constants $p, r \in Nat$ such that for any sentence $w \in L$ with $|w| > p$, there exists a decomposition of w in $w = \alpha_1 a_1 \alpha_2 \ldots \alpha_r a_r \alpha_{r+1}$ such that the following conditions hold:*

i) $|a_h| > 0$, for any $h \in [r]$;
ii) $|a_1 \ldots a_r| < p$;
iii) $\alpha_1 \alpha_2 \ldots \alpha_r \alpha_{r+1} \in L$.
iv) If $r > i + 1$ there are at most i distinct indices h such that $1 < h < r + 1$ and $|\alpha_h| > p$.

Proof It follows immediately from Theorem 1. □

Proposition 2 *Let $i, k \in Nat_0$ be two natural numbers such that $i < k$, $V = \{b_1, \ldots, b_{2k}\}$ be an alphabet, $l, a \notin V$ be two distinct symbols and L be a language over $V \cup \{l, a\}$ such that $L_{lab_1 \ldots b_{2k}} \subseteq L \subseteq L^{total}_{lab_1 \ldots b_{2k}}$. Then $L \notin \mathsf{DR\text{-}}\mathcal{LU}(i)$.*

Proof Suppose to a contradiction that $L \in \mathsf{DR\text{-}}\mathcal{LU}(i)$. It follows that Theorem 2 holds for L. Consider p and r the two constants from the deleting theorem, n a natural number such that $n > 2p$ and the sentence $w = la^{2nk}(b_1)^n \ldots (b_{2k})^n$. We have $|w| = 4kn + 1 > p$, hence, from Theorem 2, it should exists a decomposition of w in $w = \alpha_1 a_1 \alpha_2 \ldots \alpha_r a_r \alpha_{r+1}$ such that conditions i)-iv) of the theorem hold. Denote $w_0 = \alpha_1 \alpha_2 \ldots \alpha_r \alpha_{r+1}$. Since $L \subseteq L^{total}_{lab_1 \ldots b_{2k}}$, it follows that $|w_0|_{b_1} = \ldots = |w_0|_{b_{2k}} = m$ and $|w_0|_a = 2mk$. Further, it results that also $|a_1 \ldots a_r|_{b_1} = \ldots = |a_1 \ldots a_r|_{b_{2k}} = l$ and $|a_1 \ldots a_r|_a = 2lk$.

Since $|a_1 \ldots a_r| < p$ and $n > 2p$, it results that a_h for some $h \in [r]$ cannot include more than two distinct symbols from w, hence $r \geq k + 1 > i + 1$ and there are at least $k > i$ distinct indices h such that $1 < h < r + 1$ and $|\alpha_h| > p$, which yields a contradiction with condition iv) from the Theorem 2. It results that $L \notin \mathsf{DR\text{-}}\mathcal{LU}(i)$. □

However, languages of the above form can be generated with a large enough number of gaps in the structure of the DR-trees. We will consider below the case of special dependency grammars.

Proposition 3 *Let $i \in Nat_0$ be a natural number, $V = \{b_1, \ldots, b_{2i}\}$ be an alphabet and $l, a \notin V$ be two distinct symbols. Then there exists a language L_i over $V \cup \{l, a\}$ such that $L_{lab_1 \ldots b_{2i}} \subseteq L_i \subseteq L^{total}_{lab_1 \ldots b_{2i}}$ and $L_i \in \mathsf{DR\text{-}}\mathcal{LU}(i)$.*

Proof Consider $G_i = (N, V \cup \{l, a\}, S, P)$ a dependency grammar such that $N = \{S\} \cup \{A_j, B_j \mid j \in [2i]\}$, $P = \{S \rightarrow l, S \rightarrow_R A_1 a\} \cup \{A_j \rightarrow_L B_j b_j \mid j \in [2i]\} \cup \{B_j \rightarrow_R A_{j+1} a \mid j \in [2i-1]\} \cup \{B_{2i} \rightarrow_R A_1 a, A_{2i} \rightarrow_L l b_{2i}\}$. We take $L_i = \mathrm{DR\text{-}L}(G_i, i)$, hence $L_i \in \mathsf{DR\text{-}}\mathcal{L}(i)$ and, further, $L_i \in \mathsf{DR\text{-}}\mathcal{LU}(i)$.

It is easy to observe that $L_i \subseteq L^{total}_{lab_1 \ldots b_{2i}}$. We can see that the DR-tree Tr from Figure 1 is constructed by G_2. On a similar construction can be based the main part of the proof, proving that $L_{lab_1 \ldots b_{2i}} \subseteq L_i$. □

Now, we can state the following result, which is the first half from the main result of this paper.

Corollary 1 $\mathsf{DR\text{-}}\mathcal{LU}(k) \backslash \mathsf{DR\text{-}}\mathcal{LU}(i) \neq \emptyset$, *for any $i, k \in Nat_0$, with $i < k$.*

Proof Let $V = \{b_1, \ldots, b_{2k}\}$ be an alphabet and $l, a \notin V$ be two distinct symbols. From Proposition 3, we have that there exists a language L_k over $V \cup \{l, a\}$ such that $L_{lab_1 \ldots b_{2k}} \subseteq L_k \subseteq L^{total}_{lab_1 \ldots b_{2k}}$ and also $L_k \in$

DR-$\mathcal{L}(k)$. Since $i < k$, we have that $L_k \notin$ DR-$\mathcal{LU}(i)$ (from Proposition 2). Thus $L_k \in$ DR-$\mathcal{L}(k) \setminus$ DR-$\mathcal{LU}(i)$, hence $L_k \in$ DR-$\mathcal{LU}(k) \setminus$ DR-$\mathcal{LU}(i)$. That implies the corollary. □

We can state (the proofs follow immediately) the following results concerning the class of languages defined in this section (CFL^+ denote the class of context-free languages without the empty string).

Claim. The following properties hold:

i) Any language $L \in$ DR-$\mathcal{LU}(i)$, for some $i \in Nat_0$, is semilinear.
ii) DR-$\mathcal{L}(0) =$ DR-$\mathcal{LU}(0)$ is equal to CFL^+.

Now, we can state the main result of this paper.

Theorem 3 *For any $i \in Nat_0$, the following strict inclusions hold:*

i) DR-$\mathcal{LU}(i) \subset$ DR-$\mathcal{LU}(i+1) \subset$ DR-$\mathcal{LU}(\infty)$,
ii) DR-$\mathcal{DTU}(i) \subset$ DR-$\mathcal{DTU}(i+1) \subset$ DR-$\mathcal{DTU}(\infty)$.

Proof The statement i) results from the fact that DR-$\mathcal{LU}(i) \subseteq$ DR-$\mathcal{LU}(i+1)$, for any $i \in Nat_0$, and from Corollary 1.

The statement ii) is a direct consequence of the statement i). □

The next proposition follows from the results in [6, 7, 9].

Proposition 4 *There exists a sequential algorithm such that for every $i \in Nat^+$ and every $L \in DR\text{-}\mathcal{LU}(i)$ recognizes L in a polynomial time, where the degree of the polynomial increases with i.*

5 Conclusions

As an outcome of this paper, we have obtained an infinite hierarchy of classes of semilinear languages. This hierarchy is generated using topological parameters over uniform dependency grammars. The hierarchy starts by the class of context-free languages (without empty strings). A similar hierarchy was obtained for the classes of languages of dependency trees as well.

Using previous results, there exists an algorithm which recognizes the languages from the first hierarchy in a polynomial time and size. The hierarchies achieved through the classes of languages DR-$\mathcal{LU}(i)$ and DR-$\mathcal{DTU}(i)$ is the main novelty of this contribution. These results extend in a natural way the results from [7], where simpler classes of languages were considered. Also, these result illustrate the power of the parser [5]. Similar methods (gap-complexity) can be used for the study of constraints-based parsing dependency systems like [2]. As an example of such type of research can serve [8].

In the close future, we will study the classes of languages DR-$\mathcal{L}(i)$ for $i \in Nat_0$, in order to show that they are mutually incomparable to inclusion. Such results will strength the results presented here.

Also, we will study free-order dependency grammars with several kinds of topological restrictions in order to understand complex word-order and concurrency phenomena occurring in the syntax of natural languages. We believe that the study of free-order dependency grammars can also contribute to the understanding of concurrency phenomena, in general, as well. Our results can be easily interpreted as results about constraining the freedom of permutations on context-free languages.

References

1. A.Ja. Dikovskij, L.S. Modina, Dependencies on the Other Side of the Curtain. In: Traitement Automatique des Langues (TAL), 41(1), 2000, 79–111
2. R. Debusmann, D. Duchier, G.-J. Kruijff, Extensible Dependency Grammar: A New Methodology. In: Proceedings of the COLING 2004 Workshop on Recent Advances in Dependency Grammar, Geneva, 2004, 70–76
3. R. Gramatovici, M. Plátek, On the Power of Globally Restricted LD-Trivial Grammars with Global Word-Order Restrictions. In: MIS 2003, D. Obdržálek, J. Tesková (eds), Josefův Důl, 2003, 40–56
4. R. Gramatovici, M. Plátek, Proper Dependency Grammars. In: Proceedings of DCFS 2003, Fifth international Workshop, Descriptional Complexity of Formal Systems, E. Csuhaj-Varju, Ch. Kintala, and G. Vaszil (eds), Budapest, Hungary, July 2003, 255–264
5. T. Holan, Dependency Parser, proGRAM, `http://ksvi.mff.cuni.cz/~holan/proGRAM.html`
6. T. Holan, V. Kuboň, K. Oliva, M. Plátek, Two Useful Measures of Word Order Complexity. In: Proceedings of the Coling'98 Workshop "Processing of Dependency-Based Grammars", A. Polguere and S. Kahane (eds), University of Montreal, Montreal, 1998, 21–28
7. T. Holan, V. Kuboň, K. Oliva, M. Plátek, Word-Order Relaxations and Restrictions through a Dependency Grammar. In: Proceedings of the 7th International Workshop on Parsing Technologies, IWPT'01, Beijing, China, Tsingshua University Press, 2001, 237-240
8. M. Kuhlmann: Mildly Context Sensitive Dependency Languages. University of Saarland, Programming Systems Lab, Technical Report, 2005, `http://www.ps.uni-sb.de/Papers/`.
9. M. Plátek, T. Holan, V. Kuboň, On Relax-Ability of Word-Order by D-grammars. In: Combinatorics, Computability and Logic, C.S. Calude and M.J. Deneen (eds.), Springer Verlag, Berlin, 2001, 159–174
10. P. Sgall, E. Hajičová, J. Panevová, The Meaning of the Sentence in Its Semantic and Pragmatic Aspects. Dordrecht: Reidel and Prague: Academia, 1986

Knowledge extraction from data for tuning heuristic parameters of genetic algorithms

Martin Holeňa

Institute of Computer Science, Academy of Sciences of the Czech Republic,
Pod Vodárenskou věží 2, 182 07 Praha 8, Czech Republic, martin@cs.cas.cz, web:cs.cas.cz/~martin

Abstract. *The heuristic inspiration of genetic algorithms entails their dependence on many heuristic parameters. For the algorithm to work most properly, those parameters need to be empirically tuned. If the values of the objective function have to be obtained in a costly experimental way, then the algorithm can not be run with several various combinations of the values of heuristic parameters. This paper suggests to use knowledge extracted from data by means of a neural-network for parameter tuning in such situations. Moreover, it suggests to extract that knowledge in the form of a neural-network approximation of the objective function. That approximation is subsequenlty used instead of the experimentally obatined objective function values, to investigate the convergence speed of the algorithm and the diversity of the population for many various combinations of the values of heuristic parameters. On the other hand, some initial amount of data is needed to construct the approximating neural network. Those data are usually obtained from running the algorithm for several generations with default values.*

1 Introduction

Evolutionary and especially genetic algorithms (GAs) have been used since the last decade for optimization tasks in many application domains [1, 3, 4, 13–16, 22], mainly for the following reasons:

- they do not need gradient nor higher order derivatives of the objective function;
- they follow a collection of optimization paths instead of a single one;
- they require a comparatively low number of function calls parallelized with respect to the followed paths;
- they tend to find global optima rather than local ones.

Due to these reasons, GAs are particularly suitable for the optimization of functions for which the analytic form is not known and the costs of obtaining function values can not be neglected, typically when those values have to be obtained through experimental measurements. Their suitability further increases if there is a straightforward correspondence between the parallelism of the collection of optimization paths, and the way how the function values are experimentally obtained. Such situation occurs, for example, in materials science or in high-throughput chemical experiments.

On the other hand, the basic principle of evolutionary algorithms, i.e., the imitation of optimal survival in biological evolution, implies that they contain many heuristic parameters. To tune those parameters empirically would substantially increase the cost of solving the optimization task and completely invalidate the advantage of evolutionary and GAs being comparatively undemanding in terms of parallelized function calls.

This paper shows that instead of tuning the heuristic parameters by means of the experimentally obtained objective function, knowledge about that function extracted from the available data with artificial neural networks can be used. This needs some initial amount of data to be available, usually obtained from running the algorithm for several generations with some default parameter values, but then even a very comprehensive parameter tuning can be performed for no additional costs.

2 Genetic algorithms and their heuristic parameters

Genetic algorithms [5, 13–16, 19] are a stochastic optimization method. This means that when searching for maxima or minima of the objective function, the available information about that function is combined with random influences. The term "genetic algorithm" refers to the fact that their particular way of incorporating random influences into the optimization process has been inspired by the biological evolution of a genotype. Basically, that optimization method consists of:

- random exchange of coordinates of two particular points in the domain of the objective function, called *crossover or recombination*;
- random modification of coordinates of a particular point in the input space, called *mutation*; sometimes, a difference is made between *qualitative mutation* when the coordinate corresponds to a discrete-valued attribute, the individual values of which represent qualitatively different states of

objects, and *quantitative mutation otherwise*, especially when the coordinate corresponds to an interval-valued attribute (Figure 1);
- *selection* of the points for crossover and mutation according to a probability distribution, either uniform or skewed towards points at which the objective function takes high values (the latter being a probabilistic expression of the survival-of-the-fittest principle).

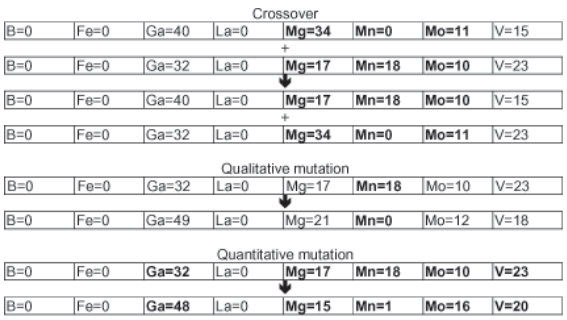


Fig. 1. Operations crossover, qualitative and quantitative mutation in genetic algorithms, illustrated on an example from materials science, in which the individual coordinates (genes) describe the proportion of the oxide of the indicated element in the material (adapted from [6].)

The operations selection, crossover and mutation require only function values of the objective function, and they can be performed in parallel for a whole population of points. At the same time, the incorporated random variables enable the optimization paths to leave the attraction area of the nearest local optimum, and to continue searching for a global one. The probability that at least one optimization path will reach the global optimum increases with the diversity of the population of points. On the other hand, those random variables heavily depend on the underlying probability distributions of possible values. Due to the biological inspiration of the random variables, a particular distribution can not be justified mathematically, but its choice is a heuristic task.

The most important heuristic choices entailed by a GA are:

- *overall probability of any modification* (crossover, qualitative or quantitative mutation) of an individual;
- *ratio between the conditional probabilities* of crossover and qualitative or quantitative mutation, conditioned on any modification;
- *distribution of the intensity of quantitative mutation*, e.g., distribution the coefficient with which the respective coordinate has to be multiplied / divided.

In addition, also the *population size* is sometimes a matter of heuristic choice, though in other cases it is determined by hardware limitations of how the values of the objective function are obtained (e.g., the number of measurement sensors, the number of channels in a chemical reactor).

3 An approach to tuning heuristic parameters based on knowledge extraction

If the values of the objective function have to be obtained in a costly experimental way, then it is not affordable to employ that costly evaluation also for tuning the involved heuristic parameters. The aim of this paper is to suggest that, in such situations, *knowledge about the objective function extracted from data* available on that function can be used for parameter tuning, instead of that function itself. In particular, knowledge extraction with artificial neural networks (ANNs) is expected to be very useful for this purpose (Figure 2), due to the ability of ANNs to approximate very general mappings [8, 9, 11, 12]. The cost of training an ANN and using it to evaluate the objective function in a population of points even for a large number of combinations of values of heuristic parameters is negligible compared to the cost of evaluating that function in those points for any single combination experimentally. In addition, experimental evaluation of the objective function for one generation of individuals typically needs hours to days of time, whereas if the ANN-based evaluation is computed instead, the GA advances to the next generation in a fragment of a second. The algorithm then can be run for many generations in that case, allowing to investigate its *convergence speed* for any particular combination of values of heuristic parameters. Moreover, also the evolution of the *diversity of the population of points* can be investigated in that way, which is important due to the role that the diversity plays in enabling optimization paths to reach the global optimum.

From a machine learning point of view, this leads to the task of approximating the objective function with an element of a function system determined through the architecture of the employed neural network. Although the approximated function is domain dependent (being the objective function of a domain-dependent optimization problem), that task is a domain-independent standard task, solved through ANN training. Therefore, it will not be recalled here, saving space for the presentation of first results obtained with the new approach.

Needless to say, training the ANN requires some *initial amount of data* from experimentally evaluating

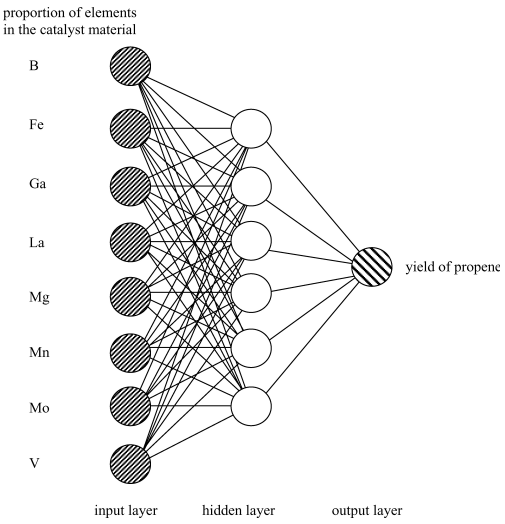


Fig. 2. Artificial neural network of the kind multilayer perceptron, employed in the application area considered in Figure 1 (reprinted from [7]).

the objective function to be gathered first. To this end, data from several early generations of the GA are usually sufficient, especially if the population size is large. Actually, data from the early generations are more uniformly distributed, making it more likely that the neural network will correctly approximate all optima, including the global optimum. Sometimes, also data from other experiments concerning the same problem are available. Once the parameters have been tuned, they can be used for all the remaining generations of the algorithm, or the tuning can be repeated every several generations, using each time a new network. In the latter case, the data from experimentally evaluating the objective function that were gathered since the last tuning are added to the ANN training data.

An obvious drawback of the proposed approach is that until the amount of data needed for training gets available, the values of heuristic parameters remain untuned, and have to be set to some kind of default values. However, first experience with the approach confirms the expectation that this drawback is outweighed with the possibility to investigate the behaviour of the GA for any combination of values of heuristic parameters.

4 Example of results obtained with the new approach

The proposed approach has been already used in several applications in chemistry and materials science. For illustration, this section describes results of tuning heuristic parameters of a GA employed to search optimal catalysts for the oxidative dehydrogenation of propane to propene [20]. The data came from running the GA for the 1.-5. generation with default parameters and population size 56, and were complemented with 48 catalysts from supplementary experiments. To evaluate the success of the GA in searching for the global maximum of the approximation, that maximum was first estimated with a deterministic optimization method. To this end, the Levenberg-Marquardt method was used, more precisely its modification for constrained optimization. Since that method searches only for local maxima, it was started from 7 different starting points. The highest of those local maxima was then considered as the global maximum of the approximation.

The approach was applied to 24 particular combinations of heuristic parameters, combined with 4 different population sizes. For each of those 96 final combinations, the algorithm was run repeatedly 10 times. This entails a certain variability of the obtained results, documented in Figure 3. Repeated runs of the GA accounted in average for an absolute error of 0.07% of the maximal value of the objective function found by the algorithm (corresponding to a relative error 0.8%), and to an absolute error of 0.1% of the population diversity (corresponding to a relative error nearly 100%).

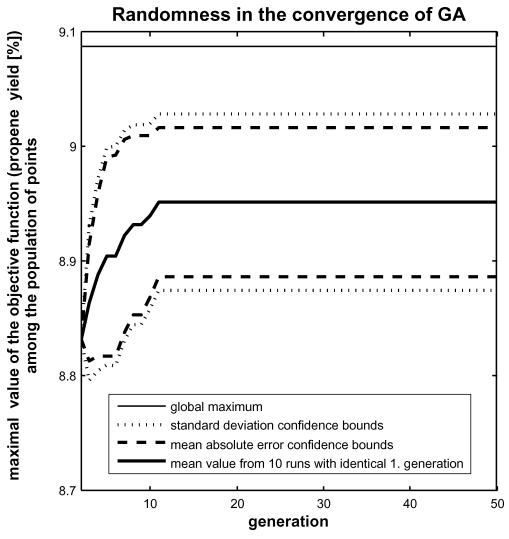


Fig. 3. Example of randomness-caused variability in the convergence of the genetic algorithm: the algorithm was run repeatedly 10 times with an identical population size and an identical combination of values of heuristic parameters, starting from an identical first generation.

For each of the final combinations, the following results have been recorded:

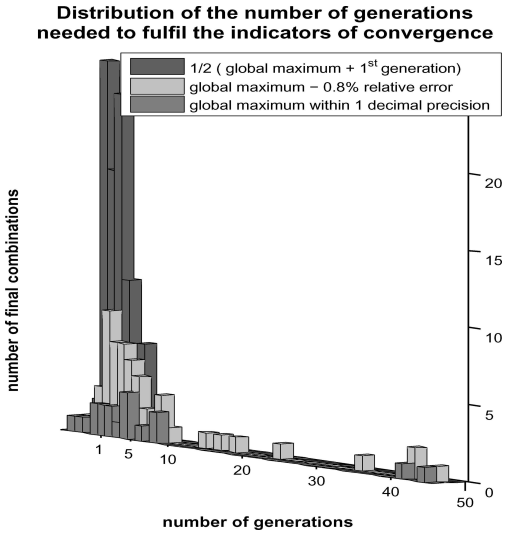


Fig. 4. Distribution of the number of generations needed for the value of the objective function to fulfil three increasingly strong indicators of convergence of the genetic algorithm (the algorithm has been run till the 50th generation, irrespectively of which of those indicators had been fulfilled).

- The convergence of the algorithm during the first 50 generations, which is measured as the evolution of the maximal value of the objective function among the population of points.
- The decreasing diversity among the population of points during the first 50 generations, where diversity is measured as the difference between the maximal and mean value of the objective function among the population of points.

In those results, the following indicators of convergence have been employed:

(i) Average of the value of the global maximum of the objective function, and the maximal value of the objective function among the population of points in the first generation, i.e., the value "1/2(global maximum + 1st generation)".

(ii) Global maximum of the objective function minus 0.8% relative error, which was the average relative error due to repeated runs of the algorithm with an identical combination of values of heuristic parameters.

(iii) The global maximum of the objective function within one decimal digit precision.

As indicators of decreasing diversity among the population of points, the following have been employed:

(i) Half the diversity in the first generation.

(ii) Diversity 1%.

(iii) Diversity 0.1%.

The most important results concerning the choice of the population size and the values of heuristic parameters obtained with the new approach can be summarized as follows:

1. The indicators of convergence that were fulfilled for a given combination of values of heuristic parameters were typically fulfilled already after early generations. Moreover, distributions of the number of generations needed to fulfil the three considered indicators of convergence are clearly skewed towards low values. Those distributions are depicted in Figure 4, which documents that early generations are much more important in optimization by means of GAs than later generations.
2. The convergence speed of the GA tends to increase with increasing population size (see Figure 5 for an example).
3. For a given generation above approximately the 10th generation, the maximal value of the objective function among the population of points tends to increase with increasing population size (cf. Figure 5).
4. The diversity among the proposed points decreases more quickly if crossover and mutation occur with equal probability than if one of them substantially prevails (Figure 6).

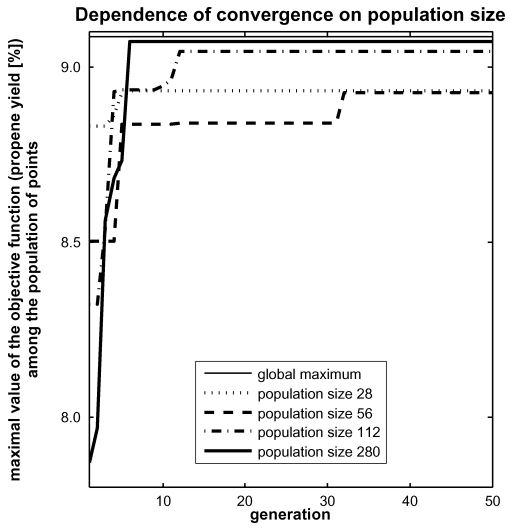


Fig. 5. Example dependence of the maximal value of the objective function among the population of points on the generation, for 4 considered population sizes.

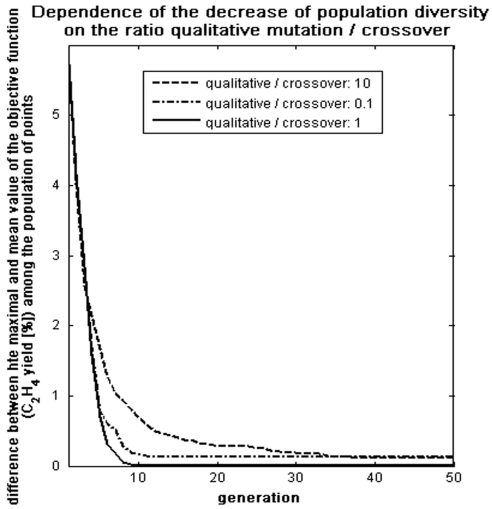


Fig. 6. Example illustrating that the diversity of the population of points in one generation of the genetic algorithm decreases more quickly if crossover and qualitative mutation occur with equal probability than if one of them substatntially prevails.

5 Conclusion

This paper dealt with the problem of tuning heuristic parameters of genetic algorithms in situations when the values of the objective function have to be obtained in a costly experimental way. It suggested to use knowledge about the objective function extracted from data by means of a neural-network instead of the function itself in such situations. In this way, it is possible to investigate the convergence speed of the algorithm and the diversity of the population of points for many various combinations of heuristic parameters. As to the author's knowledge, ANNs have not been used for tuning heuristic parameters of of genetic algorithm yet, although there already have been other attempts to use them for improving the performance of genetic algorithms [10]. Moreover, all these attempts are part of a broader direction of research into the use of machine learning methods in evolutionary computation [17, 18, 21]. The idea of replacing a costly computation of an objective function through a faster computation of its ANN-based approximation can aslo be encountered in inductive logic programming [2], where the objective function (scoring function of evaluated clauses) is costly from a computational complexity point of view.

For illustration, results from one of the first real-world applications of the approach have been presented. So far, only multilayer perceptrons have been used in the applications of the approach, extensions to other kinds of artificial neural networks are a subject of ongoing research.

Acknowledgement

The research reported in this paper has been supported by the grant No. 201/05/0557 of the Grant Agency of the Czech Republic and partially supported by the Institutional Research Plan AV0Z10300504.

References

1. T. Bäck, Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms. Oxford University Press, New York, 1996
2. F. DiMaio and Jude Shavlik, Learning an Approximation to Inductive Logic Programming Clause Evalutaion. In Proceedings of the 14th International Conference on Inductive Logic Programming, Springer Verlag, Berlin, 2004, 80–97
3. D.B. Fogel, Evolutionary Computation: Toward a New Philosophy of Machine Intelligence. IEEE Computer Society Press, New York, 1999
4. A.A. Freitas, Data Mining and Knowledge Discovery with Evolutionary Algorithms. Springer Verlag, Berlin, 2002
5. D. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading, 1989
6. M. Holeňa, Present Trends in the Application of Genetic Algorithms to Heterogeneous Catalysis. In A. Hagemeyer, P. Strasser, and A.F. Volpe (eds), High-Throughput Screening in Chemical Catalysis, Wiley-WCH, Weinheim, 2004, 153–172
7. M. Holeňa and M. Baerns, Feedforward Neural Networks in Catalysis. A Tool for the Approximation of the Dependency of Yield on Catalyst Composition, and for Knowledge Extraction. Catalysis Today, 81, 2003, 485–494
8. K. Hornik, Approximation Capabilities of Multilayer Neural Networks. Neural Networks, 4, 1991, 251–257
9. K. Hornik, M. Stinchcombe, H. White, and P. Auer, Degree of Approximation Results for Feedforward Networks Approximating Unknown Mappings and Their Derivatives. Neural Computation, 6, 1994, 1262–1275
10. J. Huhse, T. Villmann, P. Merz, and A. Zell, Evolution Strategy with Neighborhood Attraction Using a Neural Gas Approach. In Proceedings of the 7th International Conference on Parallel Problem Solving from Nature, Springer Verlag, Berlin, 2002, 391–400
11. V. Kůrková, Kolmogorov's Theorem and Multilayer Neural Networks. Neural Networks, 5, 1992, 501–506
12. V. Kůrková, Rates of Approximation by Neural Networks. In P. Sinčák and J. Vasčák (eds), Quo Vadis Computational Intelligence?, Springer Verlag, Berlin, 2000, 23–26
13. J.R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, 1992
14. J.R. Koza, Genetic Programming II: Automatic Discovery of Reusable Programs. MIT Press, Cambridge, 1994

15. J.R. Koza, F.H. Benett, D. Andre, and M.A. Keane, Genetic Programming III: Darwinian Invention and Problem Solving. Academic Press, Orlando, 1999
16. J.R. Koza, M.A. Keane, M.J. Streeter, W. Mydlowec, J. Yu, and G. Lanza, Genetic Programming IV: Routine Human-Competitive Machine Intelligence. Kluwer Academic Publishers, Dordrecht, 2003
17. X. Llorà and D.E. Goldberg, Wise Breeding GA via Machine Learning Techniques for Function Optimization. In G. Goos, J. Hartmanis, and J. van Leeuwen, (eds), Proceedings of GECCO 2003 – Genetic and Evolutionary Computation Conference, 2003, 1172–1183
18. R.S. Michalski, Learnable Evolution Model: Evolutionary Process Guided by Machine Learning. Machine Learning, 38, 2000, 9–40
19. M. Mitchell, An Introduction to Genetic Algorithms. MIT Press, Cambridge, 1996
20. U. Rodemerck, M. Baerns, and M. Holeňa, Application of a Genetic Algorithm and a Neural Network for the Discovery and Optimization of New Solid Catalytic Materials. Applied Surface Science, 223, 2004, 168–174
21. Y. Shan, R.I. McKay, H.A. Abbas, and D. Essam, Program Evolution wih Explicit Learning: A New Framework for Program Automatic Synthesis. In Proceedings of the 2003 Congress on Evolutionary Computation, 2003, 1639–1646
22. M.L. Wong and K.S. Leung, Data Mining Using Grammar Based Genetic Programming and Applications. Kluwer Academic Publishers, Dordrecht, 2000

On the nondeterministic state complexity of complements of regular languages*

Galina Jirásková[1] and Alexander Szabari[2]

[1] Mathematical Institute, Slovak Academy of Sciences, Grešákova 6, 040 01 Košice, Slovakia
jiraskov@saske.sk
[2] Institute of Computer Science, P.J. Šafárik University, Jesenná 5, 040 01 Košice, Slovakia
alexander.szabari@upjs.sk

Abstract. *The nondeterministic state complexity of a regular language is the number of states in a minimum state nondeterministic finite automaton accepting the given language. We investigate the nondeterministic state complexity of complements of regular languages represented by nondeterministic finite automata. We show that the nondeterministic state complexity of the complement of an n-state NFA language may reach the entire range of values from $\log n$ to 2^n. Our construction uses a $2n$-letter alphabet.*

1 Introduction

The state complexity of a regular language is the number of states in the minimal deterministic finite automaton for this language. The nondeterministic state complexity of a regular language is defined as the number of states in a minimum state nondeterministic finite automaton accepting the given language.

Some early results on the state complexity of regular languages can be found in [16,17,19]. Maslov [18] and Birget [2, 3] examined the state complexity of some operations on regular languages. The systematic study of the state complexity of regular language operations has been published by Yu, Zhuang, and Salomaa [24]. Pighizzini and Shallit [20] investigated the state complexity of unary regular language operations and Holzer and Kutrib [9] studied the nondeterministic state complexity of regular language operations. Further results on this topic are presented in [4–6, 15, 22].

Iwama, Kambayashi, and Takaki [11] stated the question of whether there always exists a regular language with nondeterministic state complexity n and with state complexity α for all integers n and α satisfying that $n \leqslant \alpha \leqslant 2^n$. The question has also been considered by Iwama, Matsuura, and Paterson [12]. In these two papers, the authors described binary n-state NFA languages for around $3n$ values of α in the range from 2^{n-1} to 2^n. The problem has been solved in [14] by presenting appropriate n-state NFA languages for all values of α in the range from n to 2^n, however, these languages are defined over an alphabet that grows exponentially with n. The explicit construtions that uses an alphabet of size $n+2$ has been given by Geffert [7]. The problem is still open for a fixed alphabet.

We have examined a similar problem for the nondeterministic state complexity of complements of regular languages in [13]. We have shown that the nondeterministic state complexity of the complement on an n-state NFA language over a $2(2^n-n)$-letter alphabet may reach any value in the range from $\log n$ to 2^n.

In this paper, we continue our work on this topics. We decrease the size of alphabet to $2n$ and still can prove that the nondeterministic state complexity of the complement of an n-state NFA language over a $2n$-letter alphabet may reach the entire range of values from $\log n$ to 2^n. While in the case of an exponential alphabet, we have been allowed to use a new input symbol to reach a specific subset when applying the subset construction to a given NFA, now, we must use a different approach when proving the reachability of subsets. To prove that an NFA is minimal we use a fooling-set lower-bound method known from communication complexity theory [1, 10]. This lower-bound technique has been successfully used in the field of regular languages several times [2, 3, 8, 15].

2 Definitions and notations

In this section, we give some basic definitions and notations used throughout the paper. For further details, we refer to [23, 25].

Let Σ be a finite alphabet and Σ^* the set of all strings over the alphabet Σ including the empty string ε. A *language over* Σ is any subset of Σ^*. We denote by L^c the complement of a language $L \subseteq \Sigma^*$ defined as $\{w \in \Sigma^* \mid w \notin L\}$. The cardinality of a finite set S is denoted by $|S|$ and its power-set by 2^S.

A *deterministic finite automaton* (DFA) is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is a finite input alphabet, $\delta : Q \times \Sigma \to Q$ is the transition function, $q_0 \in Q$ is the start state, and $F \subseteq Q$ is the set of final states. In this paper,

* Research supported by the VEGA grant No. 2/6089/26 and by the VEGA grant "Combinatorial Structures and Complexity of Algorithms".

all DFAs are assumed to be complete. The transition function δ is extended to a function from $Q \times \Sigma^*$ to Q in the natural way. A string w in Σ^* is accepted by the DFA M if $\delta(q_0, w) \in F$.

A *nondeterministic finite automaton* (NFA) is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$, where Q, Σ, q_0, and F are defined as for a DFA, and $\delta : Q \times \Sigma \to 2^Q$ is the transition function that can be naturally extended to the domain $Q \times \Sigma^*$. A string w in Σ^* is accepted by the NFA M if $\delta(q_0, w) \cap F \neq \emptyset$.

The *language accepted by* a finite automaton M, denoted $L(M)$, is the set of all strings accepted by the automaton M. Two automata are said to be *equivalent* if they accept the same language. A DFA (an NFA) M is called *minimal* if all DFAs (all NFAs, respectively) that are equivalent to M have at least as many states as M. By a well-known result, each regular language has a unique minimal DFA, up to isomorphism, but the same result does not hold for minimal NFAs.

The *state complexity* of a regular language L is the number of states in the minimal DFA for the language L. The *nondeterministic state complexity* of a regular language L is defined as the number of states in a minimal NFA accepting the language L. An *n-state* NFA *language* is a regular language with nondeterministic state complexity n.

Every nondeterministic finite automaton $M = (Q, \Sigma, \delta, q_0, F)$ can be converted to an equivalent deterministic finite automaton $M' = (2^Q, \Sigma, \delta', \{q_0\}, F')$ using an algorithm known as the "subset construction" [21] in the following way. Every state of the DFA M' is a subset of the state set Q. The transition function δ' is defined by $\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$ for each set R in 2^Q and each symbol a in Σ. The start state of the DFA M' is $\{q_0\}$. The set of final states F' is defined by $F' = \{R \in 2^Q \mid R \cap F \neq \emptyset.\}$

3 Results

Here we present our results concerning the nondeterministic state complexity of complements of n-state NFA languages defined over a $2n$-letter alphabet.

The upper bound on the nondeterministic state complexity of the complement of an n-state NFA language is 2^n since we can apply the subset construction to a given NFA and then exchange final and nonfinal states to get a DFA for the complement of at most 2^n states. The upper bound 2^n is tight and can be reached by complementation of binary regular languages [15]. The lower bound is, obviously, $\log n$.

In [13] we have shown that each value from $\log n$ to 2^n may be obtained as the nondeterministic state complexity of the complement of an n-state NFA language. However, to prove the result we have described regular languages over an alphabet that grows exponentially with n, namely, over an alphabet of size $2(2^n - n)$. In such a case, we can use a new input symbol to reach a specific subset when applying the subset construction to a given NFA.

In this section, we continue our work on this topic. We decrease the size of alphabet to $2n$ and still can prove that the nondeterministic state complexity of the complement of an n- state NFA language (over a $2n$-letter alphabet) may reach the entire range of values from $\log n$ to 2^n. We now must use a different approach when proving the reachability of subsets. To show that an NFA is minimal we use the fooling-set lower-bound method [1, 2, 8, 10].

After defining a fooling set, we give the lemma from [2] describing this lower-bound technique.

Definition 1. *A set of pairs of strings $\{(x_i, y_i) \mid i = 1, 2, \ldots, m\}$ is said to be a fooling set for a regular language L if for every i and j in $\{1, 2, \ldots, m\}$,*
(1) the string $x_i y_i$ is in the language L, and
(2) if $i \neq j$, then at least one of the strings $x_i y_j$ and $x_j y_i$ is not in L.

Lemma 1 (Birget [2]). *Let a set of pairs of strings $\{(x_i, y_i) \mid i = 1, 2, \ldots, m\}$ be a fooling set for a regular language L. Then every NFA for the language L needs at least m states.* □

Let us start with the following lemma that discusses the case of 1-state NFAs.

Lemma 2. *For each α in $\{1, 2\}$, there exists a binary 1-state NFA M_α such that any minimal NFA for the complement of the language $L(M_\alpha)$ has α states.*

Proof. Let $\Sigma = \{a, b\}$.

Define a 1-state NFA $M_1 = (\{q\}, \Sigma, \delta, q, \{q\})$, where $\delta(q, a) = \delta(q, b) = \{q\}$. The complement of the language $L(M_1)$ is the empty language which is accepted by a 1-state NFA.

Next, define a 1-state NFA $M_2 = (\{p\}, \Sigma, \delta', p, \{p\})$, where $\delta'(p, a) = \{p\}$ and $\delta'(p, b) = \emptyset$. The set of pairs of strings $\{(\varepsilon, b), (b, \varepsilon)\}$ is a fooling set for the language $L(M_2)^c$ since $b \in L(M_2)^c$ while $\varepsilon \notin L(M_2)^c$. By Lemma 1, every NFA for the language $L(M_2)^c$ needs at least 2 states. Since the complement of any 1-state NFA language can be accepted by a 2-state NFA, the lemma follows. □

The next lemma describes a class of binary regular languages that have the same nondeterministic state complexity as their complements.

Lemma 3. *For every $n \geqslant 2$, there exists a minimal binary NFA N of n states such that any minimal NFA for the complement of the language $L(N)$ has n states.*

Proof. Let n be arbitrary but fixed integer such that $n \geqslant 2$. Let $\Sigma = \{a, b\}$.

Define an n-state NFA $N = (Q, \Sigma, \delta, n, F)$, where $Q = \{1, 2, \ldots, n\}$, $F = \{1\}$, and for any $q \in Q$ and any $X \in \Sigma$,

$$\delta(q, X) = \begin{cases} \{n\}, & \text{if } q = 1 \text{ and } X = a, \\ \{1\}, & \text{if } q = 1 \text{ and } X = b, \\ \{q-1\}, & \text{if } 2 \leqslant q \leqslant n. \end{cases}$$

The NFA N is shown in Figure 1.

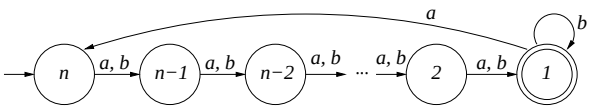


Fig. 1. The nondeterministic finite automaton N.

We are going to show that: (a) The NFA N is a minimal NFA for the language $L(N)$; (b) The language $L(N)^c$ is accepted by an n-state NFA; (c) Every NFA for the language $L(N)^c$ needs at least n states. Then, the lemma follows immediately.

To prove (a) consider the set of pairs of strings $\{(a^i, a^{n-1-i}) \mid i = 0, 1, \ldots, n-1\}$. This set is a fooling set for the language $L(N)$ because for every i and j in $\{0, 1, \ldots, n-1\}$,

(1) $a^i a^{n-1-i} = a^{n-1}$ and the string a^{n-1} is in the language $L(N)$ since it is accepted by N;
(2) if $i < j$, then $a^i a^{n-1-j} = a^{n-1-(j-i)}$ and the string $a^{n-1-(j-i)}$ is not in the language $L(N)$ since N does not accept any string a^t with $t < n-1$.

By Lemma 1, any NFA for the language $L(N)$ needs at least n states which proves (a).

To prove (b) note that the NFA N is, in fact, deterministic, and so by exchanging final and non-final states we get an n-state DFA for the language $L(N)^c$.

To prove (c) consider the set of pairs of strings $\{(b^i, b^{n-2-i}) \mid i = 0, 1, \ldots, n-2\} \cup \{(b^{n-1}, a)\}$. It is a fooling set for the language $L(N)^c$ since (1) the strings b^{n-2} and $b^{n-1}a$ are in the language $L(N)^c$, and (2) any string b^t with $t \geqslant n-1$ is not in the language $L(N)^c$. By Lemma 1, every NFA for the language $L(N)^c$ needs at least n states and our proof is complete. □

The following lemma shows that the nondeterministic state complexity of the complement of an n-state NFA language over a $2n$-letter alphabet may reach any value from $n+1$ to 2^n.

Lemma 4. *For all integers n and α such that $n \geqslant 2$ and $n < \alpha \leqslant 2^n$, there exists a minimal NFA M of n states with a $2n$-letter input alphabet such that any minimal NFA for the complement of the language $L(M)$ has α states.*

Proof. Let n and α be arbitrary but fixed integers such that $n \geqslant 2$ and $n < \alpha \leqslant 2^n$.

If $\alpha < 2^n$, then there exists an integer k such that $1 \leqslant k \leqslant n-1$ and

$$n - k + 2^k \leqslant \alpha < n - (k+1) + 2^{k+1}.$$

This means that $\alpha = n - k + 2^k + m$ for an integer m such that $0 \leqslant m < 2^k$. Let $c_{k-1}c_k \cdots c_1 c_0$ be the binary representation of m.

If $\alpha = 2^n$, then we set $k = n-1$ and $c_j = 1$ for all $j = 0, 1, \ldots, k-1$.

In both cases, we have $\alpha = n - k + 2^k + \sum_{j=0}^{k-1} c_j 2^j$.

Now, let

$$I = \{k\} \cup \{j \in \{0, 1, \ldots, k-1\} \mid c_j = 1\}$$

be the set containing the integer k and those indices j of $\{0, 1, \ldots, k-1\}$, for which c_j equals 1. Then α can be expressed as

$$\alpha = n - k + \sum_{i \in I} 2^i.$$

Next, let $\Sigma = \{a_i, b_i \mid i \in I\} - \{a_0, a_1\} \cup \{a, b\}$ be an alphabet consisting of two symbols a_i and b_i for every i in I, except for a_0 and a_1, and of two more symbols a and b.

We are going to define an n-state NFA M over the alphabet Σ such that any minimal NFA for the language $L(M)^c$ has α states. For every i in I, we define transitions on symbols a_i and b_i, see Fig. 2, so that we can prove the reachability of 2^i subsets when applying the subset construction to the NFA M. Transitions on symbols a and b, see Fig. 3, allow us to define a fooling set of size α for the language $L(M)^c$.

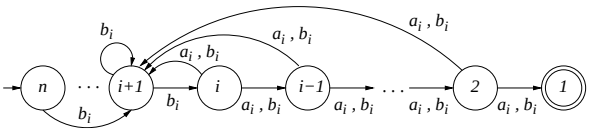


Fig. 2. Transitions on a_i and b_i in the NFA M.

Formally, define an n-state NFA $M{=}(Q, \Sigma, \delta, n, F)$, where $Q = \{1, 2, \ldots, n\}$, $F = \{1\}$, and for any $q \in Q$ and any $i \in I$,

$$\delta(q, a_i) = \begin{cases} \{q-1, i+1\}, & \text{if } 2 \leqslant q \leqslant i, \\ \emptyset, & \text{otherwise,} \end{cases}$$

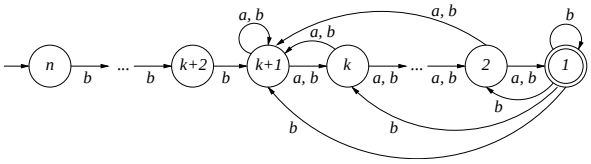


Fig. 3. Transitions on a and b in the NFA M.

$$\delta(q,b_i)=\begin{cases}\{q-1,i+1\}, & \text{if } 2\leqslant q\leqslant i+1,\\ \{i+1\}, & \text{if } q=n,\\ \emptyset, & \text{otherwise,}\end{cases}$$

$$\delta(q,a)=\begin{cases}\{q-1,k+1\}, & \text{if } 2\leqslant q\leqslant k+1,\\ \emptyset, & \text{otherwise,}\end{cases}$$

$$\delta(q,b)=\begin{cases}\{1,2,\ldots,k+1\}, & \text{if } q=1,\\ \{q-1,k+1\}, & \text{if } 2\leqslant q\leqslant k+1,\\ \{q-1\}, & \text{if } k+2\leqslant q\leqslant n,\end{cases}$$

recall that $n-k+2^k\leqslant\alpha<n-(k+1)+2^{k+1}$ if $\alpha<2^n$, and $k=n-1$ if $\alpha=2^n$. The symbols a_0 and a_1 are never used, and so the alphabet Σ has at most $2n$ letters.

We show that: (a) The NFA M is a minimal NFA for the language $L(M)$; (b) The language $L(M)^c$ is accepted by an α-state NFA; (c) Every NFA for the language $L(M)^c$ needs at least α states. Then, the lemma follows.

To prove (a) consider the set of pairs of strings $\{(b^i,b^{n-1-i})\mid i=0,1,\ldots,n-1\}$. This set is a fooling set for the language $L(M)$ because for every i and j in $\{0,1,\ldots,n-1\}$,

(1) $b^ib^{n-1-i}=b^{n-1}$ and $b^{n-1}\in L(M)$;

(2) if $i<j$, then $b^ib^{n-1-j}=b^{n-1-(j-i)}$ and the string $b^{n-1-(j-i)}$ is not in the language $L(M)$ since the NFA M does not accept any string b^t with $t<n-1$.

By Lemma 1, any NFA for the language $L(M)$ needs at least n states which proves (a).

To prove (b) let $M'=(2^Q,\Sigma,\delta',\{n\},F')$ be the DFA obtained from the NFA M by the subset construction. Let $\mathcal{R}$ be the following system of sets

$$\mathcal{R}=\{\emptyset,\{n\},\{n-1\},\ldots,\{k+2\}\}\cup\bigcup_{i\in I}\{\{i+1\}\cup S\mid S\subseteq\{1,2,\ldots,i\}\},$$

i.e., the system $\mathcal{R}$ contains the empty set, the singletons $\{n\},\{n-1\},\ldots,\{k+2\}$, and the set $\{i+1\}\cup S$ for every i in I and every subset S of $\{1,2,\ldots,i\}$. There are $n-k+\sum_{i\in I}2^i$ sets in the system $\mathcal{R}$. We are going to prove that every set in $\mathcal{R}$ is a reachable state of the DFA M' and no other states are reachable in M'.

The singletons $\{n\},\{n-1\},\ldots,\{k+2\}$, and the empty set are reachable since $\{q\}=\delta'(\{n\},a^{n-q})$ for $q=k+2,k+3,\ldots,n$ and $\emptyset=\delta'(\{n\},a_k)$; note that $k\leqslant n-1, k\in I$, and transitions on a_k in the NFA M go to the empty set for $q=k+1,k+2,\ldots,n$.

We next show that for every i in I and every subset S of $\{1,2,\ldots,i\}$, the set $\{i+1\}\cup S$ is reachable. We prove this by induction on the size of S. The set $\{i+1\}$ and the subsets $\{i+1,1\},\{i+1,2\},\ldots,\{i+1,i\}$ are reachable since $\{i+1\}=\delta'(\{n\},b_i)$ and $\{i+1,q\}=\delta'(\{i+1\},b_ia_i^{i-q})$ for $q=1,2,\ldots,i$. Let $2\leqslant m\leqslant i$ and assume that any subset $\{i+1\}\cup S$ with $|S|=m-1$ is reachable. Let $\{j_1,j_2,\ldots,j_m\}$, where $i\geqslant j_1>j_2>\cdots>j_m\geqslant 1$ be a subset of size m. Then we have $\{i+1,j_1,j_2,\ldots,j_m\}=\delta'(\{i+1,i-j_1+j_2+1,i-j_1+j_3+1,\ldots,i-j_1+j_m+1\},b_ia_i^{i-j_1})$, where the latter set is reachable by induction (note that $i\geqslant i-j_1+j_t+1\geqslant 2$ for $t=2,3,\ldots,m$). So every set in $\mathcal{R}$ is a reachable state of the DFA M'.

To prove that no other subset of the state set Q is reachable in the DFA M' it is sufficient to show that for every state R in $\mathcal{R}$ and every symbol X in Σ, the state $\delta'(R,X)$ is also in the system $\mathcal{R}$ (note that the initial state $\{n\}$ of M' is in $\mathcal{R}$). There are two cases:

(i) $R=\emptyset$ or $R=\{q\}$, where $q\in\{n,n-1,\ldots,k+2\}$. Then for every X in Σ, the set $\delta'(R,X)$ is equal either to the empty set, or to the singleton $\{q-1\}$, or to the singleton $\{i+1\}$ for an i in I. All these sets are in the system $\mathcal{R}$.

(ii) $R=\{j+1\}\cup S$, where $j\in I$ and $S\subseteq\{1,2,\ldots,j\}$. It follows that R is a subset of $\{1,2,\ldots,k+1\}$. Then the sets $\delta'(R,a)$ and $\delta'(R,b)$ are some subsets of $\{1,2,\ldots,k+1\}$ containing state $k+1$ since for every state q in $\{1,2,\ldots,k+1\}$, the transitions on a and b in the NFA M go to $\{q-1,k+1\}$. Next, for every i in I, the sets $\delta'(R,a_i)$ and $\delta'(R,b_i)$ are equal either to the empty set or to some subset of $\{1,2,\ldots,i+1\}$ containing state $\{i+1\}$ since for every state q in $\{1,2,\ldots,k+1\}$, the transitions on a_i and b_i in the NFA M go either to the empty set (if $q>i+1$ and for a_i also if $q=i+1$) or, to $\{q-1,i+1\}$ if $q\leqslant i+1$. In all cases, we get a set that is in the system $\mathcal{R}$.

Thus we have shown that the DFA M' obtained from the NFA M by the subset construction has exactly $n-k+\sum_{i\in I}2^i$ (i.e., α) reachable states. By exchanging final and non-final states in the DFA M', we get an α-state DFA for the language $L(M)^c$ which proves (b).

To prove (c) we are going to describe a fooling set for the language $L(M)^c$ of size α. We will do this in the following way. For every set S in $\mathcal{R}$, we define a pair of strings (x_S,y_S) such that the string x_Sy_S is in the language $L(M)^c$ and, moreover, if S and T are

two different sets in $\mathcal{R}$, then at least one of the strings $x_S y_T$ and $x_T y_S$ is not in the language $L(M)^c$. Then, the set $\{(x_S, y_S) \mid S \in \mathcal{R}\}$ will be a fooling set for the language $L(M)^c$ of size α.

Let $S \in \mathcal{R}$. Define the pair (x_S, y_S) as follows.

If $S = \emptyset$, let $x_S = a_k$ and $y_S = b^{n-1}$.

If $S = \{q\}$, where $q \in \{k+2, k+3, \ldots, n\}$, let $x_S = b^{n-q}$ and $y_S = b^{q-2}$.

If S is a nonempty subset of $\{1, 2, \ldots, k+1\}$ that is in $\mathcal{R}$, let x_S be an arbitrary string in Σ^* such that $\delta(n, x_S) = S$ (since every set in $\mathcal{R}$ is a reachable state of the DFA M', such string x_S must exist). We now define the string y_S.

If $S = \{1, 2, \ldots, k+1\}$, let $y_S = a_k^k$.

Otherwise, let ℓ be the *greatest* number in $\{1, 2, \ldots, k+1\}$ that is not in S. Define the string y_S of length $\ell - 1$ as follows: $y_S = y_1 y_2 \cdots y_{\ell-1}$, where for every $j = 1, 2, \ldots, \ell - 1$,

$$y_j = \begin{cases} a, \text{ if } j \in S, \\ b, \text{ if } j \notin S. \end{cases}$$

We first prove the following claim.

Claim. *For every subset S of $\{1, 2, \ldots, k+1\}$ and every state p in $\{1, 2, \ldots, k+1\}$,*

(A) if $p \in S$, then $1 \notin \delta(p, y_S)$,

(B) if $p \notin S$, then $1 \in \delta(p, y_S)$,

i.e., the string y_S is not accepted by the NFA M starting in any state of S, but it is accepted by M starting in every state in $\{1, 2, \ldots, k+1\}$ that is not in S.

Proof of Claim. The claim holds if $S = \emptyset$ or $S = \{1, 2, \ldots, k+1\}$.

Otherwise, $y_S = y_1 y_2 \cdots y_{\ell-1}$, where $\ell \notin S$, $\{\ell+1, \ell+2, \ldots, k+1\} \subseteq S$, and for every $j = 1, 2, \ldots, \ell - 1$, $y_j = a$ if $j \in S$ and $y_j = b$ if $j \notin S$.

To prove *(A)* let p be any state such that $p \in S$. There are two cases:

(i) $p > \ell$. Then the final state 1 cannot be reached from state p after reading the string y_S since the length of y_S is less than ℓ. Hence $1 \notin \delta(p, y_S)$.

(ii) $p < \ell$. Then $y_S = y_1 y_2 \cdots y_{p-1} \boldsymbol{a} y_{p+1} \cdots y_{\ell-1}$ since $p \in S$. Starting in state p and after reading the string $y_1 y_2 \cdots y_{p-1}$ of length $p-1$ we can either reach state 1 where no transition on a is defined, or we can reach state $k+1$ after reading a symbol y_j, where $j \leqslant p-1$, but then the length of the string $y_{j+1} y_{j+2} \cdots y_{\ell-1}$ is too short to reach state 1. Thus $1 \notin \delta(p, y_S)$.

To prove *(B)* let p be any state in $\{1, 2, \ldots, k+1\}$ such that $p \notin S$. There are two cases:

(i) $p = \ell$. Then state 1 can be reached from state p after reading any string in $\{a, b\}^*$ of length $\ell - 1$, so $1 \in \delta(p, y_S)$.

(ii) $p < \ell$. Then $y_S = y_1 y_2 \cdots y_{p-1} \boldsymbol{b} y_{p+1} \cdots y_{\ell-1}$ since $p \notin S$. Denote by d the length of the string $y_{p+1} y_{p+2} \cdots y_{\ell-1}$. Then $d \leqslant \ell - 2 \leqslant k - 1$. Starting in state p and after reading the string $y_1 y_2 \cdots y_{p-1}$ of length $p-1$ we can reach state 1. Then, on reading the symbol b we can reach state $d+1$, and then after reading the string $y_{p+1} y_{p+2} \cdots y_{\ell-1}$ of length d we can reach state 1. Hence $1 \in \delta(p, y_S)$ which completes the proof of the Claim.

Now, we are ready to show that the set of pairs of strings $\{(x_S, y_S) \mid S \in \mathcal{R}\}$ is a fooling set for the language $L(M)^c$. We need to show that
(1) for every S in $\mathcal{R}$, the string $x_S y_S$ is in $L(M)^c$, and
(2) if $S \neq T$, then at least one of the strings $x_S y_T$ and $x_T y_S$ is not in $L(M)^c$.

To prove (1) let $S \in \mathcal{R}$. We have three cases:

(i) $S = \emptyset$. Then $x_S y_S = a_k b^{n-1}$. The string $a_k b^{n-1}$ is not accepted by the NFA M and so it is in $L^c(M)$.

(ii) $S = \{q\}$, where $q \in \{k+2, k+3, \ldots, n\}$. Then $x_S y_S = b^{n-q} b^{q-2} = b^{n-2}$. The string b^{n-2} is not accepted by the NFA M and so it is in $L(M)^c$.

(iii) S is a nonempty subset of $\{1, 2, \ldots, k+1\}$. Then $\delta(n, x_S) = S$ and, by Claim *(A)*, the string y_S is not accepted by the NFA M starting in any state of S. Hence the string $x_S y_S$ is in $L(M)^c$.

To prove (2) let S and T be two different sets in the system $\mathcal{R}$. We have four cases:

(i) $S = \emptyset$ and T is a nonempty subset of $\{1, 2, \ldots, n\}$. Then $x_T y_S = x_T b^{n-1}$, where $\delta(n, x_T) = T$. Since the string b^{n-1} is accepted by the NFA M starting in every state in T, the string $x_T b^{n-1}$ is not in the language $L(M)^c$.

(ii) $S = \{p\}$ and $T = \{q\}$, where $k+2 \leqslant p < q \leqslant n$. Then we have $x_S y_T = b^{n-p} b^{q-2} = b^{n-2+q-p}$. Since $n - 2 + q - p \geqslant n - 1$, the string $b^{n-2+q-p}$ is accepted by the NFA M, so the string $x_S y_T$ is not in the language $L(M)^c$.

(iii) $S = \{q\}$, where $k+2 \leqslant q \leqslant n$, and T is a nonempty subset of $\{1, 2, \ldots, k+1\}$. Then $x_T y_S = x_T b^{q-2}$, where $\delta(n, x_T) = T$. Since $q - 2 \geqslant k$, the string b^{q-2} is accepted by the NFA M starting in every state of the nonempty set T. Hence the string $x_T y_S$ is not in the language $L(M)^c$.

(iv) S and T are two different nonempty subsets of $\{1, 2, \ldots, k+1\}$. Then, without loss of generality, there is a state p in $\{1, 2, \ldots, k+1\}$ such that $p \notin S$ and $p \in T$. By Claim *(B)*, the string y_S is accepted by the NFA M starting in state p. Since $\delta(n, x_T) = T$ and $p \in T$, the string $x_T y_S$ is accepted by the NFA M and so it is not in the language $L(M)^c$.

We have shown that the set of pairs of strings $\{(x_S, y_S) \mid S \in \mathcal{R}\}$ is a fooling set for the language $L(M)^c$.

By Lemma 1, any NFA for the language $L(M)^c$ needs at least α states and our proof is complete. □

We are now ready to prove the following result showing that the nondeterministic state complexity of the complement of an n-state NFA language over a $2n$-letter alphabet may reach the entire range of values from $\log n$ to 2^n.

Theorem 1. *For all positive integers n and α such that $\log n \leqslant \alpha \leqslant 2^n$, there exists a minimal NFA M of n states with a $2n$-letter input alphabet such that any minimal NFA for the complement of the language $L(M)$ has α states.*

Proof. By Lemma 2, the theorem holds if $n = 1$.

Let $n \geqslant 2$. By Lemma 3 and Lemma 4, the theorem holds if $n \leqslant \alpha \leqslant 2^n$.

Now, let $\log n \leqslant \alpha \leqslant n$. Then $\alpha \leqslant n \leqslant 2^\alpha$. By the above results, there exists a minimal α-state NFA A with a 2α-letter input alphabet such that any minimal NFA for the language $L(A)^c$ has n states. Let M be a minimal NFA for the language $L(A)^c$. We note that $L(M)^c = L(A)$ and $2\alpha \leqslant 2n$. Thus M is a minimal n-state NFA with a $2n$-letter input alphabet such that any minimal NFA for the language $L(M)^c$ has α states. This completes our proof. □

4 Conclusion

In this paper, we have examined the nondeterministic state complexity of complements of regular languages. We have shown that the nondeterministic state complexity of the complement of an n-state NFA language over a $2n$-letter alphabet may reach the entire range of values from $\log n$ to 2^n. This considerably improves our previous result [13] that has been achieved using an alphabet of size $2(2^n - n)$. Nevertheless, the problem remains open for a fixed alphabet.

References

1. A.V. Aho, J.D. Ullman, and M. Yannakakis, On Notions of Informations Transfer in VLSI Circuits. In: Proc.15th ACM STOC, ACM 1983, pp. 133–139
2. J.C. Birget, Intersection and Union of Regular Languages and State Complexity, Inform. Process. Lett. 43, 1992, 185–190
3. J.C. Birget, Partial Orders on Words, Minimal Elements of Regular Languages, and State Complexity. Theoret. Comput. Sci. 119, 1993, 267–291
4. C. Câmpeanu, K. Salomaa, and S. Yu, Tight Lower Bound for the State Complexity of Shuffle of Regular Languages J. Autom. Lang. Comb. 7, 2002, 303–310
5. M. Domaratzki, State Complexity and Proportional Removals. J. Autom. Lang. Comb. 7, 2002, 455–468
6. K. Ellul, B. Krawetz, J. Shallit, and M.W. Wang, Regular Expressions: New Results and Open Problems. J. Autom. Lang. Comb., to appear.
7. V. Geffert, (Non)determinism and the Size of One-Way Finite Automata. In: Proc. 7th DCFS 2005, 23–37
8. I. Glaister and J. Shallit, A Lower Bound Technique for the Size of Nondeterministic Finite Automata. Inform. Process. Lett. 59, 1996, 75–77
9. M. Holzer and M. Kutrib, Nondeterministic Descriptional Complexity of Regular Languages. Internat. J. Found. Comput. Sci. 14, 2003, 1087–1102
10. J. Hromkovič, Communication Complexity and Parallel Computing. Springer-Verlag, Berlin, Heidelberg, 1997
11. K. Iwama, Y. Kambayashi and K. Takaki, Tight Bounds on the Number of States of DFAs that Are Equivalent to n-State NFAs. Theoret. Comput. Sci. 237, 2000, 485–494
12. K. Iwama, A. Matsuura, and M. Paterson, A Family of NFAs which Need $2^n - \alpha$ Deterministic States. Theoret. Comput. Sci. 301, 2003, 451–462
13. J. Jirásek, G. Jirásková, A. Szabari, State Complexity of Concatenation and Complementation. Internat. J. Found. Comput. Sci. 16, 2005, 511–529
14. G. Jirásková, Note on Minimal Finite Automata. In: Proc. MFCS 2001, Lecture Notes in Comput. Sci., 2136, Springer, Berlin, 2001, 421–431
15. G. Jirásková, State Complexity of Some Operations on Binary Regular Languages. Theoret. Comput. Sci. 330, 2005, 287–298
16. O.B. Lupanov, A Comparison of Two Types of Finite Automata. Problemy Kibernetiki 9, 1963, 321–326 (in Russian)
17. Yu. I. Lyubich, Estimates for Optimal Determinization of Nondeterministic Autonomous Automata. Sibirskii Matematichskii Zhurnal 5, 1964, 337–355 (in Russian)
18. A. N. Maslov, Estimates of the Number of States of Finite Automata. Dokl. Akad. Nauk SSSR, 194, 1970 1266–1268 (in Russian) English translation: Soviet Math. Dokl. 11, 1970, 1373–1375
19. F.R. Moore, On the Bounds for State-Set Size in the Proofs of Equivalence between Deterministic, Nondeterministic, and Two-Way Finite Automata. IEEE Trans. Comput. 20, 1971, 1211–1214
20. G. Pighizzini and J. Shallit, Unary Language Operations, State Complexity and Jacobsthal's Function. Internat. J. Found. Comput. Sci. 13, 2002, 145–159
21. M. Rabin and D. Scott, Finite Automata and Their Decision Problems. IBM Res. Develop. 3, 1959, 114–129
22. A. Salomaa, D. Wood, and S. Yu, On the State Complexity of Reversals of Regular Languages. Theoret. Comput. Sci. 320, 2004, 315–329
23. M. Sipser, Introduction to the Theory of Computation. PWS Publishing Company, Boston, 1997
24. S. Yu, Q. Zhuang, and K. Salomaa, The State Complexity of Some Basic Operations on Regular Languages. Theoret. Comput. Sci. 125, 1994, 315–328
25. S. Yu, Chapter 2: Regular Languages. In: G. Rozenberg, A. Salomaa, (eds), Handbook of Formal Languages - Vol. I, Springer-Verlag, Berlin, New York, 1997, 41–110

A compression scheme for the R-tree data structure

Michal Krátký, Václav Snášel, and Radim Bača

Department of Computer Science, VŠB – Technical University of Ostrava
17. listopadu 15, 708 33 Ostrava–Poruba,Czech Republic
{michal.kratky,vaclav.snasel,radim.baca}@vsb.cz

Abstract. *Since the volume of data, e.g., web resources on the Internet and so on, increases nowadays, an efficient query processing over such data is necessary. There are a lot of applications where multi-dimensional data structures are applied. Due to the fact that the volume of data grows ad infinitum, a requirement of a compression scheme for such data becomes more and more evident. Consequently, an important problem is the efficient query processing over the compressed data. In this contribution we introduce a novel compression scheme for multi-dimensional data structures. We apply the compression scheme to well known R-tree data structure. Compressed nodes are stored in the secondary storage, but the R-tree is preserved dynamic and the compression is executed in real-time.*

Key words: *compression scheme, compression method, multi-dimensional data structures, R-tree*

1 Introduction

Many applications which utilize multi-dimensional data structures [3] exist in these days. For example some approaches to indexing the XML data [7, 10], term indexing [5, 11], spatial indexing etc. Performance of these application depends on performance of utilized multi-dimensional data structure.

Our work is focused on the well-known multi-dimensional data structure called the R-tree [8]. The compression effort should be directed on *MBB* (*minimal bounding boxes*) which are stored in the each node of R-tree. Depending on R-tree dimension MBBs can occupy more then 90% of the node space.

The first work [6], which is concerned with compressing R-tree pages, uses the relative representation of MBB to increase a fanout of the R-tree node. For better compression of their algorithm they proposed a bulk-loading algorithm, which is variation of *STR* [12] bulk-loading algorithm for R-trees. They also designed a lossy compression based on the coordinate quantization. Coordinates of MBB are then represented with fixed number of bits. Other works in this field are focused on improving the effectiveness of the main memory indexes. Those cache-conscious indexes suppose that they can store the most of the index in the main memory. Such work is *CR-tree* [9] as well, which uses similar type of MBB representation like in [6]. The compression algorithm is quite simple and fast and it is possible to check if two MBB overlap each other without decompression but the *false hits* can appear. This can lead to some overhead of query processing.

In this paper we decrease the size of index and number of leaf pages. This can leads to the better performance of R-tree. The proposed idea is based on compression of R-tree nodes in the secondary storage. More items can fit into the compressed R-tree node. Nodes are stored compressed only in the secondary storage, whenever the node is loaded into the main memory the node is decompressed. Naturally, in the main memory it can appear that the R-tree's nodes have the variable capacity. It has a minimal affect on the insert algorithm and the query algorithm is not affected at all.

We tested this approach for different compression algorithms and we developed our new algorithm for this purpose. This algorithm is able to compress node to 25% of the original size without apllying any lossy compression. This issue leads to an important reduction of the index size and the tree height can be decreased as well. Our algorithm uses the asymetric method: decompression is faster than compression because the time of querying is usually more important than the time of inserting.

In Section 2 R-tree and its variants are described in detail. In Section 3 the new compression scheme for R-tree is explained. In Section 4 we describe our new compression algorithm and in Section 5 we show results of our experiments.

2 R-tree and its variants

2.1 Introduction

Since 1984 when Guttman proposed his method [8], R-trees have become the most cited and most used as reference data structure in this area. As is required and expected by applications, they support usual point and range queries, and also some forms of spatial joins. Another interesting query supported by R-trees, to some extent, is the *k-NN* query.

R-tree can be thought of as an extension of B-trees in a multi-dimensional space. It corresponds to

a hierarchy of nested n-dimensional *minimum bounding boxes* (MBB) which may be defined by two n-dimensional points. If $\mathcal{N}$ is an interior node, it contains couples of the form (R_i, P_i), where P_i is a pointer to a child of the node $\mathcal{N}$. If R is its MBB, then the boxes R_i corresponding to the children $\mathcal{N}_i$ of $\mathcal{N}$ are contained in R. Boxes at the same tree level may overlap. If $\mathcal{N}$ is a leaf node, it contains its couples of the form (R_i, O_i), so called *index records*, where R_i contains a spatial object O_i. In Figure 2 a general structure of the R-tree for indexing point data is depicted.

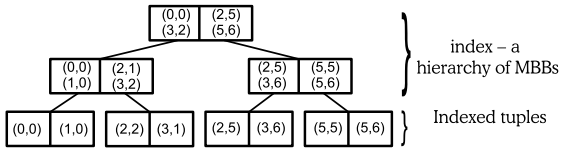


Fig. 1. Structure of the R-tree.

Each node of the R-tree contains between K and C entries unless it is the root and corresponds to a disk page. Other properties of the R-tree include the following:

- Whenever the number of a node's children drops below K, the node is deleted and its descendants are distributed among the sibling nodes. The upper bound C depends on the size of the disk page.
- The root node has at least two entries, unless it is a leaf.
- The R-tree is height-balanced; that is, all leaves are at the same level. The height of an R-tree is at most $\lfloor \log_K(m) \rfloor - 1$ for m index records ($m > 1$).

2.2 Split procedure

As a dynamic data structure, most attention of previous works on R-trees has been devoted to the split procedure during the adding of new index records into an R-tree. It significantly affects the index performance. Three split techniques (Linear, Quadratic, and Exponential) proposed in [8] are based on a heuristic optimization. The *Quadratic algorithm* has turned out to be the most effective and other improved versions of R-trees are based on this method.

The algorithm uses the following strategy: Given a set of $C + 1$ entries, each entry is assigned to one of the two produced nodes, according to the criterion of minimum area, i.e., the selected node is the one that will be enlarged the least in order to include the new entry.

Unfortunately, this criterion is taken for granted and not proved to be the best possible. The Quadratic algorithm tends to prefer the group with the largest size and higher population. In most cases this group will be least enlarged. Hence, there is a high chance it will need less area in order to accommodate the next entry, so it will be enlarged again. Over time, this will create a very uneven distribution, with most entries in one node. Also, when one of the groups becomes full, the rest of $C - K + 1$ entries are assigned to the second group without any geometric criteria. A minimum node capacity constraint also exists; thus a number of entries are assigned to the least populated node without any control at the end of the split procedure. This fact usually causes a significant overlap between the two nodes.

2.3 R-tree variants

R-tree performance is usually measured with respect to the retrieval cost (in terms of disk accesses) of queries. The majority of performance studies concerns point, range, and *k-NN* queries. Considering the R-tree performance, the concepts of node *coverage* and *overlap* between nodes are important. Obviously, an efficient R-tree search requires that both the overlap and coverage are minimized. Minimal coverage reduces the amount of dead area covered by R-tree nodes. The minimal overlap is even more critical than the minimal coverage; searching objects falling in the area of k overlapping nodes, up to k paths to the leaf nodes may have to be executed in such a way.

Variants of R-trees differ in the way they perform the split algorithm during insertions, i.e. which minimization criteria are used. Literature has identified a variety of criteria for the layout of keys on nodes that affect retrieval performance. These criteria are: minimal node area, minimal overlap between nodes, minimal node margins or maximized node utilization. It is impossible to optimize all of these parameters simultaneously. We will briefly put forward two well-known approaches to the R-tree optimization - R*-trees and R$^+$-trees. Authors of [13] put forward, in their recent exhaustive overview, another six variants.

The main feature of R*-trees [1] involves the node-splitting policy. Therefore, the R*-tree differs from the R-trees mainly in the insertion algorithm. Although original R-tree algorithms tried only to minimize the area covered by MBBs, the R*-tree algorithms also take the following objectives into account:

- The *overlap* between MBBs at the same (non-leaf) tree level should be minimized. The lesser overlap, the smaller the probability that one has to follow multiple search paths.
- *Perimeters* (*margins*) of MBBs should be minimized. For example, in 2D the preferred rectangle is the square, since this is the most compact rectangular representation.

– *Storage utilization* should be maximized. Nodes should store as many entries as possible so that the height of the tree is kept low.

According to the R*-tree split algorithm, the split axis is the one that minimizes a cost value S (S being equal to the sum of all margin values of the different distributions). Then the distribution which achieves minimum overlap-value is selected to be the final one along the chosen split axis. On the other hand, the distinction between the "minimum margin" criterion to select a split axis and the "minimum overlap" criterion to select a distribution along the split axis, followed by the R*-tree split algorithm, could cause the loss of a "good" distribution if, for example, that distribution belongs to the rejected axis. The design of the R*-tree also introduces a policy called *forced reinsert*: If a node overflows, it is not split in the right away. Moreover, p entries, $p > 0$, are removed from the node and reinserted into the tree. Authors of [1] suggest p should be about 30% of the maximal number of entries per page. Through all above mentioned techniques they reached performance improvements of up to 50% compared to the basic R-tree.

Clipping-based schemes do not allow any overlaps between bucket regions; they have to be mutually disjoint. A typical access method of this kind is the R^+-tree [16], a variant of the R-tree which allows no overlap between regions corresponding to nodes at the same tree level and an object can be stored in more than one leaf node. R^+-trees are considered to be one of the most efficient indexes for supporting point and range queries.

Other approaches to an improvement of original R-trees release some of their basic features. For example, the MBBs have been replaced by minimum bounding spheres or polygons. In [2] R^+-trees are extended to support *k-NN* queries. Special attention should be devoted to the use of signatures in connection with R-trees. The approach [14] offers an RS-tree that consists of an R-tree and an S-tree [4], i.e. a well-know hierarchical signature file. The main application of this data structure is an improvement of incremental *k-NN* query algorithm.

3 A compression scheme for the R-tree data structure

First, a motivation of our compression scheme is outlined. An R-tree node clusters similar tuples in the single node. The term "similar tuples" means the tuples are closed in a multi-dimensional space. Obviously, the similar tuples contain similar coordinates, e.g., in Table 1 we see the R-tree node. Some tuples' coordinates are the same or similar to each others.

The redundancy is possible to compress by a compression algorithm. We can apply well-known algorithms like RLE, differential encoding and so on. In Section 4 we depict the new algorithm applying a knowledge about R-tree nodes. In our scheme, compressed nodes are stored in the secondary storage. In Figure 2 we see transfer of tree's nodes between the secondary storage and tree's cache. If a tree wants to retrieve a node, the compressed node is transfered from the secondary storage. The node is decompressed and such node is stored in the cache. Therefore, tree's algorithms exploit decompressed nodes. Obviously, the R-tree is preserved as a dynamic data structure.

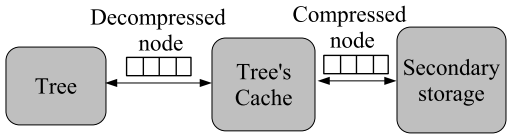


Fig. 2. Transfer of tree's nodes between the secondary storage and tree's cache.

3.1 The modified insert algorithm

When the compression scheme is taken into consideration, the original insert algorithm has to be modified. The new tree's insert algorithm is depicted in Listing 1.1. Before insertion of the tuple in the leaf node a compress algorithm has to compute if the tuple is possible to insert into the node. Consequently, the compress algorithm computes if the node is possible to insert into the single disk block its size is the same for all tree's nodes, e.g., 2048 B.

Listing 1.1. The modified insert algorithm

```
R−tree

Node node = mTree.mRootNode

while() {
  if (!node.IsLeafNode())
  {
    int order = node.GetMBBContaining(T)
    if (order == −1)      // no the MBB exists
    {
      // modify the closest MBB and set the order
      ...
    }

      // retrieve the child node
    Node node = mCache.GetNode(node.GetLink(order))
  }
  else
  {
```

```
    // a compress algorithm computes if the tuple
    // can be inserted
    bool flag = node.CanBeTupleInserted(T)

    if (flag)
    {
      node.Insert (T)
    }
    else
    {
      // split the node by an arbitrary algorithm
      // and change ascendants
       ...
    }
  }
}
```

3.2 An impact

As far as we consider the new insert algorithm we have to think about an impact. Our consideration is as follows:

1. Compressed nodes are retrieved in the secondary storage. Therefore, the lower volume of data is retrieved in the secondary storage during the query processing.
2. We create nodes with the variable capacity. Consequently, the tree's height is decreased. Since the complexity of tree's operations is depended on the tree's height, this issues can affect the efficiency of the query processing.

4 A Compression method

In this section we will describe the new compression algorithm. This algorithm is based on a separation of the matrix structure and matrix values. We consider a tree's node as a matrix. The matrix is referred as *Page Matrix* (PM). The size of the matrix is *Capacity* by *Dimension*, row contains coordinate values of one tuple. An example of the matrix is depicted in Table 1.

We will ascribe the algorithm for the example of tree's node depicted in Table 1. Steps of algorithm are as follows:

1. We compute matrix D which contains differences between rows of matrix PM, see Table 2. Obviously, the first row is the fixed point.
2. The structure of PM is saved into two arrays:
   - Linear addresses of matrix elements (see clause 3 below)
   - Indexes to the array of values (see clause 4 below)

$$PM = \begin{pmatrix} 4\ 0\ 6624\ 6625\ 1526\ 0\ 0 \\ 42\ 0\ 6624\ 6725\ 1535\ 0\ 0 \\ 9\ 0\ 6624\ 6626\ 6631\ 23\ 0 \\ 10\ 0\ 6624\ 6632\ 6633\ 26\ 0 \\ 29\ 0\ 6624\ 6660\ 6675\ 85\ 0 \\ 33\ 0\ 6624\ 6677\ 6678\ 112\ 0 \\ 33\ 0\ 6624\ 6677\ 6679\ 113\ 0 \\ 37\ 0\ 6624\ 6692\ 6695\ 113\ 0 \\ 34\ 0\ 6624\ 6680\ 6681\ 116\ 0 \\ 34\ 0\ 6624\ 6680\ 6682\ 117\ 0 \end{pmatrix}$$

Table 1. An example of the page matrix – a model of the R-tree's node.

$$D = \begin{pmatrix} 4\ 0\ 6624\ 6625\ 1526\ 0\ 0 \\ 38\ 0\ 0\ 100\ 9\ 0\ 0 \\ -33\ 0\ 0\ -99\ 5096\ 23\ 0 \\ 1\ 0\ 0\ 6\ 2\ 3\ 0 \\ 19\ 0\ 0\ 28\ 42\ 59\ 0 \\ 4\ 0\ 0\ 17\ 3\ 27\ 0 \\ 0\ 0\ 0\ 0\ 1\ 1\ 0 \\ 4\ 0\ 0\ 15\ 16\ 0\ 0 \\ -3\ 0\ 0\ -12\ -14\ 3\ 0 \\ 0\ 0\ 0\ 0\ 1\ 1\ 0 \end{pmatrix}$$

Table 2. Matrix of differences.

The linear address of a matrix element is computed as $LinearAddress(x, y) = x \times Dimension + y$. It means that the linear address is the item of a set $\{0, 1, 2, \ldots, Capacity \times Dimension\}$.

3. We divide the array of linear addresses into two arrays:
   - Array of nonzero linear addresses ($PM[x, y] <> 0$).
   - Array of zero linear addresses ($PM[x, y] = 0$).

   Array of nonzero linear addresses:
   7 10 11 14 17 18 19 21 24 25 26 28 31 32 33
   35 38 39 40 46 47 49 52 53 56 59 60 61 67 68

   Array of zero linear addresses:
   $$\begin{pmatrix} 8\ \ 9\ 12\ 13\ 15\ 16\ 20\ 22\ 23\ 27\ 29\ 30\ 34\ 36\ 37 \\ 41\ 42\ 43\ 44\ 45\ 48\ 50\ 51\ 54\ 55\ 57\ 58\ 62\ 63\ 64 \\ 65\ 66\ 69 \end{pmatrix}$$

   The smaller array is compressed by the Fibonacci compression [15].
4. Values which are stored in matrix D are divided into two arrays:
   - Array of ordered positive values
   - Array of ordered negative values

   Both arrays are compressed by the Fibonacci compression.

   Array of ordered positive values:

1 2 3 4 6 9 15 16 17 19
23 27 28 38 42 59 100 5096

Array of ordered negative values:
3 12 14 33 99

5. We have to store a bit-array which means signs of values referred by correspond linear address.

   Sign bit-array in the case of nonzero array of linear addresses:
   1110011111111111111111111000111
6. Finally, we compute the array of indexes. Each element of this array is the index in the array of positive values or array of negative values. The order of value in the array corresponds to the order of value in the array of linear addresses.

Index:
13 16 5 3 4 17 10 0 4 1 2 9 12 14 15
3 8 2 11 0 0 3 6 7 0 1 2 2 0 0

Listings 1.2 and 1.3 contain a code of methods `CompressPage()` and `DecompressPage()`,respectively.

Listing 1.2. `CompressPage(Input Page)`

```
Matrix P, D
Array NonZeroLinear, ZeroLinear, Index
BitArray Signum
Set PositiveValue, NegativeValue
P = Page
for(i = 1; i < Capacity; i++)
    D[i:] = P[1:] - P[i:]
for(i = 1; i < Capacity; i++)
    for(j = 0; j < Dimension; j++)
    if (D[i,j] == 0)
        ZeroLinear = ZeroLinear + LinearAddress(i,j)
    else
    {
        NonZeroLinear=NonZeroLinear
                      +LinearAddress(i,j)
        if (D[i,j] > 0)
        {
            Sinum = Signum + 1
            PositiveValue = PositiveValue + D[i,j]
        }
        else
        {
            Sinum = Signum + 0
            NegativeValue = NegativeValue
                          + abs(D[i,j])
        }
    }
for each x in NonZeroLinear
    Index = Index + ComputeIndex(x)

write D[1:]
if (|NonZeroLinear| > |ZeroLinear|)
    write ZeroLinear
else
    write NonZeroLinear
write FibonacciCompress(Index)
write FibonacciCompress(PositiveValue)
write FibonacciCompress(NegativeValue)
write Signum
```

Listing 1.3. `DecompressPage(Output Page)`

```
Matrix P
Array Linear, NonZeroLinear, ZeroLinear, Index
BitArray Signum
Set PositiveValue, NegativeValue

P = 0
Read P[1:]
Linear = FibonacciDecompress Read
Index = FibonacciDecompress Read
PositiveValue = FibonacciDecompress Read
NegativeValue = FibonacciDecompress Read
Signum = Read

if (Linear.Type = NonZero)
    NonZeroLinear = Linear
else
   NonZeroLinear = AllLinear - Linear

for each l in NonZeroLinear
    P[l.x, l.y] = GetValue(Index, PositiveValue,
                  NegativeValue, Signum)

for(i = 1; i < Capacity; i++)
    P[i:] = P[i-1:] + P[i:]

Page = P
```

5 Experimental results

In our experiments we compare various methods for the compression of R-tree's nodes. We show the our novel algorithm provides better compression rate. The framework *ATOM*[1] is applied in our implementation of the data structure. In our experiments[2] we build the R-tree as the index for XML data (see [10]). The index was built for the Protein Sequence Database XML document [17] its statistics are depicted in Table 3. Table 4 puts forward statistics of the R-tree index. The R-tree indexes the space of dimension 7.

In Table 5 we put forward that the both tested methods come to the good result. More than 60% of

[1] *AmphorA Tree Object Model*, `http://arg.vsb.cz`

[2] The experiments were executed on an Intel Pentium ®4 2.4Ghz, 1GB DDR400, under Windows XP.

Document size	683 MB
Number of elements	21,305,818
Number of attributes	1,290,647
Maximal length of the path	7

Table 3. Statistics of the Protein Sequence Database XML document.

Number of leaf items	8,739,522
Number of inner items	387,223
Number of leaf nodes	331,474
Number of inner nodes	55,750
Index size:	774 MB

Table 4. Statistics of the R-tree index.

Algorithm	**Compression rate**
Differential coding	29.6 %
The novel method	25.2 %

Table 5. Result of compression algorithms.

data is compressed. The novel algorithm provides better compression rate than the differential coding algorithm.

6 Conclusion

In our article we introduce the new compression scheme for the well-known R-tree data structure. The tree's node is compressed in the secondary storage but the cache contains uncompressed nodes. We introduce the novel compression algorithm to be provided the better compression rate than well-known differential coding. In our future work we would like to aim to an impact of real-time compression on the efficiency of the query processing.

References

1. N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In Proceedings of the 1990 ACM SIGMOD, ACM Press, 1990, 322–331
2. A. Belussi, E. Bertino, and B. Cataniac, Using Spatial Data Access Structures for Filtering Nearest Neighbor Queries. Data & Knowledge Engineering, 40(1), 2002, 1–31
3. C. Böhm, S. Berchtold, and D. A. Keim, Searching in High-Dimensional Spaces – Index Structures for Improving the Performance Of Multimedia Databases. ACM Computing Surveys, 33(3), 2001, 322–373
4. U. Deppisch, S-tree: A Dynamic Balanced Signature Index for Office Retrieval. In Proceedings of 9th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'86), Pisa, Italy, ACM Press, September, 1986, 77–87
5. V. Dohnal, C. Gennaro, and P. Zezula, A Metric Index for Approximate Text Management. In Proceedings of IASTED International Conference Information Systems and Database (ISDB 2002), ACTA Press, 2002
6. J. Goldstein, R. Ramakrishnan, and U. Shaft, Compressing Relations and Indexes. 1998, 370
7. T. Grust, Accelerating XPath Location Steps. In Proceedings of the 2002 ACM SIGMOD, Madison, USA, ACM Press, June 4-6, 2002
8. A. Guttman, R-Trees: A Dynamic Index Structure for Spatial Searching. In Proceedings of ACM SIGMOD 1984, Boston, USA, June 1984, 47–57
9. K. Kim, S. K. Cha, and K. Kwon, Optimizing Multidimensional Index Trees for Main Memory Access. In SIGMOD '01: Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data, New York, NY, USA, ACM Press, 2001, 139–150
10. M. Krátký, J. Pokorný, and V. Snášel, Implementation of XPath Axes in the Multi-dimensional Approach to Indexing XML Data. In Current Trends in Database Technology, Int'l Conference on EDBT 2004, Springer–Verlag 3268, 2004
11. M. Krátký, T. Skopal, and V. Snášel, Multidimensional Term Indexing for Efficient Processing of Complex Queries. Kybernetika, Journal, 40(3), 2004, 381–396
12. S. Leutenegger, M. Lopez, and J. Edgington, STR: A Simple and Efficient Algorithm for R-Tree Packing. *icde*, 1997, 497
13. Y. Manolopoulos, A. Nanopoulos, A. N. Papadopoulos, and Y. Theodoridis, R-trees Have Grown Everywhere. Submitted to ACM Computing Surveys, 2003
14. D.-J. Park, S. Heu, and H.-J. Kim, The RS-tree: An Efficient Data Structure for Distance Browsing Queries. Information Processing Letters, 80(4), November, 2001, 195–203
15. D. Salomon, Data Compression The Complete Reference. Third Edition, Springer–Verlag, New York, 2004
16. T.K. Sellis, N. Roussopoulos, and C. Faloutsos, The R+-Tree: A Dynamic Index For Multi-Dimensional Objects. In Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB'97), Morgan Kaufmann, 1997, 507–518
17. University of Washington's Database Group. The XML Data Repository, 2002, `http://www.cs.washington.edu/research/xmldatasets/`.

Sémantické vyhľadávanie v doméne pracovných ponúk*

Ján Krausko, Michal Barla, and Anton Andrejko

Ústav informatiky a softvérového inžinierstva, Fakulta informatiky a informačných technológií
Slovenská technická univerzita v Bratislave, Ilkovičova 3, 842 16 Bratislava, Slovensko
dzonyk@gmail.com

Abstrakt *Pravdepodobne najčastejšie využívanou službou na webe je vyhľadávanie informácií. Prevláda fulltextové vyhľadávanie na základe výskytu kľúčových slov v dokumentoch. Avšak výsledky sú často neuspokojivé. Možným riešením čoraz komplikovanejšieho vyhľadávania informácií na webe je sémantické vyhľadávanie. Cieľom príspevku je návrh implementácie sémantického vyhľadávacieho nástroja schopného pracovať s ontológiou pracovných ponúk vytvorenou v rámci iného projektu. Bližšie sa zaoberáme dopytovaním ontológie pracovných ponúk pomocou dopytovacieho ontologického jazyka SeRQL v ontologickom úložisku Sesame. V príspevku opisujeme softvérový prototyp sémantického vyhľadávacieho nástroja založeného na rámci Apache Cocoon, predstavíme zaujímavé spojenie tohto rámca a ontologického úložiska Sesame do sémantického vyhľadávacieho nástroja. Zameriame sa tiež na rôzne možnosti reprezentácie výsledkov prostredníctvom Apache Cocoon.*

1 Úvod

Web tak, ako ho poznáme teraz je prepojením dokumentov. Iniciatíva webu so sémantikou sa pripojením metadát k publikovaným dokumentom snaží vytvoriť dátovo prepojený web. Cieľom je dosiahnuť podobu, ktorá bude ľahšie strojovo čitateľná, jednoduchšie spracovateľná a vyhodnocovaná. Vhodným prostriedkom na reprezentáciu metadát, inšpirovaným zo znalostného inžinierstva, sú ontológie.

Jednou z najčastejšie využívaných služieb na webe je pravdepodobne vyhľadávanie informácií. V súčasnosti prevláda tzv. fulltextové vyhľadávanie informácií na základe výskytu kľúčových slov. Existujú mnohé algoritmy, ktoré následne usporadúvajú nájdené dokumenty podľa relevantnosti [4]. Fulltextové vyhľadávače nepoznajú obsah dokumentov, ktoré vyhľadali a ktoré zobrazujú používateľovi, čo často vedie k zlým alebo nepresným výsledkom [6].

Možným riešením čoraz komplikovanejšieho vyhľadávania informácií na webe je sémantické vyhľadávanie. Vyhľadávaču neposkytujeme kľúčové slová, o ktorých si myslíme, že by sa mohli často vyskytovať v hľadanom type dokumentu, ale kritéria, ktoré má spĺňať nájdený obsah. Predpokladá sa teda, že vyhľadávač bude „rozumieť" obsahu, ktorý ponúka.

Existuje niekoľko typov sémantického vyhľadávania v dokumentoch [3], [7]. Základným typom je vyhľadávanie informácií (information retrieval) – identifikácia relevantných dokumentov a ich radenia podľa miery vhodnosti. Vyšším typom je vyhľadávanie, ktoré poskytuje odpovede na jednoduché otázky (simple question answering), napríklad „Kto je prezident Slovenskej republiky?". Zdokonalením by bol vyhľadávač, ktorý poskytuje odpovede na komplexné otázky (complex question answering), napríklad „Aká je súčasná situácia vysokého školstva v Slovenskej republike?". U všetkých typov je možné očakávať zvýšenú efektívnosť vyhľadávania. Súčasne platí, že u všetkých typov vyhľadávania budú používané rôzne techniky usudzovania a odvodzovania.

Existuje viacero projektov, ktoré sa priamo zaoberajú alebo aspoň využívajú vyhľadávanie v prostredí sémantického webu. Príkladom je projekt MKSearch[1], v rámci ktorého je vyvíjaný vyhľadávací nástroj založený na indexovaní metadát vo webových dokumentoch. Nástroj vyhľadáva v naindexovaných metadátach uložených vo forme RDF v ontologickom úložisku Sesame.

Projekt Bibster[2] je nástroj na asistenciu výskumným skupinám pri manažovaní, zdieľaní a vyhľadávaní bibliografických dát. Systém pracuje v prostredí peer to peer siete. Na uloženie dát používa ontologické úložisku Sesame a dopytovanie vykonáva prostredníctvom jazyka SeRQL.

V príspevku sa venujeme návrhu sémantického vyhľadávacieho nástroja, schopného poskytnúť výsledky odpovedajúce jednoduchým otázkam v dátach opísaných ontológiami. Overenie návrhu sémantického vyhľadávača sme zrealizovali vytvorením prototypu nástroja (Semantic Search Tool – SST), ktorý umožňuje používateľovi vyhľadávanie v pracovných ponukách na základe vlastností týchto ponúk. Pracovali sme s už existujúcou databázou pracovných ponúk a s doménovou ontológiou vytvorenou v rámci projektu NAZOU[3] [5].

* Táto práca bola čiastočne podporovaná štátnym programom výskumu a vývoja „Budovanie informačnej spoločnosti" na základe zmluvy č. 1025/04.

[1] MKSearch, http://www.mksearch.mkdoc.org
[2] Bibster, http://bibster.semanticweb.org
[3] Projekt NAZOU, http://nazou.fiit.stuba.sk

2 Princíp sémantického vyhľadávania

Vyhľadávací nástroj získa od používateľa jeho kritéria na pracovné ponuky, o ktoré by mal záujem, pomocou formulára. Ten je zostavený z viacerých rolovacích menu, v ktorých si používateľ môže zvoliť požadovanú hodnotu určitej vlastnosti ponuky. Napríklad môže vybrať konkrétnu pracovnú pozíciu, ktorej by sa mala týkať ním požadovaná pracovná ponuka. Jednotlivé zoznamy všetkých rolovacích menu sa načítavajú priamo z ontológie ešte pred samotným otvorením okna formuláru.

Používateľ podľa svojho uváženia nastaví hodnoty vybraným položkám. Na základe takto vybratých hodnôt sa vytvorí dopyt do ontologického úložiska. Výsledkom je zoznam ponúk, ktoré spĺňajú tento dopyt. Nevyplnené rolovacie menu sa pri tvorbe dopytu neuplatnia. Zoznam nájdených pracovných ponúk sa zobrazí v tabuľke, kde sú ku každej ponuke uvedené jej základné atribúty (názov ponuky a meno firmy, ktorá ponuku zadala) vrátane odkazu na zobrazenie jej detailu. Používateľ si môže zobraziť detail o konkrétnej ponuke, prípadne spresniť dopyt opätovným vyplnením hodnôt vo formulároch.

Uvedená metóda umožňuje vyhľadávať pracovné ponuky na základe vlastností, ktoré nadobúdajú hodnoty z ohraničených množín. Takéto vlastnosti smerujú väčšinou do enumerovaných tried alebo tried, ktoré sú plne definované svojimi podtriedami. Pri enumerovanej triede sú vymenované všetky možné inštancie danej triedy a nemá zmysel vytvárať iné inštancie. Pri triede definovanej podtriedami nemôže existovať taká inštancia triedy, ktorá zároveň nie je inštanciou niektorej podtriedy danej triedy.

Keďže uvedené vlastnosti sú zadefinované priamo v jazyku OWL[4] (*owl:oneOf* pre enumerovanú triedu a *owl:unionOf* pre triedu definovanú podtriedami), dá sa metóda sémantického vyhľadávania implementovať dostatočne generický tak, aby sa dala použiť pre vyhľadávanie inštancií ľubovoľnej triedy, ktorá má vlastnosti smerujúce na plne definované triedy.

3 Architektúra nástroja

Navrhnutú metódu sme overili vytvorením prototypu webovej aplikácie SemanticSearchTool (SST), ktorá umožňuje sémantické vyhľadávanie nad ontológiou pracovných ponúk.

Aplikácia je integrovaná do prezentačného rámca Cocoon, ktorý je založený na architektúre dátovodov a filtrov [1]. Cocoon obsahuje množstvo použiteľných blokov, ktoré po správanej konfigurácii poskytujú aplikácii bohatú funkcionalitu. Pre uloženie ontológie používame ontologické úložisko Sesame[5]. Sesame štandardne poskytuje RDFSchema úložisko so základným odvodzovaním vzťahov medzi triedami a inštanciami.

Základné prepojenie jednotlivých častí riešenia je zobrazené na obrázku 1. Používateľ vidí vo svojom prehliadači stránku vygenerovanú servletom Cocoon, v ktorom je zasadený nástroj SST. Ten je pomocou Sesame Repository API prepojený na ontologické úložisko, ktoré teoreticky nemusí byť spustené v tom istom servlet kontajneri na tom istom stroji.

Sesame dovoľuje ontológie ukladať do súboru, do pamäte alebo do RDBMS (Relational DataBase Management System). Závisí od požiadaviek používateľa, ktorý spôsob uprednostní. Uchovávaním ontológie v pamäti počítača vo forme ontologického modelu dosiahneme vysokú rýchlosť prehľadávania a odvodzovania znalostí. V tomto prípade sme však obmedzení jej kapacitou, čo môže byť dôvodom použitia súboru, kde sa znižuje rýchlosť práce, alebo môžeme použiť relačnú databázu [2].

Sesame môže súčasne spravovať viac ontologických úložísk, a preto pri nadväzovaní komunikácie (pozri obr. 2, fáza 1) je potrebné identifikovať konkrétne úložisko prostredníctvom jeho identifikátora.

4 Dopytovanie nad ontológiou pracovných ponúk

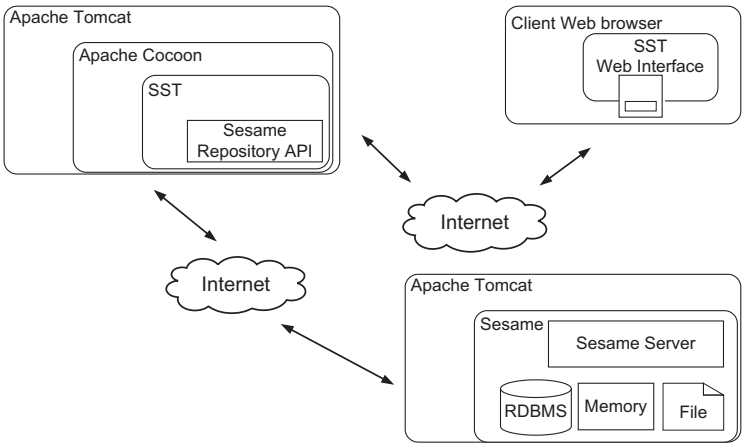
Cieľom nástroja pre sémantické vyhľadávanie je zostaviť dopyt, ktorého výsledkom bude zoznam pracovných ponúk vyhovujúcich kritériám používateľa. Nástroj SST prostredníctvom svojho formuláru (obrazovka aplikácie je na obrázku 2) umožňuje používateľovi špecifikovať tieto zúženia (kritéria) na vyhľadávanie:

- oblasť podnikania firmy, ktorá ponuku uverejnila, kde zoznam možností v rolovacom menu sa získava priamo z ontológie,
- pracovná pozícia – zoznam možností v rolovacom menu sa získava z ontológie,
- región – miesto vykonávania práce; zoznam možností v rolovacom menu sa získa z ontológie,
- druh pracovného pomeru – zoznam možností v rolovacom menu sa získa z ontológie, napr. *self employed* alebo *contract*,
- vhodnosť pre uchádzača bez praxe – k dispozícii sú predvolené možnosti: áno, nie, nezáleží.

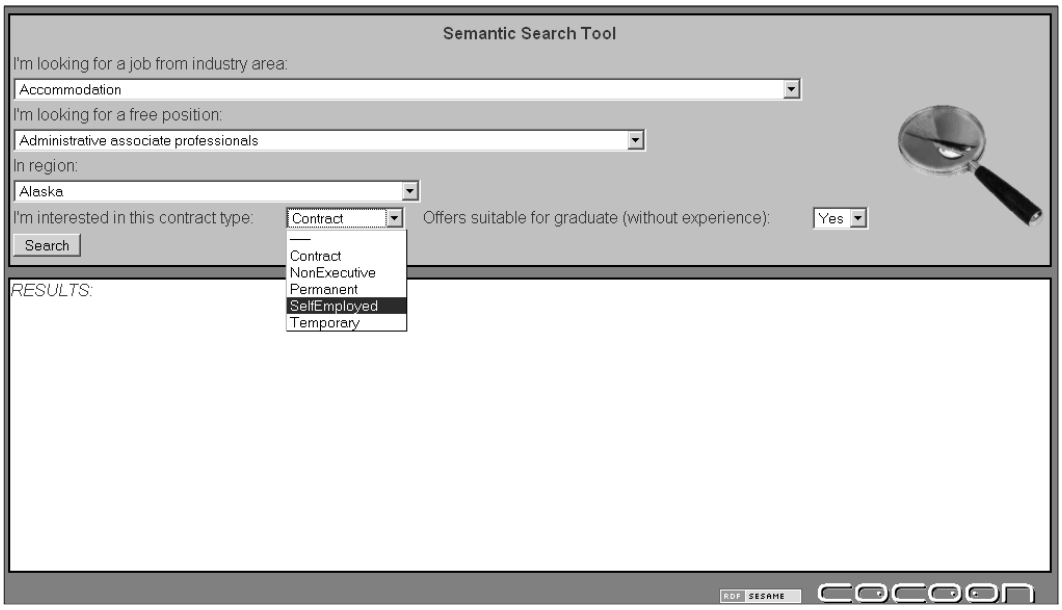
Zaujímavým kritériom je *vhodnosť pre uchádzača bez praxe*, keďže takýto atribút ponuky sa priamo v ontológií nenachádza. Ku každej ponuke však existujú

[4] OWL – Web Ontology Language, `http://www.w3.org/TR/owl-features`

[5] Sesame, `http://www.openrdf.org`



Obrázok 1. Architektúra komponentov sémantického vyhľadávacieho nástroja.



Obrázok 2. Používateľské rozhranie aplikácie.

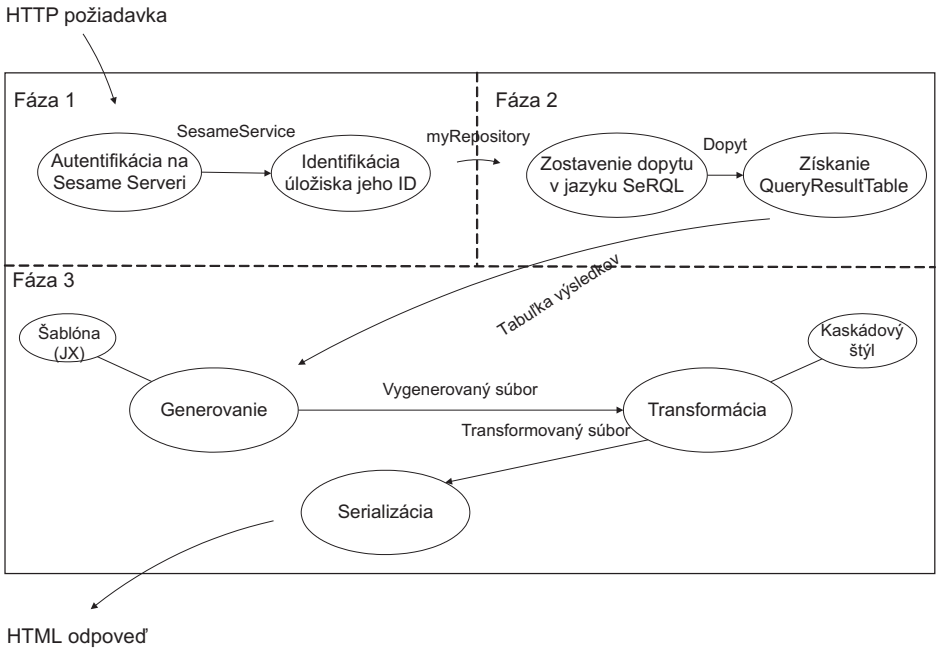
predpoklady kladené na uchádzača (*hasPrerequisites*). Tie môžu byť s ponukou spojené vlastnosťou *requires* (zamestnávateľ striktne vyžaduje splnenie predpokladu) alebo *prefers* (uchádzači, ktorí predpoklad spĺňajú sú zvýhodnení). Každý predpoklad sa môže týkať jednej z troch klasifikácií: klasifikácia skúseností, kvalifikácie a osobnostných atribútov.

Z uvedeného spôsobu modelovania vyplýva, že ponuka, ktorá je vhodná pre uchádzača bez praxe nesmie mať žiaden predpoklad prepojený s taxonómiou skúseností pomocou predikátu *requires*.

Dopyt, ktorý nám vráti zoznam ponúk nevhodných pre uchádzača bez praxe bude v jazyku SeRQL vyzerať nasledovne:

```
SELECT Offer
  FROM {Offer} jo:hasPrerequisite {P},
    {P} jo:requires {C},
    {C} rdf:type {c:ExperienceClassification}
```

Ontológia explicitne definuje jednotlivé logické časti entity (v našom prípade pracovnej ponuky) a vzťahy medzi týmito časťami. Hodnoty v každom poli formulára predstavujú možné ohraničenia priestoru ponúk podľa príslušnej logickej časti (región, pozícia a pod.). Podľa toho či používateľ dané pole vyplnil alebo nie sa vytvorením prienikov a zjednotení jednotlivých dopytov poskladá zložitejší dopyt zahrňujúci všetky požadované kritéria na hľadanú pracovnú ponuku.



Obrázok 3. Znázornenie priebehu spracovávania.

5 Implementácia nástroja

Pre zostavovanie dopytov na ontologické úložisko sme použili dopytovací jazyk SeRQL[6], ktorý vychádza z jazykov RDQL a RQL a čiastočne aj jazyk SPARQL[7] s ktorého plnou podporou sa v budúcnosti počíta po ukončení procesu štandardizácie organizáciou W3C. Okrem jazyka SeRQL rámec Sesame podporuje aj jazyk RDQL a RQL.

Vytvorená klientská aplikácia sa pripája cez Sesame Repository API k Java Servlet aplikácii umiestnenej na vzdialenom webovom serveri Apache Tomcat[8]. Sesame Repository API je jedným z dvoch Sesame Access API rozhraní umožňujúcich vysokoúrovňový prístup do úložiska s funkciami, ako dopytovanie či vkladanie RDF súborov. Druhé Graph API rozhranie poskytuje „jemnejšiu“ prácu nad úložiskom, ako je vytváranie malých RDF modelov priamo v kóde či rôznu manipuláciu s RDF. Implementácia klientskej aplikácie je založená na rámci Apache Cocoon[9] určenom pre vytváranie webových aplikácií založených na XML od Apache Foundation.

Spolupráca všetkých komponentov zahŕňa 3 navzájom nadväzujúce fázy, ktoré sú zobrazené na obrázku 3. Prvá fáza zabezpečuje nadviazanie spojenia so serverom a prvotnú komunikáciu servera s klientskou aplikáciou a servera s ontologickým úložiskom. Nástroj SST sa pokúsi pripojiť na vzdialené ontologické úložisko Sesame prostredníctvom prihlasovacieho mena a hesla. Po úspešnej autentifikácii je nutné identifikovať konkrétne úložisko pomocou *repositoryID*, s ktorým bude nástroj pracovať.

V druhej fáze je zostavený dopyt v jazyku SeRQL. Získané dáta odpovedajúce výsledku dopytu Sesame posiela vo forme tabuľky do ďalšej fázy. Dopyt môže byť zadaný priamo v kóde ako znakový reťazec dodržujúci syntax SeRQL alebo môže byť vygenerovaný na základe volieb používateľa zadaných prostredníctvom ponúknutých formulárov.

O tretiu fázu sa takmer výhradne stará rámec Cocoon, ktorý na základe šablón a výsledkov získaných z ontológie generuje, transformuje a nakoniec serializuje dáta do požadovaného formátu. V Apache Cocoon používame implementovaný JX generátor, ktorý má na vstupe šablónu, do ktorej na určené miesta doplní aktuálne hodnoty premenných. Šablóna spolu s CSS[10] určuje celkové rozloženie elementov na stránke. Cocoon vo fáze transformácie umožňuje tiež použitie XSL[11] štýlov. Rovnako je tiež možné využiť vlastný transformátor, ktorý relatívne ľahko naprogramujeme v jazyku Java alebo niektorý zo vstavaných transformátorov, napríklad i18n umožňujúci internacional-

[6] SeRQL – Sesame RDF Query Language
[7] SPARQL – SPARQL Protocol and RDF Query Language,`http://www.w3.org/TR/rdf-sparql-query/`
[8] Apache Tomcat, `http://tomcat.apache.org`
[9] V čase písania príspevku bol rámec Apache Cocoon vo verzii 2.1.8, `http://cocoon.apache.org`
[10] CSS – Cascading Stylesheet, `http://www.w3.org/Style/CSS`
[11] XSL – The Extensible Stylesheet Language Family, `http://www.w3.org/Style/XSL`

izáciu vytvorenej aplikácie. Súbor sa nakoniec serializuje do (X)HTML. Cocoon však ponúka oveľa viac možných výstupných súborov. Môžeme napríklad využiť WAP/WML Serializer na vytvorenie stránky pre mobilné zariadenia alebo pomocou PDF Serializer vygenerovať PDF dokument vhodný pre tlač.

Priebeh klientskej požiadavky je zobrazený na obrázku 4. HTTP požiadavku prichádzajúcu z webového prehliadača klientskeho počítača zachytáva aplikačný server (Apache Tomcat). Cocoon sa riadi konfiguračným súborom SiteMap (sitemap.xmap) vo formáte XML a dátovodmi (*pipelines*), ktoré sú v ňom nakonfigurované. Dátovody pracujú obdobne ako to poznáme pri dátovodoch v systéme UNIX (výstup jedného komponentu sa predáva na vstup druhého komponentu). V konfiguračnom súbore je tak isto namapovaná virtuálna adresárová štruktúra (používaná v HTTP požiadavkách) na skutočnú použitú na strane servera. Klientska požiadavka sa na základe regulárneho výrazu namapuje na príslušný dátovod. Riadenie a logiku nástroja zabezpečuje flowscript (vychádzajúci zo syntaxe jazyka Javascript), ktorého funkcie sa volajú z jednotlivých častí dátovodov. Na komunikáciu a prácu s ontologickým úložiskom Sesame Flowscript využíva metódy triedy MySesame, ktorá je vytvorená v jazyku Java a vložená do rámca Cocoon. Výsledky sú spätne odovzdané konfiguračnej mape, ktorá pre vytvorenie HTTP odpovede následne vyberá príslušný dátovod.

Prepojenie Cocoonu a aplikačnej logiky

Flowscript predstavuje jedno možné miesto prepojenia Cocoonu a aplikačnej vrstvy. Jeho funkcionalita je rozdelená do dvoch častí. Funkcia *vstup* zabezpečuje logiku pre úvodný formulár, zostavenie dopytu pomocou triedy *MySesame*, vyhľadávanie a zobrazenie nájdených ponúk. Funkcia *detail* zabezpečuje logiku pre zobrazenie detailu konkrétnej ponuky.

Prvá funkcia je v činnosti od spustenia nástroja SST. Pripojí sa na Sesame, získa všetky údaje potrebné na naplnenie rolovacích menu formulára a prostredníctvom bloku CocoonForms a generátora *JXGenerator* vygeneruje HTML dokument s formulárom.

Druhá funkcia dostane na svoj vstup ID konkrétnej ponuky a na jeho základe vyhľadá všetky dostupné informácie spojené s touto ponukou. Výsledný HTML dokument sa vygeneruje použitím *JXTemplate* šablóny, do ktorej sa vložia získané dáta.

Definícia formulára

Jednotlivé prvky formuláru (tzv. *widgety*) vyžadujú definíciu vo vstupnom súbore formou XML štruktúr. Napríklad pre rolovacie menu lokalít pracovných ponúk vyzerá takto:

```
<fd:field id="lokalita">
  <fd:label>In region: </fd:label>
    <fd:datatype base="string"/>
    <fd:selection-list type="flow-jxpath"
      list-path="List"
        value-path="value" label-path="label"/>
</fd:field>
```

Takto definovaný *widget* hovorí, že prvok formuláru *lokalita* bude ponúkať možnosť výberu jednej hodnoty zo zoznamu hodnôt, čo bude reprezentované formou rolovacieho menu (*selection-list*). Hodnoty jeho zoznamu v tomto prípade nie sú dopredu staticky nastavené, ale mu budú odovzdané z flowscriptu prostredníctvom premennej *List*.

Zobrazenie výsledkov

Zobrazenie výsledkov dopytu je možné viacerými spôsobmi – buď sa znova použije rámec CocoonForms a výsledky sa zobrazia vo formulári alebo sa použijú šablóny pre zobrazenie výsledkov.

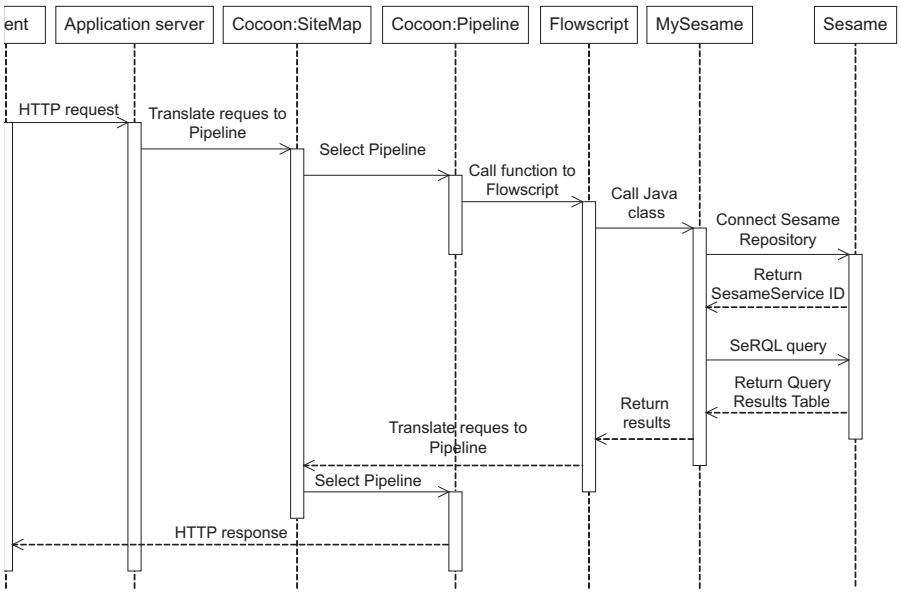
Rámec Cocoon obsahuje silný šablónový mechanizmus *JXTemplate*, pomocou ktorého sa dá nadefinovať výzor stránok spolu s pripojenými súbormi kaskádových štýlov. Šablóny používa *JXGenerator*, ktorý na určených miestach rozvinie zadefinované makrá a nahradí premenné skutočnými hodnotami.

Pri použití rámca CocoonForms sa dá využiť mapovanie medzi modelom formulára a dátami vo forme XML alebo Java bean. Toto mapovanie zadefinujeme pomocou pravidiel, ktoré priraďujú *widgety* z modelu formulára *JXPath* výrazom, ktoré adresujú polia v XML alebo premenné v Java bean objekte. Následne stačí, aby metóda, ktorá vykonáva dopyt vrátila výsledok vo forme XML alebo Java bean objekt. Takéto riešenie oddelí metódu sémantického dopytu od samotného prezentačného rámca, čo umožňuje jeho ľahkú výmenu. Zároveň je takáto metóda dopytovania flexibilná voči zmenám samotného rámca.

6 Zhodnotenie

Z viacerých možností riešení celkovej koncepcie sémantického vyhľadávacieho nástroja sme sa venovali spojeniu aplikácie postavenej na báze rámca Apache Cocoon a sémantického úložiska Sesame. Myšlienky použité pri návrhu takejto architektúry webovej aplikácie a spôsobu reprezentácie získaných výsledkov sú použiteľné všeobecnejšie.

Dôležitou časťou nášho sémantického vyhľadávacieho nástroja je jeho logika, ktorú tvorí flowscript napísaný v jazyku Javascript, konfiguračný súbor s dátovodmi pre Cocoon vo formáte XML a Java trieda pre komunikáciu s úložiskom Sesame. Používateľské



Obrázok 4. Sekvenčný diagram priebehu klientskej požiadavky.

rozhranie webovej aplikácie sa generuje na základe šablón vo formáte XML. Celkový vzhľad aplikácie je postavený na kaskádových štýloch CSS, čo umožňuje ľahkú zmenu celkového výzoru.

Riešenie založené na využití ontológií je oproti relačnej databáze flexibilnejšie, umožňuje niektoré zmeny v štruktúre dát (napr. zmenu či pridanie novej položky v pracovnej ponuke) bez nutnosti zásahu do aplikácie a tiež odvodzovanie znalostí. Dopytovanie v jazyku SeRQL je aktuálne závislé na konkrétnej doménovej ontológii. Nasadenie nástroja do inej oblasti by vyžadovalo zásah do zdrojových kódov. Skúmame možnosti generalizácie nástroja tak, aby sa pomocou neho dalo sémanticky dopytovať aspoň čiastočne nezávisle od použitej ontológie.

Referencie

1. Bass L., Clements P., Kazman R., Software Architecture in Practice. Second Edition, Addison-Wesley, 2003
2. Chong I.E., Das S., Eadon G., Srinivasan J., Supporting Keyword Columns with Ontology-Based Referential Constraints in DBMS. In 22nd International Conference on Data Engineering (ICDE'06) (2006) 95
3. Finin T., et al., Information Retrieval and the Semantic Web. In Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05) – Track 4. IEEE Computer Society 2005
4. Fürnkranz, J., Web Mining. In The Data Mining and Knowledge Discovery Handbook. Springer, 2005, 899–920
5. Návrat P., Bieliková M., Rozinajová V., Methods and Tools for Acquiring and Presenting Information and Knowledge in the Web. In International Conference on Computer Systems and Technologies – CompSysTech' 2005, Varna, Bulgaria, 2005
6. Roush W., Search Beyond Google. MIT Technology Review, 2004, 34–45
7. Sklenák V., Sémantický web. In Inforum 2003

O segmentaci českých vět

Vladislav Kuboň[1], Markéta Lopatková[1], Martin Plátek[2], and Patrice Pognan[3]

[1] ÚFAL, MFF UK, Praha, {vk,lopatkova}@ufal.mff.cuni.cz,
[2] KTIML, MFF UK, Praha, martin.platek@mff.cuni.cz
[3] CERTAL, INALCO, Paříž, mcertal@wanadoo.fr

Abstrakt *Příspěvek zavádí pojem (větného) segmentu, jednotky, která je lingvisticky motivovaná a přitom snadno automaticky rozpoznatelná. Rozpoznání segmentů umožňuje určovat segmentační strukturu věty (reprezentovanou segmentačním schématem), na jejímž základě lze vymezit jednotlivé klauze souvětí a jejich vzájemný vztah, a tím i syntaktickou strukturu souvětí. Metoda segmentace je navržena pro automatické zpracování češtiny, jazyka s relativně velmi volným slovosledem.*
V příspěvku je dále popsána sada jednoduchých pravidel, která je využita pro budování segmentačních schémat. Výsledky segmentace jsou vyhodnoceny vzhledem k malému ručně anotovanému korpusu českých vět.

1 Motivace

V tomto příspěvku zavádíme pojem (větného) segmentu, jednotky, která je lingvisticky motivovaná a přitom snadno automaticky rozpoznatelná. Je zde představena myšlenka nového modulu syntaktické analýzy (tzv. segmentační analýza), který by byl zařazen bezprostředně za modul morfologické analýzy. Myšlenka modulární automatické syntaktické analýzy češtiny je již dlouho považována za relevantní (nejméně od 80. let), přímo navazuje na myšlenku modulární automatické syntézy češtiny realizovanou pomocí *Funkčního generativního popisu češtiny* (viz [12], [13]).

Základním předpokladem pro modul segmentační analýzy je skutečnost, že čeština, pro kterou je tento přístup navrhován, má relativně striktní pravidla pro interpunkci, díky nimž lze jednoduše určit jednotlivé segmenty, ze kterých se věta či souvětí skládá. Lze předpokládat, že podobný přístup se dá uplatnit i pro řadu dalších, typologicky podobných jazyků. O přínosech obdobné metody i pro analýzu typologicky odlišných jazyků svědčí např. [5].

Tento příspěvek je rozšířením a prohloubením článku [9]. Je zaměřen na čtenáře, které zajímá problematika spojená s počítačovou syntaktickou analýzou češtiny. Téma větných segmentů (v souvislosti s počítačovou syntaktickou analýzou češtiny) bylo otevřeno v práci [7]. Hlavní motivací byla zkušenost, že automatická syntaktická analýza založená na jediném modulu závislostního typu (např. [2]) je v principu neúplná či neadekvátní. Taková analýza není schopna kontrolovat složitější interpunkční jevy ani složitější typy struktury klauzí a koordinací. (Kapitola práce [7] zabývající se větnými segmenty byla uveřejněna jako [8].)

Tento článek oproti [7] podstatně mění segmentační schéma. Nové schéma dokáže adekvátně zachytit složitější jevy, zejména rozlišit mezi větně členskou a větnou koordinací a řešit zanořování složitějších struktur.

Ve druhé kapitole přesněji zavádíme pojem separátorů a (větných) segmentů. Rozpoznání segmentů umožňuje určovat segmentační strukturu věty (reprezentovanou segmentačním schématem), na jejímž základě lze vymezit jednotlivé klauze souvětí a jejich vzájemný vztah, a tím i syntaktickou strukturu souvětí.

Ve třetí kapitole je popsána sada jednoduchých pravidel, která je využita pro budování segmentačních schémat českých vět.

Ve čtvrté kapitole jsou výsledky segmentace vyhodnoceny vzhledem k malému ručně anotovanému korpusu českých vět.

V závěru shrnujeme přínos příspěvku a naznačujeme, kterým směrem se při studiu segmentů budeme ubírat.

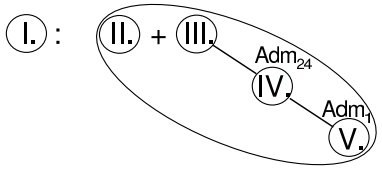
2 Popis struktury souvětí

Navrhovaná metoda je založena na předpokladu, že již morfologicky analyzovaný text obsahuje (více či méně spolehlivé) informace, které lze přímo využít k odhadu **segmentační struktury** souvětí, a tím k větší efektivnosti a přesnosti syntaktické analýzy.

Bohatým zdrojem složitých segmentačních jevů je Šmilauerova kniha [14]. Autor se s touto problematikou vyrovnává v rámci tradičního větného rozboru a tradiční terminologie (větných členů). Naším cílem je Šmilauerův větný rozbor automatickou syntaktickou analýzou nejen napodobit, ale zjemnit a rozšířit o složitější jevy související s volným slovosledem. Odtud také pramení náš zájem o větné segmenty.

Jako příklad se složitější segmentační strukturou si zvolíme větu rozebíranou Šmilauerem (viz [14], sekce 29, cvičení 16, str. 83): $_0$ *A Vesuv důvěrně mi šepce* **:**$_1$ *Jsem zbytečný* **,**$_2$ *už nesoptím* **, leda že**$_3$

dílo lidské hebce , **jak**$_4$ *sluší troskám* ‚$_5$ *zakoptím* .$_6$ Šmilauer tuto větu rozebral pomocí šesti obrázků (str. 171, kde je ovšem přidána interpunkční čárka před spojku *že*). První obrázek, který zachycuje segmentační strukturu věty, uvádíme na obr. 1, dalších pět obrázků odpovídá závislostním rozborům jednotlivých částí souvětí. V příkladové větě jsme vyznačili a očíslovali tzv. 'separátory', které větu rozdělují do šesti segmentů.



Obrázek 1. Rozbor struktury souvětí v [14].

Zdůrazněme, že nepředpokládáme, že by metoda segmentace mohla poskytnout jednoznačný odhad segmentační struktury pro každé zpracovávané souvětí – v mnoha případech ani jednoznačné určení struktury souvětí neexistuje, věty jsou (strukturálně) víceznačné. Cílem segmentace je získat co nejpřesnější odhad všech možných segmentačních struktur. Proto také terminologicky rozlišujeme **větný rozbor**, který by měl zachytit všechny možnosti, jak dané větě správně porozumět (tedy určit všechny její jazykem strukturované významy, s vyloučením mimojazykových znalostí) a **parsing** (odpovídající český termín není zaveden), kterým se obvykle rozumí určení jediné možnosti, jak větě porozumět, přičemž takový výběr je často podmíněn znalostí dalších částí textu i pragmatických znalostí získaných mimo daný text.

2.1 Základní pojmy

Pod pojmem věta/souvětí zde rozumíme část textu, která je větou v typografickém smyslu (začíná velkým písmenem a končí koncovou interpunkcí, nejčastěji tečkou, otazníkem či vykřičníkem); jde tedy o posloupnost lexikálních jednotek (v anglicky psané literatuře označované jako 'tokens') $w_1 w_2 \dots w_n$, kde každá položka w_i $(1 \leq i \leq n)$ reprezentuje buď jednu slovní formu daného přirozeného jazyka, nebo interpunkční znaménko (tečku, čárku, otazník, závorku, pomlčku, dvojtečku, středník a další speciální symboly, které se objevují v psaném textu). Implicitně předpokládáme, že ke každé lexikální jednotce je k disposici její morfologická analýza.

Separátory. Výskyty lexikálních jednotek ve větách jsou rozděleny do dvou disjunktních skupin.

Jako **separátory** označujeme takové řetězy slov a interpunčních znamének (přičemž bereme v úvahu jejich morfologickou interpretaci), která oddělují dvě větné klauze nebo dva větné členy. Přitom rozlišujeme **jednoduché separátory**, které jsou dány přímým výčtem (jde zejména o interpunkční znaménka, spojky, vztažná zájmena a zájmenná příslovce), a **separátory složené**, které zůstávají jediným funkčním oddělovačem, jsou však popsány pomocí (velmi jednoduchých) regulárních výrazů (např. čárka následovaná podřadicí spojkou, např. '*, že*'; součástí složeného separátoru může být i výraz, který není jednoduchým separátorem, třeba předložka (např. '*, pro kterou*') či zdůrazňující příslovce (např. '*, právě když*')).

Poznamenejme, že složené separátory je potřeba odlišovat od posloupností / řetězů separátorů, které jsou tvořeny dvěma (či více) po sobě jdoucími složenými separátory. Příklad posloupnosti separátorů najdeme např. ve větě *Nevěděl, že když jsem se probral k vědomí, zavolal jsem policii.*, kde dvojice '*, že*' tvoří jeden (složený) separátor a příslovce '*když*' druhý – tyto speciální konstrukce jsou popsány v [10].

Separátory (a jejich výskyty) lze relativně jednoduše rozdělit podle toho, zda uvozují klauzi nebo ji uzavírají. Mluvíme tedy o **otevíracích** separátorech (typicky např. podřadicí spojka či vztažné zájmeno; též začátek věty chápeme jako (nulový) separátor), **uzavíracích** separátorech (tečka, otazník nebo vykřičník na konci věty) a separátorech **kombinovaných** (zejména čárky či souřadicí spojky).

Segmentace a (větné) segmenty. Máme-li určeny výskyty separátorů ve větě, můžeme stanovit segmenty, ze kterých se věta skládá.

Nechť S, $S = w_1 w_2 \dots w_n$, je věta přirozeného jazyka. Jako **segmentaci věty** S označujeme každou posloupnost $D_0 W_1 D_1 \dots W_k D_k$, kde jednotlivé D_i $(0 \leq i \leq k)$ reprezentují složené separátory a kde W_i $(1 \leq i \leq k)$ označují jednotlivé **segmenty**, tedy maximální posloupnosti lexikálních jednotek, které neobsahují separátory.

Pro další úvahy využijeme též termín **rozšířený segment** popisující vždy dvojici $D_{i-1} W_i$ $(1 \leq i \leq k)$, kde D_{i-1} je otevíracím nebo kombinovaným složeným separátorem a W_i je bezprostředně následujícím segmentem.

Sekce D_0 může být prázdná (viz výš), všechny ostatní sekce D_i $(1 \leq i \leq k)$ jsou neprázdné. Každá jednotka w_i $(1 \leq i \leq n)$ náleží právě do jedné sekce D_j, pokud je součástí (složeného) operátoru; v opačném případě w_i náleží právě jednomu segmentu W_j. Sekce D_k reprezentuje koncovou interpunkci věty.

Sekce D_0 je typicky prázdná pro jednoduché věty a souvětí, které začínají hlavní klauzí. Naopak D_0 je typicky neprázdná, pokud souvětí začíná vedlejší

větou, jako např. ve větě *Když jsem se probudil, zavolal jsem policii.*

Dalším termínem, který nám umožní popis pravidel pro segmentaci, je termín **příznak podřízenosti**. Tento příznak se přiděluje jednotlivým rozšířeným segmentům na základě morfologické analýzy jejich jednotlivých slov a interpunkčních znamének (konkrétní návrh viz sekce 3) a slouží společně s klasifikací separátorů jako základ pravidel pro určování vzájemného vztahu jednotlivých segmentů v segmentačním schématu (vzhledem ke koordinace a podřízenosti), a tedy pro odhad struktury souvětí.

Segmentační struktura věty. Segmentační strukturu věty lze znázornit jedním nebo více segmentačními schématy, která zachycují vzájemný vztah jednotlivých segmentů vzhledem ke koordinaci a podřízenosti /nadřízenosti.

Segmentační schéma je orientovaný graf $G = \langle D, H \rangle$, kde D je množina obsahující uzly $D_i (0 \leq i \leq k)$, případně jejich kopie $D'_i, D''_i, ...$ a H je množina hran. Navíc stanovujeme následující pravidla pro grafické znázornění. Hranu $h \in H$ zachycujeme jako:

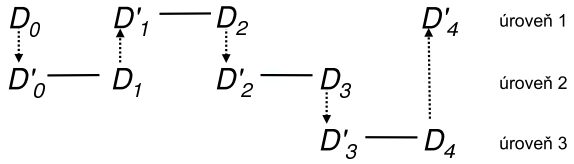
- horizontální úsečku, pokud spojuje některou z dvojic uzlů (D_i, D_{i+1}), resp. (D'_i, D_{i+1}), či (D''_i, D_{i+1}), ... , kde $D_i, D'_i, D''_i, D_{i+1} \in D, i \in \{0, ..k-1\}$
- tečkovanou vertikální šipku, pokud spojuje uzel D_i a jeho kopii D'_i, příp. D''_i,... $(0 \leq i \leq k)$

Tato pravidla (či vertikální uspořádání uzlů) dovolují mluvit o **úrovních schématu** – nejvyšší úroveň označujeme jako úroveň 1, dále úrovně číslujeme vzestupně.

V segmentačním schématu je tedy každý separátor D_i $(0 \leq i \leq k)$ reprezentován alespoň jedním uzlem. Separátor D_0 je vždy na úrovni 1, úroveň všech dalších uzlů D_i $(1 \leq i \leq k)$ je dána následujícím předpisem. Pokud je D_i otevírací separátor na úrovni l, může být vytvořena jeho kopie D'_i a umístěna na úroveň $l+1$ přímo pod uzel D_i; dále je vytvořena šipka (D_i, D'_i).[4] Pokud je D_i uzavírací separátor na úrovni l, můžou být vytvořeny jeho kopie $D'_i, D''_i, ...$ na úrovních $l-1, l-2, ...$ a příslušné šipky na všech úrovních až do nejvyšší úrovně 1. Horizontální hrany, které odpovídají jednotlivým segmentům věty W_i $(1 \leq i \leq k)$, spojují vždy uzel D_{i-1}, resp. jeho kopii D'_{i-1}, s uzlem D_i (a tím tedy určují úroveň D_i).

[4] Zatím neumíme vyloučit strukturu, kdy po jediném (složeném) separátoru následuje hlouběji zanořená klauze (i když jsme doposud žádnou takovou větu nenašli). Proto umožňujeme vytvořit i více kopií, tedy $D'_i, D''_i, ...$ na úrovních $l+1, l+2, ...$ a šipku $(D_i, D''_i)(...)$.

Ukažme si příklad segmentačního schématu na českém souvětí (obr. 2) *Zatímco neúspěch bývá sirotkem, úspěch mívá mnoho tatínků, horlivě se hlásících, že zrovna oni byli u jeho početí.*



Obrázek 2. Příklad segmentačního schématu.

D_0 - *Zatímco*
W_1 - *neúspěch bývá sirotkem*
D_1 - *,*
W_2 - *úspěch mívá mnoho tatínků*
D_2 - *,*
W_3 - *horlivě se hlásících*
D_3 - *, že*
W_4 - *zrovna oni byli u jeho početí*
D_4 - *.*

Na tomto příkladu je vidět, že úroveň 1 odpovídá hlavní klauzi souvětí (hlavní větě). Úroveň 2 reprezentuje klauzi rozvíjející (přímo) hlavní klauzi (tedy vedlejší větu rozvíjející hlavní větu souvětí) a další úroveň klauzi rozvíjející tuto vedlejší klauzi.

Pokud hlavní klauze obsahuje vnořenou vedlejší klauzi, zachycujeme druhou část hlavní klauze na stejné úrovni jako její část první, tedy na úrovni 1; to samé platí pro hlouběji zanořené vedlejší klauze, tedy podřízené klauze, které jsou vnořeny do (vedlejší) klauze, kterou rozvíjejí.

2.2 Krok 1: Segmentace věty

Metodu segmentace lze rozdělit na dva kroky. V prvním kroku (při prvním průchodu větou) na základě morfologické analýzy (v tomto projektu pracujeme s morfologickou analýzou popsanou v [3]) najdeme jednotlivé separátory a určíme jednotlivé segmenty. Tento krok sice obecně není deterministický (už proto, že pracujeme s víceznačnou morfologickou analýzou, viz dál), jde však o mechanické (jednoprůchodové) využití morfologické informace.

Přestože jednoduché separátory obvykle jednoznačně určíme pouze na základě morfologické analýzy, musíme i v tomto kroku počítat s víceznačností, a to jak morfologickou, tak 'funkční'. Příkladem morfologické víceznačnosti může být např. slovo *jak*, které je jednak substantivem (třeba ve větě *Jak je tibetský dlouhosrstý horský skot nápadný chrochtavým hlasem.*), jednak spojkou souřadicí (*Přišli jak mladí, tak*

staří.), dále spojkou podřadicí (*Díval se na jeho ruce, jak se třesou námahou.*), či zájmenným příslovcem (*Ptal se, jak se jim tam líbí. Jak je starý, tak je hloupý.*). Funkčně víceznačná je např. tečka, která je na konci věty separátorem, za řadovou číslovkou nikoli; podobně uvozovky označující přímou řeč separátorem jsou a uvozovky použité např. pro grafické zvýraznění separátorem nejsou.

Určením separátorů a rozdělením věty na jednotlivé segmenty končí první krok segmentace. Lze nahlédnout, že tento krok lze považovat za označkování lexikálních jednotek vstupní věty, které provádí konečný převodník (gsm), v obecném případě s nedeterministickým (víceznačným) výstupem.

2.3 Krok 2: Určování segmentační struktury věty

Druhý krok spočívá ve vytvoření segmentační struktury dané věty reprezentované (alespoň jedním) segmentačním schématem. Tento krok je komplikovanější, a to kvůli typické víceznačnosti zejména kombinovaných separátorů. Nejen že např. interpunkční čárka může být interpretována tak, že následující segment je umístěn na úrovni vyšší (tedy čárka uzavírá vloženou klauzi), na úrovni stejné (v případě koordinace) nebo na úrovni nižší (pokud následuje klauze rozvíjející předchozí segment), čárka může ukončovat i hlouběji vnořenou klauzi, čemuž odpovídá přechod v segmentačním schématu o dvě (případně i více) úrovně výše. Pokud se tedy takový víceznačný separátor ve větě objeví, je nutné vytvořit více segmentačních schémat tak, aby byly pokryty všechny případy.

Určení relevantních segmentačních struktur vyžaduje podrobnou lingvistickou analýzu a klasifikaci separátorů a jejich možných funkcí v souvětí. V následující sekci ukážeme, že i jednoduchá sada pravidel pro segmentaci, která využívají pouze lokální morfologickou informaci a nepředpokládají porozumění větě, může pomoci při odhadu syntaktické struktury vět přirozeného jazyka.

3 Základní sada segmentačních pravidel

Základní sada (heuristických) segmentačních pravidel nám umožní ukázat princip budování segmentačního schématu pro české věty. Pro jejich formulaci ještě definujme, co považujeme za příznak podřízenosti (viz sekce 2.1).

3.1 Příznak podřízenosti

Příznak podřízenosti se přiděluje jednotlivým rozšířeným segmentům, pokud obsahují slovní formu s morfologickou charakteristikou (viz [3]) danou následující tabulkou 1, resp. pokud jde o příslovce dané výčtem v této tabulce.

tag	popis	výčet
J,	podřadicí spojka	
P4	tázací/vztažné zájmeno	*jaký, který, čí, ...*
PE	”	*což*
PJ	”	*jenž, jíž, ...*
PK	”	*kdo, kdož, kdožs*
PQ	”	*co, copak, cožpak*
PY	”	*oč, nač, zač*
C?	číslovka	*kolik*
Cu	”	*kolikrát*
Cz	”	*kolikátý*
D.	zájmenné příslovce	*jak, kam, kde, kdy, proč*

Tabulka 1. Morfologická značka (tag) charakterizující příznak podřízenosti (1. a 2. pozice, kde tečka (.) znamená lib. hodnotu).

V následujících pravidlech užíváme pro příznak podřízenosti zkratku PP.

3.2 Základní sada pravidel

Začátek věty: Uzel reprezentující D_0 je vždu umístěn na úrovni 1.
Nemá-li první segment W_1 přiřazen PP, je hrana reprezentující tento segment na úrovni 1.

Čárka: Pokud je otevíracím separátorem D_{i-1} čárka a následující segment W_i nemá PP, je tento segment W_i umístěn na stejné úrovni jako předchozí segment W_{i-1}, NEBO jsou vytvořeny kopie $D'_{i-1}, ...$ o 1 úroveň (případně o více úrovní) výše a segment W_i je umístěn na těchto úrovních.

Čárka a segment s PP: Následuje-li za čárkou D_{i-1} segment s PP, je vytvořena kopie separátoru D'_{i-1} o úroveň níž a na tuto úroveň umístěn segment W_i.[5]

Koordinační výrazy: Koordinační spojky a další koordinující výrazy (např. pomlčka, dvojtečka, středník) zachovávají úroveň, a to i když následuje PP.

Konec věty: Poslední separátor D_k reprezentuje konec věty, tudíž není-li již na úrovni 1, vytvoří se příslušný počet kopií $D'_k, ...$ až na úroveň 1.

Otevírací uvozovky: Otevírací uvozovky jsou považovány za separátor, pouze pokud jsou na začátku věty nebo následují za jiným separátorem (čárka, středník apod.), dále pokud se mezi otevíracími a příslušnými uzavíracími uvozovkami nachází finitní sloveso.

[5] Toto pravidlo neplatí obecně, např. ve větě *Řekl, že byl, jaký byl* **, že** *je, jaký je a že bude, jaký bude.* kombinovaný separátor *', že'* uzavírá vnořenou větu a tedy následující segment patří o úroveň výš. Tento typ konstrukcí je potřeba řešit pomocí speciálních podmínek.

Pokud následující segment neobsahuje PP, je umístěn o 1 úroveň níž; pokud PP obsahuje, je umístěn o 2 úrovně níž.

Uzavírací uvozovky: Uzavírací uvozovky jsou považovány za separátory, pouze pokud příslušné otevírací uvozovky jsou separátory.
Následující segment je umístěn na úroveň segmentu, který předcházel uvozovkám.

Otevírací závorka: Otevírací závorky jsou separátory.
Pokud následuje segment bez PP, je umístěn o 1 úroveň níž; pokud PP obsahuje, je umístěn o 2 úrovně níž.

Uzavírací závorka: Uzavírací závorka je separátorem, pouze pokud příslušná otevírací závorka je též separátorem.
Následující segment je umístěn na úroveň segmentu, který předcházel otevírací závorce.

4 Předběžné vyhodnocení pravidel

Ukázalo se, že vyhodnocení této metody je poněkud složitější, než jsme předpokládali. Počítali jsme s využitím syntakticky anotovaného Pražského závislostního korpusu (PDT)[6], zejména tzv. analytické roviny. Ukazuje se ale, že tento bohatý zdroj dat nelze přímočaře využít pro vyhodnocení (přestože ho masivně využíváme pro vyhledávání příkladů a ověřování hypotéz).

Problémem je paradoxně bohatost anotace – v PDT je zachyceno velké množství syntaktických jevů, pro které je nesmírně obtížné najít konzistentní způsob anotace. Při návrhu anotačního schématu bylo nutno přijmout mnoho ad hoc rozhodnutí týkajících se anotace některých složitých jevů, jako je zejména koordinace, ale například i zachycování složených slovesných tvarů či předložkových skupin.

Tato šíře a způsob anotace může vést k problémům, pokud chceme v korpusu vyhledávat jevy, s jejichž zachycením anotační schéma nepočítalo. Příkladem může být tak jednoduchý jev, jako určení obsahu závorek. Vzhledem k tomu, že u závorek jednoznačně určíme, zda jde o otevírací nebo uzavírací separátor, je přirozené předpokládat, že takto snadno vymezitelný segment bude anotován jedním způsobem.

Ukazuje se však, že na analytické rovině PDT tomu tak není – již při zkoumání malého vzorku dat se ukázalo, že závorky jsou anotovány alespoň 7 různými způsoby v závislosti na kontextu. Dvě z těchto možností lze vidět na obr. 3, kde jsou příslušné podstromy pro větu *Před několika dny vypukl další skandál (privatizace Čokoládoven v Modřanech), v němž byl do role hlavního viníka opět obsazen Fond národního majetku (FNM) a jeho předseda Tomáš Ježek.*

Na obr. 3 je vidět, že anotace závorek se liší, navíc dokonce i vzájemná pozice otevírací a uzavírací závorek je odlišná (sourozenci v 1. podstromu, rodič a potomek v 2. podstromu). Je tedy zřejmé, že převod vět z PDT by vyžadoval velké množství manuální práce (kterou by vzhledem k mnoha způsobům anotace šlo jen částečně automaticovat), abychom získali vhodná testovací data.

Tyto úvahy vedly k rozhodnutí anotovat ručně malý vzorek dat podle jejich segmentační struktury. K této anotaci byly vybrány dva české novinové články z Lidových novin a Neviditelného psa[7] (dále LN, resp. NP), které obsahují politické komentáře. Následující tabulka 2 udává stupeň víceznačnosti segmentačních struktur při simulaci automatické procedury využívající pravidla popsaná zde v sekci 3.

	počet			počet segm. struktur					
	vět	'tokenů'	segmentů	1	2	3	4	5	více
LN	33	553	78	28	2	1	1	1	-
NP	15	334	57	12	3	-	-	-	-
total	48	887	135	40	5	1	1	1	-

Tabulka 2. Stupeň víceznačnosti segmentačních schémat.

Přestože tato testovací data jsou relativně malá, tabulka 2 jasně ukazuje, že i takto jednoduchá sada pravidel založených pouze na morfologické analýze, je velmi slibným začátkem a že víceznačnost segmentačních schémat je u reálných textů poměrně nízká. Nejdůležitější výsledkem testu je ovšem 100% pokrytí – pomocí navržených pravidel byla získána všechna správná segmentační schémata, žádné nebylo opomentuto.

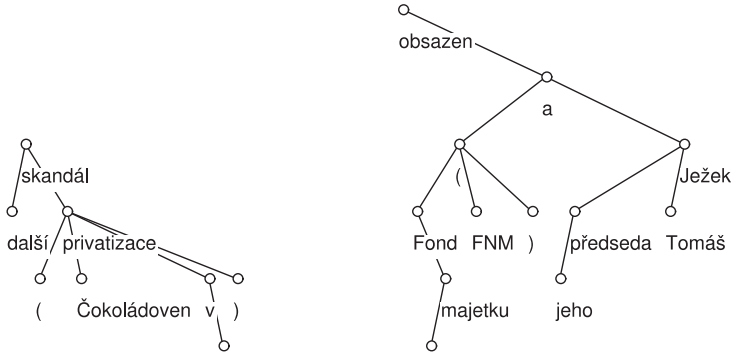
Je samozřejmě možné najít věty s komplikovanou strukturou, pro které tato jednoduchá pravidla selžou (vytvoří velké množství neadekvátních segmentačních schémat nebo naopak správné schéma neodhalí), předpokládáme však, že další zjemňování pravidel bude výsledky zlepšovat.

Závěr

Metoda segmentace popisovaná v tomto článku ukazuje, že je možné pro flektivní jazyky s relativně pevnými pravidly pro používání interpunkce odhadnout schéma reflektující vzájemný vztah klauzí a jejich částí (segmentů) v souvětí bez úplné syntaktické analýzy celé věty, pouze na základě lokální morfologické informace. Tato metoda je založená na identifikaci a klasifikaci tzv. separátorů členících větu do jed-

[6] http://ufal.mff.cuni.cz/pdt2.0/

[7] http://pes.eunet.cz



Obrázek 3. Dva typy anotace závorek v PDT.

notlivých segmentů a na relativně jednoduchých pravidlech pro odhad struktury klauzí. Článek zároveň ukazuje přímou návaznost na tradiční českou syntax, kterou budeme využívat i při dalším vývoji této metody.

Dalším krokem, na který se chceme soustředit v blízké budoucnosti, je ohodnocení segmentů (na základě morfologické analýzy), které by mohlo pomoci při odstranění nerelevantní víceznačnosti segmentačních schémat a při zpřesňování rozsahu jednotlivých klauzí souvětí.

V delším horizontu chceme využít segmentační schémata pro zvýšení rychlosti a efektivity při syntaktické analýze souvětí.

Testování segmentace popsané v tomto článku dává dobrý předpoklad, že tato metoda skutečně může pomoci při analýze souvětí, víceznačnost segmentační struktury je v reálných textech poměrně řídká a přitom jsou všechna správná segmentační schémata vytvořena.

Poděkování

Výsledky, o kterých referujeme v tomto článku, byly dosaženy za podpory grantu Informační společnosti GA AV č. 1ET100300517.

Reference

1. Holan T., Kuboň V., Oliva K., Plátek M., On Complexity of Word Order. In: Les grammaires de dépendance – Traitement automatique des langues, Vol 41, No 1, pp. 273-300, 2000
2. Holan T., Dependency parser, proGRAM. `http://ksvi.mff.cuni.cz/~holan/proGRAM.html`
3. Hajič J.: Disambiguation of Rich Inflection (Computational Morphology of Czech). UK, Nakladatelství Karolinum, Praha, 2004
4. Hajič J., Vidová-Hladká B., Zeman D., Core Natural Language Processing Technology Applicable to Multiple Languages. The Workshop 98 Final Report. Center for Language and Speech Processing, Johns Hopkins University, Baltimore, 1998
5. Jones B.E.M., Exploiting the Role of Punctuation in Parsing Natural Text. In: Proceedings of the COLING'94, Kyoto, University of Kyoto, 1994, 421–425
6. Kuboň V., A Robust Parser for Czech. UFAL Technical Report TR-1999-06, Charles Univesity, Prague, 1999
7. Kuboň V., Problems of Robust Parsing of Czech. Ph.D. Thesis, MFF UK, Prague, 2001
8. Kuboň V., A Method for Analyzing Clause Complexity. The Prague Bulletin of Mathematical Linguistics 75, 2001, 5–27
9. Kuboň V., Lopatková M., Plátek M., Pognan P., Segmentation of Complex Sentences, přijato na TSD 2006
10. Lešnerová-Zikánová Š., Oliva K., Česká vztažná souvětí s nestandardní strukturou. In: Slovo a slovesnost. Roč. 64, č. 4, 2004, 241–252
11. Oliva, K., A Parser for Czech Implemented in Systems Q. In: Explizite Beschreibung der Sprache und automatische Textbearbeitung, MFF UK Praha, 1989
12. Plátek M., Composition of Translation with D - trees, COLING' 82, 1982, 313–318
13. Plátek M., Sgall P., A Scale of Context-Sensitive Languages: Aplication to Natural Language. Information and Control, Vol. 38., No 1., July 1978
14. Šmilauer V.: Učebnice větného rozboru. SPN, Praha, 1958
15. Zeman D.: Parsing with a Statistical Dependency Model. Ph.D. Thesis. MFF UK, Prague, 2004

Klasifikace vzorů v širších souvislostech

Miloš Kudělka, Ondřej Lehečka, and Václav Snášel

VŠB – Technická univerzita Ostrava
17. listopadu 15, 708 33, Ostrava-Poruba,
{Milos.Kudelka,Ondrej.Lehecka,Vaclav.Snasel}@vsb.cz

Abstrakt *V článku se zamýšlíme nad vzory a doménami, ve kterých se vyskytují. Řešíme také otázku, jak spolu vzory z různých domén mohou souviset při řešení komplexnějších úloh. Předpokladem, který si dáváme při řešení nám neznámého nebo málo známého problému je, že v doméně tohoto problému mohou existovat nebo spíše existují vzory. Základní úlohou je pro nás efektivní orientace ve velkém množství vzorů a návrh mechanismu, který umožní nalézt ve stovkách existujících vzorů ty, které mohou být užitečné pro řešení konkrétního problému.*

1 Co je vzor

V sedmdesátých letech byly architektem Christopherem Alexandrem formulovány vzory pro řešení úloh v architektuře (viz [1]). Vzor je formulován ne jako nápad, jak řešit problém, ale jako popis a zobecnění určité zkušenosti, která vede k postupu, jak problém řešit (východiskem vzorů je praktická zkušenost, nikoli pouhý nápad, jak problém řešit). Návod řešení pak musí být tak pružný, aby se vzor dal aplikovat opakovaně a aby přitom výsledky nebyly stereotypní. Vzor je stručný text, který popisuje náčrt problému a popis jeho obecného řešení v konkrétních souvislostech.

> Alexandrova formulace vzoru: „*Každý vzor je pravidlo, které vyjadřuje vztah mezi jistou souvislostí, problémem a řešením.*“

Vzory jsou tedy opakovatelné, úspěšné a osvědčené postupy, které vedou k řešení konkrétního problému a které byly empiricky potvrzeny. Autoři vzorů kategorizují vzory podle různých aspektů oblasti – domény, kterou dané vzory pokrývají. Existující klasifikace vzorů jsou vhodná pro studium a orientaci v dané doménové oblasti.

2 Domény vzorů

V článku (viz [3]), který se zabývá především oblastí uživatelského rozhraní a komunikace člověka s počítačem, autor formuluje několik tezí, z nichž některé můžeme zobecnit a použít i v širším kontextu. Jedná se o to, že

- vzory potřebují empirické důkazy pro opodstatněnost jejich užití,
- vzory musí být čitelné pro jejich uživatele,
- vzory mohou modelovat mnoho aplikačních domén,
- použití vzorů v softwarové architektuře, uživatelském rozhraní a aplikační doméně projektu může zlepšit komunikaci v interdisciplinárních vývojových týmech.

Za uživatele lze v kterékoli ze zmíněných oblastí považovat osoby, které v této oblasti pracují a mají zkušenosti s úspěšnými řešeními problémů. Vzory lze tedy formulovat ve všech oblastech, které se nějak dotýkají softwarového řešení (a také kdekoli jinde). Můžeme např. začít od popisů aplikačních domén, přes popisy úloh a interakcí v uživatelském rozhraní, po popis architektury, návrhu a implementace systému. Pokud by v každé z těchto oblastí byly popsány vzory, návrháři počítačových systémů by si asi lépe porozuměli. V týmech by se lépe provazovaly, předávaly a používaly znalosti.

Na následujících příkladech bychom chtěli ukázat, že má smysl se vzory pracovat v širším kontextu. Bude se jednat jak o vzory všeobecně známé ve vývojářské komunitě, tak méně známé a zaměřené do jiných oblastí – domén.

Návrhové vzory. Mezi vývojáři nejznámější skupina vzorů. Většina z nich je popsána v knize [11]. Aplikují se při vytváření návrhu a jsou zaměřeny především na zajištění flexibility programu, tj. jejich schopnosti snadno se přizpůsobit přicházejícím změnám.

Analytické vzory. V této oblasti bývá zmiňována především kniha [9]. Analytickými vzory autor rozumí takové vzory, které jsou víceméně nezávislé na problémové doméně a popisují řešení problémů na konceptuální úrovni, která je minimálně závislá na budoucí implementaci.

Vzory pro rozsáhlá řešení. Oblast, které je věnována v poslední době značná pozornost, a to i ze strany komerčních firem. Jde o vytváření aplikací velkého rozsahu, například na úrovni podnikových informačních systémů. Nejlepšími zdroji pro tuto oblast jsou knihy [10] [23].

Vzory pro integraci. Integrace systémů je jednou z aktuálních úloh současné doby. Ani této oblasti se nevyhnula snaha o formulování vzorů. Zmiňována je především kniha [14], která tuto oblast popisuje velmi důsledně.

Vzory pro testování. Zajímavým, velmi stručným a čitelným katalogem vzorů je [6] kde se autor zabývá vzory pro testování jednotek kódu.

Vzory pro správu softwarových konfigurací. Ve spojení se správou verzí a s dalšími úlohami s tím spojenými vzniká potřeba dobře popsat, jak tyto úlohy řešit (viz např. [2]).

Vzory pro tvorbu uživatelského rozhraní. Grafické uživatelské rozhraní je z pohledu uživatele nejdůležitějším prvkem aplikace a jako s takovým se s ním musí zacházet. Důležitou vlastností dobře navrženého uživatelského rozhraní je dodržování ergonomických zásad a standardů, které lze popsat prostřednictvím vzorů (viz [22] [13] [15]).

Vzory pro e-learning. Katalog vzorů na [8] slouží k popisu vlastností, které jsou očekávány u elearningového systému a jeho obsahu. Vzory byly nalezeny v profesionálních systémech a jako takové na obecné úrovni popisují systémy jako sobě rovné.

Pedagogické vzory. Při výuce se učitel i posluchač časem dostává do stejných situací. Vzniká tedy určitý stereotyp – daný problém se řeší obdobně, jistou metodou. Takto vnímané metody se dají popsat prostřednictvím vzorů, které lze najít především na [20].

Vzory pro návrh zákaznicky orientovaných webů. V knize [7] je popsáno mnoho vzorů, které lze použít při návrhu a implementaci komerčních webových stránek.

Vzory pro návrh počítačových her. Ani oblasti vývoje počítačových her se nevyhnuly snahy o nalezení vzorů. V knize [3] se autoři zaměřují na studium společných rysů počítačových her a vytvoření jazyka vzorů pro vývojáře her.

Vzory pro vývoj vyhledávacího rozhraní. Moderní informační systémy vyžadují jednoduchý způsob zpřístupnění dat prostřednictvím zadání vyhledávací podmínky a přehledné zobrazení výsledků vyhledání. I zde lze popsat efektivní řešení prostřednictvím vzorů (viz [24]).

Vzory pro psaní vzorů. Nakonec v tomto zcela jistě neúplném seznamu oblastí pokrývaných vzory uvádíme materiál, ve kterém se autoři podrobně zabývají tím, jak vzory hledat a jak je správně formálně popsat [18].

3 Klasifikace vzorů

V následující části se budeme zabývat pouze vzory, které jsou uváděny v oblasti softwarového inženýrství. Důvodem je poměrně vysoká propracovanost klasifikací v této oblasti.

Vzorů jsou stovky a po desítkách se vyskytují v různých doménách. S postupem času se detailněji popisují osvědčená řešení a vzorů přibývá. Z tohoto ohledu se jeví jako klíčový problém provést klasifikaci, která pomůže uživateli v lepší orientaci. Ke klasifikaci lze přistoupit různě. Nejzákladnější rozdělení vzorů je podle oblasti použití. Další rozdělení zpravidla bývá podle problémových domén v dané oblasti použití. Další rozdělení záleží na fantazii, potřebách a úhlu pohledu. Každý autor publikace o vzorech si zpravidla vytvoří vlastní klasifikaci.

4 Jazyky vzorů

Vzhledem k tomu, že vzor se nikdy nevyskytuje v daných souvislostech osamoceně, bývají souvislosti popsány řečí, jazykem vzorů. Vzory pro danou souvislost tvoří uzavřený celek, ve kterém jsou organizovány do návazností a skupin, které mohou mít např. společný rozsah. Jazyk vzorů není jen katalog vzorů, ale je to komplexní pohled na použití vzorů v jedné (byť široké) oblasti. Vzory bývají aplikovány v sekvencích, kontext jednoho vzoru může být důsledkem jednoho nebo více jiných vzorů.

Jazyk vzorů tedy přesně definuje kontext, vzory v tomto kontextu a vztahy mezi nimi. (Podobně jako např. čeština poskytuje mechanismus skládání vět ze slov, jazyk vzorů umožňuje složit řešení složitějšího problému s jednotlivých vzorů). Důležitým faktorem pro popis vzorů je to, že i přesto, že kontext může být velmi rozličného rozsahu, rozsah jejich popisu zůstává zhruba stejný (v rámci jedné nebo několika stran). Tento podstatný rys umožňuje na relativně krátkém prostoru dát k dispozici jazyk, který pokrývá určitou oblast a umožní dobře formulovat problémy, souvislosti a řešení, které se v této oblasti vyskytují. Typickými příklady jazyků vzorů jsou např. návrhové vzory, vzory pro testování a vzory pro elearning.

4.1 Katalogy podle autora

Základní klasifikaci vzorů obvykle provádí autor nebo skupina autorů s jasným úmyslem poskytnout čtenáři dobře organizovanou pomůcku pro jeho práci. Dále uvádíme známé příklady.

4.2 Gang of Four

Do širšího povědomí se návrhové vzory dostaly v roce 1995 po vydání knihy [11], která je dnes označována zkratkou *GoF* (Gang of Four). Je to první dobře popsaný a zdokumentovaný katalog návrhových vzorů. Obsahuje 23 vzorů, které jsou rozděleny do tří kategorií.

– Vytvářecí vzory
– Strukturální vzory
– Vzory chování

Toto základní členění umožňuje čtenáři základní orientaci při hledání vzorů a přirozeným způsobem poskytuje skupiny souvisejících postupů pro řešení typických úloh.

4.3 Martin Fowler

V knize [10] jsou uvedeny vzory pro návrh a implementaci rozsáhlých informačních systémů. Na návrh se můžeme podívat z různých hledisek, která odpovídají např. úrovni abstrakce pohledu na systém či pohledu na způsob implementace některé konkrétní části systému. Autor definuje skupiny vzorů, které popisují zkušenost s návrhem či způsobem implementace společného problému. Tento problém pak může pokrývat například spolupráci prvků v různých vrstvách systému (obecněji řečeno spolupráci vrstev samotných) nebo například návrh struktur a chování objektů v jediné vrstvě celého systému. V rámci každé skupiny pak jsou jednak vzory, které lze použít společně pro řešení jednoho problému, jednak vzory, které popisují řešení stejného problému různými způsoby (v závislosti na rozsahu úlohy, která se řeší). Vzory autor dělí do následujících skupin:

– vzory pro zajištění doménových logik,
– architektonické vzory pro datové zdroje,
– objektově-relační vzory
  • pro chování objektů,
  • strukturální,
  • mapovací,
– prezentační vzory pro web,
– vzory pro distribuované úlohy,
– vzory pro souběžný přístup k datům,
– vzory pro „Session State“,
– základní vzory.

Důslednou snahou pak je, aby vzory mohl být pokryt celý vyvíjený systém.

4.4 Microsoft

Aplikaci podobného pohledu jako v předchozím odstavci můžeme najít i u velkých komerčních firem, jako je např. Microsoft. Ta ve své knize [23] ve velké míře vychází ze stejného principu a rozpracovává některé vzory zmíněné výše až na technologickou a implementační úroveň. Rozděluje vzory do skupin ve třech směrech. Prvním je rozdělení do tzv. *clusterů* (webová prezentace, nasazení, distribuované systémy, výkonnost a spolehlivost, služby). Druhým je *úroveň abstrakce* (architektura, návrh, implementace). Třetím je pak tzv. *hledisko* – viewpoint (databáze, aplikace, nasazení, infrastruktura). Tyto tři směry pak existují paralelně a v jejich průnicích existují tzv. *rámce*, v každém z nich je potom skupina vzorů pokrývající detailně řešení konkrétního problému. Takový pohled poskytne vývojáři mnohem přesnější informace o tom, kam pro vzor sáhnout, ví-li, jaký problém v rámci celého systému řeší.

5 Katalogy na internetu

Základním nedostatkem vzorů uvedených v knihách je jejich obtížné vyhledávání při řešení problému. Proto vznikají na internetu katalogy a portálová řešení s vyhledávací podporou a například s ukázkovými řešeními a knihovnami. Opět uvádíme některé příklady.

5.1 MS Patterns & Practices

Tato klasifikace vzorů vychází z knihy [23], která obsahuje vzory pokrývající téměř celý *proces vývoje software* na platformě Microsoft. Část vzorů jsou tzv. implementační vzory, popisují praktické zkušenosti autorů při použití daného vzoru v prostředí Microsoft .NET. V knize je obsažena škála vzorů od návrhových, implementačních až po vzory infrastruktury a nasazení. Dále jsou zde vzory rozděleny podle úrovně abstrakce. Na základě tohoto rozdělení autoři vytvořili organizační tabulku vzorů, která kategorizuje vzory podle několika pohledů. Důležité je, že pohled na vzory a jejich organizaci je vytvářen usazováním názorů členů komunity.

5.2 www.patternshare.org

Autoři webu se snaží vývojářům pomoci především tím, že organizují vzory především do tabulek podle známých klasifikací a autorů. Kromě toho poskytují možnost vkládání nových vzorů a vedení diskuzí. Tento systém se opět zabývá pouze vzory pro softwarová řešení a umožňuje kategorizovat vzory pouze do několika pevných kategorií.

6 Problémy klasifikace

Základními rysy současných klasifikací je to, že jsou zaměřeny na jednu doménu. Pokud bychom se chtěli na vzory podívat „přes domény“, současné klasifikace nemohou dostačovat.

Každý autor knihy o vzorech se snaží rozdělit množinu vzorů, kterými se zabývá, do několika tříd. Tyto třídy nebo kategorie vzorů jsou závislé na pohledu autora na oblast, kterou vzory řeší. Taková kategorizace vzorů umožňuje lepší orientaci v dané oblasti

a také v knize, která popisuje vzory pro danou oblast. Pokud se např. zabýváme vzory pro integraci, je velice užitečné rozdělit integrační vzory podle toho, jestli řeší komunikaci mezi aplikacemi, topologii, spojování systémů apod. Na základě této kategorizace se můžeme velice rychle orientovat ve vzorech týkajících se problému integrace systémů.

Studium vzorů vyžaduje nějakou jinou kategorizaci vzorů, a to zejména v okamžiku, kdy např. přesně nevíme, v jakém kontextu máme hledat. Každý vzor by měl kromě popisové části, která slouží k nastudování podrobností, obsahovat i kategorizační část. V *popisové části* se čtenář dozví, jak se vzor jmenuje, jakého problému se týká, v jakém kontextu se může vzor použít, jak má vypadat řešení, jaké to má důsledky apod. *Kategorizační část* by měla řešit klasifikaci vzorů a měla by poskytnout informaci, do jakých kategorií vzor patří, tzn. například, že vzor je integrační a zabývá se topologií systémů. Obě tyto části popisu vzoru jsou důležité. Je jasné, že popisová část je takřka fixní, popisuje samotný vzor, nemění se s úhlem pohledu. Zatímco kategorizační část může být předmětem diskuse, může se měnit podle různých úhlů pohledu na vzor.

> *Důsledek: Klasifikace vzorů v každé knize je užitečná pro samotné studium a pro orientaci v dané problémové oblasti.*

Tak, jako popularita používání vzorů roste, přichází další a další autoři zabývající se různými problémovými oblastmi, tak i počet nalezených vzorů neustále roste. Dnes máme zdokumentované vzory v různých oblastech lidské činnosti. Např. v softwarovém inženýrství existuje takové množství vzorů, že nastudovat všechny je téměř nadlidský úkol.

Člověk se neustále v životě dostává do situací, kdy musí řešit problémy, se kterými se doposud nesetkal. Tyto problémy začne řešit buďto svými vlastními silami a nebo si uvědomí, že je zbytečné vymýšlet vymyšlené a použije existující vzory. Použít vzor znamená použít úspěšné a osvědčené řešení na daný problém. Tento způsob rozhodně minimalizuje možný neúspěch. Ten, kdo zná vzory a má čas na jejich studium si najde příslušnou knihu pro danou doménovou oblast a vzory si nastuduje. Tento přístup ale v praxi nemá moc šancí na úspěch, neboť na plošné studium nezbývá mnoho času. A nemá ani smysl studovat celý katalog.

Pokud člověk nezná vzory pro danou problémovou doménu, pak mu ani existující dostupné klasifikace vzorů nemusí při jejich vyhledání pomoci. Pokud nezná vzory z této domény ani orientačně, pak je těžké jeden konkrétní nalézt nebo alespoň zjistit, do které kategorie má patřit. V takovém případě lze pouze popsat daný problém na základě jeho vlastností. Pokud například víme, že hledáme vzor pro zvýšení nezávislosti výpočtu na použitém algoritmu programu a neznáme návrhové vzory, konkrétně třeba vzor *Strategy*, těžko nám pomůže informace, že vzor patří do skupiny vzorů chování.

> *Důsledek: Existující klasifikace vzorů nepomáhají při hledání neznámého vzoru na daný problém.*

Pokud se člověk věnuje studiu vzorů delší dobu, tak si za nějakou dobu uvědomí, že existují vztahy mezi různými vzory z různých doménových oblastí. V poslední době vznikají nové publikace, které dávají dohromady vzory z různých oblastí pro řešení nějaké rozsáhlejší úlohy. Kupříkladu kniha [13] se věnuje problematice webových prezentací z pohledu použitelnosti, obsahu, navigace a estetiky. Představuje téměř 80 vzorů pro návrh webu. Problematikou návrhu a tvorby webů se zabývá i kniha [7], která také spojuje vzory z více oblastí.

> *Důsledek: Žádná existující klasifikace vzorů nedává přehled o tom, jak vzory z různých oblastí souvisí mezi sebou.*

7 Náš přístup

Vzory se ukazují jako jednoduchý a čitelný způsob záznamu osvědčené zkušenosti. Jako klíčový problém se nám jeví způsob nalezení existujícího vzoru, který řeší některou část rozsáhlého problému. Našim cílem je poskytnout techniky a nástroje k tomu, abychom mohli při znalosti problému, který řešíme, co nejefektivněji nacházet vzory. Přitom předpokládáme, že pracujeme na problému, který přesahuje jednu doménu. Může se jednat například o implementaci, nasazení a provoz e-learningového systému. Pak můžeme použít např. vzory pro e-learning, pedagogické vzory, analytické a návrhové vzory, vzory pro architekturu a integraci, vzory pro nasazení a testování apod.

7.1 Úlohy, které jsme řešili

Vytvoření flexibilní klasifikace. Tato klasifikace by měla vystihnout vlastnosti, které daný vzor svým řešením poskytuje. Klasifikace musí být otevřená a flexibilní tak, aby se mohla přizpůsobit novým pohledům a novým klasifikacím vzorů.

Webový portál. Cíle této práce by měla naplňovat aplikace dostupná všem potenciálním uživatelům. Tato aplikace je připravená pro lidi zabývající se (především) softwarovými řešeními. Aplikace by měla být komunitě zaměřená a měla by sloužit jako počáteční místo pro nalezení vzorů, nikoli pro detailní studium.

Vyhledávací aparát. Vyhledávání by mělo umožnit najít vzory podle ohodnocení klasifikačními atributy. Tím se získá základní množina vzorů splňující dané vlastnosti. Mechanizmus vyhledávání by měl umožňovat vyhledávání vzorů vzhledem na konkrétní situaci, ve které se uživatel nachází.

Navigace v prostoru vzorů. V prostoru vzorů by měla existovat navigační struktura typu graf. Ta by měla umožnit slučování vzorů s podobnými vlastnostmi do uzlů a pohyb mezi uzly. Tím se uživatel může dostat do nějakého uzlu, resp. do nějaké konkrétní množiny vzorů, které řeší daný problém v daném kontextu (viz např. [15]).

7.2 Vytvoření flexibilní klasifikace

Klasifikaci vzorů jsme postavili na množině kategorizačních atributů a jejich přiřazení vzorům. Kategorizační atributy vytvářejí klasifikační strukturu. Tato struktura má zohlednit vlastnosti vzorů. Kromě těch kategorizačních atributů, které vychází z uznávaných publikací, jsme založili další atributy, které vystihují jednotlivé vlastnosti vzorů. Tyto atributy jsou zjevné až po důkladnějším prostudování vzorů. Kategorizační atributy seskupené do skupin mohou vypadat takto: *Co vzor řeší – efektivita, flexibilita, propojení, robustnost*; *Úroveň znalostí – začátečník, středně pokročilý, odborník*. Vzory se dají opatřovat těmito atributy a tím zařazovat do požadovaných kategorií.

7.3 Webový portál

Podle uvedeného pohledu byla postavena webová aplikace sloužící jako webový katalog vzorů s podporou klasifikování vzorů a inteligentním vyhledáváním. Podpora klasifikace vzorů je postavena na možnosti zakládání kategorizačních atributů a možnosti opatřování vzorů těmito atributy. Aplikace umožňuje vyhledávat vzory klasickým databázovým a fulltextovým způsobem, i sofistikovaným způsobem filtrováním kategorizačních atributů.

7.4 Vyhledávací aparát

Běžný uživatel je zvyklý na klasické vyhledávání, proto i náš systém umožňuje vyhledávat vzory např. podle názvu a textového popisu. Dále vyhledávání v katalogu vzorů funguje tak, že stačí, když uživatel zadá, co o svém problému ví, a vyhledávací aparát mu vrátí iniciační množinu vzorů. Svůj dotaz může uživatel dále zpřesňovat, a tak se množina nalezených může přibližovat požadavkům.

7.5 Navigace v prostoru vzorů

Při orientaci v katalogu vzorů je důležité mít nástroj k vytváření skupin vzorů a mít možnost vidět vztahy mezi skupinami vzorů. K řešení tohoto problému se jeví jako vhodná shluková analýza. Kromě samotných shluků potřebujeme uživateli umožnit orientaci ve shlucích. Proto jsme vybrali formální konceptuální analýzu (viz např. [5]). Shluky vzorů se vytvářejí na základě společných atributů. Dále je možné se prostřednictvím přidání resp. odebrání atributu pohybovat mezi jednotlivými shluky vzorů (inteligentní navigace).

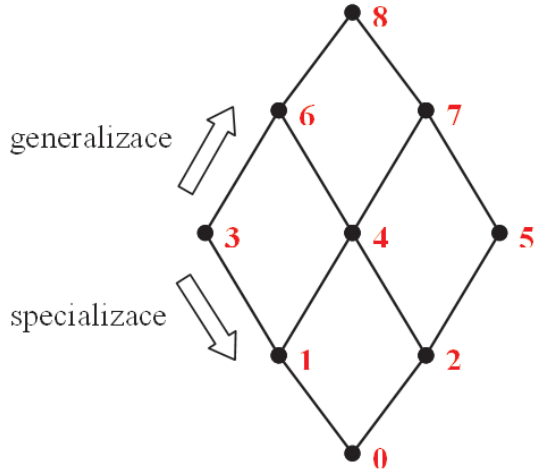
Vyhledávací aparát v kombinaci s navigací jsou navrženy tak, aby pomohly najít uživateli vzory s ohledem na konkrétní problém nebo situaci, ve které se nachází. V první fázi uživatel vyhledá vzory výběrem atributů, které vystihují jeho problém. Pro takto nalezenou sadu vzorů může uživatel najít nejbližší shluk vzorů (obsahující vzory se stejnými atributy). Další navigací může uživatel zpřesňovat a blíže specifikovat svoje požadavky a tím upravovat hledanou množinu vzorů.

8 Formální konceptuální analýza

Formální konceptuální analýza (FCA) je teorie zabývající se analýzou dat, ve kterých dokáže identifikovat konceptuální struktury. Tato teorie byla představena v roce 1982 Rudolfem Willem a od té doby byla rozvíjena a použita v mnoha odvětvích jako je medicína, psychologie, ekologie, strojírenství a ve spoustě dalších. Velikou výhodou formální konceptuální analýzy je schopnost vytvoření grafické prezentace struktur nalezených v datech.

Metodu FCA jsme použili pro shlukování vzorů a pro vytvoření navigační struktury na množině vzorů. Vstupními daty je matice vzorů a jejich atributů. Metoda FCA nám vypočítá koncepty neboli shluky. Každý shluk obsahuje množinu vzorů a množinu atributů. Shluk můžeme interpretovat tak, že nám dává množinu vzorů se stejnými vlastnostmi, které jsou vyjádřeny v atributech.

Navigační struktura je tvořena shluky vzorů a hranami spojující jednotlivé shluky. Z vlastností FCA vyplývá, že na dané struktuře se z každého shluku můžeme pohybovat dvěmi směry – „nahoru“ a „dolů“. Pokud půjdeme z daného shluku v navigační struktuře směrem dolů do jiného shluku, dostáváme se do shluku, který bude obsahovat více atributů než výchozí shluk. Navigaci směrem dolů můžeme chápat jako *konkretizační směr (specializace – viz obrázek 1)*, protože se dostáváme do shluků, které jsou definovány větším počtem atributů než předchozí, tzn. větším počtem kritérií, tzn. konkrétnější (menší) množinou vzorů.



Obrázek 1. Typy navigace.

Při navigaci směrem nahoru se dostáváme do obecnějších shluků, protože jsou definovány menším počtem atributů a tudíž větší množinou vzorů. Tento směr je *zobecňující směr (generalizace – viz obrázek 1).*

Pro implementaci FCA by použit zvolen algoritmus *Next Closure* (viz [12]), který pro námi požadovaný rozsah dostačuje. Vstupem pro algoritmus je kontext, což je incidenční matice vzorů a jejich atributů.

9 Experimenty

Prostudovali jsme desítky vzorů a konzultovali jsme jejich kategorizace se specialisty na vybrané domény. Na základě zjištěných informací jsme vytvořili skupiny atributů, které jsme několikrát revidovali. Skupiny atributů nejsou hierarchicky členěny, protože jeden atribut se obecně může vyskytovat ve více skupinách. Skupin je v této chvíli 14 a je pravděpodobné, že s dalšími vzory a analýzami dalších domén vznikne potřeba jejich rozšíření. Pro ukázku uvádíme 5 vybraných skupin atributů.

Jednotlivé vzory pak byly ručně opatřeny atributy, a to jednak podle domény a autora, a jednak individuálně podle skupin, do kterých patří.

Systém byl implementován formou portálového řešení v rámci diplomové práce (viz [17]) a je dále rozvíjen. Po implementaci jsme do systému vložili celkem 345 vzorů a 140 kategorizačních atributů. Pro plnění experimentální aplikace systému vzory jsme vytipovali vzory z různých domén. Většina z nich je z oblasti softwarového inženýrství, ale vloženy byly i některé další výše zmíněné skupiny vzorů. Konkrétně se jedná o *Vzory pro přístup k datům* [19], *Návrhové vzory*, *Vzory pro rozsáhlá řešení*, *Vzory pro integraci*, *Vzory pro uživatelské rozhraní*, *Pedagogické vzory*, *Vzory pro e-learning, Microsoft Patterns* [21]).

Skupina	**Atribut**
Vzor řeší efektivitu	recyklace objektů/zdrojů souběžný přístup k datům strategie přístupu ke zdrojům umožňuje škálování způsob výměny dat
Vzor řeší flexibilitu	nezávislost na algoritmu nezávislost na HW/SW platformě nezávislost na operaci rozšiřuje chování znovupoužití
Vzor řeší propojení	komunikaci mezi objekty spolupráci objektů propojení jednotlivých částí aplikace propojení aplikací mezi sebou objektově-relační mapování
Role	Architekt Návrhář Vývojář Tester
Úroveň znalostí	Expert Odborník Středně pokročilý Začátečník

Tabulka 1. Vybrané skupiny atributů.

Architektonický vzor	Vzor pro uživatelské rozhraní
Vzor: Model-View-Controller Zdroj: Enterprise Patterns (Fowler)	**Vzor: Overview Plus Detail** Zdroj: Designing Interfaces (Tidwell)
Atributy: Budování uživatelského rozhraní Vhodný pro webové aplikace Vhodný pro rozsáhlá řešení	**Atributy:** Budování uživatelského rozhraní Vhodný pro webové aplikace Organizuje obsah

Tabulka 2. Příklad kategorizačních atributů vzorů.

Pro formální konceptuální analýzu byl použit kontext velikosti 345 x 123, který je relativně řídký. V systému lze najít stovky různých shluků vzorů, mezi kterými se můžeme navigovat. Navigační struktura je struktura typu graf, která má dva význačné uzly – nejmenší a největší uzel. Po podrobnějším zkoumání grafové struktury svazu je vidět, že lze najít shluky odpovídající uvedenému členění podle autorů a jazyků vzorů a jejich zobecněním. Nicméně se ukázaly i zajímavé shluky vyjadřující např. souvislosti mezi návrhovými vzory a vzory pro rozsáhlá řešení nebo návrhovými vzory a vzory pro uživatelské rozhraní.

Ukazuje se potřeba permanentní revize atributů a potřeba vzniku nových atributů. V tomto ohledu

implementovaný systém nabízí jeden z klíčových rysů, kterým je možnost navrhovat, posuzovat a schvalovat nové atributy.

10 Závěr

Naší snahou je představit portál a jeho rysy širší odborné veřejnosti (aplikace je dostupná na adrese `www.pattron.net`). V našich experimentech se potvrdilo, že je velmi obtížné stanovit exaktní způsob návrhu atributů. Z testování systému studenty informatiky vyplývá, že naše řešení je dobrým nástrojem pro hledání vzorů. Zejména inteligentní navigace založená na formální konceptuální analýze poskytuje nový pohled na problematiku hledání vzoru pro řešený problém.

Reference

1. Alexander Ch., A Pattern Language: Towns, Buildings, Construction. New York. Oxford University Press 1977
2. Berczuk S.P., Appleton B., Software Configuration Management Patterns: Effective Teamwork, Practical Integration. Addison Wesley 2002
3. Björk S., Holopainen J., Patterns in Game Design. Charles River Media 2004
4. Borchers J., Interaction Design Patterns: Twelve Theses. Position Paper, Workshop Pattern Languages for Interaction Design: Building Momentum, CHI 2000. The Hague, Netherlands, April 2–3, 2000
5. Carpineto C., Romano G., Concept Data Analysis: Theory and Applications. Wiley, September 20, 2004
6. Clifton M., Advanced Unit Test, Part V - Unit Test Patterns. Code Project 2004. `http://www.codeproject.com/gen/design/autp5.asp` (31. dubna, 2006)
7. Van Duyne D.K., Landay J.A., Hong J.I., The Design of Sites: Patterns, Principles, and Processes for Crafting a Customer-Centered Web Experience. Pearson Education 2002
8. The E-LEN project. `http://www2.tisip.no/E-LEN/patterns_info.php` (May 31, 2005)
9. Fowler M., Analysis Patterns. Reusable Object Models. Addison-Wesley 1997
10. Fowler M., Patterns of Enterprise Application Architecture, Addison-Wesley 2003
11. Gamma E., Helm R., Johnson R., Vlissides J., Design Patterns – Elements of Reusable Object-Oriented Software. Addison-Wesley 1995
12. Ganter B., Wille R., Formal Concept Analysis. Mathematical Foundations, Springer 1999
13. Graham I., A Pattern Language for Web Usability. Addison-Wesley 2003
14. Hohpe G., Woolf B., Enterprise Integration Patterns. Addison-Wesley 2003
15. Kim M., Document Management and Retrieval for Specialised Domains: An Evolutionary User-Based Approach. A Thesis submitted to The School of Computer Science and Engineering. The University of New South Wales, Sydney Australia 2006
16. Kudělka M., Vzory pro HCI a GUI. Sborník konference Tvorba softwaru 2004, Ostrava 2004
17. Lehečka O., Podpora vyhledání softwarových vzorů. Diplomová práce, UP Olomouc 2006
18. Meszaros G., Doble J., A Pattern Language for Pattern Writing. `http://hillside.net/patterns/writing/patternwritingpaper.htm/` (31. dubna, 2006)
19. Nock C., Data Access Patterns: Database Interactions in Object-Oriented Applications. Addison Wesley, 2003
20. The Pedagogical Patterns Project. `http://www.pedagogicalpatterns.org/` (31. dubna, 2006)
21. Thilmany Ch., .NET Patterns: Architecture, Design, and Process. Addison Wesley, 2003
22. Tidwell J., Designing Interfaces: Patterns for Effective Interaction Design. O'Reilly Media, Inc. 2006
23. Trowbridge D., Mancini D., Quick D., Hohpe G., Newkirk J., Lavigne D., Enterprise Solution Patterns Using Microsoft.NET, Version 1.0. Microsoft Corporation 2003
24. Wellhausen T., User Interface Design for Searching. A Pattern Language. `http://www.tim-wellhausen.de/papers/UIForSearching/UIForSearching.html` (31. dubna, 2005)

Sémantika webových stránek založená na GUI vzorech

Miloš Kudělka, Ondřej Lehečka, and Václav Snášel

VŠB – Technická univerzita Ostrava
17. listopadu 15, 708 33, Ostrava-Poruba,
{Milos.Kudelka,Ondrej.Lehecka,Vaclav.Snasel}@vsb.cz

Abstrakt *V článku popisujeme nový způsob sémantického anotování obsahu webových stránek. Metoda je založena na GUI vzorech. Vzory slouží vývojářům pro porozumění požadavkům uživatele a pro implementace, které splňují uživatelovo očekávání. GUI vzory jsou využity pro sémantické hodnocení stránky a pro zadání dotazu. Metodu jsme implementovali a provedli experimenty a jejich vyhodnocení.*

1 Úvod

V článku se zabýváme úlohou vyhledávání na internetu. Dnešní přístupy se zaměřují na problémy s orientací ve stále zvětšujícím se prostoru nestrukturovaných dat. Důsledkem je problém velkého množství nedostatečně relevantních stránek v odpovědi na dotaz uživatele. Je tedy zřejmé že klasické modely vyhledávaní narážejí na své hranice.

S rostoucím počtem webových stránek s podobným obsahem lze vystopovat opakující se prvky a postupy při zobrazování a organizaci dat [5]. Zejména kvalitně zpracované webové stránky používají na podobné věci osvědčenou terminologii, GUI prvky a osvědčené uspořádání na stránce. Tyto společné rysy jsou popsány *GUI vzory* (viz např. [31]). Použité vzory na stránce uživatel snadno pozná a napříště dané vzory v dané doméně očekává. Příkladem může být vzor *Diskuze*. Uživatel v diskuzi např. očekává hierarchické řazení příspěvků pod sebe a podobnou terminologii. V naší metodě využíváme GUI vzory jako jednotící jazyk mezi tvůrci webových stránek a uživateli.

Dále ukážeme, že využitím vzorů lze dosáhnout dva důležité cíle:

1. **Zjednodušení dotazování.**
2. **Zlepšení kvality odpovědi.**

2 Současné přístupy

V oblasti zjednodušení dotazování lze nalézt zajímavé výsledky, např. v oblastech dlouhodobého sledování zájmů uživatele a přizpůsobení formulování dotazů tématům, o které se uživatel zajímá (viz např. [37] [17]), eventuelně hierarchicky organizované katalogy, ve kterých si uživatel vybírá.

V oblasti zvýšení kvality odpovědí lze pracovat s metadaty, která jsou ke stránkám přidávána v průběhu jejich tvorby nebo prostřednictvím analýzy jejich obsahu (sémantická analýza).V současné době se prosazují v oblasti sémantické analýzy dva směry.

První z nich poskytuje aparát pro ruční nebo částečně automatizované anotace stránek pomocí jazyků pro popis ontologií jakými jsou RDF, DAML+OIL, OWL apod. a tvorby SWD – sémantického webu. Výzkum se zaměřuje na řešení úloh vyhledávání, indexace a získávání informací nad takto anotovanými stránkami a také metodikami, jak k anotacím přistupovat (viz např. [17] [7] [13] [8] [33] [27] [10] [16] [2]).

Druhý směr preferuje automatizované pořízení anotace analýzou stránek reálného internetu (viz např. [34] [6] [21] [22]). Tento pohled předpokládá, že ruční pořízení anotace během vzniku stránky je obtížně realizovatelné. Klade totiž zvýšené nároky na autory stránek a nástroje, přičemž neřeší existenci neanotovaného internetu. Některé z přístupů jsou podobné našemu tím, že se snaží získat informace ze struktury stránky. Nicméně se obvykle zaměřují na zkoumání a reprezentaci HTML kódu stránky (viz [28] [32] [29] [24]).

3 Náš přístup

Do oblasti vyhledávání na internetu přicházíme s novým pohledem, který přirozeným způsobem propojuje oba naše cíle z úvodu. Klíčovým rysem tohoto pohledu je to, že je přísně zaměřen na uživatele a jeho očekávání.

Chceme vyhledávání posunout blíže k očekávání uživatele. Abychom to mohli udělat, potřebujeme, aby se s námi uživatel o svá očekávání podělil. Získání takových informací od uživatelů není jednoduchou záležitostí. Jednodušší metodou je obrátit se opačným směrem k profesionálním tvůrcům webových stránek, jejichž posláním je očekávání uživatelů plnit. Důkazem je, že kvalitní řešení jsou uživateli široce akceptována. Pro tvůrce webu platí, že jejich řešení ve stejné doméně se na určité úrovni shodují. Tuto shodu můžeme definovat tak, že na různých stránkách stejného zaměření se vyskytují opakující se prvky. Tyto prvky se označují jako vzory. Vzory nám poskytují sémantickou informaci, která je postavena na jednoznačné a empiricky ověřené dohodě mezi tvůrci a uživateli.

4 Vzory

Vzory se vyskytují v mnoha různých oblastech od architektury, kde byly formulovány poprvé, přes návrh uživatelského rozhraní až po návrh software (viz [1] [12]). V [31] můžeme najít popis toho, co vzory jsou.

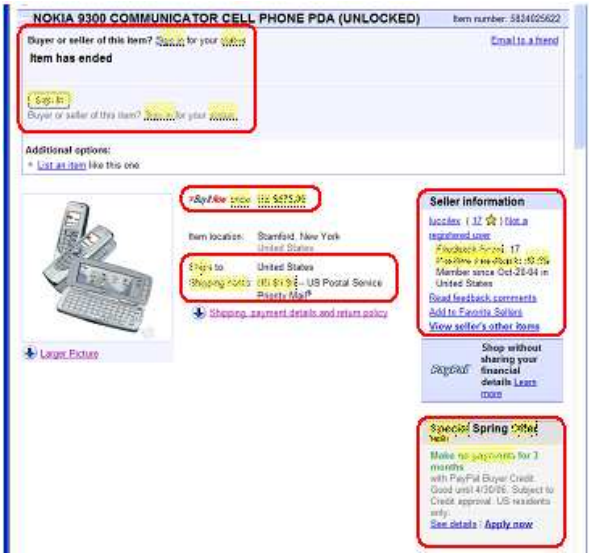
> *V podstatě jsou ve vzorech popsány charakteristické strukturální rysy a rysy chování, které zlepšují použitelnost architektury software, uživatelského rozhraní, webových stránek nebo čehokoliv jiného v určité doméně. Vzory činí věci použitelnějšími a jednoduššími na pochopení.*

GUI vzory poskytují řešení typických problémů při návrhu uživatelského rozhraní. Vzory ukazují vývojářům způsob, jak se vyrovnat s obvyklými situacemi. Typickými příklady může být např. organizace částí stránek do seznamů či záložek apod. GUI vzory na obecné úrovni popisují, jak strukturovat informace v uživatelském rozhraní, s jakými uživatelskými prvky a jak s nimi pracovat. Mnoho příkladů lze nalézt v [3] [35] [15]. Pro GUI vzory také platí, že v důsledku nepopisují, jak se má vzor technicky implementovat, ale jak se má projevovat vůči uživateli.

GUI vzory jsou spíše technické, popisují jak vyřešit konkrétní problém. Různé domény pak poskytují prostředí pro použití těchto vzorů v konkrétním kontextu. My jsme si v tomto článku vybrali doménu prodeje produktů (existují vzory zpracované i pro jiné domény, viz např. [36]). V uživatelských rozhraních v této oblasti můžeme najít společné rysy, které vyjadřují typické úlohy s informacemi (zobrazení obchodních informací, možnost objednání, zobrazení podrobných informací). Při implementaci těchto úloh postupují vývojáři podobně. Také používají vzory, i když o nich explicitně nemluví (viz [9]). V našich experimentech jsme použili vzory definované v [9].

4.1 Příklad

Na obrázku 1 je výřez ze stránky s prodejem výrobku na eBay.com. Graficky jsou v něm vyznačeny nalezené vzory. Jedná se o pět vzorů – *možnost přihlášení*, *obchodní informace*, *možnost objednání*, *hodnocení*, *speciální nabídka*.



Obrázek 1. Příklad stránky s vyznačenými vzory.

5 Vlastnosti vzoru

Je velmi obtížné nalézt formální popis toho, co uživatel bez problémů rozpozná na stránce. Na obrázku 1 je vidět pět oblastí, které můžeme označit jako vzory. Je potřeba si ovšem uvědomit, že pro různé internetové obchody se mohou lišit umístěním na stránce, způsobem implementace (v důsledku různý HTML kód) a dalšími detaily. Znovu se vrátíme k podstatě problému tak, jak jej vidí uživatel.

Již v úvodu článku bylo vysvětleno, že uživatelovo očekávání budeme modelovat vzory, jako společným nástrojem pro komunikaci mezi uživatelem a tvůrcem webových stránek. Při formalizaci tohoto přístupu musíme řešit problém, že vzor se na stránce neprojevuje nijak exaktně. Jde spíše o to, že vzory podporují více vnímání uživatele než vnímání technologa. Vzor je tedy málo závislý na způsobu implementace. Jako klíčový rys projevu vzoru na stránce se nám jeví to, že jednotlivé prvky jednoho konkrétního vzoru jsou víceméně pohromadě. Formálnější popis tohoto závěru je opět v [31]. Vizuální systémy obvykle mají čtyři základní vlastnosti (Gestalt Principles, viz [9]).

1. *Proximity* – související informace bývají blízko u sebe.
2. *Similarity* – podobně vypadající prvky obsahují podobné informace.
3. *Continuity* – informace následují plynule za sebou.
4. *Closure* – související informace bývají společně uzavřeny do celků.

Pokud vyjdeme z těchto vlastností, pak můžeme jeden vzor na stránce chápat jako skupinu charakteristických technických prvků (vycházejících z GUI vzorů – seznamy a tabulky, souvislé texty) a skupinu prvků charakteristických pro doménu, ve které se pohybujeme (především klíčová slova spojená s daným vzorem a další entity jako ceny, data, procenta apod.). Klíčovým rysem projevu vzoru na stránce je tedy to, že uvedené prvky jsou pohromadě.

Není tedy potřeba do hloubky zkoumat strukturu stránky, protože technické prvky poskytují pouze prostředí, ve kterém jsou související informace pohromadě.

Klíčové entity jsme ve vzorech na obrázku 1 zvýraznili. Množinu entit – pojmů pro jeden vzor chápeme jako *slovník vzoru*. Významnou vlastností slov ze slovníku je i to, že souvisí s doménou. Díky tomu můžeme očekávat, že budou splněny některé důležité charakteristiky (viz [18]):

1. Slovník není příliš rozsáhlý.
2. Slova se vyskytují v jistých schématech.
3. Význam slov je víceméně jednoznačný.
4. Slova se vyskytují v textu často.

5.1 Extrakce vzoru

Pro naše experimenty jsme problém formalizace vzoru zjednodušili na problém práce s množinou slov a datových entit, které jsou pro vzor charakteristické. Vezmeme-li v úvahu jeden konkrétní vzor (např. diskuzi) pak po analýze velkého množství stránek s diskuzemi zjistíme, že se opakuje poměrně úzká skupina slov a datových entit, podle kterých lze diskuzi rozpoznat. Budeme-li tedy předpokládat, že známe terminologii pro diskuze (používají se termíny jako *diskuze*, *autor*, *odpověď*, *příspěvek*, ...), pak můžeme v prostém textu stránky nalézt úseky, ve kterých se tato slova vyskytují.

Mějme tedy definovánu množinu $E = \{e_1, \ldots, e_n\}$, která obsahuje všechny entity $e_1, \ldots, e_n$ charakteristické pro daný vzor (slovník vzoru – klíčová slova a datové typy). Na potenční množině $P(E)$ můžeme definovat relaci δ tak, aby dvojice (E, δ) tvořila tzv. *proximitní prostor* (viz např. [4] [26]).

Proximitní prostor slouží jako model blízkosti skupin entit vzoru. Takto definovaná struktura nám dává prostředek pro popis a nalezení úseků textu stránky, které mohou (ale nemusí) být součástí daného vzoru.

Nechť je tedy I instance daného vzoru. Pak $I = \{S_1, \ldots, S_m\}$, kde $m > 0$ a $S_i \in P(E)$. Instancí vzoru tedy v důsledku rozumíme nějakou množinu úseků analyzovaného textu, které obsahují entity vzoru (nezabýváme se tedy analýzou významu skupiny slov, ale pouze jejich výčtem).

Pro hledání algoritmů, které nám pomohou nalézt a analyzovat vybrané úseky zkoumaného textu, nám dobře poslouží Gestalt principy.

1. Pro *proximity* jsme definovali způsob, jak měřit blízkost (vzdálenost) mezi entitami v nalezených úsecích textu. Vycházeli jsme z organizace entit reprezentujících úsek textu do stromu a z toho, že v hledaném úseku textu musí být entity dostatečně blízko u sebe (vzdálenost jsme stanovili na základě analýzy úseků textů v nalezených stránkách).
2. Pro *similarity* jsme definovali způsob měření míry podobnosti dvou nalezených úseků textu (pro diskuzi pak např. dokážeme zjistit opakování příspěvků). Vycházeli jsme z porovnání stromů entit reprezentujících úseky textu.
3. Pro *continuity* jsme definovali způsob, jak zjistit, zda dva nebo více nalezených úseků textu společně vytváří instanci vzoru. Vycházeli jsme z předpokladu, že dva nebo více málo podobných úseků textu (stromů entit z jednoho vzoru) patří k sobě.
4. Pro *closure* jsme definovali způsob výpočtu váhy jednoho nalezeného úseku textu. V principu jsme vycházeli ze dvou kritérií. Hodnotili jsme tvar stromu entit (především poměr výšky a počtu entit) a počet všech slov a odstavců v úseku textu. Na celkovém výpočtu váhy se podílí i hodnocení *proximity*.

Komplexním použitím všech uvedených principů jsme implementovali algoritmus, který poskytuje velmi dobré výsledky při extrakci vzorů ze stránek, a to pouze analýzou textu stránky (na základě ručního porovnání se jedná o úspěšnost více než 90%).

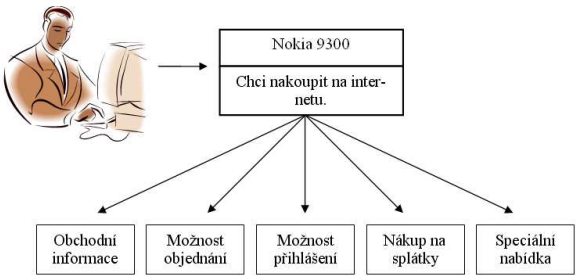
5.2 Algoritmus

Vstupem pro algoritmus je množina entit, které reprezentují jednotlivá slova a datové prvky z textu webové stránky, a množina charakteristických entit vzoru. Algoritmus tyto entity porovnává s charakteristickými entitami vzoru a vytváří reprezentace úseků textu *snippets* viz [11], které mohou patřit do vzoru. Nad těmito reprezentacemi pak realizujeme výpočty, jejichž výsledkem je hodnota reprezentující váhu výskytu vzoru na stránce.

```
FOR each page entity in all page entities
   IF page entity is pattern entity THEN
      IF doesn't exist appropriate snippet THEN
         create new snippet in list of snippets
      END IF
      add page entity to snippet
   END IF
END FOR

FOR each snippet in list of snippets
   compute proximity of snippet
   compute closure of snippet
   compute value(proximity, closure) of snippet
   IF value is not good enough THEN
      remove snippet from list of snippets
   END IF
END FOR

compute similarity of list of snippets
compute continuity of list of snippets
compute value(similarity, continuity) of pattern

RETURN value
```

5.3 Cíl 1: Zjednodušení dotazování

Vzory mohou poskytnout srozumitelný jazyk, kterým jsme schopni se s uživatelem domluvit na tom, co na stránce očekává. Musíme ovšem najít způsob, jakým nám to uživatel sdělí. V současných vyhledávačích musí vymyslet sadu slov, kterými specifikuje svůj požadavek. Může tedy použít například slovo „cena“ s očekáváním, že dostane stránky s cenou výrobku. Náš přístup k dotazování je postaven na stejném principu, ale s přístupnějším vyjadřovacím prostředkem pro uživatele. Vzor *obchodní informace* v sobě obsahuje mnohem silnější informaci o tom, že na stránce cena výrobku je. Vzory tedy nepřinášejí nic nového, pouze lépe než prostá slova modelují uvažování uživatele. Forma zadání není podstatná, důležité je, aby uživatel místo jednotlivých slov mohl s pomocí systému zvolit to, co očekává, na obecnější úrovni (tedy místo zadání slova *cena* mu systém nějakým způsobem musí nabídnout *obchodní informace* respektive další vzory).



Obrázek 2. Jednoduchá formulace dotazu z oblasti prodeje produktů.

5.4 Cíl 2: Zlepšení kvality odpovědi

Předpokládejme, že máme stránky ve své databázi anotovány s ohledem na vzory a tím uloženy i s informací o tom, jaké vzory jednotlivé stránky obsahují. Tyto informace pak můžeme použít dvěma způsoby.

1. Můžeme pro každý zobrazený odkaz na stránku ve vyhledaném výčtu přidat informaci o tom, které vzory byly na stránce nalezeny. Uživatel tak na první pohled pozná, zda se jedná o stránku, která s velkou pravděpodobností splní jeho očekávání.
2. Můžeme vzory zohlednit už při vyhledání a seřadit odkazy na stránky právě s ohledem na to, s jakou váhou jsme požadované vzory na stránce našli (obdoba PageRanking [23]). Vedlejším efektem je v tomto případě i to, že preferovány budou ty stránky, jejichž tvůrci použili při jejich implementaci postupy doporučené ve vzorech. Jak bylo řečeno výše, vzory popisují uživateli široce akceptovaná řešení. Proto uživatel dostane ve výběru dříve kvalitněji zpracované stránky.

6 Experimenty

Naše metoda extrakce a hodnocení vzorů je obecná a otevřená vůči analyzovaným vzorům. Každý vzor je popsán svým slovníkem. Nyní máme připraveny slovníky pro 9 vzorů (*Informace o ceně, Možnost nákupu, Speciální nabídka, Prodej na splátky, Informace o produktu, Diskuse, Recenze, Možnost přihlášení, Bazar a inzeráty*).

Implementovali jsme webovou aplikaci, která využívá pro testovací účely k dotazování Google APIs (viz [14]) a český vyhledávací stroj Jyxo (viz [19]). Aplikaci jsme implementovali ve MS Visual Studio 2005 a použili jsme jazyky C# a VB.NET. Součástí aplikace jsou algoritmy, které umí extrahovat vzory z textových obsahů stránek. Kromě toho aplikace poskytuje analyzátor, který pracuje se startovacími slovníky vzorů a zpracovává vybrané úseky textu s ohledem na automatické doplnění slovníku vzoru (viz [6] [20]). Pro prvotní vytvoření slovníku vzoru jsme vybrali tři až pět slov a následnou analýzou okolí těchto slov na stránkách, jsme slovník postupně doplnili. Pro jednotlivé vzory je koncová velikost slovníku přibližně 20 slov.

V našich experimentech jsme použili více než 30000 stránek se zbožím (od recenzí, přes diskuze po obchodní informace a bazary). Aplikace doplňovala na pozadí k dotazu uživatele slova upřesňující jeho očekávání (které uživatel volil při zadání výběrem ze seznamu vzorů). Příkladem mohou být slova *příspěvek* a *diskuze*, pokud uživatel hledal diskuzi k výrobku Nokia 9300. Po získání odpovědi jsme stránky analyzovali a seřadili podle váhy vybraných vzorů. Pro analýzu jsme použili prvních třicet nalezených stránek.

Kvalitu našeho návrhu a našich algoritmů jsme posuzovali na základě porovnání odpovědi zmíněných vyhledávačů s odpovědí naší aplikace. Jednalo se o pořadí stránek vrácených vyhledávačem a pořadím, které vyšlo po naší analýze. Naše hodnocení vychází z výpočtu váhy extrahovaných vzorů.

Aplikaci jsme provozovali na počítači se systémem Windows XP a s procesorem Intel Pentium M 1.60 GHz s pamětí 1,5 GB. Na každé stránce jsme extrahovali devět vzorů. Na našem vzorku více než 30000 stránek se rychlost extrakce vzorů pohybovala přibližně na 100 stránkách za vteřinu. Průměrná doba extrakce jednoho vzoru na jedné stránce byla přibližně 10^{-3}s.

Během experimentování jsme nashromáždili 31738 stránek, které nám poskytly zmíněné vyhledá-

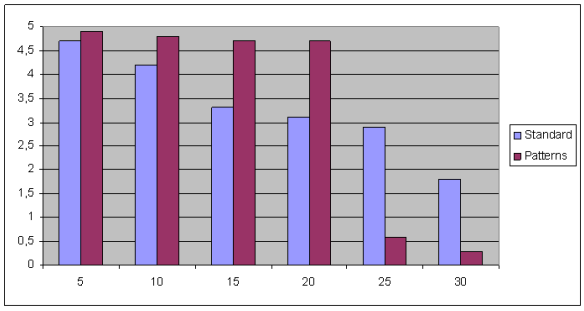
vače na dotazy na prodávané produkty. Po analýze stránek jsme zjistili, že na 11038 z nich se nepodařilo extrahovat žádný vzor, přičemž naše dotazy směřovaly k nalezení stránek s těmito vzory (dotazy z naší aplikace obsahovaly skupiny upřesňujících slov). I přesto, že musíme počítat s dotazy, na které nelze nalézt dostatek relevantních odpovědí a s nepřesnostmi našich algoritmů, ukazuje se, že *i při velmi kvalitním dotazu musí uživatel očekávat přibližně čtvrtinu až třetinu stránek, které neposkytují očekávané informace.*

Na obrázku 3 jsou zobrazeny výsledky dotazů na konkrétní výrobky. Do výsledků jsme zahrnuli pouze dotazy, na které existuje na internetu násobně více relevantních stránek, než je námi sledovaných prvních třicet dodaných vyhledávačem (jedná se o více než 200 různých dotazů).

Na uvedeném grafu je na vodorovné ose uvedena škála, na které je zobrazeno prvních třicet nalezených stránek po pěticích. Na svislé ose je uváděn počet relevantních stránek v každé pětici, přičemž levé sloupce zobrazují hodnoty odpovídající řazení námi použitých vyhledávacích strojů (v tomto ohledu se téměř neodlišovaly). V pravých sloupcích jsou zobrazeny hodnoty odpovídající naší testovací aplikaci.

Z uvedeného grafu vyplývá:

- Nerelevantní stránky jsou odsunuty na konec výběru (a na rozdíl od původního pořadí prakticky nedochází k chybám).
- S použitím našeho hodnocení se dostane uživatel k očekávaným stránkám dříve.



Obrázek 3. Rozložení relevantních stránek v prvních třiceti vyhledaných stránkách.

7 Závěr

Klíčovým technickým rysem našeho přístupu je to, že nepotřebuje analyzovat HTML kód stránky. Všechny naše algoritmy jsou založeny na analýze prostého textu stránky. Pro naše hodnocení nepoužíváme žádné metainformace o stránce (titulek, hyperlink, metatagy apod.) Také jsme si ověřili, že klíčové charakteristiky vzoru jsou nezávislé na jazykovém prostředí. Naši metodu jsme testovali v anglickém a českém jazykovém prostředí. Stačilo pouze vyměnit slovníky vzorů.

Z našich experimentů vyplývá, že je užitečné zjištěné informace o existenci vzorů na stránce považovat za metadata uložená společně se stránkou. V této chvíli máme nástroje, jak s relevancí kolem 90% o stránce zjistit, zda obsahuje či neobsahuje hledaný vzor.

Náš přístup není univerzální. Důvodem je, že základním předpokladem pro kvalitu našeho hodnocení je doména s relativně usazenými pravidly toho, jak vypadají stránky (nepředpokládáme uniformitu, předpokládáme jistou synchronizaci mezi uživateli a tvůrci stránek). Z toho vyplývá i zajímavý vedlejší efekt. Výše ve výběru jsou u nás ty stránky, které lépe dodržují zmíněnou čtveřici Gestalt principů *proximity-similarity-continuity-closure*. Kromě prodeje produktů existují i další domény, nad kterými se chystáme provádět další experimenty.

Možnost rozšíření systému o další vzory je otevřená. Pracujeme na extrakci dalších vzorů z domény prodeje produktů (jako např. *Hodnocení*, *Otázky na prodejce*). Počítáme také s rozšířením systému na další domény, ve kterých lze identifikovat vzory (*Dovolená*, *Kultura* apod.).

Reference

1. Alexander Ch., A Pattern Language: Towns, Buildings, Construction. New York. Oxford University Press 1977
2. Baumgartner R., Flesca S., Gottlob G., Visual Web Information Extraction with Lixto. Proceedings of the 27th International Conference on Very Large Data Bases, 2001, 119–128
3. Borchers J.O., A Pattern Approach to Interaction Design. Proceedings of the DIS 2000 International Conference on Designing Interactive Systems, ACM Press, 2000
4. čech, E. Toplogical Spaces. J. Wiley-Interscience Publ., New York 1966
5. Chakrabarti S., Mining the Web: Discovering Knowledge from Hypertext Data. Morgan Kaufman Publishers, 2003
6. Ciravegna F., Chapman S., Dingli A., Wilks Y., Learning to Harvest Information for the Semantic Web. ESWS 2004, LNCS 3053, Springer-Verlag Berlin Heidelberg , 2004, 312–326
7. Dill S., Eiron N., Gibson D., Gruhl D., Guha R., Jhingran A., Kanungo T., McCurley K.S., Rajagopalan S., Tomkins A., Tomlin J.A., Zien J.Y.: A Case for Automated Large-Scale Semantic Annotation. Journal of Web Semantics, 1(1), 2003, 115–132
8. Ding L., et al., Swoogle: A Search and Metadata Engine for the Semantic Web. Proc. CIKM 2004, 652–659

9. Van Duyne D.K., Landay J.A., Hong J.I., The Design of Sites: Patterns, Principles, and Processes for Crafting a Customer-Centered Web Experience. Pearson Education 2002
10. Erdmann M., Maedche A., Schnurr H.-P., Staab S., From Manual to Semi-automatic Semantic Annotation: About Ontology-based Text Annotation Tools. In P. Buitelaar, K. Hasida (eds). Proceedings of the COLING 2000 Workshop on Semantic Annotation and Intelligent Content, Luxembourg, August 2000
11. Ferragin P., Gulli A., A Personalized Search Engine Based on Web-Snippet Hierarchical Clustering. In: Proc. of 14th International Conference on World Wide Web, Chiba, Japan, 2005, 801–810
12. Gamma E., Helm R., Johnson R., Vlissides J., Design Patterns – Elements of Reusable Object-Oriented Software. Addison-Wesley 1995
13. Goble C., Bechhofer S., Carr L., De Roure D., Hall W., Conceptual Open Hypermedia = The Semantic Web? In The Second International Workshop on the Semantic Web, Hong Kong, May 2001, 44–50
14. Google Web APIs – Home. `http://www.google.com/apis/` (May 29, 2005)
15. Graham I., A Pattern Language for Web Usability. Addison-Wesley 2003
16. Handschuh S., Staab S., Ciravegna F., S-CREAM Semi-Automatic CREAtion of Metadata. The 13th International Conference on Knowledge Engineering and Management (EKAW2002), Gomez-Perez A. (ed.), Springer Verlag, 2002
17. Husek D., Owais S., Kromer P., Snasel V., Neruda R., Implementing GP on Optimizing both Boolean and Extended Boolean Queries in IR and Fuzzy IR systems with Respect to the Users Profiles. 2006 IEEE World Congress on Computational Intelligence, CEC, 2006, accepted.
18. Jianming Li, L. Z. Yu, Y., Learning to Generate Semantic Annotation for Domain Specific Sentences. In Knowledge Markup And Semantic Annotation Workshop in K-CAP 2001, 2001
19. Jyxo.cz. `http://www.jyxo.cz/` (May 29, 2005)
20. Karov Y., Edelman S., Similarity-Based Word Sense Disambiguation. Computational Linguistics 24(1), 1998, 410–59
21. Kiryakov K., Popov B., Ognyanoff D., Manov D., Kirilov A., Goranov M., Semantic Annotation, Indexing, and Retrieval. ISWC 2003, LNCS 2870, Springer-Verlag Berlin Heidelberg 2003, 484–499
22. Kiyavitskaya N., Zeni N., Cordy J.R., Mich L., Mylopoulos J., Semantic Annotation as Design Recovery. ISWC 2005, 4th International Semantic Web Conference, Galway, Ireland, submitted April 2005, 15 pp.
23. Langville A.N., Meyer C.D., A Survey of Eigenvector Methods for Web Information Retrieval. SIAM Review, Vol. 47, No . 1, 2005, 135–161
24. Li Z., Ng W.K., Sun A., Web Data Extraction Based on Structural Similarity. Knowl. Inf. Syst. 8(4), 2005, 438–461
25. Mullet K., Sano D., Designing Visual Interfaces: Communication Oriented Techniques. Englewood Cliffs, NJ. Prentice Hall 1994
26. Naimpally S.A., Warrack B.D., Proximity Spaces. Cambridge University Press, Cambridge 1970
27. Plessers P., Troyer O.D., Annotation for the Semantic Web during Website Development. ICWE 2004, 349–353
28. Pivk A., Automatic Ontology Generation from Web Tabular Structures. PhD Thesis, University of Maribor, June 2005
29. Reis D.C., Golgher P.B., Silva A.S., Laender A.F., Automatic Web News Extraction Using Tree Edit Distance. In WWW'04: Proceedings of the 13th International Conference on World Wide Web, New York, NY, USA, ACM Press, 2004, 502–511
30. Sean L., Lee S., Rager D., and Handler J., Ontology-Based Web Agents. Proceedings of the First International Conference on Autonomous Agents (Agents'97), Johnson W.L., and Hayes-Roth B. (eds), 59-68, Marina del Rey, CA, USA, 1997, ACM Press, 1997, 59–68
31. Tidwell J., Designing Interfaces: Patterns for Effective Interaction Design. O'Reilly Media, Inc. 2006
32. Tijerino Y.A., Embley D.W., Lonsdale D.W., Ding Y., Nagy G., Towards Ontology Generation from Tables. World Wide Web 8(3), 2005, 261–285
33. Thomas E., Zhang Y., Sleeman D., Preece A., McKenzie C., Wright J., OntoSearch: a Service to Support the Reuse of Ontologies. Demos and Posters of the 2nd European Semantic Web Conference (ESWC) 2005
34. Vargas-Vera M., Motta E., Domingue J., Lanzoni M., Stutt A., Ciravegna F., MnM: Ontology Driven Semi-Automatic or Automatic Support for Semantic Markup. In Proc. of the 13th International Conference on Knowledge Engineering and Knowledge Management, EKAW02. Springer Verlag, 2002
35. Van Welie M., van der Veer G.C., Pattern Languages in Interaction Design: Structure and Organization. Proceedings of Interact'03, Zürich, Switzerland. IOS Press, Amsterdam 2003
36. Wellhausen T., User Interface Design for Searching. A Pattern Language. `http://www.tim-wellhausen.de/papers/UIForSearching/UIForSearching.html` (May 29, 2005)
37. Wechsler K., Baier J., Nussbaum M., Baeza-Yates R., Semantic Search in the WWW Supported by a Cognitive Model. In Int. Conf. Web-Age Information Management, LNCS, Springer, Dalian, China, July 2004

The role of kernel function in Regularization Network*

Petra Kudová

Institute of Computer Science, Academy of Sciences of the Czech Republic,
Pod Vodárenskou věží 2, 182 07 Prague, Czech Republic
petra@cs.cas.cz,
WWW home page: http://www.cs.cas.cz/~petra

Abstract. *The problem of learning from examples, also known as supervised learning, is a subject of great interest at present. We study one approach to this problem – the Kernel Based Regularization Networks. We discuss the role of a kernel function, and show that its right choice is crucial for the performance of a Regularization Network. The influence of the choice of kernel function on training and generalization error will be demonstrated on experiments. Different types of kernel functions will be compared. First, we will show that functions with local response (such as Gaussian, Inverse Multi-quadratic) often outperform, in terms of generalization error, global functions (such as Sigmoid). Second, we will show that some kernels generalize well without the need of regularization, i.e. the training error can be kept very low and still the network exhibits good generalization. Such kernels are good choice for tasks without noise or with low level of noise.*

1 Introduction

The amount of data produced in various areas of human activity is rapidly increasing. At the same time the interest in learning algorithms increases. In many applications we encounter the problem that given a data sample of limited size we have to find a concise description of the data.

In this paper we deal with the problem of *supervised learning*. In this case the data is a sample of input-output patterns (called training sample or training set), thus a concise description of the data is typically a function that can produce the output, given the input. Then the task of learning is to find a deterministic function that maps any input to an output such that the disagreement with future input-output observations is minimized. Supervised learning covers wide range of tasks, including classification of handwritten digits, prediction of stock market share values, and weather forecasting, etc.

* This work has been partially supported by Grant Agency of the Czech Republic under grant 201/05/0557, by the European Commission's Research Infrastructures activity, contract number RII3-CT-2003-506079 (HPC-Europa), and also by the Institutional Research Plan AV0Z10300504 "Computer Science for the Information Society: Models, Algorithms, Appplications".

An important feature of the learning algorithm is a *generalization ability*. By generalization we mean that the mapping found by the learning algorithm maps correctly not only inputs included in the training set, but also gives correct answers for input points that were not seen before.

In this work we study one particular learning algorithm, called *Regularization Network*. Regularization Networks are derived from regularization theory that is based on the idea of simultaneous minimization of error on the training set and regularization term, reflecting our knowledge about a given problem.

In the next section we briefly describe the derivation of Regularization Network [2, 7, 11, 3, 4]. In section 3 we discuss the role of kernel function, which is an activation function of the hidden layer of Regularization Network. In section 4 we demonstrate how the quality of the solution depends on the choice of the regularization parameter and a kernel function. We compare several kernel functions and show that functions with local response often outperform global functions.

2 Regularization Networks

Now we will formalize the problem of supervised learning. We are given a set of examples (pairs)

$$\{(\boldsymbol{x}_i, y_i) \in R^d \times R\}_{i=1}^N$$

that was obtained by random sampling of some real function f, generally in the presence of noise. Our goal is to recover the function f from data, or find the best estimate of it.

Note that it is not necessary that the function exactly interpolates all the given data points, but we need a function with a good *generalization*, that is a function that gives relevant outputs also for the data not included in the training set. (See Fig. 1).

It is easy to see that the problem is generally ill-posed. There are many functions interpolating the given data points, but not all of them also exhibit the generalization ability.

Since the problem is ill-posed, it is necessary to consider some a priori knowledge or assumption about

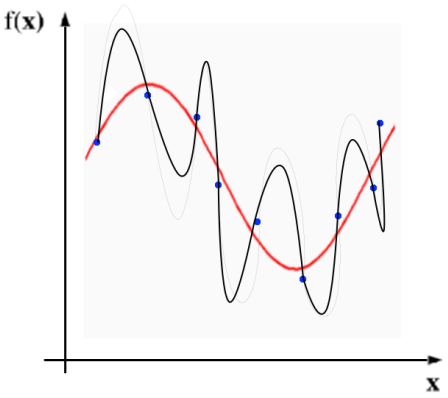


Fig. 1. The problem of learning from examples.

the function f. Typically, we assume that the function is smooth, does not oscillate too much, etc.

Then we are looking for a function minimizing the functional containing both the data term and smoothness information [10]

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

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

Poggio, Girrosi, and Jones [2] used the stabilizer

$$\Phi[f] = \int_{R^d} d\boldsymbol{s} \frac{|\tilde{f}(\boldsymbol{s})|^2}{\tilde{G}(\boldsymbol{s})}, \tag{2}$$

where $\tilde{f}$ indicates the Fourier transform of f , $\tilde{G}$ is some positive function that goes to zero for $||s|| \to \infty$, i.e. $1/\tilde{G}$ is a high-pass filter.

Under slight assumptions on $\tilde{G}$ it can be shown that the minimum of the functional (2) has the form of linear combination of basis functions G

$$f(x) = \sum_{i=1}^{N} w_i G(\boldsymbol{x} - \boldsymbol{x}_i) + \sum_{\alpha=1}^{k} d_\alpha \psi_\alpha(\boldsymbol{x}), \tag{3}$$

where $\{\psi_\alpha\}_{\alpha=1}^{k}$ is a basis of the k-dimensional null space $\mathcal{N}$ of the functional Φ. Coefficients d_α and w_i depend on the data and satisfy the following linear system:

$$(G + \gamma I)\boldsymbol{w} + \Psi^T \boldsymbol{d} = \boldsymbol{y} \tag{4}$$

$$\Phi \boldsymbol{w} = 0 \tag{5}$$

where I is the identity matrix.

For positive semi-definite function G the null space is empty, for conditionally positive semi-definite function G the basis of the null space is a set of polynomials, however in practical applications the polynomial term is omitted. Then the function f (3) is a linear combination of basis functions G and can be represented by a neural network with one hidden layer. We call such network a *Regularization Network* (RN).

Poggio and Smale [7] derived the RN using Reproducing Kernel Hilbert Spaces (RKHS). RKHS was defined by Aronszajn as a Hilbert space of functions with the property that each evaluation functional is bounded (see [1]).

Poggio and Smale minimized the functional (1) over RKHS corresponding to kernel function K, with stabilizer defined by norm

$$\Phi[f] = ||f||_K^2. \tag{6}$$

Then the solution is unique and has a norm

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} w_i K(\boldsymbol{x}, \boldsymbol{x}_i) \tag{7}$$

and the coefficients w_i satisfy

$$(K + \gamma I)\boldsymbol{w} = \boldsymbol{y}. \tag{8}$$

Thanks to this connection to RKHS, we call the basis function of RN a *kernel function.*

For a positive semi-definite kernel function and $\gamma > 0$, the matrix $K + \gamma I$ is strictly positive and the linear problem is well-defined. But for practical applicability the numerical stability, which depends on the value of γ and the type of kernel function, is also important.

3 The role of kernel function

In application of RN on a given task, we suppose that the value of γ and the type of kernel function K are given in advance. Then the training of RN consists of solving the linear system (8).

The kernel function K should be chosen according to our knowledge of the problem at hand. In general, the choice of a kernel function corresponds to

1. choice of a stabilizer - choice of kernel is equivalent to the choice of function $\tilde{G}$ in stabilizer (2), i.e. to the particular form of this functional.
2. choice of a function space for learning - when using RKHS, the choice of the kernel is in fact the choice of RKHS, that is the choice of our hypothesis space

So the kernel function is a representation of our knowledge or assumption about the problem and its solution. Therefore, as there is no "free lunch" in learning, there is no free lunch in choice of kernel [12, 9].

However, in practice we typically do not have any prior knowledge of the problem. So it is necessary to

Gaussian	$K(x,y) = e^{-\|\|x-y\|\|^2}$	positive semi-definite
Inverse Multi-quadratic	$K(x,y) = (\|\|x-y\|\|^2 + c^2)^{-1/2}$	positive semi-definite
Multi-quadratic	$K(x,y) = (\|\|x-y\|\|^2 + c^2)^{1/2}$	cond. positive semi-definite
Thin Plate Spline	$K(x,y) = \|\|x-y\|\|^{2n+1}$	cond. positive semi-definite
Sigmoid	$K(x,y) = tanh(xy - \theta)$	cond. positive semi-definite

Table 1. Kernel functions.

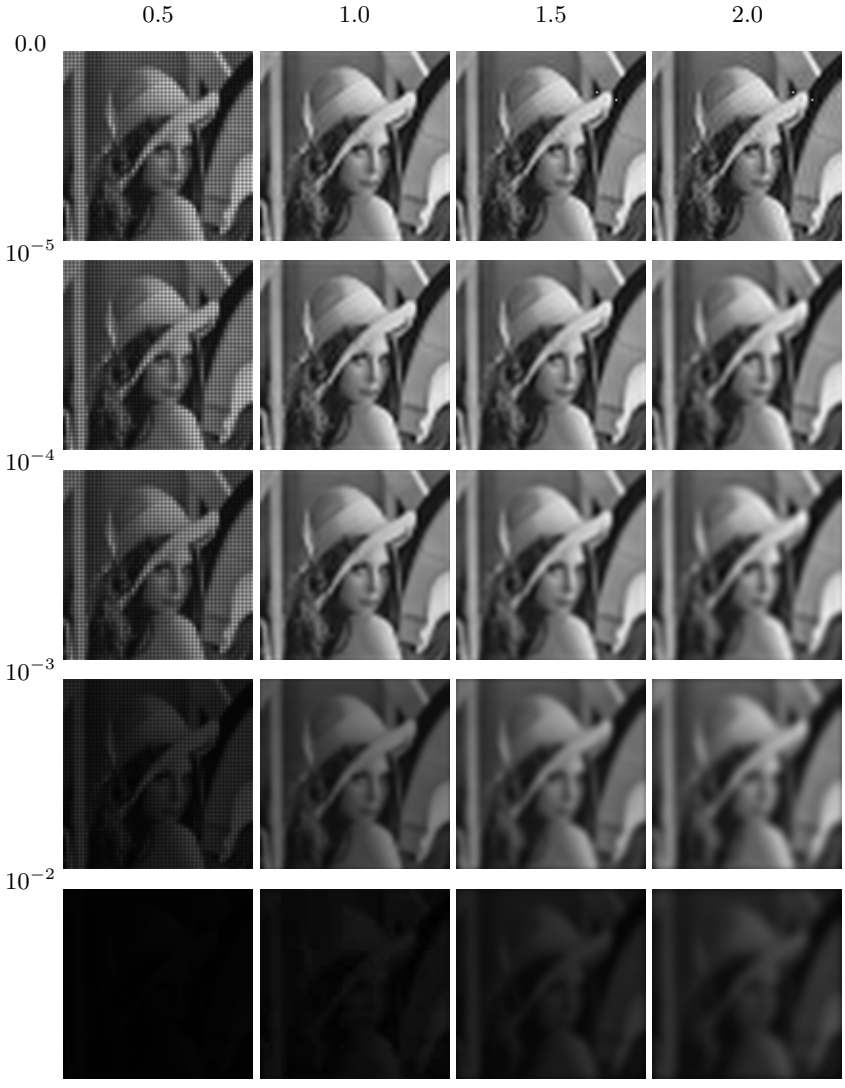


Fig. 2. Images learned by RN on Lenna image (50×50 pixels) using Gaussian kernels with widths from 0.5 to 2.0 and regularization parameters from 0.0 to 0.01.

try several kernel functions and choose the best one, for instance by cross-validation. The most common kernel functions are listed in Table 1.

In our experiments we try to answer the question whether some kernel function is better first choice than another and whether there is any class of kernel functions that are suitable for most tasks. We do not expect to find a kernel function that outperforms the others in all situations, simply because it is not possible. Rather we expect that the experimental study will help us to more understand how the choice of kernel influences the solution, and give us some clues for its right choice.

4 Experiments

This section presents results of our experiments with RN. The goal is to demonstrate the performance of RN with respect to different setup, i.e. choice of γ and kernel, and to compare RNs with different kernel functions.

In all experiments we use different data sets for training and testing (called *training set* and *testing set*). The following procedure is used:

1. find the values for γ and kernel's parameters by cross-validation on the training set
2. use the whole training set and the parameters found by Step 1 to estimate the weights of RN
3. evaluate the error on the testing set

The error on the testing set is our estimation of the real generalization ability of the network.

The error is always normalized

$$E = 100\frac{1}{N}\sum_{i=1}^{N}||\boldsymbol{y}_i - f(\boldsymbol{x}_i)||^2,$$

where N is the number of data samples, $\boldsymbol{y}_i$ is the desired output for the input vector $\boldsymbol{x}_i$, $f(\cdot)$ is the network output and $||\cdot||^2$ denotes the Euclidean norm.

The LAPACK library [6] was used for linear system solving.

We have chosen the well known picture of Lenna to study the approximation and smoothing capabilities of Regularization Networks. Our training set contains 2500 samples representing the image of 50×50 pixels. The obtained RN was then used to generate a 100×100 image.

Fig. 2 displays images obtained with RNs using Gaussian kernels of different widths and different regularization parameters. It is easy to see that the choice of these parameters is crucial for the performance of RN. Also the role of the regularization parameter is demonstrated, the higher the regularization parameter the smoother the result. For too high regularization parameters we get solutions too far from training data, i.e. the black images. Note, that the wrong choice of kernel parameter (such as too small width of Gaussians in the left column) cannot be cured by the change of the regularization parameter.

For comparison of different kernel functions we have chosen the data collection Proben1 [8]. Kernel functions listed in Table 1 were used. In addition we used a kernel function formed as a sum of two Gaussian functions (see [5] for details on kernels obtained by means of sum and product from simpler kernels).

The Table 2 compares the training and testing errors achieved with these kernels on data tasks from Proben1. The Figure 4 and Figure 3 compare the overall error (error summed over all data sets) on the training set and the testing set, respectively.

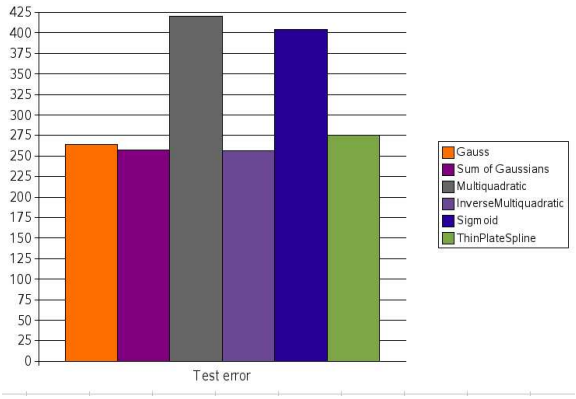


Fig. 3. Comparison of overall test errors for different kernels.

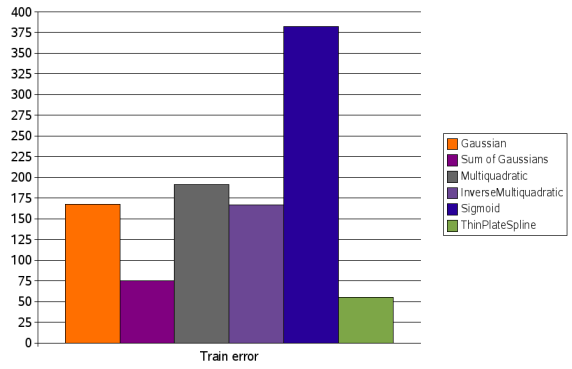


Fig. 4. Comparison of overall training errors for different kernels.

In most cases, the best results, in terms of error on the testing set, were achieved by RNs with these

kernels: Inverse Multi-quadratic kernel function, kernel formed by a sum of two Gaussian functions, and Gaussian kernel function. All these functions are functions with a local response, i.e. they give a relevant output only in a local area around its center.

The lowest error on the training set was achieved by RNs with kernels: Thin Plate Spline kernel function, sum of Gaussian functions, and Multi-quadratic. The Multi-quadratic kernel, however, failed completely on the Glass tasks (therefore the overall error is quite high). These kernel functions achieved almost zero training error on many tasks. It is caused by the fact that γ was set to zero by cross-validation. But even with the zero regularization parameter, these kernels preserve the generalization ability.

In case of the sum of two Gaussian functions, the kernel winning in cross-validation is formed by a sum of two Gaussian functions, from which one is very narrow (see Fig. 5). The diagonal of the matrix K (8) is dominant and so the linear system is numerically more stable.

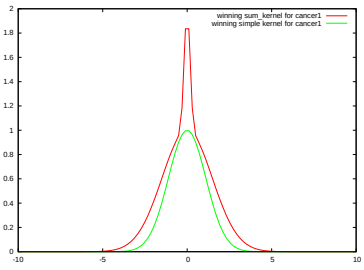


Fig. 5. Chosen sum kernels for cancer1 task.

Such kernels are suitable for tasks without noise, for which the close fitting of the training data is desirable.

5 Conclusion

We studied one approach to the problem of supervised learning, the Kernel Based Regularization Networks.

We discussed the role of a kernel function, and have shown that its choice is crucial for the performance of a Regularization Network. On experiments we demonstrated how the training and testing errors depend on the choice of kernel function.

The most common types of kernel functions were compared on the benchmark data sets. We have shown that functions with a local response (such as Gaussian, Inverse Multi-quadratic) were best in most tasks, so they represent a good first choice. Also we have shown, that some types of kernels preserve the generalization ability without the need of regularization term. Such kernel functions are useful for tasks without or with a low level of noise.

References

1. N. Aronszajn, Theory of Reproducing Kernels. Transactions of the AMS, 68, 1950, 337–404
2. F. Girosi, M. Jones, and T. Poggio, Regularization Theory and Neural Networks Architectures. Neural Computation, 2(7), 1995, 219–269
3. S. Haykin, Neural Networks: a Comprehensive Foundation. Tom Robins, 2nd edition, 1999
4. P. Kudová and R. Neruda, Kernel Based Learning Methods: Regularization Networks and RBF Networks. In Lawrence N. Winkler J., Niranjan M., (eds), Deterministic and Statistical Methods in Machine Learning, Berlin, Springer-Verlag, 2005, 124–136
5. P. Kudová and T. Šámalová, Sum and Product Kernel Regularization Networks. In L. Rutkowski, R. Tadeusiewicz, L.A. Zadeh, and J. Zurada, (eds), Artificial Intelligence and Soft Computing, Lecture Notes in Artificial Intelligence, Berlin, Springer-Verlag, 2006, 56–65
6. LAPACK. Linear Algebra Package, `http://www.netlib.org/lapack/`
7. T. Poggio and S. Smale, The Mathematics of Learning: Dealing with Data. Notices of the AMS, 50(5), 2003, 536–544
8. L. Prechelt, PROBEN1 – a Set of Benchmarks and Benchmarking Rules for Neural Network Training Algorithms. Technical Report 21/94, Universitaet Karlsruhe, 9 1994
9. B. Schoelkopf and A. J. Smola, Learning with Kernels. MIT Press, Cambridge, Massachusetts, 2002
10. A.N. Tikhonov and V.Y. Arsenin, Solutions of Ill-posed Problems. W.H. Winston, Washington, D.C, 1977
11. G. Wahba, Spline Models for Observational Data. Series in Applied Mathematics, 59, 1990
12. D.H. Wolpert, The Lack of a Priori Distinctions between Learning Algorithms. Neural Computations, 8(7), 1996, 1341–1390

	Gaussian		Sum of Gaussians		Multi-quadratic		Inverse Multi-quadratic		Sigmoid		Thin-Plate Spline	
	E_{train}	E_{test}	E_{train}	E_{test}	E_{train}	E_{test}	E_{train}	E_{test}	E_{train}	E_{test}	E_{train}	E_{test}
cancer1	2.38	1.79	0.00	1.77	0.00	1.61	1.79	**1.49**	3.02	1.83	0.00	**1.49**
cancer2	1.86	3.01	0.00	2.96	0.00	3.03	1.46	**2.88**	2.54	3.58	0.00	**2.88**
cancer3	2.07	2.79	0.00	2.73	0.00	3.25	1.89	**2.59**	2.66	2.84	0.00	2.74
card1	7.71	**10.00**	8.81	10.03	0.00	22.35	8.69	10.01	24.72	24.98	0.00	11.47
card2	6.79	12.75	0.00	**12.54**	0.00	15.21	7.31	12.56	26.17	26.45	0.00	14.06
card3	7.10	**12.32**	6.55	**12.32**	0.84	14.70	6.00	12.36	11.51	15.39	0.00	14.15
diabetes1	14.17	16.22	14.01	**16.00**	15.81	17.25	13.13	16.12	13.77	16.73	11.79	17.07
diabetes2	13.95	16.85	13.78	**16.80**	15.88	17.11	14.33	**16.80**	13.09	18.35	13.63	16.82
diabetes3	13.75	15.99	13.69	15.95	15.92	16.32	13.63	**15.93**	13.97	16.69	11.85	17.18
flare1	0.36	0.55	0.35	**0.54**	0.19	0.64	0.35	**0.54**	0.38	0.55	0.26	0.58
flare2	0.42	0.27	0.44	**0.26**	0.21	0.42	0.43	**0.27**	0.45	0.30	0.31	0.34
flare3	0.40	0.34	0.42	**0.33**	0.20	0.47	0.41	0.34	0.41	0.35	0.29	0.41
glass1	3.90	7.33	2.35	6.15	84.91	70.75	2.20	**6.12**	6.76	8.58	0.00	6.41
glass2	3.58	7.78	1.09	6.97	31.56	27.56	1.88	**6.79**	6.90	8.92	0.00	7.29
glass3	3.87	7.25	3.04	6.29	24.75	36.83	2.24	**6.14**	6.74	9.09	0.00	6.20
heartac1	3.80	3.13	0.00	3.26	0.00	4.08	4.16	**2.82**	9.55	8.80	0.00	3.51
heartac2	2.75	3.95	0.00	3.85	0.00	5.00	3.38	**3.84**	8.15	7.27	0.26	4.77
heartac3	3.12	5.17	3.36	5.01	0.00	4.92	3.34	5.08	5.43	6.14	0.00	**4.72**
hearta1	3.46	4.46	0.00	4.37	0.00	5.73	3.17	**4.31**	8.44	8.40	2.47	5.14
hearta2	3.48	4.26	3.51	**4.06**	0.00	5.79	3.18	4.13	8.11	8.31	0.00	4.86
hearta3	3.39	4.49	0.00	4.49	0.00	5.52	3.12	**4.40**	8.53	8.43	0.00	4.96
heartc1	8.78	15.93	0.00	15.69	0.00	15.93	9.46	15.94	18.86	21.64	0.00	**15.65**
heartc2	11.55	6.52	0.00	6.33	0.00	7.33	12.38	**6.16**	24.80	22.71	0.00	6.65
heartc3	6.54	13.66	0.00	**12.38**	0.00	14.23	8.03	12.72	20.34	18.68	0.00	13.82
heart1	9.76	13.69	0.00	13.91	0.00	16.95	9.58	**13.59**	26.92	27.69	7.23	13.83
heart2	9.48	13.86	0.00	13.82	0.00	19.29	9.29	**13.74**	14.34	16.80	6.82	14.69
heart3	8.92	16.01	0.00	**15.94**	0.00	21.84	8.03	16.15	26.27	27.40	0.00	18.86
horse1	4.51	12.47	0.20	11.90	0.16	12.33	7.16	**11.69**	18.43	16.75	0.16	12.13
horse2	4.14	15.38	2.84	**15.11**	1.08	17.72	5.70	15.34	18.08	17.48	0.18	16.72
horse3	0.82	14.26	0.18	14.13	0.18	14.52	0.37	**14.00**	18.36	18.20	0.18	14.23
soybean1	0.12	0.67	0.11	**0.66**	0.00	0.70	0.10	**0.66**	4.80	4.83	0.00	0.68
soybean2	0.17	0.49	0.25	0.53	0.01	**0.48**	0.22	0.49	4.78	4.88	0.01	0.50
soybean3	0.15	0.61	0.22	**0.57**	0.01	0.67	0.23	0.58	4.80	4.82	0.01	0.66

Table 2. Comparison of errors on training and testing sets obtained by Regularization Network with different kernel functions. For each task, the lowest error on the testing set is highlighted.

Komprese webového uložiště*

Jan Lánský, Leo Galamboš, and Katsiaryna Chernik

Univerzita Karlova, Matematicko-fyzikální fakulta, Malostranské nám. 25, 118 00, Praha 1
zizelevak@gmail.com, leo.galambos@mff.cuni.cz, kchernik@centrum.cz

Abstrakt *EGOTHOR je fulltextový systém, který stahuje z Webu dokumenty, indexuje je a umožňuje v nich vyhledávat. Vytvářené výsledkové listiny obsahují kromě URL dokumentu i výstřižek, který stručně vystihuje nalezený zásah. Tento výstřižek je možné téměř výhradně sestavovat ze znalosti celého originálního dokumentu (typicky ve formátu HTML), což implikuje nutnost uchovávat celé indexované dokumenty v rámci indexu.*

Při praktickém nasazení jsme se proto setkali s problémem volby vhodného kompresního algoritmu, který by nám pomohl redukovat nároky na diskový prostor. Nabízí se možnost použití obecných kompresních metod jako gzip nebo bzip2, ale může být výhodnější navrhnout vlastní metody, které budou využívat charakteru dokumentu. Existují specializované textové kompresní algoritmy a dále metody komprimující XML dokumenty. Vhodným spojením těchto dvou přístupů lze dosáhnout optimální úrovně komprese.

1 Motivace

EGOTHOR [6] je fulltextový systém implementovaný na platformě Java2. Pro volbu této platformy byly několikeré důvody: přenositelnost, snadná údržba kódu a rychlé zapojování modulů třetích stran. S ohledem na velký objem dat, které dokáže systém zpracovat, je problémem vhodná redukce prostorové náročnosti. Zde se nejedná pouze o velikost invertovaného indexu (který je pochopitelně komprimován), ale zejména o databázi s originálními dokumenty staženými z Webu. Tato databáze hraje významnou roli při generování popisků zásahů ve výsledkových listinách a je největším konzumentem prostoru v rámci celého systému.

Systém druhé generace (EGOTHOR2) se skládá z webového robota schopného stahovat stránky rychlostí 700-1000 stránek za sekundu, indexačního modulu o průtoku 500-700 dokumentů za sekundu, a nakonec vyhledávacího modulu, který pracuje nad vytvořeným invertovaným indexem. Tyto orientační výkonnostní údaje vycházejí z testů na samostatném serveru s AMD Opteron 246 a 100Mbps připojením a mohou být významně ovlivněny velikostí stahovaných dokumentů a strukturou procházené části Webu.

* Práce byla částečně podporována Národním programem výzkumu – v rámci projektu Informační společnost 1ET100300419.

Celý systém je dále volitelně doplněn o celou řadu lingvistických modulů, které následně zvyšují informační kvalitu výsledků za cenu odpovídajícího snížení průtoku dat systémem. Z výpočetně nejméně náročných modulů lze uvést například stemmer [5], který má pozitivní vliv na velikost invertovaného indexu.

Tvorba popisků probíhá pomocí běžných lingvistických algoritmů a to s ohledem na strukturu originálního textu a slova obsažená v uživatelském dotazu. Systém nejprve nalezne zásahy, respektive jejich 64bitové identifikátory, vybaví jejich textový obsah z databáze originálních dokumentů, a poté již vytvoří adekvátní výstřižky do výsledkových listin.

V základní verzi systém komprimuje bázi textů pomocí algoritmu gzip [4]. Je ale otázkou, zda je taková komprese vhodná pro HTML dokumenty, a zda není možné nalézt adekvátní náhradu na bázi jiné (nebo i blízké) kompresní metody.

S ohledem na požadované nasazení je cílem taková metoda, která zajistí kvalitnější kompresní poměr, stejný nebo lepší dekompresní čas než gzip, přičemž čas komprese nemusí být tak rychlý jako u zmiňovaného gzip-u.

2 Komprese

Komprese slouží ke zmenšení objemu dat, aby zabrala na pevném disku menší prostor, případně menší kapacitu přenosové linky při jejich transportu. Fulltextový systém EGOTHOR při své práci nashromáždí a dále využívá obrovské množství dat. Webových stránek jsou řádově miliardy a mají průměrnou velikost okolo 10-20 kB [17]. Problém s nedostatečnou velikostí disků lze částečně řešit právě kompresí.

Znalost struktury kódované zprávy může být velmi užitečná pro návrh úspěšné kompresní metody. Webové stránky jsou speciální typ dat. Základem je formát HTML [23], který je v principu tvořen textovými daty obalenými ve struktuře.

Pro kompresi můžeme rozhodnout nevyužít žádnou z vlastností HTML a zvolit obecnou kompresní metodu například gzip [4] nebo bzip2 [18]. Tím dosáhneme jistého stupně komprese, ale pravděpodobně nebude optimální. Tento přístup je praktikován v současné době, kdy se v EGOTHORu používá metoda gzip.

Druhá možnost spočívá ve využití textového charakteru dat a použití kompresních metod specializovaných na kompresi po slovech [21] či slabikách [12], podle jazyku dokumentu, tedy webové stránky. Jazyk lze buďto odhadnout z domény, ve které se stránka nachází, z hlavičky dokumentu, nebo přesněji určit statistickou analýzou textu.

Třetí varianta je založena na efektivním zakódování struktury značek, jejich názvů a atributů. Příkladem takovéto metody je XMill [13] pro kompresi dat ve formátu XML [22]. Zde se na samotné obsahy atributů používá nějaká obecná kompresní metoda, třeba gzip. Problémem je pro tuto metodu velká volnost v pravidlech pro strukturu HTML, je povoleno špatné hnízdění značek, vícenásobné atributy a další odlišnosti. V tomto případě bychom museli naše data upravit do dobře formovaného formátu XML, nebo upravit stávající metody pro práci se špatně formovanými daty.

Předchozí varianta se snažila samostatně efektivně zakódovat strukturu značek a atributů, ale samotný textový obsah se kódoval neefektivně pomocí obecné kompresní metody. Řešením tohoto problému pro XML data se zabývala práce [8], kde se struktura kóduje pomoci XMill a textový obsah slabikovými a slovními metodami. Úpravou těchto metod pro HTML, můžeme dosáhnout zajímavých výsledků.

3 Textové kompresní metody

Pro kompresi textů se používají zejména dva následující principy – po slovech a po slabikách. Slovní metody jsou starší [21] a existují implementace velkého množství klasických algoritmů pracujících nad abecedou slov: například Huffmanova kódování [21], LZW [3], Burrows Wheelerova transformace [9], PPM [1] nebo Aritmetické kódování [16]. Slabikové metody se používají poměrně krátkou dobu [12] a existují zatím pouze implementace Huffmanova kódování a LZW.

Při adaptaci klasických znakových metod na slovní nebo slabikové metody se musí většinou podstatně modifikovat datové struktury, aby byly schopny pracovat místo s 256 znaky s předem neurčeným a navíc vysokým počtem slov či slabik. Při kompresi nad velkou abecedou musí kodér informovat dekodér, jaké prvky obsahuje abeceda, kterou používá. Nejčastěji se tento problém řeší přidáním zakódované abecedy k vlastnímu zakódovanému dokumentu [10].

Slovní textové metody vycházejí úspěšněji na jazycích s jednoduchým tvaroslovím (např. angličtina), slabikové metody jsou výhodné pro jazyky s bohatým tvaroslovím (např. čeština, němčina) [11].

3.1 Slovní metody

Při použití slovních kompresních metod [21] je nutné rozdělit dokument na posloupnost slov a neslov. Jako slova se nejčastěji označují řetězce písmen a číslic, neslovy jsou pak řetězce zbylých znaků. Při dělení dokumentu se hledají maximální alfanumerické řetězce, které se označují za jednotlivá slova. Řetězce znaků, které zůstanou mezi slovy, se prohlásí za jednotlivá neslova.

Můžeme tedy vycházet z předpokladu, že v dokumentu se pravidelně střídají slova a neslova. Dále se používá heuristika, že slovo bývá obvykle následováno speciálním typem neslova - „mezerou“. Můžeme si tedy dovolit neslova „mezera“ vynechat a nekódovat. Správná dekomprese se zaručí tím, že pokud dekódujeme posloupnost dvou po sobě jdoucích slov automaticky mezi ně vložíme mezeru, která byla při kompresi vynechána.

Pro praktické použití se neuvažují neomezeně dlouhá slova a neslova, ale jejich délka se omezuje nějakou rozumnou konstantou. Příliš dlouhé řetězce se rozdělují, a aby byl zachován model střídání slov a neslov, musíme mezi ně vložit řetězec nulové délky opačného typu (prázdné slovo, neslovo). Například pokud rozdělíme slovo na dvě slova, musíme mezi ně vložit prázdné neslovo.

3.2 Slabikové metody

Slabikové kompresní metody [12] musí textový dokument rozdělit nejprve na slova a ty navíc dále na slabiky. U komprese po slovech se velmi často se používá dělení textu na řetězce slov (alfanumerické znaky) a neslov. Pro slabikovou kompresi je však vhodnější následující dělení na pět druhů slov a slabik.

Slova obsahující pouze malá písmena označíme jako malá (např. „ahoj“). Slova obsahující pouze velká písmena označíme jako velká (např. „AHOJ“). Slova, která začínají prvním písmenem velkým a zbylá písmena jsou malá označíme jako smíšená (např. „Ahoj“). Slova obsahující pouze číslice označíme jako číselná (např. „1982“). Slova obsahující pouze nealfanumerické symboly označíme jako speciální (např. „??!“). Malá, velká a speciální slova označujeme jako písmenná, speciální a číselná slova označujeme jako nepísmenná.

Rozdělení textového souboru na slova provedeme hladovým algoritmem. Nyní můžeme provést jeho další dělení na slabiky. Dělení slov na slabiky nemusí být vždy jednoduché, někdy nemusí být ani jednoznačné, často vychází i z původu slova. Pro potřeby komprese je možné vystačit i s jistou aproximací správného dělení slov na slabiky bez výrazných dopadů na dosahovaný stupeň komprese. Algoritmů dělení na slabiky

může být velké množství a mohou se lišit v tom, jaké množství údajů o daném jazyku potřebují znát. Popíšeme si zde čtyři algoritmy, které vyžadují pouze minimální množství informací o daném jazyku, konkrétně potřebují pouze znát, která písmena jsou samohlásky a které souhlásky.

Všechny čtyři algoritmy mají společnou počáteční fázi. Nepísmenná slova prohlásíme za slabiky, nebudeme je tedy dále dělit. V písmenných slovech určíme, která jejich písmena jsou samohlásky a která souhlásky. Následně se vyhledají maximální bloky samohlásek. Blok samohlásek je souvislý úsek slova složený ze samohlásek, nejvýše délky 3. Maximální blok je takový, že nejde prodloužit o další samohlásku při zachování jeho délky nejvýše 3. Tyto maximální bloky samohlásek budou tvořit základ slabik, u každého bloku si pamatujeme jeho začátek a konec. Souhlásky, které jsou před prvním blokem, jsou přiřazeny prvnímu bloku a souhlásky, které jsou za posledním blokem, jsou přiřazeny poslednímu bloku.

Jednotlivé algoritmy se liší v tom, jakým způsobem přiřazují souhlásky, které se nacházejí mezi jednotlivými maximálními bloky samohlásek, k těmto blokům. Způsob přiřazování je patrný z jednotlivých názvů. Algoritmus universal left (P_{UL}) přiřadí všechny souhlásky mezi bloky samohlásek k levému bloku samohlásek. Algoritmus universal right (P_{UR}) přiřadí všechny souhlásky mezi bloky samohlásek k pravému bloku samohlásek. Algoritmus universal middle-right (P_{UMR}) v případě $2n$ souhlásek (sudého počtu) mezi bloky přiřadí ke každému bloku samohlásek n souhlásek. V případě $2n + 1$ (lichého počtu) souhlásek mezi bloky přiřadí k levému bloku samohlásek n souhlásek a k pravému bloku $n + 1$ souhlásek.

Algoritmus universal middle-left (P_{UML}) v případě $2n$ souhlásek (sudého počtu) mezi bloky přiřadí ke každému bloku samohlásek n souhlásek. V případě $2n + 1$ (lichého počtu) souhlásek mezi bloky přiřadí k levému bloku samohlásek $n + 1$ souhlásek a k pravému bloku n souhlásek. Výjimka proti tomuto pravidlu nastane v případě, že mezi bloky je jen jedna souhláska, pak se tato souhláska přiřadí k pravému bloku. Tím zajistíme, aby na konci slov nezůstávaly samostatné bloky samohlásek, což je hlavní nevýhoda algoritmu P_{UL}.

Uvedeme si následující příklad (tabulka 1). Rozdělíme české slovo odstrčenou na slabiky jednotlivými algoritmy. Bloky samohlásek zde jsou (po řadě) o, r, e, ou.

Slabikové kompresní metody vycházejí z předpokladu, že text je strukturován do vět a lze jej popsat následujícími pravidly: Věta začíná smíšeným slovem (první písmeno je velké, zbylá malá) a končí speciálním slovem, které obsahuje tečku. Ve větě se pravidelně

Metoda	Rozdělení
Správné dělení v češtině	od-str-če-nou
universal left P_{UL}	odst-rč-en-ou
universal right P_{UR}	o-dstr-če-nou
universal middle-left P_{UML}	ods-tr-če-nou
universal middle-right P_{UMR}	od-str-če-nou

Tabulka 1. Příklad - dělení slova *odstrčenou* na slabiky.

střídají malá a speciální slova. Začíná-li věta velkým slovem, pak se ve větě střídají velká a speciální slova.

Po rozložení slov na slabiky nastává s tímto modelem problém. Každé slovo má jiný počet slabik. Zatímco malé slovo je většinou následováno speciálním slovem, tak malá slabika může být následována jak malou slabikou, tak speciální slabikou.

Tento model se pak při kompresi používá k predikci typu další slabiky.

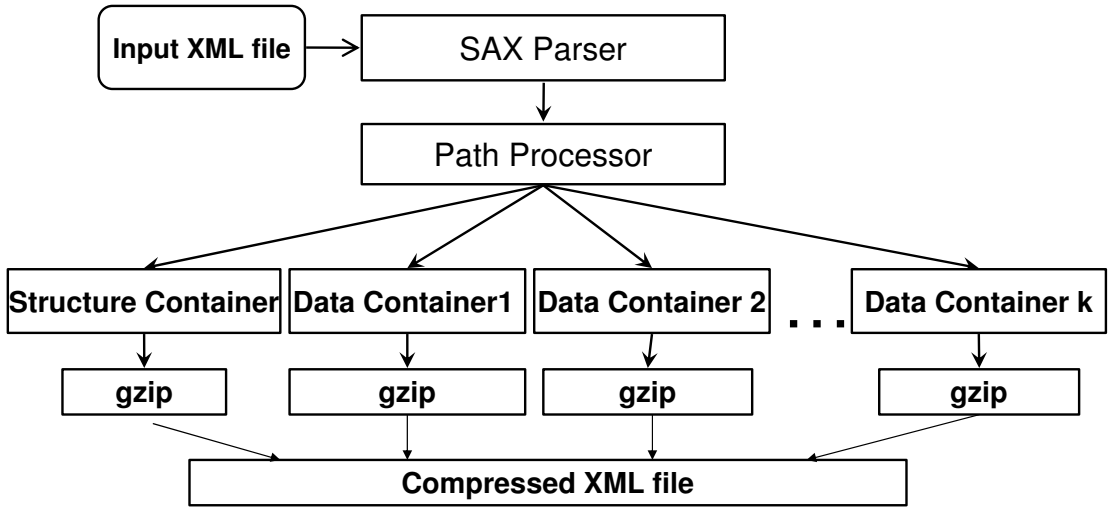
V článku [12] byly navrženy dvě kompresní metody LZWL a HuffSyllable pracující nad abecedou slabik. LZWL je slovníková metoda založená na metodě LZW [20], HuffSyllable je statistická kompresní metoda inspirována algoritmem HuffWord [21]. HuffSyllable používá adaptivní Huffmanovo kódování a pracuje na principu střídání pěti druhů slabik.

4 Metody pro kompresi struktury XML

Existuje celá řada algoritmů pro kompresi dat ve formátu XML. Jedním z prvních byl XMill [13], kterému se budeme věnovat podrobněji. Metoda XMLPPM [7] a mnoho dalších pracují na podobných principech, nebo jsou přímo z XMill odvozeny. Algoritmy XGrind [19] a XPress [15] umožňují navíc vyhledávání a pokládání dotazů nad komprimovanými daty, výměnou za horší kompresní poměr.

Algoritmus XMill je založen na následujících třech principech:

- **Oddělení struktury od dat** Za strukturu se považují značky, atributy a jejich uspořádání. Jako data označujeme posloupnost položek (textových řetězců), které jsou obsahem značek nebo hodnotami atributů.
- **Seskupení dat s podobným významem** Data s podobným významem jsou sloučeny do skupin nazývaných kontejnery a každý kontejner je komprimován odděleně. Pravidla pro seskupování musí sepsat uživatel pomoci jazyka XPath [24]. Například všechny obsahy značek `<name>` tvoří jeden kontejner, zatímco všechny obsahy značek `<phone>` tvoří druhý kontejner.
- **Aplikace různých kompresních metod na jednotlivé kontejnery** Každý kontejner může být zpracován jinou kompresní metodou. Například obsahy atributu `<name>` komprimujeme po-



Obrázek 1. Architektura kompresního programu XMill.

mocí textových kompresních metod, zatímco obsahy atributu `<phone>` komprimujeme metodami na kompresi celých čísel.

Architektura metody XMill (obrázek 1) je založena na třech výše uvedených principech. Dokument ve formátu XML je zpracován SAX parserem [14], který posíla SAX události do Path-procesoru. Zde probíhá oddělování struktury od dat a shlukování dat do kontejnerů podle podobnosti významů.

Předtím, než jsou data odeslána do kontejnerů lze na ně použít nějaký sémantický kompresor (např. text, celá čísla). Zda a na jaký kontejner se použije nějaký sémantický kompresor musí určit uživatel, což je nevýhodou této metody.

V jednom z kontejnerů je uložena struktura celého dokumentu. Obsahy značek jsou nahrazeny odkazy na čísla příslušných kontejnerů, názvy značek a atributů jsou nahrazeny odkazy do slovníku značek a atributů.

Na závěr se obsahy jednotlivých kontejnerů zakódují programem gzip a uloží do výsledného souboru.

5 Textové metody pro kompresi XML

V článku [8] jsou navrženy dvě metody XMillSyl a XMLSyl, které se snaží o spojení metod na kompresi struktury XML s metodami pro kompresi textu, konkrétně se slabikovými kompresními metodami LZWL a HufSyllable [12].

První z metod XMillSyl je modifikací XMill [13], kterým jsme se zabývali v předchozí kapitole. Rozdíl oproti původní metodě je až ve fázi komprese jednotlivých kontejnerů. V původní metodě XMill se používá obecná kompresní metoda gzip, která je v metodě XMillSyl pro datové kontejnery nahrazena slabikovými kompresními metodami LZWL a HuffSyllable. Kontejner se strukturou se nadále komprimuje pomocí metody gzip.

Metoda XMLSyl se snaží modifikovat existující slabikové metody LZWL a HuffSyllable, tak aby se značky nerozdělovaly na posloupnost slabik, ale aby se považovaly za jednu slabiku.

Metoda XMLSyl využívá SAX parser, který z dokumentu vytváří proud SAX událostí a ten je následně zpracováván v kodéru struktury. Kodér struktury vytváří dva slovníky pro kódování značek a elementů. Dále se v kodéru struktury nahrazují názvy značek a atributů ve vstupním soubor za odkazy do příslušných slovníků. Modifikovaný soubor se ukládá od datového kontejneru, který se následně kóduje slabikovými metodami LZWL a HuffSyllable. Stejnými metodami se kódují i slovníky názvů značek a atributů.

6 Otevřené problémy

Metody komprimující strukturu XML vyžadují, aby byl dokument dobře formovaný, což u HTML dokumentu nebývá většinou dodržováno. Dokumenty ve formátu HTML často porušují správné uzávorkování značek (Př. `<a><b></a></b>`), anebo se v jedné značce vyskytují dva atributy se stejným jménem. Proto bude nutné modifikovat současné metody pro kompresi XML pro práci se špatně formovanými dokumenty, i když to přinese mírné snížení účinnosti komprese.

Textové kompresní metody, zvláště pak slabikové, dokáží vhodně využít informace o jazyce dokumentu. Jazyk dokumentu by měl být podle normy HTML zjistitelný z hlavičky dokumentu, ale ve skutečnosti často

nebývá uveden, nebo je uveden nesprávně. Také se často stává, že v jednom HTML dokumentu se používá více jazyků. Například jeden odstavec je česky, následující již anglicky.

Formát HTML používá omezenou a předem známou množinu značek a jejich atributů, využití této znalosti může vést ke zlepšení účinností komprese.

Část HTML dokumentů obsahuje různé skripty, které jsou z pohledu HTML brány jako komentáře. Skripty nemají strukturu přirozeného textu a textové kompresní metody zde nedosahují nejlepších výsledků.

V současné době se v EGOTHORu ukládá 1000 webových stránek do jednoho souboru, zejména z důvodu optimalizace tvorby indexu v robotu. Když komprimujeme tento celý soubor najednou, ztrácíme možnost přístupu k jednotlivým dokumentům bez nutnosti dekomprese všech dokumentů obsažených v tomto souboru. V budoucnu uvažujeme o kompresi jednotlivých HTML dokumentů samostatně. Ve výsledném souboru, který by obsahoval 1000 komprimovaných dokumentů by se poté daly jednotlivé dokumenty vyhledávat bez nutnosti dekomprese celého souboru. Na kompresi malých souborů jsou vhodné slabikové kompresní metody [11].

Zde popsane metody najdou uplatnění nejen pro ukládání kolekcí semistrukturovanych dokumentů, ale i pro efektivní práci s rozsáhlejšími položkami v datovém stohu [2].

7 Experimenty

V tomto článku jsme diskutovali několik přístupů ke kompresi webových stránek. Provedli jsme experimenty s těmito kompresními programy na souboru obsahujícím 1000 webových stránek ze slovinské domény *si*. Výsledky jsou uvedeny v tabulce 2.

V prvním sloupci tabulky je název metody, v druhém sloupci je uvedena velikost na kterou byl originální soubor zakomprimován a ve třetím sloupci je spočítán kompresní poměr (zakomprimovaný soubor / orignál).

Metoda	Komprimovano	Poměr
originál	21 007 005	100,00%
gzip	3 830 825	18,24%
bzip2	2 950 823	14,05%
LZWL slova	3 580 571	17,05%
LZWL slabiky	3 806 980	18,12%
HufSyllable slova	6 780 966	32,28%
HufSyllable slabiky	8 721 220	41,51%
XMill	—	—
XMLSyl	—	—
XMillSyl	—	—

Tabulka 2. Výsledky komprese souboru obsahujícího 1000 webových stránek ve slovinštině.

Slabikové a slovní verze LZWL a HuffSyllable používaly jazykové nastavení pro češtinu, protože pro slovinštinu nebylo dostupné, a to negativně ovlivnilo dosažené výsledky.

Metody Xmill, XMillSyl a XMLSyl jsou schopny pracovat pouze s dobře formovanými XML daty, což testovaný soubor nesplňoval a tedy se ho nepodařilo zkomprimovat.

Důležitou vlastností kompresních programů je čas komprese a dekomprese. Program gzip provede kompresi za dvě sekundy a dekompresi za méně než jednu sekundu. Program bzip je zhruba pětkrát pomalejší (9 sekund komprese, 4 sekundy dekomprese). Programy LZWL a HuffSyllable existují pouze v testovací verzi, která neměla za cíl snižovat časovou náročnost. Program LZWL provede kompresi i dekompresi (slabiková i slovní verze) zhruba za 20-25 sekund.

Z dosažených výsledků je patrné, že současné používání metody gzip není z hlediska dosaženého kompresního poměru příliš výhodné. Například textová metoda LZWL (varianta slova i slabiky) dosahuje lepších výsledků. Předpokládáme, že případné upravené verze XMLSyl nebo XMillSyl pro formát HTML by mohly dosáhnout ještě lepších výsledků. Velmi dobré výsledky by jsme mohli získat i modifikací metody bzip tak, aby pracovala nad abecedou slabik či slov a využívala struktury HTML.

8 Závěr

Fulltextový vyhledávací systém EGOTHOR při své činnosti shromažďuje a potřebuje uchovávat značné množství dat ve formátu HTML, které je nutno komprimovat. Zabývali jsme se výběrem vhodné kompresní metody, která by využívala jak textového charakteru HTML dokumentu, tak i jejich struktury. Taková metoda má nejlepší předpoklady pro dosažení optimálního stupně komprese.

V tomto článku jsme se zabývali zástupci ze tří skupin kompresních algoritmů: textových, specializovaných na XML a kombinovaných textových pro XML. Jako nejlepší řešení považujeme upravit stávající textové metody pro XML, tak aby byly schopny komprimovat HTML.

Reference

1. Adiego J., Feunte P., On the Use of Words as Source Alphabet Symbols in PPM. Data Compression Conference, IEEE CS Press, Los Alamitos, CA, USA, 2006, 435
2. Bednárek D., Obdržálek D., Yaghob J., Zavoral F.: Data Integration Using DataPile Structure: ADBIS 2005, Proceedings of the 9th East-European Conference on Advances in Databases and Information Systems, Tallinn, 2005

3. Dvorský J., Pokorný J., Snášel V., Word-Based Compression Methods for Large Text Documents. Data Compression Conference, IEEE CS Press, Los Alamitos, CA, USA, 1999, 523
4. Gailly J.L., Gzip Program and Documentation, 1993. Source code aviable from `ftp://prep.ai.mit.edu/pub/gnu/gzip-*.tar`
5. Galambos L., Dynamization in IR Systems. Mieczyslaw A. Klopotek, Slawomir T. Wierzchon, Krzysztof Trojanowski (Eds.), IIPWM, Proc. of the Int. IIS: IIPWM'04, Poland, 2004. ASC Springer 2004, ISBN 3-540-21331-7
6. Galambos L.: EGOTHOR. `http://www.egothor.org/`
7. Cheney J., Compressing XML with Multiplexed Hierarchical PPM Models. Data Compression Conference, IEEE CS Press, Los Alamitos, CA, USA, 2001, 163
8. Chernik K., Lánský J., Galamboš L.: Syllable-Based Compression for XML Documents. In: Snášel V., Richta K., and Pokorný J., Proceedings of the Dateso 2006 Annual International Workshop on DAtabases, TExts, Specifications and Objects. CEUR-WS, Vol. 176, 2006 21–31
9. Isal R.Y.K., Moffat A., Word-Based Block-Sorting Text Compression. Proc. 24th Australasian Computer Science Conference, Gold Coast, Australia, 2001, 92–99
10. Lánský J., Žemlička M.: Compression of a Dictionary. In: Snášel V., Richta K., and Pokorný J., Proceedings of the Dateso 2006 Annual International Workshop on DAtabases, TExts, Specifications and Objects. CEUR-WS, Vol. 176, 2006, 11–20
11. Lánský J., Žemlička M., Compression of Small Text Files Using Syllables. Data Compression Conference, IEEE CS Press, Los Alamitos, CA, USA, 2006, 458
12. Lánský J., Žemlička M., Text Compression: Syllables. In: Richta K., Snášel V., Pokorný J., Proceedings of the Dateso 2005 Annual International Workshop on DAtabases, TExts, Specifications and Objects. CEUR-WS, Vol. 129, 2005, 32–45
13. Liefke H., Suciu D., XMill: an Efficient Compressor for XML Data. In Proc. ACM SIGMOD Conference, 2000, 153–164
14. Megginson D.: SAX: A Simple API for XML. `http://www.saxproject.org`
15. Min J.K., Park M.J., Chung C.W., XPRESS: A Queriable Compression for XML Data. SIGMOD 2003, San Diego, CA, USA , 2003, 122–133
16. Moffat A., Neal R.M., Witten I.H., Arithmetic Coding Revisited. ACM Transactions on Information Systems, 16, 1998, 256–294
17. O'Neill E.T., Lavoie B.F., Bennett R., Trends in the Evolution of the Public Web: 1998–2002. D-Lib Magazine, 2003, 1082–9873
18. Seward J., The bzip2 and libbzip2 official home page. `http://sources.redhat.com/bzip2/`
19. Tolani P., Haritsa J.R., XGrind: A Query-Friendly XML Compressor. In Proc. IEEE International Conference on Data Engineering, 2002
20. Welch T.A., A Technique for High Performance Data Compression. IEEE Computer, 17(6), 1984, 8–19
21. Witten I., Moffat A., Bell T.: Managing Gigabytes: Compressing and Indexing Documents and Images. Van Nostrand Reinhold, 1994
22. World Wide Web Consorcium: Extensive Markup Language (XML). `http://www.w3.org/XML/`
23. World Wide Web Consorcium: HyperText Markup Language (HTML). `http://www.w3.org/MarkUp/`
24. World Wide Web Consorcium: XML Path Language (XPath). `http://www.w3.org/TR/xpath`

Contingent parallel plans based on operators

Marián Lekavý and Pavol Návrat

Slovak University of Technology, Faculty of Informatics and Information Technologies
Ilkovičova 3, 842 16 Bratislava, Slovakia,
lekavy@fiit.stuba.sk, navrat@fiit.stuba.sk

Abstract. *This paper presents a draft of new approach to contingent planning, which is unlike other existing contingent approaches also applicable to parallel planning. This approach also allows actions with different time of execution. The approach is based on a new algorithm for creation of parallel plans, also presented in this paper. The presented algorithm doesn't need two steps (planning and parallelization), but creates parallel plans in a single step.*

1 Introduction

The first part of this paper presents a draft of a new approach to contingent planning, which is unlike other existing approaches also applicable to parallel planning, i.e. it is able to create contingent plans with parallel action execution.

This paper also introduces a new algorithm for creating parallel plans based on STRIPS-like action model. Unlike existing algorithms, the introduced Multi-Action Backchaining (MAB) algorithm creates parallel plans directly, without the need of future parallelization. Moreover, it has the features needed to be extended into a contingent parallel planner, which is also presented in this paper.

The creation of parallel plans, where several actions can be executed simultaneously, is very important, not only in the field of multi-agent systems, but also in modern production or logistics systems and other domains. Moreover, in some environments, the results of individual actions are not fully predictable, so it is important to prepare alternative plans for the most probable action outcomes.

Contingency is usually added to the planning process by allowing the actions to have several different results (with different probabilities). If an action with more results is used in a plan, a contingent planner splits the plan into two (or more) alternative plans - one for each possible result. This is quite simple for sequential plans, with only one action executed at a time. On the other hand, several problems arise if we allow parallel execution of actions.

Parallel planning is a generalisation of the classical planning problem [17]: Given a description of the initial state of the world, description of the goal and a description of the possible actions determine a plan, i.e., a sequence of actions that transforms states fitting the initial configuration of the world into one of the goal states. The only change for parallel planning is that a plan is not a sequence of actions, but a set of actions with time constraints on action execution.

In general, there are only two possible methods for creating parallel plans: Creation of non-parallel plan followed by subsequent parallelization and direct creation of a parallel plan.

The majority of different approaches to parallel planning uses the first method – creation of non-parallel plan and subsequent parallelization. The non-parallel plan can be created by arbitrary planning technique: STRIPS [8] and STRIPS-like reasoning, based on state-space search (e.g. GRT [14] using backchaining or HSP [4] using forward chaining); plan-space search, based on STRIPS-like actions, where actions and restrictions are added to partial plans until the final partially ordered plan is created [12]; Markov Decision Process (MDP) [2], [13]; approaches based on HTN [15] using manual hierarchical decomposition of actions (or tasks), which is used to create partially ordered plans (like NOAH [6], PGP [10] or STEAM [16]).

Separating the planning process and parallelization brings an advantage of simplifying the whole process. There are, however, several problems resulting from the separation:

- Sub-optimal solution. If the linear plan is optimal (for example with respect to execution time) and the parallelization process is optimal too, the resulting parallel plan is not guaranteed to be optimal. A longer linear plan can be better parallelizable, yielding a shorter parallel plan.
- Problems with contingency. With contingency, we cannot fully predict the effect of alternate action's results on the remaining actions before we parallelize the whole plan. During the first step - creation of linear plan - we don't know which actions will be executed simultaneously after the parallelization.
- Complexity. The parallelization of a linear plan is, in general, a very complex computational problem on its own. Optimal parallelization is NP-hard and even sub-optimal modifications are usually NP-hard [1]. (For comparison, the

planning process based on STRIPS-like operators is PSPACE-complete in general [5].)

There is also a small group of approaches, which are able to generate parallel plans directly (for example some approaches based on GraphPlan [3] or the contingent planner ZANDER [11]), but all these approaches use the simplification, that all actions are executed exactly in one time quantum (or immediately). For most real systems, this simplification is too restricting and these approaches are almost unusable if there are different times for actions execution.

For these reasons, we designed a new algorithm for creating parallel contingent plans. As a part of this algorithm, we also developed the Multi-Action Backchaining (MAB) algorithm for creating parallel plans directly, without the need of further parallelization. At the same time, MAB allows different execution times for individual actions, removing the main restriction for direct parallel plan creation. While the contingent planner is still in the design phase, the MAB algorithm (the non-contingent part of the parallel planning algorithm) is fully implemented and verified on a modified bricks domain.

The basic MAB algorithm is described in section 2. Section 3 describes the modifications of the basic MAB algorithm which will allow it to plan with contingency.

2 Multi-action backchaining algorithm overview

This section provides the MAB (Multi-Action Backchaining) algorithm's basic outline. This is the non-contingent planning algorithm, which is the base of the contingent planner described in the next section. First, the model of action and state is described, together with the model of action execution. Next, the parallel plan is described, followed by the algorithm pseudocode.

The same principle, MAB is based on, could also be used to modify the standard forward-chaining algorithm to allow parallel forward-chaining. This is, however, not in the scope of this paper.

2.1 Model of action and state

The basic model of action is a modification of the STRIPS-like action model, especially the Action Description Language (ADL) [9]. Executed actions are created by applying operators to the current state. An operator is defined as follows: $Op = (Pre, Eff, Pro, t)$ where Pre, Eff, Pro are sets of literals, Pre is a precondition, Eff the effect of the operator and Pro is the set of protected literals, which may not be used or changed by other operators during Op execution and t is the time of operator execution. The sets of literals Pre, Eff and Pro do not have to be grounded (i.e. can contain variables as parameters) and may be negated. The sets of literals Pre and Eff are interpreted as conjunctions of literals, forming the operator precondition and effects.

The following example shows an operator from the bricks domain, for lifting-up a box (box1) from another box (box2) by a mechanical arm (arm), with variables 'box1', 'box2' and 'arm':

```
Operator: liftUp(box1, box2, arm)
Precondition: on(box1, box2), free(box1),
   idle(arm)
Effects: ¬on(box1, box2), ¬free(box1),
   ¬idle(arm), holds(arm, box1), free(box2),
   lift(box1)
Protects: free(box1), free(box2), idle(arm)
Time: 20s
```

When an operator is selected to be executed, it forms an action:

$A = (Op, M, t_{start})$,

or

$A = (Pre, Eff, Pro, t, M, t_{start})$,

where Op is the executed operator, M is mapping of operator variables (variables appearing in literals of Pre, Eff, Pro) to constants and state variables. Additionally, t_{start} is the real time of start of action execution, counting backwards from the current state (every executed action is always bound to a state).

Like in all non-parallel backchaining algorithms, the plan search is performed by space-state search from the final state. The state representation in the Multi-Action Backchaining (MAB) contains a set of valid literals (L_{val}). The difference to other approaches is that the state representation also contains the set of currently executed actions (A_{ex}). Additionally, a state also contains constraints for state variables (C), e.g. inequality of two variables or a variable and a constant. Formally, a state $S = (L_{val}, A_{ex}, C)$.

The action (A) execution consists of two steps: the start of execution and the end of execution. At the start of execution, the action precondition has to be fulfilled: $Pre(M) \subseteq L_{val}$. $Pre(M)$ is the set of precondition literals (Pre) after applying of mapping M. At the same time, the executed action is added to the set of executed actions: $A_{ex} = A_{ex} \cup \{A\}$.

At the time of action end, the world state is modified according to $Eff(M)$. Negations of literals in $Eff(M)$ are removed from the current state and literals in $Eff(M)$ are added: $L_{val} = (L_{val} \backslash \neg Eff(M)) \cup Eff(M)$. ($\neg Eff(M)$ is a shortened notation for the set $\{l | \neg l \in Eff(M)\}$) At the same time, the executed

action is removed from the set of executed actions: $A_{ex} = A_{ex} \backslash \{A\}$.

The literal set $Pro(M)$ (literals in Pro after applying of mapping M) contains literals, which are protected during the action execution (in the time between the start and the end of action execution) - other actions using these literals in their Pre, Eff or Pro are not allowed to be executed if the scope of validity of Pre, Eff or Pro intersects with the validity scope of the executed action's Pro. Validity scope for Pre is at the start of action, for Eff at the end and Pro during the whole time of execution.

2.2 The multi-action plan

In the classical planning problem, a plan is a sequence of actions, leading from the initial state to the final state: $P = S_f A_1 S_1 A_2 S_2 A_3 \dots A_n S_n$, where A_i is the i^{th} executed action and S_i is the state at start of action A_i. S_n is the initial state, S_f the final state. The sequence is written in reverse order because speaking about backchaining and the planning algorithm starts at the final state (S_f).

In MAB, a plan is a set of actions instead of a sequence. Each action in a plan has a time of start and a time of end. Anyway, a sequence of states can be built: $P = S_f \{A_1\} S_1 \{A_2\} S_2 \{A_3\} \dots \{A_n\} S_n$. In this case, $\{A_i\}$ is the set of actions, which end in the state S_{i-1}. Time distance between states S_i and S_{i-1} is equal to minimum of the time distances from S_{i-1} to starts of actions executed in S_{i-1} (including actions from $\{A_i\}$, which end in S_{i-1}). An example of a plan can be seen in the Figure 1.

An action (A) is added to a plan before some state S_i, if its effects (Eff) fulfil some literals from the state's valid literals (L_{val}). At the same time, the action must not cancel literals from L_{val} by adding a negation of a literal from L_{val}. Also, A (i.e. Pro - protected literals of A) must not collide with other, already executed, actions. More actions may be added at the same time, but with respect to constraints resulting from other actions and current state.

2.3 Generating a new state – the algorithm

The MAB algorithm searches the state-space backwards from the final state. At start, the state-space set contains only the final state: $StS = \{S_f\}$. At each step, one state (S) is selected from StS, a set of actions $\{A_i\}$ executable in this state is generated (actions which end in this state and fulfil some of the state's literals), and these actions are applied to a copy of the current state, resulting in a new state (S'). This new state is then moved backward in time to the latest time when some action being executed in this state

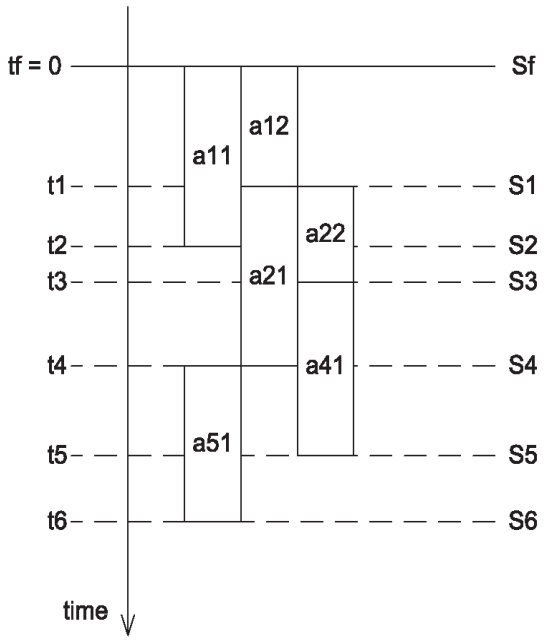


Fig. 1. Example of parallel plan. The vertical axis represents time, relative to the end of plan. The horizontal axis does not have any interpretation. States of plan represent points in time, where actions start or end.

(one of the added actions or one of the actions which already were executed in S) starts.

The basic algorithm of generating a new state in the backchaining scheme is following:

1. Select one work state.
   The work state ($S = (L_{val}, A_{ex}, C), S \in StS$) can be selected according to some value function - for example the time distance to the final state. This way, plans optimal according to this metrics are generated.
2. Choose actions.
   A set of actions $A_{gen} = \{A_i = (Pre_i, Eff_i, Pro_i, t_i, M_i, t_{start_i} = t_i)\}$ is generated. If A_{ex} is not empty ($A_{ex} \neq \emptyset$), an empty set of actions can be generated too ($A_{gen} = \emptyset$). Each of these actions fulfils part of the state S literals: $Eff_{A_i}(M_i) \cap L_{val} \neq \emptyset$. At the same time, the selected actions do not cancel any valid literals: $\{l | \neg l \in Eff_{A_i}(M_i)\} \cap L_{val} = \emptyset$ and the selected actions do not interfere with each other, nor with already executed actions (A_{ex}). These conditions generate constraints on action variables (C_{av_i}). These constraints are then mapped to the state variables and constants, using the mapping M_i: $C_{av_i}(M_i) = C_{sv_i}$. If such mapping is not possible (because of colliding constraints), backtracking starts.

3. Copy the work state.
   A new state S' is created as copy of S: $S' = S = (L'_{val} = L_{val}, A'_{ex} = A_{ex}, C' = C)$.
4. Add chosen actions to the new state.
   The chosen actions (A_{gen}) are added to S': $A'_{ex} = A_{ex} \cup A_{gen}$. At the same time, the effects of added actions are removed from literals in the state S': $L'_{val} = L_{val} \backslash \{Eff_{A_i}(M_i)\}$ for each $A_i \in A_{gen}$ and the constraints from actions are added to the state constraints: $C' = C \cup C_{sv_i}$, for each $A_i \in A_{gen}$.
5. Shift back time for the new state.
   Nothing more can happen at the current time to S', because all actions ending here were already added. So we have to move backward in time to the nearest event, which is the begin of an action. We move through time by changing execution times (t_{start_i}) of all actions in A'_{ex} using following rule: $t_{start_i} = t_{start_i} - \min_j(t_{start_j})$ for each $A_i \in A'_{ex}$.
6. Apply started actions on the new state.
   After shifting back the time, some of the actions are at their starting time ($A_{start} = \{A_i, A_i \in A'_{ex}, t_{start_i} = 0\}$). These actions' preconditions are applied to the state: $L'_{val} = L'_{val} \cup Pre_{A_i}(M_i)$, for each $A_i \in A_{start}$. Proper selection of actions in step 2 guarantees, that this step can be correctly executed and no collisions of literals occur. At the same time, these actions are removed from the current state: $A'_{ex} = A'_{ex} \backslash A_{start}$.
7. Add the new state to the state space.
   The new state S' is added into the state-space: $StS = StS \cup \{S'\}$. If the new state is identical with some previously found state, it is only added if its distance to the final state is smaller than the distance of the identical state. Also, there can be an additional constraint, which can prevent the state from being added, e.g. there can be a set of forbidden states. If the new state is a subset of the initial state, then the algorithm found a plan.

These are only the basic steps. Backtracking occurs if the steps 2 or 7 fail for some reason (for example there is no group of actions which could be executed in step 2). In the case of backtracking, no new state is created and the sequence begins with step 1 again.

3 Modifications towards contingency

This section describes the modifications of MAB needed to become a contingent planner. The first modification is in the model of action, second in the planning process.

3.1 Modifications of MAB

The operator contains several possible effects Eff_i with different probabilities $P(Eff_i)$: $Op = (Pre, \{(Eff_i, P(Eff_i)\}, Pro, t)$, while $\sum_i P(Eff_i) = 1$. The action (executed operator) then additionally contains the identification of one of the possible effects i_{Eff}: $A = (Op, M, t_{start}, i_{Eff})$. This way, an action represents one possible execution of an operator, with one given result: $A = (Pre, Eff_{i_{Eff}}, Pro, t, M, t_{start})$.

With actions defined this way, there is no need for any substantial change in the MAB algorithm. The only change is in the 2^{nd} step of the algorithm (described in the previous section). The algorithm chooses possible operators as before, but additionally it also selects one of the operators' effects. Remaining steps of the algorithm remain unchanged.

3.2 Contingent framework around MAB

The modifications of MAB are enough to account for different results of operators, but we need additional steps to acquire a contingent planner. These steps, however, are not made by MAB itself. A contingent framework has to be built around MAB. The basic idea is similar to Just-In-Case planning [7], but modified to allow parallel planning. The basic contingent algorithm around MAB is following:

1. Using MAB, find one plan (or several plans) from the initial state to the final state: $PlS = \{MAB(S_{init}, S_{final})\}$, where PlS is the plan space (set of found plans) and $MAB(S_{init}, S_{final})$ is the result of applying MAB on the initial and final state.
2. Find one alternate action's result among all actions in the previously found plans: $Eff_{j,A_i}, Eff_{j,A_i} \in A_i, A_i \in PlS$, where A_i is one action from the plan space and Eff_{j,A_i} is one of possible results of A_i. The selected result has to have the highest probability among all possible alternate results in all previously found plans: $P(Eff_{j,A_i}) = max_{k,l}\{P(Eff_{k,A_l}), Eff_{k,A_l} \in A_l, A_l \in PlS\}$.
3. Create an alternate state S', which is the result of applying the alternate result (from step 2.) instead of the result considered in the original plan. $S' = (L'_{val} = (L_{val} \backslash Eff_{k,A_i}(M)) \cup Eff_{j,A_i}(M), A_{ex}, C)$, where $S = (L_{val}, A_{ex}, C)$ is the state (from MAB), where the action A_i with the alternate result Eff_{j,A_i} ends. Eff_{k,A_i} is the "previous" result of A_i, leading to state S. M is the mapping of operator variables to constants and state variables (determined in step 2. of MAB).
4. Using MAB, find a plan from the alternate state (from step 3) to the final state. Add this plan to the set of found plans. $PlS = PlS \cup \{MAB(S', S_{final})\}$

5. Repeat steps 2.-4., until all possible results with probability higher than a threshold ($P(Eff_{j,A_i}) > Threshold$) are processed.

The probability of a result (used in step 2.) is following: $P(Eff_{j,A_i}) = P(A_i) * P(Eff_{j,A_i}|A_i)$, where $P(A_i)$ is the probability that the branch of plan containing action A_i will be executed. $P(A_i)$ is computed as the product of probabilities of results, leading to execution of this action, but taking only the actual alternative plans (PlS) into account. $P(Eff_{j,A_i}|A_i)$ is simply the probability of this effect if action A_i is executed (this probability is a part of the operator definition). As we can see, the contingent framework always processes the most probable alternative. This is responsible for the following features of the algorithm:

- Most probable alternatives are searched at first. The contingent plan becomes more exact after every iteration of the algorithm.
- It is possible to predict the probability of success of the plan.
- With enough computation time, the plan is better, accounting for more alternative contingent variants.
- Without enough computation time, there is at least some plan, accounting for the most probable variants.

4 Conclusion

A new approach to generating parallel plans (the MAB algorithm) was introduced. A prototype is already completed and verified successfully on a modified bricks domain, where the bricks are moved with several arms in parallel. If a finite plan exists, it is always found. If the processed states are selected according to some metrics (e.g. time or action cost), plans optimal according to this metrics are found. The same principle, MAB is based on, could also be used to modify the standard forward-chaining algorithm.

The contingent framework is only in the design phase. It is based on the MAB algorithm, however it should be possible to use it with other parallel planning techniques too. The main advantage of the designed contingent planner is, that it is able to use additional computing time to enhance the plan already found.

Acknowledgements: This work has been partially supported by Science and Technology Assistance Agency under the contract No. APVT-20-007104. This work has been partially supported by the Grant Agency of Slovak Republic grant No. VG 1/3102/06.

References

1. Backstrom C., Computational Aspects of Reordering Plans. In: Journal of Artificial Intelligence Research 9, 1998
2. Bellman R.E., Adaptive Control Processes: A Guided Tour. Princeton University Press, Princeton, New Jersey, 1961
3. Blum A., Furst M., Fast Planning Through Planning Graph Analysis. In: Artificial Intelligence 90, Elsevier Science Publishers Ltd., Essex, UK, 1997, 281–300
4. Bonet B., Geffner H., Planning as Heuristic search. In: Artificial Intelligence. Special issue on Heuristic Search. Vol. 129, 2001, 5–33
5. Bylander T., The Computational Complexity of Propositional STRIPS Planning. In: Artificial Intelligence 69, Elsevier Science Publishers Ltd., Essex, UK, 1994, 161–204
6. Corkill D.D.: Hierarchical Planning in a Distributed Environment. In: IJCAI-79, 1979
7. Dearden R., Meuleau N., Ramakrishman S., Smith D., Washington R., Incremental Contingency Planning. In: ICAPS-03: Proceedings of the Workshop on Planning under Uncertainty and Incomplete Information, Trento, Italy, 2003, 38–47
8. Fikes R.E., Nilsson N.J., STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving, In: Artificial Intelligence 2, Elsevier Science Publishers Ltd., Essex, UK, 1971, 189–208
9. Gelfond M., Lifschitz V., Representing Action and Change by Logic Programs. In: Journal of Logic Programming. Vol. 17, 1993, 301–321
10. Lesser V.R., et al., Evolution of the GPGP/TAEMS Domain-Independent Coordination Framework. In: Autonomous Agents and Multi-Agent Systems. Vol. 9, No. 1, Kluwer Academic Publishers, 2004, 87–143
11. Majercik S.M., Littman M.L., Contingent Planning under Uncertainty via Stochastic Satisfiability. In: Artificial Intelligence 147, Special Issue on Planning with Uncertainty and Incomplete Information, Elsevier Science Publishers Ltd., Essex, UK, 2003,
12. Navrat P., et al., Umela inteligencia, Vydavatelstvo STU, Bratislava, Slovensko, 2002, ISBN 80-227-1645-6
13. Puterman M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley, New York, 1994
14. Refanidis I., Vlahavas I., The GRT Planner: Backward Heuristic Construction in Forward State-Space Planning. In: Journal of AI Research, Vol. 15, 2001, 115–161
15. Sacerdoti E.D., A Structure for Plans and Behavior, Elsevier-North Holland 1977
16. Tambe M., Towards Flexible Teamwork. In: Journal of Artificial Intelligence Research. Vol. 7, 1997, 83–124
17. de Weerdt, M., ter Mors A., Witteveen C., Multi-Agent Planning: An Introduction to Planning and Coordination. Handout of the European Agent Summer School, 2005, 1–32

Using dimension reduction methods for image retrieval

Pavel Moravec and Václav Snášel

VŠB – Technical University of Ostrava, Department of Computer Science, FEECS
pavel.moravec@vsb.cz

Abstract. *In this paper, we compare performance of several dimension reduction techniques, namely LSI, NMF, SDD and FastMap. The qualitative comparison is based on rank lists and evaluated on a collection of faces from the Olivetti Research Lab. We compare the quality of these methods from both the visual impact, quality of generated "eigenfaces" and retrieval performance.*

Keywords: SVD, information retrieval, random projection, FastMap, NMF, rank lists, eigenfaces, intrinsic dimensionality

1 Introduction

Methods of human identification using biometric features like fingerprint, hand geometry, face, voice and iris are widely studied. Face recognition (FR) is very complex problem, mainly due to significant intra-class variation in appeareance due to pose, facial expression, aging, illumination and imaging condition. A great number of different approaches to FR have been proposed in the last decades [9,1].

The *information retrieval* [2,7] deals among other things with storage and retrieval of multimedia data which can be usually represented as vectors in multidimensional space. This is often used for *image retrieval*, where we store a *collection* of images. The images have to be transformed to the image vectors first, so that we can index them – we usually convert all images to shades of gray (because there is a problem with color information) and resize them to a common size. Since the resulting dimension is quite high (width × height of resized image), we are forced to extract some common *features* from images and index them instead of whole image. The extraction process is either based on some heuristics, or fully independent on the collection properties.

Singular value decomposition (*SVD*) adds an important step to the indexing process [6]. SVD was already successfully used for automatic feature extraction. In case of face collection (such as our test data), the *base vectors* can be interpreted as images, describing some common characteristics of several faces. These base vectors are often called eigenfaces. For a detailed description of eigenfaces, see [11].

However SVD is not suitable for huge collections and is computationally expensive, so other methods of dimension reduction were proposed. We test two of them – *Semi-Discrete decomposition* and *FastMap*, a pivot-based method based loosely on *Multi-Dimensional Scaling*.

Because the data matrix does have all elements non-negative, we can use another, new method, called *non-metric factorization* (*NMF*).

The rest of this paper is organized as follows. In the second section, we mention image retrieval. The third section explains used dimension reduction methods. In the fourth section, we briefly describe qualitative measures used for evaluation of our tests. In the next section, we supply results of tests for several methods on ORL face collection. In conclusions we give ideas for future research.

2 Dimension reduction methods

We used four methods of dimension reduction for our comparison – Singular Value Decomposition, Semi-Discrete Decomposition, Non-metric factorization and FastMap, which are briefly described bellow.

2.1 Singular value decomposition

SVD [3] is an algebraic extension of classical vector model. It is similar to the PCA method, which was originally used for the generation of eigenfaces. Informally, SVD discovers significant properties and represents the images as linear combinations of the base vectors. Moreover, the base vectors are ordered according to their significance for the reconstructed image, which allows us to consider only the first k base vectors as important (the remaining ones are interpreted as "noise" and discarded). Furthermore, SVD is often referred to as more successful in recall when compared to querying whole image vectors [3].

Formally, we decompose the matrix of images A by *singular value decomposition* (*SVD*), calculating singular values and singular vectors of A.

We have matrix A, which is an $n \times m$ rank-r matrix and values $\sigma_1, \ldots, \sigma_r$ are calculated from eigenvalues of matrix AA^T as $\sigma_i = \sqrt{\lambda_i}$. Based on them, we can calculate column-orthonormal matrices $U = (u_1, \ldots, u_r)$ and $V = (v_1, \ldots, v_r)$, where $U^T U = I_n$ a $V^T V = I_m$, and a diagonal matrix $\Sigma = diag(\sigma_1, \ldots, \sigma_r)$, where $\sigma_i > 0, \sigma_i \geq \sigma_{i+1}$.

The decomposition

$$A = U\Sigma V^T$$

is called *singular decomposition* of matrix A and the numbers $\sigma_1, \ldots, \sigma_r$ are *singular values* of the matrix A. Columns of U (or V) are called *left* (or *right*) singular vectors of matrix A.

Now we have a decomposition of the original matrix of images A. We get r nonzero singular numbers, where r is the rank of the original matrix A. Because the singular values usually fall quickly, we can take only k greatest singular values with the corresponding singular vector coordinates and create a *k-reduced singular decomposition* of A.

Let us have k $(0 < k < r)$ and singular value decomposition of A

$$A = U\Sigma V^T \approx A_k = (U_k U_0) \begin{pmatrix} \Sigma_k & 0 \\ 0 & \Sigma_0 \end{pmatrix} \begin{pmatrix} V_k^T \\ V_0^T \end{pmatrix}$$

We call $A_k = U_k \Sigma_k V_k^T$ a k-reduced singular value decomposition (rank-k SVD).

Instead of the A_k matrix, a matrix of image vectors in reduced space $D_k = \Sigma_k V_k^T$ is used in SVD as the representation of image collection. The image vectors (columns in D_k) are now represented as points in k-dimensional space (the *feature-space*). For an illustration of rank-k SVD see Figure 1.

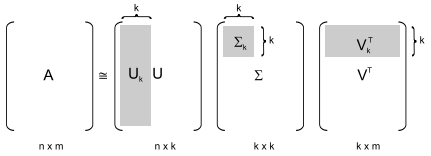


Fig. 1. rank-k SVD.

Rank-k SVD is the best rank-k approximation of the original matrix A. This means that any other decomposition will increase the approximation error, calculated as a sum of squares (*Frobenius norm*) of error matrix $B = A - A_k$. However, it does not implicate that we could not obtain better precision and recall values with a different approximation.

The value of k was experimentally determined as several tens or hundreds (e.g. 50–250), however its exact value cannot be simply determined.

Once computed, SVD reflects only the decomposition of original matrix of images. If several hundreds of images have to be added to existing decomposition (*folding-in*), the decomposition may become inaccurate. Because the recalculation of SVD is expensive, so it is impossible to recalculate SVD every time images are inserted. The *SVD-Updating* [3] is a partial solution, but since the error slightly increases with inserted images. If the updates happen frequently, the recalculation of SVD may be needed soon or later.

2.2 FastMap

FastMap [4] is a pivot-based technique of dimension reduction, suitable for Euclidean spaces.

In first step, it chooses two points, which should be most distant for calculated reduced dimension. Because it would be expensive to calculate distances between all points, it uses following heuristics:

1. A random point c_0 is chosen.
2. The point b_i having maximal distance $\delta(c_i, b_i)$ from c_i is chosen, and based on it we select the point a_i with maximal distance $\delta(b_i, a_i)$
3. We iteratively repeat step 2 with $c_{i+1} = a_i$ (authors suggest 5 iterations).
4. Points $a = a_i$ and $b = b_i$ in the last iteration are pivots for the next reduction step.

In second step (having the two pivots a, b), we use the cosine law to calculate position of each point on line joining a and b. The coordinate x_i of point p_i is calculated as

$$x_i = \frac{\delta^2(a_i, p_i) + \delta^2(a_i, b_i) - \delta^2(b_i, p_i)}{2\delta(a_i, b_i)}$$

and the distance function for next reduction step is modified to

$$\delta'^2(p'_i, p'_j) = \delta^2(p_i, p_j) - (x_i - x_j)^2$$

The pivots in original and reduced space are recorded and when we need to process a query, it is projected using the second step of projection algorithm only. Once projected, we can again use the original distance function in reduced space.

2.3 SDD method

The SDD method is similar to the SVD. The decomposition of is in a form of

$$A \approx A_k = XDY^T$$

The matrices X and Y contain only values from the set $S = \{-1, 0, +1\}$, thus requiring much less memory for storage. D is a diagonal matrix with positive scalars.

The optimal choice of the triplets (x_i, d_i, y_i) for k can be determined using greedy algorithm, based on the residual $R_k = A - A_{k-1}$, where A_0 is a zero matrix.

2.4 NMF method

The NMF method calculates an approximation of the matrix A as a product of two matrices, W and H. The matrices are usually pre-filled with random values (or H is initialized to zero and W is randomly generated). During the calculation the values in W and H stay positive. The approximation of matrix A, matrix A_k, can be calculated as $A_k = WH$.

The original NMF method tries to minimize the Frobenius norm of the difference between A and A'_k using

$$\min_{W,H} ||V - WH||_F^2$$

as the criterion in the minimalization problem.

Recently, a new method was proposed in [8,10], where the constrained least squares problem is solved

$$\min_{H_j}\{||V_j - WH_j||_2^2 - \lambda||H_j||_2^2\}$$

as the criterion in the minimalization problem. This approach is yields better results for sparse matrices, containing text documents.

Unlike in SVD, the base vectors are not ordered from the most general one and we have to calculate the decomposition for each value of k separately.

3 Qualitative measures of retrieval methods

Since we need an universal evaluation of any retrieval method, we use some measures to determine quality of such method. In case of Information Retrieval we usually use two such measures - *precision* and *recall*. Both are calculated from the number of objects relevant to the query Rel – determined by some other method, e.g. by manual annotation of given collection and the number of retrieved objects Ret. Based on these numbers we define precision (P) as a fraction of retrieved relevant objects in all retrieved objects and recall (R) as a fraction of retrieved relevant objects in all relevant objects. Formally:

$$P = \frac{|Rel \cap Ret|}{|Ret|} \text{ and } R = \frac{|Rel \cap Ret|}{|Rel|}$$

So we can say that recall and precision denote, respectively, completeness of retrieval and purity of retrieval. Unfortunately, it was observed that with the increase of recall, the precision usually decreases [7]. This means that when it is necessary to retrieve more relevant objects, a higher percentage of irrelevant objects will be probably obtained, too.

For the overall comparison of precision and recall across different methods on a given collection, we usually use the technique of rank lists [2], where we first sort the distances from smallest to greatest and then go down through the list and calculate maximal precision for recall closest to each of the 11 standard recall levels (0.0, 0.1, 0.2, ..., 0.9, 1.0). If we are unable to calculate precision on i-th recall level, we take the maximal precision for the recalls between $i-1$-th and $i+1$-th level.

4 Experimental results

For testing of the different methods, we used the Olivetti Research Laboratory collection of Olivetti Laboratory employees' faces from years 1992-1994 (the images were converted to 256 shades of gray and their size reduced to 112x92). Out of 400 images (10 images per face), we excluded 1 image per face from indexing for future querying, and generated 360 image vectors of the dimension 10304, which we stored in the image matrix A.

Fig. 2. Several faces from ORL collection.

An example of several images from ORL collection is shown in figure 2. Several query images, which were not used in the indexing step are shown in figure 12.

4.1 Generated "eigenfaces" and reconstructed images

Many of tested methods were able to generate a set of base images, which could be considered to be "eigenfaces" as is the case of PCA, SVD and several other methods. We are going to provide examples of both factors (base vectors) – "eigenfaces" and reconstructed images which can be obtained from regenerated A_k. We calculated results for all methods in several dimensions, for the demonstration images we will use $k = 100$. In some cases where it is advisable, we will

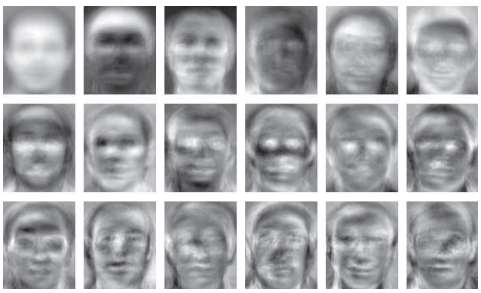


Fig. 3. First 18 eigenfaces (out of possible 360) for SVD method.

Fig. 5. First 18 base vectors (out of 100) for original NMF method.

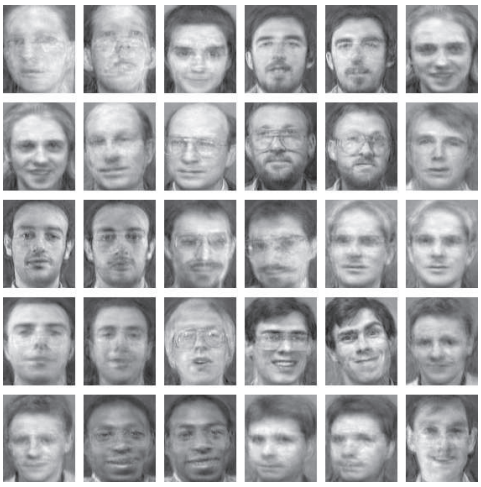


Fig. 4. Reconstructed images for SVD method.

Fig. 6. Reconstructed images for original NMF method.

provide results for $k = 40$, too. We do not provide these images for FastMap, where it is not possible (we could have provided the images used as pivots in each step of FastMap process).

With SVD, we obtain classic eigenfaces, the most general being among the first. The first few are shown in figure 3. The eigenfaces with higher index bring more details to reconstructed images.

The reconstructed images for rank-100 SVD method are somewhat blurred, but generally still recognizable, which can bee observed in figure 4.

The NMF methods yield different results. The original NMF method, based on the adjustment of random matrices W and H provides hardly-recognizable eigenfaces even for $k = 100$ and 1000 iterations (we used 100 iterations for other experiments). Moreover, these base images still contain significant salt and pepper noise and have a bad contrast. The factors are shown in figures 5. We must also note, that the NMF decomposition will yield slightly different results each time it is run, because the matrix(es) are pre-filled with random values.

The reconstructed images (shown in figure 6) suffer from slight noise, too, but it is not as noticeable as in the case of the base vectors.

The GD-CLS modification of NMF method (proposed in [8]) tries to improve the decomposition by calculating the constrained least squares problem. This leads to a better overall quality, however, some of the images were preserved better than others. The decomposition really depends on the pre-filled random matrix H. We show here two different results, one for $k = 100$ in figures 7 and 8, and one for images for $k = 36$ in figure 9. We can observe that the factors are not ordered based on their generality and while some are blurry, other are almost exact copies of original face images.

One can see that some images after GD-CLS for $k = 100$ are reconstructed almost perfectly, while other are replaced with an extremely bad approximation.

The SDD method differs slightly from all of the abovementioned ones, since each factor contains only values $\{-1, 0, 1\}$. Gray in the factors shown in figure 10 represents 0; -1 and 1 are represented with black and white respectively.

Fig. 7. First 18 base vectors (out of 100) for GD-CLS NMF method.

Fig. 8. Reconstructed images for GD-CLS NMF method ($k = 100$), examples of bad reconstruction were included.

Fig. 9. Reconstructed images for GD-CLS NMF method ($k = 36$).

Fig. 10. First 18 base vectors (out of 100) for SDD method.

Fig. 11. Reconstructed images for SDD method.

Fig. 12. Queries, excluded from indexing step.

The images in figure 11 are reconstructed least exactly from all methods (although consistently), but this is to be expected due to the three-valued encoding of base vectors.

4.2 Query evaluation

Firstly, we used rank lists and measured interpolated average precision of Euclidean distance (L_2 metrics) of the 40 queries at the 11 standard recall levels. We provide only results for $k = 40$, corresponding to the number of clusters (distinct faces) in figure 13.

We also calculated the mean average precision for all relevant documents in rank lists. The relative results against original matrix A (100%) are shown in Table 1.

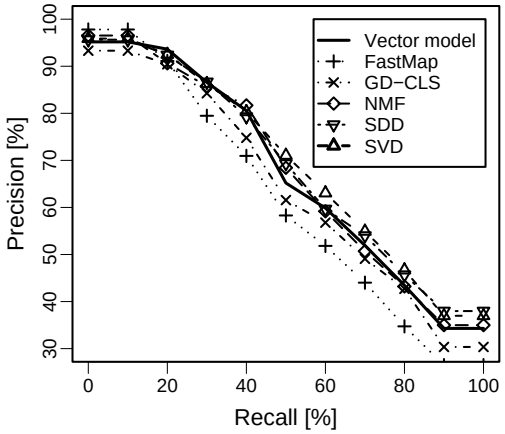


Fig. 13. Precision at the 11 standard recall levels for $k = 40$.

For original NMF, we used 100 iterations. For GD-CLS method less iterations were required, we decided to use the k iterations for each dimension with $\lambda = 0.001$.

For NMF (both methods) and SDD, we were unable to find a suitable distance function in the reduced space and the results were poor. We used the results on reconstructed matrix A_k instead, to provide a reasonable comparison.

	Reduction method				
k	**LSI**	*FastMap*	NMF	GD-CLS	SDD
18	100.4%	84.9%	100.3%	95.3%	93.9%
36	102.8%	90.4%	101.6%	95.0%	100.3%
40	102.8%	90.8%	100.1%	95.5%	101.6%
72	101.3%	93.0%	103.2%	95.5%	103.4%
100	100.5%	94.2%	103.7%	95.3%	102.9%
180	99.2%	96.1%	102.4%	93.3%	102.1%

Table 1. Mean average precision of different reduction methods.

5 Conclusion

In this paper, we have compared several dimension reduction methods on real-live image data (using cosine measure as similarity function). Whilst the SVD is known to provide quality eigenfaces, it is computationally expensive and in case we only need to beat the "curse of dimensionality" by reducing the dimension, FastMap or random projection may suffice. As expected, from methods which allowed direct querying in reduced space, SVD was the slower, but most exact method, followed by FastMap, which is faster but less accurate.

The NMF and SDD methods may be also used, but right now we would have to reconstruct the matrix A_k, since the L_2 distances (not the deviations) are not preserved enough. Their results were comparable with SVD, sometimes appearing even slightly better.

There are some other newly-proposed methods, which may be interesting for future testing, e.g. the SparseMap [5]. Additionally, faster pivot selection technique for FastMap may be considered. Finally, testing of used dimension reduction methods with deviation metrics on metric structures should answer the question of projected data indexability (which is poor for SVD-reduced data).

References

1. Face recognition homepage. May, 2006, `http://www.face-rec.org`.
2. R. Baeza-Yates and B. Ribeiro-Neto, Modern Information Retrieval. Addison Wesley, New York, 1999
3. M. Berry, S. Dumais, and T. Letsche, Computation Methods for Intelligent Information Access. In Proceedings of the 1995 ACM/IEEE Supercomputing Conference, 1995
4. C. Faloutsos and K. Lin, FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets. ACM SIGMOD Record, 24(2), 1995, 163–174
5. G.R. Hjaltason and H. Samet, Properties of Embedding Methods for Similarity Searching in Metric Spaces. IEEE Transactions on Pattern Analysis and Machine Intelligence, 25(5), 2003, 530–549
6. P. Praks, L. Machala, and V. Snášel, Iris Recognition Using the SVD-Free Latent Semantic Indexing. In L. Khan and V.A. Petrushin (Eds.); Proceeding of the Fifth International Workshop on Multimedia Data Mining (MDM/KDD'04), 2006, 67–71
7. G. Salton and G. McGill, Introduction to Modern Information Retrieval. McGraw-ill, 1983
8. F. Shahnaz, M. Berry, P. Pauca, and R. Plemmons, Document Clustering Using Nonnegative Matrix Factorization. Journal on Information Processing and Management, 42, 2006, 373–386
9. W. Skarbek, K. Kucharski, and M. Bober, Cascade of Dual LDA Operators for Face Recognition. In Geometric Properties for Incomplete Data, Springer, 2006, 199–219
10. M.W. Spratling, Learning Image Components for Object Recognition. Journal of Machine Learning Research, 7, 2006, 793–815
11. M. Turk and A. Pentland, Eigenfaces for Recognition. Journal of Cognitive Neuroscience, 3(1), 1991, 71–86

Evolutionary learning of multi-layer perceptron neural networks*

Roman Neruda and Stanislav Slušný

Institute of Computer Science, Academy of Sciences of the Czech Republic
18207 Prague 8, Czech Republic
slusny@cs.cas.cz

Abstract. *Neural networks have been used in many classification tasks. However, their success depends on determining their training algorithm and architecture, which is often a trial-and-error process. Evolutionary algorithms have been used to address these problems successfully in recent years. This paper reviews different combinations between the most widely used type of neural networks – a multi-layer perceptron – and evolutionary algorithms. Several methods to determine the architecture and train the weights of the network are tested using a real-world classification problems from Proben1 benchmark suite. It is shown, that combining evolutionary algorithms with neural networks can lead to better results than relying on neural networks alone. Comparison to gradient algorithms is discussed.*

1 Introduction

Artificial neural networks (ANNs) are a computational paradigm modeled on the human brain that have been applied to a variety of classification and learning tasks for a few reasons. Despite their simple structure, they provide very general computational capabilities and they can adapt themselves to different tasks, i. e. they are able to learn.

Evolutionary algorithms can be viewed as an alternative to classical optimization techniques, based on a biological metaphor: over many generations, natural populations evolve according to the principles of natural selection and "survival of the fittest", first clearly stated by Charles Darwin in The Origin of Species [7]. The basic principles of Evolutionary algorithms were first laid down rigorously by Holland [8].

In this paper, we present a comparison of different approaches to ANN learning problem. We will focus on combination of gradient and evolutionary techniques, and we will try to find optimal weights and topology of neural networks. Several experiments will be carried out on data taken from Proben1 [2] benchmark collection.

The organization of this paper is as follows. First we briefly describe perceptron networks and the core of genetic we have used. Then, the two main approaches — evolutionary learning of the weights, and evolution of network architecture — are presented. Finally, the experiments are reported and discussed.

* This research has been supported by the Information society project no. 1ET100300419 (of the Thematic Program II of the National Research Program of the Czech Republic) "Intelligent Models, Algorithms, Methods and Tools for the Semantic Web Realisation".

2 Multilayer perceptron network

An multilayer feedforward ANN is an interconnected network of simple computing units called *neurons*, which are ordered in layers, starting with an *input layer* and ending with an *output layer*. [1]. Between these two layers there can be a number of *hidden layers*. Connections in this kind of networks only go forward from one layer to the next. Let us denote n_I, n_O, n_H number of input, output and all hidden neurons, respectively. Each neuron has n real inputs, and each input x_i has assigned a real weight w_i.

The output $y(x)$ of a neuron is defined in equation 1:

$$y(x) = g\left(\sum_{i=1}^{n} w_i x_i\right), \tag{1}$$

where x is the neuron with n input dendrites $(x_0 \dots x_n)$, one output axon $y(x)$, $(w_0 \dots w_n)$ are weights and $g : \Re \to \Re$ is the *activation function*. One of the most common activation function is a logistic sigmoid function 2

$$\sigma(\xi) = 1/(1 + e^{-\xi t}), \tag{2}$$

where t determines its steepness.

The *architecture* of ANN defines the number of layers, number of neurons in each layer and connections between neurons.

We will focus on a learning situation known as a *supervised learning*, in which a set of input/desired-output patterns is available. The training process can be seen as an optimization problem, where we wish to minimize the *mean square error* (E_{MSE}) of the entire set of training data. E_{MSE} is defined as the squared error of the ANN for a set of patterns:

$$E_{\text{MSE}} = \sum_{k=1}^{p} E_k(w) \tag{3}$$

$$E_k(w) = \frac{1}{2} \sum_{j \in Y} (y_j(w, x_k) - d_{kj})^2, \tag{4}$$

where Y is a set of output neurons, p is the number of patterns in the set, y_j is an output of neuron j given weight vector w and k-th input pattern x_k, and d_{kj} is desired output of output neuron j for input pattern k.

The *classification problem* is that of assigning an input vector to one of M classes — when an input vector is presented to the network, the network response is the class associated with the output neuron with the largest output value (winner-takes-all).

For classification problems, E_{CLAS} reports — in a high-level manner — the quality of the trained ANN and is defined as a percentage of incorrectly classified patterns.

3 Evolutionary algorithm

Evolutionary algorithm has been investigated by John Holland [8], with a marked increase of interest within the last few years. Evolutionary search refers to a class of stochastic optimization techniques — loosely based on processes believed to operate in biological evolution — in which a population of candidate solutions (chromosomes) evolves under selection and random "genetic" diversification operators. The problem is to find minimum or maximum of *fitness function*. Every member of population is called *individual* and represents a potential solution to a problem.

The algorithm 1 reveals skeleton of genetic algorithm used in our experiments. To fully describe genetic algorithm, it is needed to define how each solution will be represented, how initial population will be created and what genetic operators will be used in the Reproduction step.

We have used the *roulette wheel selection* in all experiments. The selection operator performs the equivalent role to natural selection — it chooses individuals for next population proportionally to their fitness values. As we wanted to minimize the error function in all our experiments, the probability p_i of selection i-th individual is defined by the equation 5

$$p_i = \frac{(1+\varepsilon) * C - \mathcal{F}(I_i)}{N * (1+\varepsilon) * C - \sum_{j=1}^{N} \mathcal{F}(I_j)}, \qquad (5)$$

where N is the number of individuals and C is the maximal fitness value in the population, and ε is a small positive constant, which ensures that probability of selection the worst individual will be non-zero.

4 Evolution of weights

Weight training in ANN is usually formulated as a minimization of error function, and is carried out by some gradient descent algorithm such as Back-Propagation, or one of its many variants. Due to its use of gradient descent, these algorithms often get trapped in the local minimum of the error function, and are incapable of finding global minimum. As the evolutionary algorithm need not to use the gradient information and works with population of potential solutions, it has a better ability to avoid local minimum trap. What more, evolutionary algorithm can be applied in situations, when gradient information is not available at all, it can handle global search problem in a vast, complex and non-differentiable surface. The evolutionary approach also makes it easier to generate ANNs with some special characteristics. For example, ANNs complexity can be decreased and its generalization ability increased by including a special term in the fitness function penalizing large networks.

Input: number of individuals in population N, number of elits E, maximum number of populations $G_{\max}$.
Output: the best found solution of a problem.

1. Start: Create initial population of N individuals $P(0) = \{I_1, \ldots, I_N\}$, $i = 0$.
2. Evaluation of individuals: To compute fitness function for every individual I, build ANN corresponding to I and compute E_{MSE} for a training set. Let $\mathcal{F}(I) = E_{\text{MSE}}$.
3. Stop-condition: If $i = G_{\max}$, finish and return solution represented by individual with minimal value of fitness function.
   (a) Creation of new population $P(i+1)$ from population $P(i)$: Create empty population $P(i+1)$.
   (b) Selection: Choose E best individuals from population $P(i)$ and move them to population $P(i+1)$. With chosen operator of selection choose $N - E$ individuals and insert them to population $P'(i)$.
   (c) Reproduction: Apply chosen operators on population $P'(i)$, the result is population $P(i+1)$.
      i. Application of binary operators: If population $P'(i)$ contains odd number of individuals, insert chromosome of the first individual from population $P'(i)$ into population $P''(i)$. From population $P'(i)$ choose pairs of chromosomes C and D and apply on them reproduction operators, new chromosomes C' and D' insert to population $P''(i)$.
      ii. Application of unary operators: On every chromosome from population $P''(i)$ apply chosen unary operator, new chromosome insert to population $P(i+1)$.
4. New generation: $i = i + 1$, Continue by step 2.

Algorithm 1: Skeleton of the genetic algorithm.

4.1 Representation

The standard genetic algorithm uses binary strings to encode alternative solutions. In a such representation scheme, each connection weight is represented by

a number of bits of certain length. The advantages of this representation are mainly simplicity and generality. It is possible to use well known crossover (such as one-point crossover) or mutation operators for binary strings. Although there are several encoding methods, that encode real numbers with different range and precision, trade off between precision and range often has to be made. Real-world experiments demand big precision, what causes too long chromosome for which the evolution process becomes non-efficient. Therefore, different encoding method is used. The chromosome is interpreted as a matrix of dimensions $n_H \times n$, where $n = n_O + n_H$. In the i-th row and the j-th column there is either a special symbol $\otimes$, if neurons i and j are not connected, or the weight of connection from i to j. The following connection matrix from figure 2 encodes the ANN from the figure 1. The chromosome is then created by concatenating all the rows from the matrix. In case of feedforward ANNs, the entry on i-th row and j-th column can be non-zero only for $i < j$. This reduces the length of the chromosome and the evolutionary process becomes more effective.

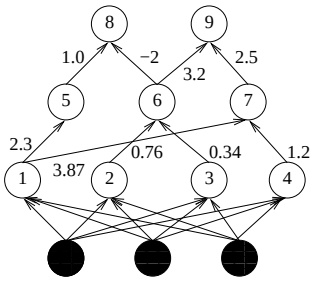


Fig. 1. Example of ANN with seven hidden units in two hidden layers, three input and two output neurons.

$$\begin{bmatrix} \otimes & \otimes & \otimes & \otimes & 2.3 & \otimes & 3.87 & \otimes & \otimes \\ \otimes & \otimes & \otimes & \otimes & \otimes & 0.76 & \otimes & \otimes & \otimes \\ \otimes & \otimes & \otimes & \otimes & \otimes & 0.34 & \otimes & \otimes & \otimes \\ \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & 1.2 & \otimes & \otimes \\ \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & 1.0 & \otimes \\ \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & -2 & 3.2 \\ \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & 2.5 \end{bmatrix}$$

Fig. 2. Example of encoded ANN from the previous figure.

4.2 Initial population and operators

Initial population consists of fully connected neural networks with randomly initialized values. To each connection a weight from $[-1.0, 1.0]$ interval is assigned.

For training the weights of ANN, we experimented with five different operators[5]: three varieties of mutation and two varieties of crossover. The *Unbiased-Weights-Mutation* operator replaces the weights of randomly selected connections in the parent network with values chosen randomly from the same probability distribution used for initialization. *Biased-Weights-Mutation* does the same as the previous one except that, instead of replacing the old values with the randomly selected values, it adds the randomly selected values to the old values. The reasoning is that any network selected as a parent (especially as the run progresses) is significantly better than a randomly selected network, and so its individual weights are usually better than randomly selected ones. *Nodes-Mutation* randomly selects nodes from the set of all hidden and output nodes and performs a biased mutation on all the incoming weights for the selected nodes. The basic unit is not a single weight, as in former operators, but a set of incoming weights. As mentioned in [5], we know a priori that the set of incoming weights for a particular node forms a low-order schemata, or building block. A mutation which disrupts as few of these schemata as possible by concentrating its changes in a small number of them should outperform a mutation which disrupts many of these schemata (by schemata theorem[8]).

Weights-Crossover performs uniform crossover with the basic units exchanged being the set of all incoming weights of a particular node. *Nodes-Crossover* performs uniform crossover with the basic units exchanged being the set of all incoming weights of a particular node.

It is satisfactory to introduce just one parameter for all of these operators — the propability, by which will be the basic unit changed by particular operator.

5 Evolution of architecture

In the previous section, we have assumed that the evolved ANN has a fixed architecture. Selection of a good architecture is state of art — although it has significant impact on network's information processing capabilities, there are not known satisfiable rules, on how to choose a good one for a particular task. ANN with only a few connections may not be able to satisfiable learn the task because of its limited capacity, while ANN with large number of connections and neurons may over fit and fail to have a good generalization ability.

Similarly to previous section, the problem of determining optimal architecture for a particular task can be formulated as an optimization problem. As stated

in [6], evolutionary algorithms are a good method for searching in such a complex surfaces. In the case of *simultaneous evolution of weights and architecture* both weights and characteristics of architecture are encoded in the chromosome. As shown on figure 3, this method contains the previously solved problem of determining optimal set of weights as a subtask. On the other side, the problem of *separated evolution of weights and architectures* is the noisy fitness evaluation, caused by random initialization of weights. Only architecture is encoded in chromosome, weights have to be learned later. The fact that an individual gained better fitness value does not mean that this individual is really better — in is necessary to repeat the evaluation and average the obtained values. This way, the computation time increases dramatically.

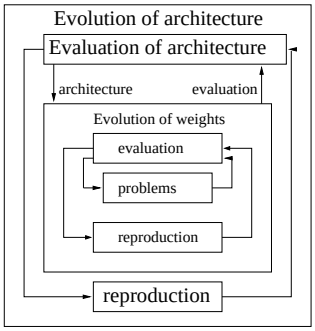


Fig. 3. Schema of evolutionary algorithm searching optimal architecture of ANN.

Number of hidden layers.
Number of neurons in the first hidden layer.
...
Number of neurons in the last hidden layer.
Connection matrix.

Table 1. Chromosome for evolving architectures of ANN.

5.1 Representation

We used the *indirect encoding scheme* in our experiments [6]. Chromosome consists of several sections, which hold information about some important characteristic of ANN's architecture. In the first section is stored the number of hidden layers, follows section with information about the number of neurons in each hidden layer and in the case of simultaneous evolution of weights and architecture, the section with connection matrix is appended. The operators used in evolutionary algorithm must be aware of this representation — operators from previous section are allowed to modify only the last section of the chromosome. To modify the remaining parts, new operators are introduced.

5.2 Operators

There are two operators that work on the level of units. The *Remove-Neuron* operator removes randomly selected neurons. Creation of new neurons is done by the *Duplicate-Neuron* operator, which selects random neuron and duplicates it. The new duplicated neuron will be connected to the same neurons as the original one and the weights will have the same values.

Similarly, there are two operators working on connections. The *Remove-Connection* operator removes the connection between neurons with given propability . It is applied to all pairs of interconnected neurons. On the other side, the *Add-Connection* operator adds connection between two neurons with a given propability. It is applied to all pairs of neurons, which can be connected (without violating the conditions on architecture stated in section 2), but the connection does not exist yet.

6 Experiments

A set of experiments has been carried out on several data sets from the Proben1 [2–4] benchmark. This way we are able not only to provide relative comparison, but also to explore the efficiency of the algorithms with respect to the best results obtained by other methods and authors. In the following we briefly describe several experiments and try to generalize the results.

We have chosen four classification problems: The goal of ANN is to classify tumor as either benign or malignant in the Cancer problem, diagnose diabetes of Pima indians in Diabetes problem, predict the heart disease in Heart problem and recognize one of 19 different diseases of soybean in Soybean problem. Characteristics of each data set are shown in table 2.

	Classes	Examples	b	c	n	Tot.	b	c	m	Tot.
Cancer	0	9	0	9	0	9	0	9	2	699
Diabetes	0	8	0	8	0	8	0	8	2	768
Heart	1	6	6	13	18	6	11	35	2	920
Soybean	16	6	13	35	46	9	27	82	19	683

Table 2. Problems and the number of binary, continuous, and nominal attributes in the original dataset, number of binary and continuous network inputs, number of network inputs used to represent missing values, number of classes, number of examples.

The results of experiments are reported in terms of the values of the two errors, E_{MSE} and E_{CLAS} measured both on the training set and previously unseen test set. In the following tables we use symbols M_t (or M_s) for E_{MSE} over training (or testing) set, and C_t (and C_s) for E_{CLAS} over training (testing, respectively) set.

6.1 Efficiency of genetic operators for weight evolution

In this experiment we have compared the previously described operators of crossover and mutation (cf. Tab. 3). The worst results were achieved using the unbiased mutation operator which is the 'plain-vanilla' operator not based on any heuristics. More surprising, however, was the poor performance of operators based on schemata theorem — namely the unbiased mutation — as opposed to paper [5]. For the case of crossover, the results were mixed, both weight and neuron crossover performed well on some tasks.

	Diabetes					Cancer				
	Cn	Cw	Mn	BMw	NMw	Cn	Cw	Mn	BMw	NMw
M_t	0.3832	0.3951	0.3718	0.3373	0.4360	0.0710	0.0658	0.0912	0.0598	0.2825
C_t	0.2786	0.3073	0.2526	0.2188	0.3464	0.0343	0.0343	0.0436	0.0324	0.0533
M_s	0.3963	0.4066	0.3756	0.3670	0.4426	0.0502	0.0470	0.0677	0.0242	0.2902
C_s	0.2786	0.3203	0.2656	0.2448	0.3516	0.0115	0.0057	0.0282	0.0115	0.0402
	Soybean					Heart				
	Cn	Cw	Mn	BMw	NMw	Cn	Cw	Mn	BMw	NMw
M_t	0.9220	0.9144	0.9291	0.9332	1.7271	0.2831	0.2747	0.3245	0.2917	0.3690
C_t	0.8363	0.6842	0.7895	0.7836	0.8246	0.1652	0.1812	0.2056	0.1754	0.2087
M_s	0.9156	0.9259	0.9238	0.9293	1.7475	0.2949	0.3141	0.3479	0.3148	0.3809
C_s	0.8446	0.7185	0.7537	0.7683	0.8152	0.2043	0.1957	0.2304	0.1826	0.2261

Table 3. Error results for different version of GA for weight evolution: neuron crossover (Cn), weight crossover (Cw), neuron mutation (Mn), biased weight mutation (BMw), and unbiased weight mutation (NMw).

6.2 Comparison of genetic and gradient learning

There are several reports on efficiency of the genetic learning of neural networks, and they differ in a relevant way [6]. We have chosen to compare the classical back propagation and the *rprop*[9] gradient learning algorithm with GA using the weight mutation and weight crossover that achieved best results in the previous experiment (cf. Tab. 4). Typically we run the gradient algorithm for 5000-25000 epochs, while the GA runs for 1000-2000 generations (with population of 300 individuals). The results differ from task to task. Generally, one can say that gradient algorithms achieve better results faster. GA are about and order of magnitude slower, and on more difficult tasks they are not able to achieve the precision obtained by gradient learning within a reasonable time at all. Surprisingly, the solution obtained by GA has often better generalization capabilities. It is fair to mention that GAs are easy to parallelize, and thus using a parallel architecture provides relevant speedup, which is not true for the gradient learning.

6.3 Hybrid learning algorithms

In this experiment we have employed the mutation operator which realized 50 steps of the rprop gradient learning. Thus, we can think of this as a combination of global impact GA search with local tuning by the rprop mutation. Such a hybrid algorithm is able to beat the back propagation in terms of quality of the solution on most of the tasks. Surprisingly, the crossover operator appeared to be quite counter-productive in this experiment, using just the weight mutation and rprop mutation provided best results (cf. Tab. 5).

	Diabetes			Cancer		
	BP	RP	GA	BP	RP	GA
M_t	0.2675	0.2473	0.2882	0.0463	0.0390	0.0484
C_t	0.2645	0.2616	0.2083	0.0280	0.0438	0.0286
M_s	0.3217	0.3154	0.3268	0.0242	0.0180	0.0159
C_s	0.2645	0.2615	0.2266	0.0280	0.0438	0.0057
	Soybean			Heart		
	BP	RP	GA	BP	RP	GA
M_t	0.0150	0.2112	0.7416	0.1840	0.0904	0.2372
C_t	0.2536	0.3080	0.5585	0.2033	0.2079	0.1536
M_s	0.1196	0.2907	0.7474	0.2574	0.2559	0.2904
C_s	0.2536	0.3078	0.5513	0.2033	0.2077	0.1913

Table 4. Error comparison for back propagation (BP), rprop (RP) and genetic algorithm (GA) used for weights learning.

	Diabetes		Cancer		Soybean		Heart	
	NX	CX	NX	CX	NX	CX	NX	CX
M_t	0.2573	0.2608	0.0274	0.0371	0.0033	0.0437	0.1556	0.1626
C_t	0.1745	0.1875	0.0152	0.0171	0.0029	0.0117	0.0899	0.0928
M_s	0.3211	0.3215	0.0202	0.0215	0.2424	0.2714	0.2642	0.2644
C_s	0.2214	0.2266	0.0057	0.0115	0.1261	0.1496	0.1783	0.1826

Table 5. Error results for 2 flavors of hybrid GA: a) GA using 2 types of mutation only (the NX columns), b) GA with 2 mutations and a crossover (the CX columns).

6.4 Searching for suitable connections

This experiment tested the separated evolution of architecture and weights. Thus, the algorithm consists of two steps - in the first one the direct architecture encoding described in Tab. 1 is used and evolved. To determine a fitness of such an individual, several (30 in our case) randomly initiated runs of rprop algorithm are carried. In the second step, the evolved architecture is trained by full-fledged run of a back propagation algorithm (10 times for different random weights initializations). Results of this experiment were quite satisfiable (cf. Tab. 6): GA was able to find better architectures, containing less connections, and achieving better training error. As a side effect, these specialized architectures usually achieved worse generalization error, in comparison to fully connected architectures, which is understandable (compare Tab. 4 and Tab. 6).

6.5 Searching for a suitable architecture

The goal of the last experiment was to combine the search for architecture with the evolution of weights

	Diabetes	Cancer	Soybean	Heart
M_t	0.4541	0.0436	0.0069	0.0566
C_t	0.3516	0.0157	0.1615	0.1764
M_s	0.4568	0.0238	0.1369	0.2501
C_s	0.3516	0.0157	0.1615	0.1764

Table 6. Search for connections: Error results for architectures recommended by GA.

in one GA. We have started with individuals with random number of hidden layers (between 1–4), which remained constant during the algorithm. However, the number of neurons and connections were varying, as well as the values of weights. The following genetic operators we used:

operator	p_m
Duplicate-Neuron	0.05
Remove-Neuron	0.05
Add-Connection	0.05
Remove-Connection	0.05
Biased-Weights-Mutation	0.1

Table 7. Operators probability.

The overall results were very satisfiable (cf. Tab. 8, compare with Tab. 4). The GA was able to evolve suitable architectures for a given task. It is interesting, that most of the solutions had just one hidden layer, some of them had two. In some papers authors use penalization for more complex solutions, which was not necessary here, because the evolution tends to exclude more complex networks based on their fitness anyways. The architectures recommended by the GA provided even better results than the ones reported as best (found by human) in [2].

	Diabetes	Cancer	Soybean	Heart
M_t	0.2679	0.0382	0.0176	0.1025
C_t	0.1927	0.0190	0.0088	0.0551
M_s	0.3108	0.0184	0.0989	0.2342
C_s	0.2266	0.0057	0.0616	0.1435

Table 8. Search for an architecture and weights: Error results for genetically evolved network architectures.

7 Conclusions

As shown, evolution can be introduced into ANN learning problem at different levels. Suggested algorithms were tested on real-world problems from Proben1 benchmark suite. The evolutionary algorithm is a complex and robust method, which can be used to search both optimal weights and architecture of ANN. Although the evolutionary process can be easily parallelized, computation is always very time consuming. The evolutionary algorithm alone did not manage to gain better results than back propagation, but the combination of gradient descent and evolutionary algorithm performed very well. The best results were obtained by simultaneous evolution of weights and architecture. Not only the resulting ANNs gained best results on the training set, they showed a good generalization ability. The complexity of found architecture for a particular task mirrored it's real difficulty.

The combination of evolutionary techniques and ANNs can lead to better intelligent systems, than relying on ANNs alone. With the increasing power of parallel computers, the evolution of large ANNs becomes feasible.

References

1. J. Šíma, R. Neruda, Teoretické otázky neuronových sítí, Matfyzpress, Praha, 1996
2. Lutz Prechelt, PROBEN1 - A Set of Benchmarks and Benchmarking Rules for Neural Network Training Algorithms. Technical Report 21/94, Universitt Karlsruhe, Germany, September 1994
3. O.L. Mangasarian, W.H. Wolberg, Cancer Diagnosis via Linear Programming. SIAM News, Vol. 23, No. 5, September 1990, 1–18
4. R.S. Michalski, R.L. Chilausky, Learning by Being Told and Learning from Examples: An Experimental Comparison of the Two Methods of Knowledge Acquisition in the Context of Developing an Expert System for Soybean Disease Diagnosis. International Journal of Policy Analysis and Information Systems, Vol. 4, No. 2, 1980
5. D. Montana, L. Davis, Training Feedforward Neural Networks Using Genetic Algorithms. In Proc. 11th Int. Joint Conf. Artificial Intelligence. San Mateo, CA: Morgan Kaufmann, 1989, 762-767
6. X. Yao, Y. Liu, Evolving Neural Network Ensembles by Minimization of Mutual Information. International Journal of Hybrid Intelligent Systems, 1(1), January 2004, 12-21
7. Darwin Charles, The Origin of Species. New American Library, Mentor paperback, 1859
8. Holland J., Adaptation in Natural and Artificial Systems. The University of Michigan Press, Ann Arbour, 1975
9. M. Riedmiller, H. Braun, A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm, Proceedings International Conference on Neural Networks, San Francisco, CA, 1993, 586-591

Analýza selektorov pre selektívne šifrovanie*

Richard Ostertág and Peter Košinár

Katedra informatiky, Fakulta matematiky, fyziky a informatiky Univerzity Komenského,
Mlynská dolina, 842 48 Bratislava
{ostertag,kosinar}@dcs.fmph.uniba.sk
http://www.dcs.fmph.uniba.sk

Abstrakt *Niektoré aplikácie požadujú vysokú rýchlosť šifrovania aj za cenu zníženia jeho bezpečnosti. Zvýšenie rýchlosti sa dá pri zachovaní pôvodného (kvalitného ale výpočtovo náročného) šifrovacieho algoritmu dosiahnuť napríklad zašifrovaním len časti dát. Článok analyzuje bezpečnosť algoritmov selektívneho šifrovania postavených na rôznych metódach výberu tejto časti.*

1 Motivácia, predpoklady, ciele

Čoraz častejšie sa stretávame s potrebou spracovávať dáta veľkých objemov. Typickým príkladom je najmä oblasť mutimédií. Rýchle šifrovanie často vyžaduje výkonný hardvér. Ak robíme niekoľko šifrovaní súčasne, neraz je už potrebný špecializovaný hardvér. Bez hardvérovej akcelerácie by nebolo možné dosahovať požadované šifrovacie rýchlosti napríklad pri „on-demand“ distribúcii filmov.

Niekedy však nie je možné ísť cestou zvyšovania výkonu hardvéru. Potom je možné použiť napríklad rýchlejší šifrovací algoritmus – ten ale môže byť menej bezpečný. Inou používanou alternatívou je selektívne šifrovanie. V takomto prípade sa nešifrujú všetky dáta, ale iba ich nejaká vybraná časť, celkovo povedzme p percent[1]. Tým sa aj pri použití pôvodneho algoritmu zníži záťaž systému spôsobená šifrovaním. Je však zrejmé, že so znižujúcim sa p klesá aj poskytovaná bezpečnosť.

Selektívne šifrovanie sa používa napríklad pri „online“ šifrovaní MPEG videa [1]. V tomto prípade sa používa znalosť vnútornej štruktúry dát a šifrujú sa iba znamienka pohybových vektorov a DC koeficienty MPEGu. Podobný prístup sa používa aj pre obrázky [2]. Prehľad metód selektívneho šifrovania je možné nájsť napríklad v [3]. Bezpečnosť týchto algoritmov však viac menej nie je formálne dokazovaná.

V tomto článku sme sa rozhodli formálne posúdiť bezpečnosť niektorých spôsobov výberu časti dát pre selektívne šifrovanie. Vzhľadom no to, že nám ide o všeobecné závery, nepredpokladáme žiadnu konkrétnu vnútornú štruktúru, či štatistické vlastnosti otvoreného textu. Preto pri analýze vychádzame z nasledovných idealizovaných predpokladov, aj keď tieto predpoklady nie sú totožné s bežným praktickým nasadením selektívneho šifrovania:

1. *Šifrovanie je realizované Vernamovou šifrou.*
   Túto jedinú absolútne bezpečnú šifru sme zvolili, aby sme abstrahovali od prípadných nedostatkov samotného šifrovacieho algoritmu. Takto získané výsledky nám potom môžu v prípade použitia iného šifrovacieho algoritmu poslúžiť ako horný odhad bezpečnosti na ňom postaveného selektívneho šifrovacieho algoritmu.
2. *Útočník pri útoku môže použiť iba získaný šifrový text a znalosť selektívneho šifrovacieho algoritmu. To znamená, že pozná šifrovací algoritmus aj spôsob výberu šifrovanej časti dát.*
   Útok iba so znalosťou šifrového textu je typickým predpokladom pri analýze bezpečnosti šifrovacích algoritmov.
3. *Útočník môže na šifrový text útočiť iba hrubou silou. Pri útoku najprv vytvorí množinu potencionálnych otvorených textov. Tú získa dešifrovaním zachytenej správy pri všetkých možných kľúčoch. Nakoniec z tejto množiny vyberie najvhodnejší otvorený text.*
   Vo všeobecnosti sa nedá očakávať, že útočník môže použiť iný útok, než útok hrubou silou. V konkrétnom prípade sa však môže stať, že vzhľadom na špecifické vlastnosti otvoreného textu[2] je možné viesť aj oveľa efektívnejší útok.
4. *Zložitosť útoku definujeme ako mohutnosť množiny potencionálnych otvorených textov. Zložitosť výberu najvhodnejšieho otvoreného textu pritom nebudeme brať do úvahy.*
   Zložitosť výberu najvhodnejšieho otvoreného textu je veľmi závislá na ich štruktúre a vlastnostiach. Väčšinou je tento problém ľahko mechanicky riešiteľný, alebo je takmer neriešiteľný[3]. Preto

* Táto práca vznikla s prispením grantu Univer. Komenského č. UK/406/2006 a VEGA grantu č. 1/3106/06.

[1] Percento p používame v matematickom zmysle. To znamená, že celok predstavuje $p = 1$ a nie $p = 100$.

[2] Pri selektívnom šifrovaní môže byť veľkým problémom najmä vysoká redundancia otvoreného textu. Tú je však možné znížiť jeho kompresiou.

[3] Napríklad ľahko a mechanicky sa dá posúdiť či daný otvorený text je syntakticky správny program v jazyku C++, naopak posúdiť, či daný otvorený text je korektný obrázok, je ťažké a nemechanizovateľné.

sme redukovali zložitosť útoku iba na mohutnosť množiny potencionálnych otvorených textov.

Selektívne šifrovanie však okrem urýchlenia získaného redukovaním počtu šifrovaných bitov prináša so sebou aj novú réžiu spojenú práve s výberom týchto bitov. Vzhľadom na v praxi predpokladanú oveľa vyššiu zložitosť šifrovania jedného bitu ako jeho výberu, túto dodatočnú réžiu zanedbáme. Potom, ak p je percento šifrovaných bitov, môžeme písať:

$$\frac{\text{zložitosť selektívneho šifrovania}}{\text{zložitosť úplného šifrovania}} = p \ .$$

Naším primárnym cieľom je dosiahnuť čo najmenšiu redukciu práce útočníka pri už spomínanej redukcii práce používateľa na p percent. Konkrétny spôsob výberu bitov pre selektívne šifrovanie budeme považovať za primerane bezpečný, pokiaľ platí, že:

$$\frac{\text{zložitosť útoku na selektívne šifrovanie}}{\text{zložitosť útoku na úplné šifrovanie}}/p \geq 1 \ . \quad (1)$$

Ďalším zámerom je docieliť čo najmenší priemer a disperziu dĺžky súvislej nezašifrovanej postupnosti bitov. Táto vlastnosť má význam napríklad pri selektívnom šifrovaní kompresovaných dát. Použitý kompresný algoritmus môže mať samosynchronizujúce sa vlastnosti. Potom môže byť žiadúce, aby sa v šifrovom texte nevyskytovalo veľa súvislých nezašifrovaných postupností bitov dlhších než počet bitov potrebný na samosynchronizáciu.

2 Selektory

V článku analyzujeme rôzne spôsoby výberu časti dát pre selektívne šifrovanie. Na dáta sa budeme pozerať ako na postupnosti núl a jednotiek, čiže ako na postupnosti bitov. Dĺžku týchto postupností budeme označovať n. Pre šifrovanie budeme z týchto n bitov vyberať iba p percent[4], čiže iba $k = np$ bitov. V ďalšom texte budeme predpokladať, že $n \geq 1$ a $n \geq k \geq 1$ sú celé čísla a $p \in (0, 1\rangle$ je reálne číslo. Niekedy budeme potrebovať rozdeliť postupnosť dĺžky n na k blokov rovnakej dĺžky b. Vtedy budeme bez ujmy na všeobecnosti predpokladať, že $n = bk$, pričom $b \geq 1$ je celé číslo. Aby sme zjednodušili vznikuté výrazy a zvýraznili ich podstatu, často budeme počítať ich limitu pre n idúce do $+\infty$. Uvedené predpoklady odrážajú praktické aplikácie, kde sa často selektívne šifrujú veľmi veľké dáta (rádovo napríklad 10^{11} bitov). Snáď len $p = 1$ nemá zmysel pre selektívne šifrovanie, ale ponechávame túto možnosť pre úplnosť a porovnanie s plným šifrovaním.

[4] Pripomíname, že percento p používame v matematickom zmysle. To znamená, že celok predstavuje $p = 1$.

Pojmom selektor označujeme spôsob výberu časti bitov pre selektívne šifrovanie. Selektor pre dané n a k vytvorí výber k bitov z pôvodnej n bitovej postupnosti určených pre selektívne šifrovanie. Tento výber je možné opäť reprezentovať bitovou postupnosťou dĺžky n. Takáto postupnosť bude mať práve k jednotiek, ktoré reprezentujú pozície jednotlivých vybraných bitov pre selektívne šifrovanie. Tieto pojmy zavedieme v ďalšom texte formálne.

Definícia 1. *Pojmom (n, k)-**výber** označujeme bitovú postupnosť dĺžky n, ktorá má práve k jednotiek. Bit s hodonotou 1 označuje pozíciu vybranú pre selektívne šifrovanie. Bit s hodnotou 0 označuje pozíciu, ktorá sa pri selektívnom šifrovaní prenesie bezo zmeny. **Dĺžka výberu** – n – je dĺžka postupnosti, ktorou je reprezentovaný. **Hodnosť výberu** – k – je počet jeho jednotkových bitov.*

Napríklad 011101 je $(6, 4)$-výber. Je to výber dĺžky 6, hodnosti 4. Vyberá na šifrovanie všetky bity okrem prvého a predposledného.

Definícia 2. *(n, k)-**selektor** definujeme ako neprázdnu množinu (n, k)-výberov. Selektor pri svojej **aplikácii** vráti náhodne jeden z výberov, ktoré obsahuje. Všetky výbery sa vyberajú s rovnakou pravdepodobnosťou.*

Napr. množina $\{010101, 111000, 000111\}$ je $(6, 3)$-selektor s tromi výbermi. Pri aplikácii tohto selektora dostávame s pravdepodobnosťou $\frac{1}{3}$ výber, ktorý vyberá každý druhý bit, výber, ktorý vyberá prvú polovicu bitov alebo výber, ktorý vyberá druhú polovicu bitov.

Teraz definujeme selektory, ktorých vlastnosti budeme v práci analyzovať.

Definícia 3. *Symbolom[5] $\mathrm{RBS}(n, k)$ označme (n, k)-selektor obsahujúci práve všetky (n, k)-výbery.*

$\mathrm{RBS}(n, k)$ vyberá náhodne ľubovoľných k bitov z postupnosti dĺžky n.

Definícia 4. *Nech $n = bk$. Potom (n, k)-selektor obsahujúci práve všetky (n, k)-výbery, ktoré po rozdelení na bloky veľkosti b obsahujú v každom bloku práve jednu jednotku, označme symbolom[6] $\mathrm{BRBS}(n, k)$.*

$\mathrm{BRBS}(n, k)$ virtuálne rozdelí postupnosť dĺžky $n = bk$ na k blokov dĺžky b, pričom z každého bloku náhodne vyberie práve jeden bit na zašifrovanie. Napríklad $\mathrm{BRBS}(4, 2) = \{01|01, 01|10, 10|01, 10|10\}$.

[5] RBS ako Random Bits Selector

[6] BRBS ako Block Random Bit Selector

Definícia 5. *Nech $n = bk$. Potom označme symbolom*[7] $\mathrm{BLBS}(n,k)$ *nasledovný (n,k)-selektor:*

$$\mathrm{BLBS}(n,k) = \{\overbrace{\underbrace{0\ldots01}_{b}\ldots\underbrace{0\ldots01}_{b}}^{k}\} \ .$$

$\mathrm{BLBS}(n,k)$ virtuálne rozdelí postupnosť dĺžky $n = bk$ na k blokov dĺžky b, pričom z každého bloku vyberie vždy práve posledný bit. Inými slovami $\mathrm{BLBS}(n,k)$ vyberá na zašifrovanie každý b-ty bit.

Definícia 6. *Symbolom*[8] $\mathrm{FS}(n)$ *označíme práve taký (n,n)-selektor, ktorý obsahuje iba (n,n)-výber.*

$$\mathrm{FS}(n) = \{\underbrace{11\ldots1}_{n}\} \ .$$

$\mathrm{FS}(n)$ má iba jeden výber, ktorý vyberá všetky bity postupnosti. Tento selektor nemá priamu aplikáciu pri selektívnom šifrovaní, ale budeme ho používať pre porovnanie selektívneho s plným šifrovaním. Ide vlastne o prípad selektora, na ktorý „zdegenerujú“ všetky vyššie uvedené selektory pokiaľ bude $p = 1$ a teda $k = n$:

$$\mathrm{BLBS}(n,n) = \mathrm{BRBS}(n,n) = \mathrm{RBS}(n,n) = \mathrm{FS}(n) \ .$$

Prípad, keď $p \in (0,1)$ musíme rozdeliť na dve časti. Pokiaľ je $1 < k < n$, tak platí:

$$\mathrm{BLBS}(n,k) \subset \mathrm{BRBS}(n,k) \subset \mathrm{RBS}(n,k) \ .$$

Keď je $k = 1$ dostávame:

$$\mathrm{BLBS}(n,1) \subset \mathrm{BRBS}(n,1) = \mathrm{RBS}(n,1) \ .$$

Definícia 7. ***Beh núl*** *je súvislá (aj prázdna) podpostupnosť nulových bitov z každej strany ohraničená buď jednotkovým bitom alebo hranicou postupnosti.*

Napríklad postupnosť 011001 má dva behy núl dĺžky 0, jeden beh núl dĺžky 1 a jeden beh núl dĺžky 2.

3 Analýza vlastností selektorov

Behy núl v selektoroch predstavujú vlastne nezašifrované časti postupností. Preto sa táto časť práce zaoberá práve analýzou vlastností behov núl vo výberoch generovaných vyššie uvedenými selektormi. Výsledky budú použité v závere práce pri analýze vhodnosti jednotlivých selektorov pre selektívne šifrovanie vzhľadom na kritérium (1).

[7] BLBS ako Block Last Bit Selector
[8] FS ako Full Selector

3.1 Priemerná dĺžka behov núl

Ľahko sa dá nahliadnuť, že počet núl a behov núl v ľubovolnom (n,k)-výbere nezáleží na obmedzeniach, ktoré budeme klásť na možné umiestnenie k jednotkových bitov. Počet núl je vždy $n-k$ a počet behov núl je vždy práve $k+1$.

$\mathrm{BLBS}(n,k)$ obsahuje vždy práve jeden výber. Pri tvorbe výberov do $\mathrm{BRBS}(n,k)$ máme k blokov dĺžky b, pričom z každého bloku vyberáme práve jeden bit. Počet výberov v tomto (n,k)-selektore je teda b^k. Mohutnosť $\mathrm{RBS}(n,k)$ je $\binom{n}{k}$.

Celkový počet	$\mathrm{BLBS}(n,k)$	$\mathrm{BRBS}(n,k)$	$\mathrm{RBS}(n,k)$
- výberov	1	b^k	$\binom{n}{k}$
- núl	$n-k$	$(n-k)b^k$	$(n-k)\binom{n}{k}$
- behov núl	$k+1$	$(k+1)b^k$	$(k+1)\binom{n}{k}$

Tabuľka 1. Celkový počet výberov, núl a behov núl pre analyzované (n,k)-selektory. (Poznámka: $b = n/k$.)

Na základe tabuľky 1 predelením súčtu dĺžok behov núl (celkového počtu núl) a celkového počtu behov dostávame priemernú dĺžku behov núl pre jednotlivé selektory. Tabuľku uvádzame pre dokreslenie vlastností skúmaných selektorov. K výsledku sa dá dopracovať priamejšie, využijúc to, že každý (n,k)-výber má rovnaký počet núl a behov núl a teda aj priemernú dĺžku behu núl. Z toho dostávame aj všeobecnejší záver, že priemerná dĺžka behu núl každého (n,k)-selektora je:

$$\mu_{n,k} = \frac{n-k}{k+1} \ .$$

Pre zjednodušenie výrazu k nahradíme pn a urobíme limitu pre $n \to \infty$:

$$\lim_{n\to\infty} \mu_{n,pn} = \lim_{n\to\infty} \frac{n(1-p)}{pn+1} = \\ = (1-p)\lim_{n\to\infty} \frac{n}{pn+1} = (1-p)\frac{1}{p} = \frac{1}{p} - 1$$

Tento asymptotický tesný odhad pre $\mu_{n,pn}$ budeme označovať μ_p:

$$\mu_p = \frac{1}{p} - 1$$

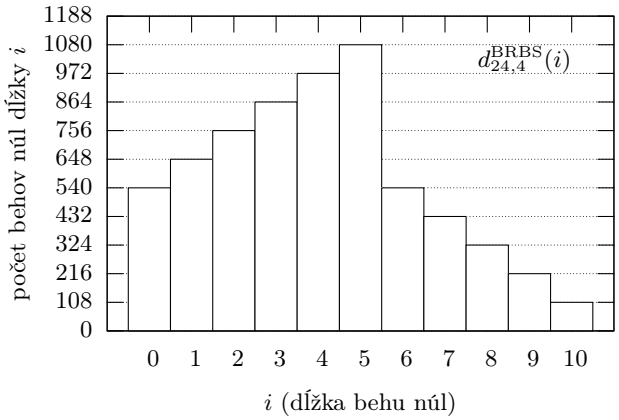
Pre úplnosť ešte spomeňme, že $\mathrm{FS}(n)$ má práve jeden výber, 0 núl, $n+1$ behov núl a priemernú dĺžku behov núl rovnú 0 (čo sedí s $\mu_{n,n}$ aj s μ_p pre $p = 1$).

3.2 Rozdelenie dĺžok behov núl

Aby sme mohli neskôr analyzovať varianciu dĺžky, potrebujeme spočítať, koľko je behov núl danej dĺžky pre jednotlivé selektory.

Pre BLBS(n,k) a FS(n) je analýza jednoduchá a urobíme ju priamo pri výpočte variancie. Pre selektory BRBS(n,k) a RBS(n,k) urobíme túto analýzu samostatne v tejto časti.

Dĺžky behov pre BRBS(n,k). Označme $d_{n,k}^{\mathrm{BRBS}}(i)$ počet behov núl dĺžky i v selektore BRBS(n,k). Opäť predpokladajme, že $n = bk$. Pri tomto selektore sa môžu vyskytovať behy núl dĺžky 0 až $2b-2$. Rozdelenie dĺžok behov ilustruje nasledujúci obrázok.



Obrázok 1. Tento graf znázorňuje $d_{n,k}^{\mathrm{BRBS}}(i)$ pre $n = 24$ a $k = 4$. Veľkosť bloku b je v tomto prípade 6.

Veta 1. *Nech $n,k,b \geq 1$, $n = bk$, $2b-2 \geq i \geq 0$ sú celé čísla. Potom $d_{n,k}^{\mathrm{BRBS}}(i)$ je rovné:*

$$\begin{array}{ll} \{2b & +(i+1)(k-1)\}b^{k-2}, \text{ ak } i < b \\ \{2b\,(k-1) & -(i+1)(k-1)\}b^{k-2}, \text{ ak } b \leq i \end{array} .$$

Dôkaz. Pri dôkaze budeme sčítavať počet postupností obsahujúcich beh núl dĺžky i v danom bloku alebo medzi danými susednými blokmi. Postupne prejdeme cez všetky možné umiestnenia. Takto započítame každý beh núl dĺžky i práve raz. Aj keď nasledujúce úvahy predpokladajú, že $k \geq 2$, ľahko sa dá nahliadnuť, že platia aj pre $k = 1$. Budeme rozlišovať dva prípady:

1. prípad: $0 \leq i \leq b-1$**.** Beh núl dĺžky i sa v prvom bloku vyskytuje práve pri b^{k-1} postupnostiach, pretože prvý blok je pevne určený a v ostatných blokoch môžme jednotky rozmiestniť ľubovoľne. Podobne, beh núl dĺžky i sa v poslednom bloku vyskytuje pri rovnakom počte postupností.

Ostatné behy núl dĺžky i (tie čo sú medzi dvoma blokmi) sa vyskytujú v práve $(k-1)(i+1)b^{k-2}$ postupnostiach, lebo máme $k-1$ susedných blokov, $i+1$ možností ako v dvojici susedných blokov umiestniť beh núl dĺžky i ohraničný jednotkami tak, aby jednotky boli v rôznych blokoch a b^{k-2} možností ako ľubovoľne umiestniť jednotky v ostatných blokoch.

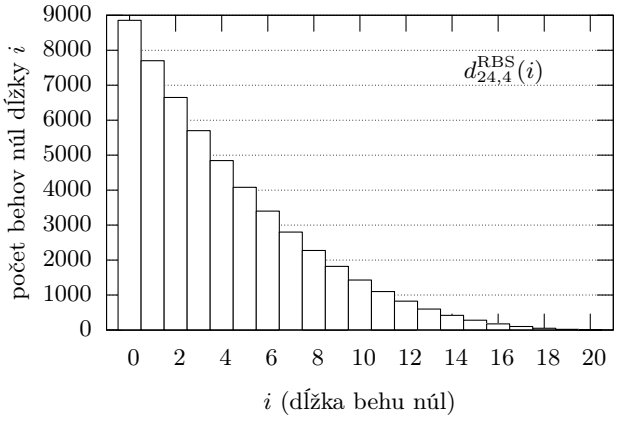
Celkový počet behov v tomto prípade teda je:

$$2b^{k-1} + (k-1)(i+1)b^{k-2} = \{2b + (i+1)(k-1)\}b^{k-2}$$

2. prípad: $b \leq i \leq 2(b-1)$**.** Prvý, ani posledný blok nemôže obsahovať beh núl takejto dĺžky. Zároveň ale musí obsahovať jednotku a teda beh núl kratšej dĺžky, ktorý ale teraz nepočítame.

Ostáva iba možnosť vytvoriť takto dlhý beh núl medzi dvomi susednými blokmi v postupnosti. Podobne ako v 1. prípade, počet takýchto postupností je $(k-1)(2b-i-1)b^{k-2}$. Rozdiel je len v počte možností ako v dvojici susedných blokov umiestniť beh núl dĺžky i ohraničený jednotkami tak, aby jednotky boli v rôznych blokoch. V tomto prípade je počet možností rovný $2b-i-1$. □

Dĺžky behov pre selektor RBS(n,k). Označme $d_{n,k}^{\mathrm{RBS}}(i)$ počet behov núl dĺžky i v selektore RBS(n,k). Pri tomto selektore sa vyskytujú behy núl dĺžky 0 až $n-k$. Rozdelenie dĺžok behov je ilustrované na nasledujúcom obrázku.



Obrázok 2. Tento graf znázorňuje $d_{n,k}^{\mathrm{RBS}}(i)$ pre $n = 24$ a $k = 4$. Aj keď to z grafu nie je dobre vidieť, $d_{24,4}^{\mathrm{RBS}}(19)$ a $d_{24,4}^{\mathrm{RBS}}(20)$ sú nenulové (ich hodnota je 20, respektíve 5).

Veta 2. *Nech $n \geq 1$, $n \geq k \geq 1$ a $n-k \geq i \geq 0$ sú celé čísla. Potom pre $d_{n,k}^{\mathrm{RBS}}(i)$ platí:*

$$d_{n,k}^{\mathrm{RBS}}(i) = (k+1)\binom{n-1-i}{k-1} .$$

Dôkaz. Všetky behy núl dĺžky i môžeme získať nasledovným spôsobom: Zoberme postupnosť bitov dĺžky $n-i-1$ obsahujúcu práve $k-1$ jednotiek. Pred aj za túto postupnosť dopíšeme jednu jednotku ako „zarážku". Takýchto postupností existuje práve $\binom{n-i-1}{k-1}$.

Teraz môžeme ktorúkoľvek jednotku (tých je vrátenie zarážiek $k+1$) nahradiť behom núl dĺžky i ohraničeným jednotkami. Tým pridáme $i+1$ nových bitov a celá postupnosť tak bude mať $n+2$ bitov a $k+2$ jednotiek (vrátane zarážiek). Zmazaním prvého a posledného bitu (budú vždy jednotkové – zarážky) dostávame postupnosť dĺžky n s práve k jednotkami. Celkový počet behov núl dĺžky i je teda $(k+1)\binom{n-i-1}{k-1}$. □

3.3 Variancia dĺžky behov núl

Ako sme už ukázali, každý (n,k)-selektor má rovnakú priemernú dĺžku behov núl. Na porovnaie rovnomernosti rozloženia šifrovaných bitov potrebujeme ešte poznať varianciu tejto dĺžky.

Variancia pre selektor BLBS(n,k). Tento selektor má najviac dve dĺžky behov. Celkovo má k behov núl dĺžky $b-1$ a jeden beh núl dĺžky 0.

$$\mathrm{var}_{n,k}^{\mathrm{BLBS}} = \frac{\big[\mu_{n,k}-(b-1)\big]^2 k + \mu_{n,k}^2}{k+1} = \\ = \frac{1}{k}\left(\frac{n-k}{k+1}\right)^2 = \frac{\mu_{n,k}^2}{k} \ .$$

Varianciu pre BLBS(n,k) kazí iba jeden nulový beh za poslednou jednotkou. Preto keď pôjdeme s $n\to\infty$ dostávame, že variancia je 0 pre každé p.

$$\lim_{n\to\infty} \mathrm{var}_{n,pn}^{\mathrm{BLBS}} = \frac{n(1-p)^2}{n^2p^3+n2p^2+p} = 0 \ .$$

Variancia pre selektor BRBS(n,k). Rozdelenie dĺžok behov núl pre tento selektor sme analyzovali v predošlej časti. Využijúc vetu 1 môžeme varianciu pre BRBS(n,k) zapísať ako:

$$\mathrm{var}_{n,k}^{\mathrm{BRBS}} = \frac{\sum_{i=0}^{2(b-1)}(\mu_{n,k}-i)^2 d_{n,k}^{\mathrm{BRBS}}(i)}{\sum_{i=0}^{2(b-1)} d_{n,k}^{\mathrm{BRBS}}(i)} = \\ = \frac{\frac{b^k(n-k)[k^3+k^2(n-2)+k(4n+3)-3n]}{6k^2(k+1)}}{(k+1)b^k} = \\ = \frac{(n-k)[k^3+k^2(n-2)+k(4n+3)-3n]}{6k^2(k+1)^2}.$$

$$\lim_{n\to\infty} \mathrm{var}_{n,pn}^{\mathrm{BRBS}} = \\ = \frac{1-p}{6p^2}\lim_{n\to\infty}\frac{n^2p^2(1+p)+n2p(2-p)-3(1-p)}{n^2p^2+n2p+1} \\ = \frac{(1-p)(1+p)}{6p^2} \ .$$

Variancia pre RBS(n,k). Rozdelenie dĺžok behov núl pre tento selektor sme taktiež analyzovali v predošlej časti. Využijúc vetu 2 môžeme varianciu pre RBS(n,k) zapísať ako:

$$\mathrm{var}_{n,k}^{\mathrm{RBS}} = \frac{\sum_{i=0}^{n-k}(\mu_{n,k}-i)^2 d_{n,k}^{\mathrm{RBS}}(i)}{\sum_{i=0}^{n-k} d_{n,k}^{\mathrm{RBS}}(i)} = \\ = \frac{\frac{(n-k)(n+1)n}{(k+1)(k+2)}\binom{n-1}{k-1}}{(k+1)\binom{n-1}{k-1}\frac{n}{k}} = \frac{(n-k)(n+1)k}{(k+1)^2(k+2)} \ .$$

$$\lim_{n\to\infty} \mathrm{var}_{n,pn}^{\mathrm{RBS}} = \lim_{n\to\infty}\frac{n^2(1-p)(n+1)p}{(pn+1)^2(pn+2)} = \frac{1-p}{p^2} \ .$$

Variancia pre selektor FS(n). Vzhľadom na to, že FS(n) má iba $n+1$ behov núl dĺžky 0 je možné varianciu tohto selektora vyjadriť nasledovne:

$$\mathrm{var}_{n}^{\mathrm{FS}} = \frac{\mu_{n,n}^2(n+1)}{n+1} = 0 \ .$$

Zhrnutie variancie. Označme asymptoticky tesný odhad $\mathrm{var}_{n,pn}^{\mathrm{S}}$ pre $n\to\infty$ symbolom $\mathrm{var}_{p}^{\mathrm{S}}$. V tabuľke 2 je prehľad týchto asymptoticky tesných odhadov variancie pre skúmané selektory. Ako sa dá intuitívne očakávať, pre $p\in(0,1)$ platí:

$$\mathrm{var}_{n}^{\mathrm{FS}} = \mathrm{var}_{p}^{\mathrm{BLBS}} < \mathrm{var}_{p}^{\mathrm{BRBS}} < \mathrm{var}_{p}^{\mathrm{RBS}} \ .$$

Pre $p=1$ sú všetky variancie nulové, keďže v (n,n)-selektore má každý beh núl dĺžku 0.

$\mathrm{var}_{p}^{\mathrm{BLBS}}$	$\mathrm{var}_{p}^{\mathrm{BRBS}}$	$\mathrm{var}_{p}^{\mathrm{RBS}}$	$\mathrm{var}_{n}^{\mathrm{FS}}$
0	$\frac{(1-p)(1+p)}{6p^2}$	$\frac{1-p}{p^2}$	0

Tabuľka 2. Variancia dĺžky behov núl jednotlivých skúmaných (n,pn)-selektorov pre $n\to\infty$.

3.4 Počet otvorených textov

Naším primárnym cieľom je podľa (1) analyzovať priebeh funkcie:

$$w_n^{\mathrm{S}}(p) = \frac{\text{zložitosť útoku na selektívne šifrovanie so selektorom } \mathrm{S}(n, pn)}{\text{zložitosť útoku na úplné šifrovanie}} / p \ .$$

Opäť sa budeme snažiť funkciu zjednodušiť asymptoticky tesným odhadom pre $n \to \infty$. Pre tento odhad zavedieme označenie $w^{\mathrm{S}}(p)$:

$$w^{\mathrm{S}}(p) = \lim_{n\to\infty} w_n^{\mathrm{S}}(p) \ .$$

Keďže zložitosť útoku meriame mohutnosťou množiny potencionálnych otvorených textov, potrebujeme najprv spočítať túto mohutnosť pre selektívne šifrovanie používajúce jednotlivé selektory. Pre zjednodušenie budeme namiesto pojmu mohutnosť množiny potencionálnych otvorených textov ďalej písať iba počet otvorených textov.

Počet otvorených textov pre BLBS(n, k). Tento prípad je triviálny. Keďže $\mathrm{BLBS}(n, k)$ má iba jeden výber s k jednotkami, je počet otvorených textov rovný 2^k. Odtiaľ dostávame, že:

$$w_n^{\mathrm{BLBS}}(p) = \frac{2^{np}}{2^n} / p \ .$$

$$w^{\mathrm{BLBS}}(p) = \lim_{n\to\infty} \frac{2^{np}}{2^n} / p = \begin{cases} 0 \text{ ak } p \in (0,1) \\ 1 \text{ ak } p = 1 \end{cases} \ .$$

Počet otvorených textov pre BRBS(n, k). Pre každý blok je b možností výberu bitu pre zašifrovanie. Tento bit sa po zašifrovaní zmení na opačný, alebo ponechá bez zmeny. V každom bloku preto dostávame b potencionálnych otvorených blokov so zmeneným jedným bitom a jeden bez zmeny bitov. Počet otvorených textov pre $\mathrm{BRBS}(n, k)$ je teda rovný $(b+1)^k$.

$$w_n^{\mathrm{BRBS}}(p) = \frac{\left(\frac{1}{p}+1\right)^{np}}{2^n} / p \ .$$

$$w^{\mathrm{BRBS}}(p) = \lim_{n\to\infty} \frac{\left(\frac{1}{p}+1\right)^{np}}{2^n} / p = \begin{cases} 0 \text{ ak } p \in (0,1) \\ 1 \text{ ak } p = 1 \end{cases} \ .$$

Počet otvorených textov pre RBS(n, k). Pri šifrovaní vybraných k bitov sa tieto bity môžu zmeniť na opačné alebo ostať bezo zmeny. Nezmenené bity nemenia možný otvorený text. Preto môžme spočítať počet otvorených textov pre $\mathrm{RBS}(n, k)$ ako $\sum_{i=0}^{k} \binom{n}{i}$.

$$w_n^{\mathrm{RBS}}(p) = \frac{\sum_{i=0}^{np} \binom{n}{i}}{2^n} / p \ .$$

$\sum_{i=0}^{np} \binom{n}{i}$ – súčet prvých p percent binomických koeficientov sa vo všeobecnosti nedá vyjadriť v uzavretom tvare a pre $n \to \infty$ ide tiež do nekonečna.

Označme symbolom $\delta_n(p)$ podiel tejto čiastočnej sumy na celkovej sume:

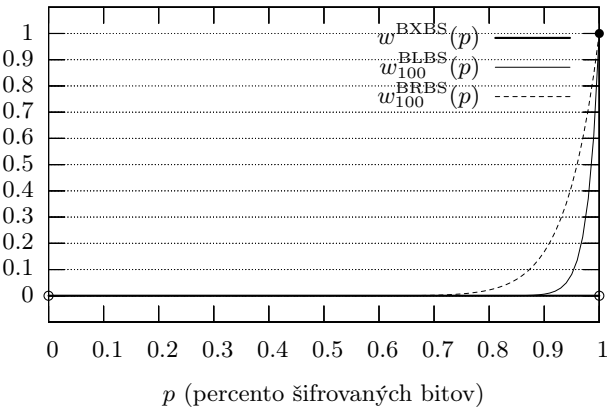
$$\delta_n(p) = \sum_{i=0}^{np} \binom{n}{i} / 2^n \ .$$

Tento podiel sa však už pre $n \to \infty$ dá asymptoticky tesne odhadnúť nasledovnou funkciou [4]:

$$\delta(p) = \lim_{n\to\infty} \delta_n(p) = \begin{cases} 0 \text{ ak } p < \frac{1}{2} \\ \frac{1}{2} \text{ ak } p = \frac{1}{2} \\ 1 \text{ ak } p > \frac{1}{2} \end{cases} \ .$$

Teraz už môžeme vypočítať asymptoticky tesný odhad $w_n^{\mathrm{RBS}}(p)$ pre $n \to \infty$:

$$w^{\mathrm{RBS}}(p) = \lim_{n\to\infty} \delta_n(p)/p = \begin{cases} 0 \text{ ak } p \in (0, \frac{1}{2}) \\ 1 \text{ ak } p = \frac{1}{2} \\ \frac{1}{p} \text{ ak } p \in (\frac{1}{2}, 1\rangle \end{cases} \ .$$

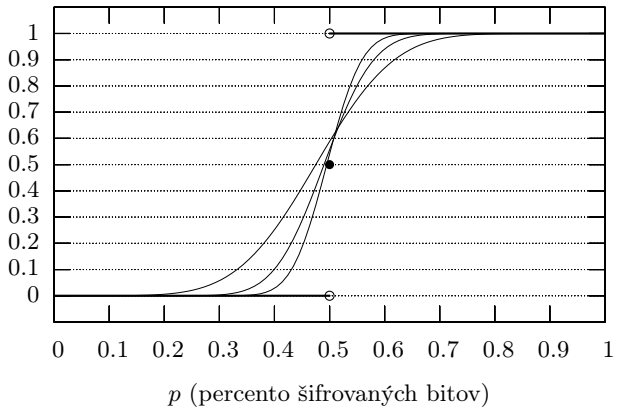


Obrázok 3. Tento graf znázorňuje $w_{100}^{\mathrm{BLBS}}(p)$, $w_{100}^{\mathrm{BRBS}}(p)$ ako aj asymtotický tesný odhad oboch funkcií pre $n \to \infty$ označený ako $w^{\mathrm{BXBS}}(p)$.

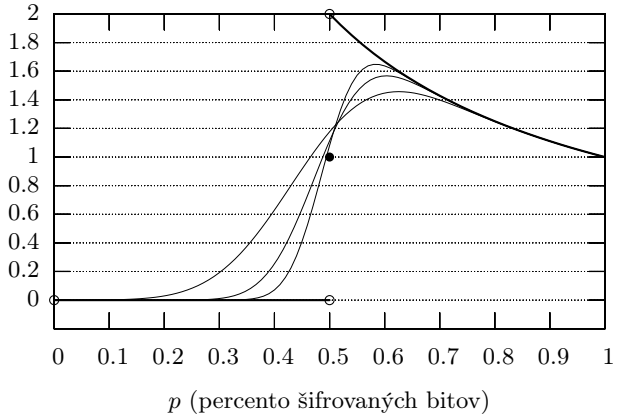
Počet otvorených textov pre FS(n). Tento prípad je jednoduchý. Keďže $\mathrm{FS}(n)$ má iba jeden výber s n jednotkami, je počet otvorených textov rovný 2^n.

4 Analýza vhodnosti selektorov

Za hlavné kritérium vhodnosti selektora S pre selektívne šifrovanie sme si zvolili podmienku (1). Inak povedané, má platiť $w_n^{\mathrm{S}}(p) \geq 1$, pre $p \in (0,1)$. Pretože



Obrázok 4. Tento graf znázorňuje $\delta_{20}(p)$, $\delta_{50}(p)$, $\delta_{100}(p)$ a asymptoticky tesný odhad $\delta(p)$.



Obrázok 5. Tento graf znázorňuje $w_{20}^{\mathrm{RBS}}(p)$, $w_{50}^{\mathrm{RBS}}(p)$, $w_{100}^{\mathrm{RBS}}(p)$ a asymptoticky tesný odhad $w^{\mathrm{RBS}}(p)$.

selektívne šifrovanie sa používa najmä na veľkých dátach, bude nás zaujímať platnosť tejto podmienky hlavne pre $n \to \infty$, teda:

$$w^{\mathrm{S}}(p) \geq 1, \text{ pre } p \in (0,1) \ .$$

Žiaľ ani jeden z analyzovaných selektorov túto vlastnosť nespĺňa na celom intervale $(0,1)$. Ako vidíme na obrázku 3, $w^{\mathrm{BLBS}}(p)$ aj $w^{\mathrm{BRBS}}(p)$ sú na tomto intervale nulové. To znamená, že útočníkovi sa práca pri dešifrovaní oveľa viac zjednoduší ako používateľovi pri šifrovaní. Inak povedané strata bezpečnosti je neúmerná urýchleniu šifrovania.

Z obrázku 5 však vidíme, že $w^{\mathrm{RBS}}(p)$ podmienku spĺňa, ale iba na intervale $p \in \langle 1/2, 1\rangle$. Dokonca:

$$w^{\mathrm{RBS}}(p) > 1, \text{ pre } p \in (1/2, 1) \ .$$

Pre p blížiace sa k $1/2$ dosahuje $w^{\mathrm{RBS}}(p)$ najväčšie hodnoty (blíži sa k 2). Pre $p = 1/2$ je už $w^{\mathrm{RBS}}(p) = 1$. To znamená, že pokiaľ pre selektívne šifrovanie zvolíme selektor $\mathrm{RBS}(n,k)$, tak ak šifrujeme aspoň polovicu bitov, zjednodušenie útočníkovej práce nie je väčšie ako zjednodušenie práce používateľa pri šifrovaní. Takéto selektívne šifrovanie považujeme na základe (1) za primerane bezpečné.

Pokiaľ by sme šifrovali p percent dát, pričom p je tesne nad $1/2$, tak ako vidieť na obrázku 4, útočníkova práca pri dešifrovaní sa vlastne nezmení (oproti plnému šifrovaniu) ale používateľova práci pri šifrovaní klesne tesne nad $1/2$. Ak by sme však zvolili p čo i len tesne pod $1/2$, tak útočníková práca pri dešifrovaní sa opäť oveľa viac zjednoduší ako používateľovi pri šifrovaní.

Vedľajším zámerom bolo dosiahnuť čo najrovnomernejšie rozdelenie šifrovaných bitov. V tomto prípade sme ukázali, že každý (n,k)-selektor má priemernú dĺžku nezašifrovanej postupnosti rovnú $\mu_{n,k} = \frac{n-k}{k+1}$. To pre $n \to \infty$ znamená, že sa zašifruje v priemere každý $1/p$-ty bit, čo je ideálny výsledok. Žiaľ, $\mathrm{RBS}(n,k)$ má najväčšiu varianciu tejto dĺžky. Toto kritérium však považujeme za menej dôležité.

5 Záver

V článku sme študovali niektoré selektory, ktoré človeku prirodzene napadnú, keď chce šifrovať iba p percent bitov. Ukázali sme, že väčšina týchto selektorov výrazne oslabuje bezpečnosť výsledného selektívneho šifrovania. Z analyzovaných selektorov môžeme odporučiť iba $\mathrm{RBS}(n,k)$, pre $k > n/2$, pričom čím je k bližšie k $n/2$, tým je bezpečnosť vyššia.

Referencie

1. Shi C., Bhargava B., A Fast MPEG Video Encryption Algorithm. In: Proceedings of the Sixth ACM International Conference on Multimedia, ACM Press, 1998, 81–88
2. Droogenbroeck M.V., Benedett R., Techniques for a Selective Encryption of Uncompressed and Compressed Images. In: Proceedings of ACIVS 2002 (Advanced Concepts for Intelligent Vision Systems), Ghent, Belgium, 2002, 90–97
3. Liu X., Eskicioglu A.M., Selective Encryption of Multimedia Content in Distribution Networks: Challenges and New Directions. ASTED International Conference on Communications, Internet and Information Technology (CIIT 2003), 2003
4. Olejár D., Stanek M., On Cryptographic Properties of Random Boolean Functions. J.UCS: Journal of Universal Computer Science, 4, 1998, 705–718

Generalized PGV hash functions are not collision resistant

Ľubica Staneková[1] and Martin Stanek[2]

[1] Department of Mathematics, Slovak University of Technology,
Radlinského 11, 813 68 Bratislava, Slovak Republic,
ls@math.sk
[2] Department of Computer Science, Comenius University,
Mlynská dolina, 842 48 Bratislava, Slovak Republic
stanek@dcs.fmph.uniba.sk

Abstract. *Cryptographic hash functions are important tools for building various cryptographic applications. We study generalized rate-2 PGV hash functions and show that none of them is collision resistant. For every hash function, we present an adversary, running in a constant time, that produces collisions. Our result answers an open question stated in [2], and supplements the general upper bound for the rate of collision resistant hash functions proved in [4].*

1 Introduction

Almost all modern hash functions are built by iterating a compression function according to the Merkle-Damgård paradigm. The compression functions can be based on some underlying block cipher. The first systematic study of 64 block cipher-based hash functions was done by Preneel, Govaerts, and Vandewalle [3] (we call them PGV hash functions). Subsequently, Black, Rogaway, and Shrimpton [1] analyzed these constructions in the black-box model and proved that 20 of them are collision resistant up to the birthday-attack bound.

An important property of a hash function is its rate – the number of message blocks processed with one block cipher transformation (when the message block and cipher block lengths are equal). All PGV hash functions are rate-1, they process one message block with one transformation.

The rate-2 PGV-like compression functions, where a single block cipher transformation processes two message blocks, were studied in [2]. It was proved that none of these compression function is collision resistant. However, collision resistance of compression function is a sufficient, but not necessary condition for collision resistance of iterated hash function [1]. Thus, there might exist rate-2 PGV-like compression functions that form a collision resistant hash function. This possibility was stated as an open problem in [2].

General upper bounds for the rate of collision resistant compression functions and hash functions were proved in [4]. For the particular case of high-rate PGV-like hash functions, one gets 2 as the upper bound for their rate. This is an additional motivation to study these hash functions.

We study natural rate-2 generalizations of PGV hash functions. We show that none of these hash functions is collision resistant. Surprisingly, the collisions can be constructed by a very weak adversary, running in a constant time. This result answers an open question from [2], and complements the results obtained in [4].

The paper is structured as follows. In Section 2 we introduce basic definitions and notations used through the paper. Section 3 contains analysis of 512 rate-2 PGV-like hash functions. These hash functions are partitioned into distinct sets, according the properties of underlying compression functions, and each set of hash functions is analyzed separately.

2 Definitions

We present definitions tailored to the purposes of our analysis of rate-2 PGV-like hash functions. Let V_n be the set of all n-ary binary vectors, i.e. $V_n = \{0,1\}^n$, for a positive integer n. A block cipher is a function $E : V_n \times V_n \to V_n$, where for each key $K \in V_n$, the function $E_K(\cdot) = E(K, \cdot)$ is a permutation on V_n (we assume that the block length and the key length in the block cipher are equal). The decryption function will be denoted by E^{-1}.

Let M be a message we want to hash. First, the message is divided, possibly after some padding, into blocks of equal length: $M = m_1, \ldots, m_l$, where $|m_i| = rn$ for a positive integer r. The iterated hash $H(M)$ is computed as follows ($h_0 \in V_n$ is a fixed initialization vector):

$$\begin{aligned} h_i &= f(h_{i-1}, m_i) \quad \text{for } i = 1, \ldots, l \\ H(M) &= h_l, \end{aligned} \tag{1}$$

where $f : V_n \times V_{rn} \to V_n$ is a compression function. If a compression function f uses a block cipher to compute its value, we call it a block cipher-based compression function. Moreover, if f uses single E transformation, we say it has rate-r. The rate denotes the number of n-bit message blocks processed by single block cipher transformation. The higher the rate is, the faster hash function one can expect.

Remark 1. When necessary, we divide rn-bit message block m_i into n-bit blocks: $m_i = m_i^{(1)}, m_i^{(2)}, \ldots, m_i^{(r)}$.

One of the most important properties of hash functions is collision resistance. Informally, a hash function H is collision resistant if it is infeasible to produce two distinct messages M, and M' having equal hash, i.e.

$$M \neq M' \quad \wedge \quad H(M) = H(M').$$

Remark 2. Since our paper shows that certain hash functions are *not* collision resistant, this informal definition is sufficient for such purpose.

2.1 General upper bound

A more general model of high-rate compression/hash functions was proposed and studied in [2, 4]. The computation of the (rate-r) compression function $f : V_a \times V_{nr} \to V_a$ is defined as:

$$f(h, m) = f_3(h, m, E_{f_2(h,m)}(f_1(h, m))),$$

where $f_1 : V_a \times V_{rn} \to V_n$, $f_2 : V_a \times V_{rn} \to V_k$, and $f_3 : V_a \times V_{rn} \times V_n \to V_a$ are arbitrary functions.

Notice, that the model allows distinct block length, key length, and length of f output. A hash function is obtained by iterating the compression function f, see (1).

In this model, we proved [4] the following upper bound on the rate of collision resistant compression function:

$$r \leq 1 + \frac{k - \varepsilon a/2}{n},$$

and the following upper bound on the rate of collision resistant hash function:

$$r \leq 1 + \frac{k + a/(2^{\varepsilon a/2})}{n}, \tag{2}$$

for arbitrary $0 \leq \varepsilon < 1$.

When $a = k = n$ we get compression/hash functions described above. Moreover, by suitable choice of f_1, f_2, and f_3 functions we obtain generalized rate-2 PGV hash functions, see Section 2.2. In this case the general upper bound (2) simplifies to $r \leq 2$.

2.2 Generalized rate-2 PGV hash functions

Rate-2 PGV-like hash functions are hash functions obtained by iterating the following compression functions:

$$f(h, (m^{(1)}, m^{(2)})) = E_a(b) \oplus c, \tag{3}$$

where a, b, c are arbitrary linear combinations of inputs or some fixed value $v \in V_n$. Thus,

$$a, b, c \in \{h, m^{(1)}, m^{(2)}, h \oplus m^{(1)}, h \oplus m^{(2)}, m^{(1)} \oplus m^{(2)}, h \oplus m^{(1)} \oplus m^{(2)}, v\},$$

where $\oplus$ denotes a bitwise addition mod 2 (i.e. XOR operation). This way we obtain 512 compression functions, and by iteration according to (1) the same number of hash functions. We denote this set of rate-2 PGV-like hash function by $\mathcal{H}_{\mathrm{PGV}}$.

3 Collisions in $\mathcal{H}_{\mathrm{PGV}}$

We analyze all hash functions from $\mathcal{H}_{\mathrm{PGV}}$, and show that none of them is collision resistant. We employ technique similar to [2], where collisions in compression functions (3) were presented. We partition $\mathcal{H}_{\mathrm{PGV}}$ into sets according to our ability to design a collision-producing adversary. However, finding collisions in a hash function is (usually) more challenging task than finding collisions in a compression function, since

1. Collisions must start with the same fixed initial vector h_0.
2. Collisions should be of equal length. It ensures that collisions are not affected by length-encoding padding, such as Merkle-Damgård strengthening.

The following property is useful for our analysis of hash functions from $\mathcal{H}_{\mathrm{PGV}}$.

Definition 1. *Compression functions f, f' of the form* (3) *are collision-equivalent (or c-equivalent) if f can be transformed to f' using some of the following substitutions:*

$$\begin{aligned}(m^{(1)}, m^{(2)}) &\mapsto (m^{(1)}, m^{(1)} \oplus m^{(2)}) \\ &\mapsto (m^{(2)}, m^{(1)}) \\ &\mapsto (m^{(2)}, m^{(1)} \oplus m^{(2)}) \\ &\mapsto (m^{(1)} \oplus m^{(2)}, m^{(1)}) \\ &\mapsto (m^{(1)} \oplus m^{(2)}, m^{(2)})\end{aligned}$$

Example 1. Let $f(h, (m^{(1)}, m^{(2)})) = E_{m^{(1)}}(m^{(2)} \oplus h) \oplus m^{(1)} \oplus m^{(2)}$. Then the substitution $(m^{(1)}, m^{(2)}) \mapsto (m^{(1)} \oplus m^{(2)}, m^{(1)})$ leads to the compression function $f'(h, (m^{(1)}, m^{(2)})) = E_{m^{(1)} \oplus m^{(2)}}(m^{(1)} \oplus h) \oplus m^{(2)}$.

Let f and f' are c-equivalent compression functions. It is easy to see that finding collisions in the hash function based on f (we denote it H_f) is equally hard (easy) as finding collisions in the hash function $H_{f'}$ based on f'. We illustrate this on substitution $(m^{(1)}, m^{(2)}) \mapsto (m^{(1)} \oplus m^{(2)}, m^{(1)})$, the opposite substitution and the other cases are analogous. Let $M = (m_1^{(1)}, m_1^{(2)}, \ldots, m_l^{(1)}, m_l^{(2)})$ and $M' = (m'^{(1)}_1, m'^{(2)}_1, \ldots, m'^{(1)}_l, m'^{(2)}_l)$ form a collision in hash function H_f, i.e.

$$M \neq M' \quad \wedge \quad H_f(M) = H_f(M').$$

Then a collision for $H_{f'}$ can be obtained immediately:

$$(m_1^{(2)}, m_1^{(1)} \oplus m_1^{(2)}, \ldots, m_l^{(2)}, m_l^{(1)} \oplus m_l^{(2)})$$
$$(m'_1{}^{(2)}, m'_1{}^{(1)} \oplus m'_1{}^{(2)}, \ldots, m'_l{}^{(2)}, m'_l{}^{(1)} \oplus m'_l{}^{(2)})$$

We call two hash functions c-equivalent, if their underlying compression functions are c-equivalent. It is clear from the previous discussion that c-equivalent hash functions have the same collision resistance.

The set $\mathcal{H}_{\mathrm{PGV}}$ is partitioned into 7 sets, see Table 1. A hash function $H \in \mathcal{H}_{\mathrm{PGV}}$ is assigned to the ith set (partition) if H or any of its c-equivalent hash functions satisfies the conditions for this set, and does not satisfy conditions for the sets $1, \ldots, i-1$. Thus, each function is assigned to the first suitable set. The last set contains "unassignable" hash functions.

A detailed description of each set, together with conditions for assigning hash functions, is given in the following subsections.

	set	# functions
1	$\mathcal{H}_{\mathrm{rmb}}$	120
2	$\mathcal{H}_{\mathrm{rhv}}$	49
3	$\mathcal{H}_{\mathrm{cvc}}$	133
4	$\mathcal{H}_{\mathrm{cvb}}$	91
5	$\mathcal{H}_{\mathrm{bal}}$	35
6	$\mathcal{H}_{\mathrm{hvc}}$	12
7	$\mathcal{H}_{\mathrm{spc}}$	72

Table 1. Partitions of $\mathcal{H}_{\mathrm{PGV}}$ and their cardinalities.

The hash functions from the first 6 sets can be attacked quite straightforwardly. The last set $\mathcal{H}_{\mathrm{spc}}$ must be analyzed more carefully, see Sect. 3.7.

3.1 Redundant message blocks – $\mathcal{H}_{\mathrm{rmb}}$

The set $\mathcal{H}_{\mathrm{rmb}}$ contains those hash functions, for which the underlying compression function does not depend on all message blocks (i.e. $m^{(1)}$ and $m^{(2)}$). Then it is easy to compute colliding messages by altering the redundant message block.

Example 2. A compression function

$$f(h, (m^{(1)}, m^{(2)})) = E_{m^{(2)}}(v) \oplus h \oplus m^{(2)}$$

does not depend on $m^{(1)}$. Hence, any pair of two n-bit block messages $\langle x, w\rangle$ and $\langle x', w\rangle$ collides: $H(\langle x, w\rangle) = E_w(v) \oplus h_0 \oplus w = H(\langle x', w\rangle$.

3.2 Redundant hash value – $\mathcal{H}_{\mathrm{rhv}}$

The set $\mathcal{H}_{\mathrm{rhv}}$ is a selection of hash functions from $\mathcal{H}_{\mathrm{PGV}} \setminus \mathcal{H}_{\mathrm{rmb}}$. We select hash functions with compression functions that do not depend of the intermediate hash value (i.e. h). The value of these hash functions depend only on the last two n-bit blocks of input message. Collisions can be produced, for example, by taking any 4-block message and altering first two blocks.

Example 3. A compression function

$$f(h, (m^{(1)}, m^{(2)})) = E_{m^{(1)}}(m^{(1)} \oplus m^{(2)}) \oplus m^{(2)}$$

does not depend on h. Then trivially any pair of 4-block messages $\langle x, y, w, z\rangle$ and $\langle x', y', w, z\rangle$ collides.

3.3 Compensable values in "c" – $\mathcal{H}_{\mathrm{cvc}}$

The set $\mathcal{H}_{\mathrm{cvc}}$ contains hash functions not contained in two previous sets, that satisfy the following condition: either the block $m^{(1)}$ or the block $m^{(2)}$ appears solely in position "c", see (3), in the underlying compression function. We call this message block *out-standing* message block.

An adversary can construct collisions of 2-block messages, because (s)he can "compensate" resulting hash value by suitable choice of the out-standing message block.

Example 4. A compression function

$$f(h, (m^{(1)}, m^{(2)})) = E_{m^{(2)}}(h \oplus m^{(2)}) \oplus m^{(1)}$$

has an out-standing message block $m^{(1)}$. The colliding 2-block messages are $\langle x, y\rangle$ and $\langle x', y'\rangle$, where $x' = E_y(h_0 \oplus y) \oplus x \oplus E_{y'}(h_0 \oplus y')$:

$$\begin{aligned} H(\langle x, y\rangle) &= E_y(h_0 \oplus y) \oplus x \\ &= x' \oplus E_{y'}(h_0 \oplus y') \\ &= H(\langle x', y'\rangle) \end{aligned}$$

3.4 Compensable values in "b" – $\mathcal{H}_{\mathrm{cvb}}$

The condition for the set $\mathcal{H}_{\mathrm{cvb}}$ is similar to the previous case. This time the condition on underlying compression function is as follows: either the block $m^{(1)}$ or the block $m^{(2)}$ appears solely in position "b", see (3). We call this message block *in-standing* message block.

An adversary can construct collisions of 2-block messages by compensating the input of the block cipher by suitable choice of the in-standing message block.

Example 5. A compression function

$$f(h, (m^{(1)}, m^{(2)})) = E_{m^{(2)}}(h \oplus m^{(1)} \oplus m^{(2)}) \oplus v$$

has an in-standing message block $m^{(1)}$. The colliding 2-block messages are $\langle x, y\rangle$ and $\langle x', y'\rangle$, where $x' = h_0 \oplus y' \oplus E_{y'}^{-1}(E_y(h_0 \oplus x \oplus y))$:

$$\begin{aligned} H(\langle x', y'\rangle) &= E_{y'}(h_0 \oplus x' \oplus y') \oplus v \\ &= E_{y'}(E_{y'}^{-1}(E_y(h_0 \oplus x \oplus y))) \oplus v \\ &= E_y(h_0 \oplus x \oplus y) \oplus v \\ &= H(\langle x, y\rangle) \end{aligned}$$

3.5 Balanced combinations – $\mathcal{H}_{\mathbf{bal}}$

The set $\mathcal{H}_{\mathrm{bal}}$ contains hash functions (not contained in the previous sets) with balanced combinations of message blocks in their underlying compression function. Balanced combinations of message blocks means that for all parameters "a", "b", and "c", the following condition holds: $m^{(1)}$ is contained in the parameter if and only if $m^{(2)}$ is contained in the parameter.

It is easy to produce collision in this case. It is sufficient to have constant value of $m^{(1)} \oplus m^{(2)}$.

Example 6. A compression function

$$f(h, (m^{(1)}, m^{(2)})) = E_h(h \oplus m^{(1)} \oplus m^{(2)}) \oplus m^{(1)} \oplus m^{(2)}$$

has balanced combination of message blocks. Let $x \neq y$ be n-bit blocks. Then $H(\langle x, y\rangle) = H(\langle y, x\rangle)$ is a collision.

3.6 Hash value in "c" – $\mathcal{H}_{\mathbf{hvc}}$

The compression functions of hash functions in $\mathcal{H}_{\mathrm{hvc}}$ share this property: the intermediate hash value "h" appears solely in the "c" parameter. Hence, the hash value is computed as a XOR of some intermediate hash values. Changing (permuting) the order of $(2n)$-bit blocks does not change the hash value – this is the way how to produce collision for this set of hash functions.

Example 7. A compression function

$$f(h, (m^{(1)}, m^{(2)})) = E_{m^{(1)} \oplus m^{(2)}}(m^{(2)}) \oplus h \oplus m^{(1)}$$

has h solely in the "c" parameter. One can easily check that $H(\langle x, y, w, z\rangle) = H(\langle w, z, x, y\rangle)$ for arbitrary n-bit blocks x, y, w, z.

3.7 Special functions – $\mathcal{H}_{\mathbf{spc}}$

The set of special hash functions contains the remaining hash functions from $\mathcal{H}_{\mathrm{PGV}}$:

$$\mathcal{H}_{\mathrm{spc}} = \mathcal{H}_{\mathrm{PGV}} \backslash (\mathcal{H}_{\mathrm{rmb}} \cup \mathcal{H}_{\mathrm{rhv}} \cup \mathcal{H}_{\mathrm{cvc}} \cup \mathcal{H}_{\mathrm{cvb}} \cup \mathcal{H}_{\mathrm{bal}} \cup \mathcal{H}_{\mathrm{hvc}}).$$

There are 72 hash functions in $\mathcal{H}_{\mathrm{spc}}$. These hash functions can be partitioned according to the c-equivalent relations into 12 subsets (6 hash functions in each subset). Since c-equivalent hash functions have the same complexity of finding collisions, it suffices to analyze one member of each subset. Table 2 shows one of the possible selections (hash functions are represented by their underlying compression functions). We will refer compression/hash functions from Table 2 as $f_1/H_1, \ldots, f_{12}/H_{12}$.

i	f_i
1	$E_{m^{(2)}}(h \oplus m^{(1)}) \oplus m^{(1)}$
2	$E_{m^{(2)}}(h \oplus m^{(1)}) \oplus m^{(1)} \oplus m^{(2)}$
3	$E_{m^{(2)}}(h \oplus m^{(1)}) \oplus h \oplus m^{(1)}$
4	$E_{m^{(2)}}(h \oplus m^{(1)}) \oplus h \oplus m^{(1)} \oplus m^{(2)}$
5	$E_{h \oplus m^{(2)}}(m^{(1)}) \oplus m^{(1)}$
6	$E_{h \oplus m^{(2)}}(m^{(1)}) \oplus m^{(1)} \oplus m^{(2)}$
7	$E_{h \oplus m^{(2)}}(m^{(1)}) \oplus h \oplus m^{(1)}$
8	$E_{h \oplus m^{(2)}}(m^{(1)}) \oplus h \oplus m^{(1)} \oplus m^{(2)}$
9	$E_{h \oplus m^{(2)}}(h \oplus m^{(1)}) \oplus m^{(1)}$
10	$E_{h \oplus m^{(2)}}(h \oplus m^{(1)}) \oplus m^{(1)} \oplus m^{(2)}$
11	$E_{h \oplus m^{(2)}}(h \oplus m^{(1)}) \oplus h \oplus m^{(1)}$
12	$E_{h \oplus m^{(2)}}(h \oplus m^{(1)}) \oplus h \oplus m^{(1)} \oplus m^{(2)}$

Table 2. Members of 12 subsets of $\mathcal{H}_{\mathrm{spc}}$ (non c-equivalent compression functions).

A manual analysis (see bellow) revealed that none of these hash function is collision resistant. Hence, all hash functions from $\mathcal{H}_{\mathrm{spc}}$ are not collision resistant.

For each f_i/H_i we denote by $\tilde{c}$ an expression we get from the "c" parameter by deleting $m^{(1)}$ and changing h to h_0 (where applicable). So, $m^{(1)} \oplus m^{(2)}$ becomes $m^{(2)}$, $h \oplus m^{(1)} \oplus m^{(2)}$ becomes $h_0 \oplus m^{(2)}$, etc.

Functions H_1 – H_4. An adversary constructs messages consisting of four n-bit blocks $\langle x, y, w, z\rangle$ as follows:

1. y is chosen arbitrarily
2. $x = E_y^{-1}(h_0) \oplus h_0$
3. $w = \tilde{c}$
4. $z = y$

Functions H_5 – H_8. An adversary constructs messages consisting of four n-bit blocks $\langle x, y, w, z\rangle$ as follows:

1. y is chosen arbitrarily
2. $x = E_{h_0 \oplus y}^{-1}(h_0)$
3. $w = x$
4. $z = h_0 \oplus h_1 \oplus y$ (h_1 is an intermediate hash value, computed as $f_i(h_0, (x, y))$)

Functions $H_9 - H_{12}$. An adversary constructs messages consisting of four n-bit blocks $\langle x, y, w, z\rangle$ as follows:

1. y is chosen arbitrarily
2. $x = E^{-1}_{h_0 \oplus y}(h_0) \oplus h_0$
3. $w = \tilde{c}$
4. $z = h_0 \oplus h_1 \oplus y$

The resulting hash values of the messages produced by the adversary are in Table 3.

i	$H_i(\langle x, y, w, z\rangle)$	i	$H_i(\langle x, y, w, z\rangle)$
1	h_0	7	h_0
2	h_0	8	h_1
3	h_1	9	h_0
4	h_1	10	h_1
5	h_1	11	h_1
6	h_0	12	h_0

Table 3. Hash values of $H_i(\langle x, y, w, z\rangle)$.

In the case $H_i(\langle x, y, w, z\rangle) = h_0$, the adversary can produce collisions by varying the value y (it can be chosen arbitrarily), and computing x, w, z according to the given procedure.

In the case $H_i(\langle x, y, w, z\rangle) = h_1$, the adversary can produce collisions of different lengths, i.e. $H_i(\langle x, y\rangle) = H_i(\langle x, y, w, z\rangle) = h_1$. The adversary converts this to regular (equal length) collisions as follows. She (he) repeats this procedure for a new initial vector $h'_0 = h_1$, and finds $\langle x', y', w', z'\rangle$ such that $H_i(\langle x', y'\rangle) = H_i(\langle x', y', w', z'\rangle) = h'_1$ (starting with h'_0). Combining these messages we get desired equal length collisions (see Figure 1):

$$H_i(\langle x, y, w, z, x', y'\rangle) = H_i(\langle x, y, x', y', w', z'\rangle).$$

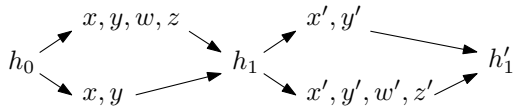


Fig. 1. Producing equal length collisions.

4 Conclusion

We have analyzed rate-2 PGV-like hash functions in terms of collision resistance. We have shown that none of these 512 hash functions is collision resistant. This answers an open problem stated in [2]. Moreover, we have shown that (very weak) adversary with constant complexity is sufficient to produce collisions in any of rate-2 PGV-like hash functions.

Acknowledgments. The first author was partially supported by APVT grant No. 20-023302, and VEGA grant No. 1/9176/02. The second author was supported by UK grant No. 401/2006, and VEGA grant No. 1/3106/06.

References

1. Black J., Rogaway P., Shrimpton T., Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In Advances in Cryptology – CRYPTO '02, LNCS 2442, Springer-Verlag, 2002
2. Ostertág R., Stanek M., Attacking PGV-like Rate-2 Cryptographic Compression Functions. In ITAT 2005: Information Technologies – Applications and Theory, Pavol Jozef Šafárik University, 2005, 123–131
3. Preneel B., Govaerts R., Vandewalle J., Hash Functions Based on Block Ciphers: A Synthetic Approach. In Advances in Cryptology – CRYPTO '93, LNCS 773, Springer-Verlag, 1994
4. Stanek M., Analysis of Fast Blockcipher-Based Hash Functions. In Computational Science and Its Applications – ICCSA 2006, LNCS 3982, Springer-Verlag, 2006, 416–425

Sdílení dat v prostředí s nehomogenními skupinami uživatelů*

Roman Špánek[12] and Miroslav Tůma[1]

[1] Ústav informatiky Akademie věd ČR,
Pod Vodárenskou věží 2, 182 07 Praha 8, Česká Republika
spanek@cs.cas.cz
WWW home page: http:www.cs.cas.cz/~spanek/, http:www.cs.cas.cz/~tuma/
[2] Technická univerzita v Libereci, Hálkova 6, Česká Republika
roman.spanek@vslib.cz

Abstrakt *Článek představuje možné řešení problému bezpečnosti v různých prostředích, která kladou důraz na sdílení zdrojů mezi jednotlivci nebo skupinami uživatelů. Mezi taková prostředí můžeme například zařadit mobilní telekomunikace a sebou nesoucí pojem mobilních databází, superpočítačů tvořených na bázi gridů, peer-to-peer sítí, vizi sémantického webu a v neposlední řadě i technologie počítačových agentů. Všechna tato prostředí mají svá specifika, ale také mají řadu společných jmenovatelů. Naše řešení je založeno na využití virtuálních organizací, které lze definovat jako dynamicky vytvářené skupiny uživatelů a organizací sdílející přístup k počítačům, softwaru, datům a ostatním zdrojům s přesným řízením přístupu a jasnou definicí co, s kým a za jakých podmínek je sdíleno. Náš přístup využívá matematického aparátu hypergrafů. Vzhledem s různorodosti a rozsahu cílových prostředí je naším cílem návrh bezpečnostního modelu, který bude mít schopnost samostatného vývoje, bez toho, že by struktura virtuální organizace degenerovala. Cílem článku je podat přesný rozbor jednoho z hlavních problémů v decentralizovaných prostředích a to nalezení konsenzu mezi uživateli virtuální organizace, který je v našem případě představován volbou vedoucího člena virtuální organizace.*

1 Úvod

S příchodem nových technologií umožňující připojení uživatelů k počítačové síti prakticky kdekoli a kdykoli, společně se souvisejícím nárůstem počtu uživatelů, vyvstala nutnost řešit otázky zabezpečení. Jedním z možných řešení je aplikace velmi odolných šifrovacích algoritmů. Tyto algoritmy řeší pouze zabezpečení komunikace. Proto je také nutné řešit otázky důvěry mezi skupinami, případně jednotlivými uživateli. Jedno z možných řešení je využití bezpečnostních modelů navržených pro prostředí Virtuálních organizací. Virtuální organizace (VO) jsou dynamicky vytvářené skupiny uživatelů a organizací sdílející přístup k počítačům, softwaru, datům a ostatním zdrojům s přesným řízením přístupu a jasnou definicí co je sdíleno, kým je sdíleno a za jakých podmínek je sdíleno. Model VO je využíván v peer-to-peer sítích, mobilních databázích, sémantickém webu a v neposlední řadě i superpočítačích vytvořených na bázi gridů, pro které byl model původně navržen. Široké spektrum aplikací poukazuje na použitelnost takového řešení.

Náš příspěvek navazuje na bezpečnostní model navržený v [2], který klade důraz na možnost automatického vývoje a správy virtuální organizace. Takový přístup je vhodný v prostředích, kde může být velký počet nehomogeních uživatelů. Jako příklad lze uvést počítačové agenty v prostředí ah hoc sítí nebo sémantického webu. V takovýchto prostředích je nutné mít dostatečně robustní řešení pro správu uživatelů, které bude pracovat co možná nejvíce samostatně bez toho, aby organizace degenerovala nebo ztrácela vlastnost důvěryhodnosti. Degenerací budeme myslet vývoj takové VO k jednomu z limitních stavů:

1. jedné VO obsahující všechny uživatele
2. mnoha velmi malých VO

Postup v [2] kombinuje matematický model založený na hypergrafech s vhodnou implementací umožňující nasazení v distribuovaném prostředí. Pro ověření navržených postupů byla napsána experimentální aplikace SECGRID v jazyce ANSI C.

Zbytek příspěvku je organizován následovně: odstavec 2 shrnuje současný stav problematiky bezpečnosti v prostředí virtuálních organizací. Naše konkrétní implemetace je popsána v odstavci 3.1 a volba vedoucího skupiny je uvedena v odstavci 3.2. Příspěvek je shrnut závěrem.

2 Současný stav problematiky

Pojem virtuálních organizací [3] byl zaveden v prostředí gridů [4]. Gridy jsou rozsáhlé distribuované systémy,

* Práce byla podpořena projektem 1ET100300419 programu Informační společnost (Tématického programu II Národního programu výzkumu v ČR: Inteligentní modely, algoritmy, metody a nástroje pro vytváření sémantického webu), výzkumným záměrem AV0Z10300504 „Informatika pro informační společnost: Modely, algoritmy, aplikace“ a výzkumným centrem: Pokročilé sanační technologie a procesy 1M4674788502, Ministerstva Ministerstvo školství, mládeže a tělovýchovy České Republiky.

tvořené heterogenními výpočetními, datovými a informačními zdroji, propojenými počítačovou sítí, tak aby mohly být využívány jako řešení velmi výpočetně nebo prostorově náročných problémů. Takto propojené zdroje mohou být, a také často jsou, alokovány i velmi daleko od sebe. Velká vzdálenost a také různorodost propojených zdrojů jsou hlavními rozdíly mezi gridy a clustery. V případě gridů je navíc velmi komplikovanou otázkou vyřešení správy přístupu jednotlivých uživatelů. Vzhledem ke geografické různorodosti zdrojů gridu, je velmi moudré předpokládat i stejnou různorodost v případě uživatelů. Tato různorodost bude jistě komplikovat řešení oprávnění přístupu ke gridu, zejména tím, že různé organizace zapojené do gridu mohou mít různá řešení vlastního zabezpečení, různá nastavení přístupových práv a zejména různé způsoby ověřování vlastních uživatelů. Na druhou stranu je nutné, aby uživatel nebyl nucen stále zadávat hesla, případně další osobní data, při připojení k jinému zdroji. Jako další požadavek lze vysledovat možnost delegovat část, případně všechna uživatelova práva na třetí subjekt, tak aby mohl provádět úkoly svěřené uživatelem a tedy měl i přístup ke zdrojům na základě uživatelových práv.

Jako jedno z vhodných řešení se ukazuje vytvoření virtuálních organizací. Virtuální organizace je v mnohých aspektech velmi podobná skutečným organizacím. Jedním z hlavních důvodů vytváření VO je poskytovat prostředky pro správu a vytváření důvěry mezi jejími členy. Postupy pro vytváření důvěry v takovém prostředí lze rozdělit na *Policy based* a *Reputation based* přístupy.

Policy based přístup byl navržen pro distribuované architektury služeb [5],[6],[7],[8],[9] a také v kontextu s gridy [10], jako řešení problému autorizace a řízení přístupu. Motivací takového přístupu je vybudovat systém pravidel a postupů pro vytváření a rozhodování o důvěře jednotlivých uživatelů. K tomuto cíli je využíváno jazyků s dobře definovanou sémantikou. Rozhodnutí o důvěře se pak provádí na základě nepřímých atributů uživatele (např. adresa nebo věk), které jsou certifikovány důvěryhodnou třetí stranou.

Reputation based postupy jsou velmi vhodné pro prostředí elektronických komerčních systémů (např. eBay), v peer-to-peer systémech, mobilních databázích a poslední době i pro prostředí sémantického webu [11], [12]. Charakteristikou takového přístupu je odvozování důvěry uživatele na základě jeho chování v minulosti. Důvěra je tedy založena na doporučeních a zkušenostech ostatních členů skupiny [13], [14], [15], [16].

Společným jmenovatelem všech výše zmíněných postupů je skutečnost, že uživatelé jsou do VO vloženy jistou autoritou (např. administrátorem). Toto řešení však nemusí být nejvhodnější v případě, že vezmeme v potaz prostředí s velkým počtem různorodých uživatelů (typicky sémantický web nebo mobilní databáze). Vezmeme-li v potaz takováto prostředí, je vhodné mít nástroj pro automatické vytváření a správu VO.

3 SecGRID

Úkolem SecGRID je umožnit automatické vytváření a správu VO v prostředích s velkým počtem nehomogenních uživatelů. Model SecGRID je založen na matematickém aparátu hypergrafů, který mu poskytuje dostatečně silné protředky pro jeho realizaci a ověření. VO je v SecGRID reprezentována jako ohodnocená hypergrafová struktura. Vztahy mezi členy jsou reprezentovány pomocí váhy ohodnocené hyperhrany. Vyšší ohodnocení hrany implikuje vyšší důvěru mezi členy. Uzly reprezentují jednotlivé uživatele. Ohodnocení uzlu reprezentuje jeho důvěryhodnost, dostupné výpočetní a komunikační prostředky.

3.1 Implementace

Struktura VO v SecGRID je hierarchická. Je tvořena libovolným počtem menších skupin uživatelů, které se dále dělí na menší organizační jednotky. Vzhledem ke skutečnosti, že implementace nerozlišuje mezi VO a jejími dílčími organizacemi, budeme dále používat jen termínu VO. Spodní vrstva hierarchie je tvořena vlastními členy VO. Každá VO má zvoleného vedoucího skupiny, tzv. VO Leader (VOL). Nad touto vrstvou uživatelů je jedna nebo více vrstev tvořených pouze VOL. VOL odpovídá za podřízené jednotky a umožňuje komunikaci mezi jednotlivými VO. Tato struktura zvyšuje důvěryhodnost a bezpečnost SecGRID, neboť právě komunikace členů z jedné VO do jiné představuje největší bezpečnostní riziko pro ostatní členy. Tím že komunikace mezi skupinami je kontrolována VOL je zaručeno, že nedojde k úniku citlivých informací.

Pro potřeby experimentální aplikace bylo použito hypergrafů s mohutností incidence hyperhran dvě. Rozšíření aplikace na plnou kardinalitu hyperhran je evidentní.

Vzhledem k požadavku automatického vytváření a správy VO bylo nutné navrhnout řadu pravidel pro ohodnocování hran. Byly vysledovány tři základní varianty, které mohou při vývoji grafové struktury nastat:

1. přidání tranzitivní hrany,
2. vytvoření neorientovaného cyklu,
3. vytvoření orientovaného cyklu.

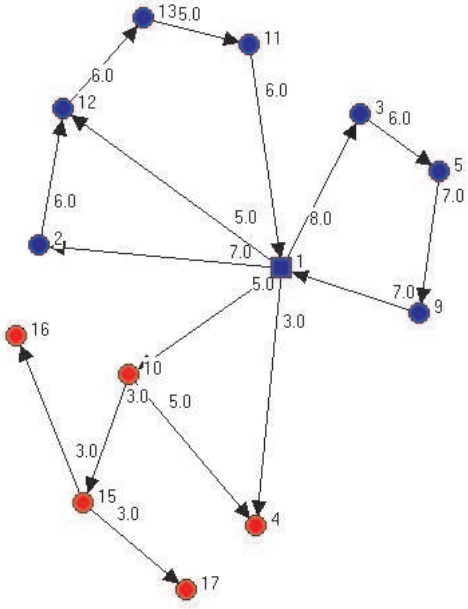
Nejdůležitější z nich je vytvoření nového orientovaného cyklu (souvislé komponenty v grafu). Tento případ je reprezentován jako vznik uzavřené skupiny uživatelů,

kde lze komunikovat mezi všemi členy. Proto je taková souvislá komponenta vhodným kandidátem na vytvoření nové VO. Nová VO je vytvořena na základě ohodnocení hran v komponentě. Dojde-li k vytvoření nové VO, je nutné pro ni zvolit nového vedoucího skupiny (VOL). Ostatní přípustné varianty nejsou pro příspěvek zajímavé, neboť nevyžadují vytvoření nové skupiny (VO) a tedy nevyžadují volbu VOL, která je hlavním tématem příspěvku. Podrobný popis ostatních případů, včetně všech ohodnocovacích pravidel lze nalézt v [2].

3.2 Volba VOL

Z předchozího odstavce je patrné, jak důležitou roli hrají VOL. Z toho důvodu je nutné mít vhodně vyřešené volení VOL ze členů skupiny a to tak, aby bylo splněno následující:

1. nový VOL musí být bezpodmínečně velmi důvěryhodným členem skupiny
2. volba VOL nesmí příliš zatěžovat členy VO
3. volba VOL musí být implementovatelná v distribuovaném prostředí



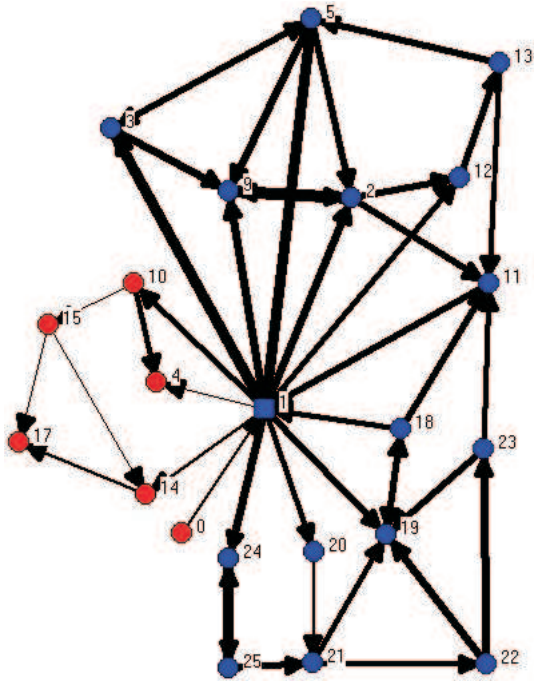
Obrázek 1. Výchozí stav VO.

Choulostivá operace volby nového VOL je řešena v SecGRID pomocí následující procedury:

- jako první VOL je zvolen zakládající člen VO
- v případě nutnosti zvolit nového VOL je do skupiny vyslána RESIGN zpráva původním VOL
- RESIGN zpráva je odeslána sousedovi s nejlepším vztahem (po hraně s největším ohodnocením)
- při přijetí RESIGN zprávy se příjemce rozhodne, zda-li bude novým VOL
- pokud ano, oznámí to skupině pomocí NEWVOLARRIVES zprávy
- v opačném případě předá RESIGN zprávu opět svému sousedu s nímž má nejlepší vztah
- v momentě, kdy kterýkoli člen skupiny, mimo odcházejícího VOL, obdrží RESIGN zprávu podruhé, stává se automaticky novým VOL

Procedura pro volbu nového VOL má všechny požadované vlastnosti, viz. požadavky výše.

- ad 1. Důvěryhodnost nového VOL je zaručena, neboť k jeho volbě jsou přizváni pouze členové *základní skupiny* (viz. níže) uživatelů
- ad 2. volba nového VOL není náročnou operací, neboť mimo starého VOL žádný ze členů základní skupiny uživatelů nemusí přeposlat RESIGN zprávu vícekrát než jednou. Počet členů základní skupiny je menší než počet všech členů
- ad 3. celá procedura využívá systém zpráv, který lze přímo využít v distribuovaném prostředí.



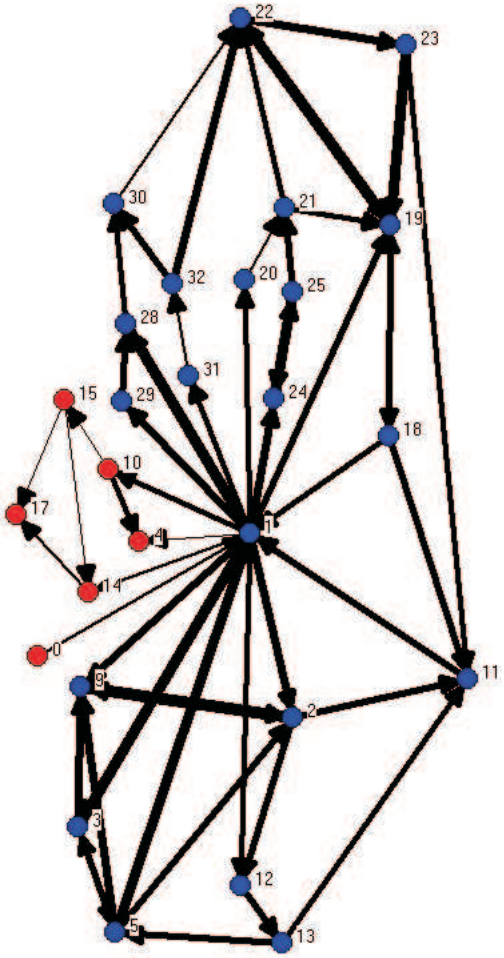
Obrázek 2. Stav VO po přidaní hran a uzlů.

Pojem *základní skupiny* uživatelů poukazuje na skutečnost, že při vytváření struktury VO dochází k vytvoření ustálené skupiny důvěryhodných uživatelů. Na

obrázku 1 je stav nově vytvořené VO. Členové VO jsou zobrazeny modře a VOL je naznačen čtverečkem. Síla hran odpovídá ohodnocení. Z obrázku je patrné, že základní skupina je tvořena členy {1,3,5,9}. Druhá dobře profilovaná skupina nemá takovou důvěryhodnost. Pokud by došlo k volbě v této konfiguraci, byl by nový VOL zvolen právě ze členů {3,5,9}. Uvažujme situaci, kdy byl jako nový VOL zvolen člen 5. Pokud po nějaké době došlo opět k volbě nového VOL, byl by znovu volen pouze ze členů {1,3,9}. Je tedy zřejmé, že možní kandidáti na VOL jsou alokováni pouze mezi členy základní skupiny uživatelů, která má menší počet členů, než celá organizace. Například počet členů skupiny na obrázku 1 je osm, nicméně nově volený VOL bude volen pouze ze skupiny tří členů.

Situace po přidání nových členů a provedení přehodnocení je na obrázku 2. Z obrázku je patrné, že nedošlo k dramatickému zvětšení základní skupiny uživatelů. Přestože značně narostl jak počet hran, tak i počet vrcholů, základní skupina uživatelů se rozrostla pouze o jednoho člena na 1,3,5,9,2. Vezmeme-li v porovnání stávající počet členů, který s zdvojnásobil na šestnáct a počet členů základní skupiny, je tento poměr 16/5. Přitom při původní konfiguraci na obrázku 1 byl tento poměr 8/4. Z toho je jasně patrné, že základní skupina podléhá pomalejšímu růstu. Tento poměr, jak ukazují naše simulace, se bude s přibývajícím počtem členů dále zvětšovat.

Tuto skutečnost lze vysvětlit vznikem izolovaných *hnízd*. Hnízdem budeme myslet skupinu uživatelů, majících k sobě navzájem velmi dobré vztahy a navíc tvořících souvislou komponentu s vysokým ohodnocením hran. Na obrázku 3 je taková struktura dobře patrná mezi členy 19, 22 a 23. Simulace ukazují, že s přibývajícím počtem členů takovýchto struktur uvnitř VO přibývá a navíc bývají alokována dále od VOL (vzhledem k délce orientované cesty v grafu). Skutečnost, že hnízda jsou izolována a nejsou v blízkosti VOL je snadné vysvětlit, neboť pokud by byla blízko VOL, stala by se součástí základní skupiny uživatelů. Vzhledem k patrné izolovanosti hnízd a k ohodnocení jeho hran je dobré se zamyslet, zda-li by nebylo lépe takové struktury úplně od VO oddělit a vytvořit z nich vlastní nové menší VO. Odpověď na tuto otázku nelze položit jednoznačně, neboť oddělení od zbytku VO by mělo za následek ztrátu spojení s dalšími členy skupiny a tedy izolovanost, která by ovšem mohla být na škodu členům a to jak hnízda tak i zbytku VO. Na druhou stranu je nutné podotknout, že jistá míra izolovanosti je již zachycena ve vlastním ohodnocení hran uvnitř hnízda. Sdílení informací je tedy daleko snazší mezi členy hnízda, než mezi zbytkem VO. Dalším silným argumentem pro nevytváření nové VO je skutečnost, že každá nově vytvořená organizace musí uchovávat informace o okolních strukturách (ve smyslu uložení dat o okolních VOL). Okolní skupiny pak musí ukládat informace o nově vznikajících skupinách. Pokud by tedy byly nové VO vytvářeny příliš rychle a s malým počtem členů, znamenalo by to značnou zátěž pro všechny zainteresované VOL. Druhou stranou mince je skutečnost, že příliš velké VO se špatně udržují. Proto náš model počítá s rozdělením VO na dílčí menší v momentě, kdy bude splněna podmínka $d(k) > \delta$, kde $d(k)$ je průměr VO a δ je celé kladné číslo.



Obrázek 3. Vznik *hnízd* ve struktuře VO.

Dalším zajímavým aspektem procedury pro volbu nového VOL je skutečnost, že kandidátní VOL jsou alokovány vždy v blízkosti původních VOL. Tato skutečnost je důsledkem podmínky pro přeposlání RESIGN zprávy, aby příjemce měl s odesílatelem nejdůvěrnější vztah. Tedy aby hrana mezi odesílatelem a příjemcem měla nejvyšší ohodnocení a tedy byla mezi členy základní skupiny.

Algoritmus volby VOL zvolí nového VOL při složitosti $O(n)$, kde n je délka orientovaného cyklu v hypergrafu představujícího VO. Vzhledem ke skutečnosti, že vytvoření nové VO je podmíněno vznikem orien-

tovaného cyklu je tedy zaručeno, že algoritmus vždy skončí zvolením nového VOL. Složitost je ve skutečnosti nižší vzhledem k vytvoření základní skupiny uživatelů.

4 Závěr

Cílem příspěvku bylo navrhnout a experimentálně ověřit metodu pro dosažení konsenzu mezi členy virtuálních organizací. Celý příspěvek je začleněn do širšího problému návrhu bezpečnostního modelu pro prostředí virtuálních organizací, který bude použitelný v prostředích majících velký počet různorodých uživatelů. V takovýchto prostředích je nutné, aby měl model schopnost samostatného vývoje. Jedním z klíčových momentů v životě VO je volba vedoucího člena (VOL), který je zodpovědný za její správu a také za komunikaci mezi ostatními skupinami. Proto byl navržen postup jak dosáhnou shody mezi členy VO bez toho, aby tato volba měla dopad na efektivitu a použitelnost našeho modelu v reálném prostředí. Hlavní limitující faktory jsou požadavky, aby byl algoritmus volby přímo implementovatelný v distribuovaném prostředí a zachovával důvěru mezi členy VO. Naše experimenty ukazují, že navržený postup volby splňuje všechny požadavky na něj kladené. Pro ověření korektnosti byla použita experimentální aplikace SecGRID. Na základě provedených experimentů, byla dále ukázána celá řada zajímavých momentů ve vývoji VO, které mají klíčový dopad především na reálné využití našeho modelu.

Reference

1. Clarke F., Ekeland I., Nonlinear Oscillations and Boundary-Value Problems for Hamiltonian Systems. Arch. Rat. Mech. Anal. 78, 1982, 315–333
2. Špánek R., Tůma M., Secure Grid-based Computing with Social-Network Based Trust Management in the Semantic Web. submitted to NNW., 2006, 15str.
3. Foster I., Kesselman C. and S. Tuecke, The Anatomy of the Grid. Enabling Scalable Virtual Organizations. International Journal of Supercomputer Applications, 2001
4. Foster I., Kasselman C., The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, 1999
5. Bonatti P. and Samarati P., Regulating Service Access and Information Release on the Web. In CCS '00: Proceedings of the 7th ACM Conference on Computer and Communications Security, ACM Press, 2000, 134-–143
6. Li N. and Mitchell J., A Role-Based Trust-Management Framework. In DARPA Information Survivability Conference and Exposition (DISCEX), Washington, D.C., Apr. 2003
7. Gavriloaie R., Nejdl W., Olmedilla D., Seamons K.E., and Winslett M., No Registration Needed: How to Use Declarative Policies and Negotiation to Access Sensitive Resources on the Semantic Web. In 1st European Semantic Web Symposium (ESWS 2004), Lecture Notes in Computer Science, Springer, Heraklion, Crete, Greece, Vol. 3053, May 2004, 342–356
8. Becker M.Y. and Sewell P., Cassandra: Distributed Access Control Policies with Tunable Expressiveness. In 5th IEEE International Workshop on Policies for Distributed Systems and Networks, Yorktown Heights, June 2004
9. Bonatti P.A. and Olmedilla D., Driving and Monitoring Provisional Trust Negotiation with Metapolicies. In 6th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2005), Stockholm, Sweden, IEEE Computer Society, June 2005, 14-–23
10. Basney J., Nejdl W., Olmedilla D., Welch V., and Winslett M., Negotiating Trust on the Grid. In 2nd WWW Workshop on Semantics in P2P and Grid Computing, New York, USA, May 2004
11. Kagal L., Finin T., and Joshi A., A Policy Based Approach to Security for the Semantic Web. In Proceedings of the 2nd International Semantic Web Conference, Sanibel Island, Florida, USA, Oct. 2003
12. Tonti G., Bradshaw J.M., Jeffers R., Montanari R., Suri N., and Uszok A., Semantic Web Languages for Policy Representation and Reasoning: A Comparison of KAoS, Rei and Ponder. In Proceedings of the 2nd International Semantic Web Conference, Sanibel Island, Florida, USA, Oct. 2003
13. Aberer K. and Despotovic Z., Managing Trust in a Peer-2-Peer Information System. In Proceedings of 10th International Conference on Information and Knowledge Management, 2001, 310–317
14. Damiani E., di Vimercati S.D.C., Paraboschi S., Samarati P., and Violante F., A Reputation-Based Approach for Choosing Reliable Resources in Peer-to-Peer Networks. In Proceedings of ACM Conference on Computer and Communications Security, 2002, 202–216
15. Kamvar S.D., Schlosser M.T., and Garcia-Molina H., Eigenrep: Reputation Management in p2p Networks. In Proceedings of 12th International WWW Conference, 2003, 640–651
16. Duma C., Shahmehri N., and Caronni G., Dynamic Trust Metrics for Peer-to-Ppeer Systems. In Proceedings of 2nd IEEE Workshop on P2P Data Management, Security and Trust (in connection with DEXA'05), August 2005

ITAT'06

Information Technology – Applications and Theory

STUDENT PAPERS

Level-of-detail pro umělou inteligenci: Je tato technika přínosná?

Cyril Brom[1]

Karlova Universita, Matematicko-fyzikální fakulta, Malostranské nám. 2/25, Praha, Česká Republika
brom@ksvi.mff.cuni.cz

Abstrakt *Virtuální bytosti jsou umělé organismy, které obývají virtuální prostředí modelující přirozený svět. Pokud je svět veliký, nedá se modelovat ve své celistvosti díky omezeným výpočetním zdrojům. Nabízí se ovšem použít techniku level-of-detail pro automatické zjednodušení simulace na místech, která v danou chvíli nejsou podstatná. Tato technika byla robustním a teoreticky podloženým způsobem implementována jako součást projektu IVE (MFF UK, 2005). Předmětem článku je studie použití level-of-detail v IVE. Studie ukazuje, že pro rozsáhlé světy přináší způsob, jakým byla technika v IVE pojata, významnou přidanou hodnotu oproti variantám techniky běžně používaným pro zjednodušování simulace v počítačových hrách.*

1 Úvod

IVE je softwarový nástroj pro tvorbu virtuálních světů modelujících přirozený lidský svět na té úrovni, na jaké ho my lidé běžně vnímáme a chápeme. Stěžejní součástí virtuálních světů jsou enti – virtuální lidé nebo zvířata – behaviorální modely živých bytostí. IVE bylo vyvinuto studenty MFF UK v roce 2005 a v současné době je volně k disposici verze 1.1 [12]. Základní motivací bylo vytvořit platformu pro výzkum především na polích kognitivních věd a umělé inteligence pro počítačové hry a virtuální vyprávění („virtual storytelling“).

Akronym IVE znamená *inteligentní virtuální prostředí* („intelligent virtual environment“). „Inteligence“ IVE spočívá ve dvou bodech, které zároveň z technického hlediska představovaly cíle projektu:

1. Chování postav je representováno distribuovaným způsobem v prostředí, nikoli v „hlavách“ entů. Prostředí tak umí samo „inteligentně“ enty navigovat. To má tu podstatnou výhodu, že jde do prostředí přidávat nové prvky – objekty a akce – aniž by se s nimi museli enti učit zacházet. To výrazně usnadňuje design aplikace.
2. Simulace je automaticky „inteligentně“ zjednodušována v místech, která nesleduje uživatel nebo kde se obecně nic podstatného neděje. To umožňuje simulovat rozsáhlé virtuální světy, na něž je především IVE zaměřeno. Způsob zjednodušování je de facto použitím techniky level-of-detail (dále též LOD) na úrovni řízení postav.

Vytváření podobných simulací je nezávisle na těchto dvou bodech obecně netriviální, neboť:

- postavy se musí chovat věrohodně, to jest navenek projevovat chování, jež se jeví rozeznatelně jako lidské (nebo zvířecí),
- postavy mohou *interagovat* mezi sebou navzájem,
- postavy jednají v prostředí, které je *dynamické*, *nepředvídatelné* a pouze *částečně pozorovatelné* (podle [13], str. 46),
- prostředí je částečně *interaktivní* (uživatel není vtělen prostřednictvím avatara, ale může měnit stav světa – přesunovat objekty, měnit jejich vlastnosti apod.).

K řešení dvou cílů a s ohledem na uvedené obecné problémy bylo v IVE použito několik technik a jejich rozšíření. Kvůli požadavkům na prostředí a interaktivitu je pro řízení entů použit mechanismus vycházející z Bratmanova myšlenkového aparátu *praktického rozhodování* [1] (později přetaveného do architektury BDI), jenž používá reaktivních rozhodovacích pravidel. Kvůli rychlosti je ohodnocování pravidel prováděno pomocí on-line varianty algoritmu Rete [9], známého z oblasti expertních systémů. Kvůli prvnímu cíli jsou pravidla representována v prostředí způsobem vycházejícím z percepční *teorie afordancí* psychologa Gibsona [10]. S ohledem na druhý cíl však byla tato teorie rozšířena hierarchickým způsobem. Pro koordinaci více entů na jednom úkolu byla navíc implementována technika *předávání rolí*, která umožňuje to, že řízení jednotlivých entů může být dočasně delegováno na centrální mechanismus zajišťující koordinaci.

Většina vyjmenovaných technik již někdy v oblasti virtuálních lidí, počítačových her či kognitivních věd byla teoreticky popsána, či dokonce implementována – pokud ale vím, vždy samostatně. Abstraktní architektura IVE všechny techniky robustním způsobem spojuje a IVE je všechny implementuje. IVE bylo obecně popsáno v [3] a [6], level-of-detail v [15] a distribuovaná representace v [5]. IVE hodláme v budoucnu využít jako framework pro simulaci virtuální firmy [4] a pro projekt týkající se virtuálního vyprávění [7].

Jak bylo řečeno, jedním z důvodů, proč se IVE vytvářelo, bylo nasazení techniky LOD v oblasti řízení postav – narozdíl od jejího běžného použití v počítačové grafice. Za tímto cílem stál implicitní předpoklad, že použití techniky simulace zrychlí, přičemž ale zároveň dění neztratí na věrohodnosti. To například znamená, že pokud by uživatel náhle přepnul pohled do takové

oblasti virtuálního světa, která zatím simulována nebyla, IVE by mělo okamžitě začít dění v dané oblasti simulovat na maximální úrovni detailu a uživatel by to neměl zaregistrovat. Do jaké míry se podařilo splnit toto očekávání?

Tento článek předkládá studii LOD v IVE na jednom konkrétním prototypovém virtuálním světě. Studie má za cíl objasnit klíčovou otázku: *je použitá level-of-detail přínosná a má smysl v jejím zkoumání pokračovat?* Na IVE jako takovém je toho totiž k budoucímu výzkumu více a LOD představuje pouze jeden možný směr.

Studie má dvě části. První sestává ze sady měření výkonnosti IVE při simulaci světa na různých úrovních detailu a přechodech mezi nimi. Vzhledem ke zkoumanému prostředí a povaze otázky, na niž hledáme odpověď, nedává smysl předkládat rigorózní statistickou analýzu, místo toho předkládáme sérii naměřených dat a jejich interpretaci. Druhá sestává z pozorování konkrétních situací, jež ve virtuálním světě nastávají.

V článku nejprve krátce vysvětlím, jakým způsobem v IVE technika LOD funguje a proč funguje právě tímto způsobem. Dále zmíním podobné práce, na kterých uvidíme, že způsob fungování level-of-detail je skutečně ojedinělý. Poté představím výsledky samotné studie.

2 Jak funguje LOD v IVE

Aby bylo možné hovořit o studiích a měřeních provedených na IVE, je třeba vysvětlit, jakým způsobem LOD technika v IVE funguje. Tomu se věnuje tato sekce.

2.1 Požadavky a předpoklady

Při návrhu IVE jsme vycházeli z následujících požadavků a předpokladů:

- Nástroj má sloužit pro simulace velikých virtuálních světů. Takové světy obsahují desítky oblastí a desítky virtuálních postav.
- Nástroj by teoreticky měl umožňovat interakci několika uživatelů zároveň.
- Nepředpokládá se, že by uživatelé koncových aplikací mohli nahlížet do různých oblastí, jak se jim zamane. Budou moci nahlížet jen na místo, kde se nachází jimi ovládaná postava, a ta se z oblasti může přesunout pouze do některé ze sousedních oblastí.
- Přechod postavy do oblasti, která zatím nemá být simulována detailně, a s tím spojené zvýšení úrovně detailu v dané oblasti, nesmí být výpočetně náročné.
- Ve světě se může odehrávat nějaký příběh, jehož dílčí události mohou být významné. Takové musí být simulovány detailně, i když je zrovna žádný uživatel nesleduje.
- Povolujeme, že výsledek dění v určitém místě může být jiný při simulaci na vysoké a nízké úrovni detailu (neboť to, co je simulováno na nízké úrovni detailu, je z definice nepodstatné, a tudíž na výsledku tolik nezáleží).

S ohledem na tyto body jsme měli na techniku LOD následující požadavky:

1. LOD musí umožňovat detailní simulaci více míst, a to i těch, kde se nenachází žádná postava uživatele.
2. Místa v okolí postavy uživatele musí být částečně simulována, a tím předpřipravena na to, že do nich uživatel může vstoupit.
3. Místa právě opuštěná uživatelem musí být ještě po určitou dobu simulována na nejvyšší úrovni detailu, protože uživatel by se mohl vrátit.
4. Z hlediska programátora virtuálního světa musí existovat prostředek proto, aby bylo možné kdykoli změnit aktuální úroveň detailu v libovolné oblasti.

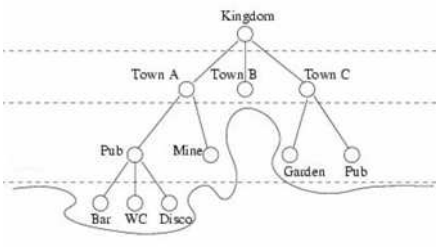
2.2 Řešení

Řešení požadavků ze sekce 2.1 spočívalo v navržení specielní hierarchické architektury pro representaci virtuálního světa a úkolů, které v něm lze provádět. Každá další hladina hierarchie představuje komplexnější úroveň popisu. Representaci říkáme ISMA (*intence – vhodnost – činnost – rada*).

ISMA pracuje se dvěma dekomposicemi – „fyzického“ světa a chování. Virtuální prostor je representován vrstvenými multigrafy jako síť oblastí, které se rozpadají na podoblasti a tak dále, až k nedělitelným místům. Místem je například prostor před lednicí, nadřazenou oblastí je kuchyň, ještě vyšší oblastí dům atd.

Analogicky je representováno chování. Každý ent má hlavní cíle, jichž může dosáhnout prostřednictvím činností, které se mohou rozpadnout na podcíle, jež mohou být dosaženy podčinnostmi, a tak dále až k nedělitelným akcím. Hlavním cílem je například najíst se, podcílem například najít jídlo, atomickou akcí například kousnout si chleba. Technicky je tento hierarchický rozklad dán pomocí rozhodovacích pravidel.

V rámci IVE byly navržena komponenta, tzv. *LOD manager*, která během simulace přiřazuje jednotlivým oblastem *úroveň detailu*. Toto přiřazení popisuje jakousi „membránu“ proloženou prostorovou hierarchií, přičemž všechny lokace těsně nad ní existují jako atomární body v prostoru a vše pod ní neexistuje – viz obr. 1.



Obrázek 1. Ilustrace level-of-detail v IVE.

Každý objekt má přiřazeny dvě hodnoty – *úroveň existence* a *úroveň pohledu*. V kostce řečeno, první říká, při jaké úrovni detailu objekt začíná existovat, druhý říká, jakou úroveň si vynucuje, pokud už existovat začne. Pokud tedy například existuje vesnice pouze jako atomární místo, nemusí existovat půllitry v hospodě. Ty mohou vzniknout například až tehdy, když se vesnice „rozpadne" na podoblasti, z nichž jedna bude hospoda. Analogicky nemusí při nízké úrovni detailu existovat ani enti. Pomocí LOD manageru je možné přidávat do světa specielní neviditelné objekty – *zarážky* – které toliko vynucují určitou úroveň detailu v dané oblasti.

Stěžejní je, že každá hladina prostorové hierarchie koresponduje s určitou úrovní v hierarchii chování. Ty činnosti, které se v hierarchii chování nachází na úrovni, jež koresponduje s tou úrovní prostorové hierarchie, která je právě „těsně nad membránou", se v daném okamžiku budou provádět jako atomární – tedy nebudou se k nim dohledávat podcíle. Pokud například bude hospoda existovat atomárně, mohou se v ní enti atomárně bavit, výsledkem čehož může být, že v hospodě ubude pivo. Teprve když uživatel nahlédne do hospody, úroveň detailu se zvýší na maximální a enti začnou provádět akce v plném detailu – tedy pít pivo, chodit na toaletu a podobně.

Pokud je detail v určité oblasti zbytečně veliký, nic se neděje, dokud není třeba výpočetní prostředky jinde. Pak teprve dojde ke snížení úrovně.

Podrobnosti k algoritmu změny úrovně detailu jsou popsány v [15].

Klíčová otázka pro průzkum není, jestli aplikace skutečně běží rychleji, pokud je někde detail snížen (pokud by tomu tak nebylo, postrádala by celá záležitost smysl). Stěžejní otázky se týkají režie práce LOD manageru při změně úrovně a snižováním věrohodnosti simulace při nižších úrovních detailu.

3 Příbuzné práce

Existuje nepřeberné množství prací týkajících se použití level-of-detail pro počítačovou grafiku, ovšem relativně málo prací týkající se LOD pro umělou inteligenci. Zřejmě zatím jedinými oblastmi, kde je možné ad hoc aplikace této techniky vysledovat, jsou počítačové hry a simulace rozsáhlých světů ve virtuální realitě. Typicky lze vidět dva přístupy. První lze označit jako „vidím – nevidím", kdy je simulováno v plném detailu právě místo pozorované uživatelem (někdy může být takových míst více, pokud je více uživatelů). Druhý můžeme nazvat „všechno – nebo nic". Zde cokoli, co není simulováno v plném detailu, není simulováno vůbec. Často, ale ne vždy, jsou tyto přístupy kombinovány.

Žádný z přístupů obecně nelze použít, pokud se v aplikaci odvíjí nějaký příběh. Příběh totiž vyžaduje často simulovat dění i mimo výhled uživatele, alespoň částečně, jinak dochází k dějovým nekonzistencím. Přístup „všechno – nebo nic" navíc obecně přináší problémy se zpomalením simulace v okamžicích, kdy určité místo začínáme simulovat (a tedy alokujeme všechny objekty v dosud nesimulované části a podobně).

Triviální LOD pomocí kombinace výše zmíněných přístupů je popsán například v [11]. Metoda „vidím – nevidím" s postupným zjednodušováním je aplikována v [2]. Poměrně robustní řešení pomocí hierarchických konečných automatů popisuje Champandard [8] – jde však pouze o ideu, která, pokud vím, nebyla dále implementována. Jiný robustní přístup zaujímají v [14] pomocí techniky předávání rolí, kdy virtuální postavy „nehrají určité role", pokud jsou mimo výhled uživatele. Jedná se o variantu postupného zjednodušování v kombinaci s přístupem „vidím – nevidím". Robustní přístup k (non-preemptivnímu) přidělování procesorového času individuálním virtuálním bytostem je presentován v [16] – nicméně z pohledu celé simulace se zdá, že jde o přístup „všechno – nebo nic".

IVE narozdíl od uvedených řešení představuje LOD s pozvolným zjednodušováním simulace a s možností simulovat libovolné místo na libovolné úrovni detailu. Navíc se pozvolné zjednodušování týká nejen složitosti chování postav, ale i prostoru – narozdíl od většiny výše uvedených aplikací, kde prostor zůstává buď neměnný, nebo není simulován za horizontem výhledu vůbec. Level-of-detail technika v IVE je dále teoreticky podložena – architekturou BDI [1] a rozšířením teorie afordancí [10].

Poznamenávám ovšem, že pro určité typy aplikací, například [11], může být řešení v IVE zbytečně složité. Řešení v IVE se odvíjí od požadavků na ně kladených (viz kap. 2), které vyplynuly z toho, k čemu bylo IVE v konečném důsledku zamýšleno. Aplikace, které jsou určeny jinak, mohou vystačit se „slabší" variantou LOD.

4 Studie

V této sekci jsou popsány jednotlivé studie techniky LOD aplikované v IVE. V první části sekce se věnuji

prostému měření výkonnosti IVE, jak časové tak paměťové, v druhé části studiu určitých specifických situací.

Scénář. Všechna měření a studie se týkají prototypového světa IVE. Tento svět má pět úrovní detailu:

1. Na první úrovni detailu existuje pouze svět jako takový.
2. Na druhé úrovni existují 4 města.
3. Na třetí úrovni detailu se rozpadá každé město na 5 dolů, 4 sídliště a hospodu. Dále začínají existovat enti – v každém městě jich je 22 (převážně horníků).
4. Na čtvrté úrovni detailu se důl rozpadá na horní, střední a spodní část; hospoda na bar, toaletu, salónek a prostor před hospodou; sídliště na 3 domy a trávník. Začínají existovat předměty jako je semafor v dole a pípa či kasa v hospodě. V každém dole navíc vzniká vozík, který je z technického hlediska řízen stejně jako virtuální lidé – tedy je ent.
5. Na páté úrovni vznikají jednotlivá atomární místa. Typická oblast má řádově desítky míst, celá simulace řádově několik tisíc. Dále vznikají další objekty – půllitry, jukebox, uhlí.

Zkoumaný scénář je následující: V šest hodin ráno se budí první směna 10-ti entů (z každého města) a vyráží do práce. Druhá vyráží v deset hodin do baru. Po desáté hodině je při simulaci za plného detailu v aplikaci na stovku objektů a celkem 108 entů (vč. 20-ti vozíků), kteří jsou všichni dohromady řízeni přibližně 5000 rozhodovacími pravidly (jež jsou neustále ohodnocována ve smyslu algoritmu Rete).

Metodologické poznámky

1. S barem pracujeme především proto, že se jedná o typickou oblast případových studií podobných aplikací – v baru interaguje více aktorů, dění je bohaté.
2. Měříme zátěž v těch časových úsecích, „kde se něco zajímavého děje“.
3. Aplikace je tak rychlá, že za běžné rychlosti je zátěž procesoru pod chybou měření. Proto jsou všechna měření provedena při 30-ti násobné rychlosti s jednou výjimkou, jež bude zmíněna, a následně znormována na normální rychlost.
4. Zátěž procesoru uvádíme v jednotce p.b. (procentní bod). 1 p.b. znamená, že zátěž bylo 1% procesorového času (100% znamená maximální zátěž procesoru).

Platforma. Všechny studie byly provedeny na počítači s 3 GHz Intel Pentium 4 a 1 GB RAM pod OS Windows XP 2002, service pack 2. Byla použita verze IVE 1.1 běžící v Javě 2 (standard edition, build 1.5.0_03-b07). Při všech měřeních nebylo dění v žádné části světa zobrazováno v grafickém rozhraní.

4.1 Měření

Test 1. Běžná rychlost. Smyslem testu bylo odhalit základní možnosti IVE co se týče rychlosti.

Popis testu. Byly provedeny celkem 4 různé testy; rychlosti simulace v 6:05 – 6:09 (A) respektive 10:15 - 10:19 (B) při celém světě simulovaném na úrovni 4 respektive 5. V situaci A 48 entů ze 108 spalo (tedy měli aktivních méně rozhodovacích pravidel než ti vzhůru). V situaci B byly všichni enti aktivní. Časové úseky byly zvoleny náhodně. Každý individuální test byl proveden třikrát. (Metodologická poznámka: v situaci B byla měřeno s 5-ti násobnou rychlostí.)

Výsledky testu. Následující tabulka udává přibližné hodnoty zátěže procesoru.

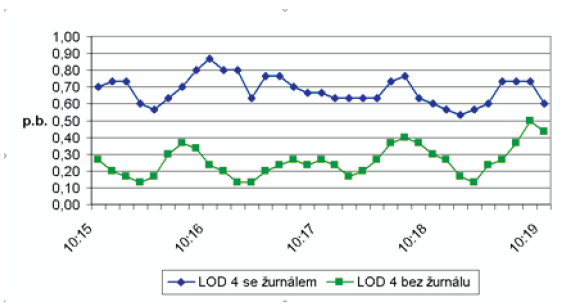
čas	# pravidel	normovaná zátěž
6.05 - 6.09	2200	0,07 - 0,27 p.b.
10.15 - 10.19	2500	0,17 - 0,50 p.b.

Tabulka 1. Průměrné zátěže procesoru pro situaci A4 a B4 a přibližné počty ohodnocovaných pravidel.

čas	# pravidel	normovaná zátěž
6.05 - 6.09	3700	5 - 7 p.b.
10.15 - 10.19	5000	6 - 8 p.b.

Tabulka 2. Průměrné zátěže procesoru pro situaci A5 a B5 a přibližné počty ohodnocovaných pravidel.

Naměřené hodnoty odpovídají situaci, kdy nejsou zobrazovány žurnály. Při zobrazení žurnálu s úrovní 4 zátěž stoupne o 0,3 – 0,6 p.b., v případě úrovně 5 až o 6 p.b. Pro ilustraci připojuji data z jednoho konkrétního sledování zátěže při situaci B4 bez žurnálu a se žurnálem. Při jiných časových úsecích (mimo specifických situací) jsou data obdobná.



Obrázek 2. Konkrétní zátěž procesoru dvou pozorování (s žurnálem a bez žurnálu).

Interpretace testu. Můžeme si všimnout čtyř věcí. Zaprvé, rychlost na úrovni 4 je více jak o řád vyšší než rychlost na úrovni 5. To není nic překvapivého. Zadruhé, počet ohodnocovaných pravidel má na tomto zpomalení určitý podíl, ale jen dílčí. Svou roli zřejmě také hraje počet simulovaných objektů, složitost prostředí a náročnost ohodnocení jednotlivých dotazů, která roste na úrovni 5 (s tím, jak roste počet objektů). Zatřetí, rychlost není konstantní. To asi také není nic překvapivého – svět se dynamicky mění. Začtvrté, při reálném provozu se vyplatí omezit zápis do žurnálů.

Test 2. Rychlost při hledání cesty. Smyslem testu bylo naměřit rychlost IVE při hromadném přemisťování entů.

Popis testu. Byly provedeny celkem 4 různé testy; rychlosti simulace v 6:00 – 6:04 (A) respektive 10:00 – 10:04 (B) při celém světě simulovaném na úrovni 4. V tyto časové úseky nastávají specifické situace. V A se 40 entů vzbudí a začne se přemisťovat, přičemž všichni intenzivně hledají cestu (pomocí algoritmu hierarchický A*). V B 40 entů pracuje v dolech (kde je navíc 20 vozíků), 48 entů se vzbudí a opět se začne hromadně přemisťovat. Každý individuální test byl proveden třikrát.

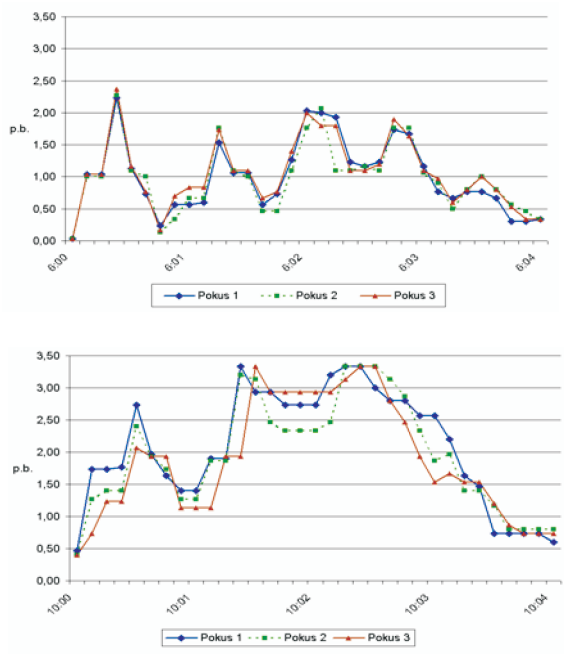
Výsledky. Následující grafy ukazují naměřené hodnoty v situacích A a B, žurnály jsou zapnuté.

Vidíme, že průběh zátěže je v testu A i B v individuálních měřeních poměrně repetitivní. Vidíme dále, že zátěž kolísá od „normálních“ hodnot až k několikanásobným. Kolísání je dáno tím, v které fázi chůze enti jsou respektive kolik jich v daný okamžik hledá nebo dohledává cestu.

Interpretace testu. Test ukázal, že se simulace téměř o řád zpomalí ve chvíli, kdy se masivně přemisťuje větší množství entů. Jelikož se všichni přemisťují stejným směrem, šlo by v tomto případě hledání cesty optimalizovat, například pomocí předpočítání určitých hodnot.

Test 3. Rychlost při zvyšování LOD. Jedná se o klíčovou studii. Jejím smyslem bylo odhalit, jestli je přínosné postupné zjemňování simulace v místě, kam se blíží uživatel. Hypotéza je, že postupné zjednodušování umožní oproti přístupu „nic – nebo všechno“ rozložit v čase režii na zvyšování úrovně z nejnižší na maximální.

Popis testu. Byly provedeny 2 hlavní testy; jakým způsobem se zvýší zátěž procesoru při zvýšení úrovně detailu v baru (bar byl vybrán proto, že je nejkomplexnější oblastí). Při testu (A) byl celý svět simulován



Obrázek 3. Průběh závislosti zátěže procesoru na čase v situaci A (nahoře) a B (dole) ve třech konkrétních pozorováních.

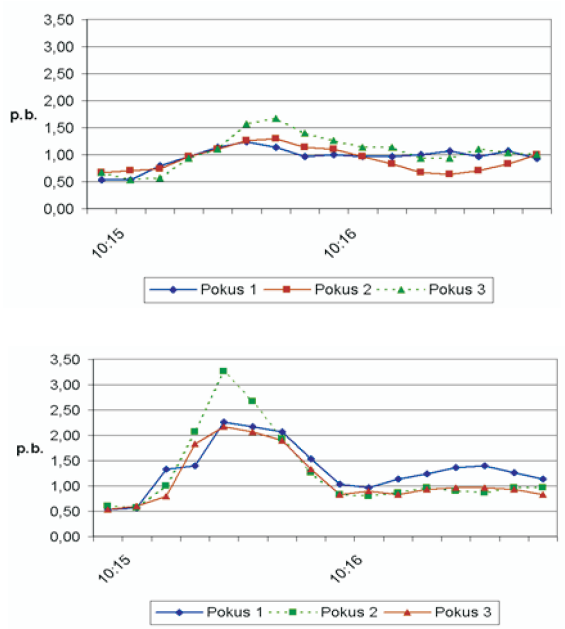
na úrovni 4 a v 10:15 byla úroveň v jednom z barů zvýšena na 5. Při testu (B) byl celý svět simulován na úrovni 4 a jedna hospoda na úrovni 3 a v 10:15 byla zvýšena úroveň dané hospody na 4 zároveň se zvýšením úrovně v baru dané hospody na 5. Zvýšením z 3 na 5 je napodobován přístup „nic – nebo všechno“. V hospodě bylo 12 entů (už od úrovně 3), z toho v baru 5 entů. Testy A i B byly provedeny třikrát.

Výsledky. Grafy na obr. 4 ukazují naměřené hodnoty v situacích A a B, žurnály jsou zapnuté.

Navíc byly změřeny další doplňující výsledky. Ukázalo se, že každá hospoda simulovaná na úrovni 5 zvyšuje zátěž oproti simulaci na úrovni 4 zhruba o 0,8 p.b. (s žurnály), z toho samotný bar o cca 0,5-0,7 p.b. Zvýšení úrovně z 4 na 5 v dolu stojí pouze 0,3 p.b.

Interpretace testu. Doplňující výsledky ukázaly, že bar je skutečně nejkomplexnější oblastí v daném světě. Hlavní testy pak ukázaly, že v dané situaci dochází k výraznému zvýšení zátěže ve chvíli, kdy roste úroveň detailu v baru na 5. Po krátké době se zvýšení stabilizuje na hodnotě odpovídající běžné simulaci. Zvýšení je ovšem pozorovatelně menší v případě, kdy k němu dochází z úrovně 4, než když k němu dochází z úrovně 3.

To je klíčový výsledek. Za prvé se ukazuje, že existuje režie přímo spojená se zvyšováním detailu. Za druhé výsledek naznačuje, že pozvolné zvyšování úrovně (z 3 na 4, ze 4 na 5) je oproti přístupu „nic – nebo



Obrázek 4. Průběh závislosti zátěže procesoru na čase v situaci A (nahoře) a B (dole) ve třech konkrétních pozorováních. V 10:15 dochází ke zvýšení úrovně detailu.

všechno“ (který byl modelován skokovým zvýšením z 3 na 5) skutečně přínosné, neboť přirozeně podporuje rozložení režie na dobu, kdy se postava uživatele blíží k dané oblasti.

Test 4. Rychlost při snižování LOD. Popis testu. Bylo zkoumáno, jaká je režie spojená se snižováním úrovně v dané oblasti. Byl opět testován bar – snižování z 5 na 4.

Výsledky. Nebylo pozorováno žádné viditelné zvýšení zátěže spojené s režií.

Test 5. Paměť. Popis testu. Při všech výše uvedených testech bylo rovněž zkoumáno, jak roste velikost paměti simulace s tím, jak se roste úroveň detailu simulace.

Výsledky. Výsledky shrnuje následující tabulka.

Interpretace testu. Vidíme, že k výraznému zvýšení dochází při změně z úrovně 2 na 3 – to je zřejmě kvůli tomu, že na úrovni 3 začínají vznikat enti a objekty. K dalšímu razantnímu zvýšení dochází při změně z úrovně 4 na 5. To je zjevně kvůli vzniku míst a dalších objektů. Spotřeba paměti je obecně veliká. Je to dáno tím, že IVE využívá veliké množství pomocných struktur (typu index, hash-mapa) za účelem zrychlení simulace. Snížení paměťových nároků by bylo doprovázeno zvýšením procesorové zátěže.

Situace	Paměť
Nahrání světa (LOD = 1)	10 MB
Přírůstek zvýšení úr. celého světa z 1 na 2	0-1 MB
Přírůstek zvýšení úr. celého světa z 2 na 3	15 MB
Přírůstek zvýšení úr. celého světa z 3 na 4	10 MB
Přírůstek zvýšení úr. jedné hospody z 4 na 5	5 MB
Přírůstek zvýšení úr. celého světa z 4 na 5	150 MB

Tabulka 3. Průměrné zátěže procesoru pro situaci A4 a B4 a přibližné počty ohodnocovaných pravidel.

4.2 Pozorování

V prototypovém světě bylo provedeno pozorování několika specifických situací souvisejících se změnou úrovně detailu.

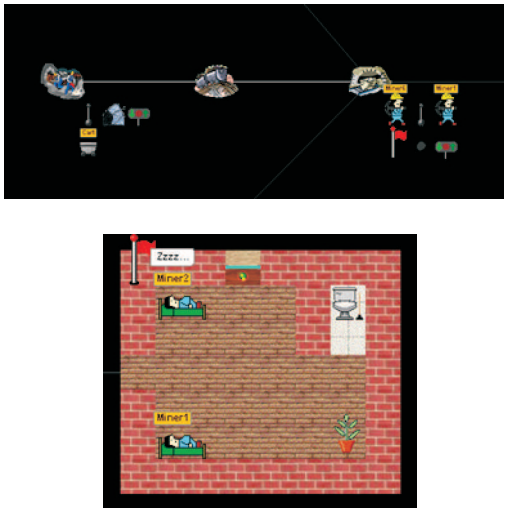
Pozorování 1. Částečné provedení činností. Pokud zvýšíme v určité oblasti úroveň detailu, musíme od té chvíle všechny činnosti začít provádět detailněji. Problém ovšem je, že část činnosti je už provedena a tento „mezivýsledek“ je často třeba nějak zohlednit, aby simulace byla věrohodná. IVE s tímto zohlednění standardně nepočítá (obr. 5 nahoře), dá se ho ovšem pro každý jednotlivý případ doprogramovat ad hoc způsobem (obr. 5 dole).

Spaní probíhá na úrovni 4 tak, že enti atomárně spí. Na úrovni 5 si jdou nejprve lehnout, pak spí a pak vstanou. V průběhu toho, kdy enti spali na úrovni 4, došlo k zvýšení úrovně na 5. Vidíme, že enti spí i na úrovni 5. Kdyby nedošlo k zohlednění „mezivýsledku“, enti by byli po zvýšení detailu vygenerováni doprostřed místnosti a teprve by si šli lehnout. Tato situace nastává, pokud zvýšíme úroveň detailu z 3 na 4 v dole. Vidíme, že po zvýšení detailu jsou oba enti v horní části dolu (vlevo), ačkoli jeden z nich už by měl nějakou dobu dolovat uhlí ve spodní části (vpravo). Simulace není věrohodná.

K tomuto pozorování se váže i ten fakt, že se zvyšující se úrovní detailu vznikají nové objekty, které musí být zapojovány do prováděných činností. Pokud se na úrovni 4 enti baví v baru bez půllitrů, na úrovni 5 tomu již tak není – půllitry jako objekty musí být konzistentně přiřazeny entům a jejich činnostem.

K budoucímu výzkumu se nabízí otázka, jestli by toto ad hoc zohledňování „mezivýsledku“ určité činnosti a přiřazování nově vzniklých objektů nešlo podchytit nějak obecně, alespoň pro určitou třídu případů.

Pozorování 2. Snížení a zvýšení. Pokud snížíme v určité oblasti detail a vzápětí ho zvýšíme, postavy nebudou pokračovat v provádění činností, které dělali před snížením. Mohou se dokonce nacházet na úplně jiných místech. Toto činí simulaci nevěrohodnou obzvláště v místech, kde se nachází hodně entů (např. v hospodě). Ve chvíli před snížením detailu je ent A u barového pultu, ve chvíli po zvýšení na toaletě.



Obrázek 5. Nahoře – standardní situace v dole ihned po zvýšení úrovně detailu z 3 na 4. Oba enti jsou v horní části dolu. Dole – situace v domku ihned po zvýšení úrovně detailu z 4 na 5 během spánku enta.

V IVE je tento problém obcházen mechanismem garbage collector – pokud má někde dojít ke snížení detailu, stane se tak až tehdy, pokud je zatížení procesoru příliš veliké. Pokud by tedy za normálního provozu postava uživatele opustila bar a vzápětí se do něj vrátila, našla by pravděpodobně vše v pořádku, neboť v baru by mezitím nestihlo dojít ke snížení detailu.

V budoucnu by nicméně bylo vhodné tento problém odstranit – například tak, že by došlo k zapamatování informací o tom, co enti před snížením dělali, a k použití této informace při opětovném zvýšení úrovně.

Pozorování 3. Ovlivnění úrovně detailu v sousední oblasti. Simulace se chová tak, že úroveň detailu ve dvou sousedních oblastech, pokud jsou v různé nadoblasti, se může lišit více než o jedna. Například trávník před hospodou může mít úroveň 5, ale náves před staveními (která s trávníkem sousedí) úroveň 3. To by v případě přechodu postavy uživatele mezi těmito oblastmi způsobilo příliš velikou režii (viz test 3).

V IVE je za tím účelem implementována možnost, aby se sousední oblasti ovlivňovaly co se úrovně svého detailu týče, zatím nicméně nebyla využita.

5 Závěr

V tomto článku jsem studoval techniku level-of-detail pro umělou inteligenci v projektu IVE, především v kontrastu s přístupy „nic – nebo všechno“ a „vidím – nevidím“, které bývají pro zjednodušování simulace používány v počítačových hrách. Hlavní výsledek studie ukázal, že pozvolné snižování detailu, na kterém je LOD v IVE postaven, je skutečně přidanou hodnotou, neboť přirozeně umožňuje rozložit v čase režii spojenou se změnou úrovně detailu. Samozřejmě míra zrychlení závisí také na tom, jakým způsobem budeme virtuální svět dekomponovat a s kolika úrovněmi abstrakce budeme pracovat. V budoucnu by bylo dobré navíc odstranit problémy spojené s nízkou věrohodností simulace v určité oblasti těsně po zvýšení úrovně detailu.

Poděkování

IVE bylo naimplementováno v roce 2005 studenty MFF UK, Praha: O. Šerým, T. Pochem, P. Šafratou, J. Kubrem, J. Kulhánkem a Z. Šulcem, jímž tímto patří dík. Práce na tomto textu byla částečně podpořena grantovým projektem GA UK 351/2006/A-INF/MFF a grantem „Information Society“ pod číslem projektu: 1ET100300517. Za podnětné komentáře děkuji R. Krylovi, O. Šerému a E. Dufkové.

Reference

1. Bratman N., Intention, Plans, and Practical Reason. Cambridge, Mass: Harvard University Press, 1987
2. Brockington M., Level-Of-Detail AI for a Large Role-Playing Game. In: AI Game Programming Wisdom, 2002
3. Brom C., Virtual Humans: How to Represent Their Knowledge. In: Proceedings of ITAT 2005, Slovak Republic, 2005, (in Czech)
4. Brom C., Kocáb P., Virtual Agents in a Simulation of an ISO-Company. A poster. In: Proc. of Intelligent Virtual Agents, Springer, Berlin, 2005
5. Brom C., Lukavský J., Proč je hranice mezi virtuální bytostí a jejím světem neostrá. In: Sborník z Kognice a umělý život, 2006
6. Brom C., Lukavský J., Šerý O., Poch T., Šafrata P., Affordances and Level-of-Detail AI for Virtual Humans. In: Proceedings of Game Set and Match 2, Delft, 2006
7. Brom C., Abonyi A., Petri-Nets for Game Plot. In: Proceedings of AISB–Artificial Intelligence and Simulation Behaviour Convention, Bristol III, 2006, 6–13
8. Champandard A.J., AI Game Development: Synthetic Creatures with Learning and Reactive Behaviors. New Riders, USA, 2003
9. Forgy C., Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. In: Artificial Intelligence, 19, 1982, 17–37
10. Gibson J.J., The Ecological Approach to Visual Perception. Boston: Houghton Muffin, 1979
11. Grinke S., Minimizing Agent Processing in „Conflict: Desert Strom“. In: AI Game Programming Wisdom II, 2004
12. Projekt IVE, webovská stránka: `http://urtax.ms.mff.cuni.cz/ive/public/about.php`

13. Russell S.J., Norvig P., Artificial Intelligence, A Modern Approach. Prentice Hall, Englewood Cliffs, New Jersey, 1995
14. O'Sullivan C., Cassell J., Vilhjálmsson H., Dingliana J., Dobbyn S., McNamee B., Peters C., Giang T., Level of Detail for Crowds and Groups. In: Computer Graphics Forum, 21(4), 2002, 733–742
15. Šerý O., Poch T., Šafrata P., Brom C., Level-Of-Detail in Behaviour of Virtual Humans. In: Proceedings of SOFSEM 2006: Theory and Practice of Computer Science, LNCS 3831, Czech Republic, 2006, 565–574
16. Wright I., Marschall J., More AI in Less Processor Time: 'Egocentric' AI. In: Gamasutra on-line, 2000 `http://www.gamasutra.com`

Algoritmus na získanie všetkých konceptov v retrográdne lexikografickom usporiadaní (pre jednostranne fuzzy konceptové zväzy)

Lucia Gotthardová

Abstrakt *V tomto článku si ukážeme algoritmus, ktorý pracuje s fuzzifikovanou verziou konceptov a nájde všetky koncepty formálneho kontextu. Dôležitým predpokladom je to, že množina všetkých podmnožín množiny objektov je v retrográdne lexikografickom (resp. retrográdnom) usporiadaní. Tento algoritmus využíva uzáverový operátor kontextu a je možné ho použiť s rôznymi fuzzifikáciami.*

1 Úvod

Ganterov Uzáverový Algoritmus Ďalšieho Konceptu (Ganter´s Next Closure Algorithm) [1] je známy algoritmus na generovanie jednoduchých konceptových zväzov. R. Bělohlávek, V. Sklenář a J. Zacpal v [2] aplikovali túto myšlienku pre fuzzy prípad a vytvorili algoritmus na získanie všetkých formálnych fuzzy konceptov.

Získavané koncepty sú usporiadavané lektikograficky. Vzhľadom na malú mieru prirodzenosti tohto usporiadania, je tento fakt nedostatkom spomínaného algoritmu.

Cieľom tohto článku je ukázať algoritmus založený na uzáverovom operátore a postupnom generovaní konceptov, ktoré sú usporiadané lexikograficky retrográdne (resp. retrográdne, zostupne). Čo sa týka prirodzenosti, tento typ usporiadania sa k nej približuje vo väčšej miere.

2 Jednostranný fuzzy konceptový zväz

Predpokladom algoritmu je využitie uzáverového operátora, pričom nezáleží na použitej fuzzifikácii (t.j. zobrazení z množiny objektov do množiny atribútov a následnom zobrazení z množiny atribútov do množiny objektov).

Nech A je neprázdna množina atribútov, nech B je neprázdna množina objektov a nech R je fuzzy relácia na ich karteziánskom súčine, t.j. $R : A \times B \to [0, 1]$.

S. Krajči v [3] definoval pojem jednostranného fuzzy konceptového zväzu. Jednu z možností fuzzifikácie predstavujú nasledujúce zobrazenia:

Definícia 2.1 *Zobrazenie z množiny prvkov do intervalu* $[0, 1]$ *sa nazýva fuzzy množina.*

Definícia 2.2 *Nech* $\nearrow: \mathcal{P}(B) \to \mathcal{F}(A)$ *je zobrazenie, ktoré každej množine* X *objektov z* B *priraďuje fuzzy množinu* $\nearrow(X)$ *atribútov, ktorej hodnota pre atribút* $a \in A$ *je*

$$\nearrow(X)(a) = \inf\{R(a, b) : b \in X\},$$

t.j. táto funkcia každému atribútu priraďuje najväčšiu hodnotu takú, že všetky objekty z X *majú tento atribút minimálne v tejto miere.*

Definícia 2.3 *Nech* $\searrow: \mathcal{F}(A) \to \mathcal{P}(B)$ *je zobrazenie, ktoré každej funkcii* $f : A \to [0, 1]$ *priraďuje množinu*

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

t.j. tie objekty, ktoré majú všetky atribúty aspoň v tomto stupni, ktorý je určený funkciou f *(inými slovami, funkcia ich fuzzy-príslušnosti k objektom dominuje* f*).*

Ľahko vidieť, že tieto zobrazenia majú nasledujúce vlastnosti (pod $f_1 \geq f_2$ rozumieme, že pre všetky prvky x z definičného oboru (rovnakého pre f_1 aj f_2) platí $f_1(x) \geq f_2(x)$):

Lemma 2.1 *Nech* $\nearrow, \searrow$ *sú zobrazenia definované v definíciach 2.1 a 2.2. Potom pre každé* $X, X_1, X_2 \subseteq B$ *a pre každé* $f, f_1, f_2 \in \mathcal{F}(A)$ *platí:*
1.*a*) $X_1 \subseteq X_2 \Rightarrow \nearrow(X_1) \geq \nearrow(X_2)$,
1.*b*) $f_1 \leq f_2 \Rightarrow \searrow(f_1) \supseteq \searrow(f_2)$,
2.*a*) $X \subseteq \searrow(\nearrow(X))$,
2.*b*) $f \leq \nearrow(\searrow(f))$.

Teraz definujeme uzáverový operátor.

Lemma 2.2 *Uzáverový operátor* $\mathrm{cl} : \mathcal{P}(B) \to \mathcal{P}(B)$ *spĺňa vlastnosti:*

Pre každé $X, X_1, X_2 \subseteq B$ *platí:*

1. $X \subseteq \mathrm{cl}(X)$
2. $X_1 \subseteq X_2 \Rightarrow \mathrm{cl}(X_1) \subseteq \mathrm{cl}(X_2)$
3. $\mathrm{cl}(\mathrm{cl}(X)) = \mathrm{cl}(X)$

Dôkaz. 1. Táto vlastnosť je priamo vlastnosť 2.a) z lemy 2.1.

2. Nech $X_1 \subseteq X_2$. Potom z vlastnosti 1.a) lemy 2.1 vyplýva, že $\nearrow(X_2) \leq \nearrow(X_1)$. Z vlastnosti 1.b) vyplýva, že $\searrow(\nearrow(X_1)) \subseteq \searrow(\nearrow(X_2))$, teda $\mathrm{cl}(X_1) \subseteq \mathrm{cl}(X_2)$.

3. To, že platí $\mathrm{cl}(\mathrm{cl}(X)) = \mathrm{cl}(X)$ ukážeme tak, že dokážeme platnosť obojstranných inklúzií. Najprv ukážeme, že platí $\mathrm{cl}(X) \subseteq \mathrm{cl}(\mathrm{cl}(X))$. Nech $S = \mathrm{cl}(X)$, kde $S \subseteq B$. Spojením s 2. vlastnosťou ($S \subseteq \mathrm{cl}(S)$),

ktorú sme už dokázali, dostaneme, že $\mathrm{cl}(X) \subseteq \mathrm{cl}(\mathrm{cl}(X))$. Teraz ukážeme, že $\mathrm{cl}(X) \supseteq \mathrm{cl}(\mathrm{cl}(X))$. Vieme, že platí vlastnosť 2.b) $(f \leq {\nearrow}({\searrow}(f)))$ z lemy 2.1. Ak $S = {\nearrow}(X)$, máme, že ${\nearrow}(X) \leq {\nearrow}({\searrow}({\nearrow}(X)))$. Ak $S_1 = {\nearrow}(X)$ a $S_2 = {\nearrow}({\searrow}({\nearrow}(X)))$, teda máme, že platí $S_1 \leq S_2$. Vieme, že platí vlastnosť 1.b) z lemy 2.1, teda $S_1 \leq S_2 \Rightarrow {\searrow}(S_1) \supseteq {\searrow}(S_2)$. Teda platí, že ${\searrow}({\nearrow}(X)) \supseteq {\searrow}({\nearrow}({\searrow}({\nearrow}(X))))$. A to je $\mathrm{cl}(X) \supseteq \mathrm{cl}(\mathrm{cl}(X))$. Teda platí rovnosť $\mathrm{cl}(\mathrm{cl}(X)) = \mathrm{cl}(X)$.

Nech $(X, {\nearrow}(X))$ je taká dvojica , kde X je množina objektov, pre ktoré platí $X = \mathrm{cl}(X)$ (pretože vtedy platí $f = {\nearrow}(X)$, akk $X = {\searrow}(f)$). Potom táto dvojica sa nazýva **jednostranný fuzzy koncept** (keďže X je klasická množina objektov a ${\nearrow}(X)$ je fuzzy množina atribútov). Potom X je **extent** tohto konceptu a príslušná fuzzy množina ${\nearrow}(X)$ je jej **intent**. Vzhľadom na vzájomnú odvoditeľnosť oboch zložiek konceptu sa stačí zaoberať len jednou zložkou, napr. prvou z nich. Potom množina $\mathcal{B}(A, B, R) = \mathcal{B}$ všetkých takých extentov, t.j. $\mathcal{B} = \{X \subseteq \mathcal{P}(B) : X = \mathrm{cl}(X)\}$, usporiadaná inklúziou, tvorí zväz, ktorého operácie sú definované nasledovne: $X_1 \wedge X_2 = X_1 \cap X_2$ a $X_1 \vee X_2 = \mathrm{cl}(X_1 \cup X_2)$. Tento zväz budeme nazývať **jednostranný fuzzy konceptový zväz**.

3 Retrográdne usporiadanie

Definícia 3.1 *Nech $B = \{1, 2, \ldots, n\}$ je množina objektov. Nech $X, Y \subseteq B, X \neq Y$ sú navzájom rôzne podmnožiny objektov. Hovoríme, že podmnožina X je **retrográdne menšia** ako podmnožina Y, ak najväčší prvok, ktorý odlišuje X od Y, patrí Y. Teda formálne:*

$$X <_{\mathrm{R}} Y, \text{ akk } (\exists i \in Y \setminus X)(X \cap \{i+1, \ldots, n\} = \\ = Y \cap \{i+1, \ldots, n\}).$$

Príklad 3.1 *Nech $B = \{1, 2, 3, 4, 5\}$. Nech $X = \{1, 3, 4, 5\}$ a $Y = \{2, 3, 4, 5\}$. Chceme zistiť, či $X <_{\mathrm{R}} Y$ alebo $Y <_{\mathrm{R}} X$. Ak by platilo $Y <_{\mathrm{R}} X$, tak by musel existovať prvok z $X \setminus Y$ taký, že od tohto prvku (okrem neho) sa tieto množiny rovnajú. Teda to by mohol byť len prvok 1, ale potom dostaneme nerovnosť $\{3, 4, 5\} \neq \{2, 3, 4, 5\}$, čiže neplatí $Y <_{\mathrm{R}} X$. Naopak, keď zoberieme maximálny prvok z $Y \setminus X$, teda prvok 2, tak dostaneme rovnosť $\{3, 4, 5\} = \{3, 4, 5\}$, a preto platí $X <_{\mathrm{R}} Y$.*

Príklad 3.2 *Majme množinu extentov Extenty = $\{\{\}, \{1\}, \{2\}, \{2, 1\}, \{3, 2, 1\}, \{5\}, \{5, 1\}, \{5, 2\}, \{5, 2, 1\}, \{5, 3, 2, 1\}, \{5, 4, 2, 1\}, \{5, 4, 3, 2, 1\}\}$. Platí, že $\{\} <_{\mathrm{R}} \{1\}, \{1\} <_{\mathrm{R}} \{2\}, \{2\} <_{\mathrm{R}} \{2, 1\}, \ldots, \{5, 4, 2, 1\} <_{\mathrm{R}} \{5, 4, 3, 2, 1\}$. Teda extenty v tejto množine sú usporiadané retrográdne.*

Lemma 3.1 *Relácia $<_{\mathrm{R}}$ z definície 3.1 je usporiadanie a pre všetky podmnožiny $X, Y, Z \in B$ spĺňa nasledujúce vlastnosti:*

1. *neplatí, že $X <_{\mathrm{R}} X$,*
2. *ak $X <_{\mathrm{R}} Y$, tak neplatí, že $Y <_{\mathrm{R}} X$,*
3. *ak $X <_{\mathrm{R}} Y$ a zároveň $Y <_{\mathrm{R}} Z$, tak $X <_{\mathrm{R}} Z$, kde minimálne prvky, v ktorých sa dvojice X, Y a Y, Z odlišujú, sú rôzne.*

Dôkaz. 1. Aby platilo, že $X <_{\mathrm{R}} X$, tak by musel existovať nejaký prvok i z $X \setminus X$. Ale keďže $X \setminus X$ je prázdna množina, taký prvok i nenájdeme. Teda neplatí, že $X <_{\mathrm{R}} X$.

2. Ak $X <_{\mathrm{R}} Y$, tak $(\exists i \in Y \setminus X)(X \cap \{i+1, \ldots, n\} = Y \cap \{i+1, \ldots, n\})$. Predpokladáme, že $Y <_{\mathrm{R}} X$, teda $(\exists j \in X \setminus Y)(X \cap \{j+1, \ldots, n\} = Y \cap \{j+1, \ldots, n\})$. Nemôže platiť, že $j = i$, pretože neexistuje taký prvok, ktorý by patril do $Y \setminus X$ a zároveň aj do $X \setminus Y$. Ak $j < i$, tak by muselo platiť, že $A = X \cap \{j+1, \ldots, i+1, i, \ldots, n\} = Y \cap \{j+1, \ldots, i+1, i, \ldots, n\} = B$, ale to neplatí, pretože v množine B by (okrem iných) bol aj prvok i a tento prvok v množine A byť nemôže, keďže $i \in Y \setminus X$ (a teda $i \notin X$). Ak $j > i$, tak by muselo platiť, že maximálny prvok, ktorým sa X a Y odlišujú, je j, a to je spor, pretože týmto maximálnym prvkom je i.

3. Nech $X <_{\mathrm{R}} Y$, teda $(\exists i \in Y \setminus X)(X \cap \{i+1, \ldots, n\} = Y \cap \{i+1, \ldots, n\})$. Nech $Y <_{\mathrm{R}} Z$, teda $(\exists j \in Z \setminus Y)(Y \cap \{j+1, \ldots, n\} = Z \cap \{j+1, \ldots, n\})$. Nemôže platiť, že $i = j$, lebo potom i by bol taký prvok, že $i \neq Y$ (lebo $j \in Z \setminus Y$). Ak $j < i$, tak X a Z majú spoločné prvky od $i+1$ až po n, zároveň $i \in Z$ a $i \notin X$, teda maximálny prvok, ktorým sa X a Z odlišujú, je práve i, teda $X <_{\mathrm{R}} Z$. Ak $j > i$, tak X a Z majú spoločné prvky od $j+1$ až po n a zároveň $j \in Z$, ale platí aj, že $j \notin X$, lebo ak by $j \in X$, tak j by bol maximálny prvok, ktorým sa X a Y odlišujú a platilo by $Y <_{\mathrm{R}} X$. Teda $X <_{\mathrm{R}} Z$.

Lemma 3.2 *Usporiadanie z definície 3.1 je lineárne usporiadanie na $\mathcal{P}(B)$, teda pre ľubovoľné $X \neq Y$ z $\mathcal{P}(B)$, kde $B = \{1, 2, \ldots, n\}$ platí nasledujúca podmienka:*

$$\text{buď } X <_{\mathrm{R}} Y, \text{ alebo } X >_{R} Y.$$

Dôkaz. Ak $X \neq Y$, tak stačí zobrať $i = \max(X \triangle Y)$, pretože i je maximálne také, že sa v ňom množiny X a Y líšia, a teda vo všetkých prvkoch väčších ako i sa zhodujú. Ak $i \in X$, tak $Y <_{\mathrm{R}} X$ a ak $i \in Y$, tak $X <_{\mathrm{R}} Y$.

4 Myšlienka algoritmu

Myšlienkou tohto algoritmu je pre ľubovoľnú podmnožinu $X \subseteq B$ nájsť vždy extent, ktorý je najmenší

hneď za X s ohľadom na retrográdne usporiadanie. Potom všetky extenty pre daný kontext nájdeme takto: Retrográdne najmenší konceptový extent je cl({}). Ďalšie extenty nájdeme hľadaním množiny, ktorá je retrográdne najbližšie k naposledy nájdenému extentu. Na konci získame retrográdne najväčší extent, a to množinu B. Je potrebné poznamenať, že extenty získavame tak, že objekty sú v nich usporiadané lexikograficky (teda vzostupne). Aby sme dosiahli retrográdne usporiadanie výslednej množiny extentov, musíme objekty v každom novozískanom extente preusporiadať od najvačšieho po najmenší (teda zostupne).

Definícia 4.1 *Nech* $X, Y \subseteq B, i \in B$. *Potom*

$$\mathrm{RI}\,(Y,X) = \begin{cases} i, & \textit{ak } \max(Y \setminus X) = \max(Y \triangle X) \\ & = i, \\ \text{nedef.} & \textit{inak} \end{cases}$$

a

$$X \oplus i = \mathrm{cl}((X \cap \{i+1, \ldots, n\}) \cup \{i\}).$$

Poznámka 4.1 RI je skratka pre RozlišujúciIndex. Ak i je RozlišujúciIndex
(Y, X), to znamená, že $i \in Y \setminus X$ a X je zhodné s Y od objektu $i+1$ po objekt n.

	a	b	c	d	e
1	1	0.8	0.2	0.3	0.5
2	0.8	1	0.2	0.6	0.9
3	0.2	0.3	0.2	0.3	0.4
4	0.4	0.7	0.1	0.2	0.3
5	1	0.9	0.3	0.2	0.4

Obrázok 1. Formálny kontext.

Príklad 4.1 *Majme kontext, ktorý je na obrázku 1, kde* $A = \{a, b, c, d, e\}$ *je množina atribútov a* $B = \{1, 2, 3, 4, 5\}$ *je množina objektov. Nech* $X = \{1, 2\}$ *a* $i = 3$ *Potom* $X \oplus i = \mathrm{cl}((\{1, 2\} \cap \{4, 5\}) \cup \{3\}) = \mathrm{cl}(\{1, 2, 3\}) = \{1, 2, 3\}$.

Príklad 4.2 *Majme opäť kontext z obrázku 1. Nech* $X = \{1, 2, 3, 5\}$ *a* $Y = \{1, 2, 4, 5\}$. *Potom* $\max(Y \setminus X) = \max(\{4\}) = 4 = \max(Y \triangle X) = \max((Y \cup X) \setminus (Y \cap X)) = \max(\{1, 2, 3, 4, 5\} \setminus \{1, 2, 5\}) = \max(\{3, 4\})$. *Teda* $\mathrm{RI}\,(Y, X) = 4$.

Lemma 4.1 *Nech* $X, Y, Z \subseteq B$ *sú podmnožiny množiny objektov a nech* $i \in B$. *Potom platia nasledujúce tvrdenia:*

1. *Ak* $X \subseteq Y$, *tak* $X \leq_{\mathrm{R}} Y$.
2. $X \leq_{\mathrm{R}} Y$, *akk* $\mathrm{RI}(Y, X) = i$ *pre nejaké* $i \in B$.
3. *Ak* $\mathrm{RI}(Y, X) = i, \mathrm{RI}(Z, X) = j$ *a* $i > j$, *tak* $\mathrm{RI}(Y, Z) = i$.
4. *Ak* $i \notin X$, *tak* $X <_{\mathrm{R}} X \oplus i$.
5. *Ak pre nejaký extent* Y *platí, že* $\mathrm{RI}(Y, X) = i$, *tak* $X \oplus i \subseteq Y$ *a takisto* $X \oplus i \leq_{\mathrm{R}} Y$.
6. *Ak pre nejaký extent* Y *platí, že* $\mathrm{RI}(Y, X) = i$, *tak* $\mathrm{RI}(X \oplus i, X) = i$.

Dôkaz. 1. Ak $X \subseteq Y$, tak Y bude obsahovať nejaký prvok $i \in Y \setminus X$ taký, že X sa zhoduje s Y od objektu $i+1$, teda $X <_{\mathrm{R}} Y$. Ak $X = Y$, tak sa samozrejme zhodujú vo všetkých prvkoch.

2. Vyplýva priamo z definícií 3.1 a 4.1.

3. Ak X je s zhodné s Y od objektu $i+1$, X je zhodné so Z od objektu $j+1$ a zároveň $i > j$, tak Y a Z sa zhodujú od objektu $i+1$. Z sa v objekte i zhoduje s X, ale s Y už nie, teda Y a Z sa v ňom líšia. Keďže $i \in B$ (lebo $i \in Y \setminus X$), tak z definície vyplýva, že $\mathrm{RI}(Y, Z) = i$.

4. $X \oplus i = \mathrm{cl}((X \cap \{1, 2, \ldots, i-1\}) \cup \{i\})$. Teda uzáverom získame najmenší extent obsahujúci objekty $X \cap \{1, 2, \ldots, i-1\}$ a $\{i\}$. Čiže $X \oplus i$ sa s X budú zhodovať po objekt $i-1$ a keďže $i \notin X$, tak platí, že $i \in (X \oplus i) \setminus X$, a teda $X <_{\mathrm{R}} X \oplus i$.

5. Nech Y je ľubovoľný extent. Keďže $\mathrm{RI}(Y, X) = i$, tak X sa s Y zhoduje od objektu $i+1$ a objekt i, v ktorom sa nezhodujú, patrí do Y. Z 2. vlastnosti uzáverového operátora (lema 2.2) a z toho, že Y je extent, dostávame, že ak $(X \cap \{i+1, \ldots, n\}) \cup \{i\} \subseteq Y$, potom $\mathrm{cl}((X \cap \{i+1, \ldots, n\}) \cup \{i\}) \subseteq \mathrm{cl}(Y)$, teda $X \oplus i \subseteq Y$. A z toho (z tvrdenia 1.) priamo vyplýva, že $X \oplus i \leq_{\mathrm{R}} Y$.

6. Nech Y je ľubovoľný extent. Keďže $\mathrm{RI}(Y, X) = i$, tak $i \in Y \setminus X$ a $X \cap \{i+1, \ldots, n\} = Y \cap \{i+1, \ldots, n\}$. Chceme ukázať, že $i \in (X \oplus i) \setminus X$ a $X \cap \{i+1, \ldots, n\} = (X \oplus i) \cap \{i+1, \ldots, n\}$. Keďže z 5. tvrdenia tejto lemy máme, že $X \oplus i \subseteq Y$, tak platí $(X \oplus i) \cap \{i+1, \ldots, n\} \subseteq Y \cap \{i+1, \ldots, n\} = X \cap \{i+1, \ldots, n\}$. Ukážeme aj opačnú implikáciu: $X \cap \{i+1, \ldots, n\} \subseteq (X \cap \{i+1, \ldots, n\}) \cup \{i\}$. Z 1. vlastnosti uzáverového operátora (lema 2.2) vyplýva, že $(X \cap \{i+1, \ldots, n\}) \cup \{i\} \subseteq \mathrm{cl}((X \cap \{i+1, \ldots, n\}) \cup \{i\}) = X \oplus i$, a teda $X \cap \{i+1, \ldots, n\} \subseteq (X \oplus i) \cap \{i+1, \ldots, n\}$. Keďže $i \in X \oplus i$ a $i \notin X$, tak $\mathrm{RI}(X \oplus i, X) = i$.

5 Algoritmus Všetky koncepty usporiadané retrográdne

Algoritmus VŠETKY KONCEPTY USPORIADANÉ RETROGRÁDNE pre nájdenie všetkých extentov daného kontextu môžeme popísať takto: Retrográdne najmenší extent je cl({}). Pre danú množinu $X \subseteq B$ nájdeme nasledujúci extent kontrolovaním všetkých prvkov i z $B \setminus X$, a to tak, že začneme od najmenšieho a postupne hodnotu i zvyšujeme, kým nebude platiť $RI(X \oplus i, X) = i$. Potom ďalší extent, ktorý sme hľadali, je práve $X \oplus i$. Algoritmus končí vtedy, ak nájdeme retrográdne najväčší extent, t.j. $\{n, ..., 1\}$.

Nech A je množina atribútov, B je množina objektov, $R \subseteq A \times B$ je kontext. Nech Extenty je množina nájdených extentov, AktuálnyExt $\subseteq B$ je aktuálny extent, ku ktorému hľadáme nasledujúci extent, NovýExt $\subseteq B$ je nový extent, ktorý testujeme, či spĺňa vlastnosť, že je najmenším extentom väčším ako AktuálnyExt. Nech MinID je najmenší objekt z B, PO(B) je celkový počet objektov v množine B, PO(AktuálnyExt) je počet objektov v aktuálnom extente. Nech Retrográdne je funkcia, ktorá objekty, ktoré sú v extente usporiadané lexikograficky (vzostupne) usporiadá od najväčšieho po najmenší (zostupne).

Tento algoritmus funguje len pre konečné množiny objektov B. Algoritmus VŠETKY KONCEPTY USPORIADANÉ RETROGRÁDNE, ktorého výstupom je množina Extenty, vyzerá nasledovne:

```
Extenty := {};
AktuálnyExt := cl({});
 while PO(B) <> PO(AktuálnyExt) do
    begin
       i := MinID;
       NovýExt := AktuálnyExt ⊕ i;
       while RI(NovýExt, AktuálnyExt) <> i do
          begin
             inc(i);
             NovýExt := AktuálnyExt ⊕ i;
          end;
       AktuálnyExt := NovýExt;
       NovýExt := Retrográdne(NovýExt);
       Extenty := Extenty ∪ NovýExt;
    end;
```

iterácia	5	4	3	2	1	i
1						1
2					×	2
3				×		1
4				×	×	3
5			×	×	×	5
6	×				×	1
7	×					2
8	×			×		1
9	×			×	×	3
10	×		×	×	×	4
11	×	×		×	×	3
12	×	×	×	×	×	

Obrázok 2. Tabuľka extentov postupne získavaných algoritmom VŠETKY KONCEPTY USPORIADANÉ RETROGRÁDNE.

Príklad 5.1 *Majme kontext, ktorý je na obrázku 1. Postupné získavanie extentov naším algoritmom je znázornené v tabuľke na obrázku 2. V prvom stĺpci je číslo iterácie. V prvej iterácii do množiny extentov Extenty pridáme retrográdne najmenší extent, t.j. prázdnu množinu* $\{\}$*. V stĺpci i sa nachádza hodnota nášho hľadaného i a krížiky v stĺpcoch 5, ..., 1 označujú ktoré objekty patria do extentu získaného v danej iterácii. V poslednej iterácii dostaneme extent, ktorý je rovný množine B, t.j. algoritmus skončil. Výsledkom sú všetky extenty v retrográdnom usporiadaní.*

Tento algoritmus teda končí vtedy, keď nájde retrográdne najväčší extent, ktorým je samotná množina objektov B, a to usporiadaná zostupne (t.j. $\{n, ..., 1\}$). Ako prvý získa retrográdne najmenší extent $\{\}$ a postupne do výslednej množiny extentov pridáva ďalšie extenty, a to s ohľadom na retrográdne usporiadanie. O tom, že najmenší extent, ktorý je retrográdne väčší ako práve nájdená množina X, je $X \oplus i$, hovorí nasledujúca veta.

Veta 5.1 *Najmenší extent konceptu, väčší ako daná množina* $X \subseteq B$ *(s ohľadom na retrográdne usporiadanie), je* $X \oplus i$*, kde i je najväčší prvok množiny B s vlastnosťou:*

$$\mathit{RozlišujúciIndex}\,(X \oplus i, X) = i.$$

Dôkaz. Nech $X^{'}$ je najmenší extent väčší ako X s ohľadom na retrográdne usporiadanie, teda $X <_{\mathrm{R}} X^{'}$. Potom z lemy 4.1 (2. tvrdenie) vyplýva, že $\mathrm{RI}(X^{'}, X) = i$ pre nejaké $i \in B$ a takisto $\mathrm{RI}(X \oplus i, X) = i$ vyplýva zo 6. tvrdenia lemy 4.1. Z toho, že $\mathrm{RI}(X^{'}, X) = i$ vyplýva, že $i \notin X$, a teda zo 4. tvrdenia dostaneme, že $X <_{\mathrm{R}} X \oplus i$. Z 5. tvrdenia platí, že $X \oplus i \subseteq X^{'}$, a teda $X \oplus i <_{\mathrm{R}} X^{'}$. Z týchto všetkých dôsledkov, z predpokladu, že $X^{'}$ je najmenší extent väčší ako X (teda medzi nimi sa nevyskytuje žiadny iný extent) a z toho, že $X <_{\mathrm{R}} X \oplus i \leq_{\mathrm{R}} X^{'}$ vyplýva, že $X \oplus i = X^{'}$.

Teraz ukážeme, že i je najväčší prvok z množiny B taký, že $\mathrm{RI}(X \oplus i, X) = i$. Nech $\mathrm{RI}(X \oplus j, X) = j$ pre $j \neq i$. Potom z toho, že $X \oplus i$ je najmenší extent väčší ako X, vyplýva, že $X \oplus i = X^{'} <_{\mathrm{R}} X \oplus j$. Keďže $i \neq j$, tak buď $i < j$, alebo $j < i$. Nech $i < j$. Potom z 3. tvrdenia lemy 3.1 vyplýva, že $\mathrm{RI}(X \oplus i, X \oplus j) = j$, kde $j \in B$ a z 2. tvrdenia ďalej dostaneme, že $X \oplus j <_{\mathrm{R}} X \oplus i$, a to je spor s tým, že $X \oplus i = X^{'} <_{\mathrm{R}} X \oplus j$. Teda platí, že $j < i$.

6 Záver

V tomto článku sme ukázali algoritmus, ktorý generuje všetky koncepty usporiadané retrográdne. Povedzme si teraz niečo o zložitosti predvedeného algoritmu.

Nech vstupom algoritmu je trojica (A, B, R) (formálny kontext), Extenty je výstupná množina získaných extentov a zatiaľ posledným získaným extentom bola množina $X \subseteq B$. Potom časová zložitosť výpočtu nasledujúceho extentu $X \oplus i$ je polynomiálna, a to $O(|B| \cdot |A|^2)$. Teda celková zložitosť algoritmu je $O(|\mathrm{Extenty}| \cdot |B| \cdot |A|^2)$.

Neuvažujme teraz o zložitosti výpočtu uzáverového operátora. Za funkciu uzáveru cl vezmime nejakú inú funkciu $f : B \to B$, ktorá spĺňa vlastnosti uzáveru. Všimnime si, že zložitosť algoritmu je od množiny atribútov A nezávislá. A vystupuje iba v zložitosti výpočtu uzáveru. Teda zložitosť algoritmu medzi jednotlivými krokmi (resp. nájdením nasledujúceho extentu) je lineárna, t.j. $O(|B|)$. Celková zložitosť algoritmu závisí iba na výslednom počte extentov a množine objektov B. Teda zložitosť je v takomto prípade $O(|Extenty| \cdot |B|)$.

Referencie

1. B. Ganter, R. Wille, Formal Concept Analysis: Mathematical Foundations, Springer–Verlag, Berlin/Heidelberg, 1999
2. R. Bělohlávek, V. Sklenář, J. Zacpal, Crisply Generated Fuzzy Concepts, Dept. Computer Science, Palacký University, Olomouc, 2004
3. S. Krajči, Cluster Based Efficient Generation of Fuzzy Concepts, Neural Network World 5, 2003, 521–530

Lineární logika, formalismus pro problémy s omezenými zdroji

Lukáš Chrpa

Katedra teoretické informatiky a matematické logiky
Matematicko-fyzikální fakulta, Karlova Univerzita, Praha
chrpa@kti.mff.cuni.cz

Abstrakt *Tento článek se bude zabývat lineární logikou, kterou je možno chápat jako formalismus, pomocí kterého lze popisovat problémy, ve kterých se vyskytuje omezený počet zdrojů. Pojmem zdroj nemusíme nutně chápat pouze materiální objekt, ale také objekt abstraktní. Takové abstraktní objekty jsou například značky v místech Petriho sítí, které provedením libovolného přechodu spotřebováváme či vytváříme. Kódování Petriho sítí pomocí lineární logiky ukazuje modelovací sílu, kterou lineární logika disponuje. Praktičtějšími problémy, které lze snadno kódovat pomocí lineární logiky, jsou plánovací problémy. Kódování plánovacích problémů v lineární logice je obdobné jako kódování Petriho sítí (v lineární logice). Pro řešení problémů kódovaných v lineární logice existují řešiče postavené na bázi logického programování (Prolog), které jsou při řešení těchto problémů mnohem efektivnější, ale tyto řešiče zatím nejsou dostatečně silné pro řešení všech problémů kódovaných v lineární logice (např. plánovací problémy). Pro řešení těchto problémů je zapotřebí jiných technik, které zatím nejsou příliš efektivní.*

1 Úvod

Lineární logika je formalismus, pomocí kterého můžeme kódovat problémy, ve kterých se vyskytuje omezený počet zdrojů. V reálném světě existuje mnoho problémů s omezeným počtem zdrojů, se kterými se setkáváme každý den (například nakupování v obchodě). Problémy s omezeným počtem zdrojů lze rovněž kódovat pomocí 'klasické' logiky, nicméně toto kódování vzhledem k velikosti problému, může růst exponenciálně.

Lineární logika je poměrně mladý vědní obor (představena v roce 1987), nicméně ve výzkumu lineární logiky byly učiněny zajímavé objevy, i když spíše v teoretické oblasti. Asi nejvýznamnější z těchto objevů byl fakt, že pomocí lineární logiky můžeme kódovat Petriho sítě. Z mého hlediska byl tento objev důležitý i proto, že ukázal poměrně velkou modelovací sílu lineární logiky. Obdobně jako Petriho sítě lze v lineární logice rovněž kódovat plánovací problémy, kde jako omezené zdroje chápeme predikáty, vyjadřující stavy světa a které se během výpočtu mění. Na bázi lineární logiky vznikl v roce 2003 plánovač jménem RAPS [11], který v porovnání z nejlepšími účastníky IPC (International Planning Competition) 2002 dosáhl zajímavých výsledků.

Další oblastí výzkumu je lineárně logické programování. Lineárně logické programování je odvozeno převážně od klasického logického programování (Prolog). Pomocí lineárně logického programování jsme schopni nejen jednoduše zakódovat problémy s omezeným počtem zdrojů, ale navíc i dosáhnout větší efektivity (menšího času) při hledání řešení.

V tomto článku bude dále popsaná lineární logika, kódování některých problémů v lineární logice, kódování Petriho sítí a plánovacích problémů v lineární logice, lineárně logické programování, jeho výhody a nevýhody a nakonec možné směry mého dalšího výzkumu v této oblasti.

2 Lineární logika

Lineární logika poprvé spatřila světlo světa v roce1987, kdy byla představena J.Y. Girardem [4,6,12]. Jak již bylo zmíněno v úvodu, lineární logika byla navržena jako formalismus, který umí správně popsat problémy s omezeným počtem zdrojů. Základní Girardovou myšlenkou bylo navrhnout implikaci tak, aby na rozdíl od 'klasické' výrokové logiky byla schopna popsat situaci, kdy musí být nějaký zdroj spotřebován, aby mohl další zdroj vzniknout. Takovou situaci můžeme vidět například v obchodě, kde za peníze nakoupíme zboží (po nákupu jsme bez peněz, ale máme zboží).

2.1 Operátory lineární logiky

V následujícím popisu operátorů lineární logiky vyjadřují predikáty A a B (spotřebovatelné) zdroje.[1]

implikace $A \multimap B$ — Základní operátor lineární logiky, který vyjadřuje, že B je obdrženo použitím (jednoho) A.

multiplikativní konjunkce $A \otimes B$ — Tento operátor vyjadřuje, že A a B jsou použity společně. (Jsou-li na levé straně implikace, pak A a B jsou spotřebovány, jsou-li na pravé straně implikace A a B jsou získány.)

multiplikativní disjunkce $A \wp B$ — Tento operátor vyjadřuje výrok: 'pokud ne A pak B' ($A^{\perp} \multimap B$).

[1] V následujícím textu bude používán pojem 'lineární fakty', který má zdůraznit, že se jedná o predikáty v lineární logice.

additivní konjunkce $A\&B$ — Tento operátor vyjadřuje volbu mezi A a B. Tato volba závisí na uživateli.

additivní disjunkce $A \oplus B$ — Tento operátor vyjadřuje volbu mezi A a B. Tato volba závisí na 'někom jiném'.

exponenciál $!A$ — Tento operátor vyjadřuje A jako nespotřebovatelný zdroj (tj. máme vždy dostatek A).

exponenciál $?A$ — Tento operátor znamená, že máme nějaký počet A, který je na nás nezávislý.

negace $A^{\perp}$ — Tento operátor zajišťuje dualitu mezi multiplikativními operátory $(\otimes, \wp)$, additivními operátory $(\&, \oplus)$ a exponenciály (!,?). Dvojitá negace zajišťuje identitu $((A^{\perp})^{\perp} = A)$.

Lineární logika navíc definuje konstanty $(1, \top, \bot, 0)$ jako neutrální prvky vůči operátorům $(\otimes, \&, \wp, \oplus)$.

2.2 Dokazování v lineární logice

Dokazování v lineární logice je postaveno na Gentzenově notaci, obdobně jako u jiných druhů logik. Zápis $\Delta \vdash \Gamma$ vyjadřuje, že multiplikativní konjunkce $(\otimes)$ formulí z Δ implikuje $(\multimap)$ multiplikativní disjunkci $(\oplus)$ formulí z Γ. Odvozovací pravidla jsou postaveny na obdobné bázi jako v 'klasické' logice a mají následující tvar:

$$\frac{\text{Předpoklad1} \qquad \text{Předpoklad2}}{\text{Úsudek}}$$

Sekvenční kalkul lineární logiky je poměrně obsáhlý, proto je zde uvedena jen jeho část obsahující axiomy a prakticky využitelné operátory $(\multimap, \otimes, \&, \oplus, !)$, celý kalkul je popsán v [6,12]:

$$A \vdash A \qquad \frac{\Delta_1 \vdash A, \Gamma_1 \quad \Delta_2, A \vdash \Gamma_2}{\Delta_1, \Delta_2 \vdash \Gamma_1 \Gamma_2}$$

$$\frac{\Delta, A, B \vdash \Gamma}{\Delta, (A \otimes B) \vdash \Gamma} \qquad \frac{\Delta_1 \vdash A, \Gamma_1 \quad \Delta_2 \vdash B, \Gamma_2}{\Delta_1, \Delta_2 \vdash (A \otimes B), \Gamma_1 \Gamma_2}$$

$$\frac{\Delta_1 \vdash A, \Gamma_1 \quad \Delta_2, B \vdash \Gamma_2}{\Delta_1, \Delta_2, (A \multimap B) \vdash \Gamma_1 \Gamma_2} \qquad \frac{\Delta, A \vdash B, \Gamma}{\Delta \vdash (A \multimap B), \Gamma}$$

$$\frac{\Delta, A \vdash \Gamma}{\Delta, (A \& B) \vdash \Gamma} \quad \frac{\Delta, B \vdash \Gamma}{\Delta, (A \& B) \vdash \Gamma} \qquad \frac{\Delta \vdash A, \Gamma \quad \Delta \vdash B, \Gamma}{\Delta \vdash (A \& B), \Gamma}$$

$$\frac{\Delta, A \vdash \Gamma \quad \Delta, B \vdash \Gamma}{\Delta, (A \oplus B) \vdash \Gamma} \qquad \frac{\Delta \vdash A, \Gamma}{\Delta \vdash (A \oplus B), \Gamma} \quad \frac{\Delta \vdash B, \Gamma}{\Delta \vdash (A \oplus B), \Gamma}$$

$$\frac{\Delta \vdash \Gamma}{\Delta, !A \vdash \Gamma} \qquad \frac{\Delta, !A, !A \vdash \Gamma}{\Delta, !A \vdash \Gamma}$$

$$\frac{\Delta, A \vdash \Gamma}{\Delta, !A \vdash \Gamma} \qquad \Delta \vdash \top, \Gamma$$

$$\frac{\Delta, A \vdash \Gamma}{\Delta, (\forall x) A \vdash \Gamma} \qquad \frac{\Delta, A \vdash \Gamma}{\Delta, (\exists x) A \vdash \Gamma}$$

2.3 Rozhodnutelnost a složitost lineární logiky

Následující tabulka dává přehled o rozhodnutelnosti a složitosti fragmentů lineární logiky:

Fragmenty	Složitost
$\multimap, \otimes, \wp, \&, \oplus, !, ?$	Nerozhodnutelný [13]
$\multimap, \otimes, \wp, \&, \oplus$	PSPACE-úplný [13]
$\multimap, \otimes, \wp$	NP-úplný [9]
$\multimap, \otimes, \wp, !, ?$	Není známo

3 Kódování problémů v lineární logice

Jak už bylo mnohokrát zmíněno dříve, lineární logika je vhodný formalismus pro kódování problémů s omezenými zdroji. Následující podkapitoly ukáží kódování jednodušších problémů.

3.1 Hamiltonova kružnice

Z teorie grafů víme, že Hamiltonova kružnice je taková kružnice, která obsahuje všechny vrcholy daného grafu. Nechť $G = (V, E)$ je orientovaný graf, kde $V = \{v(1), \ldots, v(n)\}$ je množina jeho vrcholů a $E \subseteq V \times V$ je množina jeho hran. Víme, že Hamiltonova kružnice obsahuje každý vrchol (kromě prvního a posledního) právě jednou, tudíž se tyto vrcholy dají reprezentovat pomocí lineárních faktů. Hrany není nutné spotřebovávat, tudíž je musíme reprezentovat jako 'klasické' fakty, v lineární logice použijeme exponenciál !. Následující pravidlo zajišťuje hledání nenavštíveného souseda $v(j)$ vrcholu $v(i)$ (predikát at vyjadřuje aktuálně prohledávaný vrchol):

$$at(i) \otimes v(i) \otimes e(v(i), v(j)) \multimap at(v(j))$$

Celý problém nalezení Hamiltonovy kružnice v grafu zakódovaný v lineární logice vypadá následovně (Hamiltonovu kružnici hledáme z vrcholu $v(1)$):

$$v(1), \ldots, v(n), !e(v(i_1), v(j_1)), \ldots$$
$$\ldots, !e(v(i_m), v(j_m)), at(v(1)),$$
$$!(at(v(i)) \otimes v(i) \otimes e(v(i), v(j)) \multimap at(v(j))) \vdash at(v(1))$$

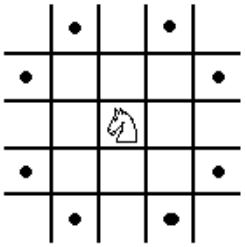
Graf obsahuje Hamiltonovu kružnici právě tehdy, když je předchozí výraz dokazatelný v lineární logice.

3.2 Tah jezdce

Problém tahu jezdce je takový problém, kde máme jezdce na šachovnici a jeho úkolem je navštívit každé políčko právě jednou. V tomto případě je jasné, že políčka šachovnice můžeme reprezentovat lineárními fakty. Předpokládejme, že existuje relace $next : (i,j) \rightarrow (\bar{i}, \bar{j})$, která nám spočítá následující tah jezdce (viz. Obr. 1). Predikát at vyjadřuje aktuální pozice jezdce. Problém tahu jezdce může být kódován v lineární logice následujícím způsobem (jezdec začíná na políčku (1,1) a končí na políčku (k,l)):

$$b(1,1), \ldots, b(n,n), at(1,1),$$
$$!(at(i,j) \otimes b(i,j) \multimap at(next(i,j))) \vdash at(k,l) \otimes b(k,l)$$

Problém tahu jezdce má řešení právě tehdy, když je předchozí výraz dokazatelný v lineární logice.



Obrázek 1. Možné tahy jezdce.

4 Kódování Petriho sítí v lineární logice

Petriho sítě lze chápat jako modelovací nástroj pro problémy se zdroji, což má samo o sobě velmi blízko k lineární logice. Petriho sítě obsahují místa, která obsahují značky vyjadřující jednotlivé zdroje, a přechody jejichž provedením se značky z některých míst odebírají (podle vstupní funkce přechodu) a na jiná místa se přidávají (podle výstupní funkce přechodu). Veškeré informace ohledně teorie Petriho sítí lze nastudovat například v [16].

Kódování Petriho sítí v lineární logice je poměrně jednoduché [15], stačí si hlavně uvědomit fakt, že značky v místech Petriho sítě můžeme chápat jako omezené zdroje a tudíž je v lineární logice můžeme reprezentovat pomocí lineárních faktů. Přechody kódujeme pomocí lineární implikace tak, že na levé straně implikace budeme mít multiplikativní konjunkci všech lineárních faktů reprezentujících vstupní místa přechodu a na pravé straně implikace budeme mít multiplikativní konjunkci všech lineárních faktů reprezentujících výstupní místa přechodu. Formálně zapsáno (předpokládejme, že $I(t)$ je multimnožina vstupních míst přechodu t a $O(t)$ je multimnožina výstupních míst přechodu t):

$$\bigotimes I(t) \multimap \bigotimes O(t)$$

Dále předpokládejme, že $S(\mu)$ je multimnožina míst odpovídající značení μ (tj. $\forall p \in P : \#(p, S(\mu)) = \mu(p)$). Značení μ_g je dosažitelné právě když je následující výraz dokazatelný v lineární logice.

$$S(\mu_0), !\left(\bigotimes I(t_1) \multimap \bigotimes O(t_1)\right), \ldots$$
$$\ldots, !\left(\bigotimes I(t_m) \multimap \bigotimes O(t_m)\right) \vdash \bigotimes S(\mu_g)$$

Pokrytí značení μ_g existuje právě když je následující výraz dokazatelný v lineární logice.

$$S(\mu_0), !\left(\bigotimes I(t_1) \multimap \bigotimes O(t_1)\right), \ldots$$
$$\ldots, !\left(\bigotimes I(t_m) \multimap \bigotimes O(t_m)\right) \vdash \bigotimes S(\mu_g) \otimes \top$$

Nyní si všimněme, že jediný rozdíl v předchozích dvou výrazech je v použití konstanty $\top$. Použitím této konstanty oslabíme důkaz z rovnosti na inkluzi (tzn. můžou nám nějaké zdroje přebývat).

5 Kódování plánovacích problémů v lineární logice

Použití lineární logiky v plánování bylo studováno řadou výzkumníků [14,10,2]. V této kapitole bude uvedeno, jak lze kódovat plánovací problémy pomocí lineární logiky (budeme uvažovat plánování v prostoru stavů).

5.1 Základní kódování plánovacích problému v lineární logice

Z předchozí kapitoly víme, že v lineární logice můžeme poměrně jednoduše kódovat Petriho sítě. Petriho sítě samy o sobě dobře slouží k modelování plánovacích problémů, ale tím se v tomto článku zabývat nebudeme, využijeme pouze myšlenky kódování Petriho sítí v lineární logice při kódování plánovacích problémů v lineární logice.

Mějme stav $s = \{p_1, p_2, \ldots, p_n\}$ (stav chápeme jako množinu predikátů, které jsou v daném stavu pravdivé), jeho kódování v lineární logice vypadá následovně:

$$(p_1 \otimes p_2 \otimes \ldots \otimes p_n)$$

Mějme akci $a = \{p(a), e^-(a), e^+(a)\}$, kde $p(a)$ je prekondice (množina predikátů, které musí být pravdivé v daném stavu), $e^-(a)$ je množina negativních efektů (množinu predikátů, které po provedení akce nebudou platit), $e^+(a)$ je množina pozitivních efektů (množinu predikátů, které po provedení akce budou platit). Kódování akce a v lineární logice vypadá následovně:

$$\forall i \in \{1, 2, \ldots, k\}, l_i \in p(a) \cup e^-(a)$$

$$\forall j \in \{1, 2, \ldots, m\}, r_j \in e^+(a) \cup (p(a) - e^-(a))$$

$$(l_1 \otimes l_2 \otimes \ldots \otimes l_k) \multimap (r_1 \otimes r_2 \otimes \ldots \otimes r_m)$$

Toto znamená, že predikáty na levé straně implikace nebudou pravdivé po provedení akce a a predikáty na pravé straně budou pravdivé po provedení akce a. Celý plánovací problém může být zakódován následujícím způsobem ($a_1, a_2 \ldots a_m$ jsou substituce za kódování všech akcí v plánovacím problému, kde $s_0 = \{p_{0_1}, p_{0_2}, \ldots, p_{0_n}\}$ je počáteční stav a $g = \{g_1, g_2, \ldots, g_q\}$ je množina cílových predikátů):

$$p_{0_1}, p_{0_2}, \ldots, p_{0_n}, !a_1, !a_2, \ldots !a_m \vdash g_1 \otimes g_2 \otimes \ldots \otimes g_q \otimes \top$$

Plán existuje tehdy a jen tehdy, je-li předchozí výraz dokazatelný v lineární logice.

5.2 Příklad

V tomto příkladě budeme využívat predikátové rozšíření lineární logiky, které funguje stejně jako u 'klasické' logiky. Představme si verzi 'světa kostek', ve kterém máme palety a kostky a na každé paletě může být nejvýše jedna kostka. Máme rovněž jeřáb, který může v jednom okamžiku nést nejvýše jednu kostku.

Počáteční stav: 3 palety (1,2,3), 2 kostky (a, b), prázdný jeřáb, kostka a na paletě 1, kostka b na paletě 2, volná paleta 3.

Akce:

$ZVEDNI(Kostka, Paleta) = \{$

$$p = \{prazdnyjerab, na(Kostka, Paleta)\},$$
$$e^- = \{prazdnyjerab, na(Kostka, Paleta)\},$$
$$e^+ = \{nese(Kostka), volna(Paleta)\}$$
$$\}$$

$POLOZ(Kostka, Paleta) = \{$

$$p = \{nese(Kostka), volna(Paleta)\},$$
$$e^- = \{nese(Kostka), volna(Paleta)\},$$
$$e^+ = \{prazdnyjerab, na(Kostka, Paleta)\}$$
$$\}$$

Cíl: Kostka a na paletě 2, kostka b na paletě 1.

Kódování akcí $ZVEDNI(Kostka, Paleta)$ a $POLOZ(Kostka, Paleta)$ vypadá následovně:

$ZVEDNI(Kostka, Paleta)$:
$prazdnyjerab \otimes na(Kostka, Paleta) \multimap$
$\quad \multimap nese(Kostka) \otimes volna(Paleta)$

$POLOZ(Kostka, Paleta)$:
$nese(Kostka) \otimes volna(Paleta) \multimap$
$\quad \multimap prazdnyjerab \otimes na(Kostka, Paleta)$

Celý plánovací problém zakódovaný v lineární logice vypadá takto:

$$na(a, 1), na(b, 2), volna(3), prazdnyjerab,$$
$$!ZVEDNI(X, Y), !POLOZ(X, Y) \vdash$$
$$\vdash na(b, 1) \otimes na(a, 2) \otimes \top$$

5.3 Kódování možných optimalizací

Doposud zde bylo popsáno pouze čisté kódování plánovacích problémů v lineární logice. Nevýhoda čistého kódování je jeho malá efektivita při výpočtu. V následujících odstavcích nastíním ideje kódování některých optimalizací.

První optimalizací je blokování akcí. Některé akce, o kterých víme, že nepovedou k cíli, můžeme zablokovat, čímž výrazně snížíme počet prohledávání. Typickým příkladem jsou navzájem inverzní akce (tj. pokud provedeme takové akce za sebou, ocitneme se znovu ve stavu, ve kterém jsme byli před jejich provedením). Abychom mohli otestovat, zda akce a není zablokovaná, musíme do prekondice této akce přidat predikát $lze(a)$. Chceme-li akci a zablokovat v nějakým stavu s, musíme zajistit, aby stav s neobsahoval predikát $lze(a)$. Toto můžeme zjistit tak, že rozšíříme množinu negativních efektů předchozí akce o predikát $lze(a)$. Chceme-li v budoucnu akci odblokovat je nutné predikát $lze(a)$ přidat do pozitivních efektů nějaké akce.

Další optimalizací může být skládání akcí. Tato optimalizace je výhodná hlavně tehdy, pokud se v plánu vícekrát vyskytuje stejná sekvence akcí (jen s jinými proměnnými). Předpokládejme, že akce a_1 a a_2 jsou zakódovány v lineární logice následujícím způsobem:

$$a_1 : \quad (p_1 \otimes p_2) \multimap (p_1 \otimes p_3)$$
$$a_2 : \quad (p_3 \otimes p_4) \multimap (p_3 \otimes p_5)$$

Pak zakódování akce a, která vznikne složením akcí a_1, a_2 (v tomto pořadí), bude následující:

$$(p_1 \otimes p_2 \otimes p_4) \multimap (p_1 \otimes p_3 \otimes p_5)$$

Je zřejmé, že použitím těchto optimalizací snížíme počet prohledávání a tím výrazně urychlíme výpočet.

6 Lineární logické programování

Lineární logické programování můžeme chápat jako nový obor, který rozšiřuje metody 'klasického' logického programování o podporu lineární logiky. Mezi nejznámější a nejvýznamnější lineární logické programovací jazyky patří: Lolli [8,7], LLP [1], Lygon [17], LTLL[2]. LTLL a LLP patří v této době mezi nejefektivnější jazyky lineárního logického programování. Testy jednoznačně prokázaly, že použití jazyků lineárního logického programování v problémech kódovaných v lineární logice je o poznání efektivnější než 'klasické' logické programování.

Konstrukce lineárních logických programovacích jazyků a jejich interpreterů je popsána v [1,8]. Největší nevýhodou těchto jazyků je fakt, že tyto jazyky nejsou schopny zdroje vytvářet (jen je spotřebovávají), což znamená, že pomocí těchto jazyků jsme schopni pouze řešit problémy jako Hamiltonova kružnice v grafu, či tah jezdce, ale nejsme schopni řešit problém dosažitelnosti značení v Petriho sítích či plánovací problémy.

[2] Vyvinut na UP Olomouc, RNDr. Arnoštem Večerkou (`http://www.inf.upol.cz/vecerka`)

7 Emulace lineární logiky v Prologu

V mé diplomové práci [3] jsem navrhl lineární logický programovací jazyk SLLL, který byl zkonstruován jako překladač do Prologu. Hlavní idea, kterou jsem při konstrukci použil, byla taková, že lineární fakty mohly být uloženy ve speciálním seznamu. Aby tento seznam byl během výpočtu konzistentní, bylo zapotřebí přidat ke každému pravidlu, které mělo co do činění z lineárními fakty, dva parametry. První parametr reprezentoval seznam lineárních faktů, který vstupoval do pravidla. Druhý parametr reprezentoval seznam lineárních faktů, který vystupoval z pravidla. Výhoda tohoto přístupu je jednoduchost implementace a odolnost vůči backtrackingu. Nevýhodou tohoto přístupu je nízká efektivita výpočtu.

Přestože jazyk SLLL, obdobně jako dříve zmíněné jazyky lineárního logického programování, není schopen zdroje (lineární fakty) vytvářet, dá se idea seznamu lineárních faktů využít při emulaci lineární logiky v Prologu. Spotřebovávání zdrojů vyřešíme tak, že odstraníme příslušný lineární fakt ze seznamu. Přidávání zdrojů vyřešíme tak, že přidáme příslušný lineární fakt do seznamu. Na dalších řádcích bude uvedena emulace jen těchto operátorů $(\otimes, \&, \oplus, \multimap)$, ostatní operátory lineární logiky mají nízkou praktickou využitelnost. Emulace multiplikativní konjunkce $\otimes$ je velmi snadná, neboť stačí místo ní použít klasickou konjunkci, která je přítomná v Prologu. Additivní konjunkce & a additivní disjunkce $\oplus$ se dá snadno emulovat pomocí klasické disjunkce, rovněž přítomné v Prologu, jen v případě, že se additivní disjunkce $\oplus$ nachází na levé straně implikace a additivní konjunkce & na pravé straně implikace.[3] Emulace lineární implikace je rovněž snadná, stačí si jen uvědomit, že na levé straně implikace zdroje spotřebováváme a na pravé straně implikace zdroje vytváříme.

Modifikace předchozího přístupu je ta, že seznam lineárních faktů udržujeme setříděný. Výhoda tohoto přístupu spočívá v tom, že pokud z tohoto seznamu odebíráme více lineárních faktů naráz, stačí nám pouze jeden průchod seznamu. Nevýhoda tohoto přístupu spočívá v přidávání lineárních faktů do seznamu, které již nemůžeme provádět v konstantní složitosti. Další nesporná výhoda tkví v tom, že setříděné seznamy se dají porovnat v lineární složitosti, což například můžeme využít k detekci cyklů (tj. pokud po provedení několika kroků obdržíme stejný seznam lineárních faktů, který jsme měli před jejich provedením).

[3] Pokud bychom použili additivní konjunkci na levé straně implikace, či additivní disjunkci na pravé straně implikace, museli bychom výpočet rozdělit do více větví a najít řešení pro každou z těchto větví. Tento postup je nejen komplikovaný a časově náročný, ale téměř bez nemá praktické použití.

8 Další výzkum

Ve svém dalším výzkumu se zaměřím na možnosti využití lineární logiky při kódování různých problémů a optimalizací. Dále se zaměřím na možnosti využití stávajících lineárních logických programovacích jazyků při řešení těchto problémů a možnosti efektivní emulace lineární logiky v Prologu. Následující odstavce představí mé plány budoucího výzkumu detailněji.

8.1 Kódování problémů a optimalizací v lineární logice

V předchozích kapitolách jsem uvedl několik problémů a jejich kódování v lineární logice. Hlavně plánovací problémy se dají dobře zakódovat v lineární logice. Bohužel čisté kódování (bez optimalizací) plánovacích problémů není efektivní. Použitím výše zmíněných optimalizací dosáhneme nezanedbatelně větší efektivity výpočtu. V budoucnu se tedy zaměřím na to, jak zakódovat více optimalizací v lineární logice, které nám pomohou získat mnohem větší efektivitu výpočtu.

Další problémy, které jsem v článku nezmínil, jsou rozvrhovací problémy. V rozvrhovacích problémech mapujeme nějaké aktivity na zdroje (které tyto aktivity můžou vykonávat) za určitých podmínek. Věřím, že lineární logika bude dobrým formalismem i pro rozvrhovací problémy a proto budu studovat možnosti jejího využití na tyto problémy.

8.2 Použití lineárních logických programovacích jazyků

Jak bylo uvedeno dříve, výhodou jazyků lineárního logického programování je vysoká efektivita výpočtu. Bohužel tyto jazyky nejsou dostatečně silné pro řešení takových problémů, ve kterých během výpočtu přidáváme zdroje (např. plánovací problémy). Na druhou stranu věřím, že tyto jazyky budou využitelné při řešení rozvrhovacích problémů, či jako podpůrné nástroje při řešení plánovacích problémů.

8.3 Emulace lineární logiky v Prologu

Jak bylo uvedeno dříve, emulace lineární logiky (hlavně její potřebné části) v Prologu se dá udělat poměrně jednoduchým způsobem. Bohužel tento přístup, i když je korektní, není zatím příliš efektivní. Nejlepších výsledků zatím dosahuje metoda použití setříděných seznamů lineárních faktů s detekcí cyklů, ale tato metoda má přílišnou spotřebu paměti. Na druhou stranu jsem přesvědčen, že kdyby se povedlo vytvořit knihovnu v Prologu, která by nám zajišťovala efektivní práci s lineární logikou, výrazně by to pomohlo při řešení problémů s omezeným počet zdrojů.

8.4 Použití temporální logiky

Možné rozšíření lineární logiky je přidání temporální Logiky. temporální logika je sama o sobě schopna kódovat problémy, ve kterých se vyskytuje časové uspořádání akcí. Nicméně spojitost temporální logiky s lineární logikou nám zajistí větší efektivitu při výpočtech řešení problémů. Již existují lineární logické programovací jazyky s podporou temporální logiky: LLTL (již bylo zmíněno) a TLLP (rozšíření již zmíněného LLP).

8.5 Lehká lineární logika

lehká lineární logika je modifikací lineární Llogiky, více v [5]. Motivací pro vznik lehké lineární logiky bylo snížení složitosti v dokazování oproti standardní lineární logice. V budoucnu se rovněž budu snažit zjistit více o této variantě lineární logiky a její využitelnosti v kódování problémů.

9 Závěr

Tento článek ukázal sílu lineární logiky při modelování problémů s omezeným počtem zdrojů (např. plánovací problémy). Výhoda tohoto přístupu tkví v tom, že pokud dojde k vylepšení řešiče lineární logiky, automaticky se nám zvedne efektivita řešených problémů kódovaných v lineární logice. Přestože současné řešiče (lineární logické programovací jazyky) nejsou dostatečně silné na to, aby vyřešili plánovací problémy, věřím, že povede-li se efektivně emulovat lineární logiku v Prologu, budeme schopni dosáhnout vysoké efektivity při řešení problémů kódovaných v lineární logice.

Poděkování

Výzkum je podporován Grantovou Agenturou Karlovy Univerzity (GAUK) pod číslem 326/2006/A-INF/MFF.

Reference

1. Banbara M., Design and Implementation of Linear Logic Programming Languages. Ph.D. Dissertation, The Graduate School of Science and Technology, Kobe University, 2002
2. Chrpa L., Linear Logic in Planning. To appear at Doctoral Consorcium ICAPS 2006, 2006
3. Chrpa L., Linearni logika. Master's thesis, Department of Computer Science, Palacky University, Olomouc, 2005, (in Czech)
4. Girard J.-Y., Linear Logic. Theoretical Computer Science, 50, 1987, 1–102
5. Girard J.-Y., Light Linear Logic. Technical Report, Institute de Mathematiques de Luminity, 1998
6. Girard J.-Y., Linear Logic: Its Syntax and Semantics. Technical Report, Institute de Mathematiques de Luminity, 1995
7. Hodas J., Lolli: An Extension of Lambdaprolog with Linear Logic Context Management. Proceedings of the 1992 Workshop on the lambdaProlog Programming Language, 1992
8. Hodas J., Logic Programming in Intuitionistic Linear Logic: Theory, Design, and Implementation. Ph.D. Dissertation, University of Pennsylvania, Department of Computer and Information Science, 1994
9. Kanovich M., The Multiplicative Fragment of Linear Logic is NP-Complete. Technical Report X-91-13, Institute for Language, Logic and Information, 1991
10. Kanovich M., and Vauzeilles J., The Classical ai Planning Problems in the Mirror of Horn Linear Logic: Semantics, Expressibility, Complexity. Mathematical Structures in Computer Science 11(6), 2001
11. Küngas P., Linear Logic for Domain-Independent ai Planning. Proceedings of Doctoral Consorcium ICAPS 2003, 2003
12. Lincoln P., Linear Logic. Proceedings of SIGACT 1992, 1992
13. Lincoln P., Mitchell J., Scedrov M., and Shankar N., Decision Problems for Propositional Linear Logic. Technical Report SRI-CSL-90-08, CSL, SRI International, 1990
14. Masseron M., Tollu C., and Vauzeilles J., Generating Plans in Linear Logic i-ii. Theoretical Computer Science, 1993
15. Oliet N.M., and Meseguer J., From Petri Nets to Linear Logic. Springer LNCS 389, 1989
16. Reisig W., Petri Nets, An Introduction. Springer Verlag, Berlin, 1985
17. Winikoff M., Hitch Hiker's Guide to Lygon 0.7. Technical Report 96/36, The University of Melbourne, Australia, 1996

Asociativní paměti pro ukládání korelovaných vzorů*

Jana Štanclová

Katedra softwarového inženýrství
Matematicko-fyzikální fakulta Univerzity Karlovy v Praze
jana.stanclova@mff.cuni.cz

Abstrakt. *V našem výzkumu jsme se zaměřili na problematiku ukládání a vybavování velkého množství korelovaných dat. Pro tuto oblast lze použít modely asociativních neuronových sítí. Standardní model asociativní paměti má robustní vybavovací schopnosti, avšak jeho nedostatkem je relativně malá kapacita a požadavek na ortogonalitu ukládaných dat. V praxi je nutné zpracovávat velké množství dat a proto jsme se zaměřili na modely složené z více asociativních pamětí, které jsou uspořádány do více vrstev – tzv. HAM modely. Náš původní model HAM1 jsme vylepšili a navrhli model HAM2, kde asociativní paměti vytvářejí stromovou strukturu. Každá vrstva je rozdělena do disjunktních skupin asociativních pamětí a každé skupině odpovídá jedna asociativní paměť v předchozí vrstvě. Článek porovnává navržené modely HAM1 i HAM2 a diskutuje získané experimentální výsledky s ohledem na zpracování velkého množství korelovaných dat.*

1 Úvod

Představme si situaci, že cestujeme krajinou, kde jsme již předtím byli. Na konkrétním místě obvykle vidíme jen nejbližší okolí. Pokud se pohybujeme v již známém prostředí, pak jsme na základě scenérie okolo nás schopni vybavit si krajinu, kterou zatím nevidíme, ale víme, že bude za chvíli následovat (ve směru našeho dalšího putování). Jsme tedy schopni vybavit si např. co je za nejbližším kopcem, co je za nejbližší zatáčkou atd. Na základě vybavené krajiny můžeme vybavovat vzdálenější krajinu.

Pro proces vybavování krajiny ve směru dalšího pohybu je možné použít princip navržený Fukushimou a spol. [2]. Nejbližší okolí, které z daného místa vidíme, představuje aktuální vzor (stojíme v jeho středu). Při pohybu se aktuální vzor posune tak, abychom byli opět ve středu aktuálního vzoru. Posunutím vzoru podle směru našeho pohybu se v novém aktuálním vzoru objeví prázdná oblast. Prázdná část odpovídá krajině, kterou jsme předtím nemohli vidět (např. byla skryta za zatáčkou). Takto vytvořený vzor (tzv. neúplný vzor) je předložen asociativní paměti k vybavení. Vybavený vzor by měl mít doplněnou prázdnou oblast. Jiné přístupy k vybavování krajiny mohou být založeny např. na kognitivních mapách [9].

* Tento výzkum je podporován Výzkumným záměrem 0021620838: Moderní metody, struktury a systémy informatiky.

Pro jednoduché vybavování krajiny je možné použít model standardní asociativní paměti [5]. Tento základní model však není příliš použitelný při vybavování rozsáhlejší krajiny a to hned ze dvou zásadních důvodů. Prvním důvodem je relativně malý počet vzorů, které lze do modelu uložit a následně je správně vybavit. Druhým důvodem je fakt, že není zaručeno správné uložení a vybavování korelovaných vzorů. Proto jsme náš výzkum v oblasti neuronových sítí zaměřili na modely Hierarchických asociativních pamětí (HAM), které jsme navrhovali s ohledem na zpracování velkého množství i korelovaných vzorů. Tyto modely se skládají z libovolného počtu asociativních pamětí uspořádaných do několika vrstev. Náš původní model HAM1 (popsán např. v [7]) jsme rozšířili do modelu HAM2, kde asociativní paměti jednotlivých vrstev jsou rozděleny do disjunktních skupin a vytvářejí stromovou strukturu. Model HAM2 využívá informaci vybavenou v předchozí vrstvě k nalezení asociativní sítě ve vrstvě následující. Očekáváme, že model HAM2 bude mít lepší vybavovací schopnosti v porovnání původním modelem HAM1 a také se standardním modelem asociativní paměti.

2 Modely asociativní paměti

Standardní model asociativní paměti [5] je model neuronové sítě, kde jsou všechny neurony současně vstupní i výstupní. Neurony jsou mezi sebou propojeny pomocí orientovaných vazeb. Všechny váhy vazeb jsou symetrické a každý neuron je spojen se všemi ostatními neurony. Výstupem sítě je vektor výstupních hodnot všech neuronů asociativní paměti. Váhová matice W asociativní paměti s n ($n > 0$) neurony je matice $W = (w_{ij})$ velikosti $n \times n$, kde w_{ij} značí váhu mezi neuronem i a j.

Pro učení asociativní paměti lze použít tzv. Hebbovské učení. Na počátku procesu jsou váhy nastaveny na nulové hodnoty. Po předložení trénovacího vzoru $\boldsymbol{x}^k = (x_1^k, \ldots, x_n^k)$ dojde k adaptaci vah w_{ij} podle vztahu: $w_{ij} \leftarrow w_{ij} + x_i^k x_j^k \;\; i, j = 1, \ldots, n \;\; i \neq j$. Vybavování asociativní sítě je iterativní proces. V každé iteraci je vybrán neuron, který aktualizuje svůj stav (podle znaménka potenciálu neuronu). Neuron, který nebyl vybrán k aktualizaci, svůj stav nemění. Lze ukázat, že asociativní paměť s asynchronní dynamikou

(tj. výběr neuronu k aktualizaci náhodný a nezávislý) konverguje k lokálnímu minimu energetické funkce [5].

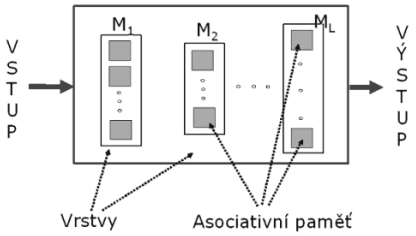
Asociativní paměť představuje základní model, který je možné použít pro vybavování uložených vzorů. Síť je schopna vybavit i vzory, které jsou „poničené“ nebo „neúplné“. Nedostatkem asociativní sítě je její relativně malá kapacita (přibližně $0.15n$, kde n je dimenze ukládaných vzorů [5]) a požadavek na ortogonalitu ukládaných vzorů. Vzory jsou ortogonální, pokud jejich skalární součin je nulový. V opačném případě nazveme vzory korelované. Ukládání korelovaných vzorů může způsobit problémy a dojde pravděpodobně ke ztrátě některých dříve uložených vzorů, protože tzv. crosstalk (součet skalárních součinů vzorů s ostatními uloženými vzory) je nenulový [1].

Existuje hodně modelů asociativní paměti, které se snaží rozšířit standardní model asociativní paměti s ohledem na zpracování korelovaných vzorů (např. [3], [4], [6]). Gutfreund [3] navrhl model, který se skládá ze dvou asociativních pamětí - pro každou úroveň hierarchie jedna asociativní paměť. První asociativní paměť (AM1) ukládá vzory první úrovně (tzv. vzory předků). Druhá asociativní paměť (AM2) pracuje se vzory druhé úrovně (tzv. vzory potomků). Gutfreundův model umí zpracovávat hierarchicky korelované vzory, ale jeho kapacita závisí na parametru, který představuje největší korelaci mezi vzory potomků [3]. Hirahara a spol. [4] navrhli model Kaskádové asociativní paměti (CASM), který je svojí strukturou velmi podobný Gutfreundovu modelu. Rozdíl spočívá v ukládání tzv. rozdílových vzorů do AM2 místo vzorů potomků. S rostoucí korelací vzorů potomků jsou rozdílové vzory „řidší“ a zvyšují kapacitu modelu CASM. Bohužel vlastnosti celého modelu CASM jsou dány vlastnostmi sítě AM1 a tedy celková kapacita sítě zůstává omezena kapacitou asociativní paměti AM1 ($\sim 0.15n$) [4].

3 Model Hierarchické asociativní paměti

Standardní asociativní paměť má robustní vybavovací schopnosti, pokud počet vzorů uložených v síti je relativně malý a uložené vzory jsou ortogonální (nebo téměř ortogonální). V praxi je nutné pracovat s velkým množstvím dat, která jsou téměř vždy korelovaná. Abychom se vyhnuli těmto omezením a bylo možné pracovat s libovolným počtem i korelovaných vzorů, navrhli jsme model tzv. Hierarchické asociativní paměti (HAM). Model HAM byl inspirován Gutfreundovým modelem [3] a modelem Kaskádové asociativní paměti [4]. Bohužel oba tyto modely jsou tvořeny jen dvěma asociativními paměťmi ve dvou vrstvách. Z tohoto důvodu jsme se rozhodli navrhnout model HAM, který bude využívat většího počtu asociativních pamětí uskupených do vrstev. Tímto způsobem bude možné uložit téměř libovolný počet vzorů a požadavek na ortogonalitu vzorů nebude nutný. Korelované vzory budou uloženy do různých asociativních pamětí na téže vrstvě.

Hierarchická asociativní paměť H s L ($L > 0$) vrstvami je uspořádaná L–tice $H = (M_1, \ldots, M_L)$, kde $M_1, \ldots, M_L$ jsou konečné neprázdné množiny asociativních pamětí (tzv. lokální asociativní paměti); každá z lokálních asociativních pamětí má stejný počet neuronů $n (n > 0)$. Množina M_k $(k = 1, \ldots, L)$ se nazývá vrstva hierarchické paměti H. $|M_k|$ značí počet lokálních asociativních pamětí ve vrstě M_k $(k = 1, \ldots, L)$. Trénovací množina T hierarchické asociativní paměti H je uspořádaná L–tice $T = (T_1, \ldots, T_L)$, kde T_k $(k = 1, \ldots, L)$ je konečná neprázdná množina trénovacích vzorů pro vrstvu M_k. Základní struktura HAM–modelů (HAM1 i HAM2) je zobrazena na obrázku 1. V dalším textu se detailně zaměříme na strukturu a činnost modelu HAM2. Popis modelu HAM1 je možné najít např. v [7].

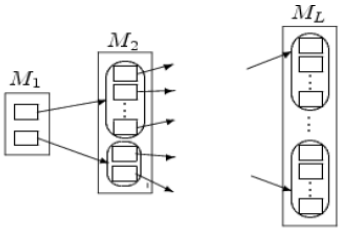


Obrázek 1. Struktura Hierarchické asociativní paměti.

3.1 Model HAM2

Základní odlišnost modelu HAM2 od modelu HAM1 spočívá ve stromové struktuře lokálních asociativních pamětí. Lokální asociativní paměti každé vrstvy jsou uspořádány do disjunktních skupin. Každá skupina lokálních asociativních pamětí přísluší jedné asociativní paměti v předchozí vrstvě. Model HAM2 sdružuje asociativní paměti do skupin podle informace v předchozí vrstvě. Stromová struktura modelu HAM2 je zachycena na obrázku 2.

Pro učení modelu HAM2 jsme navrhli tzv. dynamický algoritmus učení. Jednotlivé vrstvy $M_1, ..., M_L$ jsou učeny postupně. Při učení vrstev se využívají již naučené předchozí vrstvy. Trénovací vzory z množiny T_k $(k = 1, ..., L)$ jsou uloženy do lokálních asociativních sítí vrstvy M_k. Během učení vrstvy M_k $(k = 1, ..., L)$ jsou trénovací vzory vybírány z příslušné trénovací množiny T_k sekvenčně a předkládány vrstvě M_k.



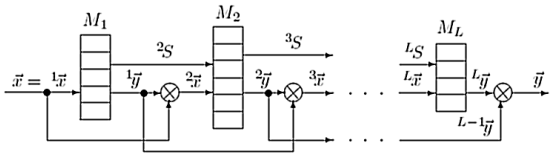
Obrázek 2. Stromová struktura modelu HAM2 s L vrstvami $M_1, \ldots, M_L$.

Učení jednoho trénovacího vzoru $\boldsymbol{x}$ z trénovací množiny T_k vrstvy M_k se skládá ze dvou kroků. Nejprve je vzor $\boldsymbol{x}$ vybaven již naučenými vrstvami $M_1, \ldots, M_{k-1}$ (podle algoritmu vybavování popsaného dále v textu). Výstup poslední dosud naučené vrstvy M_{k-1} poskytne informaci, do které skupiny vrstvy M_k trénovací vzor $\boldsymbol{x}$ patří. Nalezenou skupinu ve vrstvě M_k označíme ${}^kS$. Vzor $\boldsymbol{x}$ bude uložen v některé z lokálních asociativních pamětí skupiny ${}^kS$ vrstvy M_k. V druhém kroku probíhá již vlastní uložení vzoru $\boldsymbol{x}$ do lokální asociativní paměti skupiny ${}^kS$. Vzor $\boldsymbol{x}$ je uložen do takové lokální asociativní paměti skupiny ${}^kS$ vrstvy M_k, kde vzor $\boldsymbol{x}$ (resp. jeho „zašuměný“[1] obraz) bude po naučení správně vybaven. Je-li nalezena požadovaná lokální asociativní paměť, vzor $\boldsymbol{x}$ je uložen do této lokální sítě. Není-li nalezena žádná taková lokální asociativní paměť, dojde k dynamickému vytvoření nové lokální asociativní paměti. Nově vytvořená lokální asociativní paměť se přidá do skupiny ${}^kS$ vrstvy M_k a vzor $\boldsymbol{x}$ je uložen do nově přidané sítě skupiny ${}^kS$.

Během procesu vybavování je na vstup modelu HAM2 předložen vzor $\boldsymbol{x}$. Předložený vzor $\boldsymbol{x}$ je vstupním vzorem první vrstvy M_1 (tj. $\boldsymbol{x} = {}^1\boldsymbol{x}$). V každém kroku k ($1 \leq k \leq L$) je vrstvě M_k předložen vzor ${}^k\boldsymbol{x}$ a informace o množině ${}^kS$, kam vzor ${}^k\boldsymbol{x}$ ve vrstvě M_k patří (v případě první vrstvy množina ${}^1S$ zahrnuje všechny asociativní paměti první vrstvy). Poté začne proces vybavování vzoru ${}^k\boldsymbol{x}$ ve vrstvě M_k. Po skončení procesu vybavování ve vrstvě M_k je výstupem vrstvy vzor ${}^k\boldsymbol{y}$, který představuje odezvu vrstvy na předložený vstup ${}^k\boldsymbol{x}$. Výstup ${}^k\boldsymbol{y}$ je zkombinován s výstupem ${}^{k-1}\boldsymbol{y}$ z předchozí vrstvy M_{k-1} a vytvoří se vstup ${}^{k+1}\boldsymbol{x}$ do další vrstvy M_{k+1}. Výstup poslední vrstvy M_L zkombinován s výstupem ${}^{L-1}\boldsymbol{y}$ vrstvy M_{L-1} představuje výstup $\boldsymbol{y}$ celého modelu. Schéma procesu vybavování je zobrazeno na obrázku 3. Symbol „$\otimes$“ v obrázku 3 reprezentuje funkci, která vytvoří vstup do další vrstvy (v naší implementaci modelu HAM2 je použita funkce výpočtu rozdílového vzoru).

Nyní popíšeme proces vybavování vzoru ${}^k\boldsymbol{x}$ ve vrstvě M_k detailněji. Výstup z předchozí vrstvy

[1] vzor, ve kterém náhodně vybrané prvky změní svoji hodnotu

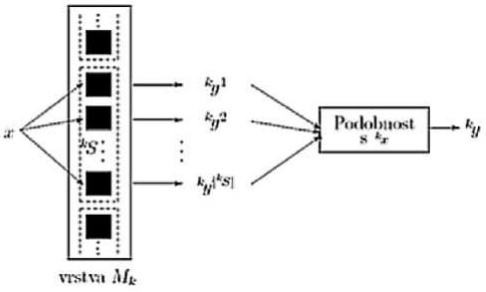


Obrázek 3. Proces vybavování v modelu HAM2.

M_{k-1} poskytne informaci, do které skupiny ${}^kS$ vrstvy M_k vstupní vzor ${}^k\boldsymbol{x}$ patří. Poté je vstupní vzor ${}^k\boldsymbol{x}$ předložen všem lokálním asociativním sítím skupiny ${}^kS$, které spočtou svůj výstup ${}^k\boldsymbol{y}_i$ ($i = 1, \ldots, |{}^kS|$). Výstup ${}^k\boldsymbol{y}$ vrstvy M_k je definován jako výstup, který je „nejpodobnější“ vstupnímu vzoru ${}^k\boldsymbol{x}$, tj.

$$ {}^k\boldsymbol{y} = \max \{\text{podobnost}({}^k\boldsymbol{x}, {}^k\boldsymbol{y}_i) \;\; i = 1, \ldots, |{}^kS|\} $$

(obrázek 4). V experimentech používáme Hammingovu vzdálenost jako míru podobnosti vzorů, ale je možné použít i jiné míry podobnosti. Výstup ${}^k\boldsymbol{y}$ vrstvy M_k je zkombinován s výstupem předchozí vrstvy M_{k-1} a tak je vytvořen vstup do další vrstvy M_{k+1}.



Obrázek 4. Schéma vybavování ve vrstvě M_k v modelu HAM2.

3.2 Vlastnosti modelu HAM2

Model HAM2 je tvořen lokálními asociativními pamětmi, které jsou uspořádány do vrstev a skupin. V modelu HAM2 není nutné dopředu znát počet lokálních asociativních pamětí v jednotlivých vrstvách a skupinách. Během procesu učení jsou lokální asociativní paměti dynamicky přidávány do odpovídajících vrstev modelu. Na začátku učení může být v každé vrstvě jen jediná skupina s jedinou lokální asociativní sítí. Počáteční struktura modelu HAM2 tak tvoří cestu (degenerovaný strom). Ostatní lokální sítě jsou přidávány do jednotlivých vrstev dynamicky během procesu učení na základě zpracovávaných dat. Výsledný počet lokálních sítí i jejich uspořádání do skupin závisí na počtu a tvaru ukládaných dat. Před procesem učení modelu

HAM2 je nutné znát pouze počet vrstev (tj. znalost základní struktury dat).

Dynamický algoritmus učení modelu HAM2 používá jednoduchou heuristiku na nalezení lokální asociativní paměti pro uložení předloženého vzoru. Vzor zůstane uložen v lokální síti odpovídající skupiny, kde je právě ukládaný vzor (popř. jeho „zašuměný“ obraz) korektně vybaven. Při použití této metody učení však nejsme schopni nic říci o tom, jak se po uložení nového vzoru budou vybavovat dříve uložené vzory.

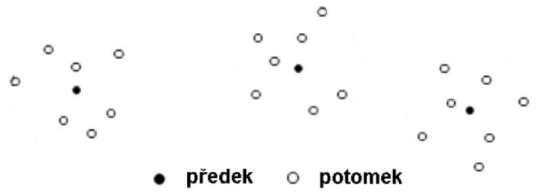
Prostorová složitost lokální asociativní paměti s n neurony odpovídá $n(n-1)/2$ (váhová matice velikosti $n \times n$ je symetrická s nulovou diagonálou). Maximální kapacita jedné lokální paměti v modelu HAM2 je omezena vztahem $q = 0.15 \cdot n \cdot c$ (viz Kapitola 4.1). Pro uložení p vzorů je potřeba p/q lokálních asociativních pamětí v modelu HAM2 při maximálním využití kapacity q každé lokální sítě.

Časová složitost procesu uložení jednoho vzoru do lokální asociativní paměti odpovídá změně každého prvku příslušné váhové matice (tj. $n(n-1)/2$). Proces vybavování v lokální asociativní paměti je iterativní proces a v nejhorším případě má exponenciální složitost [8]. Avšak v praxi obvykle proces vybavování konverguje velmi rychle [8]. Procesy vybavování v lokálních sítích téže vrstvy mohou běžet paralelně. Složitost procesu vybavování v modelu HAM2 je závislá na počtu vrstev a době vybavení v lokální paměti.

4 Experimentální výsledky

Jedna z nejdůležitějších vlastností neuronových sítí je jejich schopnost vybavovat. Při použití modelu HAM2 (např. při vybavování krajiny) jsou nezbytné robustní vybavovací schopnosti modelu s ohledem na: 1. vybavování uložených vzorů a 2. vybavování neúplných vzorů. Experimenty jsou zaměřeny na výše uvedené vlastnosti. Experimentální výsledky jsou srovnány s původním modelem HAM1 i standardním modelem asociativní paměti.

V našich experimentech se omezíme na dvouvrstvý model HAM2 (resp. HAM1) a tedy na dvouvrstvou hierarchii dat. Vzory předkládané první vrstvě jsou tzv. vzory předků. Vzory zpracovávané druhou vrstvou jsou tzv. vzory potomků. Důvodem pro volbu této hierarchie byla jednoduchá geometrická interpretace dvouvrstvé hierarchie dat. Můžeme předpokládat, že vstupní data jsou uskupena do shluků. Vzory předků mohou reprezentovat jednotlivé shluky (např. středy shluků) a vzory potomků pak vlastní data. Vzory potomků jednoho shluku jsou podobné (korelované) vzoru předka, který leží uprostřed tohoto shluku dat. Struktura dvouvrstvé hierarchie dat je zachycena na obrázku 5. Zobecnění struktury dat do libovolného



Obrázek 5. Dvouvrstvá hierarchie dat.

počtu vrstev je přímočaré, avšak geometrická interpretace vícevrstvé struktury dat nemusí být snadná (v budoucnu plánujeme provést experimenty i pro větší počet vrstev). Pro zlepšení rozpoznávacích schopností modelu druhá vrstva nepracuje přímo se vzory potomků ale s tzv. rozdílovými vektory. Rozdílové vzory obsahují pouze informaci o rozdílu mezi vzorem potomka a odpovídajícím vzorem předka.

Vzory předků jsou uloženy do první vrstvy modelu. Při učení vzorů potomků je pro každý vzor potomka nalezen jeho odpovídající vzor předka (např. vybavením v již naučené první vrstvě modelu HAM2). Na základě předloženého vzoru potomka a nalezeného vzoru předka je vytvořen rozdílový vzor a uložen do druhé vrstvy modelu. Druhá vrstva modelu HAM2 je tvořena skupinami lokálních asociativních pamětí. Každá skupina je odpovědná za vybavení těch jen rozdílových vzorů, které přísluší odpovídajícímu vzoru předka v první vrstvě.

Při vybavování je vstupní vzor předložen první vrstvě modelu HAM2. První vrstva vybaví odpovídající vzor předka (tj. detekuje shluk, do kterého vzor patří). Na základě vybaveného vzoru předka a předloženého vstupního vzoru je vytvořen rozdílový vzor. Tento rozdílový vzor je propagován do druhé vrstvy k vybavení. Současně se využije informace o vybaveném vzoru předka k nalezení odpovídající skupiny lokálních asociativních sítí v druhé vrstvě. Tato skupina lokálních sítí vybaví rozdílový vzor. Výstup modelu HAM2 je vytvořen kombinací vybaveného rozdílového vzoru v příslušné skupině druhé vrstvy a vybaveného vzoru předka (operace inverzní k operaci vytvoření rozdílového vzoru).

Pro experimenty bylo vytvořeno 100 sad náhodně vygenerovaných vzorů o rozměrech 15×15 prvků. Každá sada dat obsahuje 100 bipolárních vzorů (tj. vzory, které jsou tvořeny jen hodnotami $+1$ nebo -1). Každý experiment pracuje s jednou sadou dat (nezávisle na ostatních). V experimentech jsou testovány relativně malé vzory. Důvodem pro tuto volbu je nutnost provést velké množství experimentů, aby bylo možné analyzovat vlastnosti modelu HAM2 (v porovnání s modelem HAM1). Bylo provedeno také několik experimentů s většími daty (75×75) a výsledky byly srovnatelné. S opakovaným prováděním experimentů souvisela i volba náhodně generovaných vzorů. Náhodně

generované vzory jsou snadno dostupné a pro ověření vlastností modelů postačující.

I když jsou data náhodně vygenerována, je pravděpodobné, že vzory budou korelované (skalární součin dvou náhodně generovaných bipolárních vzorů typicky není nulový). Proto z každé sady dat jsou vybrány vzory s nejmenší kumulativní korelací s ostatními vzory a tyto vzory představují vzory předků. Zbylé vzory tvoří vzory potomků. Pro analýzu výsledků zavedeme parametr r, který určuje poměr počtu předků ku počtu potomků v dané sadě dat (např. $r = 1$ odpovídá sadě dat s 50 vzory předků a 50 vzory potomků).

4.1 Vybavování uložených vzorů

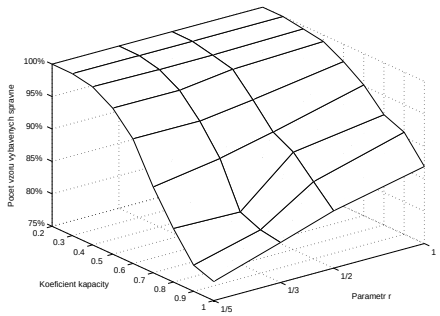
Nejprve analyzujeme schopnosti modelu HAM2 vybavit uložené vzory a výsledky porovnáme s původním modelem HAM1 i standardním modelem asociativní paměti. V prvním experimentu zjistíme procento vzorů, které jsou v modelu HAM2 uloženy správně (tj. správné vybavení uložených vzorů). Vzor je vybaven správně, pokud se zcela shoduje s původním vzorem (tj. žádný prvek vzoru není vybaven s chybou).

Definujeme tzv. kapacitní koeficient c redukující maximální počet vzorů, které je možné uložit do lokální asociativní paměti. Maximální kapacitu q lokální asociativní paměti definujeme vztahem:

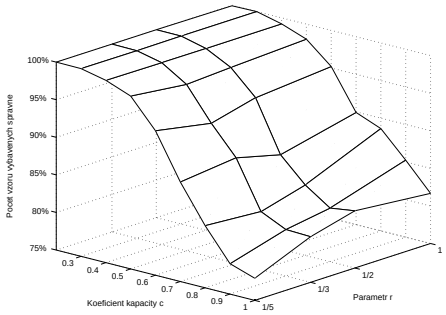
$$q = 0.15 \cdot n \cdot c.$$

Pro standardní asociativní paměť nabývá kapacitní koeficient c hodnoty 1. Se snižujícími se hodnotami koeficientu c klesá maximální kapacita lokálních sítí a zlepšují se schopnosti modelu rozpoznat uložené vzory. Získané výsledky modelu HAM2 jsou shrnuty na obrázku 6. Stejné experimenty jsou provedeny i pro původní model HAM1. Získané výsledky modelu HAM1 jsou zachyceny na obrázku 7.

Na obrázcích 6 i 7 je zobrazen průměrný počet vzorů, které jsou vybaveny správně. Pro $c \leq 0.5$ jsou



Obrázek 6. Procento korektně vybavených trénovacích vzorů modelem HAM2 s ohledem na kapacitní koeficient c a parametr r.



Obrázek 7. Procento korektně vybavených trénovacích vzorů modelem HAM1 s ohledem na kapacitní koeficient c a parametr r.

výsledky podobné v obou modelech (nezávisle na zvolené hodnotě parametru r) a více než 97% uložených vzorů je správně vybaveno. V těchto případech je kapacita jednotlivých lokálních asociativních pamětí relativně malá a k uložení všech trénovacích dat je potřeba většího počtu lokálních sítí. Když $c \leq 0.5$, stromová struktura modelu HAM2 způsobí nepatrné zlepšení vybavovacích schopností. S rostoucím koeficientem kapacity c je rozdíl mezi HAM2 a HAM1 více zřetelný. Větší hodnoty parametru r odpovídají většímu počtu vzorů předků a tedy většímu počtu lokálních asociativních pamětí v první vrstvě. Zde se využije stromová struktura modelu HAM2 a model má lepší vybavovací schopnosti (oproti modelu HAM1).

Stejné experimenty jsou provedeny i pro standardní model asociativní paměti. Protože počet ukládaných vzorů přesahuje teoretickou kapacitu modelu [5], standardní model asociativní paměti nevybaví žádný uložený vzor správně. Uložené vzory jsou navíc korelované a tak během učení dochází k poškození dříve ukládaných vzorů a následné nemožnosti rozpoznat uložené vzory.

4.2 Vybavování neúplných vzorů

V dalších experimentech se zaměříme na vybavovací schopnosti s ohledem na vybavování neúplných vzorů. Experimenty provedeme pro modely s kapacitním koeficientem $c = 0.7$ a parametrem $r = 1$. Vytvoříme tři skupiny neúplných vzorů, které obsahují 13%, 25% a 36% neznámých prvků ve vzoru. Velikost neznámé oblasti odpovídá diagonálnímu posunu vzoru o 1, 2 nebo 3 body (tj. každý prvek vzoru se posune o 1, 2 nebo 3 body v daném diagonálním směru). V experimentech uvažujeme 4 druhy neúplných vzorů podle směru posunu: $\searrow$, $\nearrow$, $\swarrow$ a $\nwarrow$. V praxi jsou možné i další směry. Experimenty jsou provedeny odděleně pro jednotlivé směry posunutí a výsledek odpovídá průměrné hodnotě pro danou velikost neznámé oblasti.

Analyzujeme vybavovací schopnosti modelu s ohledem na velikost neúplné oblasti vzoru. Se zvětšující

se neznámou oblastí vzoru klesá vybavovací schopnost modelu. V předchozích experimentech jsou uvažovány jen vzory, které jsou vybaveny zcela správně. Tento požadavek je příliš silný. V praxi nemusí být vzory vybaveny zcela správně (tj. malé procento chybně vybavených vzorů je přípustné). Výsledky vybavování neúplných vzorů v závislosti na velikosti posunutí a přípustné chybě jsou zachyceny na obrázku 8.

Přípustná chyba	Posun 1		Posun 2		Posun 3	
	HAM2	HAM1	HAM2	HAM1	HAM2	HAM1
0%	54%	53%	51%	50%	48%	47%
1%	59%	58%	53%	52%	51%	50%
2%	64%	63%	55%	54%	52%	50%
3%	68%	67%	57%	56%	53%	51%
4%	73%	72%	60%	59%	55%	52%
5%	76%	75%	62%	61%	56%	54%
...						
10%	100%	100%	72%	70%	63%	60%

Obrázek 8. Počet neúplných vzorů vybavených s přípustnou chybou.

V případě vybavování neúplných vzorů model HAM2 vykazuje lepší vybavovací schopnosti, avšak rozdíly jsou jen velmi malé. Když je přípustná chyba 5%, pak model HAM2 má relativně robustní vybavovací schopnosti (76%) pro 13% neznámých prvků. Všechny experimenty pro modely HAM2 i HAM1 byly provedeny opakovaně na jednotlivých sadách vzorů a sady dat byly pro oba modely vždy stejné. Tabulky a grafy zachycují střední hodnotu získaných výsledků (a tedy v sobě nesou variabilitu dat).

Stejné experimenty jsou provedeny i pro standardní asociativní paměť. Vzhledem k tomu, že standardní asociativní paměť není schopna vybavit uložené vzory správně, pak ani žádný neúplný vzor není vybaven správně. Pokud je přípustné malé procento chyby vybavení, pak některé vybavené fantomové vzory již padnou do přípustného procenta chyby.

Teorie říká, že s každým uloženým vzorem si asociativní paměť zapamatuje také jeho inverzní vzor. V našich experimentech tato skutečnost není nijak zohledňována. Tedy je možné, že některé neúplné vzory jsou vybaveny jako inverzní (a tedy jsou detetovány jako chybně vybavené). Pokud bychom detekovali tyto případy, pak by mohlo dojít k dalšímu zlepšení vybavovacích schopností modelu.

5 Závěr

Náš výzkum v oblasti neuronových sítí je zaměřen na zpracování velkého množství (korelovaných) vzorů. Náš původní model HAM1 jsme vylepšili a navrhli model HAM2, kde asociativní paměti vytvářejí stromovou strukturu a každá vrstva je rozdělena do disjunktních skupin asociativních pamětí. Provedené experimenty ukazují, že model HAM2 má relativně robustní vybavovací schopnost. V tomto článku byly porovnány vlastnosti modelu HAM2 s modelem HAM1 a se standardním modelem asociativní paměti. Nicméně pro reálné aplikace (např. rozpoznávání krajiny) je nutné dále zlepšit vybavovací schopnosti našich modelů HAM.

V současné době pracujeme na porovnání našich modelů také s modelem CASM (Hirahara a spol.) a případně s dalšími modely asociativních pamětí. Také je nezbytné provést experimenty pro větší počet vrstev a zjistit závislost sítě na velikosti korelace vzorů. Při ukládání vzorů požadujeme, aby po uložení dalšího vzoru nedošlo k poničení či zapomenutí dříve uložených vzorů. V současné implementaci byla použita jednoduchá heuristika, která však výše uvedený požadavek nezaručuje. Do budoucna je nutné vylepšit algoritmus učení a nalézt sofistikovanější metody učení, které dále zlepší robustnost modelu.

Literatura

1. D.J. Amit, H. Gutfreund, H. Sompolinsky, Information Storage in Neural Networks with Low Levels of Activity. Physical Review A, 35, 1987, 2293–2303
2. K. Fukushima, Y. Yamaguchi, M. Okada, Neural Network Model of Spatial Memory: Associative Recall of Maps. Neural Network, Vol. 10, No. 6, 1997, 971–979
3. H. Gutfreund: Neural Networks with Hierarchically Correlated Patterns. In: Physical Review A, Vol. 37, No. 2, 1988, 570–577
4. M. Hirahara, N. Oka, T. Kindo, A Cascade Associative Memory Model with a Hierarchical Memory Structure. Neural Networks, Vol. 13, No. 1, 2000, 41–50
5. J.J. Hopfield, Neural Networks and Physical System with Emergent Collective Computational Abilities. Proc. Natl. Acad. Sci. USA, 79, 1982, 2554–2558
6. N. Matsumoto, D. Ide, M. Watanabe, M. Okada, Synaptic Depression Enlarges Basin of Attraction. Neurocomputing, 65-66, 2005, 571–577
7. I. Mrázová, J. Tesková, Hierarchical Associative Memories for Storing Spatial Patterns. Proceedings of ITAT 2002, 2002, 143–154
8. J. Šíma, R. Neruda, Teoretické otázky neuronových sítí, Matfyzpress, 1997
9. H. Voicu, Hierarchical Cognitive Maps. In Neural Networks, Vol. 16, 2003, 569–576

ITAT'06

Information Technology – Applications and Theory

WORK IN PROGRESS

Segmentace vět pomocí šablon*

Tomáš Holan

KSVI MFF UK Praha
Tomas.Holan@mff.cuni.cz

Abstrakt *Jedním z problémů syntaktické analýzy vět přirozeného jazyka je její úspěšnost klesající s rostoucí délkou věty. Jedním ze způsobů, jímž se různí autoři pokoušejí tento problém řešit, je dělení věty na části-segmenty, které by bylo možné analyzovat odděleně.*
Tento text ukazuje způsob, jak pomocí šablon vět získat rozdělení vět na segmenty spolu s hranami tvořícími kostru věty. Text popisuje myšlenku a uvádí výsledky provedených experimentů.

1 Úvod

Jedním z problémů syntaktické analýzy vět přirozeného jazyka je to, že úspěšnost analýzy klesá s rostoucí délkou věty.

Vedle toho, že spolu s délkou věty roste počet možností, jak zapojit dané slovo, může být příčinou i *lokální* charakter informací v popisech jazyka. Slovo lokální přitom nevztahujeme jenom k povrchové vzdálenosti slov ve větě, ale i k vzdálenosti ve stromě měřené počtem hran.

Přitom nezáleží na tom, zda jsou tyto informace vytvářeny lidmi nebo extrahovány z korpusů; ať analyzátor používá gramatiku, informace o možných hranách, n-gramy nebo posloupnost operací vedoucí k vytvoření nebo přeskupení stromu, tyto informace vždy berou do úvahy jen určitou část analyzované věty a z toho plynou potíže při zapojování dlouhých hran.

Z podobných důvodů je těžké správné určení role symbolů, jako jsou například čárky.

Tyto problémy se někteří autoři ([3], [4]) snaží odstraňovat rozdělením věty pomocí význačných slov (delimiterů) na části-segmenty, které by bylo možno analyzovat odděleně. Problémem ovšem zůstává zjištění postavení jednotlivých segmentů v rámci věty, stejně jako více možných významů některých delimiterů (opět například čárka).

2 Pražský závislostní treebank

Věty uváděné jako příklady v tomto textu, a věty použité jako vstupní data popisovaných experimentů pocházejí z Pražského závislostního treebanku, PDT 2.0, `http://ufal.mff.cuni.cz/pdt2.0`.

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

Pražský závislostní treebank, byl vyvinut Ústavem formální a aplikované lingvistiky a Centrem komputační lingvistiky University Karlovy (viz `http://ufal.mff.cuni.cz/`
a `http://ckl.mff.cuni.cz/` nebo [1]).

3 Šablony vět

Náš přístup se pokouší řešit problém lokality znalostí o jazyku a zároveň problém segmentace a role jednotlivých segmentů ve větě. Ukazuje se navíc, že může otevírat i nové možnosti v syntaktické analýze vět.

Základní princip je podobný principu *island grammars* popsaných v [6]. Vycházíme z toho, že některá slova ve větě mají pro hledání struktury věty větší význam než ostatní. Takovým slovům budeme říkat *milníky* a v prvním kroku analýzy věty si jiných slov nebudeme všímat.

Která slova budou patřit mezi milníky, je otázka na další zkoumání, v současné době mezi milníky zahrnujeme slovesa, rozlišená druhým znakem morfologické značky (subPoS) a tím, zda jde o tvar slovesa „být“, dále interpunkční znaménka, závorky, uvozovky, spojky, slovo „se“ a slova uvozující vedlejší věty (když, že, pokud...).

Když z analyzované věty ponecháme pouze milníky, přičemž slovesa nahradíme prvními dvěma znaky morfologické značky a přidaným písmenem „b“ u tvarů slovesa „být“, získáme *šablonu věty*.

Například předchozí věta by měla šablonu (jednotlivé prvky oddělujeme znakem „–“):

Když–VB–,–přičemž–VB–a–”–”–”–Vfb–”–,–VB–.

Detekcí milníků získáme rozdělení vět na části, tato šablona nám navíc dává docela přesnou představu o postavení a roli jednotlivých segmentů (ve smyslu částí oddělených čárkami): rozpoznáváme větu hlavní v posledním segmentu, vedlejší větu na začátku a vsuvku uprostřed.

4 Kostřičky

Jdeme ovšem ještě dál a na trénovací části treebanku se naučíme, jak jsou u jednotlivých šablon milníky propojeny hranami.

Tato informace nemusí být jednoznačná, i když shoda je hodně vysoká. Během procházení vět trénovací množiny vždy určíme milníky, tím získáme šablonu věty a poznamenáme si způsob zapojení hran mezi milníky, pro každou šablonu zvlášť.

Po zpracování celé trénovací množiny jsou pro každou šablonu vět vyhodnoceny všechny pozorované způsoby zapojení milníků a je určena pro každý milník taková hrana k řídícímu uzlu, která se vyskytla nejvícekrát. Výslednému grafu říkáme *kostřička věty*.

Kostřička ovšem nemusí být strom, hrany, které ve zkoumaných větách trénovací množiny vedly jinam než do milníku, také evidujeme a pokud v závěrečném vybírání hran pro milník získaly převahu, poznamenáme si zvláštní hodnotu závislosti udávající, že pro tento milník nemáme žádnou hranu.

Pro každý milník potom můžeme spočítat, jaký je podíl četnosti vybrané nejčastější závislosti na všech závislostech pro tento milník vyskytujících se ve všech nalezených výskytech stejné šablony. Průměr takto zjištěných podílů nazýváme *míra jednoznačnosti* šablony a kostřičky.

Krom toho víme, kolikrát celkem se která šablona věty vyskytla v trénovací množině, tento údaj nazýváme *četnost* šablony.

Závislosti milníků dále budeme zapisovat jako řetězce obsahující na i-té pozici číslo milníku (v uvedených príkladech toto číslo nebude větší než 9) nadřízeného i-tému milníku, '0' pro milníky zavěšené na umělý kořen věty, případně '?', pokud pro tento milník nemáme hranu.

Například věta

V případě, že se nedovoláte přes den, vytočte číslo ve večerních nebo nočních hodinách a svůj dotaz namluvte na telefonní záznamník.

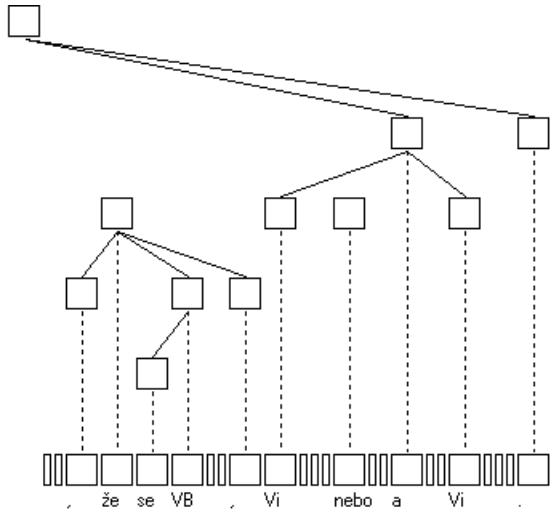
má šablonu

,–že–se–VB–,–Vi–nebo–a–Vi–.

a závislosti milníků

2?4228?080

Její kostřička by potom vypadala takto (čtverečky znázorňují milníky, zúžené obdélníky ostatní slova):



Obrázek 1. Kostřička šablony **,–že–se–VB–,–Vi–nebo–a–Vi–.**

5 Co dělat s kostřičkami, když je máme

Veškeré použití kostřiček můžeme parametrizovat požadavky na *míru jednoznačnosti* a *četnost* a tím řídit volbu mezi recall a precission.

5.1 Hrany kostřiček

Nejjednodušší možnost použití šablon vět a jejich kostřiček nijak nesouvisí s myšlenkou segmentů a spočívá prostě v tom, že hranami z kostřiček budeme přepisovat výsledky jiného analyzátoru.

Úspěšnost zapojení hran kostřiček je poměrně velká (viz dále) a tak máme naději, že úspěšnost analyzátoru i tímto jednoduchým způsobem zlepšíme. K syntaktickému analyzátoru tedy přidáme krok, v němž hranami ze známé kostřičky nahradíme hrany se stejným závislým členem.

Dosavadní výsledky:

Popsaný postup jsme zkusili na výstupech tří analyzátorů: náš history-based zásobníkový analyzátor popsaný v [2] jako R2L, nestatistický Žabokrtského analyzátor také popsaný v [2] a McDonaldův statistický analyzátor popsaný v [5].

Pokud jenom u uzlů, které mají svůj protějšek v kostřičce, nahradíme údaj o jejich řídícím členu údajem z kostřičky (to znamená že neřešíme, zda výsledné hrany tvoří strom!), změní se úspěšnost analyzátorů tak, jak ukazuje Tabulka 1.

	Holan	Žabokrtský	McDonald
původní úspěšnost	73.99%	76.06%	84.24%
po opravě	74.45%	76.13%	84.27%
zlepšení	+0.46%	+0.07%	+0,03%

Tabulka 1. Zlepšení úspěšnosti analyzátorů prostým převzetím hran z kostřiček.

5.2 Střecha

Další možnost použití kostřiček je sofistikovanější. Její základní myšlenkou je, že známé hrany, konkrétně ty připojené až ke společnému předkovi, omezují množinu možných řídících členů a tím i hran pro dosud nezapojená slova.

Uveďme příklad:
Vezměme si již výše citovanu větu z PDT

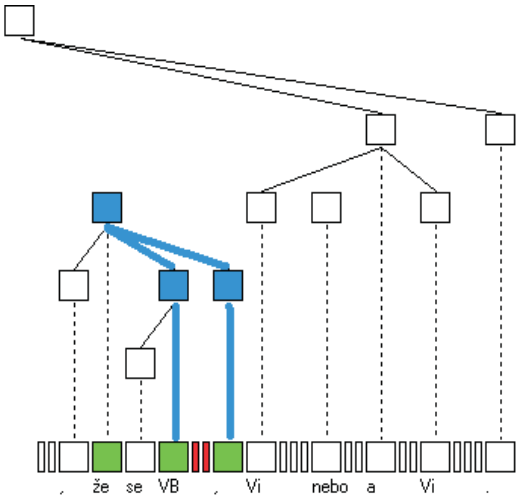
V případě, že se nedovoláte přes den, vytočte číslo ve večerních nebo nočních hodinách a svůj dotaz namluvte na telefonní záznamník.

Šablona této věty je

,–že–se–VB–,–Vi–nebo–a–Vi–.

a hrany její kostřičky

2?4228?080

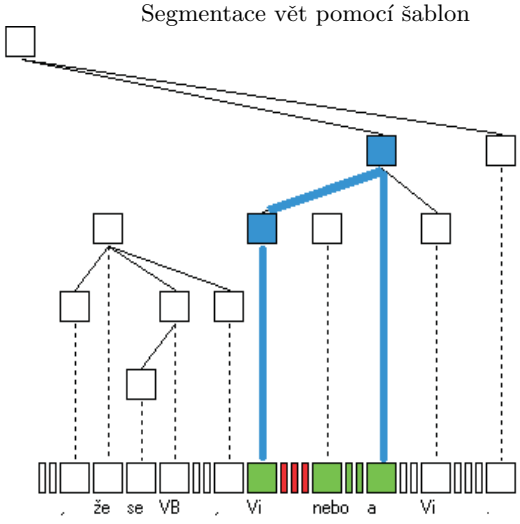


Obrázek 2. Omezení možných hran střechou.

Když bude analyzátor hledat řídící uzly pro dosud nepřipojená slova „přes“ a „den“ (v obrázku jim odpovídají zvýrazněné obdélníčky), potom, za předpokladu, že je kostřička správně a pokud neuvažujeme neprojektivní hrany, připadají krom těchto dvou slov samých v úvahu jedině slova „že“, „nedovoláte“ a čárka.

Hranám takto vymezujícím možné řídící členy pro určitou množinu slov budeme říkat *střecha*.

Máme-li slova, pro něž hledáme řídící člen, obklopena zleva i zprava milníky, je střecha tvořena uzly na cestě od milníků ke kořeni stromu, až po první společný uzel těchto cest. V případě, že obklopující milníky nemají v kostřičce společný nadřízený uzel, zkusíme postupně použít pro určení střechy vzdálenější milníky (viz Obrázek 3).



Obrázek 3. Střecha při přeskočení nezapojeného milníku.

6 Kvantitativní údaje

Na trénovací množině PDT 2.0 bylo nalezeno celkem 456 šablon, které se vyskytly alespoň desetkrát, tyto šablony pokrývají přes 40 procent vět.

Několik příkladů ze seznamu setříděného podle počtu opakování v trénovací množině (pořadí, počet, procentní zastoupení a součet procent):

```
  1 2434 3.83%  3.83%  VB-.-
  2 2384 3.76%  7.59%  Vp-.-
  3 1061 1.67%  9.26%  VBb-.-
  4  835 1.32% 10.58%  VB-
  5  754 1.19% 11.77%  .-
  6  741 1.17% 12.93%  (-)-
  7  546 0.86% 13.79%  se-VB-.-
  8  529 0.83% 14.63%  VB-Vf-.-
  9  477 0.75% 15.38%  se-Vp-.-
       ...
 41  111 0.17% 25.04%  Vp-,-kt-Vp-.-
 42  109 0.17% 25.21%  Vp-,-že-VB-.-
       ...
106   37 0.06% 31.49%  Vp-,-že-VB-Vf-.-
       ...
163   25 0.04% 34.14%  VB-,-kt-VB-Vf-.-
       ...
364   12 0.02% 39.39%  ,-kt-VB-,-se-VB-.-
       ...
455   10 0.02% 40.89%  Vp-,-že-Vpb-Vs-.-
```

Tabulka 2. Šablony nalezené v trénovací množině.

Pokud si zapamatujeme všechny šablony a kostřičky z trénovací množiny treebanku, na testovací množině (zde i dále vždy míněna množina *devtest*) nalezneme šablonu pro 51.96 procent všech vět.

Údaje o použití kostřiček potom záleží na jejich výběru; můžeme se omezit pouze na ty, které svou mírou jednoznačnosti, případně svou četností překračují určitou hodnotu. Je zřejmé, že s vyšším požadavkem na úspěšnost bude klesat pokrytí vět kostřičkami a naopak.

Při požadavku míry jednoznačnosti 0,00 (tj. žádné omezení) dostáváme hodnoty:
- Kostřička věty je k dipozici pro 51.96% všech vět.
- Správnost hran obsažených v kostřičce je 87.95%.
- Omezení závislosti střechou splňuje 99.01% hran.
- Po tomto omezení a po použití kostřiček vybíráme řídící člen v průměru z 85.17% možných slov.

Naopak při požadavku míry jednoznačnosti 0,90 dostáváme hodnoty:
- Kostřička věty je k dipozici pro 23.43% všech vět.
- Správnost hran obsažených v kostřičce je 93.91%.
- Omezení závislosti střechou splňuje 99.87% hran.
- Po tomto omezení a po použití kostřiček vybíráme řídící člen v průměru z 95.32% možných slov.

7 Co dělat s kostřičkami, když je nemáme

Z předchozích údajů jsme viděli, že 68 tisíc vět trénovací množiny nám poskytlo kostřičky pro přibližně polovinu vět testovací množiny.

Pro věty, jejichž šablonu jsme nenašli v trénovací množině, nemáme kostřičku a máme zhruba tyto možnosti, jak k nim přistupovat:
1. Tyto věty analyzovat dosavadními postupy, bez použití kostřiček.
2. Zkusit pro ně najít částečnou kostřičku zkrácením šablony, od začátku, od konce, z obou stran...
3. Zkusit pro ně složit kostřičku z několika překrývajících se šablon a kostřiček.
4. Zkusit vůbec vztah šablon vět a kostřiček popsat gramatikou, která by dovolovala generování kostřiček pro jakoukoliv šablonu.
5. Obecně namísto problému závislostní analýzy původního přirozeného jazyka řešit problém závislostní analýzy jazyka milníků a šablon.

Vyzkoušeli jsme zkracování šablony věty tak dlouho, než se nám podaří najít kostřičku.

Zkusili jsme zkracovat větu od konce a od začátku, zkracování od začátku dávalo o něco lepší výsledky (s požadavkem na míru jednoznačnosti kostřičky 0.00):
- Kostřička věty je k dipozici pro 93.52% všech vět.
- Správnost hran obsažených v kostřičce je 77.71%.
- Omezení závislosti střechou splňuje 96.93% hran.
- Po tomto omezení a po použití kostřiček vybíráme řídící člen v průměru z 53.87% možných slov.

Za povšimnutí stojí omezení možných řídících členů závisostí na 53 procent původního počtu při dodržení toho, že toto omezení je správné pro 96.93% hran.

To, že kostřičku nemáme k dispozici pro 100% vět je zřejmě způsobeno neexistencí některých šablon délky jedna.

8 Závěr

Práce na využití myšlenky šablon vět a kostřiček je v počátcích, přestože už teď jsou některé výsledky zajímavé.

Jako další kroky práce na tomto poli vidíme:
1. Hledat nejlepší množinu milníků
2. Studovat a řešit různost kostřiček pro stejnou šablonu
3. Zkusit překlad šablon do kostřiček popsat gramatikou (již zmíněné island grammars)
4. Zkusit překlad šablon do kostřiček řešit pomocí statistické závislostní analýzy
5. Popisovat jazyk a jeho jevy pomocí šablon a kostřiček.
6. Formulovat a vyzkoušet algoritmy analýzy využívající kostřiček (shora).

Reference

1. Hajič J., Building a Syntactically Annotated Corpus: The Prague Dependency Treebank. Issues of Valency and Meaning, Karolinum, Praha 1998, 106–132
2. Tomáš Holan, Zdenek Žabokrtský, Combining Czech Dependency Parsers. In: Proceedings of TSD'2006, Springer-Verlag, 2006 (to appear)
3. Vladislav Kuboň, Problems of Robust Parsing of Czech. PhD Thesis, Charles University Prague, 2001
4. Vladislav Kuboň, Markéta Lopatková, Patrice Pognan, Segmentation of Complex Sentences. In: Proceedings of TSD'2006, Springer-Verlag, 2006 (to appear)
5. McDonald, R., Pereira, F., Ribarov, K., Hajič, J., Non-Projective Dependency Parsing using Spanning Tree Algorithms. In: Proceedings of HTL/EMNLP'05, Vancouver, BC, Canada, 2005
6. Leon Moonen, Generating Robust Parsers using Island Grammars. Proceedings of the 8th Working Conference on Reverse Engineering, (WCRE 2001). IEEE Computer Society Press, 2001.

Measures for comparing rules extracted from data

Vojtěch Hlaveš[1] and Martin Holeňa[2]

[1] Student of Charles University, Faculty of Mathematics and Physics,
Ke Karlovu 3, Prague, Czech Republic
vhlaves@seznam.cz,
[2] Institute of Computer Science, Academy of Sciences of the Czech Republic,
Pod Vodárenskou věží 2, Prague, Czech Republic
martin@cs.cas.cz

Abstract. *This article deals with different measures for comparing rules extracted from data. At the beginning, the significance of rules as a type of structured knowledge is demonstrated. Then, the necessity of measures comparing rules is explained. Later, the measures are divided into a few groups according to their origin. Several examples are shown with their possible advantages and disadvantages. Finally, an example of evaluating specific measures of rules produced by various methods for extracting structured knowledge from the input data is described.*

1 Introduction

Data mining has been developing since the 90. rapidly. This kind of information technology consists of methods, which extract from input data, which are generally enormous, knowledge in a structured form.

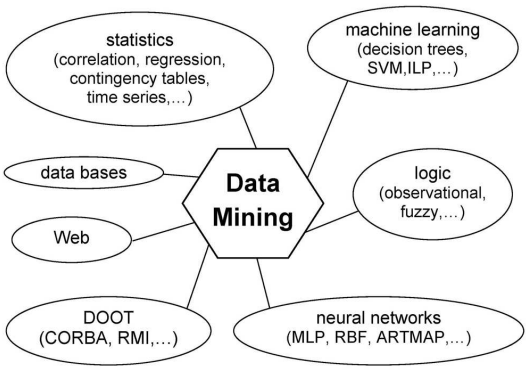


Fig. 1. Methods and technologies underlying data mining.

There is a great variety in the types of structured knowledge [3], e.g. decision and classification rules, hierarchy of classes, clusters, regression functions. The majority of these are directly connected to a single type of methods used for extracting the knowledge, e.g. hierarchy of classes to classification, clusters to cluster analysis, regression functions to linear and nonlinear regression. Thus, methods which use the same structure of the extracted knowledge are usually based on the same paradigms, even though the methods may be of different origins. For example, clusters are encountered in methods based on both statistical analysis and neural networks - however, if each method is examined more closely, very similar paradigms can be found.

Nevertheless, one important exception exists: representation as a sentence of some formal logic. This structure, usually called rules, is used in many methods based on very different paradigms. Tersely, these methods expect set of data and return sets of rules. The rules can be obtained from data by counting frequency of occurrence of individual combinations of values of attributes, or in more sophisticated way such as a result of various statistical methods, e.g. hypotheses testing in a contingency table or estimation of probability distributions, or even non-statistical methods which include artificial neural networks, decision trees or inductive logic programming. These methods are differently computationally demanding and in general different methods produce different sets of rules. Thus, the importance of measures which can compare quality of the extracted rules grows.

2 Different approaches to the measures for comparing rules

Measures for comparing sets of rules can be based on several principles. Below, the principles of the three main kinds of such measures will be briefly sketched.

2.1 Measures derived from the confusion matrix

The measures can arise from confusion matrix (see the table below).

The simplest way to compare two rules is based on comparing their consistency and completeness. They can be expressed as follows

$$Cons = \frac{a}{m}$$

	Consequent C	¬ C	
Antecedent A	a	b	m
Antecedent ¬ A	c	d	n
	k	l	S

Table 1. Confusion matrix.

where a is a number of examples that fulfill a rule that has antecedent A and consequent C, b is a number of examples that fulfill a rule that has antecedent A but the consequent is not C etc.

and

$$Compl = \frac{a}{k}$$

However, there might be arise setbacks if there is no other information added, which would characterize the input data or expected characteristic of the rules. It is impossible to say whether a rule having higher consistency and lower completeness is better than a rule having higher completeness but smaller consistency. Furthermore, in real set of data, there is always some noise. Rules which cover all examples from class C are rather odd or they do not reveal any new information. Thus, more sophisticated measures are needed.

A possible way to improve their qualities is relativize them to some threshold, e.g., to the trivial rule 'all instances belong to this class'. Such measures should give more information about the utility of a rule than absolute measures. For instance, if the completeness of a rule is lower than the $\frac{m}{S}$, then the rule actually performs badly, regardless of its absolute completeness. However, there is a problem with relative completeness that it is easy to obtain high relative completeness with highly specific rules. Therefore, weighted relative measures, such as

$$WRCompl = \frac{k}{S} * \left(\frac{a}{k} - \frac{m}{S}\right)$$

are introduced [5].

Another way to get over problems with completeness and consistency itself is to construct measures which represents a combination of these two criteria [1, 4]. For example, the following measure belongs to this class

$$Q_{Michalski} = w_1 * Cons(R) + w_2 * Compl(R)$$

where $w1, w2 \in (0, 1)$.

Receiver operating characteristic (ROC) curve measures rules obtained not with a sigle method, but rather with a family of methods, differing for example through a tuneable parameter (e.g., threshold in the case of perceptrons, significance in the case of some LISP-Miner quantifiers). It is based on two measures: sensitivity (synonym to completeness described above) and 1-specificity [2, 6].

$$Specificity = \frac{d}{b+d}$$

It connects points in $\Re^2$, the coordinates of which are the pairs (Compl,1-Specificity) corresponding to different values of that parameter.

Furthermore, accuracy is an important measure, which represents the group of measures that can be based on the confusion matrix:

$$Accuracy = \frac{a+d}{S}$$

Accuracy describes how effective the rule is in assigning an object to the correct class.

2.2 Measures based on estimated probability distributions

To the set of measures based on estimated probability distributions belong imprecision and inseparability, and even accuracy can be classed into this group of measures [2]: for example, based on a sum of contributions

$$1 - |\delta(j|x_i) - \hat{f}(j|x_i)|$$

where x_i is a member of a test set, $\hat{f}(j|x_i)$ is the rule's estimated probability that ith object belongs to class j and $\delta(j|x_i)$ is 1 if $c_j = j$ and 0 otherwise.

Precision describes, how close the estimated probabilities $\hat{f}(j|x_i)$ are to the true probabilities $f(j|x_i)$. In principle, such a measure could be based on the differences $1 - |f(j|x_i) - \hat{f}(j|x_i)|$, where $f(j|x)$ represents the conditional probability that an object with measurement vector x belongs to class j.

Finally, separability characterizes similarity of the true probabilities of belonging to each class at measurement vector x, averaged over x. If the probabilities at x are similar, the distribution across classes at x is not dominated by any one class - the most probable class is from the probabilistic point of view not well separated from the remaining classes. Since we are hoping for clear differences between the classes, low separability is to be avoided.

2.3 Complexity of the rules

The third approach is based on complexity of rules. The complexity of rules includes both the number of

rules in each set and number of attributes in the antecedent of every rule. The smaller those numbers, the easier to understand the rule or the set. Generally speaking, having two models with the same error rate on unseen (testing) examples, the simpler one should be preferred because simplicity is desirable in itself (Occam's 'first' razor [7]).

3 Evaluation of measures

The measures were tested on real sets of rules produced by various methods for extracting structured knowledge from data. The used methods (programs) include LISp-Miner 4ft, AQ21, classification trees in Matlab and method based on neural networks.

Three different sets of input data consist of 'BUPA liver disorders', 'Iris Plants Database' and 'Pima Indians Diabetes Database'.

3.1 Cross-validation

It is usually problematic to divide small data sets into design and test set (obviously both even smaller). However, it is essential for proper studying of behaviour of the measures because the rules are always optimised for the design set, which might include some rare or odd cases, and these situations might not be valid for independent data, leading to undesirable overfitting. A compromise between splitting the data into two independent sets and using all data as the design set is the cross-validation [2]. This involves extracting mutually exclusive subsets of the data to test the performance of the method applied to the remaining data. This is then repeated for other subsets and the results are averaged. In our case, the data were divided into 10 subsets.

It is worth mentioning that even rules with low support

$$Support = \frac{a}{S}$$

- therefore the correctly classified cases might not occur in each test set the - are not penalised. In some (very few) cases the rule might is assesed as completely useless, but in average it is assesed correctly.

3.2 Transformation

The first step, when the rules were extracted, was to transform different outputs of each method, such as

$$\begin{aligned} Petal\_length(< 5.100; 6.900 >)\& \\ Petal\_width(< 1.500; 2.500 >)\& \\ epal\_length(< 6.300; 7.900 >)\& \\ Sepal\_width(< 3.100; 4.400 >) \\ \dots \\ Class(Iris\_virginica) \end{aligned}$$

the layout of a rule extracted by LISp-Miner 4ft from Iris Plants Database or

$$\begin{aligned} [Class = Iris\_virginica] \\ \#Rule6 \\ <-- \\ [Sepal\_length = 5.95..6.25 : 12, 20, 37\%, 12, 20, 37\%] \\ [Sepal\_width <= 3.05 : 66, 100, 39\%, 10, 18, 35\%] \\ [Petal\_length = 4.7..5.05 : 16, 10, 61\%, 8, 0, 100\%] \\ [Petal\_width >= 1.45 : 98, 30, 76\%, 8, 0, 100\%] \end{aligned}$$

the layout of a rule extracted by AQ21 from the Iris Plants Database to one common shape. This new form was represented by two two-dimensional arrays, containing lower bounds of each attribute in first dimension and upper bounds in the second dimension. This is possible because each rule produced by these systems is an implication.

For example, a rule originally created by LISp-Miner 4ft classifying Iris Virginica was transformed into a form:

$$\begin{matrix} 5.10 & 1.50 & 6.30 & 3.10 \\ 6.90 & 2.50 & 7.90 & 4.40 \end{matrix}$$

Finally, the measures are being computed as was described above.

Conclusion

This contribution presents a part of the first author's work on his master thesis. This ongoing work will continue also for further rules extraction methods and further data, including real-world data from recent applications. Its objective is not only a comprehensive comparison of important rules extraction methods by means of the measures presented here, but also an investigation, how those methods should be modified, or which further measures could be used, to allow a better comparison of rule sets extracted with important rules extraction methods given data.

Acknowledgement

The research reported in this paper has been supported by the grant No. 201/05/0325 of the Grant

Agency of the Czech Republic and partially supported by the Institutional Research Plan AV0Z10300504 and also partially supported by the Ministry of Education of the Czech Republic (grant MSM0021620838).

References

1. Bruha I., Quality of Decision Rules: Empirical and Statistical Approaches. Informatica, 17, 1993, 233-243
2. Hand D.J., Construction and Assessment of Classification Rules. University of London, UK, 1996
3. Holeňa M., Získávání pravidel z dat. Statistika vol. 83, 2003
4. Kaufman K.A., Michalski R.S., An Adjustable Description Quality Measure for Pattern Discovery Using the AQ Methodology. Journal of Intelligent Information systems, 14, 2000, 199–216
5. Nada Lavrač, Peter Flach, and Blaz Zupan, Rule Evaluation Measures: A Unifying View. ILP-99, LNAI 1634, 1999, 174–185
6. Peter A. Flach, The Many Faces of ROC Analysis in Machine Learning. ICML 2004 Tutorial, `www.cs.bris.ac.uk/~flach/ICML04tutorial/`
7. Pedro Domingos, The Role of Occam's Razor in Knowledge Discovery. Data Mining and Knowledge Discovery 3(4), 1999, 409–425

Robot soccer strategy - 11th Fira Roboworld Cup

Václav Snášel[1], Jan Martinovič[1], and Bohumil Horák[2]

[1] VŠB – Technical University of Ostrava, Faculty of Electrical Engineering and Computer Science, Department of Computer Science, Ostrava, Czech Republic
[2] VŠB – Technical University of Ostrava, Faculty of Electrical Engineering and Computer Science, Department of Measurement and Control, Ostrava, Czech Republic

Abstract. *This paper deals with method of representing robotic soccer game in a simulated and/or real form. This representation is used for controlling robots playing soccer. Our approach to robot soccer is to view it as a local interaction game. We describe our concept of virtual grid and implementation of robot soccer simulator in this paper, as tools for building the strategy and tactical movement database for real game. This strategy we will use as strategy model for 11th FIRA RoboWorld Cup. Strategy learning from game observation is important for discovering strategies of the opponent team and searching of tactical movements groups replaying, simulation and synthesis of anti-strategies.*

1 Introduction

The typical example of distributed control system with embedded subsystems is the task of controlling physical robots playing soccer. The selection of this game as a laboratory task was motivated by the fact that the realization of this complicated multidisciplinary task is very hard. The task can be divided into a number of partial tasks (evaluation of visual information, image processing, hardware and software implementation of distributed control system, wireless data transmission, information processing, strategy planning and controlling of robots). The task is attractive both for students and teachers, and allows direct evaluation and comparison of various approaches. For the improvement of the game strategy, we develop an abstract description of the game and propose how to use this description for e.g. learning of rules. We also take inspiration from the ant-like systems that reduce the need of complexity of individual robots and lead to robust, scalable systems [2, 4, 14, 10]. We build on our previous work – the hardware implementation and basic control of robots – and we would like to achieve higher level control of the game strategy. The rest of the paper is organized as follows: First we briefly describe the base hardware and software implementation. Then we describe the representation of the game field using virtual grids. Then we describe possible game strategies. Using the virtual grids and game strategies, we show how to learn rules that describe particular game strategy. Particular attention is paid to the learning using latent semantic analysis. We conclude with the discussion of the presented approach.

2 Base implementation

The game system can be described as up to twice eleven autonomous mobile robots (home and visiting players), which are situated at the field of the size of 280x220cm. The core of each of our mobile robots is digital signal processor. The higher level of control system is represented by personal computer. The PC receives a view of the playing field from the CCD camera as an input, and gives commands to the mobile robots as an output. The software part is implemented by decision making and executive agents. The agents corresponding to individual robots are controlled by a higher level agent [5, 8, 11, 12]. The task of conversion of the digital image into the object coordinates is solved separately. The coordinates are saved in the scene database [1], which is common for all agents. Both agent teams have a common goal to score the goal and not to get any goal. For a success, it is also important to extract the strategy of the opponent team. The extraction and knowledge of opponent game strategy is an approach that is known to be successful in other situations as well [13].

3 Game representation – virtual grid

The game can be represented as a trajectory in what we call the virtual grid. The virtual grid generally allows us to reduce data volume for easy description of player motion and subsequently for controlling the game or for learning game strategies. The natural coordinate system is provided by accurate optical sensing of the subject position using lens for optical transformations and the CCD camera. This natural coordinate system can be easily mapped to a virtual grid. A sample picture before processing is shown in the figure 1. The data volume of the description using the virtual grid is obviously smaller than the description using natural coordinates. The exact values depend on the frequency of samples and on the maximal velocity of

the mobile robot movement at game field. The dimensions of the primary virtual grid are determined by the possible distance of the robot position in two subsequent frames from the CCD camera. The primary virtual grid can be divided to (2, 4, 8,) parts, which creates secondary virtual grid (in next SVG).

Using the virtual grid, it is possible to describe the position and movement direction of the robot using an alphanumeric description. This description is illustrated in the figure 2 – let us explain the notation on the example description [A2AB13AA14]. The first letter describes the role of the player attacker (A), goalkeeper (G) and defender (D). The second number is an index of the player with the given role in the team (i.e. 1, 2,).

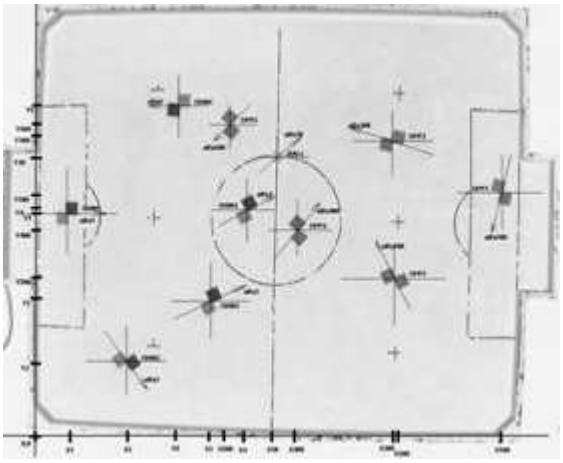


Fig. 1. Sample of the game field with marked positions.

The next two letters and two numbers represent the current position of the player on the field – here, the position is AB12 (see the figure 2). The last two letters and two numbers describe the planned movement, i.e. the planned position in the next moment. The tercial grid strategy grid depends on the partition of the game field (the left-right wing, the central field, and transversely the attack-defence field and the central field). In the discrete frame samples it is possible to study movements and movement strategies of the robots. The A1 tercial grid described right wing in defence field.

4 Game strategy

The game strategy can be dynamically changed based on the game progress (i.e. the history and the current position of the players and the ball [16]). The game progress can be divided in time into the following three ground playing classes (GPC):

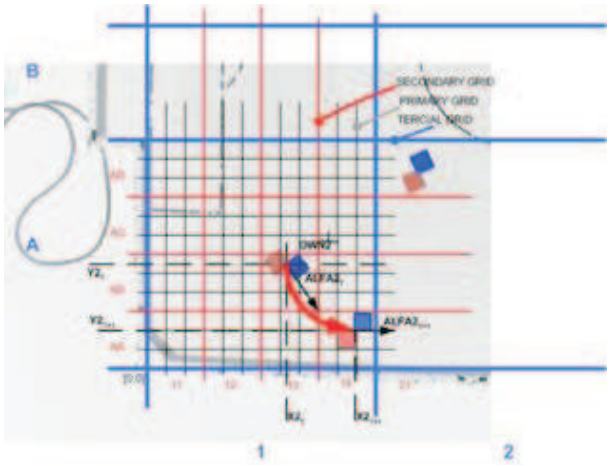


Fig. 2. The alphanumeric representation of robot OWN2 position [X2,Y2] and turn-angle [alfa2] in real coordinates. Description of movement [A2AB13AA14] in secondary grid and description of A1 strategy position in tercial grid.

- GPC of game opening (GPCO)
- GPC of movements in game site (GPCS)
- GPC of game end (GPCE)

The game progress, especially in the GPCS class, can be also divided into the following two game playing situations (GPS):

- GPS of attack (GPSA). The interactions of simple behaviours cause the robots to fall into a V-formation where the ball is in motion roughly towards the opponents goal.
- GPS of defence (GPSD). When the ball is not moving roughly towards the opponents goal, the robots move around it to form an effective barrier and to be in a good position for recovery.

Each GPC has its own movement rules. The classes GPCO and GPCE consist of finite number of possible movements that are determined by initial positions of players and the ball. The class GPCS has virtually unlimited number of possible movements. The movements are determined by the current game situation (GPS) and by the appropriate global game strategy (in next GGS). The movement of the particular robot is determined by the current game class and situation, and also by the robot role. For example, the goalkeepers task is to prevent the opponent to score a goal. His movements are in most cases limited along the goal-mouth near of goal line. The preferred movements are in goal line direction. The preference of these movements comes from the particular GGS, where the goalkeeper prevents to score a goal in the way of moving in the position between the central goal point and the ball (or the expected ball position). The preference

of other movement directions is created using GPSA, where the movements of goalkeeper secure kicking the ball from the defence zone.

5 Learning game strategy from observation

In this section we describe our approach for learning game strategy from observation. Our goal is to learn an abstract strategy. The simplified scheme of process is shown in figure 6. The main steps of the learning process are:

- Transformation of observations into virtual grids.
- Transformation of observations into strategy grids.
- Learning a strategy based on the observed transitions in the strategy grid.

We adopt definition of strategy [7]: Strategy is the direction and scope of an organization over the long-term: which achieves advantage for the organization through its configuration of resources within a challenging environment...

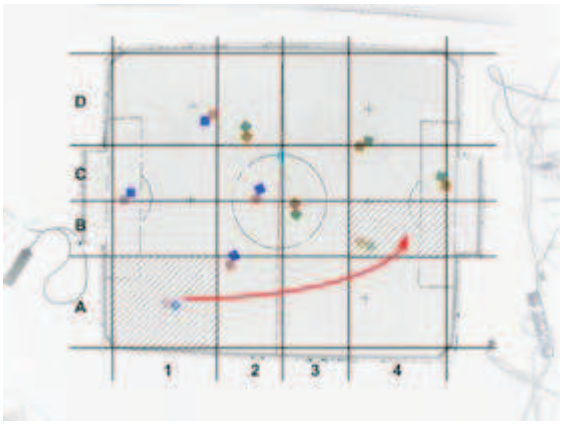


Fig. 3. Example of movement in the strategy grid.

In addition to this definition, we adopt the strategy grid for the description of strategy. The strategy grid has the same dimension as the virtual grid. We define strategy as movements in strategy grid. In this grid, the ground playing situations (GPCO, GPCS, GPCE, GPSA, and GPSD) can be easily observed. For learning a strategy from observation, a game space reduction is needed. Game space reduction is the transformation from virtual grid to virtual strategy grid (VSG).

Observations of events in VSG are timely sampled. In one time sample (TSA) describe situation in VSG one movement sample (MS). The movement samples are formalized and saved in a database of tactical movement groups in a vector form. One movement group takes time from change defence/attack up to next change attack/defence or goal. The time slots (TSL) of single TMG have not the same time length. Structure of records in TMG database is shown figure 4.

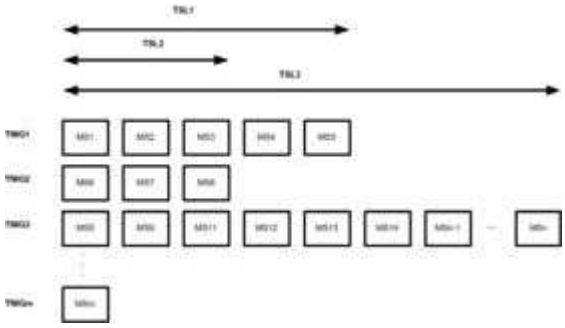


Fig. 4. Records structure in TMG database.

Learning process incorporates searching of TMG recurrences and synthesis of own tactical movements in groups under own anti-strategy. Simulation of robot soccer is used for modeling game situations and environment for learning process. In simulation tools inputs information from TMG database, from other movements and strategy databases of real game system and from operator. Incorporation of simulation subsystem show figure 5.

Fig. 5. Incorporation of simulation subsystem.

Designed incorporation of simulation subsystem allows to process data from real game or from real game picture records.

6 Basic description of strategy selection process

Strategy application for one movement of players is computed in following steps:

- Get coordinates of players and ball from camera
- Convert coordinates of players into strategic grid
- Convert ball and opponents' positions into virtual and strategic grids
- Choose goalkeeper and attacker, exclude them from strategy and calculate their exact positions.
- Detect strategic rule from opponents' and ball positions
- Convert movement from strategic grid to physical coordinates
- Send movement coordinates to robots

Each strategy is stored in one file and currently consists of about 15 basic rules.

```
.Strategy "test"
.Algorithm "Offensive"
.Author "Vaclav Snasel"
.Date "1.5.2004"
.Size 11 9
.PriorityMine      100 100 100 100 100
.PriorityOpponent  50   50  50  50  50
.PriorityBall      50

.Rule  1  "Attack1"
.Mine a6 c7 d6 e3 f9
.Opponent d3 e7 e8 g2 k6
.Ball i6
.Move a6 g7 f5 j3 i8

.Rule  2  "Attack2"
.Mine a6 c7 d6 e3 f9
.Opponent d3 e7 e8 g2 k6
.Ball i5
.Move a6 g7 g5 h3 h8

...
```

Furthermore the file contains following metadata:

- Information about the name of strategy
- The algorithm to strategy choosing
- The author responsible for current strategy
- The date of last modification
- The size of strategic grid
- Strategic rules

Each strategic rule consists of five records:

- The rule ID and description (e.g. Rule 1 "Attack1"),

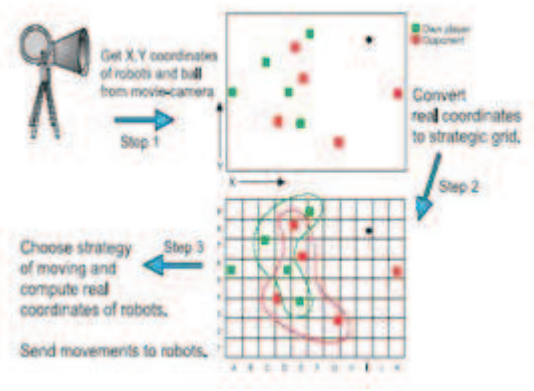


Fig. 6. Base process description.

- the coordinates of our players in strategic grid (e.g. .Mine a6 c7 d6 e3 f9),
- the coordinates of opponent's players in strategic or virtual grid (e.g. .Opponent d3 e7 e8 g2 k6),
- the ball coordinates in virtual or strategic grid (e.g. .Ball i6)
- strategic or virtual grid positions of the move (e.g. .Move a6 g7 f5 j3 i8).

```
// algorithm for rule selection
// Game.Mine      -- actual positions
// Game.Opponent  -- actual positions
// Game.Ball      -- actual position

maxWeight = 0
selectRule = 0

foreach r in Rule
{
  weight = 0
  ruleTmp = r.Mine
  foreach p in Game.Mine
  {
    s = nearest position in ruleTmp to p
    w = 1 / (distance(s, p) + 1)
    w *= Strategy.PriorityMine
    weight += w
    remove s from ruleTmp
  }

  ruleTmp = r.Opponent
  foreach p in Game.Opponent
  {
    s = nearest position in ruleTmp to p
    w = 1 / (distance(s, p) + 1)
    w *= Strategy.PriorityOpponent
    weight += w
    remove s from ruleTmp
  }
```

```
  w = 1 / (distance(Game.Ball, r.Ball) + 1)
  w *= Strategy.PriorityBall
  weight +=  w

  if weight > maxWeight
  {
    maxWeight = weight
    selectRule = r
  }
}
return SelectRule
```

From observation of opponent's strategy a new set of rules can be written, without necessity of program code modification. Furthermore, there is a possibility of automatic strategy (movement) extraction from running game.

There exist two main criteria in the Rule selection process. The selection depends on opponents' coordinates, mines' coordinates and ball position. The strategy file contains rules, describing three possible formations suggesting danger of current game situation. The opponent's team could be in offensive, neutral or defensive formations. Furthermore, we need to weigh up the ball position risk. Generally, opponent is not dangerous if the ball is near his goal. The chosen rule has minimal strategic grid distance from current configuration of players and ball.

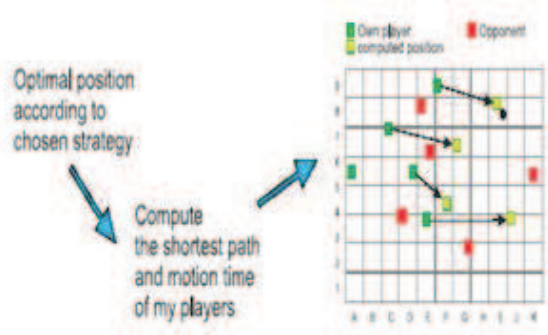


Fig. 7. Final steps of the control process, after selection rule 1 "Attack1" from strategy "test".

Optimal movements of our robots are calculated by applying minimal distance from strategic grid position and rotation penalty see figure 8. The goalkeeper and attacking player, whose distance is closest to the ball are excluded from strategic movement and their new position is calculated in exact coordinates.

To summarize, the strategy management can be described in the following way:

- Based on incoming data from the vision system, calculate virtual and strategy grid Coordinates of the players and the ball.
- The virtual grid is then used to decide which player has under the ball control.
- This player is issued a kick to command that means that it has to try to kick the ball to a given strategy grid coordinates.
- All other players are given (imprecise) go to coordinates. These coordinates are determined by the current game strategy and are determined for each robot individually. The goalkeeper is excluded from this process since its job is specialized, and does not directly depend on the current game strategy.

Fig. 8. Game Simulator.

7 Conclusion

The main goal of the control system is to enable immediate response in the real time. The system response should be shorter than time between two frames from camera. When the time response of the algorithm exceeds this difference the control quality deteriorates. The method we described provides fast control. This is achieved by using rules that are fast to process. We have described a method of game representation and a method of learning game strategies from observed movements of players. The movements can be observed from the opponents behaviour, or e.g. also from the human players behaviour. We believe that the possibility of learning the game strategy that leads to a fast control is critical for success of the robotic soccer players. We implemented a tested this approach in our software see figure 9. Like in chess playing programs, the

database of game strategies along with the indication of their success can be stored in the database and can be used for subsequent matches. In future we want to use the modular Q-learning architecture [9]. This architecture was used to solve the action selection problem which specifically selects the robot that needs the least time to kick the ball and assign this task to it. The concept of the coupled agent was used to resolve a conflict in action selection among robots.

Presented method will be used as main strategy model for Amphora team at 11th FIRA RoboWorld Cup see [3]. We will attend Simurosot Middle League.

Acknowledgements

The Grant Agency of Czech Academy of Science supplied the results of the project No. p. 1ET101940418 with subvention.

References

1. Bernatik R., Horak B., Kovar P., Quick Image Recognize Algorithms. In: Proceeding International Workshop Robot-Multi-Agent-Systems R-MAS 2001. VSB-TU Ostrava 2001, Czech Republic 2001, 53–58
2. Deneubourg J.L., Goss S., Franks N., Sendova-Franks A., Detrain C., Cretien L., The Dynamics of Collective Sorting: Robot-Like Ants and Ant-Like Robots. In Proceedings of the First International Conference on Simulation of Adaptive Behavior: From Animals to Animats, MIT Press, 1991, 356-363
3. FIRA RoboWorld Cup, `www.firaworldcup.de` (May 2006)
4. Holland O., Melhuish C., Stigmergy, Self-Organisation, and Sorting in Collective Robotics. Artiffcial Life, 5(2), 2000, 173-202
5. Horák B., Obitko M., Smid J., Snášel V., Communication in Robotic Soccer Game. Communications in Computing 2004, 295–301
6. Horák B. Obitko M., Smid J., Snášel V., Strategy and Communication in Robotic Soccer Game. EUROCAST 2005, 565–570
7. Johnson G., Scholes K., Exploring Corporate Strategy: Text and Cases. FT Prentice Hall, 2001
8. Obitko M., Snášel V., Ontology Repository in Multi-Agent System. IASTED, International Conference on Artifical Intelligence and Applications (AIA 2004), Innsbruck, Austria, 2004
9. Park K.H., Kim Y.J., Kim J.H., Modular Q-learning Based Multi-Agent Cooperation for Robot Soccer. Robotics and Autonomous Systems, Elsevier, 35, 2001, 109-122
10. Sng H.L., Gupta G.S., Messom C.H., Strategy for Collaboration in Robot Soccer. The First IEEE International Workshop on Electronic Design, Test and Applications (DELTA '02), 2002, 347
11. Smid J., Obitko M., Snášel V., Communicating Agents and Property-Based Types versus Objects. SOFSEM. MatfyzPress 2004
12. Srovnal V., Horák B., Bernatk R., Snášel V., Strategy Extraction for Mobile Embedded Control Systems Apply the Multi-agent Technology. International Conference on Computational Science 2004, 631–637
13. Slywotzky A.J., Morrison D., Moser T., Mundt K., Quella J., Profit Patterns: 30 Ways to Anticipate and Profit from Strategic Forces Reshaping Your Business, 1999
14. Werger B-B., Mataric M.J., From Insect to Internet: Situated Control for Networked Robot Teams. Annals of Mathematics and Artiffcial Intelligence, 2000
15. Kim J., Kim D., Kim Y., Seow K., Soccer Robotics (Springer Tracts in Advanced Robotics), Springer-Verlag, 2004
16. Veloso M. and Stone P., Individual and Collaborative Behaviours in a Team of Homogeneous Robotic Soccer Agents. Proceedings of International Conference on Multi-Agent Systems, 1998, 309–316
17. Nakashima T., Takatani M., Namikawa N., Ishibuchi H. and Nii M., Robust Evaluation of RoboCup Soccer Strategies by Using Match History, 2006 IEEE Congress on Evolutionary Computation, Vancouver, BC, Canada

Automated content-based message annotator – ACoMA*

Martin Šeleng, Michal Laclavík, Zoltán Balogh, and Ladislav Hluchý

Institute of Informatics, Slovak Academy of Science, Dúbravská cesta 9, 845 07 Bratislava, Slovakia
martin.seleng@savba.sk,
WWW home page: http://ikt.ui.sav.sk

Abstract. *V príspevku navrhujeme anotáciu e-mailových správ ako nový spôsob využitia predpripravených znalostí pre organizácie, ktoré využívajú e-mailovú komunikáciu ako súčasť svojich procesov. Ďalej v príspevku opisujeme nástroj, ktorý umožňuje poskytovanie znalostí v organizácií (ACoMA- Automated Content-based Message Annotator) pri riešení pracovných postupov. Ak vieme, že analyzovaný text je prepojený na špecifickú aplikačnú doménu a existuje ontologický model domény môžeme náš nástroj priamo prepojiť na pracovný kontext organizácie. Tento nástroj je používaný a ďalej vyvíjaný pre potreby projektu RAPORT APVT-51-024604 a projektu K-WfGrid EU RTD IST FP6-511385.*

1 Úvod

Na dosiahnutie pracovných cieľov a efektívne riadenie svojho pracovného procesu potrebuje každá spoločnosť komunikáciu, ktorá je dôležitou súčasťou pracovného procesu a spolupráce. Podľa [1] sú kategórie komunikačných cieľov v organizácii nasledovné:

- riadenie konkrétnej úlohy,
- riadenie kolektívnej úlohy,
- poskytovanie a vyhľadávanie informácií pre ďalšie úlohy.

Preto sa e-mailová komunikácia v organizácii týka konkrétnej úlohy, komunikácia je jasná a krátka a po textovom spracovaní a analýze je tak čiastočne zrozumiteľná aj pre počítače.

V znalostne orientovaných systémoch je dôležité mať dynamické nie statické nemeniace sa znalosti a takisto ich správne a rýchlo použiť v situáciách, kde sú potrebné. Toto je možné dosiahnuť použitím elektronickej komunikácie [2] (napr. e-mailu), pretože:

- každá organizácia má alebo bude mať e-mailovú infraštruktúru skôr, než bude mať potrebu vytvárať organizačnú pamäť,
- elektronická komunikácia v moderných organizáciách sa podľa štúdií [3] týka konkrétnej úlohy, pričom takmer každej úlohe v rámci organizácie musí predchádzať komunikácia, ktorá sa väčšinou uskutočňuje prostredníctvom e-mailov [4],
- manažéri rôzneho typu štandardne pracujú s e-mailami, takže použitie e-mailov v rámci komunikácie len minimálne ovplyvní každodenný pracovný proces,
- manažéri sú motivovaní, aby komunikovali zrozumiteľne a krátko, a aby ich odpovede boli zrozumiteľné a jasné.

Pri vytváraní riešenia založeného na e-mailoch organizácia nemusí meniť pracovné návyky, čo je vhodné aj zo sociologického hľadiska. Používatelia môžu komunikovať rovnako ako predtým s tým rozdielom, že e-maily obsahujú priložené informácie a znalosti relevantné pre problém alebo úlohu, ktorej sa e-mail týka.

V niektorých projektoch, ako napríklad kMail [2], ktorý integruje e-mailovú komunikáciu s organizačnými pamäťami, bolo použité prepojenie znalostí s e-mailmi. Nevýhodou tohto projektu je však nutnosť použiť špeciálneho e-mailového klienta. Podobné prepojenie sa použilo napríklad aj v projekte Gmail [5], ktorý zobrazuje reklamné informácie na základe obsahu.

2 Ciele

Na splnenie požiadaviek, ktoré vyplynuli z prvej kapitoly, je potrebné poskytovať:

- spoločnú organizačnú pamäť (Organisation Memory),
- spracovanie e-mailov v danom kontexte pre získavanie znalostí,
- mechanizmus pre aktualizovanie znalostí v organizacnej pamäti.

Jedným z cieľov bolo vyvinúť nástroj na spracovanie e-mailov ACoMA, ktorý je sčasti založený na existujúcom nástroji EMBET [6] (Experience Management based on Text Notes) vyvinutom na našom pracovisku.

3 Prístup a riešenie

E-maily sú silne napojené na prácu v organizácii, avšak ich obsah je väčšinou neštrukturalizovaný.

* Tento nástroj vznikol za podpodry projektov RAPORT APVT-51-024604 a K-WfGrid EU RTD IST FP6-511385

Vyvinutý nástroj je priamo prepojený na pracovný kontext organizácie, takže nie je ťažké analyzovať a pochopiť daný kontext týkajúci sa znalostí v organizačnej pamäti.

Používanie e-mailov umožňuje získať aktívny zdieľaný znalostný kanál, pretože používateľ nemusí používať rozsiahle hľadanie na získanie určitej znalosti. Zdieľané znalosti sú priamo doručené v e-mailovej správe na základe aktuálneho problému alebo aktivity riešenej používateľom. Používateľ dostane e-mail s pripojenými informáciami na konci správy (textové alebo html prílohy resp., text priamo vložený do textu e-mailovej správy). V e-maili sa tiež zobrazia informácie o ďalšom probléme alebo aktivite v danom pracovnom procese. Riešenie pomocou textových alebo html príloh sa javí ako najvhodnejšie, pretože sa nemení text pôvodného e-mailu, len sa dopĺňa o relevantné informácie (text, prepojenia, a pod.). Nástroj ACoMA je vyvíjaný v rámci projektu RAPORT [7] a K-WfGrid [8] (tu je pomenovaný ako WXA - Workflow XML Analyzer) a pracuje v nasledovnom cykle (obr. 1).

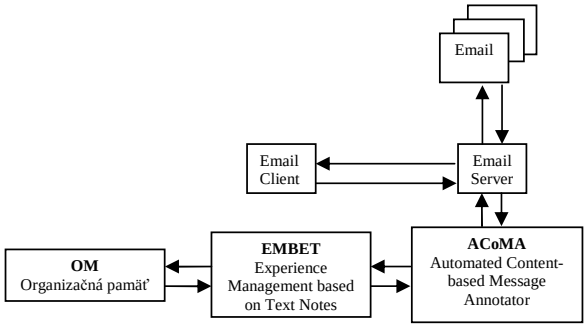


Fig. 1. Cyklus práce nástroja ACoMA.

Nástroj ACoMA je nainštalovaný na poštovom serveri podobne ako antivírové alebo antispamové programy. Po prijatí e-mailu nástroj ACoMA daný e-mail zanalyzuje pomocou sémantickej anotácie [6][9] a získaný kontext vo forme prvkov z ontologického modelu aplikácie pošle nástroju EMBET [6]. Ten na základe získaného kontextu vyberie z organizačnej pamäte všetky relevantné informácie, ktoré následne pošle späť nástroju ACoMA. ACoMA tieto informácie naformátuje a pripojí k prijatému e-mailu a e-mail ponechá na serveri. Používateľ následne pri preberaní pošty dostane už takto upravený e-mail. Pri odosielaní e-mailu je cyklus podobný.

V prípade projektu RAPORT [7] rozlišujeme dva druhy e-mailov: generované portálom (formálne) a vytvárané používateľom (neformálne). Formálne e-maily rozposiela portál buď automaticky (na základe aktivity v danom procese), alebo na základe požiadavky používateľa pri práci s portálom (opäť podľa aktivity, v ktorej sa daný proces nachádza). Neformálne e-maily píšu samotní používatelia zaradení v pracovnom procese (urgencie, potvrdenie dodania dokumentov, a pod.) V projekte RAPORT [7] sa e-mailová komunikácia používa pri vytváraní simulácie bojového cvičenia. Tento proces je jednoznačne definovaný aktivitami:

- príprava simulácie cvičenia,
- špecifikácia cvičenia,
- predpis pre cvičenie,
- bojová dokumentácia,
- plán riadenia simulácie,
- technický riadiaci a komunikačný plán a
- plán podpory cvičenia.

Dané aktivity majú svoj časový harmonogram. Výsledkom každej aktivity je výstupný dokument (dokumenty) založený na vstupných dokumentoch.

Na nasledujúcich obrázkoch je zobrazený spôsob práce nástroja ACoMA tak, ako sa používa v projekte RAPORT pre formálne e-maily (obr. 2 a obr. 3) a neformálne e-maily (obr. 4 a obr. 5). V tejto ukážke budeme simulovať prvú aktivitu pracovného procesu: príprava simulácie cvičenia.

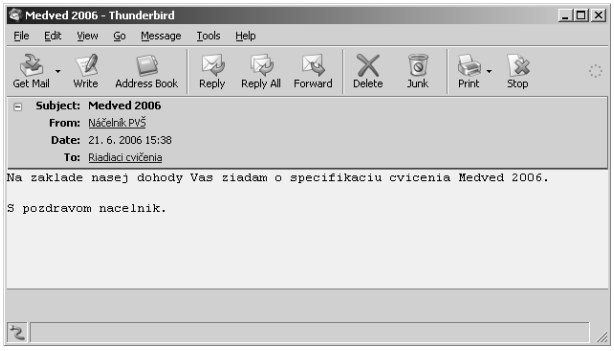


Fig. 2. Príklad e-mailu odoslaného používateľom (požiadavka na zaslanie špecifikácie cvičenia).

Nástroj ACoMA odosielaný e-mail (obr. 2) zanalyzuje, z predmetu správy získa informáciu (v tomto prípade MEDVED2006) a z textu e-mailu získa ďalšiu informáciu (specifikacia). Získané informácie následne odošle nástroju EMBET, ktorý z organizačnej pamäte zistí, že pracovný proces "simulácia cvičenia" sa nachádza v aktivite "'príprava simulácie cvičenia"' (pričom výstupným dokumentom je formulár A), z časového harmonogramu vyčíta, že proces je v stave "'D-75"' (špecifikáciu treba dodať do "'D-70"', co je 5 dní, a preto sa v poznámkach nachádza tento údaj) a podobne z textu "'specifikaciu"' (samotný

regulárny výraz je napísaný iba pre "'specifikaci"') vie odporučiť, kde sa tento formulár nachádza (`http://pellucid.ui.sav.sk/ğiang/raport-xml/`). Tieto informácie zašle naspäť nástroju ACoMA, ktorý daný e-mail upraví do nasledujúceho tvaru (obr. 3) a e-mail odošle poštovému serveru (pridaný text sa v e-maily zobrazuje ako HTML príloha).

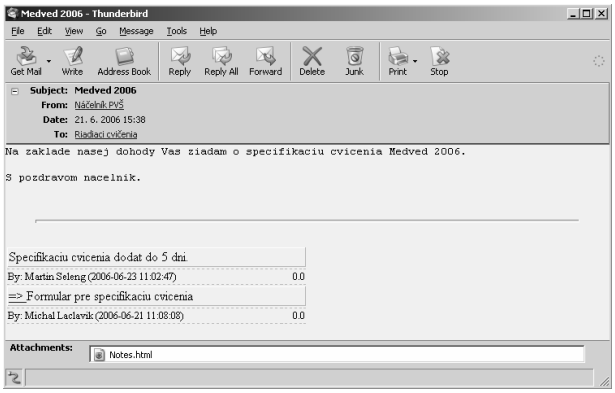


Fig. 3. Príklad e-mailu upraveného nástrojom ACoMA (upravený e-mail na špecifikáciu cvičenia).

Riadiaci cvičenia takto dostane presnejšiu, jasnú a jednoduchú informáciu o ďalšom postupe. V prípade, že Riadiaci cvičenia nedodá v dohodnutom termíne špecifikáciu môže mu Náčelník PVŠ poslať e-mail s nasledovným textom (obr. 4, daný e-mail je už aj o anotovaný).

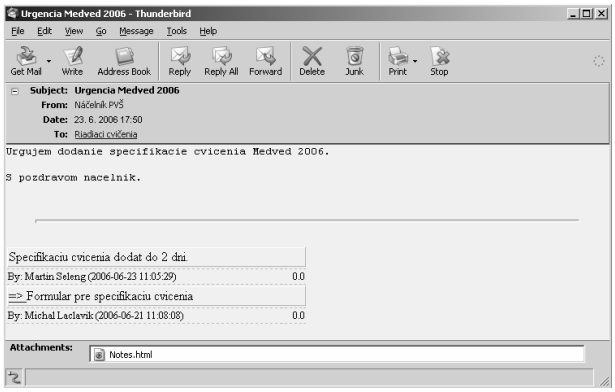


Fig. 4. Príklad neformálneho e-mailu upraveného nástrojom ACoMA (o anotovaná urgencia dodania špecifikácie cvičenia).

Nástroj ACoMA opäť z predmetu správy zistí, že sa jedná o cvičenie Medveď 2006 (používanie od koho komu nie je postačujúce z dôvodu viacnásobného zaradenia jednotlivých účastníkov vo viacerých pracovných procesoch v rôznych funkciách). Podobne ako v predchádzajúcom príklade (obr. 3) zistí, že sa jedná o špecifikáciu cvičenia a zároveň zistí v akom časovom stave sa nachádza pracovný proces.

Záverom prvej aktivity je potvrdzovací neformálny e-mail od Riadiaceho cvičenia zobrazený na nasledujúcom obrázku (obr. 5).

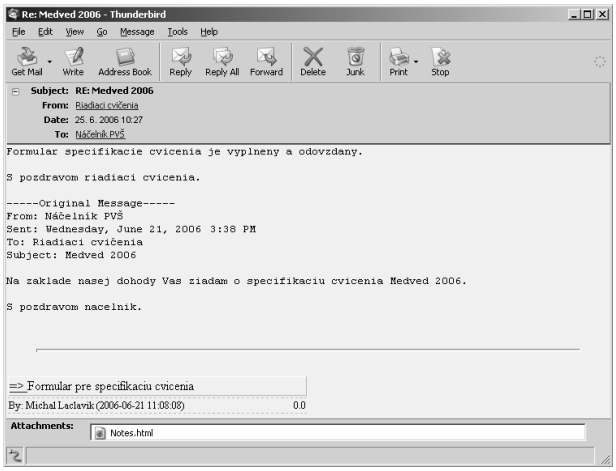


Fig. 5. Potvrdzovací neformálny e-mail od Riadiaceho cvičenia Náčelníkovi PVŠ (o anotovaný o odkaz kde sa nachádza vyplnená špecifikácia cvičenia).

4 Architektúra a technológia

V nasledujúcej časti si rozoberieme architektúru a technológiu nástroja ACoMA a nástroja EMBET.

Nástroj ACOMA sa skladá z 2 hlavných častí:

- ACoMA Core
- ACoMA E-Mail

ACoMA Core je hlavnou súčasťou nástroja ACoMA a zabezpečuje analýzu e-mailu pomocou sémantickej anotácie [6][9] a získaný kontext vo forme prvkov z ontologického modelu aplikácie pošle nástroju EMBET. Tieto znalosti zasiela a prijíma cez XML-RPC [12].

ACoMA E-Mail slúži na prijatie, vytvorenie a odoslanie e-mailu doplneného o relevantné informácie na základe kontextu e-mailu. Nástroj ACoMA používa na prácu s e-mailami JavaMail API [10]. Na vytvorenie HTML prílohy e-mailu sa použije XSLT [11] transformácia textových poznámok získaných z organizačnej pamäte pomocou nástroja EMBET na HTML dokument, ktorý následne pomocou JavaMail API [10] pripojí k už existujúcemu e-mailu.

Nástroj EMBET sa skladá z 3 hlavných častí:

- EMBET Core
- EMBET GUI
- Organizačná pamäť

EMBET Core podobne ako pri nástroji ACoMA predstavuje hlavnú funkcionalitu nástroja EMBET. Na základe kontextu hľadá a vyberá znalosti (vo forme textových poznámok) zo svojej organizačnej pamäte. Vybrané znalosti z organizačnej pamäte sú zasielané cez XML-RPC [12] alebo SOAP [13] nástroju ACoMA (v momentálnej verzii sa používa XML RPC [12])

Rozhranie k **Organizačnej pamäti** je používané pre ukladanie a vyberanie znalostí. Je založené na RDF[14]/OWL[15] práce s dátami a používa Jena API [16].

časti **EMBET GUI** (v projekte RAPORT je EMBET nainštalovaný na portáli, kde sa nachádza aj **EMBET GUI**) sa nebudeme venovať, pretože nástroj ACoMA zabezpečuje zobrazovanie relevantných informácií.

5 Záver

Článok opisuje ako je možné využiť znalosti v organizácii tak aby implementácia ich využitia nezasahovala do zabehnutého pracovného procesu v organizácii. Pri väčšine projektov manažmentu znalostí sa v organizáciách inštalujú nové systémy s ktorými sa užívateľ musí naučiť pracovať. Toto sa javí ako problém aj pri našom systéme EMBET ktorý má vlastné webové rozhranie. V prípade ACoMA užívateľ dostane vhodné informácia a znalostí priamo pri vybavovaní úloh pomocou emailovej komunikácie. Dané informácie môže alebo nemusí využiť pričom ho neobťažujú v zabehnutých pracovných postupoch. Javí sa že je vhodné použiť takýto systém všade tam kde sa elektronická komunikácia používa ako primárny nástroj na manažovanie pracovného procesu.

V ďalšej práci sa budeme snažiť vyhodnotiť používanie takéhoto systému, vylepšenie vizuálnej prezentácia zobrazených znalostí v emailoch ako aj zavedenie mechanizmov na spätnú väzbu od užívateľa na zobrazené informácie a znalosti.

References

1. Habermas J., The Theory of Communicative Action. Beacon, Boston, 1981
2. Schwartz D.G., Te'eni D., Bar-Ilan University, Tying Knowledge to Action with kMail, MAY/JUNE 2000, IEEE Knowledge Management, 33–39
3. Te'eni D., Schwartz D.G., Contextualization in Computer-Mediated Communication. Information Systems-The Next Generation, L. Brooks and C. Kimble (eds), McGraw-Hill, New York, 1999, 327–338
4. O'Reilly C.A., Pondy L.R., Organisational Communication. Organisational Behavior, S. Kerr (ed.), Grid, Columbus, Ohio, 1979, 119–150
5. Google Mail, `http://gmail.com`
6. Laclavík M., Gatial E., Balogh Z., Habala O., Nguyen G., Hluchý L., Experience Management Based on Text Notes (EMBET), Proc. of eChallenges 2005 Conference, 19 - 21 October 2005, Ljubljana, Slovenia, Innovation and the Knowledge Economy, Volume 2, Part 1: Issues, Applications, Case Studies; Edited by Paul Cunnigham and Miriam Cunnigham; IOS Press, ISSN 1574-1230, ISBN 1-58603-563-0, 261–268
7. Raport Project Website, 2006, `http://raport.ui.sav.sk`, RAPORT APVT-51-024604
8. K-Wf Grid Consortium: K -Wf Grid IST Project Website, 2005, `http://www.kwfgrid.net/`, K-Wf Grid EU RTD IST FP6-511385
9. Laclavík M., Šeleng M., Gatial M., Balogh Z., Hluchý L., Ontology Based Text Annotation – OnTeA. In: Proc. of 16-th European-Japanese Conference on Information Modelling and Knowledge Bases, EJC'2006, Y.Kiyoki et.al. (eds), Dept.of Computer Science, VSB - Technical University of Ostrava, ISBN 80-248-1023-9. Trojanovice, Czech Republic, 2006, 280–284
10. Sun Developer Network: JavaMail, 2006, `http://java.sun.com/products/javamail/`
11. XSL Transformations (XSLT), 1999, `http://www.w3.org/TR/xslt/`
12. XMLRPC.com: XML-RPC Home Page, 2005, `http://www.xmlrpc.com/`
13. W3C: Simple Object Access Protocol SOAP, 2003, `http://www.w3.org/TR/soap12`
14. W3C: Resource Description Framework (RDF), 2004, `http://www.w3.org/RDF/`
15. W3C: Web Ontology Language OWL, 2004, `http://www.w3.org/TR/owl-features/`
16. Jena – A Semantic Web Framework for Java, `http://jena.sourceforge.net/`

Budování infrastruktury sémantického webu*

Jakub Yaghob, Filip Zavoral

Katedra softwarového inženýrství, MFF UK Praha
{Jakub.Yaghob,Filip.Zavoral}@mff.cuni.cz

Abstrakt *Idea sémantického webu je široce diskutována mezi odbornou veřejností již mnoho let. Přestože je vyvinuta řada technologií, jazyků, prostředků a dokonce i softwarových nástrojů, málokdo někdy nějaký reálný sémantický web viděl. Za jeden z hlavních důvodů tohoto stavu považujeme neexistenci potřebné infrastruktury pro provoz sémantického webu. V našem článku popisujeme návrh takové infrastruktury, která je založena na využití a rozšíření technologie datového stohu a nástrojích pro něj vyvinutých a jejich kombinaci s webovými vyhledávači a dalšími nástroji a prostředky.*

1 Úvod – současný stav sémantického webu

Kdyby sémantický web byl alespoň z poloviny tak dobrý, jak se snaží proklamovat jeho vizionáři, jistě by se ho chopila komerce a každý by se s ním denně setkával, podobně jako dnes s emailem nebo webovými stránkami. Realita současnosti je však zcela jiná – asi jen málokdo někdy viděl nebo používal něco, co by se dalo nazvat sémantickým webem. Jedním z hlavních důvodů je neexistence nějaké jednotné infrastruktury, na které by bylo možné sémantický web efektivně provozovat.

Tento problém lze nejlépe demonstrovat srovnáním s 'obyčejným' webem. Zde je infrastruktura jasná a dlouhodobě stabilně používaná. Webové servery (nebo farmy serverů) mají na svých discích uložené stránky a zdrojové texty webových aplikací, server data poskytuje typicky protokolem http nebo https klientskému prohlížeči. Data jsou nejčastěji ve formáltu html doplněném případně o další aktivní prvky. Tato data prohlížeč zobrazí nebo interpretuje a umožní uživateli další navigaci.

Sémantický web takovou 'standardní' infrastrukturu nemá. Jsou sice vyvinuty a relativně stabilizovány různé popisné prostředky pro zaznamenávání ontologií (RDF, RDFS, OWL), navrženy a pilotně implementovány specializované dotazovací jazyky (SPARQL [4], RQL [8], SeRQL [9] nebo RDQL [10]), avšak kde a jak jsou data a metadata ukládána (RDF a RDFS jsou sice vhodné prostředky pro uchavávání relativně malých objemů dat, avšak pro velmi velké datové objemy samy o sobě příliš vhodné nejsou), jak se plní daty, jak jsou data vázána na metadata, čím se na ně lze dotazovat, kdo a jak zpracovává odpovědi, jaké protokoly se používají pro vzájemnou komunikaci – to jsou všechno technické detaily, kterým doposud byla věnována pouze marginální pozornost, a to zejména vzájemné komplexní provázanosti jednotlivých otázek. Nedořešenost těchto technických otázek je jednou z příčin reálné neexistence sémantického webu.

Další kapitoly tohoto článku jsou organizovány následujícím způsobem: v kap. 2 je popsána použitelnost stohových systémů pro datové úložiště sémantického webu, kap. 3 popisuje jednotlivé moduly infrastruktury, kap. 4 shrnuje současný stav a nastiňuje další vývoj.

2 Stohové systémy a jejich vztah k sémantickému webu

2.1 Stoh

V rámci vývoje konkrétního informačního systému jsme vyvinuli datovou strukturu pro centrální úložiště dat odpovídající požadavkům na systém kladeným – stoh [1,2].

Základní ideou stohových systémů je vertikalizace dat, tj. nevyužívá se tradiční horizontální pojetí databázové tabulky, kdy jedna řádka představuje nějakou množinu spolu souvisejících atributů nějaké entity. Místo toho je každý atribut 'tradiční' řádky představován jedním řádkem datového stohu a odpovídající si atributy jsou pak spojeny identifikací entity, které tyto atributy náleží. Celé datové schéma všech zúčastněných aplikací je nahrazeno dvěma základními tabulkami – hodnotami atributů a strukturou entit.

Během práce na grantu Sémantického webu jsme ukázali [3], že tato datová struktura je ve skutečnosti velmi dobře použitelná i pro sémantický web.

Původní základní požadavky na použití stohu jakožto centrálního datového úložiště pro integraci byly:

1. Zachování většího množství stávajících provozních aplikací
2. Sjednocení dat z různých pracovišť
3. Aktivní distribuce změn údajů
4. Relativně snadná možnost přidání dalších sbíraných a skladovaných informací
5. Plná informace o změnách dat na časové ose

* Tato práce byla částečně podporována projektem 1ET100300419 Programu Informační společnost Tématického programu II Národního programu výzkumu České republiky.

Tyto požadavky velmi dobře odpovídají i požadavkům na datové úložiště sémantického webu:

1. Zachování zdrojů – zdroje dat jsou rozmístěny po celém webu a nejsou pod správou nikoho konkrétního, tudíž nelze ovlivnit jejich datové schéma.
2. Data uchovávaná v jedné instanci stohu odpovídají jedné ontologii. Ta však nemusí odpovídat ontologiím jiných zdrojů. Při importu dat z jiných zdrojů se tato data převádějí na námi zvolenou a udržovanou ontologii.
3. Stoh je schopen zajistit export dat včetně aktivního exportu (push).
4. Lze poměrně snadno měnit strukturu dat ve stohu, neboť struktura dat není určena přímo databázovým schématem, ale obsahem metatabulek uchovávajících strukturu dat. Změna struktury dat odpovídá změnám v udržované ontologii.
5. Stoh je navíc schopen udržet informace o časovém průběhu dat, takže jsme schopni zjistit informace o stavu světa vzhledem k nějakému časovému bodu.

2.2 Uložení dat ve stohu a vztah k RDF

Data jsou ve stohu aktuálně uložena v jedné tabulce, která obsahuje pro každou hodnotu atributu entity v daném časovém úseku jednu řádku. V každé této řádce jsou uloženy následující položky: číslo entity, typ atributu a hodnota atributu, zdroj dat, validitu a relevanci. První tři položky velmi dobře modelují RDF model dat, kde entita představuje subjekt, typ atributu je predikát a hodnota atributu je objekt. Zbývající atributy jsou vhodné pro implementaci reifikací.

Tím, že datová tabulka stohu odpovídá RDF, jsme získali v rámci výše uvedeného projektu velkou databázi (řádově desítky miliónů záznamů) reálných RDF dat. Drobnou nevýhodou je fakt, že některá data jsou privátní a nemohou být zveřejněna, což se dá napravit vhodným anonymizováním těchto dat.

2.3 Reifikace

Atribut zdroj dat zmiňovaný v předešlé podkapitole určuje odkud data pocházejí, relevance reprezentuje dohad důvěryhodnosti zdroje dat a dat samotných a validita určuje časový rozsah platnosti dat. V kontextu sémantického webu lze tyto údaje o každé datové položce považovat za reifikace př9slušné RDF trojice. Explicitním modelováním těchto vztahů pomocí čistého RDF bychom dostali několikanásobně větší data a dotazování nad takovými daty by bylo znatelně pomalejší. Jednoduchým využitím základních vlastností stohu můžeme tyto vztahy velmi jednoduše a přitom efektivně použít.

2.4 Kontextová ontologie a mapování ontologií

Ontologie uchovávaná v metatabulkách stohu je ontologií kontextovou, tj. popisuje pouze data uložená ve stohu. Data časem přibývají a mění se i ontologie typicky zvětšováním (přidáváním dalších oblastí), někdy i změnou. Metatabulky stohu pak musejí zvládnout tyto změny ontologií beze změny obsahu datové části stohu.

Velkým problémem, který brání celosvětovému rozšíření sémantického webu, je mapování různých ontologií na sebe. Různí autoři se snaží tento problém různými prostředky řešit, bohužel v současné době problém není uspokojivě vyřešen.

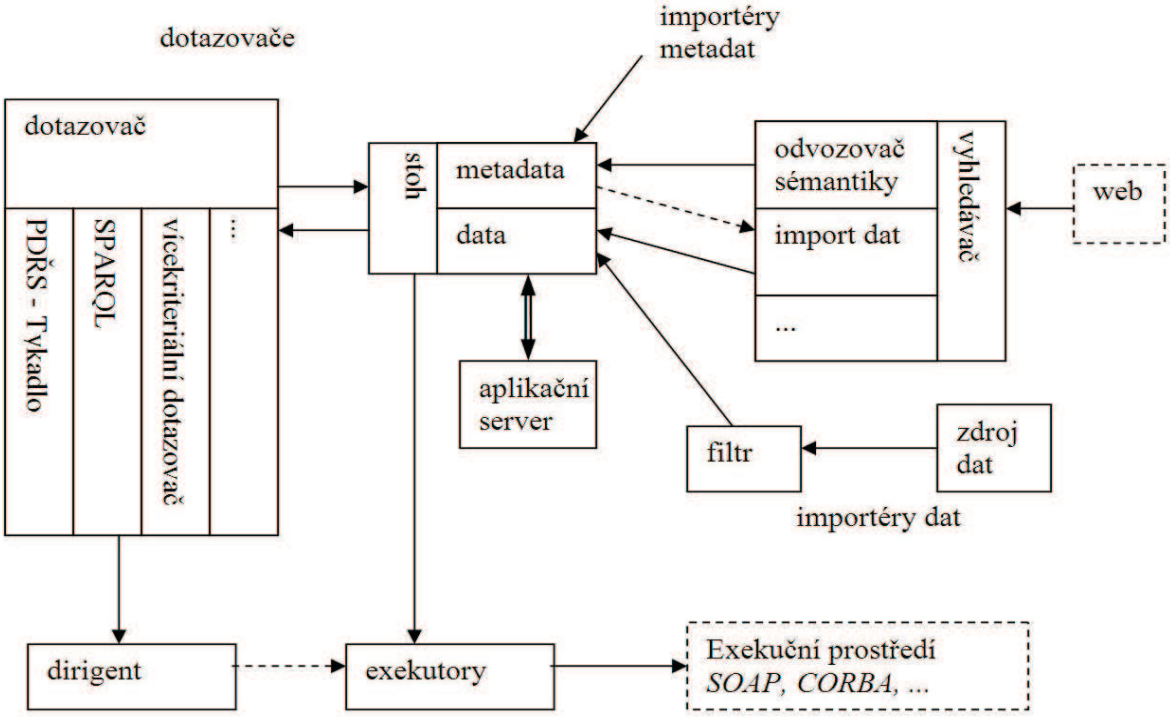
Pokud použijeme kontextovou ontologii, pak musíme i zajistit mapování s jinými ontologiemi při přijetí nových dat a metadat. Zde se nabízejí tři různé metody:

- Jednou z možných metod je metoda Rosetské desky, tj. existuje nějaký spolehlivý, dobře známý zdroj, který zajišťuje alespoň částečný překlad mezi různými ontologiemi.
- Druhá metoda je mapování map na sebe. Mějme mapy nějakého území z různých časových období, takže zobrazují trochu jinou situaci. Pokud se nám podaří na mapách najít několik málo styčných bodů (např. význačná města), pak už jsme schopni zbytek map na sebe také namapovat.
- Poslední metodou je domluva dvou lidí, kteří mluví jiným jazykem. S využitím neverbální komunikace si vytvoří základní slovník, pomocí kterého pak vytváří další bohatší slovník, čímž mapují postupně jazyky na sebe.

První dvě metody vyžadují nějaký lidský zásah – nalezení nebo vybudování Rosetské desky, v druhém případě typicky lidská obsluha najde styčné body na mapě. Třetí metodu lze nejspíše využít pro čistě strojové mapování. Bude nutné navrhnout nějaký protokol, který nahradí neverbální prostředky lidské komunikace nějakými jinými prostředky dostupnými ve světě počítačů.

3 Infrastruktura pro provoz sémantického webu

Základem navrhované infrastruktury pro sémantický web je stoh, kde jsou uložena všechna metadata a data na ně vázaná. Stoh poskytuje čtyři základní druhy rozhraní - pro import dat, import a aktualizaci metadat, dotazování a exekutory.



Obrázek 1. Infrastruktura pro provoz sémantického webu.

3.1 Importéry

Rozhraní pro import dat umožňuje libovolnému modulu doplňovat do stohu data. Typickým představitelem importérů jsou filtry, které data z libovolného zdroje (databáze, XML, web, ...) konvertují do fyzické podoby zpracovatelné stohem a do logického tvaru odpovídajícímu metadatům, na která jsou tato data navázána.

Důležitou součástí importérů je schopnost detekovat již existující data a tato aktualizovat. K tomu slouží rozhraní pro unifikce, kde na základě určujících a relevantních atributů lze pomocí unifikačních algoritmů na sebe vázat existující a nově importovaná data.

Zvláštní význam mezi importéry mají vyhledávače – ty spojují sémantický web s webem. Pro naše účely jsme použili systém Egothor [6], který svojí modulární koncepcí umožňuje komfortně doplnit příslušné moduly pro spolupráci se stohem. V původní podobě Egothor na základě stažených dat vytváří záznam webu v inverzní vektorové podobě. Doplněním extrakčních modulů umožní vybraná data ukládat do stohu. Vzhledem k obrovskému množství těchto dat je konverze zprostředkovávána pomocí specializovaného kompresního modulu [12].

Pouhý přísun samotných dat by v dlouhodobějším provozu sémantického webu nedostačoval – svět sémantického webu je velmi dynamický a jakákoliv struktura dat již v okamžiku jejího zaznamenání může být zastaralá. Proto důležitou úlohu mají importéry metadat. Jejich cílem je aktualizovat metadata tak, aby co nejvěrněji odpovídala aktuálnímu stavu. Podobně jako importéry dat mohou být importéry metadat libovolné komponenty s libovolnou logikou, jedinou podmínkou je implementace vyhovující definovanému rozhraní.

Importéry metadat lze rozdělit na manuální a automatické. Typickým představitelem manuálních importérů metadat jsou filtry exportů různých datových modelů, XML Schemat apod., které umožní přímý import z takto popsaných zdrojů dat. Vystavět sémantický web pouze nad těmito relativně pevně strukturovanými daty by však bylo příliš omezující, současný web obsahuje o mnoho řádů více dat nestrukturovaných.

Připravený framework pro automatické importéry metadat umožňuje vytvářet samostatné moduly, které na základě různých algoritmů založených heuristických, statistických a pravděpodobnostních metodách a v neposlední řadě také na umělé inteligenci mohou automaticky generovat metadata ze zpracovávaných dat. Jedním z příkladů automatických importérů je vyhledávač Egothor doplněný o modul pro automatické odvozování sémantiky na základě stažených dat.

3.2 Dotazovače

Zřejmě nejdůležitějším účelem uložených dat a metadat sémantického webu je možnost dotazování. Na rozdíl od tradičních relačních databází s pevným datovým schématem a standardizovanými dotazovacími nástroji je dotazování nad daty sémantického webu zatím ve stádiu návrhů a pilotních implementací. Tomu odpovídá i navrhovaná infrastruktura. Základem je opět definované rozhraní, jehož prostřednictvím lze na stoh klást dotazy a získávat odpovědi. Tento popis je záměrně velmi široký, neboť obě jeho části (kladení dotazů a získávání odpovědí) mohou nabývat mnoha podob.

Vlastní dotazování je komplikováno tím, že uživatel typicky nezná strukturu dat, která je navíc rozsáhlá a v čase dynamická. Proto jedním z modulů dotazovače je prohlížeč dat řízený sémantikou (PDŘS) pracovně nazvaný Tykadlo, který umožňuje vyhledávat a prohlížet metadata, filtrovat data vztažená k těmto metadatům, a přes datové vazby prohlížet další data na tato vázaná.

Modul SPARQL [3,4] přeloží dotaz zapsaný v jazyce SPARQL do SQL a nechá ho vyhodnotit databázový stroj. Tento způsob dotazování je vhodný pro jednodušší dotazy, složitější dotazy s velkým počtem spojení jsou zatím výkonnostně nedostačující.

Dalším modulem je vícekriteriální dotazovač [5], který umožňuje specifikovat několik vyhledávacích kritérií, přičemž výsledek je nějakou obecnou (monotónní) funkcí jednotlivých kritérií [7]. Uživatel má vlastní preference pro jednotlivé atributy i pro jejich celkovou integraci. Úlohou modulu je najít nejlepší odpověď, případně k-nejlepších odpovědí. V dynamické verzi může být modul rozšířen o použití výsledků předešlých dotazů, a to jak vlastních tak i jiných uživatelů.

Ve fázi vizí je doplnění dotazovače o moduly umožňující formulaci dotazů v přirozeném jazyce a zapojení lingvistických metod, případně použití pokročilých metod umělé inteligence a dolování dat ve stylu 'ukažte, data, co je na vás zajímavého'. Přestože přiblížení těchto vizí je hudbou vzdálenější budoucnosti, infrastruktura je na tyto moduly připravena.

3.3 Exekutory

Doposud jen volně zmiňovaný 'výsledek dotazu' lze interpretovat mnoha různými způsoby. Od relačně orientovaného data-setu přes seznam odkazů známý z webových vyhledávačů nebo seznam entit doplněný jejich vazbami až po aplikační funkčnost typu zavolání vhodné služby SOA.

Tradiční způsob reprezentace výsledků dotazování je pevně vázaný na použitý dotazovač. Tykadlo zobrazuje vzájemně propojené html stránky se zobrazenými vyhledanými daty a jejich vazbami, vyhledávač zobrazuje webové odkazy s případným podrobnějším popisem, SPARQL vrací řádky n-tic vyhledaných atributů.

Technika exekutorů zavádí do infrastruktury procesní modely. Úkolem exekutoru je provést sémantickou akci, tj. interakci dat získaných dotazovačem s ostatním světem, a to nejen světem sémantického webu. Tyto atomické exekutory lze složit a vytvořit exekutory složené. Orchestraci, tj vzájemné propojení exekutorů za účelem dosažení požadované komplexnější funkčnosti, provádí modul dirigent.

Ideu exekutorů lze ilustrovat následujícím příkladem. Onemocní-li někdo, potřebuje lék. Dotazovač nalezne nejbližší lékarnu nabízející vhodný lék. Jeden exekutor je zodpovědný za nákup léku zatímco druhý zařídí jeho dodávku až domů. Modul dirigent orchestruje tyto exekutory tak, aby byly vzájemně synchronizované, aby vzájemně spolupracovaly a předávaly si relevantní data.

4 Závěr

Předkládané řešení infrastruktury pro sémantický web je svým zaměřením otevřený framework, nikoliv jedno uzavřené řešení. To je podle nás zcela nezbytné vzhledem k rozmanitosti sémantického webu, jeho současné nevyzrálosti a potřebě velmi flexibilní budoucí rozšiřitelnosti.

Jednotlivé části infrastruktury jsou v různých stádiích dokončenosti. Centrální databáze založená na technologii stohu je hotová a funkční, v reálném projektu byla pilotně otestována na řádově desítkách miliónů záznamů. Stejně tak základní nástroje spolupracující se stohem, zejména Tykadlo a unifikační algoritmy. Pro plnohodnotné použití stohu pro infrastrukturu sémantického webu je však vhodné rozšířit strukturu metadat tak, aby umožňovala pojmout komplexnější ontologie.

Technika filtrů jakožto základních prostředků pro importéry dat i metadat je také převzata z pilotního nasazení [11], konkrétní importéry jsou však ve fázi implementace. Vyhladávač Egothor je plně funkční, avšak nezaintegrován do infrastruktury. Navrhované moduly importu dat a odvozovače sémantiky jsou nyní ve fázi specifikace.

Z dotazovačů je Tykadlo implementované a funkční, SPARQL a vícekriteriální dotazovače jsou pilotně implementovány, avšak doposud nezaintegrovány do zbytku prostředí. Formáty exekutorů a přesná funkčnost dirigentů jsou ve fázi specifikace.

Budoucí práce bude spočívat především v implementaci zbývajících komponent, jejich integraci a následném experimentálním zkoumání vlastností jak celkové architektury tak i jednotlivých modulů. Předpokládáme, že na základě získaných výsledků a zkuše-

ností bude v tomto článku popisovaná otevřená infrastruktura doplněná o další moduly a datové toky.

Reference

[1] Bednárek D., Obdržálek D., Yaghob J., Zavoral F., Data Integration Using DataPile Structure. ADBIS 2005, Proceedings of the 9th East-European Conference on Advances in Databases and Information Systems, Tallinn, 2005

[2] Bednárek D., Obdržálek D., Yaghob J., Zavoral F., Synchronisation of Large Heterogenous Data Using DataPile Structure. ITAT 2003, Information Technologies – Applications and Theory, University of P.J.Safarik Kosice, 2003

[3] Dokulil J., Transforming Data from DataPile Structure into RDF. In Proceedings of the Dateso 2006 Workshop, Desna, Czech Republic, Vol-176 2006, 54–62 `http://sunsite.informatik.rwth-aachen.de/Publications/CEURWS`,

[4] Dokulil J., Použití relačních databází pro vyhodnocení SPARQL dotazů. ITAT 2006, Information Technologies – Applications and Theory, University of P.J.Safarik Kosice, 2006

[5] Eckhardt A.: Metody pro nalezení nejlepší odpovědi s různými uživatelskými preferencemi. Diplomová práce MFF UK Praha, 2006

[6] Galambos L., Sherlock W., Getting Started with ::egothor, Practical Usage and Theory Behind the Java Search Engine, `http://www.egothor.org/`, 2004

[7] Pokorný J., Vojtáš P., A Data Model for Flexible Querying. In Proc. ADBIS'01, A. Caplinskas and J. Eder (eds), Lecture Notes in Computer Science 2151, Springer Verlag, Berlin 2001, 280–293

[8] Alexaki S., Christophides V., Karvounarakis G., Plexousakis D., Schol D.: RQL: A Declarative Query Language for RDF, in Proceedings of the Eleventh International World Wide Web Conference, USA, 2002

[9] Broekstra J., Kampman A., SeRQL: A Second Generation RDF Query Language, SWAD-Europe, Workshop on Semantic Web Storage and Retrieval, Netherlands, 2003

[10] Seaborne A., RDQL - A Query Language for RDF, W3C Member Submission, 2004 `http://www.w3.org/Submission/2004/SUBM-RDQL-20040109`

[11] Kulhánek J., Obdržálek D., Generating and Handling of Differential Data in DataPile–oriented systems. IASTED 2006, Database and Applications DBA 2006, 503–045

[12] Chernik K., Lánský J., Galamboš L., Syllable-Based Compression for XML Documents. In: Snášel V., Richta K., and Pokorný J., Proceedings of the Dateso 2006 Annual International Workshop on Databases, Texts, Specifications and Objects. CEUR-WS, Vol. 176, 21–31