Dept. of Computer Science & Engineering, Czech Technical University, Prague
Department of Computer Science, VŠB – Technical University of Ostrava
Department of Software Engineering, Charles University, Prague
Czech Society for Cybernetics and Informatics, Workgroup on Computer Science & Informatics, Prague

Proceedings of the Dateso 2005 Workshop

DATESO 2005

Databases, Texts, Specifications, and Objects

http://www.cs.vsb.cz/dateso/2005/

April 13 – 15, 2005
Desná – Černá Říčka

DATESO 2005

Technical editor:
Pavel Moravec, pavel.moravec@vsb.cz

Faculty of Electrical Engineering and Computer Science,
VŠB – Technical University of Ostrava

Page count:	150
Impression:	150
Edition:	1^{st}
First published:	2005

This proceedings was typeset by PDFLaTeX.
Cover design by Tomáš Skopal (tomas@skopal.net) and Pavel Moravec (pavel.moravec@vsb.cz).
Printed and bound in Ostrava, Czech Republic by TiskServis Jiří Pustina.

Published by Faculty of Electrical Engineering,
Czech Technical University in Prague

Preface

DATESO 2005, the international workshop on current trends on Databases, Information Retrieval, Algebraic Specification and Object Oriented Programming, was held on April 13 – 15, 2005 in Desná – Černá Říčka. This was the 5^{th} annual workshop organized by FEL ČVUT Praha, Department of Computer Science and Engineering, MFF UK Praha, Department of Software Engineering, and VŠB-Technical University Ostrava, Department of Computer Science. The DATESO aims for strengthening the connection between this various areas of informatics. The proceedings of DATESO 2005 are also available at DATESO Web site `http://www.cs.vsb.cz/dateso/2005/`.

The Program Committee selected 12 papers from 15 submissions, based on two independent reviews.

We wish to express our sincere thanks to all the authors who submitted papers, the members of the Program Committee, who reviewed them on the basis of originality, technical quality, and presentation. We are also thankful to the Organizing Committee and Amphora Research Group (ARG, `http://www.cs.vsb.cz/arg/`) for preparation of workshop and its proceedings.

March, 2005 K. Richta, V. Snášel, J. Pokorný (Eds.)

Program Committee

Karel Richta (chair)	Czech Technical University, Prague
Václav Snášel	VŠB-Technical University of Ostrava, Ostrava
Jaroslav Pokorný	Charles University, Prague
Vojtěch Svátek	University of Economics, Prague
Peter Vojtáš	University of P. J. Šafárik, Košice

Organizing Committee

Pavel Moravec	VŠB-Technical University of Ostrava
Michal Valenta	Czech Technical University, Prague
Yveta Geletičová	VŠB-Technical University of Ostrava

Table of Contents

Comparison of parallel and random approach to a candidate list in the multifeature querying**

Peter Gurský

Institute of Computer Science, Faculty of Science
P.J.Šafárik University in Košice, Jesenná 9, 040 01, Košice, Slovak Republic
gursky@upjs.sk

Abstract. In the field of the multifeature querying it is possible to use many heuristics to retrieve top k objects to became low number of accesses to the sources. When the sources have many equal values, it is often hard to choose which source should be accessed next. In this paper we compare previous random approach with the parallel approach to the set of actual candidate sources.

Key words: multifeature querying, top-k objects, aggregation

1 Introduction

Many times we want to find the best object or top k objects in the possible huge set of objects. The reason of which object is better than the other, is based on the properties of the objects. Such properties are typically fuzzy. For example, when we want to find top k hotels, we can look at a distance from the beach, price per night, number of stars, travel expenses, etc. We need is to specify, how to compute the overall score of each object to became the order of the objects. Moreover all these particular data can be accessible by different sources (web services).

There are several algorithms in this area, solving this problem. Ronald Fagin introduced "Fagin's algorithm", which solves this problem first time [6]. Fagin et al. [2] presented "threshold" algorithm that made the search much faster. Güntzer et al. [1] defined "quick-combine" algorithm using first heuristic. Other heuristics was presented by P. Gurský and R. Lencses [4].

The "quick-combine" algorithm was originally developed over multimedial data. Such a data are typically continuous i.e. it is very unusual to have two objects with the same value of a property. The experimental comparison of the heuristics [4] showed, that the heuristics used in [1] is quite ineffective, when

** This work was partially supported by the grant VEGA 1/0385/03 and 'Štátna úloha výskumu a vývoja "Nástroje pre získavanie, organizovanie a udržovanie znalostí v prostredí heterogénnych informačných zdrojov" prierezového štátneho programu "Budovanie informačnej spoločnosti".'

the sources have few discretized values, e.g. number of stars of hotels. In this paper, we show a possible improvement of the performance of this heuristic over discretized data. We also try to use the same approach to other relevant heuristics presented in [4].

In chapter 2 we describe a formal model of data. In chapter 3 we present a generalized version of all mentioned algorithms and compare three different heuristics. The experimental comparison of the heuristics is showed in chapter 4. Chapter 5 concludes our paper.

2 Model

Assume we have a finite set of objects. Cardinality of this set is N. Every object x has m attributes $x_1, \ldots, x_m$. All objects (or identificators of objects) are in lists $L_1, \ldots, L_m$, each of length N. Objects in list L_i are ordered descending by values of attribute x_i. We can define two functions to access objects in lists. Let x be an object. Then $s_i(x)$ is a grade (or score, rank) of object x in list the L_i and $r_i(j)$ is an object in list the L_i in the j-th position. Using the function $s_i(x)$ we can realize the *random access*[1] to the lists. The second type of access we will use, is the *sorted access*[2]. Using this type of access the grades of objects are obtained by proceeding through the list sequentially from the top.

We have also monotone aggregation function F, which combine grades of object x from lists $L_1, \ldots, L_m$. The overall value of an object x we denote as $S(x)$ and it is computed as $F(s_1(x), \ldots, s_m(x))$.

Our task is to find top k objects with highest overall grades. We also want to minimize time and space. That means we want to use as low sorted and random accesses as possible.

3 Generalized Threshold algorithm and heuristics

For each list L_i, let $u_i = s_i(r_i(z_i))$ be the value of the attribute of the last object seen under sorted access, where z_i is the number of that possition. Define the threshold value τ to be $F(u_1, \ldots, u_m)$. Because we assume that we have a monotone aggregation function F and the lists are sorted descend by their values, the threshold τ is the value, which none of still unseen objects can reach [2]. Hence when all objects in the top k list have their values greater or equal to the threshold, then this top k list is the final and there is none unseen object with greater value. This property is very important to have the algorithm correct.

Let $z = (z_1, \ldots, z_m)$ be a vector, which assigns for each $i = 1, \ldots,$ m the position in list L_i last seen under sorted access. Let H be a heuristic that decides which list (or lists) should be accessed next under sorted access (notice that

[1] *Random access* - direct access via an indexing mechanism. Please do not confuse this term with the term "random approach". The random approach to a set means, that we choose one element of this set randomly.

[2] *Sorted access* - sequential access to a sorted list.

heuristics can change during the computation). Moreover, assume that H is such, that for all $j \leq m$ we have $H(z)_j = z_j$ or $H(z)_j = z_j+1$ and there is at least one $i \leq m$ such that $H(z)_i = z_i+1$. The set $\{i \leq m : H(z)_i = z_i + 1\}$ we call the set of candidate lists (or simply candidates) for the next sorted access.

In this paper we use three types of heuristics.

First heuristic (denote $H1$) does the sorted access in all m lists parallelly. It means, that for each $i \leq m$ holds $H(z)_i = z_i+1$. This heuristic was firstly presented by Fagin et al. [2] in the "Threshold algorithm". This kind of heuristic we use only (if ever) in the first phase of computation to retrieve the beginnings of the lists. Next two heuristics are used in the rest of computation.

The use of the $((\delta F/\delta x)*\Delta x)$ heuristic ($H2$) was firstly presented by Güntzer et al. [1] as a part of "Quick-combine algorithm". Let us look at the next non-equality. To keep algorithm correct, for each object x in the final top k list must hold:

$$S(x) = F(s_1(x), \ldots, s_m(x)) \geq \tau \tag{1}$$

Hence, when we can say, that this non-equality holds, we have the final top k list. Obviously, there are two ways to make (1) hold: to increase the left side or to decrease the right side. Heuristic $H2$ tries to decrease τ as fast as possible. As a criterion for a list L_i, $i \leq m$ to be given to the set of candidates for the next sorted access, is to have Δ_i maximal. Δ_i is defined as:

$$\Delta_i = \left(\frac{\delta F}{\delta x_i}(s_1(r_i(z_i)), \ldots, s_m(r_i(z_i)))\right)^{-} * (s_i(r_i(z_i - p)) - s_i(r_i(z_i))) \tag{2}$$

The constant p is some suitable (small) natural number. Hence Δ_i is the multiplication of the partial derivative of aggregation function F from the left in the point $(s_1(r_1(z_1)), ..., s_m(r_m(z_m)))$ and the expected change of values in p steps (Δx factor) of the i-th list. When we have Δ_i for each $i \leq m$, we can set the value of $H(z)_i$. Heuristic $H2$ sets $H(z)_i$=z_i+1 if Δ_i =max$\{\Delta_j;\ j \leq m\}$. Otherwise $H(z)_i$=z_i. The only necessary condition we required from F is the continuity from the left.

The $(\delta F/\delta x) * x)$ heuristic ($H3$) is a variation of the last one. This heuristic was presented by Gurský et al.[4] at the first time. Instead of the Δx factor, $H3$ chooses an x-factor, thus the last seen value in the i-th list. The criterion for this heuristic is:

$$\chi_i = \left(\frac{\delta F}{\delta x_i}(s_1(r_i(z_i)), \ldots, s_m(r_i(z_i)))\right)^{-} * s_i(r_i(z_i)) \tag{3}$$

The criterion (3) computes the partial derivation of F from the left in the point $(s_1(r_1(z_1)), ..., s_m(r_m(z_m)))$ and multiply it with value in the point $s_i(r_i(z_i))$ in L_i. This heuristic sets $H(z)_i$=z_i+1 if $\chi_i = \max\{\chi_j;\ j \leq m\}$. Otherwise $H(z)_i$=z_i. We need the continuity from the left for the function F again.

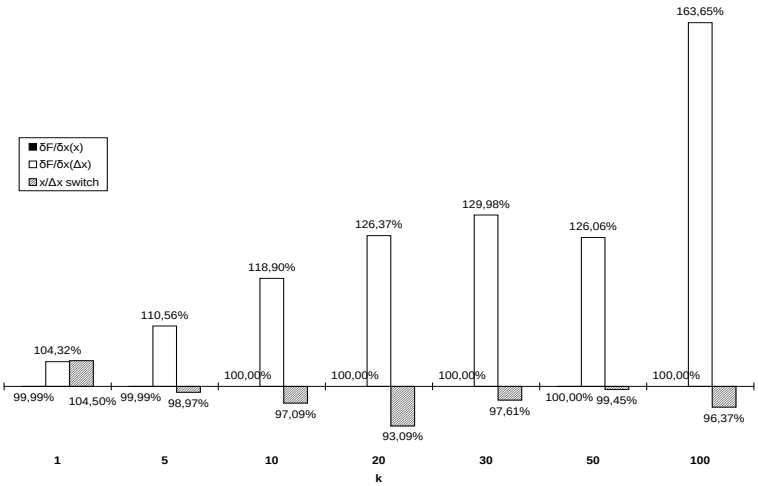


Fig. 1. Change of performance using parallel approach (benchmark data)

When the aggregation function is the simple weighted mean, the derivation used by these heuristics is a constant, more precise it is the weight of the attribute.

Now we can describe the generalized Threshold algorithm:

0. z:=(0,...,0), in case of algorithms $((\delta F/\delta x) * \Delta x)$ or $\Delta x/x$, set the suitable small natural p.
1. Set the heuristic H:
   - $((\delta F/\delta x) * x)$: H:=$H3$
   - $((\delta F/\delta x) * \Delta x)$: if any $z_i < p$ then H:=$H1$; otherwise H:=$H2$
   - $\Delta x/x$: if any $z_i < p$ then H:=$H1$; otherwise if H=$H1$ then H:=$H2$; if H=$H2$ then H:=$H3$; and if H=$H3$ then H:=$H2$.
2. - *parallel approach*: Do the sorted access in parallel to each of the sorted lists to all positions where $H(z)_i = z_i+1$. Put $z_i = H(z)_i$.
   - *random approach*: Do the sorted access to randomly chosen sorted list L_i where $H(z)_i = z_i+1$. Put $z_i = z_i + 1$ and for each $j \leq m, j \neq i$ do nothing.
3. First control: Compute the threshold value τ. As soon as at least k objects have been seen whose grade is at least equal to τ , then go to step 6.
4. For every object x that was seen under sorted access in the step 2, do the random access to the other lists to find the grade $s_i(x)$ of object in every list. Then compute the grade $S(x) = F(s_1(x), \ldots, s_m(x))$ of object x. If this grade is one of the k highest ones we have seen, then remember object x and its grade $S(x)$.

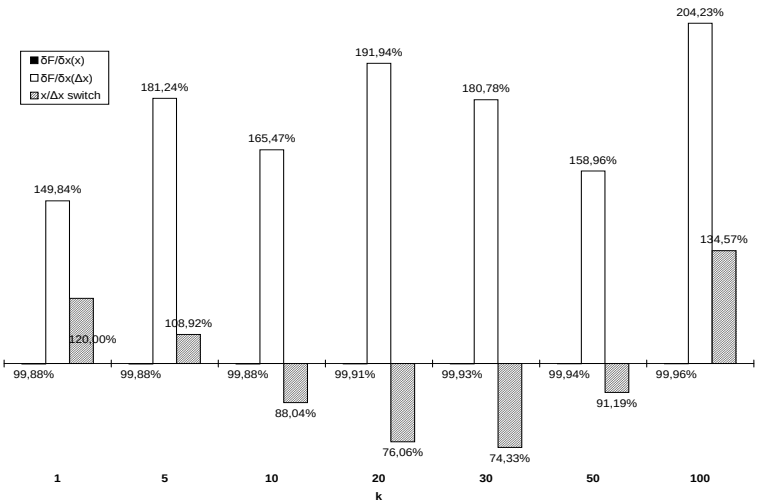


Fig. 2. Change of performance using parallel approach (artificial data)

5. Second control: As soon as at least k objects have been seen whose grade is at least equal to τ , then go to step 6, otherwise go to step 1.
6. Let Y be a set containing the k objects that have been seen with the highest grades. The output is then the graded set $\{(x, S(x)) : x \in Y\}$.

Individual algorithms differ in the steps 2 and 3. Shortly the $(\delta F/\delta x)*x)$ algorithm use heuristic $H3$ directly on beginning. The Quick-combine (or $((\delta F/\delta x) * \Delta x)$) algorithm use for the first p steps in each list the heuristic $H1$ and than the heuristic $H2$. The $\Delta x/x$ algorithm use for first p steps in each list the heuristic $H1$, too. After that it switches between the heuristics $H2$ and $H3$.

Original algorithms choose the random approach in the step 3. All mentioned algorithms can have their "parallel variant" too. In our experiments we compare all 6 possible variants.

4 Experiments

4.1 Testing data

Benchmark data The first sort of data are real data that come from randomly generated queries in information retrieval system with support of relational database. We used different combinations of local and global weights as different ways for weighting of occurrences of terms in documents to generate 6 sets of benchmark data. Each set include over 25 000 objects (documents) with 50 different attributes (terms). To measure all particular experimental results we

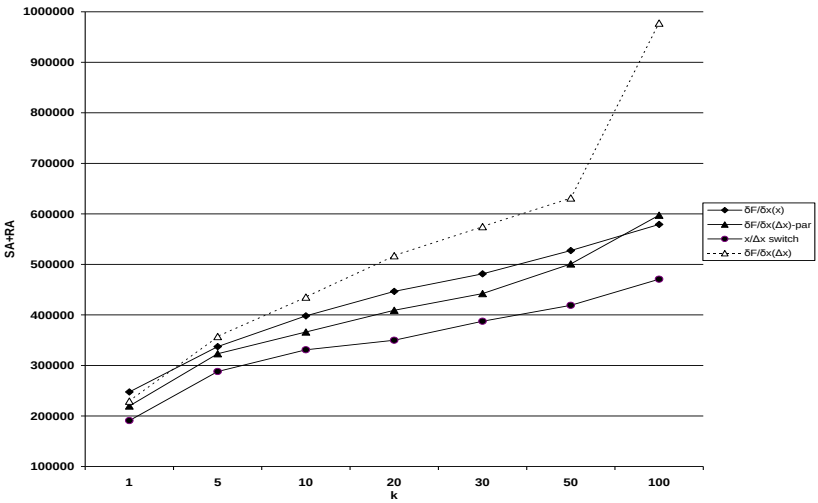


Fig. 3. Number of all accesses (benchmark data)

compare the average values from 6 sets of these benchmark data with the use of aggregation functions with randomly chosen weights for each list. Histograms of data are exponential - there is very low number of objects with high values and a lot of objects with low values. Such distribution is typical in the information retrieval area but in many other areas too.

Artificial data The second sort of data were generated with various distribution of values. We used 2 exponential and 2 logarithmic distributions with 10000 objects and 6 types of aggregation functions. The values of the attributes was rounded to 10 discrete values. Such a discretisation is quite common in real data e.g. number of stars of hotels, rating classes of companies, and other human produced ratings. Finest discretisation can be e.g. prices of some product or a guess for the length of a trip. Continuous data are typical for some physical experiments, precise measurements or multimedial data. In this paper we focus on the discrete data. Using different combination of the source data and aggregation functions we became 16 different inputs for algorithms. In the final results we use the averages of the particular results.

4.2 Results

In our experiments we wanted to compare the random and parallel approach to the set of candidates. On the figure 1 and 2 we take as the base the random approach (=100%). We can see the performance of the parallel approach compared

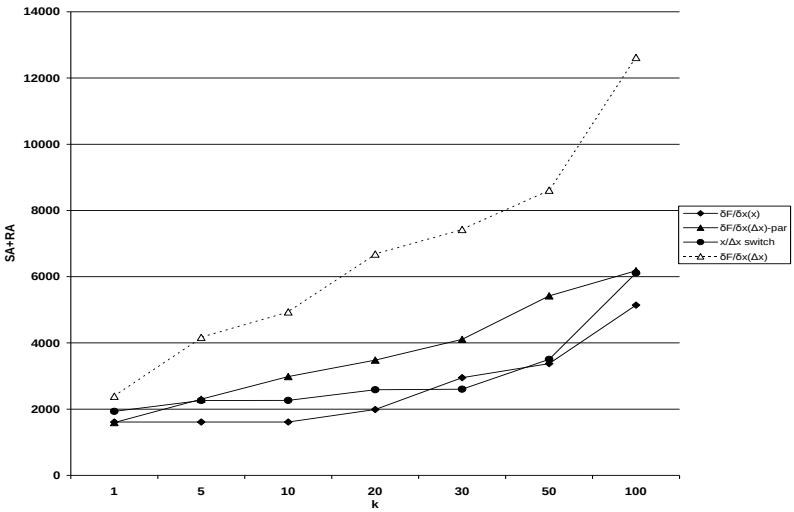


Fig. 4. Number of all accesses (artificial data)

with the random approach for each heuristic. Both artificial and benchmark data shows, that different approach to the candidates does not change anything for the $(\delta F/\delta x) * x$ heuristic. For the $x/\Delta x$ heuristic the random approach seems to be better in most cases.

In our results, the new parallel approach positively improved the performance of the $(\delta F/\delta x) * \Delta x$ heuristic over discretised values. This heuristics was originally developed for multimedial data. Using our improvement this heuristic can be quite efficient over discretized data too, but as we can see in figures 3 and 4 when we work over discretized values the new approach to the $(\delta F/\delta x) * \Delta x$ heuristic is still not better than quite stable $x/\Delta x$ heuristics.

Why we have such a results? The parallel version of the $(\delta F/\delta x) * x$ algorithm has almost same results as its random variant maybe because there was almost none situation in which we had more than one candidate in the candidate set. On the other side the $(\delta F/\delta x) * \Delta x$ algorithm had many such a situations. The reason of this situation is the fact, that the expression $(s_i(r_i(z_i - p)) - s_i(r_i(z_i)))$ almost always equals to zero for small p and discrete values in the lists. As experiments show, for this heuristic when we don't know to prefer one list, it is better to access all lists. For the $x/\Delta x$ heuristic seem the random approach to be better when k is greater or equals to 5.

5 Conclusions

We proposed the new parallel approach to the candidate set and compare it with the previous random approach. We experimentally showed that this type of approach improved the $(\delta F/\delta x) * \Delta x$ heuristic over discrete data. On the other hand the $x/\Delta x$ heuristic keeps its first place in lower number of accesses as was shown in [4].

References

1. U.Güntzer, W.Balke, W.Kiessling *Optimizing Multi-Feature Queries for Image Databases*, proceedings of the 26th VLDB Conference, Cairo, Egypt, 2000
2. R.Fagin *Combining fuzzy information from multiple systems*, J. Comput. System Sci., 58:83-99, 1999
3. R.Fagin, A.Lotem, M.Naor *Optimal Aggregation Algorithms for Middleware*, proc. 20th ACM Symposium on Principles of Database Systems, pages 102-113, 2001
4. P.Gurský, R.Lencses *Aspects of integration of ranked distributed data*, proc. Datakon , ISBN 80-210-3516-1, pages 221-230, 2004
5. R.Fagin *Combining Fuzzy Information: an Overview*, ACM SIGMOID Record 31, Database principles column, pages 109-118, 2002
6. R.Fagin *Combining fuzzy information from multiple systems*, 15th ACM Symposium on Principles of Databases Systems, pages 216-226, 1996
7. P.Gurský, R.Lencses, P.Vojtáš *Algorithms for user dependent integration of ranked distributed information*, technical report, 2004

Finite State Automata as a Data Storage

Marian Mindek and Martin Hynar

Department of Computer Science, VŠB – Technical University of Ostrava
17. listopadu 15, 708 33 Ostrava–Poruba, Czech Republic
{marian.mindek, martin.hynar}@vsb.cz

Abstract. In this paper, we summarize ideas to use finite automata as a tool for specification and compression of data aggregates (e.g. images, electrical signals, waves, large (sparse) matrixes, etc.). We describe different ways of data access. Then we describe an approach how make a resultant automata with included interesting information, how to focus on interesting information in our data, and how to link together resultant automata.

Keywords: finite automata, compression, large sparse matrix, searching, pattern

1 Introduction

Finite automata is an useful tool for matrix representation of commonly used information resources (e.g. images, texts, sound waves, electrical signals etc.), for their compression and for obtaining interesting information about given data [1,2,4,5,7,8,9].

In our opinion, such technique could be used also to represent large matrixes, which are usually hard to manipulate. A traditional approach (compression using common algorithms) solves only part of the problem. It consumes less space but on the other hand, there is no way to make changes to original matrix. Moreover, there is no way to use another additional information and if it is required it has to be computed using other means (e.g. nearest neighbors of some 1-position, interest points, carrier, base, etc.). With our approach, we can focus at this issue and improve predication capabilities about data. The resultant automaton (or automata) contains this interesting information. We can use it for comparing per pattern or search similar information (e.g. part of faces, medical pictures, buildings tracing, part of large sparse matrixes, similar noise, similar trends, etc.).

If we want to have certain benefit from such advantages and if we want to have some mean to store matrixes in database with included interesting information, we can use the approach of storing resultant automata in some well known structure such as table, matrix or XML.

In the following examples we describe for simplicity our approach on the images, if will not remark alternatively.

2 Data specification

2.1 Finite State Automata (FSA)

Background about automata theory in this chapter is the most necessary. We describe only simple procedure for storing matrixes to automata too. For more about automata theory please read [6] and for more about automata as a tool for specifying image, please read [3,4,5,7].

In order to facilitate the application of FSA to matrix description we will assign each pixel at 2^n x 2^n resolution (2^n for vector) a word of length n over the alphabet $\Sigma=\{0,1,2,3\}$ for basic approach and $\Sigma=\{0,1\}$ for offset (read vector) approach, as its address. Offset (vector) approach is useful for matrix approach too.

Example. The large sparse matrix can by represented as set of coordinates [x, y] where x is a row and y is a column. If we separate x part and y part we obtain two vectors. These vectors can have value as set of positions in matrix, or difference between previous and followed position.

A part at 2^n x 2^n (2^n) resolution corresponds to a sub-part of size 2^{-n} of the unit part. We choose ε as the address of the whole unit part. Single digits as shown in figure 1a; on the left address its quadrants. The four sub-squares of the square with address w are addressed *w0, w1, w2* and *w3,* recursively. Addresses of all the sub-squares (pixels) of resolution 4 x 4 are shown in figure 1, middle. The sub-square (pixel) with address *3203* is shown on the right of figure 1. Clearly for offset (vector) approach is sub-part with address w denoted only as *w1* and *w2*, recursively. For comparison see figure 1b, black part of vector has address 1101.

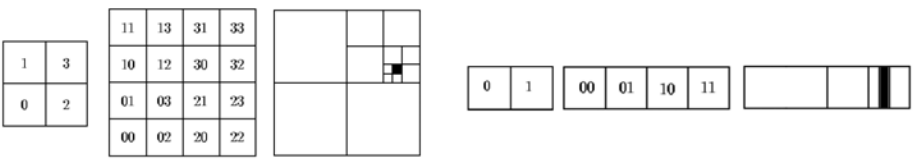


Fig. 1. The addresses of the quadrants, of the sub-square of resolution 4 x 4, and the sub-square specified by the string 3203.

For simplicity, in the following we describe theory only for matrixes. (Offset) vectors approach is very similar.

In order to specify a binary matrix of resolution 2^m x 2^m, we need to specify a language $L \subseteq \Sigma^m$. Frequently, it is useful to consider multi-resolution images, sounds or el. signals simultaneously specified for all possible resolutions (discriminability), usually in some compatible way (We denote Σ^m the set of all words over Σ of the length m, by Σ^* the set of all words over Σ).

In our notation a binary matrix is specified by a language $L \subseteq \Sigma^*$, $\Sigma=\{0,1,2,3\}$, i.e. the set of addresses of all the evaluated squares.

A word in the input alphabet is accepted by the automaton if there exists labeled path from the initial state to the final state. The set (language accepted by automaton *A*) is denoted *L(A)*.

Example. The 2 x 2 chessboards in figure 2 (a) look identically for all resolutions. The multi-resolution specification is the regular set $\{1,2\}\Sigma^*$. The 8 x 8 chessboard in figure 2 (b) is described by the regular set $\Sigma^2\{1,2\}\Sigma^*$ or by FSA *A* figure 2(c).

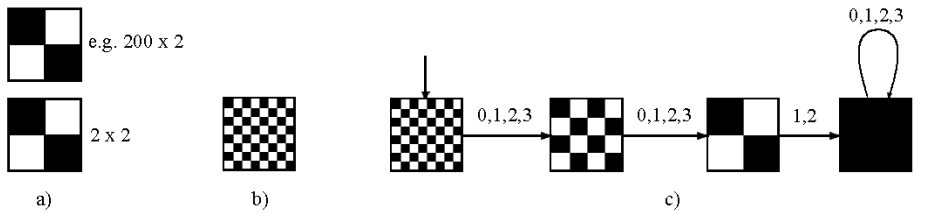


Fig. 2. 2 x 2, 8 x 8 chessboards and corresponding automaton.

Note that here we used the fact that the regular expression $\Sigma^2\{1,2\}\Sigma^*$ is the concatenation of two regular expression Σ^2 and $\{1,2\}\Sigma^*$.

Example. By placing the triangle $L = L_1L_2$ where $L_1 = \{1,2\}^*0$ and $L_2=\Sigma^*$ into all squares with addresses $L_3=\{1,2,3\}^*0$ we get the image $L_3L=\{1,2,3\}^*0\{1,2\}^*0\Sigma^*$ shown at the left of figure 5.

Zooming [3] is easily implemented for matrixes represented by regular sets (automaton) and is very important for loss compression.

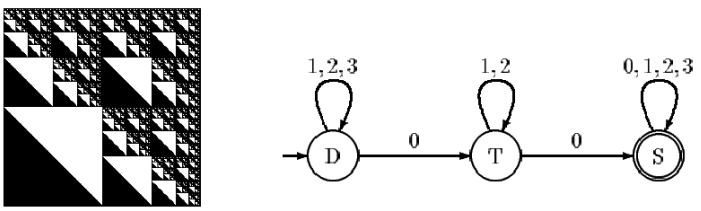


Fig. 3. The diminishing triangles defined by $\{1,2,3\}^*0\{1,2\}^*0\Sigma^*$, and the corresponding automaton.

We have just shown that a necessary condition for binary matrixes to be represented by a regular set (FSA) is that it must have only a finite number of different sub-matrixes in all the sub-squares with addresses from Σ^*. We will show that this condition is also sufficient. Therefore, matrixes that can be perfectly (i.e. with infinite precision for loss compression) described by regular expressions are matrixes of regular or fractal character. Self-similarity is a typical property of fractals. Any matrix can by approximated by a regular expression however; an approximation with a smaller error might require a larger automaton. Multi-resolution (fractal) principle is at most useful for images, sounds, and descriptions of function or electrical signals.

2.2 Basic procedure

Our algorithm for matrix compression (both approaches) is based on basic procedure for black-and-white images proposed in [4], but it will use evaluated finite automata (like WFA) introduced in [3] and only replacing black and white color to real values, without possibility to create loops and adding some option for setup compression and facilitation storage for likely representation.

Example. For the image *diminishing triangles* in figure 3, the procedure constructs the automaton shown at the right-hand side of figure 3. First, the initial state *D* is created and processed. For sub-square with address 0 a new state *T* is created, for addresses 1,2 and 3 create the new states with deep *n* (where deep is length of route from the root and *n* is length of part of word *w* with the same symbol; loop of edge from previous algorithm). Then state *T** is processed for sub-square with address 0 and new state *S* is created, for 1 and 2 a connection to a last of new states. There is no edge labeled 3 coming out of *T* since the quadrant 3 for *T* (triangle) is empty (in binary matrix there is 0 everywhere). Finally, the state *S* (square) is processed by creating edge back to *S* for all four inputs. In this way it represents end of automaton or loop to state itself for multi-resolution approach.

Now we demonstrate in brief a generalized method for matrix compression applicable on construction of resultant matrix storage, or matrix database with included information presented furthermore. There lead four edges from each node at most (for offset approach lead two edges at most) and these are labeled with numbers representing matrix / vector part. Every state can store information of average value of sub-part represented thereby state.

The procedure *Construct Automaton for compression* terminates if exists an automaton that perfectly (or with small-defined error) specifies the given matrix and produces a deterministic automaton with the minimal (interpret as optimal for our problem solution) number of states. The count of states can be reduced a bit or extended by changing error or do tolerance for average values of matrix part. This principle is naturally useful only for matrixes, where we can obtain matrix reconstructed with small error (only if we make tolerance, it is loss-compression.)

Changing the part (or only one matrix element) in source matrix can change the count of states in resultant automata. We can use certain principle to optimize this algorithm for non-recompress all matrix. Details are described in the furthermore in the text.

Procedure *Construct Automaton for Compression*
For given matrix M (in arbitrary representation e.g. full matrix, difference vector, [x, y] representation, etc.), we denote M_w the zoomed part of M in the part addressed w, where $w \in \{0,1,2,3...X\}$. For simplicity we use $w \in \{0,1\}$, see figure 1. The matrix represented by state numbered x is denoted by u_x.

Procedure *Construct Automaton for Compression*
$i = j = 0$
create state *0* and assign $u_0 = M$ (matrix represented by empty word and define average value of M)
assume $u_i = M_w$

```
    loop
        for k ∈ {0,1} do
                if M_wk = u_q (or with small error, only for loss compression)
                or if the matrix M_wk can be expressed as a part or expanded part of
                        the matrix u_q for some state q
                then    create an edge labelled k from state i to state q
                else    j = j + 1
                        u_j = M_wk
                        create an edge labelled k from state i to the new state j
                end if
        end for
        if i = = j than
                Stop (all states have been processed)
        else    i = i + 1
        end if
    end loop
end procedure
```

It is clear, that procedure for vector (offset) approach is very similar. We do not describe it, but we give some confrontation later.

Procedure for reconstruction matrixes from automaton is very simple, for more information see [9].

2.3 Tests

Every test was carried out on standard PC with Intel Celeron 1,3GHz and 384MB RAM. We used two different algorithms for computing resultant automata without loss compression, but results are very similar, such that we describe only one of the results. Tested data was generated randomly. Some of the test matrixes correspond with the worst test data (for our procedure) for comparison.

In table 1 there are depicted matrixes and corresponding counts of evaluated elements. The last column contains counts of similar parts of matrixes. Sizes of these parts are between 1/16 - 1/128, for larger matrixes. Maximum range of similar parts is 6 for matrix with 32000x32000 elements. This setup is only for testing, real data are generally more similar but we show tests for worse cases of data. In next comparison, we test matrixes without similar parts. The resultant automata were perfectly (without loss) representing the source matrix.

In table 2 there are depicted results of our tests. In first column there are source matrixes, in second resulting time for procedure with using vector approach and then follow two columns with counts of state of corresponding resultant automata. In last two columns there are times for procedure using full matrix (*Procedure Construct Automaton for Compression* described before) and counts of states of resultant automata.

Table 1. The tested matrixes.

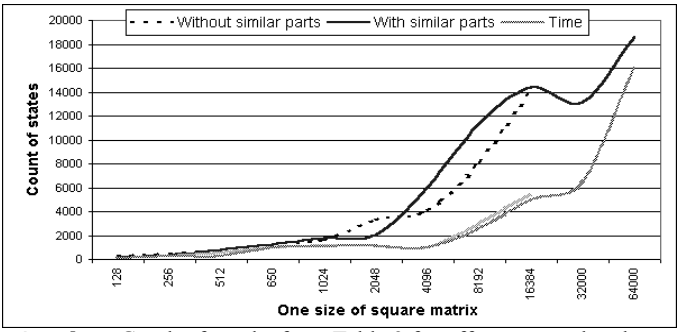
Matrix	Count of elements	Similar parts
128 x 128	67	2
	57	0
256 x 256	128	2
	124	0
512 x 512	266	2
	244	0
650 x 650	515	2
	526	0
1024 x 1024	533	3
	500	0
2048 x 2048	1035	2
	1011	0
4096 x 4096	2058	2
	2022	0
8192 x 8192	4000	3
	4059	0
16000 x 16000	*1*	*0*
16384 x 16384	8193	4
	8000	0
32000 x 32000	9000	6
64000 x 64000	18124	0

Table 2. The results of tests. From the left: source matrixes, time for offset approach in seconds, count of states of resultant automata for X and Y parts, time for classical procedure and counts of states of resultant automata.

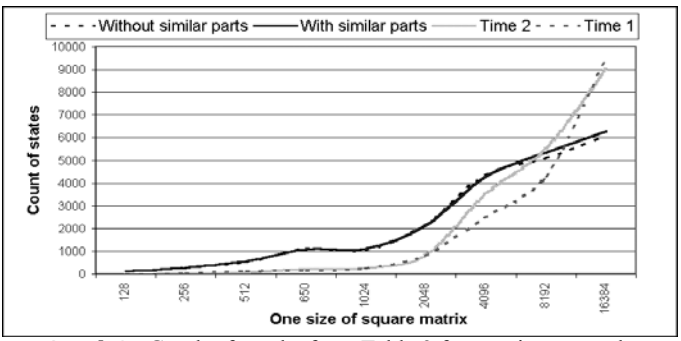
Matrix	time (XY)	X states	Y states	time	states
128 x 128	0,1	72	132	0,1	134
	0,1	111	119	0,2	118
256 x 256	0,3	129	254	0,8	271
	0,3	163	249	0,9	280
512 x 512	0,3	310	516	1,2	552
	0,5	289	491	2,3	516
650 x 650	1	427	813	4,5	1080
	1	419	835	3	1126
1024 x 1024	1,1	740	1027	4,9	1120
	1,2	593	1003	5	1044
2048 x 2048	1,2	1008	1028	16	2161
	1,2	1279	2027	16	2145
4096 x 4096	1	2019	4049	70	4289
	1	2016	2052	50	4353
8192 x 8192	2,6	4010	7350	110	5350
	3	4304	3890	86	5112
16000 x 16000	*0,1*	*13*	*13*	*240*	*14*
16384 x 16384	5	6218	8192	180	6308
	5,5	5905	8100	190	6100
32000 x 32000	6,5	4929	8190	NA	NA
64000 x 64000	16	8111	10450	NA	NA

Highlighted row is the worst case for our solution for matrix approach (Source matrix contains only one element at unlikely position.) It is clear that offset approach is faster for larger matrixes but produces more states. Matrix approach is better for

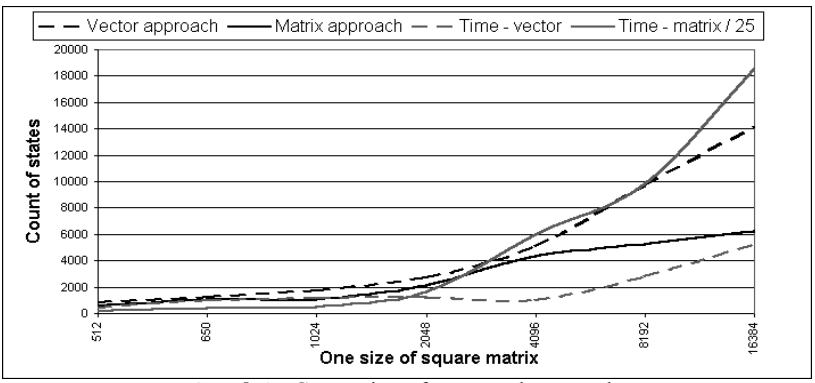
compression of small matrixes. If we want to have more compressed large matrix then the matrix approach is useful too but at the expense of machine time. For lucidity see following graph, where time is only on informative scale.



Graph 1. Graph of results from Table 2 for offset approach, where count of states is X states plus Y states.



Graph 2. Graph of results from Table 2 for matrix approach.



Graph 3. Comparing of presented approach.

2.4 Changes in source

In this section we describe in brief, how to solve the changes in the matrix. If we compress the matrix with traditional algorithm (e.g. zip, LZW, Huffman, etc.) and some element is changed, we must re-compress all matrixes every time. But if we represent matrix as a FSA, we can change/re-compress only the corresponding part of resultant automaton, the one with changed element.

There exist at least three basic solutions for selection of corresponding part of Finite State Automata or corresponding sub-square of source matrix:

1) Re-calculating the biggest corresponding sub-square:
This approach leads to a big quantum of data manipulation (up to one quarter), but with this approach we can reach the high compression ratio. The method is useful for all types of source matrixes.

2) Re-calculating the least corresponding sub-square:
This approach re-calculates the least quantum of data (approximately tens elements), but with this approach we can gain only very small compression ratio and changes often lead to the growth of the automata. Compression become here the disutility, but if we want only obtain the interesting information from our resultant automata, this method is useful too. This approach is useful for all types of source matrixes.

3) Re-calculating the optimal corresponding sub-square:
Retrieval of such sub-square may be difficult, but in most cases, it shows that it has no sense to work with sub-square greater than three or four least corresponding sub-squares. Naturally, it greatly depends on the character of the matrixes. If we know that the matrixes contain many equal blocks, we can state the amount of levels which we should still take in consideration. This choice naturally has not influence on algorithm, but only on machine time and resultant size of compressed matrixes.

3 Resultant aggregate / database

3.1 Resultant automata

Composition is useful for storing resultant FSA in one structure with value-added information. In this section, we describe only necessary generalized procedure, for more information read [9]. This procedure can be used for both approaches; *object oriented* and *prevailing* approach. This approach can be simply upgraded to loss-composition and makes possible to save more space and setup some additional options. We focus on this in future work.

Procedure *Composition Automaton for Storage*
For given automaton A and automaton B (resultant automaton from previously composition) compute new resultant automaton $B' \in A \cup B$ and combine similar parts of both ones.

Procedure Composition Automaton (Automaton *A*, Stored automaton *B*)
Assign state q_x from *A* to the corresponding state in stored automaton *B*.
if such case does not exists, assign a new state and take q_{x+1} from *A*.
end if
for all state of automaton *A* **do**
if not exists edge from state q_i labeled with same word *w* as edge from correspond state in stored automaton **then**
create a new edge labeled *w* to a new state *i*
otherwise
take next edge
end if
if all edges from actual state is processed, take next state
end if
end for
end procedure

This principle can be used for no-loss or loss compression for saving matrixes. Additional information can be obtained from structure of resultant automaton, for example the information about the similarity of the stored matrixes or its parts, common lines, etc. We can also easily get the group of equal matrix parts.

3.2 Focus on a interesting information

If we store the source matrixes in more than one automaton, we can focus on the interesting part of the matrix and compute the automaton with various lengths. This principle was introduced in [10].

On other part of matrix, we can compute automaton with less number of states. For this purpose, we can use the pattern matrix shown in table 3, where the values in cells are the counts of profundity of automaton, which represents that part of matrix. This matrix can be used for some image, see figure 4. This principle can be used only for loss compression (e.g. images, signals, etc.). The part with less count of states stores much fewer information than the part with more states.

It is clear that with this principle we can save much more space preserving high information value of data. We can transfer only interesting part of matrix or any nearest part and save machine time or network capacity. It is sufficient to choose a state from resultant automata, which represents the interesting part of the matrix, and operate with this as with the root. This principle is used in the automata composition. Procedure for focusing on interesting information is very simple. Pattern may be arbitrary.

Now we have a background for using finite state automata as a database with included information.

Procedure *Focus on interesting information*
For given matrix *M* and pattern matrix *P* compute resultant automaton. This procedure use procedure *Construct Automaton for Compression (CAfC),* there in before.

Procedure Focus on interesting information (Matrix *M*, pattern matrix *P*)
i = *j* = 0
create state *0* and assign $u_0 = M$
assume $u_i = M_w$
loop
for $x \in S$ (part of pattern matrix)
assume $|w| = P(u_i)$
$i = CAfC(M, \text{new } w)$
j++;
end for
if *i* = = *j* **than**
Stop (all parts from pattern are processed)
else *S* = *S* + 1
end if
end loop
end procedure

Table 3. Example pattern for procedure *Focus on interesting information*

2	2	2	2	2	2	2	2	2	2	...	2
2	2	3	3	3	3	3	3	3	2	...	2
2	2	3	4	4	4	4	4	3	2	...	2
2	2	3	4	5	5	5	4	3	2	...	2
2	2	3	4	5	6	4	4	3	2	...	2
2	2	3	4	5	5	5	4	3	2	...	2
2	2	3	4	4	4	4	4	3	2	...	2
2	2	3	3	3	3	3	3	3	2	...	2
2	2	2	2	2	2	2	2	2	2	...	2

Fig. 4. Example image with used focus on interesting information.

4 Conclusions

In this paper was summarized an idea to use finite state automata as a tool for specification and compression of data aggregates. We compared two basic approaches of computing the resultant automata, namely: the matrix approach and the vector (or offset) approach. The first one is better applicable for representation and compression of smaller matrixes and for manipulation after decompression. The vector approach is pretty faster for larger matrixes, but on the other hand it produces more states. If we want to have large matrix to be more compressed (independently on the machine time) then the matrix approach is useful too. Both methods can be loss or loss-free, and both types have high predicate ability about stored matrixes and save space and network capacity.

The linked resultant automaton is able to create matrix database with included value and to make manipulation with matrixes easier. We also described simple and generalized procedure *focus on interesting information* in the source data and some illustrative results were shown. This procedure can make a better compression ratio together with maintaining the high information level of data.

5 References

1. Alur, R. and Dill, D. L. A Theory of Timed Automata. In Theoretical Computer Science, 126(2):183–235, 1994.
2. Daniela Berardi, Fabio De Rosa, Luca De Santis and Massimo Mecella. Finite State Automata as Conceptual Model for E-Services. In Integrated Design and Process Technology, IDPT- 2003, June 2003.
3. K. Culik II and J. Kari. Image compression using weighted finite automata. In Computers & Craphics, 17:305–313, 1993.
4. K. Culik II and V. Valenta. Finite automata based compression of bi-level and simple color images. In Computers & Craphics, 21:61–68, 1997.
5. K. Culik II and J. Kari. Image compression Using Weighted Finite Automata, in Fractal Image Compression. In Theory a Techniques, Ed. Yuval Fisher, Springer Verlag, pp 243-258, 1994.
6. J.E.Hopcroft and J.D.Ullman. Introduction to automata theory, languages and computation. In Addison-Wesley, 1979.
7. Marian Mindek. Finite State Automata and Images. In Wofex 2004, PhD Workshop, Ed. V. Snášel, ISBN: 80-248-0596-0, 2004
8. Marian Mindek. Finite State Automata and Image Recognition. In Dateso 2004, Ed. V. Snášel, J. Pokorný, K. Richta, pp 132-143, ISBN: 80-248-0457-3, 2004
9. Marian Mindek. Finite State Automata and Image Storage. In Znalosti 2005, Eds. Lubomír Poplínksý, Michal Krátký, ISBN: 80-248-0755-6
10. Marian Mindek. Konečné automaty jako obrázky s multi-rozlišením. In posters Znalosti 2005
11. W3C (2004) XML Protocol. XML Protocol Web Page http://www.w3.org/XML (January 2005)

Characteristics of cosymmetric association rules

Michal Burda, Marian Mindek, and Jana Šarmanová

Department of Computer Science, VŠB – Technical University of Ostrava
17. listopadu 15, 708 33 Ostrava–Poruba, Czech Republic
{michal.burda, marian.mindek, jana.sarmanova}@vsb.cz

Abstract. Association rules are essential data mining tool and as such has been well researched. Many new types of association rules based on both categorial or quantitative data have been founded ([8], [7], [2], [4]). Our work is directed to the theoretical features of association rules; especially, we study a specific class of association rules called *δ-cosymmetric rules*. We present here some interesting properties of such rules and provide a definition of rules expressing the significant difference in position, as an example. We show here that even the usual implicational rules are special cases of δ-cosymmetric rules.

Key words: Cosymmetric rules, association rules, typed relations, data mining

1 Preface

This paper is intended to motivate the rise of a new class of association rules called *δ-cosymmetric rules*. First of all, we describe here briefly the notions of the *Logic of typed relations* used to write the association rules down (for more information see [6]). After that, we provide some motivating examples of the representative cosymmetric rule types. We also study several features of the δ-cosymmetric rules and define the δ-cosymmetric rule of significant difference in position. The end of this paper is dedicated to some notes on how to mine the δ-cosymmetric rules.

2 Logic of typed relations

In [6], we have developed the *Probabilistic Logic of Typed Relations* (PLTR) suitable for the formal association rules representation. In this section we briefly and informally describe main notions of that logic to understand the meaning of its formulae.

The main notion of PLTR is *typed relation*. Typed relation can be simply viewed as a data table with finite number of columns and rows. Each column represents one *attribute* and a set of such attributes is a *type* of the relation.

A typed relation is similar to classical concept of mathematical relation. We can perform usual set operations as union (∪), intersection (∩) or difference

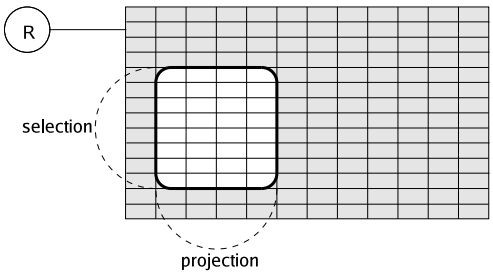


Fig. 1. Selection and projection on the relation R.

$(-)$. Furthermore, there exist two crucial relational operations: *selection* and *projection*. Selection is an unary operation of the form

$$R(c_1 \wedge c_2 \wedge \neg c_3)$$

where R is typed relation and $c_1 \wedge c_2 \wedge \neg c_3$ is a formula called *selection condition*. Selection is used to *select* only the rows satisfying the given condition. For example, when R is a data table (typed relation) of university students, the selection

$$R(\mathsf{age} > 25)$$

picks only the students older than 25. The projection on relation R is an unary operation of the form

$$R[A_1, A_2, \ldots, A_n].$$

The projection is used to take out only several columns (attributes) of the relation R. The choosed attributes are simply written in the comma-separated list in the brackets. The projection

$$R[\mathsf{name}, \mathsf{date\_of\_birth}]$$

results simply in the two-column data table with student's basic personal information. Obviously, we can combine selection and projection together to pick up an arbitrary sub-relation of the original typed relation, e.g.

$$R(\mathsf{age} > 25)[\mathsf{name}, \mathsf{date\_of\_birth}],$$

which results in a relation of basic personal information of students older than 25. (See also figure 1.) The rules written in PLTR use the relational operations described above to explicitly express a knowledge. For example,

$$R(\mathsf{age} > 65)[\mathsf{blood\_pressure}] >^{\star}_{mean} R(\mathsf{age} < 21)[\mathsf{blood\_pressure}]$$

tells that the blood pressure of people older than 65 is *in average* significantly higher than for people younger than 21. In the above rule we use a mapping $>^{\star}_{mean}$ to express the strong difference in the mean value between two "data columns". (See also figure 2.) The mapping $>^{\star}_{mean}$ is simply a function, which computes a truth value of the strong difference in mean from the given two typed relations.

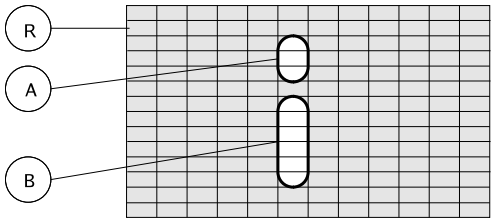


Fig. 2. Comparison of the two disjunctive sub-tables.

3 The δ-cosymmetric rules

There exist a wide variety of the association rule types. The best-known are the rules in the implicational form, which say that when the object satisfies some condition (called *antecedent*), it (very probably) gratifies some other condition (*succedent*), e.g.:

$$\mathsf{tequila} \wedge \mathsf{salt} \Rightarrow \mathsf{lemon}. \tag{1}$$

This rule simply says that customers who buy tequila and salt often buy lemons, too. However, there are many other rule types (e.g. associational, correlational etc. – see [1], [2], [5], [7], [8], [9]). It is not our goal to mention each of them. We preferable move the focus to the rules, which we later name δ-*cosymmetric*. Consider the subsequent rule from [4]:

$$\mathsf{sex} = \text{“female”} \;\Rightarrow\; \mathsf{wage}\text{: mean} = \$7.90/\text{hr (overall mean } \mathsf{wage} = \$9.02). \tag{2}$$

It indicates that the women's wage mean is significantly different to the rest of examined objects. That is, the rule says that women earn in average less than men. (The overall wage is in the rule for information only. To be statistically consistent, we must compare two disjoint sets of values, e.g. female againts male – see [4].) In general, the statistical test in the background of the rule *compares two sets of quantitative data* – women's wage against the wage of the remaining data table (in fact, against men's wage). We can apply the same mechanism and mine similar rules, e.g.:

$$\mathsf{non\text{-}smoker} \wedge \mathsf{wine\text{-}drinker} \;\Rightarrow\; \mathsf{life\text{-}expectancy} = 85 \quad (\text{overall} = 80). \tag{3}$$

Such rule says that people who drink wine and do not smoke live in average longer than the other people. One can see, we *compare* the life expectancy of people who don't smoke and drink wine against the rest of the data table. Such property is more visible when re-writing the original rules (2) and (3) (see also [4]) into PLTR:

$$R(\mathsf{sex} = \text{“female”})[\mathsf{wage}] <^{\star}_{mean} R(\mathsf{sex} \neq \text{“female”})[\mathsf{wage}] \tag{4}$$

and

$$R(\mathsf{non\text{-}smoker} \wedge \mathsf{wine\text{-}drinker})[\mathsf{life\text{-}expectancy}] >^{\star}_{mean} R(\neg(\mathsf{non\text{-}smoker} \wedge \mathsf{wine\text{-}drinker}))[\mathsf{life\text{-}expectancy}]. \tag{5}$$

Our research shows that many types of the associational rules can be transformed to the fashion of *comparing* "something" against "something else" (later in this paper, we mention some of them). Thus, it is natural to expect that such rules will have some equal properties and that it will behave similarly in alike situations. Therefore it is reasonable to identify the common features and use them in general definition of a new class of association rules. Later in this paper we try to do so and name the class of such association rules the *δ-cosymmetric rules*.

Moreover, it is obvious to contemplate rules of type (4) or (5) as formulae of PLTR. That is, one can treat the symbol $<^{\star}_{mean}$ as a predicate, whose truth value is the probability (quantity in interval $[0, 1]$). Such approach corresponds to the fact that the statistical test gets never the absolute truth – there is always a chance (non-zero probability) of a false result. In [6], we have developed a logic, whose truth values are probability intervals $i = \langle l, h\rangle$ where $0 \leq l \leq h \leq 1$.

3.1 Domain

The following subsections try to highlight some properties that are common in the class of association rules we want to name *δ-cosymmetric*. After that, we provide the first prototype definition of what δ-cosymmetric rule is.

We start with the domain of the δ-cosymmetric predicate $<^{\star}$. We can see, the rules of type (4) or (5) compare two typed relations. It is natural to expect that when $\langle A, B\rangle$ is comparable then $\langle B, A\rangle$ is comparable too.

Let $\mathcal{R}$ is a set of all typed relations. We may expect that each δ-cosymmetric predicate's domain D equals to the carthesian product of some set of typed relations:

$$\exists K \subseteq \mathcal{R} : D = K \times K.$$

This property tells us that for each typed relations $A, B \in K$, $\langle A, A\rangle$, $\langle A, B\rangle$ and $\langle B, A\rangle$ are comparable by the δ-cosymmetric predicate. That is, one can ask the truth value of the formulae $<^{\star}(A, A)$, $<^{\star}(A, B)$, $<^{\star}(B, A)$ for each $A, B \in K$.

3.2 Minimum difference

When mining the rules of type (4), it is useful to introduce an user-definable *minimum difference parameter* δ. (See also [4].) Its purpose is as follows: Finding conditions for which the means of some attribute are merely different does not lead to interesting information. If we were to discover, for example, a group of people with life expectancy five days more than the rest population, it may not be of interest to us even if it passes a statistical test.

The same concept can be used when comparing variances, probability or anything else. – The next thing common to each cosymmetric rule is the possibility to employ the minimum difference δ to it.

In the following, we will write the rule of the minimum difference δ the subsequent way:

$$R(C_1)[A] >^{\star}_{\delta} R(C_2)[A] \tag{6}$$

or prefixually:

$$>^{\star}_{\delta}\big(R(C_1)[A],\ R(C_2)[A]\big). \tag{7}$$

E.g. see the rule of the difference in wage of at least \$5:

$$R(\mathsf{sex} = \text{“female”})[\mathsf{wage}] <^{\star}_{mean;\$5} R(\mathsf{sex} \neq \text{“female”})[\mathsf{wage}]. \tag{8}$$

3.3 Non-symmetricity

In the following, we will need to define *the negation* of formula F. Suppose F is formula of PLTR (e.g. (4)) whose truth value is $i = \langle l, h\rangle$. (It stands for the fact that F is true with probability $p \in [l, h]$.) We define a truth value of formula's F negation (denoted $\neg F$) as $i' = \langle 1 - h,\ 1 - l\rangle$.

The third common feature of rules similar to (4) is its non-symmetricity. Suppose we are convinced of the validity of the rule $>^{\star}(A, B)$. What can we say about the truth value of the rule $>^{\star}(B, A)$? It is clear, if values of relation A are significantly higher than values of relation B, the contrary statement can't be true as well (so the formula (10) holds).

More generally, the truth value of a statement "objects of relation B are minimally over δ less than objects of relation A" equals to a negation of the statement "objects of relation A are minimally over $(-\delta)$ less than objects of the relation B". Formally written:

$$<^{\star}_{\delta}(B, A) \ \Leftrightarrow\ \neg\big(<^{\star}_{-\delta}(A, B)\big). \tag{9}$$

When $\delta = 0$ is omitted, it leads to

$$<^{\star}(B, A) \ \Leftrightarrow\ \neg\big(<^{\star}(A, B)\big). \tag{10}$$

3.4 Monotony

Let $>^{\star}_{\delta_1}\big(A, B\big) = \langle l_1, h_1\rangle$ and $>^{\star}_{\delta_2}\big(A, B\big) = \langle l_2, h_2\rangle$ where $>^{\star}$ is a predicate similar to the previously discussed. One can observe that the following holds all the time:

$$\big(\delta_1 < \delta_2\big) \Rightarrow \big((l_1 \geq l_2) \wedge (h_1 \geq h_2)\big). \tag{11}$$

Informally, this property says that the increase of the minimum difference δ leads to the reduction of the rule's probability.

3.5 Quasi-transitivity

We name *probable* the rule, which truth value $i = \langle l, h\rangle$ satisfies the condition $0,5 < l$. Let $<^{\star}_{\delta}(A, B) = \langle l_1, h_1\rangle$, $<^{\star}_{\delta}(B, C) = \langle l_2, h_2\rangle$ and $<^{\star}_{\delta}(A, C) = \langle l_3, h_3\rangle$. The last property of rules similar to (4) named *quasi-transitivity* tells the following:

$$\big((0,5 < l_1) \wedge (0,5 < l_2)\big) \Rightarrow \big(0,5 \leq l_3\big). \tag{12}$$

Informally, when some sub-table A is *probably* lower than B and B is *probably* lower than C, it implies that A is probably not higher than C.

Please note, we can't say that the probability of $A <^{\star} C$ is higher or equals to the maximum or minimum of the probabilities of $A <^{\star} B$ and $B <^{\star} C$, because such condition holds in fact very seldom.

3.6 The definition of δ-cosymmetric rules

Actually, we are still working on the precise definition of the δ-cosymmetric relationship predicate. We try to unhide the important properties of the rules similar to (4). The subsequent definition should be considered as the first prototype of our effort. As our knowledge about the rules increases, we will modify the definition to better pick up the reality.

Definition 1. *Let $\mathcal{R}$ be the set of all typed relations, V the set of all truth values, $K \subseteq \mathcal{R}$ and $D = K \times K$. We name $<^\star$* the cosymmetric predicate schema *if $<^\star$ is a set of relationship predicates $<^\star_\delta$: $D \to V$ (defined for each $\delta \in \mathbb{R}$) and if the following holds:*

1. *For each typed relations $A, B \in K$ and $\delta \in \mathbb{R}$ holds:*

$$<^\star_\delta (A, B) = \neg\big(<^\star_{-\delta} (B, A)\big),$$

2. *For each typed relations $A, B \in K$ and $\delta_1, \delta_2 \in \mathbb{R}$ and $i_1 = \langle l_1, h_1\rangle$, $i_2 = \langle l_2, h_2\rangle$ such that $<^\star_{\delta_1} (A, B) = i_1$, $<^\star_{\delta_2} (A, B) = i_2$ holds:*

$$\big(\delta_1 < \delta_2\big) \Rightarrow (l_1 \geq l_2) \wedge (h_1 \geq h_2),$$

3. *For each typed relations $A, B, C \in K$ and $\delta \in \mathbb{R}$ and $i_1 = \langle l_1, h_1\rangle$, $i_2 = \langle l_2, h_2\rangle$, $i_3 = \langle l_3, h_3\rangle$, such that $<^\star_\delta (A, B) = i_1$, $<^\star_\delta (B, C) = i_2$, $<^\star_\delta (A, C) = i_3$, holds:*

$$\big((0,5 < l_1) \wedge (0,5 < l_2)\big) \Rightarrow \big(0,5 \leq l_3\big).$$

The elements $<^\star_\delta$ of the set $<^\star$ are called δ-cosymmetric relationship predicates. *The set D is also called* the domain of the cosymmetric predicate schema.

4 Concrete δ-cosymmetric predicates

In the above section we have discussed several properties of a so-called cosymmetric rules. In this section, we provide an exemplary definitions of such rule type.

4.1 Cosymmetric rules of significant difference in position

The idea for cosymmetric rules of significant difference in position is subsequent. One may have data which are quantitative and may ask, for which subsets of data the focused quantitative attribute is rather higher or lower in contrast to the rest (c.f. rule (4) or (5)). In the other words, one may enquire for all hypotheses about the differences in position that are supported within data. We can determine the difference and measure the significancy with appropriate statistical test of hypotheses.

For such purpose we use the Aspin–Welch statistical test (see [3]), which is two-sample test on means. The test is similar to the common Student's t test. It

assumes the two random samples X and Y to be normally distributed (there is no need of equal variances) and it tests the zero hypothesis $H_0 : \mathsf{E}X - \mathsf{E}Y = \delta$ against the two-sided alternative hypothesis $H_A : \mathsf{E}X - \mathsf{E}Y \neq \delta$. The test statistic is

$$T = \frac{\bar{X} - \bar{Y} - \delta}{S}, \quad \text{where} \quad S = \sqrt{\frac{S_X^2}{m} + \frac{S_Y^2}{n}}; \qquad \left(f = \frac{S^4}{\frac{S_X^4}{m^2(m-1)} + \frac{S_Y^4}{n^2(n-1)}} \right).$$

The hypothesis H_0 is rejected if $|T| \geq t_f(1 - \frac{\alpha}{2})$, where t_f is a distribution function of Student's distribution with f degrees of freedom.

Pursuant to the one-sided Aspin–Welch statistics, we can define the relationship predicate $<^\star_{AW;\delta}$ as follows.

Definition 2. *Predicate $<^\star_{AW;\delta}$ is a function where an interval of probability $i = \langle p, p \rangle$ is mapped the following way to each pair of typed relations $\langle X, Y \rangle$, which both are non-empty and both contain just one column.*

$$<^\star_{AW;\delta} (X, Y) = \langle p, p \rangle$$

for such p where $T = t_f(p)$ for T, f and t_f as above.

The usage example comes after. Suppose we have a data table D about patients suffering certain disease. Let such table contains categorial column sex and quantitative column pressure. One may be interested whether $D(\mathsf{sex} = \text{“male”})[\mathsf{pressure}]$ gives higher values than $D(\mathsf{sex} = \text{“female”})[\mathsf{pressure}]$. That is, one enquires the validity of the following rule:

$$D(\mathsf{sex} = \text{“male”})[\mathsf{pressure}] >_{AW;0} D(\mathsf{sex} = \text{“female”})[\mathsf{pressure}].$$

Now we can take a closer look at the Aspin–Welch predicate $<^\star_{AW;\delta}$ to see, whether it has all the properties enumerated in section 3.6.

Theorem 1. *The set of all Aspin–Welch relationship predicates $<^\star_{AW;\delta}$ $(\forall \delta)$ is cosymmetric predicate schema.*

Proof. (a) Non-symmetricity. We should check the equivalence (9). Suppose typed relations X, Y and value δ. Let $>^\star_{AW;\delta} (X, Y) = \langle p_1, p_1 \rangle$ and $>^\star_{AW;-\delta} (Y, X) = \langle p_2, p_2 \rangle$. We are going to show that $p_1 = 1 - p_2$. Computing the values of p_1 and p_2 means accordingly to the definition 2 computing the T characteristics. Thus,

$$T_1 = \frac{\bar{X} - \bar{Y} - \delta}{S} = t_f(p_1) \qquad \text{and} \qquad T_2 = \frac{\bar{Y} - \bar{X} - (-\delta)}{S} = t_f(p_2).$$

We see that $T_1 = -T_2$, so $t_f(p_1) = -t_f(p_2)$. It is commonly known that $t_f(p) = -t_f(1 - p)$, so $p_1 = 1 - p_2$.

(b) Monotony. It is commonly known that $t_f(p)$ is monotone, so when we increase δ, the value of characteristics T gets lower and so does the value of the resultant probability p.

(c) Quasi-transitivity. the validity of quasi-transitivity condition is evident from the fact that $\forall f \in \mathbb{N} : t_f(0, 5) = 0$.

4.2 Funded cosymmetric rules

We can go on and define various other δ-cosymmetric predicates similar to the definition of Aspin–Welch predicate. We don't have enough space for such definitions, so let us leastwise mention some possibilities.

We can define many other predicates for determining the significant difference in position. Such definitions could be based on various existing statistical tests – it is possible e.g. to employ the *rank tests* to achieve robust cosymmetric predicates etc. Similarly to the significant difference in position, we can define predicates deciding of the difference in variance (dispersion). For example, we can mine rules telling us whether the presence of some attribute puts there significant increase of dispersion of some other attribute etc.

We can employ the two-sample tests on binomial distribution to generate rules about discrete attributes. Generally said, almost every two-sample statistical test may be considered to be used in a definition of appropriate cosymmetric predicate.

Let's have a look on the *implicational rules* of type (1). We show that we can define δ-cosymmetric rules that are analogous to them. Before doing so, we should describe shortly the meaning of the implicational rules.

The GUHA method ([8], [7]) works with the so-called *generalized quantifiers*. These quantifiers form the base for the association rule creation. The rules are of the form $\varphi \sim \psi$, where φ and ψ are formulae and $\sim$ the generalized quantifier. The truth of the rule is determined from a *4-field table* (see table 1), which summarizes the amount of objects satisfying ceratin configurations.

Table 1. 4-field table of φ and ψ

	ψ	$\neg\psi$
φ	a	b
$\neg\varphi$	c	d

The a value denotes the number of objects satisfying both φ and ψ, b is the number of objects satisfying φ and not ψ etc.

The quantifier $\Rightarrow_{p,base}$ called also *the funded implication* is defined for $0 < p \leq 1$ and $base \geq 0$ as follows. The rule $\varphi \Rightarrow_{p,base} \psi$ is true if and only if (*iff*)

$$\frac{a}{a+b} \geq p \wedge a \geq Base.$$

More on such rules can be read from [8], [7] or [10]. The example of the rule based on the funded implication is (1).

Now, we provide a definition of a predicate that is similar to the quantifier of funded implication. After that, we show that it is δ-cosymmetric.

Definition 3. *Let A and B be the typed relations, each containing exactly one column with values from the set $\{0,1\}$ and let $\delta \in [-1,1]$. Let us denote $sum(A)$ the number of A's rows possessing "1". We define the* Funded relationship predicate $<^{\star}_{fnd;\delta}$ *as follows:*

$$>^{\star}_{fnd;\delta}(A,B) = \langle 1,1\rangle \qquad \textit{iff} \qquad \frac{sum(A)}{sum(A)+sum(B)} > \frac{1+\delta}{2},$$

$$>^{\star}_{fnd;\delta}(A,B) = \langle 0,5,0,5\rangle \qquad \textit{iff} \qquad \frac{sum(A)}{sum(A)+sum(B)} = \frac{1+\delta}{2},$$

$$>^{\star}_{fnd;\delta}(A,B) = \langle 0,0\rangle \qquad \textit{iff} \qquad \frac{sum(A)}{sum(A)+sum(B)} < \frac{1+\delta}{2}.$$

Theorem 2. *The set of all funded relationship predicates $<^{\star}_{fnd;\delta}$ $(\forall\delta)$ is cosymmetric predicate schema.*

Proof. (a) Non-symmetricity. We must prove that $\frac{a}{a+b} > \frac{1+\delta}{2}$ iff $\frac{b}{a+b} < \frac{1-\delta}{2}$.

$$\frac{a}{a+b} > \frac{1+\delta}{2} \quad\Leftrightarrow\quad \frac{2a}{a+b} - 1 > \delta \quad\Leftrightarrow\quad \frac{2a+2b-2b}{a+b} - 1 > \delta \quad\Leftrightarrow$$
$$\Leftrightarrow\quad 1 - \frac{2b}{a+b} > \delta \quad\Leftrightarrow\quad \frac{b}{a+b} < \frac{1-\delta}{2}.$$

(b) Monotony and *(c) Quasi-transitivity* are obvious.

If we omit the minimum support constraint in the definition of the funded implication, we get the same rules as with the funded δ-cosymmetric predicate. In the other words, the rule

$$\varphi \Rightarrow_{p,0} \psi$$

is true on data table R iff the following rule has truth value equal to $\langle 1,1\rangle$:

$$R(\psi)[\varphi] >^{\star}_{fnd;(2p-1)} R(\neg\psi)[\varphi].$$

As a result we can say that implicational GUHA rules are just special cases of δ-cosymmetric rules. This surprising result convinced us of the importance of the δ-cosymmetric rules research.

5 Schemes of δ-cosymmetric association rules

Consider the general pattern of a δ-cosymmetric rule:

$$R(C_1)[A] >^{\star} R(C_2)[A]. \tag{13}$$

When mining such rules, we can generate and test virtually every combination of C_1, C_2, A, but doing so makes not much sense. It is because the association rule mining process results often in a wide range of association rules and it is sometimes hard to be acquainted with it. Moreover, only several combinations of conditions C_1 and C_2 are easy to interpret. Consider the following rule – although it may be true, the analyst has probably no usage for it.

$$R(\mathsf{eyes} = \text{“blue”} \wedge \mathsf{sex} = \text{“male”})[\mathsf{fat}] >^{\star} R(\mathsf{age} > 30 \wedge \mathsf{wage} < \$200)[\mathsf{fat}] \tag{14}$$

In the following, we try to recognize the patterns of δ-cosymmetric rules of better interest than general pattern (13).

5.1 Scheme "one-against-the-rest"

The easiest pattern of interesting δ-cosymmetric rules is

$$R(C)[A] >^{\star} R(\neg C)[A]. \tag{15}$$

We take one condition C and compare values of some quantitative attribute A for two sub-tables where the first satisfies the given condition C and the second doesn't. Such rules express the condition at which the values of attribute A are "somehow" significantly higher (or lower) than the rest of the data table. This basic pattern we name *one-against-the-rest*.

A pattern similar to (15) is *conditional one-against-the-rest*:

$$R(C_1 \wedge C_2)[A] >^{\star} R(\neg C_1 \wedge C_2)[A]. \tag{16}$$

This pattern stands for "when considering only values fulfilling the condition C_2, the additional condition C_1 indicates the significant increase of value A (in the sense of $>^{\star}$)." That is, we fisrt restrict ourselves on data rows satisfying C_2 only and then we search simply the one-against-the-rest rules on them.

5.2 Scheme "one-on-one"

The pattern *one-on-one* is a little more tricky. It is good in situations, when we want to compare groups created accordingly to one categorial attribute. Suppose attribute B be categorial with domain $\{b_1, b_2, \ldots, b_n\}$. Let moreover attribute A be quantitative. The pattern one-on-one is as follows:

$$R(B = b_i)[A] >^{\star} R(B = b_j)[A] \qquad (\text{for } i \neq j). \tag{17}$$

The rule of such type means: "The objects with value b_i in attribute B involve significantly higher values of attribute A than objects with value b_j in attribute B." Generally, we can generate and test $\binom{n}{2}$ different hypotheses for a categorial attribute with n various values.

We can add an additional condition C to form *conditional one-on-one* pattern, too:

$$R(B = b_i \wedge C)[A] >^{\star} R(B = b_j \wedge C)[A] \qquad (\text{for } i \neq j). \tag{18}$$

6 Some notes of how to reduce the number of resultant rules

The large size of the association rule mining results is the common problem. Analyst hardly orientates himself or herself in a big list of mined rules. Therefore, we enumerate here some hints of how to prune the result from less-interesting rules and so to restrict the resultant δ-cosymmetric formulae to the reasonable amount.

1. *The significance level* – the basic restriction on eventual rules is stating the minimum probability of its validity – in the other words, one may set a number p_{min} and throw away every rule, which truth value is below that threshold. Significance level can be pre-set to any of the usual values as 0,95 or 0,99.
2. *Contradictory conditions* – the conditions appearing in the rule should be contradictory. That is, when considering the rule

$$R(C_1)[A] >^{\star} R(C_2)[A]$$

   then the formula $C_1 \wedge C_2$ should be contradiction. Rules satisfying that criterion are more easily interpretable and we avoid the uncorrect statistical comparing of non-disjunctive samples. (Compare with rule (14). Note also that the rules based on one-against-the-rest or one-on-one are all of contradictory conditions.)
3. *Minimum support* – minimum support is the best-known instrument for pruning away the non-interesting conditions from which the association rules are going to be formed. The minimum support criterion simply says that there must exist minimally *minsup* objects satisfying condition that appears in the rule. If not, such condition isn't used in the association rule generating process. The definition of the *minsup* value greatly improves the efficience of association rule mining algorithms (see [1], [9], [11] for more information). Minimum support should be set by expert only.
4. *Minimum difference* – setting the minimum difference δ is analogous to the stating of minimum rule probability. Doing so we express that we are interested in the rules, which confirm the dissimilarity to be at least of size δ. Minimum difference should be set by expert only.
5. *Easy-to-interpret rules only* – in section 5 we have shown that generating all possible rules makes no sense. One may to generate only the rules, which are easy to interpret. That is, we should generate rules conforming to the patterns discussed in section 5. A similar criterion on that topic is to use conditions in conjunctive form only.

7 Conclusion and future work

In this paper, we have introduced the new class of association rules – the δ-cosymmetric rules. We are the first who has shown, how to use the Probability logic of typed relations (PLTR, see [6]) to express rules of such type. This paper also shows the benefit of using PLTR as a language for writing the association rules in, too.

We have identified the basic properties of δ-cosymmetric rules and provided the definition of rules of significant difference in position, as an example. The second part of this paper was dedidacted to some notes on how to generate the δ-cosymmetric rules to obtain the interesting rules only.

This paper also presents two basic examples of concrete δ-cosymmetric rules: the Aspin–Welch predicate and the Funded predicate. The second is surprise for

us, since it shows that GUHA's implicational rules are just the special cases of more general δ-cosymmetric rules.

Our future work will address the deeper research of δ-cosymmetric rules. We will try to unhide more interesting features of that rule class. For example, our actual research shows that the cosymmetric rules can be used in the definition of a function that is *metric*. An interesting task will be undisputably the clustering using such metrics, etc.

We are also focused on finding the fast and efficient algorithm to mine the δ-cosymmetric rules. A lot of work was done in [4] by Aumann and Lindell. (However, they didn't know that they are mining cosymmetric rules – their algorithm should be slightly modified to comply the wide range of possible rule types.)

We are also interested in the methods of visualisation of δ-cosymmetric rules. The properties of δ-cosymmetric rules make rational to use the slightly modified Hasse's diagrams to visualize the rules mined according to the pattern "one-on-one" discussed above. We also work on employing the conceptual lattices to represent mined δ-cosymmetric rules.

References

1. Agrawal, R. Fast discovery of association rules. In *Advances in knowledge discovery and data mining* (1996), AAAI Press / MIT Press, pp. 307–328.
2. Agrawal, R., Imielinski, T., and Swami, A. Mining associations between sets of items in massive databases. In *ACM SIGMOD 1993 Int. Conference on Management of Data* (Washington D.C., 1993), pp. 207–216.
3. Anděl, J. *Statistické metody.* MATFYZPRESS, Praha, 1998.
4. Aumann, Y., and Lindell, Y. A statistical theory for quantitative association rules. In *Knowledge Discovery and Data Mining* (1999), pp. 261–270.
5. Berka, P. *Dobývání znalostí z databází.* Academia, Praha, 2003.
6. Burda, M., Hynar, M., and Šarmanová, J. Pravděpodobnostní logika typovaných relací. In *Znalosti, poster proceedings* (2005).
7. Hájek, P., and Havránek, T. *Mechanizing Hypothesis Formation.* Springer–Verlag, Berlin, 1978. Internet: `http://www.cs.cas.cz/~hajek/guhabook/` (May 2004).
8. Hájek, P., Havránek, T., and Chytil, M. K. *Metoda GUHA – automatická tvorba hypotéz.* Academia, Praha, 1983.
9. Han, J., and Kamber, M. *Data Mining: Concepts and Techniques.* Morgan Kaufmann Publishers, USA, 2000.
10. Rauch, J. Asociační pravidla a matematická logika. In *Znalosti* (Brno, 2004), pp. 114–125.
11. Rauch, J., and Šimůnek, M. Alternative approach to mining association rules. In *FDM* (Japan, 2002), pp. 157–162.

Text Compression: Syllables

Jan Lánský and Michal Žemlička

Charles University, Faculty of Mathematics and Physics
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
zizelevak@gmail.com, michal.zemlicka@mff.cuni.cz

Abstract. There are two basic types of text compression by symbols – in the first case symbols are represented by characters, in the second case by whole words. The first case is useful for very short files, the second case for very long files or large collections. We supposed that there exist yet another way where symbols are represented by units shorter than words – syllables. This paper is focused to specification of syllables, methods for decomposition of words into syllables, and using syllable-based compression in combination of principles of LZW and Huffman coding. Above mentioned syllable-based methods are compared with their counterpart variants for characters and whole words.

1 Introduction

Knowledge on structure of coded messages can be very useful for design of a successfull compression method. When compressing text documents, the structure of messages is dependent on used language. We can expect that documents written in the same language will have similar structure.

Similarity of languages can be seen according many aspects. Language classification can be made, for example, according their use of fixed or free word order or whether they have simple or rich morphology.

To the languages with rich morphology they belong for example Czech and German. In these languages is syllable a natural element logically somewhere between characters and words. Each word is often created by two or more syllables.

At the beginning we supposed that:

- Syllable-based compression will be suitable for middle-sized files, character-based compression will be more suitable for small files, and word-based compression will suit best for very large files.
- Syllable-based compression will be more suitable for languages with rich morphology.
- The number of unique syllables in given language is much lower than the number of unique words. This lead to lower memory requirements of syllable-based compression algorithms than of word-based ones.
- The sets of syllables of two documents written in the same language are usually more similar than sets of words of these documents.

© K. Richta, V. Snášel, J. Pokorný (Eds.): Dateso 2005, pp. 32–45, ISBN 80-01-03204-3.
VŠB – Technical University of Ostrava, Dept. of Computer Science, 2005.

2 Languages and Syllables

We can classify languages according their morphology. There are languages like English having simple morphology where from one stem there can be derived only a few word forms. There are also languages with rich morphology (like Czech) where from one stem it can be derived several hundred word forms.

We will demonstrate it on some examples from Czech and English. The verb *take* has in English only next 5 forms: *take*, *takes*, *taking*, *took*, *taken*. Czech verb *vzít*, which corresponds to the English verb *take*, has next 24 forms: *vzít*, *vzíti*, *vezmu*, *vezmeš*, *vezme*, *vezmeme*, *vezmete*, *vezmou*, *vzal*, *vzala*, *vzalo*, *vzali*, *vzaly*, *vezmi*, *vezměme*, *vezměte*, *vzat*, *vzata*, *vzato*, *vzati*, *vzaty*, *vzav*, *vzavši*, *vzavše*. Next difference is in creation of words with similar meaning and negations. In English there are used combinations of more words for getting different meaning, for example *get on*, *get off*. The negation in English is created by combination with word not, for example *not get on*. In Czech prefixes and suffixes are used instead. To English *get on*, *get off*, *not get on* correspond Czech *nastoupit*, *vystoupit*, *nenastoupit*. In Czech we can create from verb *skočit* (*jump* in English) using prefixes 9 next similar verbs: *přeskočit*, *nadskočit*, *podskočit*, *odskočit*, *rozskočit*, *naskočit*, *vskočit*, *uskočit*, *vyskočit*. For each of these verbs we can create their antonyms by using prefix *ne*: *neskočit*, *nepřeskočit*, *nenadskočit*, *nepodskočit*, *neodskočit*, *nerozskočit*, *nenaskočit*, *nevskočit*, *neuskočit*, *nevyskočit*. For each of these 20 verbs there exist 24 grammatical forms. So from this one word *skočit* we can derive over 400 similar words, but these words are composed from only a few tens of syllables.

2.1 Syllables

According Compact Oxford English Dictionary [10] syllable is defined as: 'A unit of pronunciation having one vowel sound, with or without surrounding consonants, and forming all or part of a word.'

As the decomposition to syllables is used in data compression, it is not necessary to decompose words into syllables always correctly. It is sufficient if the decomposition produces groups of letters that occur quite frequently. We therefore use simplified definition below that is not equivalent with the grammatically correct definition. 'Syllable is a sequence of sounds, which contains exactly one maximal subsequence of vowels.' This definition implies that the number of syllables in a word is equal to the number of maximal sequences of vowels in the same word. For example, the word *famous* contains two maximal sequences of vowels: *a* and *ou*, so this word is created from two syllables: *fa* and *mous*. Word *pour* contains only one maximal sequence of vowels *ou*, so whole this word is created by only one syllable.

Decomposition words into syllables is used for example for text formatting when we want to split word exceeding end of line. Disadvantage of this way is that we cannot decompose all words and some words must be left unsplitted.

One of the reasons why we selected syllables is that documents contain less unique syllables than unique words. Example Czech document (Karel Čapek:

Hordubal) with the size of 195 kB has 33,135 words form which are 8,071 distinct and 61,259 syllables from which are 3,187 distinct. English translation of bible [9] with the size of 4MB has 767,857 words from which are 13,455 distinct and 1,073,882 syllables from which are 5,604 distinct.

2.2 Problems in Decomposition of Words into Syllables.

Decomposition of words into syllables is not always unique. To determine it, we must often know origin of the word. Some problems will be demonstrated on selected Czech words. We supposed that for compression it is sufficient to use some approximation of correct decomposition of words into syllables. We supposed that this approximation would have only a small negative effect on reached degree of compression.

An example of non-uniqueness of decomposition of words into syllables is the word *Ostrava* which can be correctly decomposed into *Os-tra-va* and also into *Ost-ra-va*. Generally sequence of letters *st* is often a source of ambiguity of decomposition of Czech words to syllables.

An example of a variant decomposition of similar sequences of letters, which is caused by origin of words, is words *obletí* and *obrečí*, that have first two letters same. Word *obletí* (will fly around) was created by adding prefix *ob* to the word *letí* (flies). Word *obrečí* (will cry over) was created by adding prefix *o* to the word *brečí* (cries). So the word *obletí* is decomposed into *ob-le-tí*, word *obrečí* is decomposed into *o-bre-čí*. A big group of problems is brought by words of foreign origin and their adapted forms.

Sometimes it can by difficult to recognize real number of syllables of given word. Although the word *neuron* is a prefix of the word *neuronit*, these words have different decomposition to syllables. Word *neuron* is decomposed to *neu-ron*, word *neuronit* is decomposed to *ne-u-ro-nit*. In the first case the sequence of letters *neu* is composed by one syllable, in the second case it is composed from two syllables.

Full correctness of decomposition of words into syllables can be reached only at the price of very high effort. For the use in compression it is not important whether the decomposition is absolutely correct, but whether the produced groups of letters are frequent enough.

Other goal is to formalize terms letter, vowel, consonant, word, syllable, and language. When we try to define expressions vowel and consonant, we must interesting about position letter in the word. For example in Czech letters r and l can be according their context both vowels and consonants. In English similar role is played by the letter y.

2.3 Definition of Syllable.

Definition 1. *Let Σ be finite nonempty* set of symbols *(alphabet). Symbol $\lambda \notin \Sigma$ is called* empty word. *Let $\Sigma_{\mathrm{P}} \subseteq \Sigma$ be* set of letters*, then $\Sigma_{\mathrm{N}} = \Sigma \backslash \Sigma_{\mathrm{P}}$ is called* set of nonletters. *Let $\Sigma_{\mathrm{C}} \subseteq \Sigma_{\mathrm{N}}$ be* set of digits*, then $\Sigma_{\mathrm{Z}} = \Sigma_{\mathrm{N}} \backslash \Sigma_{\mathrm{C}}$ is called* set of special characters.

Definition 2. *Let Σ be finite nonempty set of symbols. Let $\Sigma_P \subseteq \Sigma$ be set of letters. Let $\Sigma_M \subset \Sigma_P$ be set of small letters, then $\Sigma_V = \Sigma_P \backslash \Sigma_M$ is called set of capital letters. If there exists bijection $\psi : \Sigma_M \to \Sigma_V$, then Σ_P is called* correct set of letters.

Note 1. The set of letters for most of natural languages is correct, but exists exceptions like German. In German letter ß can be according context or small or capital letter. If we want to work with non-correct sets of letters, we must modify definition 2.

Definition 3. *Let Σ be finite nonempty set of symbols. Let $\Sigma_P \subseteq \Sigma$ be set of letters.*

Let $\phi : (\Sigma \cup \{\lambda\}) \times \Sigma_P \times (\Sigma \cup \{\lambda\}) \to \{0, 1, 2, 3\}$ be a function. Let $\beta \in \Sigma_P$, let $\alpha, \gamma \in (\Sigma_P \cup \{\lambda\})$. Then:

- *If $\phi(\alpha, \beta, \gamma) = 0$ then β in context α, γ is called* vowel. *($\beta \in \Sigma_A$).*
- *If $\phi(\alpha, \beta, \gamma) = 1$ then β in context α, γ is called* consonant. *($\beta \in \Sigma_B$)*
- *If $\phi(\alpha, \beta, \gamma) = 2$ then β in context α, γ is consisting from vowel β_1 followed by consonant β_2. As β_1 and β_2 are together one letter, they cannot be split into different syllables.*
- *If $\phi(\alpha, \beta, \gamma) = 3$ then β in context α, γ is consisting from consonant β_1 followed by vowel β_2. As β_1 and β_2 are together one letter, they cannot be split into different syllables.*

Note 2. It is probable that there exist languages where we do not need to know context α, γ to decide if letter β is a vowel or a consonant. In both Czech and English the use of context is necessary.

In Czech can letters r and l be according their context used as vowels or as consonants. If α or γ are vowels, then $\beta = r$ (respectively $\beta = l$) is a consonant, in the opposite case it is a vowel. Examples: *mlčet*, *mluvit*, *vrtat*, *vrátit*.

In English the letter y in context of two vowels has the role of a consonant (example: *buying*).

In English words of type trying $\phi(r, y, i)$ has value 2, so y is composed from vowel y_1 followed by consonant y_2.

We supposed that there exist a language where $\phi(\alpha, \beta, \gamma)$ can have value 3, but in English and Czech it is not so.

The size of context one from left and right side is sufficient for Czech and for most of English words. We suppose that this size of context can be sufficient too.

Definition 4. *Let Σ be a finite nonempty set of symbols. Let $\Sigma_P \subset \Sigma$ be a set of letters. Let $\Sigma_C \subset \Sigma_N$ be a set of digits. Let $\Sigma_C \cap \Sigma_P = \emptyset$. Let $\alpha = \alpha_1, \ldots, \alpha_n$, $\alpha_i \in \Sigma$. If one of the following cases is valid, then α is called a* word over alphabet Σ.

If $\alpha_i \in \Sigma_Z$ for $i = 1, \ldots, n$, word α is called other.
If $\alpha_i \in \Sigma_C$ for $i = 1, \ldots, n$, word α is called numeric.
If $\alpha_i \in \Sigma_M$ for $i = 1, \ldots, n$, word α is called small.

If $\alpha_i \in \Sigma_V$ *for* $i = 1, \ldots, n$, *Word* α *is called* capital.
If $\alpha_1 \in \Sigma_V$ & $\alpha_i \in \Sigma_M$ *for* $i = 2, \ldots, n$, *word* α *is called* mixed.

Note 3. Numeric words and other-words are called together as words from non-letters. Small, capital, and mixed words are called as words from letters.

Definition 5. *Let* Σ *be finite nonempty set of symbols. Let* $\alpha = \alpha_1, \ldots, \alpha_n, \alpha_i \in \Sigma$. *Let* $\beta_1, ..., \beta_m$ *are words over alphabet* Σ. *Let* $\beta_i \cdot \beta_{i+1}$ *are not words over alphabet* Σ *for* $i = 1, \ldots, m-1$. *Let* $\alpha = \beta_1 \cdot ... \cdot \beta_m$. *Then* $\beta_1, ..., \beta_m$ *is called* decomposition of the string α into words.

Note 4. From definitions 4 and 5 follows that decomposition of strings into words is fast unique. There is an exception when after two ore more capital letters follows at least one small letter (example CDs), but this case is very rare in natural languages.

There exist two types of algorithms of decomposition of strings into words. Both types differ only in solving that exception. Algorithms of type A_1 create from this string capital-word and mixed-word (example C and Ds). Algorithms of type A_2 create from this string capital-word and small-word (example CD and s).

Words are decomposed into syllables (see Def. 9).

Definition 6. *Language* L *is an ordered 6-tuple* $(\Sigma, \Sigma_P, \Sigma_C, \Sigma_M, \phi, A)$, *where*

- Σ *is a finite nonempty set of symbols.*
- $\Sigma_P \subset \Sigma$ *is a correct set of letters.*
- $\Sigma_C \subset \Sigma_N$ *is a set of digits, where* $\Sigma_N = \Sigma \backslash \Sigma_P$.
- $\Sigma_M \subset \Sigma_P$ *is a set of small letters.*
- $\phi : \ (\Sigma \cup \{\lambda\}) \times \Sigma \times (\Sigma \cup \{\lambda\}) \to \{1, ..., 4\}$ *is a function which according definition 3 specify whether letter* β *in context* α, $\gamma \in (\Sigma \cup \{\lambda\})$ *is a vowel or a consonant.*
- A *is an algorithm which for each string* $\alpha = \alpha_1, \ldots, \alpha_n$, $\alpha_i \in \Sigma$ *finds some decomposition of given string into words.*

Definition 7. *Let* $L = (\Sigma, \Sigma_P, \Sigma_C, \Sigma_M, \phi, A)$ *be a language. Let* $\alpha, \gamma \in \Sigma_B^*$, $\beta \in \Sigma_A^*$, $|\beta| \in \{1, 2, 3\}$. *If* $\alpha \cdot \beta \cdot \gamma$ *is a word over alphabet* σ, *then it is syllable of the language* L.

Note 5. Notation $\alpha \in \Sigma_B^*$ (respective $\beta \in \Sigma_A^*$) mean that β is a sequence of consonants (respective vowels).

From definitions 4 and 7 follows that each syllable is also a word. So we will recognize five types of syllables: other syllables, numeric syllables, small syllables, capital syllables, and mixed syllables.

Condition that each syllable must be also word is necessary. For example, the string *xxAxx* is not word, therefore it cannot be (according Def. 7) syllable.

Definition 8. *Let* $L = (\Sigma, \Sigma_P, \Sigma_C, \Sigma_M, \phi, A)$ *be a language. Let* $\alpha = \alpha_1, \ldots, \alpha_n$, $\alpha_i \in \Sigma_P$ *is a word over alphabet* Σ.

If $(\exists k)\alpha_k \in \Sigma_A$, *then* α *is called* word decomposable into syllables.
If $(\forall k)\alpha_k \in \Sigma_A$ *then* α *is called* non-syllable word.

Definition 9. *Let* $L = (\Sigma, \Sigma_P, \Sigma_C, \Sigma_M, \phi, A)$ *be a language. Let* $\alpha = \alpha_1, \ldots, \alpha_n$, $\alpha_i \in \Sigma_P$ *is a word decomposable into syllables. Let* $\beta_1, ..., \beta_m$ *be syllables of the language L. Let* $\alpha = \beta_1 \cdot ... \cdot \beta_m$, *then* $\beta_1 \cdot ... \cdot \beta_m$ *is called* decomposing word α into syllable.

Definition 10. *Let* $L = (\Sigma, \Sigma_P, \Sigma_C, \Sigma_M, \phi, A)$ *be a language. Let P be an algorithm which input is document D decomposed into words by algorithm A into words* $\alpha_1, ..., \alpha_n$ *over alphabet* Σ. *If for all* α_i *they are valid both the following conditions, then P is called* algorithm of decomposition into syllables for language L.

- *If* α_i *is a word decomposable into syllables, then output of the algorithm is a decomposition of the word* α_i *into syllables.*
- *If* α_i *is a non-syllable word, numeric-word, or other-word, then output of the algorithm is a word* α_I.

Definition 11. *Let P be an algorithm of decomposition into syllables for language* L_1. *If B is an algorithm of decomposition into syllables for all other languages L, then we say that P is a universal algorithm of decomposition into syllables. If there exist a language* L_2 *for which P is not algorithm of decomposition into syllables, then we say that P is* specific algorithm of decomposition into syllables.

Note 6. Each universal algorithm of decomposition into syllables P has to use all information from definition of language L. In the other case we can construct language for which P will not be an algorithm of decomposition into syllables.

2.4 Examples of languages

There are two examples of languages: English and Czech. Czech languages has in comparison with English new diacritical letter.

English language can be characterized as $\mathrm{L_{EN}} = (\Sigma, \Sigma_P, \Sigma_C, \Sigma_M, \phi, A)$ where:

- Σ = ANSI character set
- $\Sigma_P = \{a, \ldots, z, A, \ldots, Z\}$
- $\Sigma_C = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- $\Sigma_M = \{a, \ldots, z\}$
- ϕ is defined as:
  - $(\forall \alpha \in (\Sigma \cup \{\lambda\}), \forall \beta \in (M \backslash \{y, Y\}), \forall \gamma \in (\Sigma \cup \{\lambda\}))\ \phi(\alpha, \beta, \gamma) = 0$
  - $(\forall \alpha \in (\Sigma \cup \{\lambda\}), \forall \beta \in (\Sigma_P \backslash M), \forall \gamma \in (\Sigma \cup \{\lambda\}))\ \phi(\alpha, \beta, \gamma) = 1$
  - $(\forall \alpha \in ((\Sigma \backslash M) \cup \{\lambda\}), \forall \beta \in \{y, Y\}, \forall \gamma \in ((\Sigma \backslash M) \cup \{\lambda\}))\ \phi(\alpha, \beta, \gamma) = 0$
  - $(\forall \alpha \in (\Sigma_P \backslash M), \forall \beta \in \{y, Y\}, \forall \gamma \in M)\ \phi(\alpha, \beta, \gamma) = 2$
  - $(\forall \alpha \in M, \forall \beta \in \{y, Y\}, \forall \gamma \in (\Sigma \cup \{\lambda\}))\ \phi(\alpha, \beta, \gamma) = 1$
  - $(\forall \beta \in \{y, Y\}, \forall \gamma \in M) \phi(\alpha, \beta, \gamma) = 1$
  - where $M = \{a, e, i, o, u, y, A, E, I, O, U, Y\}$

– $A = A_1$ from Note 4.

Czech language can be characterized as $\mathrm{L_{CZ}} = (\Sigma, \Sigma_P, \Sigma_C, \Sigma_M, \phi, A)$ where:

– Σ = ANSI character set
– Σ_P = {a,. . . ,z,A,. . . ,Z,á,č,ď,é,ě,í,ň,ó,ř,š,ť,ú,ů,ý,ž,Á,Č,Ď,É,Ě,Í,Ň,Ó,Ř,Š,Ť,Ú,Ů,Ý,Ž}
– Σ_C = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
– Σ_M = {a,. . . ,z,á,č,ď,é,ě,í,ň,ó,ř,š,ť,ú,ů,ý,ž}
– ϕ is defined as:
  - $(\forall\alpha \in (\Sigma \cup \{\lambda\}), \forall\beta \in M, \forall\gamma \in (\Sigma \cup \{\lambda\}))\ \phi(\alpha, \beta, \gamma) = 0$
  - $(\forall\alpha \in (\Sigma\ M), \forall\beta \in \{r, l, R, L\}, \forall\gamma \in ((\Sigma \cup \{\lambda\})\backslash M))\ \phi(\alpha, \beta, \gamma) = 0$
  - else $\phi(\alpha, \beta, \gamma) = 1$
  - where M = {a,á,e,é,ě,i,í,o,ó,u,ú,ů,y,ý,A,Á,E,É,Ě,I,Í,O,Ó,U,Ú,Ů,Y,Ý}
– $A = A_1$ from Note 4.

2.5 Algorithms of Decomposition Into Syllables

We describe four universal algorithms of decomposition into syllables: universal left $\mathrm{P_{UL}}$, universal right $\mathrm{P_{UR}}$, universal middle-left $\mathrm{P_{UML}}$, universal middle-right $\mathrm{P_{UMR}}$. These four algorithms are called as algorithms of class $\mathrm{P_U}$. The names of these algorithms are derived from way these algorithms work. Inputs of these algorithms are language $L = (\Sigma, \Sigma_P, \Sigma_C, \Sigma_M, \phi, A)$ and document $D = \alpha_1, \ldots, \alpha_n$, $\alpha_i \in \Sigma$. These algorithms are composed from two parts. The first part is an initialization and this is common for all algorithms of the class $\mathrm{P_U}$. The second part is different for each algorithm.

At the beginning of the initialize part we decompose document D into words by algorithm A. Algorithm of class $\mathrm{P_U}$ is processing single words. Words from non-letters are automatic declared as syllables. For each word from letters is according function ϕ decided whose its letters are consonants and which are vowels. Maximal blocks (blocks that cannot be extended) of vowels are found afterwards. Blocks of vowels longer than three are not usually in natural languages, so maximal length of block of vowels is set to three. These blocks of vowels will create bases of syllables. For each block of vowels we must keep in memory its begin and end. Consonants, which are in the word before first block of vowels, are added to this block. Consonants, which are in the word after last block of vowels, are added to this block.

Single algorithms of class $\mathrm{P_U}$ are different in the way of adding consonants, which are between two blocks of vowels. After these ways of adding are algorithms named.

Algorithm universal left $\mathrm{P_{UL}}$ adds all consonants between blocks of vowels to the left block.

Algorithm universal right $\mathrm{P_{UR}}$ adds all consonants between blocks of vowels to the right block.

Algorithm universal right $\mathrm{P_{UMR}}$ in the case of $2n$ (even count) consonants between blocks adds to both blocks n consonants. In the case of $2n + 1$ (odd

count) consonants between blocks it adds to the left block n consonants and to the right block $n+1$ consonants.

Algorithm universal right P_{UML} in case of $2n$ (even count) consonants between blocks adds to both blocks n consonants. In the case of $2n+1$ (odd count) consonants between blocks it adds to the left block $n+1$ consonants and to the right block n consonants. The only exception from this rule is the case when between blocks it is only one consonant, this consonants is added to the left block.

Example: We will decompose word *priesthood* into syllables. We are using language L_{EN}. Blocks of vowels are (in order): *ie*, *oo*.

correct decomposition into syllables:	priest-hood
universal left P_{UL}:	priesth-ood
universal right P_{UR}:	prie-sthood
universal middle-left P_{UML}:	priest-hood (correct form)
universal middle-right P_{UMR}:	pries-thood

3 Compression methods

We used hypothesis that compressed text is structured into sentences and it is described by following rules. A sentence begins with mixed word and ends with other word, which contains a dot. Inside the sentences are switching regularly small words and other words. If the sentence begins with capital-word, then inside are switching regularly capital-words and other-word. Numeric-words appear rarely and are usually followed by other-words.

When we decompose words into syllables, we have problem with this model. Each word has different count of syllables. Small-word is usually followed by other-word, whereas small-syllable can be followed not only by other-syllables but also by another small-syllable.

To improve a compression of alphabet of syllables (or words) we have created for each language a database of frequent words. More details will be in section 4. Words from this database are used for initialization of compressing algorithms. When coding alphabet of given document we can code only words, which are not from out databases of frequent words. This is useful for smaller documents, on the bigger documents is that effect lower.

3.1 LZWL

Algorithm LZW [6] is a dictionary compression character-based method. Syllable-based version of this method has been named LZWL. Algorithm LZWL can work with syllables obtained by all algorithms of decomposition into syllables. This algorithm can be used for words (see Def. 4) too.

First we shortly recall classical method LZW [6]. Algorithm is using dictionary of phrases, which is represented by data structure trie. Phrases are numbered by integers afterwards order of adding.

In initialization step the dictionary is filled up with all characters from alphabet. In each next step it is searched for maximal string S, which is from dictionary and matches the prefix of still non-coded part of the input. Number of phrase S is sent to the output. A new phrase is added to the dictionary. This phrase is created by concatenation of string S and character that follows after S in file. Actual input position is moved forward by the length of S.

Decoding has only one situation for solving. We can receive number of phrase, which is not from dictionary. In this case we can create that phrase by concatenation of the last added phrase with its first character.

Syllable-base version is working over alphabet of syllables. In initialization step we add to the dictionary empty syllable and small syllables from database of frequent syllables. Finding string S and coding its number is analogical with character-based version, only that string S is a string of syllables. Number of phase S is encoded to output. It is possible that string S can be empty syllable. If S is empty syllable, then we must get from file one syllable called K and encode K by methods for coding new syllables, see section 4.2. Syllable K is added to dictionary. Actual position in the file is moved forward by the length of S, in the case when S is empty syllable, the input position is moved forward by the length of K.

In adding a phrase to dictionary there is a difference to character-based version. Phrase from the next step will be called S_1. If S and S_1 are both non-empty syllables, then we add new phrase to the dictionary. New phrase is created by concatenation S_1 with the first syllable of S. This solution has two advantages. The first advantage is that strings are not created from syllables that appear only once. Second advantage is that we cannot receive in decoder number of phrase that is not from dictionary.

3.2 HuffSyllable

HuffSyllalbe is statistical compression method based on adaptive Huffman coding and using structure of sentence in natural language. The idea of this algorithm was inspired by HuffWord [7]. Algorithm LZWL can work with syllables obtained by all algorithms of decomposition into syllables mentioned above. This algorithm can be used for words too.

For each type of syllables (small, capital, mixed, number, other) it is build adaptive Huffman tree [2], which is coding syllables of given type. In the initialization step of algorithm we add to Huffman tree for small syllables all syllables and their frequencies from database of frequent syllables.

In each step of the algorithm it is calculated expected type of actually processed syllable K. If syllable K has different type than it is expected, then an escape sequence is generated. Syllable K is encoded by Huffman tree corresponding to the syllable type. Calculating of expected type of syllable uses information from encoded part of input. We need to know the type of last syllable. If the last syllable is other-syllable, then it is known that this syllable contains a dot and that the type of the last syllable is letter-syllable.

Table 1. Expected types of syllables according type of previous syllable.

previous / expected type of syllable	Expected syllable
small	small
capital	capital
mixed	small
number	other
other syllable without dot, last syllable from letters is not capital	small
other syllable with dot, last syllable from letters is not capital	mixed
other, last syllable from letters is capital	capital

4 Technical Details

We supposed that all natural languages have their own characteristical set of syllables. Its approximation for English and Czech was created by set of testing documents. We created for each language and algorithm of decomposition into syllables one database of frequent syllables from letters. For each language we also created databases of frequent other syllables. Condition for adding syllable to database was that its frequency is greater than 1 : 65,000. Each database of syllables from letters contains approximately 3,000 syllables, whose sum of frequency is 96–98 % of all syllables. Each database of other syllables contains approximately 100 syllables, which sum of frequency is 99.5 % of all syllables.

These databases are used in initializations steps of compressing algorithms. This can improve compression ratio on smaller documents.

Although we have database of frequent syllables, sometimes we receive syllable K, which is not from this database and we must encode it. They are two basic ways. The first way is to encode K as code of length of K followed by the codes of individual characters from the syllable. The second (and better) way is to encode K as code of syllable type followed by code of length of K and codes of individual characters. We use the second way, because domain of coding function for distinct characters is given by the type of syllable and as it is smaller than in the first way. Numeric syllables are coded differently.

Encoding type of syllable depends on types of previous syllable and other criteria as in HuffSyllable. Length of codes for each types are 1, 2, 3, and 4. Average code length is 1.5 bits.

For encoding length of syllables are used two static Huffman trees, the first one for letter-syllables and the second one for other-syllables. Trees are initialized from statistics received from text documents.

For encoding distinct characters there are used two adaptive Huffman trees, the first one for syllables from letters and the second one for other syllables.

Numeric-syllables are coded differently from other types of syllables. We discover that numbers in text are naturally divided into a few categories. The first category contains small numbers (1–100), the second category represent year (1800–2000), in the third category there are very large numbers (for example

5,236,964) that usually have separated groups of digits to blocks by three. But these large numbers are decomposed into numeric-words and other-words. So we set maximal length of numeric-word to 4, longer numeric-words are split. For coding number of digits 2-bits binary coding is used. For coding distinct digits binary phased coding [4] is used.

5 Experimental Results

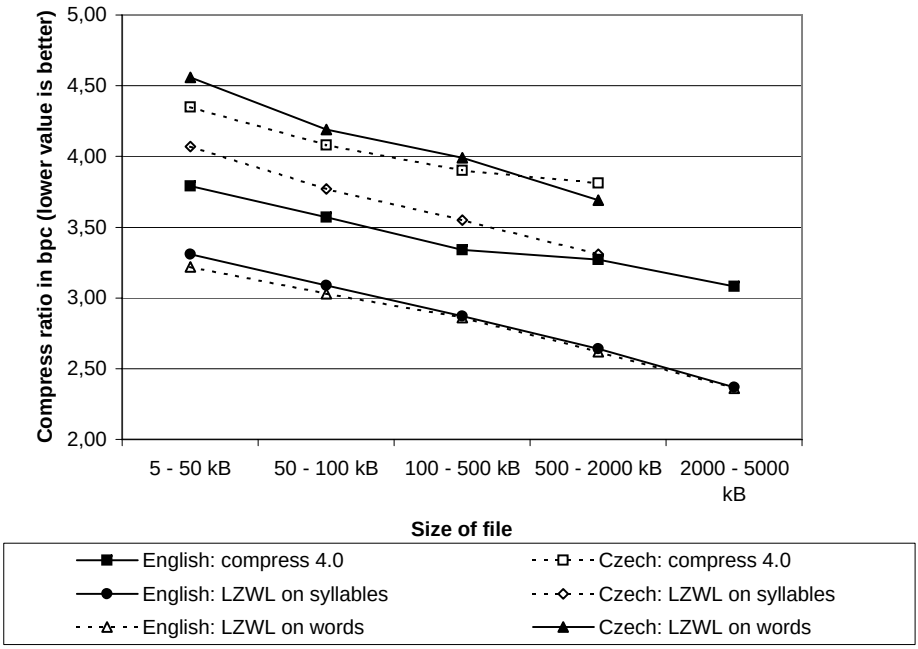


Fig. 1. Comparison of LZW-based methods on English and Czech

For testing there were used two sets of documents in plain text format. The first set contains 69 documents in Czech with total size of 15 MB. Most of these documents were received from [11]. The second set contains 334 documents in English with total size of 144 MB. In this set there are documents from project Gutenberg [12] and `bible.txt` from Canterbury corpus [9]. From each file form project Gutenberg there were removed first 12 kB of information about project because it was the same in all documents.

Table 2. Comparison of compression ratio in bits per character on English documents

Method\file	5–50 kB	50–100 kB	100–500 kB	500–2000 kB	2000–5000 kB
LZWL+P_{UL}	3.31	3.09	2.87	2.64	2.37
LZWL+P_{UR}	3.36	3.14	2.92	2.69	2.39
LZWL+P_{UML}	3.32	3.10	2.88	2.65	2.38
LZWL+P_{UMR}	3.32	3.10	2.89	2.66	2.38
LZWL(words)	3.22	3.03	2.86	2.62	2.36
compress 4.0	3.79	3.57	3.34	3.27	3.08
HS+P_{UL}	3.23	3.18	3.15	3.10	2.97
HS+P_{UR}	3.30	3.26	3.22	3.18	3.03
HS+P_{UML}	3.26	3.22	3.19	3.15	3.02
HS+P_{UMR}	3.27	3.23	3.20	3.16	3.02
HS(words)	2.65	2.58	2.52	2.38	2.31
ACM(words) [3]	2.93	2.74	2.55	2.35	2.27
FGK [2]	4.59	4.60	4.60	4.58	4.54
bzip2 [8]	2.86	2.60	2.40	2.21	2.03

Table 3. Comparison of compression ratio in bytes per character on Czech documents

Method\file	5–50 kB	50–100 kB	100–500 kB	500–2000 kB	2000–5000 kB
LZWL+P_{UL}	4.14	3.83	3.59	3.34	—
LZWL+P_{UR}	4.07	3.77	3.56	3.32	—
LZWL+P_{UML}	4.07	3.77	3.56	3.31	—
LZWL+P_{UMR}	4.07	3.77	3.55	3.31	—
LZWL(words)	4.56	4.19	3.99	3.69	—
compress 4.0	4.35	4.08	3.90	3.81	—
HS+P_{UL}	3.97	3.89	3.89	3.81	—
HS+P_{UR}	3.86	3.79	3.80	3.75	—
HS+P_{UML}	3.86	3.79	3.80	3.74	—
HS+P_{UMR}	3.87	3.79	3.80	3.75	—
HS(words)	3.71	3.51	3.43	3.21	—
ACM(words)	3.83	3.50	3.29	3.14	—
FGK	4.97	4.95	5.00	4.99	—
bzip2	3.42	3.10	2.88	2.67	—

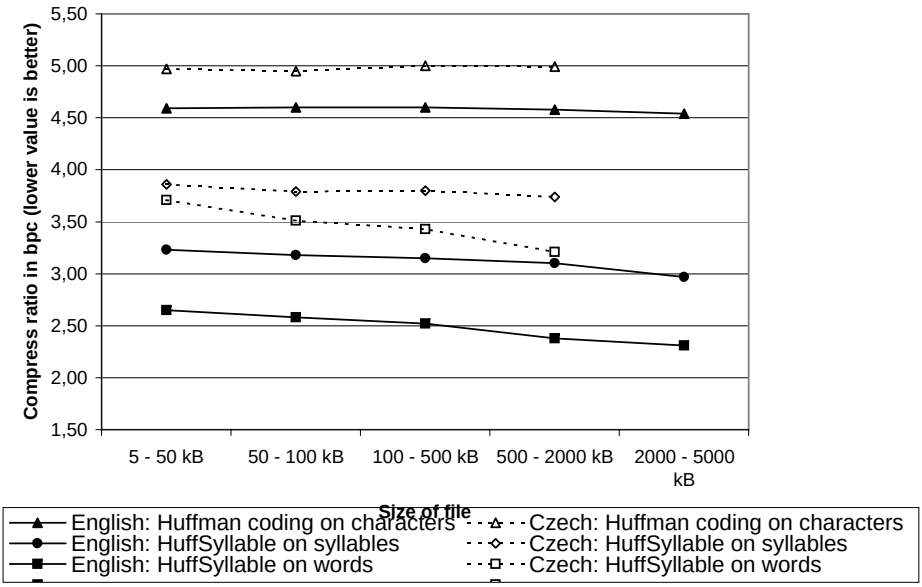


Fig. 2. Comparison of Huffman-based methods on English and Czech texts

6 Conclusion

In this paper we have introduced idea of syllable-based compression, its advantages and disadvantages. We have formally defined letters, syllable, words and algorithm of decomposition into words. We have introduced four universal algorithms of decomposition into words. We have created two syllable-based compression methods that use alternation of syllable types in sentences. The first method is based on algorithm LZW, the second on Huffman coding. The experimental results of these algorithms confirm our predictions, that tested syllable-based algorithms outperformed their character-based counterparts for both tested languages. Comparison of word-based and syllable-based versions of Huffman and LZW codings led to the result that in English the word-based versions of both algorithms outperform their syllable-based counterparts and in Czech the results are ambiguous: for Huffman coding word-beased version outperformed syllable-based one, for LZW coding the syllable-based one outperformed the word-based one.

In the future we want to decrease space and time requirements of implemented algorithms. We planned to addapt next syllable-based algorithms from their character-based versions, for example bzip2. We want to test our algorithms on more languages with rich morphology, for example on German and Hungarian.

References

1. D. A. Huffman. A method for the construction of minimum redundancy codes. *Proc. Inst. Radio Eng.*, 40:1098–1101, 1952.
2. D. E. Knuth. Dynamic Huffman coding. *J. of Algorithms*, 6:163–180, 1985.
3. A. Moffat, R. M. Neal, and I. H. Witten. Arithmetic coding revisited. *ACM Transactions on Information Systems*, 16:256–294, 1998.
4. P. Elias. Universal codeword sets and representation of the integers. *IEEE Trans. on Information Theory*, 21(2):194–203, 1975.
5. S. W. Thomas, J. McKie, S. Davies, K. Turkowski, J. A. Woods, and J. W. Orost. Compress (version 4.0) program and documentation, 1985.
6. T. A. Welch. A technique for high performance data compression. *IEEE Computer*, 17(6):8–19, 1984.
7. I. Witten, A. Moffat, and T. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Van Nostrand Reinhold, 1994.
8. The bzip2 and libbzip2 official home page. http://sources.redhat.com/bzip2/ as visited on 6th February 2005.
9. Canterbury corpus. http://corpus.canterbury.ac.nz.
10. Compact Oxford English Dictionary. http://www.askoxford.com/ as visited on 3rd February 2005.
11. eknihy. http://go.to/eknihy as visited on 2nd February 2005.
12. Project Gutenberg. http://www.promo.net/pg.

Vector model improvement by FCA and Topic Evolution

Jan Martinovič and Petr Gajdoš

Department of Computer Science, VŠB – Technical University of Ostrava
17. listopadu 15, 708 33 Ostrava-Poruba, Czech Republic
{Petr.Gajdos, Jan.Martinovic}@vsb.cz

Abstract. Presented research is based on standard methods of information retrieval using the vector model for representation of documents (objects). The vector model is often expanded to get better precision and recall. In this article we have mentioned two approaches of vector model expansion. The first approach is based on hierarchical clustering. Its goal is to find a list of all documents they have most similar topic to the requested document. The second one is the document classification based on formal concept analysis. We have tried to evaluate all concepts and computed the importances of documents. At last have compared the results of our approach based on formal concept analysis and the results of classical vector model.

Keywords: Vector, FCA, Moebius, Topic Evolution, Clustering

1 Introduction

There are various systems for searching in document collections. They are based on vectors, probabilistic and another models for representation of documents, queries, rules and procedures. Each of the models contains a rank of limitations. Therefore we usually don't obtain all relevant documents. In our research we have to mentioned two approaches of vector model expansion. The first approach is based on hierarchical clustering. Its goal is to find a list of all documents they have most similar topics to the requested document. The second one is the document classification based on formal concept analysis. In following chapter, we have described classic vector model, cluster analysis and basic definition from formal concept analysis, which we needed for next computation. Then we have described our approaches for vector model improvement. In the fourth chapter, we have demonstrated benefits of our approach.

2 Background

2.1 Vector model

The vector model [12] of documents is dated back to 70th of the 20th century. In vector model there are documents and users queries represented by vectors.

We use m different terms $t_1 \dots t_m$ for indexing n documents. Then each document d_i is represented by a vector:

$$d_i = (w_{i1}, w_{i2}, \dots, w_{im}),$$

where w_{ij} is the weight of the term t_j in the document d_i.
An index file of the vector model is represented by matrix:

$$D = \begin{pmatrix} w_{11}\, w_{12} \dots w_{1m} \\ w_{21}\, w_{22} \dots w_{2m} \\ \vdots \quad \vdots \quad \ddots \quad \vdots \\ w_{n1} w_{n2} \dots w_{nm} \end{pmatrix},$$

where i-th row matches i-th document, and j-th column matches j-th term.
In a vector model a query is represented by m dimensional vector:

$$q = (q_1, q_2, \dots, q_m),$$

where $q_j \in \langle 0, 1 \rangle^m$. On the basis of the query q we can compute *coefficient of similarity* for each document d_i. This coefficient can be understood as "distance" between the document's vector and the vector of the query. We used cosine measure for computing this similarity:

$$sim(q, d_i) = \frac{\sum_{k=1}^{m} (q_k w_{ik})}{\sqrt{\sum_{k=1}^{m} (q_k)^2 \sum_{k=1}^{m} (w_{ik})^2}}$$

The similarity of two documents is given by following formula:

$$sim(d_i, d_j) = \frac{\sum_{k=1}^{m} (w_{ik} w_{jk})}{\sqrt{\sum_{k=1}^{m} (w_{ik})^2 \sum_{k=1}^{m} (w_{jk})^2}}$$

For more information see [12].

2.2 Cluster analysis

The main goal of the cluster analysis is to find the fact, if there are any groups of similar objects. These groups are called *clusters*. We focuse on object clustering that can be divided in two steps. Firstly, we create the clusters and then we look for relevant clusters [7]. The reason of the cluster analysis is contained in the clusters hypothesis [12].

The searching process of an ideal fragmentation of objects is also called clustering. We use an agglomerative hierarchical clustering based on the similarity matrix:

At the beginning each object is considered as one cluster. Clusters are joined together in sequence. The algorithm is over, when all objects form only one cluster.

Similarity matrix Sim_C for collections C may be described with:

$$Sim_C = \begin{pmatrix} sim_{11}\, sim_{12} \ldots sim_{1n} \\ sim_{21}\, sim_{22} \ldots sim_{2n} \\ \vdots \quad \vdots \quad \ddots \quad \vdots \\ sim_{n1} sim_{n2} \ldots sim_{nn} \end{pmatrix},$$

where i-th row matches i-th document and j-th column j-th document.

2.3 Formal Concept Analysis

FCA has been defined by R. Wille and it can be used for hierarchical order of objects based on object's features. The basic terms are formal context and formal concept. In this section there are all important definitions the one needs to know to understand the problematics.

Definition 1. *A formal context C = (G,M,I) consists of two sets G and M and a relation I between G and M. Elements of G are called objects and elements of M are called attributes of the context. In order to express that an object g is in a relation I with an attribute m, we write gIm or and read it as "the object g has the attribute m". The relation I is also called the incidence relation of the context.*

Definition 2. *For a set $A \subset G$ of objects we define*

$$A^{\uparrow} = \{m \in M \mid gIm \ for\ all\ g \in A\} \tag{1}$$

-the set of attributes common to the objects in A. Correspondingly, for a set $B \subset M$ of attributes we define

$$B^{\downarrow} = \{g \in G \mid gIm \ for\ all\ m \in B\} \tag{2}$$

-the set of objects which have all attributes in B.

Definition 3. *A formal concept of the context (G,M,I) is a pair (A, B) with $A \subseteq G$, $B \subseteq M$, $A^{\uparrow} = B$ and $B^{\downarrow} = A$. We call A the extent and B the intent of the concept (A, B).*

Definition 4. *Let M be the totality of all features deemed relevant in the specific context, and let $I \subseteq G \times M$ be the incidence relation that describes the features possessed by objects, i.e. $(g, m) \in I$ whenever object $g \in G$ possesses a feature $m \in M$. For each relevant feature $m \in M$, let $\lambda(m) \geq 0$ quantify the importance or weight of feature m. The diversity value of a set S is defined as*

$$v(S) = \sum_{m \in M:(g,m) \in I \ for\ some\ g \in S} \lambda(m) \tag{3}$$

Our approach is also based on Conjugate Moebius Function and the on some properties go out from the Theory of diversity and Formal concept analysis.

Theorem 1. *For any function* $v : 2^M \to \mathbb{R}$ *with* $v(\emptyset) = 0$ *there exists unique function* $\lambda : 2^M \to R$*, the Conjugate Moebius Inverse function, such that* $\lambda(\emptyset) = 0$ *and for all* S*,*

$$v(S) = \sum_{A: A \cap S \neq \emptyset} \lambda(A) \tag{4}$$

Furthermore, the Conjugate Moebius Inverse λ *is given by the following formula. For all* $A \neq \emptyset$*,*

$$\lambda(A) = \sum_{A: A \cap S \neq \emptyset} (-1)^{|A|-|S|+1} * v(\overline{S}) \tag{5}$$

where $\overline{S}$ *denotes the complement of* S *in* M*.*

The diversity of an object (document) g is the sum of all weights of all features which are related to the object according to the incidence matrix. It conveys information about partial importance of an object but doesn't clearly display other dependences.

$$do(g) = \sum_{m: m \in M \text{ and } (gIm) \in I} \lambda(m) \tag{6}$$

Next characteristic is called the sum of diversities of all objects. Actually, the objects of one concept can "cover" all features.

$$sdo(C) = \sum_{g: g \in C} do(g) \tag{7}$$

The importance of the object (document) g is the main point of our method. The value represents the importance from these aspects:

- Uniqueness - Is there any other similar object?
- Range of description - What type of dimension does the object describe?
- Weight of description - What is the weight of object in each dimension?

$$impo(g) = \sum_{C: C \ni g} \frac{sdo(C)}{v(S)} \lambda(A)\ do(g) \tag{8}$$

For more information see [3]

3 Vector Model Improvement

3.1 Using FCA to obtain the importance of documents

This method is based a) on the partial ordering of concepts in the concept lattice and b) on the inverse calculation of weights of objects using Moebius function and defined characteristics. Particular steps are illustrated by fig.[1] and briefly described in this chapter.

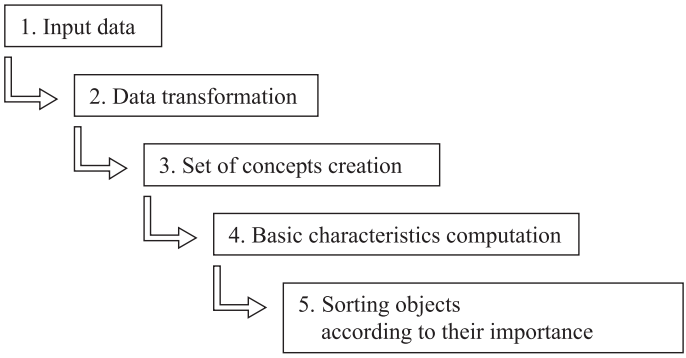


Fig. 1. Getting importances of objects (documents)

First we obtain the input data (documents and words) like a table or matrix. The second step - scaling method is used to create an input incidence matrix. Every dimension can be scaled to a finite number of parts to get the binary values or we can only change non-zero values for number one, otherwise number zero. The output of transformation is an incidence matrix that we need as input for the concept calculation. Next the power set of concepts is computed using FCA algorithms. We can create the "concept lattice" and draw the Hasse diagram, but it's not important in our method. But it can be useful to show dependences between concepts, if we need it. We use only the list of concepts. After that, we can compute the basic characteristics for each concept according to the formulas (4), (5), (6), (7). Finally, we compute the importance of objects according to the formula (8). Obtained values provide us the criteria to sort the set of objects.

3.2 Evolution of topic

Our research concerns with the topics undergo an evolution. Lets assume document from collection of documents, that describes the same topic. It is clear, there are some other documents in the collection that describes the same topic, but they use different words to characterize the topic. The difference can be caused by many reasons. The first document focused on the topic use some set of words and next documents may use synonyms or for example exploration of new circumstances, new fact, new political situation etc. [4].

The result of searching an evolution of topic is to engaged query finding the lists of documents related by thematic with engaged query. We mean the query as query sets by terms or as document which is set as relevant.

We define this algorithm based on formal concept analysis and another algorithm for clustering. Our research gives us the answer for the question "What is the better way to improve the results of vector model?"

This is our algorithm using FCA and Moebius function:

Algorithm TOPIC-FCA:

1. We make the query transformation. It means that we create weighted vector of terms.
2. We compute the importances of documents (objects) by the formula 8. and we make the list of the documents and their importances.
3. We find the relevant document rel_d in the ordered list.
4. In finite steps, we look for "nearest" documents. The "nearest" document is the document, that has the smallest difference between its weight and the weight of rel_d. Founded document is excluded before next step.

Then we use this algorithm for clustering:

Algorithm TOPIC-CA:

1. We choose the total number of documents we want ('level').
2. We find leaf cluster which contain selected relevant document.
3. We get up in hierarchy.
4. We explore neighbouring clusters. First we select the cluster created on the highest sub-level. Each document, which we find, we add to the result list. When the count of all documents in the result list equal to 'level' we break finding.
5. We repeat the step 3.

3.3 Sort Response in Vector Model

The collection of documents responses to the query in the vector model, which is ordered by the coefficient of similarity of the query and the document. In this part, we present the method that can change this response by asking to the evolution of topic from clusters or concepts. Our approach is based on removing all non-relevant documents from the query and next on adding another relevant documents to the query. We have developed next algorithm for this change:

Algorithm SORT-EACH, this algorithm moves all documents in a result of the vector model query so that the documents belong to the same evolution of topic are closer to each other:

1. Collection of the documents from the vector query is marked as C_V.
2. The new sorted collection is marked as C_S and the count of its documents is a new value of the variable *count*.
3. We choose the total number of documents in evolution of topic and we mark it as *level*.
4. We do next sorting:

foreach document D_V in C_V **do**
  if C_S is empty **then**
    add D_V to collection C_S
    goto Continue

end
To document D_V found by algorithm TOPIC-FCA (or TOPIC-CA) collection of evolution of topic C_T. Count of documents in topic is $level + 1$ (document D_V).
foreach document D_T in C_T within document D_V **do**
if document D_T is in C_S **then**
add the document D_V behind D_T do C_S
goto Continue
end
end
if not added D_V **then**
add D_V to end of collection C_S
label: Continue
end

5. We return collection C_S to user.

4 Illustrative sample of vector model improvement

Following tables show experimental results on generated data. Documents' importances were computed according to formula 8. The document selected by user is highlighted. This is the input document in TOPIC algorithms above. Each query is transformed to the vector of weights of terms. We use simplified matrix of documents and terms. The number "1" means that the document on the row consists the term in given column.

In the tables, we can see, that a vector queries give us worse results in some occurrence because they return zero-values of documents that don't have common terms. But, these documents can be about the same theme described by different terms (words). So we use the SEARCH-EACH algorithm for improve vector query by TOPIC-FCA or TOPIC-CA. We use the new TOPIC-FCA algorithm in these samples. See [4] to get another experiments.

Table 1. The results after inserted query "111111111111"

query	1 1 1 1 1 1 1 1 1 1 1 1	Document's importance	Vector query
	t_1 t_2 t_3 t_4 t_5 t_4 t_7 t_8 t_9 t_{10} t_{11} t_{12}		
doc. 1	1 1 1	36	0.5
doc. 2	**1 1 1**	**36**	**0.5**
doc. 3	1 1 1	36	0.5
doc. 4	1 1 1	36	0.5

Table 1 is very simple. We enter the query "111111111111". It means that we are looking for all relevant documents which contain all possible words. A vector query return all documents with the same relevancy because each of

them contains three requested terms. Computed TOPIC-FCA (Importances of objects) brings zero improvement.

Table 2. The results after inserted query "111111111111"

query	1	1	1	1	1	1	1	1	1	1	1	1		
	t_1	t_2	t_3	t_4	t_5	t_4	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	Document's importance	Vector query
doc. 1	1	1	1		1								66.66666667	0.57735
doc. 2				**1**	**1**	**1**							**38**	**0.5**
doc. 3							1	1	1				36	0.5
doc. 4										1	1	1	36	0.5

Next, we describe table 2. The values of documents' importances show us the relative importances according to inserted query. There are only small differences between the importances of objects and vector query. The distance between document number 1 and selected document number 2 is larger then the distance between document number 2 and 3 (see the difference between documents' importances). The distance of the vector query and each document plus the distance between documents are the main reason of this appearance. It is better to describe this effect in the following table 3.

Table 3. The results after inserted query "000111000000"

query	0	0	0	1	1	1	0	0	0	0	0	0		
	t_1	t_2	t_3	t_4	t_5	t_4	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	Document's importance	Vector query
doc. 1	1	1	1	1	1	1	1	1	1	1	1	1	295	0.5
doc. 2				**1**	**1**	**1**							**37.33333**	**1**
doc. 3							1	1	1	1			54.4	0.288675
doc. 4										1	1	1	10.8	0

The vector query is "000111000000". We selected the document number 2 again. Although the first document contains the same terms as the second document, the distance between them is very large because of great number of terms the second document does not contain. Then, the evolution of topic of the second document is doc3, doc4 and at last doc1. So we get different ordering than the ordering after vector query.

The table 4 shows the main deficiency of the vector query. When we insert query "000111000000" we can not obtain the fourth document. But our method include this document because of a similarity to selected document number 2. So we can find new dependences between documents they can be about the same theme.

Table 4. The results after inserted query "000111000000"

query	0	0	0	1	1	1	0	0	0	0	0	0		
	t_1	t_2	t_3	t_4	t_5	t_4	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	Document's importance	Vector query
doc. 1				1	1			1	1	1	1	1	94.93333333	0.436436
doc. 2				**1**	**1**	**1**			**1**				**53.2**	**0.866025**
doc. 3						1	1	1	1				47	0.288675
doc. 4									1	1	1	1	26	0

Table 5. The results after inserted query "000111000000"

query	0	0	0	1	1	1	0	0	0	0	0	0		
	t_1	t_2	t_3	t_4	t_5	t_4	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	Document's importance	Vector query
doc. 1								1	1	1	1	1	41.86111111	0
doc. 2				**1**	**1**	**1**			**1**				**44.5**	**0.866025**
doc. 3						1	1	1	1				45.83333333	0.288675
doc. 4									1	1	1	1	28.6	0

The last table 5 shows better all hidden dependences between documents. The documents number 1 and 4 are not included in vector query, but we can say there can be some references between them because of common term number 9. The evolution of topic of selected document is doc3, doc1 and doc4.

We tried to show the importance of our method in simple examples. If we use the TOPIC-FCA or TOPIC-CA for vector query improvement we can find another dependences between documents and we can get better ordering of requested documents.

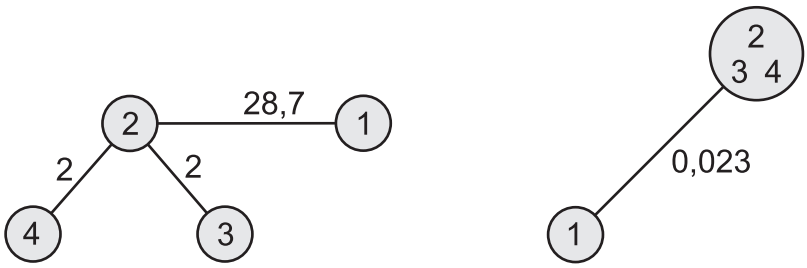
4.1 Sample graphs

Following graphs show documents' distances from selected document number two. The graphs on the left show distances of documents after using TOPIC-FCA algorithm and the graphs on the right correspond to the results of the vector query. All distances were computed from selected document number 2.

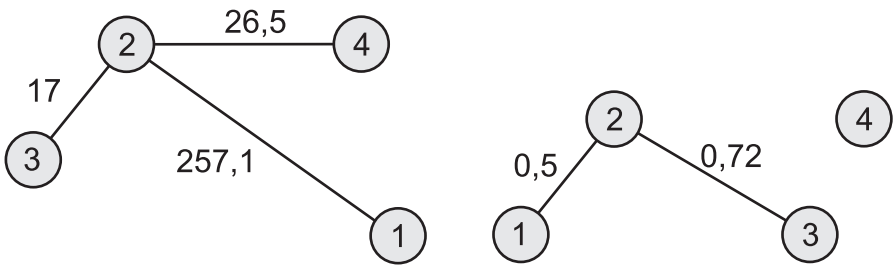
Graph description:

- Node represent a document or a cluster of document if the documents' distance is zero. Node's numbers correspond to number of documents.
- Edge connect comparable documents (nodes). The value means the distance of appropriate documents.

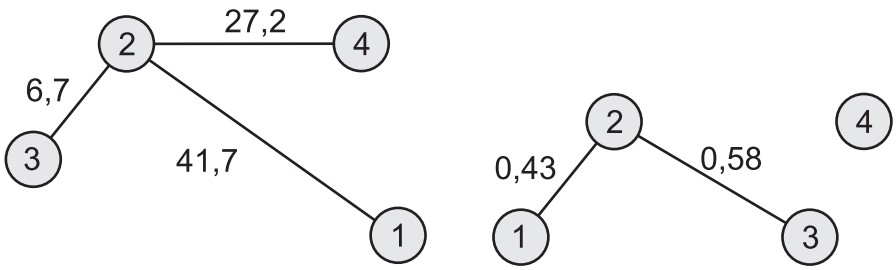
Documents' distances computed from table 1.



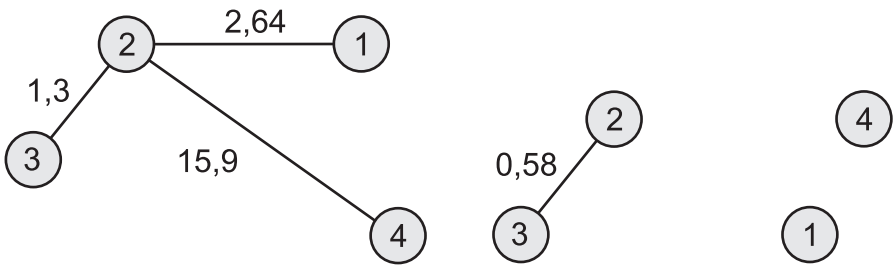
Documents' distances computed from table 2.



Documents' distances computed from table 3.



Documents' distances computed from table 4.



Documents' distances computed from table 5.

5 Conclusion and future work

We have described new method for vector query improvement based on formal concept analysis and Moebius inverse function. The known deficiencies of vector model have been suppressed using TOPICs and SEARCH-EACH algorithms. In the future work we would like to test our methods on real data. Our presented methods can be applied on small data sets or on large collections of documents.

References

1. Berry, M. W (Ed.): Survey of Text Mining: Clustering Classification, and Retrieval. Springer Verlag 2003.
2. Baeza-Yates R., Ribeiro-Neto B.: Modern Information Retrieval. Addison Wesley, New York, 1999.
3. Ďuráková, D., Gajdoš, P.: Indicators Valuation using FCA and Moebius Inversion Function. DATAKON, Brno, 2004, IBSN 80-210-3516-1

4. Dvorský J., Martinovič J., Snášel V.: Query Expansion and Evolution of Topic in Information Retrieval Systems, DATESO 2004, ISBN: 80-248-0457-3.
5. Dvorský J., Martinovič J., Pokorný J., Snášel V.: A Search topics in Collection of Documents.(in Czech),Znalosti 2004, ISBN: 80-248-0456-5.
6. Ganter B., Wille R.: Formal Concept Analysis. Springer-Verlag, Berlin, Heidelberg, 1999.
7. Christis Faloutsos, Douglas Oard: A Survey of Information Retrieval and Filtering Methods, Univ. of Maryland Institute for Advanced Computer Studies Report, College Park, 1995.
8. Keith Van Rijsbergen: The Geometry of Information Retrieva, Cambridge University Press, 2004.
9. Kummamuru K, Lotlikar R., Roy S., Singal K., Krishnapuram R.: A Hierarchical Monothetic Document Clustering Algorithm for Summarization and Browsing Search Results, WWW2004, New York, USA.
10. Nehring, K. and Puppe, C.: Modelling phylogenetic diversity. Resource and Energy Economics (2002).
11. Nehring, K.: A Theory of Diversity. Ecometrica 70 (2002) 1155 1198.
12. Pokorný J., Snášel V., Húsek D.: Dokumentografické informační systémy. Karolinum, Skriptum MFF UK Praha, 1998.
13. C.J. van Rijsbergen: Information Retrieval (second ed.). London, Butterworths, 1979.
14. Tsunenori Ishioka: Evaluation of Criteria for Information Retrieval, International Conference on Web Intelligence, IEEE Computer Society, 2003, ISBN 0-7695-1932-6.
15. S. Vempala: The Random Projection Method, Dimacs Series in Discrete Mathematics and Theoretical Computer Science, 2004.

Unsupervised clustering with growing self-organizing neural network – a comparison with non-neural approach

Martin Hynar, Michal Burda, and Jana Šarmanová

Department of Computer Science, VŠB – Technical University of Ostrava
17. listopadu 15, 708 33 Ostrava-Poruba, Czech Republic
{martin.hynar, michal.burda, jana.sarmanova}@vsb.cz

Abstract. Usually used approaches for non-hierarchical clustering of data are well known k-means or k-medoids methods. However, these fundamental methods are poorly applicable in situations where number of clusters is almost unpredictable. Formerly, they were adapted to allow splitting and merging when some defined criterion is met. On the other hand there are also methods based on artificial neural networks concretely on self-organizing maps. One of the interesting ideas in this domain is to allow growing of the net which corresponds to adapted k-means method. In this article we are going to compare both approaches in a view of ability to detect clusters in unknown data.

Key words: data mining, cluster analysis, partitioning algorithms, competitive learning, self-organizing map, growing neural network.

1 Introduction

In common practice of various domains, e.g. pattern recognition (recognition of objects in range of data, letters), information retrieval (grouping documents, looking for common topics), data mining (searching for interesting patterns in arbitrary data sets) there could be used as a valuable tool some clustering technique. The most widely used techniques are mainly *k-means* based methods (see [5], [3], [1]), which are easy to use and obtained results are fairly understandable. Nevertheless, these methods are too static in a particular point of view; at the beginning we have to specify the number of expected clusters and the algorithm is then responsible to find them in the data set. But, what if we could not determine this number? There are two alternatives to solve such problem.

- Firstly, we can do multiple computations with varying number of expected clusters. Such approach is admissible only in situations where input data set is not to extensive; on large data it will be very time consuming. Moreover, the decision what partitioning is appropriate is then based on a subjective estimation.

- Second solution is in adaptation of the algorithm where it will be allowed to split and/or merge clusters according to some predefined condition. Usually, clusters are splitted if the inter-cluster variability increases over some threshold and merged if the typical points of neighbouring clusters are near enough. We can also ignore insufficiently numerous clusters.

The second solution was used for example in algorithms like ISODATA (see [7], [4], [5]) or CLASS (see [7].

On the other hand, we can use a model of artificial neural network based on competitive learning known as the self-organizing map (SOM) or Kohonen map (see [6] or [8]). It is known that fundamental feature of SOM is to preserve data topology. So, neurons of the map are likely to occupy the place in the input space where more dense places are situated. The basic model of SOM consists of neurons whose quantity is specified in advance. That is, with this model we are able to discover only a predefined number of clusters.

Like in previous case also SOM was adapted to allow the topology grow. One of the usable approaches is the *Growing neural gas* (see [2])

In this article we first re-introduce the *k-means* algorithm in section 2 and one of its derivative (CLASS method) allowing adaptation of the number of clusters in section 3. In the section 4 we focus on fundamentals of SOM and on brief introduction of *Growing neural gas* algorithm in section 5. In the 6 there are provided some experiments with the comparison of the results obtained with both types of methods.

2 *k*-means based methods

So called *partitional* clustering methods could be stated as "given N patterns in n-dimensional space, determine the partition of the patterns into k groups where patterns in the same group are more similar than patterns from different groups" The notion of the patterns similarity have to be adopted in advance and different algorithms use different ones.

Moreover, the issue of determining the appropriate k is not decidable in all situations. If there exist some general perspective over clustered data, we can use a fixed value of k. The task of a partitional algorithm is then to locate clusters in the input space. If we are not able to determine k at the beginning, we could use trial-and-error way with modifying clustering parameters. However, there are also methods trying to locate clusters in the input space and to determine the number of such clusters at a time.

An algorithm, generally known as k-means clustering is the one where the number of expected clusters have to be given as the clustering parameter. Such algorithm given the set of patterns then tries to locate k clusters.

Choose typical points:

Place k typical points of clusters according to chosen method into the multidimensional space containing examined patterns.

Clustering:
Assign each pattern to exactly one typical point – to the nearest one.

Recompute typical points:
Using all patterns assigned to particular cluster recompute its typical point as the mean value.

Check termination condition:
The computation ends if the termination condition is fulfilled. Typical condition is that there are no or minimal changes in cluster memberships.

The problem of initial placing of k typical points could be solved using several techniques. The simplest ones are choosing random points or randomly chooses k input points.

Each pattern of the examined data set is then assigned to some typical point. The appropriate one is determined as the one with the smallest Euclidean distance.

$$d_E(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

When all patterns are processed the new typical points should be computed. New typical point of a cluster is determined as a mean vector of patterns in the group. The end of the algorithm usually becomes when no change in cluster membership occurs in two subsequent iterations.

3 CLASS – where clusters may arise and disappear

A clustering method CLASS is inspired in former method ISODATA which is in comparison with k-means method able to refine the number of clusters during the computation but it has some crucial disadvantages, mainly that many parameters need to be set by user. Such approach usually leads to incorrect results if data are not understood properly. On the other hand, the CLASS method in most cases determines the parameters from the input data.

The CLASS method proceeds in few steps. At the beginning, user has to set three input parameters: maximum number of iterations GAMMA, minimum number of members in a cluster THETAN and the initial splitting threshold S_0. For the testing purposes we used a modified version of CLASS method where it is also possible to set the initial number of typical points K and choose arbitrary method for placing them in the input space.

The very first step in the clustering is assigning each pattern to exactly one of the typical points. This proceeds using the k-means algorithm described above. After this the CLASS method goes through three following steps:

Excluding small clusters:
All clusters with less than THETAN members which were not changed during last two iterations are excluded from the subsequent analysis.

Splitting clusters:

In the m^{th} iteration we have to compute the splitting threshold S_m

$$S_m = S_{m-1} + \frac{1 - S_0}{GAMA}$$

In each cluster we need to compute deviations (d_{ij}) between the typical point and each pattern belonging to this typical point. Then, for the j^{th} attribute we can compute the average deviations D_{j1} of patterns situated on the right and D_{j2} of the patterns situated on the left side.

$$D_{j1} = \frac{1}{k_1} \sum_{i=1}^{k_1} d_{ij}, \qquad D_{j2} = \frac{1}{k_2} \sum_{i=1}^{k_2} d_{ij},$$

where k_1 and k_2 are the numbers of points situated on the right and on the left side of the typical point within given attribute. Now we are ready to determine parameters controlling the partitioning of the clusters. For each cluster we compute parameters a_1 and a_2 for patterns on the right side and left side.

$$a_1 = \max_j \left(\frac{D_{j1}}{max\, x_{ij}} \right), \qquad a_2 = \max_j \left(\frac{D_{j2}}{max\, x_{ij}} \right)$$

where $j = 1, 2, \ldots, p$ and i denotes all points from the same cluster. If in m^{th} iteration holds: number of clusters is less than $2K$, $a_1 > S_m$ or $a_2 > S_m$ and number of processed patterns greater than $2(THETAN + 1)$ then we split the cluster with respect to attribute j where a_1 or a_2 is maximal. The newly created clusters contain patterns from the right and left side of the typical point within the j^{th} attribute.

Revoking clusters:

To decide which cluster has to be revoked we need to compute average minimum distance of clusters in advance.

$$TAU = \frac{1}{h} \sum_{i=1}^{h} D_i$$

where h is current number of clusters and D_i is the minimum distance of i^{th} typical point from the other ones. If $D_i < TAU$ for some cluster i and number of clusters is greater than $\frac{K}{2}$ we revoke i^{th} cluster. Patterns belonging to revoked cluster are dispersed to the nearest typical points.

These steps proceed until clusters remaining unchanged or maximum number of iterations GAMA is reached.

4 Self-organizing map expresses the topology

The self-organizing map (SOM) is an artificial neural network based on an issue of competitive learning. The net consists of a set $\mathcal{A}$ with n neurons, represented

with weight vectors $\boldsymbol{w}_i$. Furthermore, neurons are mutually interconnected and these bindings form some topological grid (usually rectangular or triangular). If we present a pattern $\boldsymbol{x}$ into this network then exactly one neuron could be the *winner* and its weights are adapted proportionally to the pattern (the neuron is then closer). Moreover, neurons from the neighbourhood of the winner are adapted too, but not so intensively. Neighbourhood $N(c)$ could be formally defined as set of neurons that are topologically near to the winner.

The winner of the competition is determined as the neuron with the minimum distance to the pattern.

$$c = arg \min_{a \in \mathcal{A}} \{||\boldsymbol{x} - \boldsymbol{w}_a||\} \tag{1}$$

Then, adaptation of the weights proceeds using the equation (2) generally known as *Kohonen's rule*.

$$w_{ji}(t+1) = \begin{cases} w_{ji}(t) + h_{cj}(t)(x_i(t) - w_{ji}(t)) & j \in N(c) \\ w_{ji}(t) & \text{otherwise.} \end{cases} \tag{2}$$

Weight vectors for the next iteration $t+1$ of the winner and neurons in the neighbourhood are adapted in a way that current weights are modified (either added or subtracted) with a variance of current weight and input pattern.

Parameter $h_{cj}(t)$ is usually represented with unimodal Gauss function with center in c, width $\sigma(t)$ and maximal unit movement $h_0(t)$. Values of $\sigma(t)$ and $h_0(t)$ are decreasing in time – this corresponds with rough learning in the beginning and fine learning later.

$$h_{cj}(t) = h_0(t)\, exp\Big(- \frac{||\boldsymbol{w}_c - \boldsymbol{w}_j||^2}{2\sigma^2(t)} \Big)$$

One of the features of SOM is its topology preserving behaviour. This means, that SOM tries to adapt weights of neurons to cover the most dense regions and therefore SOM naturally finds data clusters. The limitation of SOM lies in fact that it is designed to have number of neurons specified as the input parameter and immutable during the learning process.

5 A self-organizing map that grows

The *Growing Neural Gas* (GNG) method [2] is the modification of the SOM where number of neurons is not immutable input parameter but is changed during the competition. Connections between neurons are not permanent as well. The result of competition could be then set of separate neural networks covering some region of the input data.

In the beginning the network itself contains only two neurons a_1 and a_2 representing two randomly chosen input patterns. Denote set of neurons as A and set of connections as $\mathcal{C}$ which is empty in the beginning.

Competition

The pattern $\boldsymbol{x}$ is presented to the network. The winner s_1 of competition and the second nearest s_2 neurons are determined using equation (1). If there was not a connection between neurons s_1 and s_2 then it is created ($\mathcal{C} = \mathcal{C} \cup \{(s_1, s_2)\}$). The *age* of the connection is set or updated to 0 ($age(s_1, s_2) = 0$). The squared distance between the winner and the pattern is added to local error variable.

$$\Delta E_{s_1} = ||\boldsymbol{x} - \boldsymbol{w}_{s_1}||^2$$

Adaptation

The weight vectors of the winner and its direct topological neighbours[1] N_{s_1} are adapted by fractions ϵ_b and ϵ_n of the distance to the input pattern. This is analogous to the *Kohonen's rule* (equation (2)) described above.

$$\begin{aligned} \Delta w_{s_1} &= \epsilon_b(\boldsymbol{x} - \boldsymbol{w}_{s_1}) \\ \Delta w_i &= \epsilon_b(\boldsymbol{x} - \boldsymbol{w}_i) \qquad \forall i \in N_{s_1} \end{aligned}$$

The age of all connections leading from the winner neuron are increased by 1 ($age(s_1, i) = age(s_1, i) + 1$ for all $i \in N_{s_1}$).

Removing

If there exist some connections with age greater than given a_{max} then all are removed. If this step results in neurons with no connections then remove also these standalone neurons.

Inserting new neurons

If the number of processed patterns reached an integer multiple of given parameter λ then new neuron is inserted using following steps:

1. First of all, the neuron p with the largest accumulated local error is determined using following equation.

$$p = arg \max_{a \in \mathcal{A}} \{E_a\}$$

   Among the neighbours of neuron p determine neuron r with largest accumulated local error.
2. Insert new neuron q to the network ($\mathcal{A} = \mathcal{A} \cup \{q\}$) and set its weight to the mean value of p and r weight vectors.

$$\boldsymbol{w}_q = \frac{1}{2}(\boldsymbol{w}_p + \boldsymbol{w}_r)$$

3. Insert new connection between new neuron q and neurons p and r ($\mathcal{C} = \mathcal{C} \cup \{(p, q), (r, q)\}$). Remove the old connection between neurons p and r ($\mathcal{C} = \mathcal{C} - \{(p, r)\}$).

[1] Note, neuron i is in direct topological neighbourhood of the winner c if there exists connection (i, c) between these two neurons.

4. Local accumulated error variables of neurons p and r are decreased by given fraction α.

$$\Delta E_p = -\alpha E_p \qquad \Delta E_r = -\alpha E_r$$

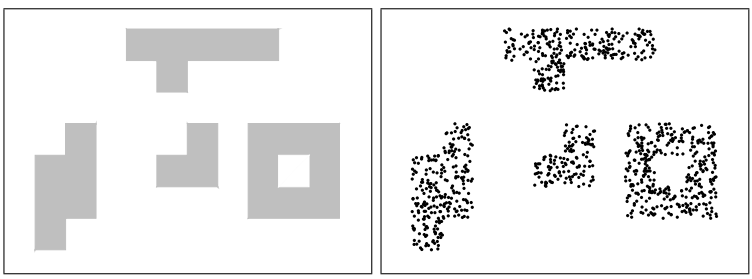
   The accumulated error variable of the new neuron q is set to the mean value of neurons p and r accumulated error variables.
5. Local accumulated error variables of all neurons in the network are decreased by given fraction β.

$$\Delta E_a = -\beta E_a \qquad \forall a \in \mathcal{A}$$

These several steps proceed until pre-defined termination condition is met. This could be some performace measure or usually net size.

6 Examples and comparison

As an example data for the subsequent clustering we will use data determined with distribution depicted in figure (1(a)). The highlighted regions are those where data are located and white regions contain no data. For the purposes of clustering using k-means method and CLASS method we have to generate the points complying given distribution in advance. A set of 1000 points from given distribution is depicted in (1(b)). For the usage with the SOM and GNG methods we may use continuous pseudo-random generator of patterns from the given distribution[2].

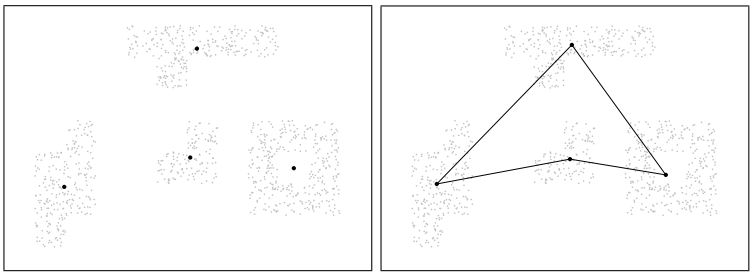


(a) Distribution (b) Objects from distribution

Fig. 1. Sample data for clustering.

[2] The same generator with the same seed we have also used to obtain set of discrete patterns for "classical" clustering. Without injury on generality we may say that explored sets of patterns were practically same.

First of all we have tested if k-means algorithm will produce similar partitioning as SOM neural network. If the number of typical points and number of neurons are set to actual number of clusters in data then both methods will place its representatives to the centers of clusters. Results of clustering using both methods with $K = 4$ are depicted in figure (2).



(a) k-means (b) SOM

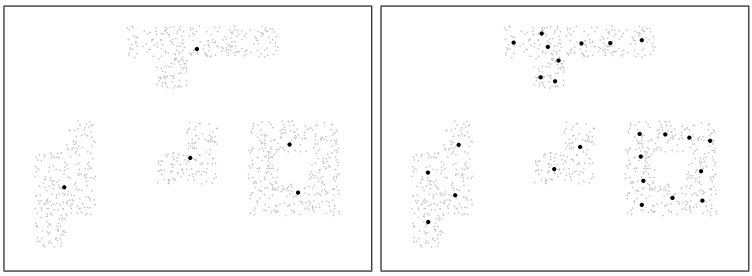
Fig. 2. Clustering using k-means algorithm and SOM neural net with number of representatives set to 4.

If the number of representatives is very slightly different from the actual number of clusters then obtained results are hardly interpretable. If the number is lesser then some representatives are trying to cover more clusters. If the number is greater then extra representatives will join another representative and they will cover one cluster with some more dense areas. The latter case may be incorrectly explained as presence of more clusters. In fact, representatives started to explain topology of clusters and besides finds more fine grained clusters. Figure (3) depicts situation where K was set to value 5 and 25.

It is clear that SOM is able to discover clusters in the input space as well as the k-means method.

Now we can focus on comparing the methods where splitting and merging of clusters is allowed. In case of neural net approach we may talk about net growth. One of the basic disadvantages of SOM is the inability to adapt its size according to proportions in the data (like k-means). On the other side, GNG can either add neurons where the local error is to large or remove neurons that have no relation to the rest (no existing connection).

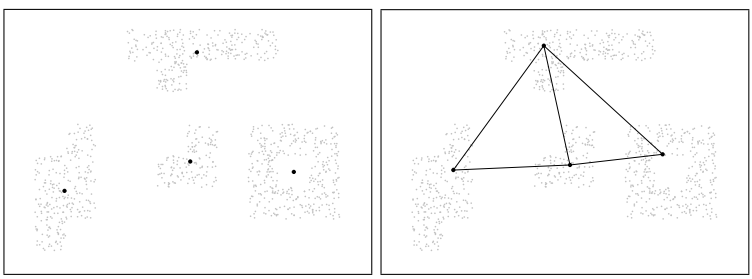
The issue we are interested at this time is how the CLASS method and the GNG method will proceed in clustering given data. Both methods are starting with two representatives. The CLASS method allows to add more representatives in one iteration in contrast to GNG which adds neurons after the pre-defined chunk of steps passes. So, we compared obtained partitionings at moment where both methods had identical number of representatives.



(a) k-means with $K = 5$ (b) k-means with $K = 25$

Fig. 3. Clustering using k-means algorithm with number of representatives set to 5 and 25.

The very first comparison was taken when both methods reached 4 representatives (the situation is depicted in figure (4)). As it is predictable, representatives are placed nearby the centers of the clusters to cover them as a whole.

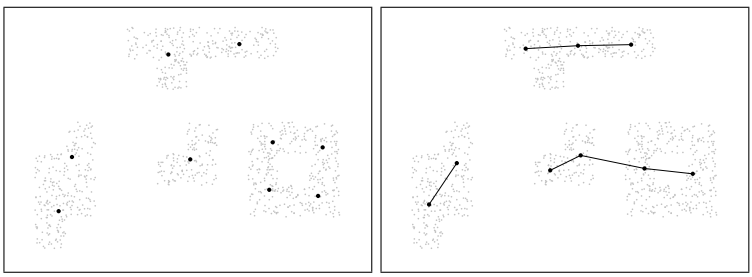


(a) CLASS (b) GNG

Fig. 4. Clustering using CLASS and GNG algorithms - results with 4 representatives.

More interesting results were reached when both methods started co cover clusters at the finer level. With 9 representatives we can distinguish little different behaviour of both methods. The dislocation of representatives obtained with CLASS method can be seen as dislocation of centers of circles with similar perimeter in an effort to cover all patterns – spherical clusters. On the other side the GNG proceeds with an extended goal. The GNG algorithm is covering all patterns with the neurons and moreover it covers the cluster's topology with the connections. We may see that in the figure (5(b)) there are three groups of interconnected neurons. We may interpret this situation for example as "there are three clusters with topology given by connections", but it is not so defini-

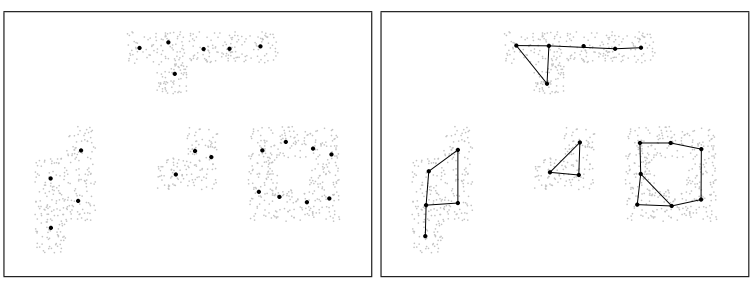
tive. In fact, there are four clusters but the GNG method does not discover them appropriately. Nevertheless, 9 representatives is still too little.



(a) CLASS (b) GNG

Fig. 5. Clustering using CLASS and GNG algorithms - results with 9 representatives.

Much more interesting results we can see when the number of representatives reaches value 21. The dislocation of representatives obtained by both methods is very similar and representatives cover all clusters effectively enough – they express the topology. But, there is remaining one more question – what is the resulting partitioning? In case of the CLASS method we have a set of non-related representatives. In 2-dimensional space we can do some visualisations to decide but in multi-dimensional space is the resulting partitioning uncertain. The problem lies in fact that the number of clusters is not the only thing we want to know. We need to know something more, it is how the clusters look like and how to formally express the category the cluster represents. Another big issue is the correctness of results obtained with these methods on multi-dimensional data.



(a) CLASS (b) GNG

Fig. 6. Clustering using CLASS and GNG algorithms - results with 21 representatives.

The connections between neurons of GNG are very helpful in this point of view. They inform about which neuron belongs to which cluster. From the figure (6(b)) we may decide on existence of 4 clusters and using the connections we can also state some conclusions on their shape or topology. But note, in the network could remain some edges that connect two neighbouring not so distant clusters. Thus, the results obtained with GNG method have to be evaluated carefully.

7 Conclusions

From the comparisons between k-means method and SOM neural network and between the CLASS method and GNG neural network we see that utilizing neural networks (with competitive learning) is good idea in the clustering domain. The most important feature of such neural networks is their natural ability to find dense areas in the input space. The extension of the basic SOM algorithm to dynamically reflect relations in the data (possibility of the net growth) makes neural networks even much more interesting.

The results obtained using both types of methods show that neural networks are able to give at least the same results. Moreover, the fact that SOM-like neural networks are likely to preserve the topology can mean that the results could be even better. Nevertheless, as using any other method for arbitrary knowledge discovery, the results have to be interpreted very carefully to be correct. This holds also in this case.

References

1. Everitt, B. S., Landau, S., and Leese, M. *Cluster analysis.* Oxford University Press, 2001.
2. Fritzke, B. A growing neural gas network learns topologies. *Advances in neural information processing systems 7* (1995).
3. Han, J., and Kamber, M. *Data mining: Concepts and techniques.* Morgan Kaufmann Publishers, 2000.
4. Jain, A. K., and Dubes, R. C. *Algorithms for clustering data.* Advanced reference series. Prentice hall, 1988.
5. Jain, A. K., Murty, M. N., and Flynn, P. J. Data clustering: A review. *ACM Computing Surveys 31*, 3 (September 1999), 264 – 323.
6. Kohonen, T. *Self-organizing maps.* Springer Verlag, 1984.
7. Lukasová, A., and Šarmanová, J. *Metody Shlukové Analýzy.* SNTL, 1985.
8. Šíma, J., and Neruda, R. *Teoretické otázky neuronových sítí.* Matfyzpress, 1996.

On classification of XML document transformations*

Jana Dvořáková

Department of Computer Science, Faculty of Mathematics, Physics and Informatics, Comenius University, Bratislava, Slovak Republic
Jana.Dvorakova@dcs.fmph.uniba.sk

Abstract. As XML has become a very popular standard for data in many fields, the domain of XML documents transformations is becoming more and more important. In this paper we propose classification hierarchy for XML document transformations. We assign implemented transformation systems into defined groups according to the type of possible transformations. This enables user to choose appropriate transformation system according to the requirements for transformation. Secondly, we are concerned with the group of transformation systems, which enable type transformation. We define underlying formal models in common framework and discuss their applicability for these transformations.

Key words: XML, Structured document, Document type, Document transformation, Transformation classification

1 Introduction

XML (Extensible Mark-up Language) [28] is a meta-language recommended by W3 Consortium in order to create structured documents. In XML document structure, content and presentation are strictly separated. Unlike plain text, it contains special tags, which decompose document into logical parts. Therefore we say that XML belongs to the group of mark-up languages. Presentation of XML document can be described by several languages, usually in external file. Most common are XSL (Extensible Stylesheet Language) and CSS (Cascading Stylesheet).

Nowadays the usage of XML is growing very fast. It is a suitable tool in every field, where it is necessary to create document standards. Furthermore it has become very popular as a format for data exchange among applications since for programs it is necessary to mark semantics of data explicitly.

For this reasons various transformations are needed. Many transformation systems for structured documents have been implemented, some of them are based on formal models while others were created only ad-hoc. We have seen several reviews of existing transformation systems ([19, 18, 20, 11]). The author

* This work was supported in part by the grant VEGA 1/0131/03

in [20] introduces also some basic categorization, however in none of these works a complete classification system has been defined.

The rest of this paper is organized as follows: After defining some notions used through-out the paper we introduce defined classification system in Section 3. We discuss reasons for choosing particular criteria and we assign implemented transformations systems for structured documents into defined groups. In section 4, we are dealing with one specific group of transformations, namely type transformations. We are examining formal models on which these transformations are based and we define them in a common framework. At last we present several results obtained by comparing these formal models. Section 5 contains a brief conclusion and outlines our future research.

2 Notions and notations

2.1 XML document

Basic logical unit of an XML document is *an element*. The content of an element is delimited by a start-tag and an end-tag. It can contain text and other nested elements as well. Variables called *attributes* can be assigned to an element. Obviously, an XML document has a hierarchical structure and it can be represented by a tree, where internal nodes are elements and leaves contain textual content.

The framework considered in this paper is restricted in two ways. Firstly, we do not consider element attributes of XML documents since we are concerned mainly with transformations at the level of elements. Secondly, we assume that element names of XML documents are from a finite and known set denoted by E. This enables us to define XML documents as trees over finite alphabets, which is the most natural way as far as we consider XML documents transformation. Furthermore we will use symbol $Char$ for alphabet of characters allowed to appear in the textual content of XML document. $Char$ is specified in W3C recommendation [28].

Definition 1. *Let Σ, Γ be alphabets. The set of trees over (Γ, Σ) denoted by $T_\Gamma(\Sigma)$ is defined as follows:*

- *$a \in \Sigma$, then $t = a \in T_\Gamma(\Sigma)$ and $root(t) = a$.*
- *$A \in \Gamma$, $t_1, \ldots, t_k \in T_\Gamma(\Sigma)$, $k > 0$, then $t = A(t_1, \ldots, t_k) \in T_\Gamma(\Sigma)$ and $root(t) = A$.*
- *nothing else belongs to $T_\Gamma(\Sigma)$.*

We call Σ *a leaf alphabet* and Γ *an internal node alphabet*. Alphabets Σ, Γ do not need to be disjoint.

Obviously the set of XML documents denoted by D equals to the set of trees over $(E, Char)$, i.e., $D = T_E(Char)$.

2.2 DTD

An XML document type is a class, which contains XML documents with similar structure. It is specified by *a type definition*, which describes the set of elements, that should be contained in the document as well as relationships among the elements. W3C has defined two ways of notation - DTD and XML schema. DTD is simpler and describes particularly syntax of the type while XML schema introduces more aspects like namespaces, restrictions for attribute values, etc. However, syntactic features of both models can be easily captured by context-free grammar. We will use this concept in the rest of this paper. Now we will define context-free grammars and their subset - type grammars, which describe XML document types.

Definition 2. *Context-free grammar (CFG) is a 4-tuple* $G = (N, T, P, S)$*, where* N *is an alphabet of nonterminal symbols,* T *an alphabet of terminal symbols,* $P \subseteq N \times (N \cup T)^*$ *is a finite set of production rules and* $S \in N$ *is a starting symbol.*

Definition 3. *Let* $G = (N, T, P, S)$ *to be a CFG, then a set of derivation subtrees of* G *denoted by* S_G *is defined as follows:*

- $a \in T$*, then* $t = a \in S_G$*.*
- $A \in N$*,* $t_1, \ldots, t_k \in S_G$ *and* $A \rightarrow root(t_1), \ldots, root(t_k) \in P$*, then* $t = A(t_1, \ldots, t_k) \in S_G$*.*
- *nothing else belongs to* S_G*.*

A set of derivation trees of G *-* T_G *is a subset of* S_G *such that* $t \in T_G \Leftrightarrow t \in S_G$ *and* $root(t) = S$*.*

A type grammar is a context free grammar $G = (N, T, P, S)$, such that $N = E$, $T = Char$. We denote its set of derivation trees D_G since we consider documents rather then general trees. Obviously it holds $D_G \subseteq D$. Nonterminals of a type grammar represent element names and the set of terminals equals to the text alphabet.

If we have a given type grammar, it generates a class of XML documents, which equals to the set of its derivation trees. *Validation* of XML document against a grammar is a process, which gives us as a result an answer, whether given XML document belongs to the class of documents generated by given grammar. If the answer is yes, we say that the XML document is *valid* for a given type grammar (or *correct*).

2.3 XML transformations

A transformation of an XML document takes some source documents as an input, it processes them according to the transformation specification and it gives target documents as an output.

It is reasonable to define a transformation of XML documents as a relation on the set of XML documents rather then a function [1]. As usual we begin with more general definition, i.e., tree transformation.

Definition 4. *Let Σ, Σ', Γ, Γ' be alphabets. A tree transformation from (Γ, Σ) to (Γ', Σ') is a relation $\tau \subseteq T_\Gamma(\Sigma) \times T_{\Gamma'}(\Sigma')$.*

If we apply previous definition on XML documents, we obtain restricted case again. Thus, *an XML documents transformation* is a relation $\tau \subseteq D \times D$. Obviously if we consider any XML document transformation, the source leaf alphabet equals to the target leaf alphabet.

2.4 Tree properties

Now we introduce several definitions related to trees that we will need later in this paper.

Let $\Sigma, \Sigma', \Gamma, \Gamma'$ be alphabets.

Let $X = \{x_1, x_2, \ldots\}$ be a set of *variable symbols*. For each $k > 0$, we denote by X_k a set of first k variables, thus $X_k = \{x_1, \ldots, x_k\}$. We denote by $T_\Gamma(\Sigma \cup X)$, $T_\Gamma(\Sigma \cup X_k)$ sets of trees with these variables, where it must hold $\Sigma \cap X = \emptyset$.

Definition of *a tree substitution* follows. Let $t \in T_\Gamma(\Sigma \cup X_k)$, $k > 0$ and $t_1, \ldots, t_k \in T_{\Gamma'}(\Sigma')$. Then $t[x_1 \leftarrow t_1, \ldots, x_k \leftarrow t_k]$ is a tree obtained from t by replacing each occurrence of x_i by a tree t_i, $1 \leq i \leq k$[2].

Let $Z = \{z_1, z_2, \ldots\}$ be a set of variables disjoint to X and $k > 0$. A set of (Γ, Σ, k)-*contexts* denoted $C_\Gamma(\Sigma, k)$ is the set of trees t from $T_\Gamma(\Sigma \cup Z_k)$, such that for each $1 \leq i \leq k$ the symbol z_i appears exactly once in t.

We will use a simpler notation for context substitution. Let t to be a (Γ, Σ, k)-context and $t_1, \ldots, t_k$ trees. Then we use $t[t_1, \ldots, t_k]$ instead of $t[z_1 \leftarrow t_1, \ldots, z_k \leftarrow t_k]$.

Let $t, t' \in T_\Gamma(\Sigma)$. Then t' is a *subtree* of t if there exists such a context $\beta \in C_\Gamma(\Sigma, 1)$ that $t = \beta[t']$.

For each tree $t \in T_\Gamma(\Sigma)$ $path(t) \subseteq \{1, 2, \ldots\}^*$ is a set of all *paths* of the tree t, so that each of them unambiguously identifies a node of the tree t. Then symbol ε represents $root(t)$. For each path $w \in path(t)$, we denote by $label_t(w)$ a label of the node identified by w and by $t|_w$ a subtree under this node.

Let $t \in T_\Gamma(\Sigma)$, $t' \in T_{\Gamma'}(\Sigma')$. We obtain tree $t[w \leftarrow_p t'] \in T_{\Gamma \cup \Gamma'}(\Sigma \cup \Sigma')$ from t by replacing subtree in $w \in path(t)$ by tree t'.

Let $t \in T_\Gamma(\Sigma)$, $patt \in T_\Gamma(\Gamma \cup \Sigma)$. We say, that t and *patt match*, if there is such a context $\gamma \in C_\Gamma(\Sigma, k)$ that $t = \gamma[t_1, \ldots, t_k]$ for some $t_1, \ldots, t_k \in T_\Gamma(\Sigma)$ and it holds $patt = \gamma[root(t_1), \ldots, root(t_k)]$.

[1] For example. in the system SynDoc [17], several target documents can be generated as the result of transformation and user can choose the most appropriate one.

[2] In the definition of tree substitution variables are not typed, however, later we will require newly connected subtrees to satisfy some specific conditions.

3 Classification hierarchy

There are several possibilities, how to divide XML document transformations into categories. According to the driving element of the transformation we obtain three basic categories as follows. In each of them different techniques for implementing particular transformation systems are used, thus, it is reasonable to start with this criterion.

1. *Source grammar transformations* - Transformation process is driven by the source structure. First, the source document is parsed and then according to recognized syntactic elements parts of the output documents are constructed. Usually it is possible to check correctness of input documents, but target correctness is not ensured in this case. If necessary, user must perform validation explicitly. Next we can divide this category into two more specific groups according to the way how the source document is parsed:
   - *Event-driven transformations* - The source document is read as a data stream. As the result, we obtain a list of recognized events. The event can be an occurrence of a start-tag, an end-tag, an attribute, etc. The transformation system reads the list o events and performs corresponding actions. If we allow side-effect functions, an output can be generated already at parsing time. As a formal model, an attribute grammar is often used. Event-driven models require little memory, and provide fast and effective way of accessing XML data. On the other hand, only simple transformations can be performed since it is not possible to return to an earlier part of the document. In some document transformers (e.g. CoST [12, 7], OmniMark [8]) the list of recognized events is represented by *ESIS (Element Structure Information Set)*, which is the part of ISO SGML standard [13]. However, currently especially SAX (Simple API for XML) [26] is more and more in use as event-driven mechanism for parsing XML documents. It generates specific SAX events and sends them to the application.
   - *Tree-based transformations* - First, an internal syntactic tree of the source document is constructed and a target document is generated via queries on this tree. In most of the cases transformation systems are based on tree pattern matching and replacement. We can assign widely used XSLT [29] and its predecessor DSSSL [14] as well as systems Scrimshaw [2], Metamorphosis [22], Balise [3] and TranSID [20] to this group.

     XSLT, a recommendation of W3 Consortium, has become very popular recently. It is a language, itself written in XML, with powerful capabilities for specifying transformations of XML documents. XSLT program (called stylesheet) consists of a set of template rules. A template rule associates a pattern, which matches nodes in the source document with corresponding template that can be instantiated to form the target tree. Although XSLT is a powerful transformation language, it has several drawbacks. Firstly, it appears to be a complex language and therefore

transformation specification must be written by an expert. Secondly, XSLT processor cannot guarantee the correctness of target documents as well as other systems in this group. However, an XSLT stylesheet can be produced as an output of some two-grammars systems (see next section). Then it is ensured, that generated stylesheet specifies a type transformation and XSLT processor can be used to perform transformation.

2. *Target grammar transformations* - Target document is created according to the rules of a given target grammar while relevant data from a source document are extracted via queries. This ensures target correctnes. However, we do not need to have any information about source document type. We found only one transformation system, which belongs to this group - TREX [30].
3. *Two grammar transformations (type transformations)* - Documents of a given source type are translated into documents of a given target type. Transformation systems are mostly based on one of these formal models - syntax directed translation schema, tree transformation grammar, descending tree transducer and higher order attribute grammar. We discuss these models as well as two grammar transformation systems in detail in the next section. However, there are few systems, which performed two-grammars transformations, but underlying formal model cannot be clearly recognized. These are Grif [25] and Thot [4].
   From a different point of view, we distinguish *static type transformations*, which translate whole documents, and *dynamic type transformations*, which are used to perform dynamic operations on XML documents. An example is *cut and paste* operation, which is a standard operation in XML document editors. Unlike in common editors, in this case it is necessary to preserve document structure. Thus, a local transformation must be performed in the place, where a part of another document has been pasted.

4 Type transformations

The group of type transformations (or two-grammar transformations) is a specific one. In this case there must be only correct documents taken as an input and correct documents must be generated on the output side. To satisfy these conditions, it is usually inevitable to implement transformation system performing this kind of transformations on some underlying formal model. In this section we introduce four formal models on which some of the implemented transformation systems are based. We developed common framework in which we define these models. Our intention was to create a formal base, so that the models can be studied and mutually compared later. We also present current results obtained by comparing some of them according to their transformational power. Obviously there is a direct proportion between the complexity of transformation specification and the set of possible transformations. Thus, the more powerful particular model is, the more complex specification the user is supposed to write.

4.1 Syntax-directed translation schema

Definition 5. *A syntax-directed translation schema - SDTS [1] is a 5-tuple* $\Omega = (N, \Sigma, \Delta, R, S)$*, where* N *is an alphabet of nonterminals,* Σ*,* Δ *are input and output alphabets,* $S \in N$ *is the start symbol and* R *is a finite set of rules of the form* $A \to \alpha, \beta$*, where* $\alpha \in (N \cup \Sigma)^*$*,* $\beta \in (N \cup \Delta)^*$ *and nonterminals in* β *are a permutation of nonterminals in* α*.*

In a rule $A \to \alpha, \beta$ with each nonterminal from α there is associated an identical nonterminal of β. If there is a multiple occurence of some nonterminal, we indicate association by using integer superscripts. Obviously if we extend definition of a SDTS by a renaming homomorphism for nonterminals, we can use this model also in the case, when it is neccesary to associate nonterminals with different names.

A context-free grammar $G_s = (N, \Sigma, P_s, S)$, where $P_s = \{A \to \alpha \mid \exists \beta \in (N \cup \Delta)^*, A \to \alpha, \beta \in R\}$ is a *a source grammar* and $G_t = (N, \Delta, P_t, S)$, where $P_t = \{A \to \beta \mid \exists \alpha \in (N \cup \Sigma)^*, A \to \alpha, \beta \in R\}$ is *a target grammar* of SDTS Ω.

Now we mentioned two modifications of SDTS, which differ from the basic model in a definition of rules:

• *Simple SDTS - SSDTS*

R is a finite set of rules of the form $A \to \alpha, \beta$, where $\alpha \in (N \cup \Sigma)^*$, $\beta \in (N \cup \Delta)^*$, nonterminals in α, β are identical and the order of nonterminals is preserved.

• *Extended SDTS - ESDTS*

R is a finite set of rules of the form $A \to \alpha, \beta$, where $\alpha \in (N \cup \Sigma)^*$, $\beta \in (N \cup \Delta)^*$ and nonterminals in β are a permutation of a subset of nonterminals in α.

A syntax-directed translation schema simulates derivations of two context-free grammars with similar set of rules simultaneously. While using a SDTS in the domain of syntax analysis it was sufficient to store only frontiers of the derivation trees in configurations, if we consider tree transformations it is more natural to keep the whole derivation tree. Then a transformation consists of pairs of trees, where the first one is a derivation tree of the source grammar and the second one is a derivation tree of the target grammar. We started from an algorithm for structured documents transformations using an ESDTS presented in [19]. We skip an operation of adding new nodes into the source tree since authors do not specify exactly what subtree is embedded to the newly created node.

Definition 6. *A configuration of SDTS (SSDTS, ESDTS)* Ω *is a tree from the set* $T_{N \cup \{\uparrow\}}(\Sigma \cup \Delta)$*.*

Definition 7. *We say, that a pair of trees* $(A(t_1, \ldots, t_n), A(s_1, \ldots, s_m))$*,* $A \in N, t_1, \ldots, t_n, s_1, \ldots, s_m \in T_{N \cup \{\uparrow\}}(\Sigma \cup \Delta)$ *realizes a rule* $A \to u_1 \ldots u_n, v_1 \ldots v_m \in R$ *if the following holds:*

1. $u_1 \ldots u_n = root(t_1) \ldots root(t_n)$ *and* $v_1 \ldots v_m = root(s_1) \ldots root(s_m)$*,*

2. $h_N(u_1 \dots u_n) = u_{i_1} \dots u_{i_r}$ *and* $h_N(v_1 \dots v_m) = v_{j_1} \dots v_{j_p} = u_{i_{k_1}} \dots u_{i_{k_p}}$, $i_1, \dots, i_r \in \{1, \dots, n\}$ *(different),* $j_1, \dots, j_p \in \{1, \dots, m\}$ *(different),* $k_1, \dots, k_p \in \{1, \dots, r\}$ *(different),*
3. $s_{j_l} = t_{i_{k_l}}$ *for* $l \in \{1, \dots, p\}$, $s_l = v_l$, $v_l \in \Delta \cup \{\varepsilon\}$ *for* $l \notin \{j_1, \dots, j_p\}$ *and* $t_l = u_l$, $u_l \in \Sigma \cup \{\varepsilon\}$ *for* $l \notin \{i_1, \dots, i_r\}$.

Definition 8. *A translation step of SDTS (SSDTS, ESDTS)* Ω *is a relation* $\Rightarrow_\Omega$ *over the set of configurations defined as follows:*

1. $\beta[\uparrow (a)] \Rightarrow_\Omega \beta[a]$, $a \in \Delta \cup \{\varepsilon\}$.
2. $\beta[\uparrow (A(t_1, \dots, t_n))] \Rightarrow_\Omega \beta[A(\uparrow (s_1), \dots, \uparrow (s_m))]$ *and a pair of trees* $(A(t_1, \dots, t_n), A(s_1, \dots, s_m))$ *realizes some rule* $r \in R$.

In both cases β *is a context from* $C_{N \cup \{\uparrow\}}(\Sigma \cup \Delta, 1)$.

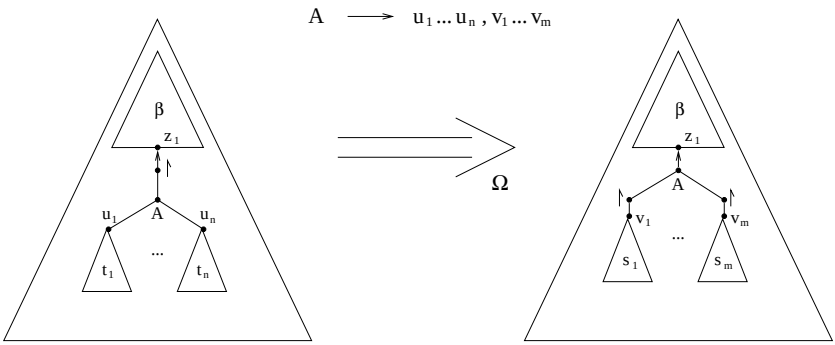


Fig. 1. A translation step of a SDTS

An input tree is processed from the root to the leaves, while unprocessed subtrees are marked by symbol $\uparrow$. Initially, the whole source tree is marked as unproccessed. If the root of a currently processed subtree has a label from output alphabet or ε, then it is a leaf and we stop translation in this branch by removing the symbol $\uparrow$. If we are processing a subtree, whose root is an internal node, first we reorder and delete children subtrees according to corresponding rule. Subsequently, if there are some leaf childrens, we reorganize them in such a way, that the result adheres to the output side of the rule.

Definition 9. *A transformation generated by a SDTS (SSDTS, ESDTS)* Ω *is a set* $\tau(\Omega) = \{(t_s, t_t) \mid t_s \in T_N(\Sigma), root(t_s) = S, t_t \in T_N(\Delta), \uparrow (t_s) \Rightarrow_\Omega^* t_t\}$.

A SSDTS enables very simple transformations of trees. It does not change structure of the source tree, it is only possible to delete, add or reorder leaves. A SDTS enables reordering of subtrees connected to the same node moreover. An ESDTS is the most powerful modification, unlike SDTS it enables also deleting of arbitrary subtree in the source tree.

A transformations performed by SSDTS as well as SDTS satisfy the definition of the type transformation. In the case of ESDTS a problem arises, because if

we delete a subtree in the source tree, it is not processed anymore. Thus, it is not possible to check, whether this part of the source tree was correct.

There are several implemented transformation systems that are using SDTS and its modifications as underlying model. System SynDoc [19] is based on ES-DTS, and SDTT [5], ICA [21], HST [16] are based on SDTS.

4.2 Tree transformation grammar

A tree transformation grammar is similar to SDTS, however in this case we associate two groups of production rules. The first group - *a source subgrammar* contains some of the source grammar production rules and the second one - *a target subgrammar* conatins some of the target grammar production rules. Unlike in SDTS, nonterminals in the source and target subgrammar must be associated explicitely. This means, that it is possible to associate also nonterminals with different names. Additionally, tree-transformation grammars work with as many levels in the source tree as necessary.

Definition 10. *A tree transformation grammar - TTG is a 6-tuple* $G = (G_s, G_t, Sub_s, Sub_t, PA, SA)$*, where*

- $G_s = (N_s, \Sigma_s, P_s, S_s)$ *and* $G_t = (N_t, \Sigma_t, P_t, S_t)$ *are the source and target grammars,*
- $Sub_s \subseteq 2^{P_s}$ *and* $Sub_t \subseteq 2^{P_t}$ *are sets of source and target subgrammars,*
- $PA \subseteq Sub_s \times Sub_t$ *is a set of production group associations,*
- $SA \subseteq N_s \times N_t$ *is a set of symbol associations.*

There are some restrictions put on the source subgrammar. It must contain a single start nonterminal, and every other nonterminal in the source subgrammar must be derivable from this start nonterminal. Source subgrammars represent subtree patterns to be matched against in the source tree; target subgrammars represent subtrees, that are to be generated as part of the target tree. A target subgrammar can contain several start nonterminals, thus we can obtain a forest of target subtrees as a result. Transformation via TTG is performed by processing source tree nodes one by one. For a particular node we look for matching source subgrammars. If there is some, corresponding target subtrees are constructed and links among nodes in source subgrammar and target subgrammar are marked according to symbol associations. After the whole source tree has been processed, target subtrees can join to bigger trees according to the marked symbol associations.

Tree transformation grammars are described in [15]. Authors' intention was to define a powerful two-grammar transformations model and, at the same time, provide simple and natural way to write transformation specification.

Several modifications of tree transformation grammars have been implemented:

- *Dual grammar translation scheme (DGTS)* - system SSAGS [24].
- *Single input production-explicitely qualified (SIPEQ)* - system SIPEQ [15].
- *TT grammar* - system Alchemist [20].

However, semantics of the model remains unclear as no formal definitions have been introduced in any of the resources mentioned. Now we present definitions, that we created according to the alghorithm of tree transformation via TT grammars mentioned in [20].

Definition 11. *A configuration of TTG Ω is a pair (t_s, T_t), $t_s \in T_{N_s \cup \{\uparrow\}}(\Sigma_s)$ and $T_t \subseteq \{(t, W) \mid t \in T_{N_t}(N_t \cup \Sigma_t), W \subseteq 2^{(path(t) \times path(t))}\}$.*

A tree t_s is the source tree with marked unprocessed nodes. T_t is *a set of target linked trees*, i.e., it contains pairs of the form (t, W), where t is a particular target tree and W is a set of path pairs that we call *a set of links of a tree t*. We use the notation W_t as well.

Definition 12. *A translation step of TTG Ω is a relation $\Rightarrow_\Omega$ over configurations defined by $(t_s, T_t) \Rightarrow_\Omega (t'_s, T'_t) \iff$*

(1) generating step

- *there is such a path w_s in a tree t_s that $t_s|_{w_s} = \uparrow (A(t_1, \ldots, t_n))$,*
- *$t'_s = t_s[w_s \leftarrow_p A(\uparrow (t_1), \ldots, \uparrow (t_n))]$,*
- *there is such a source subgrammar $sg \in Sub_s$ that t_{sg} (a derivation tree corresponding to the rules sg) and $A(t_1, \ldots, t_n)$ match,*
- *there is such a target subgrammar $tg \in Sub_t$ that $(sg, tg) \in PA$ and $T_{tg} = \{r_1, \ldots r_m\}$ (derivation trees corresponding to the rules tg). Let us denote $R_i = (r_i, \{(w_1, w_s w_2) \mid (label_{r_i}(w_1), label_{t_{sg}}(w_2)) \in SA\})$, $i \in \{1, \ldots, m\}$. Then $T_{t'} = T_t \cup \bigcup_{1 \leq i \leq m} R_i$.*

(2) connecting step

- *$t'_s = t_s$,*
- *there are such $R_1, R_2 \in T_t$, $R_1 = (r_1, W_1)$, $R_2 = (r_2, W_2)$ that*
  - *$root(r_2) = A \in N_t$,*
  - *there is such a path $w \in path(r_1)$ that $r_1|_w = A$,*
  - *there is such a path $w_s \in path(t_s)$ that $(w, w_s) \in W_1$ and $(\varepsilon, w_s) \in W_2$.*

  Let $r_3 = r_1[w \leftarrow_p r_2]$, $W_3 = W_1 \cup \{(ww_1, w_2) \mid (w_1, w_2) \in W_2\}$ a $R_3 = (r_3, W_3)$. Then $T_t = T_{t'} - R_1 - R_2 \cup R_3$.

Transformation works in similar way as with SDTS. The source tree is passed in the top-down manner, while unprocessed nodes are marked by symbol $\uparrow$, initially the whole source tree is marked.

During the generation step, we first choose such a source subgrammar (pattern), that matches a subtree under currently processed node (Fig. 2). The model is nondeterministic, i.e., there can be more suitable source subrammars. Then we add target subtrees (corresponding to the associated target subgrammar) together with links to the set of linked target trees. Links are created according to symbol associations (Fig. 3). At last we remove symbol $\uparrow$ from current subtree and again we mark all connected subtrees to indicate they need to be processed.

If we apply connecting step (Fig. 4), we first choose two subtrees r_1 and r_2 from the set of target linked trees. These trees must satisfy some conditions;

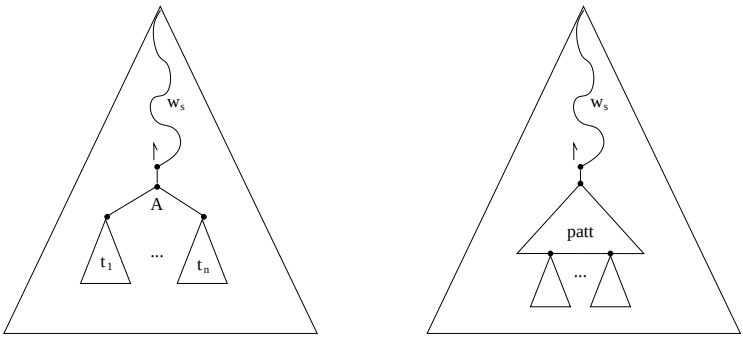


Fig. 2. Different views of the same source tree

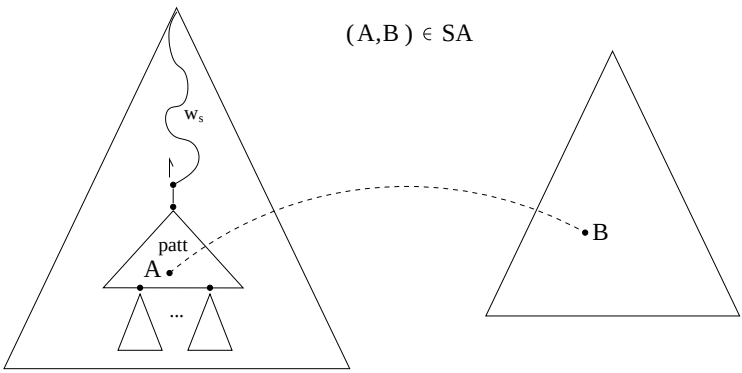


Fig. 3. New links created by the generation step

the root of the first one and a leaf of the second one must have the same label and must be linked to the same node in the source tree. Consequently, the set of target linked trees is updated so that subtrees r_1, r_2 are removed and a new subtree created by connection of r_1, r_2 is added. Links between this new subtree and the source tree will be updated as well according to the links in old subtrees. However, we want to obtain one tree as a result of transformation. Therefore in following definition of tree transformation via TTG we require the last configuration to consist of a single element.

Definition 13. *A transformation generated by TTG Ω is a set* $\tau(\Omega) = \{(t_s, t_t) \subseteq T_{N_s}(\Sigma_s) \times T_{N_t}(\Sigma_t) \mid (\restriction(t_s), \emptyset) \Rightarrow^*_\Omega (t_s, \{(t_t, W)\})\}$.

4.3 Descending tree transducer

A descending tree transducer is an automaton, which passes a source tree from the root to the leaves and performs changes according to the current state, node

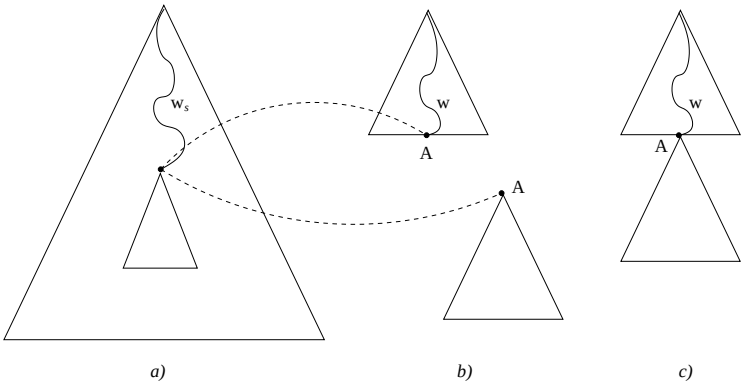


Fig. 4. Linking trees by connecting step a) source tree b) target trees before the connecting step c) new target tree

and lookeahead. Unlike previous formal models, it works with trees, where labels of internal nodes are from a ranked alphabet. Basically, ranked alphabet is a pair $(\Gamma, rank)$, where Γ is an alphabet and $rank : \Gamma \rightarrow N$ is a ranking function, which assigns positive integer to each symbol in Γ. We usually omit the function $rank$ in the notation and we say that Γ is a ranked alphabet. If Σ is an alphabet and Γ a ranked alphabet, then each internal node in a tree over (Γ, Σ) must have exactly as many children as is the value of the rank of its label. We will not introduce formal definitions here, all of them can be found in [10] and the notation is very similar to that one used in previous cases.

A DTT does not represent a model for two-grammars transformations since it works on higher level of abstraction. However, it is easy to restrict definitions in such a way, that we obtain DTT, which transforms derivation trees of a given source grammar into derivation trees of a given target grammar.

A DTT with finite and regular lookahead has been used in the transformation system SynDoc [17]. This system generates also an XSLT stylesheet as an output and it is possible to use existing tools for XSLT if further processing is needed. However, in that case the target correctness is not guaranteed anymore.

4.4 Higher order attribute grammar

Basically, *an attribute grammar* is a context free grammar, such that each rule is associated with a set of semantics rules. These rules have a form $X.b := f(Y_1.c_1, \ldots, Y_n.c_n)$, where X, Y_i are nonterminals, b, c_i are attributes and f is an n-ary function.

All definitions presented in this section results from [27].

Definition 14. *An attribute grammar - AG is a triple* $AG = (G, A, R)$*, where*

- $G = (N, \Sigma, P, S)$ *is a context-free grammar,*

- $A = \bigcup_{X \in N \cup \Sigma} AIS(X)$ *is a finite set of attributes, where* $AIS(X)$ *is a set of attributes associated with nonterminal* X*,*
- $R = \bigcup_{p \in P} R(p)$ *is a finite set of attribute rules. If* $AIS(X) \cap AIS(Y) \neq 0$*, then* $X = Y$*. For every occurrence of a nonterminal in a derivation tree of* G *there must be exactly one attribute rule applicable for computation of a value for an attribute* $a \in A$*. Rules in* $R(p)$ *have a form* $\alpha = f(\ldots, \gamma, \ldots)$*, where* f *is a name of the function,* α *and* γ *are attributes of a form* $X.a$*.*

Definition 15. *For each* $p : X_0 \to X_1 \ldots X_n \in P$ *we define a set of attribute evaluation occurrences as* $AF(p) = \{X_i.a \mid X_i.a = f(\ldots) \in R(p)\}$*. An attribute* $X.a$ *is called a synthetized attribute if there is a rule* $r : X \to \chi$ *and* $X.a \in AF(r)$*; an inherited attribute, if there is a rule* $q : Y \to \mu X \nu$ *and* $X.a \in AF(q)$*.*

We denote by $AS(X)$ a set of synthetized attributes of a nonterminal X and by $AI(X)$ a set of inherited attributes of a nonterminal X. After evaluation process these attributes contain a subtree as a computed value. Thus, unlike in AG, in HAG the domain of the derivation tree and the domain of attributes overlap.

A higher-order attribute grammar (HAG) is an extended attribute grammar. A special type of attribute - *nonterminal attribute* is introduced.

Definition 16. *For each* $p : X_0 \to X_1 \ldots X_n \in P$ *a set of nonterminal attributes is defined:* $NTA(p) = \{X_j | X_j := f(\ldots) \in R(p)\}$*.*

An unevaluated nonterminal attribute represents a leaf node of the actual tree. At this point we call it *virtual nonterminal*, since there is no subtree connected to it. After the evaluation is performed, a subtree is computed as a value for nonterminal attribute and consequently it is connected to this attribute (leaf node). Evaluated nonterminals and common nonterminals we also call *instatiated nonterminals* since they have got an instance, i.e. connected subtree. The set of instantiated nonterminals represents internal nodes of the actual tree.

A HAG has been implemented in transformation system SIMON [9], which takes advantage of the power of this model and enables several complex transformations. However, there is one major drawback. System requires a user to write a complete HAG as a transformation specification and this is usually a nontrivial task.

For HAG there has been defined formal model - *a higher-order attribute tree transducer* [23], which represents some abstraction and therefore it is more suitable for studying its properties.

4.5 Comparison results

Finally, we briefly introduce results obtained by comparing formal models for type transformations according to their transformational power. Since we defined transformations as relations over sets of trees, we can consider them as sets containing pairs of trees as elements. Thus, we can also compare them as sets. See Table 1 for comparison results. Symbol N indicates that transformations of two corresponding formal models are not comparable. More details and all formal proofs can be found in [6].

Table 1. Comparison results

	SDTS	**ESDTS**	**d-DTT**	**DTT**
SDTS		$\subsetneq$	$\not\supseteq$	$\subsetneq$
ESDTS	$\supsetneq$		N	N
d-DTT	$\not\subseteq$	N		$\subseteq$
DTT	$\supsetneq$	N	$\supseteq$	

SDTS: syntax directed translation schema, ESDTS: extended syntax directed translation schema, d-DTT: deterministic descending tree transducer, DTT: descending tree transducer

5 Conclusion and future work

In this paper we introduced a system for classification of XML documents transformations. Our intention was to simplify choosing of appropriate transformation system according to given requirements for transformation. We defined categories in bottom-up way, i.e. first we were examining implemented transformation systems and consequently we abstracted from them to higher-level groups of transformations sharing similar features.

We were examining in detail formal models used for type transformation. We defined common framework for these models and we introduced current results of their mutual comparison. In our future work, we plan to make more comparisons according to transformational power and complexity as well. We intend to include also other models, which might appear to be suitable for XML document transformations.

References

1. A. V. Aho and J. D. Ullman. *The theory of parsing, translation and compiling, Vol. I: Parsing.* Prentice-Hall, Inc., Englewood Cliffs, N.J., USA, 1972.
2. D. S. Arnon. Scrimshaw: A language for document queries and transformations. *Electronic Publishing, 6(4)*, 1993.
3. Berger-Levrault/AIS: *Balise Reference Manual, Release 3*, 1996.
4. S. Bonhomme. *Transformation de documents structurés, une combinaison des approches explicite et automatique.* PhD thesis, University Joseph Fourier - Grenoble, 1998.
5. K. Chiba and M. Kyojima. Document transformation based on syntax-directed tree translation. *Electronic Publishing, 8(1)*, 1995.
6. J. Dvořáková. *Transformácie XML dokumentov.* Master thesis, FMFI UK Bratislava, 2004.
7. Joe English. *CoST 2 Reference Manual. Release 3*, 1996. `http://www.art.com/cost/manual.html`.
8. Exoterica Corporation. *OmniMark Programmer's Guide*, 1993.
9. A. Feng and T. Wakayama. SIMON: A grammar-based transformation system for structured documents. *Electronic Publishing, 6(4)*, 1993.

10. Z. Füllöp and H. Vogler. *Syntax-directed semantics: Formal models based on tree transducers*. Springer-Verlag, 1998.
11. V. Hambálková. *Transformácie štruktúrovaných dokumentov*. FMFI UK Bratislava, 2000.
12. K. Harbo.*CoST Version 0.2 - Copenhagen SGML Tool*. Department of Computer Science and Euromath Center, University of Copenhagen, 1993.
13. ISO - International Organization for Standardization. *Information Processing - Text and Office Systems - Standard Generalized Markup Langugage (SGML), ISO 8879*, 1986.
14. ISO - International Organization for Standardization. *Information Technology - Text and Office Systems - Document Style Semantics and Specification Language (DSSSL), ISO/IEC DIS 10179:1996*, 1996.
15. S. Keller, J. A. Perkins, T. F. Payton and S. P. Mardinly. Tree transformation techniques and experience. *Proceedings of the ACM SIGPLAN '84 Symposium on Compiler Construction, SIGPLAN Notices, 19(6)*, Montreal, Canada, 1984.
16. P. Kilelinen, Greger Lindén, H. Manilla and E. Nikunen. A structured document database system. *EP90 - Proceedings of the International Conference on Electronic Publishing, Document Manipulation and Typography, Gaithersburg, Maryland*, Cambridge University Press, 1990.
17. E. Kuikka, P. Leinonen and M. Penttonen. Towards automating of document structure transformation. *DocEng '02*, 2002.
18. E. Kuikka and E. Nikunen. *Survey of software for structured text.* Technical report, Department of Computer Science and Applied Mathematics, University of Kuopio, 1998.
19. E. Kuikka and M. Penttonen. Transformation of structured documents. *Electronic Publishing, 8(4)*, 1995.
20. G. Lindén. *Structured document transformations*. PhD thesis, University of Helsinki, 1997. ISBN 951-45-7766-3.
21. S. A. Mamrak, C. S. O'Connel and J. Barnes. *Integrated Chameleon Architecture*. Prentice Hall, Englewood Cliffs, USA, 1994.
22. MID/Information Logistics Group GmbH. *MetaMorphosis Reference Manual*, 1995.
23. T. Noll and H. Vogler. The universality of higher-order tree transducers. *Theory of computing systems, 34(1)*, 2001
24. T. F. Payton. SSAGS: A syntax and semantics analysis and generation system. *Attribute Grammars. Definitions, Systems and Bibliography. Lecture Notes in Computer Science 323*, Springer-Verlag, Berlin, 1988.
25. V. Quint and I. Vatton. GRIF: An interactive system for structured document manipulation. *EP 86 - Proceedings of the International Conference on Text Processingand Document Manipulation, Nottingham, UK*, Cambridge University Press, 1986.
26. *SAX - Simple API for XML, version 2.0.2*, 2004. `http://www.saxproject.org/`.
27. D. Swierstra and H. Vogt. *Higher Order Attribute Grammars*. Technical report. Utrecht University, 1991.
28. W3C. *Extensible Markup Language (XML) 1.0 (Third edition), W3C Recommendation*, 2004. `http://www.w3.org/TR/REC-xml`.
29. W3C. *XSL Transformations XSLT Version 2.0, W3C Recommendation*, 1999. `http://www.w3.org/TR/xslt20/`.
30. A. Zhou, Q. Wang, Z. Guo, X. Gong, S., Zhengand H. Wu, J. Xiao, K. Yue and W. Fan. TREX: DTD-conforming XML to XML transformations. *SIGMOD 2003*, 2003.

Multimedia information extraction from HTML product catalogues

Martin Labský[1], Pavel Praks[2], Vojtěch Svátek[1], and Ondřej Šváb[1]

[1]Department of Information and Knowledge Engineering, University of Economics, Prague, W. Churchill Sq. 4, 130 67 Praha 3, Czech Republic
{labsky,svatek,xsvao06}@vse.cz
[2]Department of Applied Mathematics, VŠB – Technical University of Ostrava 17. listopadu 15, 708 33 Ostrava-Poruba, Czech Republic
pavel.praks@vsb.cz

Abstract. We describe a demo application of information extraction from company websites, focusing on bicycle product offers. A statistical approach (Hidden Markov Models) is used in combination with different ways of image classification, including latent semantic analysis of image collections. Ontological knowledge is used to group the extracted items into structured objects. The results are stored in an RDF repository and made available for structured search.

1 Introduction

Tools and techniques for *web information extraction* (WIE) have recently been recognised as one of key enablers for semantic web (SW) scaling. In our long-term project named *Rainbow*[3] we address several intertwined topics that we consider important for efficient 'WIE for SW' applications:

1. Exploitation of *multiple information modalities* available in web documents
2. Synergy of *learning* and reuse of *ontological information*
3. Automated acquisition and labelling of *training data* for extractor learning
4. Bridging between automated acquisition of SW data and their *usage*
5. Support for easy design of WIE applications *from components*.

In this paper, we focus on an ongoing demo application in the domain of *bicycle product offers*. Section 2 presents the core method: automated HTML annotation based on Hidden Markov Models. Section 3 extends the analysis of HTML code with that of images. Section 4 describes the composition of product offer instances with the help of a simple ontology. Section 5 outlines the architecture of the demo application and the subsequent usage of extracted data in an RDF repository. Finally, section 6 focuses on future work.

[3] http://rainbow.vse.cz

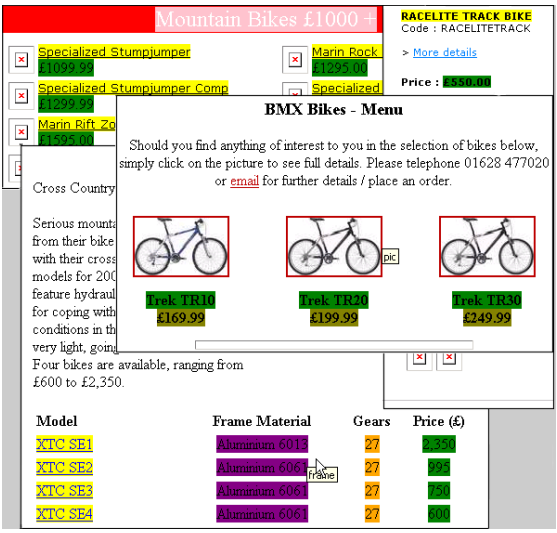


Fig. 1. Hand-annotated training data

2 Web Page Annotation Using HMMs

For extracting product entries from web catalogues, we built a Hidden Markov Model (HMM) tagger, which assigns a semantic tag to each token from a document. Tokens are either words, formatting tags or images. In our experiments, we evaluated the HMM performance on a diverse set of web pages, which come from different web sites and have heterogenous formattings.

We manually annotated a set of 100 HTML documents chosen from the Google Directory *Sports-Cycling-BikeShops-Europe-UK-England*. Each document contains from 1 to 50 bicycle offers, and each offer consists of at least the bicycle name and price. There are typically 3–4 documents from the same shop in the data. Annotations for 15 bicycle characteristics were made using SGML tags[4]. A sample annotated data is shown in Figure 1.

To represent web documents, we employed extensive pre-processing. Similarly to [7], we transform each document into XHTML and perform canonicalisation of XML entities[5]. Certain HTML tags and tag groups are replaced by their generalisations[6]. Since only words and images can be extracted, we dispose of mark-up blocks that do not directly contain words or images.

HMMs are probabilistic finite state machines, which represent text as a sequence of tokens. An HMM consists of *states* which generate tokens, and of

[4] The training data and a demo are available at `http://rainbow.vse.cz`.

[5] This step unifies different ways of writing the same characters in XML.

[6] Most tags are only represented using their names, disregarding any attributes. Often-occuring design patterns, such as add-to-basket buttons, are identified using several manually authored rules, and replaced by dedicated tokens.

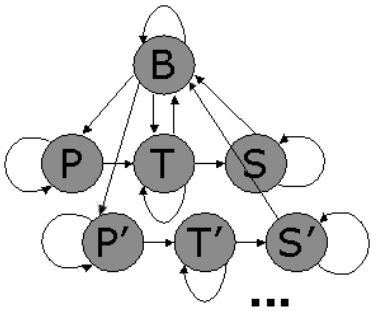


Fig. 2. HMM architecture

transitions between these states. States are associated with token generation probabilities, and transitions with transition probabilities. Both kinds of these probabilities are estimated from training data. For the purposes of information extraction, states are typically associated with semantic tags to be extracted. To annotate a document using a trained HMM, that document is assumed to have been generated by that HMM. The most probable state sequence is then found using the Viterbi algorithm [12].

The structure of our HMM is inspired by [6] and is sketched in Figure 2. Extracted slots are modelled using *target* states (denoted as T). Each target state is accompanied by two types of helper states responsible for representing the slot's characteristic context – the *prefix* and *suffix* states (P and S). Irrelevant tokens are modelled by a single *background* state (B). Contrary to [6] and [17], which use independent HMMs trained for each slot separately, we train a single composite HMM capable of extracting all slots at once. Our model thus contains multiple target, prefix and suffix states. This approach, also used in [1], captures the ordering relations between nearby slots (e.g. product image often follows its name). We experimented also with other HMM architectures, with results presented in [16].

3 Impact of Image Classification

For the purpose of extracting product images, we examined the impact of image information available to the HMM tagger. As a baseline approach, we measured the tagging performance when no image information was available for tagging. In this case, all images were represented by the same token and product pictures could only be distinguished based on the context in which they appeared.

In order to provide our tagger with more information, we built image classifiers to determine whether the extracted product is depicted in a particular image. We used the following features for classification: *image dimensions*, *similarity* to training product images, and whether there is *more than one occurrence* of the same image in the containing document.

3.1 Image dimensions

For our domain, we modelled images of bicycles using a 2-dimensional normal distribution, only estimated from positive training examples[7]. The dimensions x, y of a new image I are first evaluated using the estimated normal density N. The density value is then normalized to the interval (0,1) using the density's maximum value N_{max}.

$$Dim(I) := \frac{N(x,y)}{N_{max}} \tag{1}$$

An image I is then classified as Pos or Neg by comparing its $Dim(I)$ score to a threshold T_{Dim}. This threshold was estimated by minimizing the classification error rate on a separate heldout set of 150 images.

$$class(I) = \begin{cases} Pos\ if(Dim(I) \geq T_{Dim}), \\ Neg\ otherwise. \end{cases} \tag{2}$$

Within our document collection, image dimensions appeared to be the best single predictor with the error rate of 6.6%. However, this is mainly due to our collection being limited to relevant product catalogues only. When dealing with more heterogeneous data, features describing the actual image content will become necessary.

3.2 Image similarity

We experimented with a *latent semantic* approach to measuring image similarity, described in [10] and [11]. This kind of image similarity has been applied to image retrieval from collections, where the task often is to find the most similar image to a query. We used this image-to-image similarity measure $sim(I, J)$ to compute $sim_C(I)$, the similarity of an image I to a *collection* of images C. In our experiments, C contained the training bicycle pictures (positive examples only). To compute $sim_C(I)$, we used the K nearest neighbor approach and averaged the similarities of the K most similar images from the collection.

$$sim_C(I) = \frac{\sum_{K\ best\ images\ J \in C} sim(I,J)}{K} \tag{3}$$

Experimentaly, we set $K = 20$, since lower values of K lead to a decrease in the similarity's robustness[8] and higher values did not bring further improvement. To build a classifier, a similarity threshold T_{Sim} was estimated on a heldout set in the same way as for the dimension classifier above. The error rate of the classifier was 26.7% on our document collection.

[7] The positive examples comprise of *all* bicycle pictures found in the documents, not only those labeled as parts of bicycle offers. For information extraction, this increases the role of image context for correct tagging.

[8] With low values of K, $sim_C(I)$ became too sensitive to individual images J with misleading values of $sim(I, J)$.

3.3 Combined classifier

For the combined image classifier, we used the above described dimension score $Dim(I)$, similarity score $Sim(I)$ and a binary feature indicating whether the image occurs more than once in the document. We experimented with different classifiers available in the Weka[9] environment, and the best error rate[10] of 4.8% was achieved by the *multilayer perceptron algorithm*.

Results for all three classifiers are compared in Table 1. All results were measured using 10-fold cross-validation on a set of $1,507$ occurences of 999 unique images taken from our training documents. The first two algorithms used additional 150 heldout images to estimate their decision thresholds. The cross-validation splitting was done at the level of documents, so that all images from a single document were either used for training or for testing.

Table 1. Image classification results

	Dimension	Similarity	Combined
Error rate (%)	6.6	26.7	4.8

3.4 Using Image Information for Extraction

To improve extraction results, we need to communicate the image classifier's results to the HMM tagger. Currently we do this simply by substituting each image occurence in a document by its class. Since these binary decisions would leave little room for the HMM tagger to fix incorrect classifications, we adapted the above binary classifiers to classify into 3 classes: Pos, Neg, and Unk. In this way, the HMM tagger learns to classify the Pos and Neg classes correspondingly, and the tagging of the Unk class depends more strongly on the context.

To build the ternary versions of the dimension- and similarity-based classifiers, we introduced *costs* for the classifier's decisions. Each wrong decision was penalized by $C_{Miss} = 1$ and the cost of each Unk decision was $C_{Unk} \in (0, 1)$. We set C_{Unk} manually such that the classifier produced 5-10% of Unk decisions on the heldout set. While minimizing the sum of these costs on the heldout set, two thresholds were estimated for both the dimension- and similarity-based classifiers, delimiting their Neg, Unk and Pos decisions.

For the combined ternary classifier, we achieved the best results with a decision list shown in Table 2. The list combines image occurence count with the results of the dimension- and similarity-based ternary classifiers, denoted as $class^3_{Dim}$ and $class^3_{Sim}$ respectively.

We evaluated information extraction results with all three ternary classifiers and compared the results to the case where no image information was available.

[9] `http://www.cs.waikato.ac.nz/~ml`

[10] This error rate comes from 10-fold cross-validation *without* using heldout data.

Table 2. Decision list for the combined ternary classifier

Order	Rule
1	$class(I) = Neg$ if$(occurences(I) > 1)$
2	$class(I) = Pos$ if$(class^3_{Dim}(I) = Pos)$
3	$class(I) = Unk$ if$(class^3_{Dim}(I) = Unk)$
4	$class(I) = Unk$ if$(class^3_{Sim}(I) = Pos)$
5	$class(I) = Neg$

The new image information from the combined classifier lead to an increase of 19.1% points in picture precision and also to subtle improvements for other tags. Improvements in precision and recall for 3 chosen slots (product pictures, names and prices), measured on a per-token basis, are shown in Table 3 for all three classifiers.

Table 3. 10-fold cross-validation results for selected tags over 100 documents

Tag	Precision	Recall	F-measure	Precision	Recall	F-measure
	No image information			Image similarity		
Picture	67.8	87.1	76.2	78.5	87.3	82.7
Name	83.7	82.5	83.1	83.9	82.5	83.2
Price	83.7	94.4	88.8	84.0	94.4	88.9
	Image Dimensions			Combined		
Picture	85.6	88.4	87.0	86.9	89.1	88.0
Name	83.8	82.5	83.1	83.8	82.5	83.2
Price	84.0	94.4	88.9	84.0	94.4	88.9

4 Ontology-Based Instance Composition

Semantic web is not about isolated tagged items but about complex and interrelated entities; we thus need to group the labels produced by automated annotation into instances. We currently use a simple sequential algorithm that exploits constraints defined in a tiny *presentation ontology*[11] [9], which partly pertain to the generic domain (bike offers) and partly to the way of presenting information in web catalogues. Figure 3 shows an experimental presentation ontology containing the class 'Bike offer'. The utilized constraints are *uniqueness*, *multiplicity* and *optionality* of certain properties, the latter two indicated with the `*` and `?` symbols, respectively[12]. In addition, *'sticky' properties* (indicated with `!`) are distinguished: as soon as the value of sticky property is discovered

[11] Similar to 'extraction ontologies' used by Embley [5].

[12] Although not shown in the example, we can also use e.g. property value types or regular expressions.

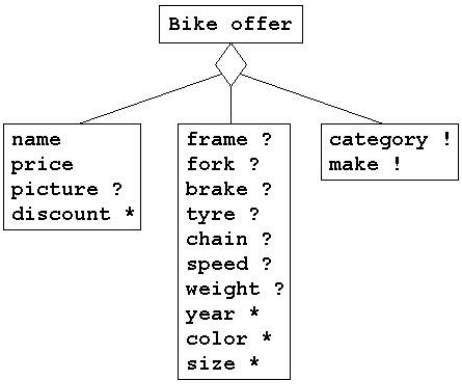


Fig. 3. Bicycle offer presentation ontology

on a page, it is filled to all objects extracted afterwards, until a new value is discovered for this property.

An annotated item is added to the currently assembled (bike offer) instance unless it would cause inconsistency; otherwise, the current instance is saved and a new instance created to accommodate this item and the following ones. Despite acceptable performance on error-free, hand-annotated training data, where the algorithm correctly groups about 90% of names and prices, this 'baseline' approach achieves very poor results on automatically annotated data: on average, less than 50% of corresponding annotations are grouped properly, often for trivial reasons. The most critical problems are connected with *missing or extra annotations*, *multiple different references* to a single slot, and with *transposed HTML tables*.

5 Result Transformation, Storage And Retrieval

All components developed within the *Rainbow* project are wrapped as web services. The WIE component itself is currently being called by a simple *control routine* (written in Java), which also optionally calls other analysis tools: in the bicycle application, we so far experimented with URL-based navigation over the website, extraction of the content of selected META tags, and extraction of 'company profile sentences' from free text[13]. The results are transformed to RDF (with respect to a 'bicycle-offer RDFS ontology') and stored in a *Sesame* [2] repository. An end-user search interface to this repository[14] is shown in Fig. 4. It relies on a collection of *query templates* expressed in SeRQL (the native query language of *Sesame*) and enables a simple form of navigational retrieval [16].

[13] These three approaches to website analysis, implemented independent of the bicycle demo application, are evaluated in [13].

[14] Available at `http://rainbow.vse.cz:8000/sesame`.

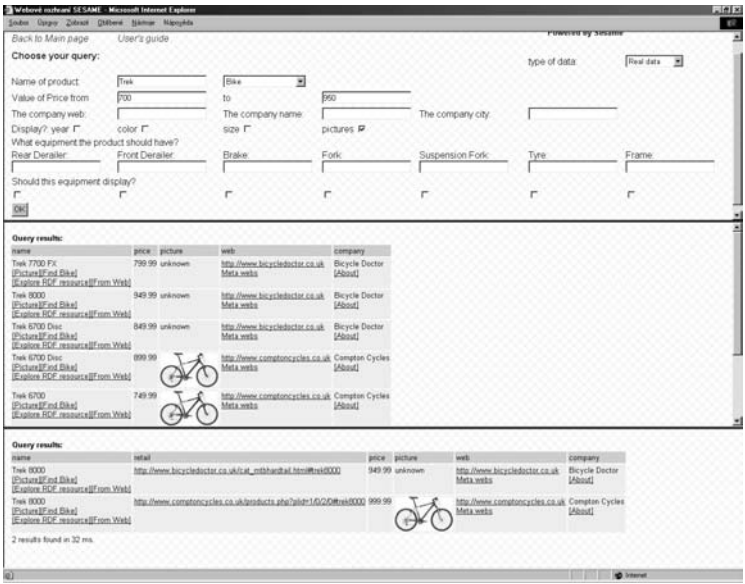


Fig. 4. End-user search interface

6 Future Work

Most urgently, we need to replace the 'toy' implementation of ontology-based *instance composition* with a version reasonably robust on automatically annotated data. For some of the *layout-oriented* problems mentioned in section 4, partial solutions recently suggested in IE research (e.g. [3, 5]) could be reused. We also consider introducing HMMs even to this phase of extraction; a modified version of Viterbi algorithm supporting domain constraints (such as those in our presentation ontology) has already been described in [1]. Another aspect worth investigation is the possibility of (semi-)automatic construction of presentation ontologies from the corresponding *domain ontologies*.

A critical bottleneck of ML-based IE methods (in particular of statistical ones) is the volume of *labelled training data* required. In our experiments with product catalogues, we noticed that the tagger often classifies most product entries correctly but misses a few product names that are very different from the training data. We developed a simple symbolic algorithm that identifies similar *structural patterns* in a document. For example, the HTML tag sequence `<td> <a> <font> <br/> </font> </a> </td>` with arbitrary words in between appears 34 times in one of our training documents: the tagger successfully annotated 28 product names contained in these patterns between `<font>` and `<br/>`, but missed the remaining 6. In such cases, we could collect the remaining product names and use them to enrich the model's training data. By learning novel product names from these 'easy' pages, the model will learn to also recognise

them in less structured documents[15]. We also plan to bootstrap the method with data picked from *public resources* related to product offering, following up with our earlier experiments with Open Directory headings and references [8].

Another important task is to replace hard-coded *control routines* with semi-automatically constructed, implementation-independent application models. A knowledge modelling framework has already been introduced for this purpose [14]; currently we examine the adaptability of a PSM-based semantic *web-service configuration* technique in connection with this framework [15].

Eventually, we plan to associate our efforts with the popular *Armadillo* project [3], with which we share most of our abovementioned research interests.

The research is partially supported by grant no.201/03/1318 of the Grant Agency of the Czech Republic, "Intelligent analysis of the WWW content and structure".

References

1. Borkar V., Deshmukh K., Sarawagi S.: Automatic segmentation of text into structured records. In: SIGMOD Conference, 2001.
2. Broekstra J., Kampman A., van Harmelen F.: Sesame: An Architecture for Storing and Querying RDF and RDF Schema. In: Proc. ISWC 2002, Springer LNCS no. 2342.
3. Ciravegna, F., Chapman, S., Dingli, A., Wilks, Y.: Learning to Harvest Information for the Semantic Web. In: ESWS-04, Heraklion, Springer LNCS 2004.
4. Dingli A., Ciravegna F., Guthrie D., Wilks Y.: Mining Web Sites Using Unsupervised Adaptive Information Extraction. In: EACL, 2003.
5. Embley, D.W., Tao, C., Liddle, S.W.: Automatically extracting ontologically specified data from HTML tables with unknown structure. In: ER2002, Tampere 2002, 322-337.
6. Freitag D., McCallum A.: Information extraction with HMMs and shrinkage. In: Proceedings of the AAAI-99 Workshop on Machine Learning for IE, 1999.
7. Grover C., McDonald S., Gearailt D., Karkaletsisy V., Farmakiotouy D., Samaritakisy G., Petasis G., Pazienza M., Vindigni M., Vichotz F., Wolinskiz F.: Multilingual XML-Based Named Entity Recognition for E-Retail Domains. In: LREC Conference, Las Palmas, 2002.
8. Kavalec, M., Svátek, V.: Information Extraction and Ontology Learning Guided by Web Directory. In: ECAI Workshop on NLP and ML for ontology engineering. Lyon 2002.
9. Labský, M., Svátek, V., Šváb, O.: Types and Roles of Ontologies in Web Information Extraction. In: ECML/PKDD04 Workshop on Knowledge Discovery and Ontologies, Pisa 2004.
10. Praks P., Dvorský J., Snášel V.: Latent semantic indexing for image retrieval systems. In: Proceedings of the SIAM Conference on Applied Linear Algebra (LA03), Williamsburg, USA, The College of William and Mary, 2003.
11. Praks P., Machala L., Snášel V.: Iris Recognition Using the SVD-Free Latent Semantic Indexing. In: MDM/KDD 2004 - Fifth International Workshop on Multimedia Data Mining, Seattle, USA, 2004.

[15] Similar bootstrapping strategies are shown in [4].

12. Rabiner, L.R.: A tutorial on hidden Markov models and selected applications in speech recognition. In: Proceedings of the IEEE, 77(2), 1989.
13. Svátek V., Berka, P., Kavalec, M., Kosek, J., Vávra, V.: Discovering Company Descriptions on the Web by Multiway Analysis. In: Intelligent Information Processing and Web Mining, IIPWM'03., Springer Verlag, 2003.
14. Svátek, V., Labský, M., Vacura, M.: Knowledge Modelling for Deductive Web Mining. In: Proc. EKAW 2004, Springer Verlag, LNCS, 2004.
15. Svátek, V., ten Teije, A., Vacura, M.: Web Service Composition for Deductive Web Mining: A Knowledge Modelling Approach. In: Proc. Znalosti 2005, VSB-TU Ostrava, to appear 2005.
16. Šváb, O., Labský, M., Svátek, V.: RDF-Based Retrieval of Information Extracted from Web Product Catalogues. In: SIGIR'04 Semantic Web Workshop, Sheffield.
17. Valarakos A., Sigletos G., Karkaletsis V., Paliouras G.: A Methodology for Semantically Annotating a Corpus Using a Domain Ontology and Machine Learning. In: RANLP Conference, Borovets, 2003.

Text mining tool for ontology engineering based on use of product taxonomy and web directory

Jan Nemrava and Vojtěch Svátek

Department of Information and Knowledge Engineering,
University of Economics, Prague, W. Churchill Sq. 4, 130 67 Praha 3, Czech Republic
{nemrava, svatek}@vse.cz

Abstract. This paper presents our attempt to build a text mining tool for collecting specific words – verbs in our case – that usually occur together with particular product category as support for ontology designers. As the ontologies are headstone for the success of the semantic web, our effort is focused on building small and specialized ontologies concerning one product category and describing its frequent relations in common text. We describe the way we use web directories to obtain suitable information about the products from UNSPSC taxonomy and we propose the method how the extracted information could be further processed.

1 Introduction

Information Extraction (IE) and Ontology (OL) learning are frequently discussed issues in the field of Semantic Web. The problems of information extraction using hand-crafted patterns have been addressed in many papers and it is obvious that the most promising way is automated or semi-automated ontology-based extraction of information. Since the results of IE from rigidly structured and semi-structured texts are already quite satisfying, the problems remain in field of unstructured free text processing. Large amount of knowledge-sparse text with full linguistic analysis would be too demanding. Shallow linguistic methods typically rely on POS tagging and/or shallow parsing. In our work we focus on finding verbs as simple POS category (in [4] called "indicator terms") that usually occur with some product selected from The United Nations Standard Products and Services Code[1] (UNSPSC) *product catalogue* so that we can:

- construct *ontologies* containing relations labeled with extracted verbs
- use these verbs for *extracting* further product categories from web pages

Web directory hierarchies (e.g. DMOZ[2]) are sometimes mistaken for ontologies; however, as already observed by Uschold [11], they are rarely valid taxonomies. It is easy to see that subheadings are often not specializations of headings; some of them are even not concepts (names of entities) but properties that implicitly restrict the

[1] http://www.unspsc.org
[2] http://www.dmoz.org

© K. Richta, V. Snášel, J. Pokorný (Eds.): Dateso 2005, pp. 94–102, ISBN 80-01-03204-3.
VŠB – Technical University of Ostrava, Dept. of Computer Science, 2005.

extension of a preceding concept in the hierarchy. Consider for example .../Industries/Construction and Maintenance/Materials and Supplies/ /Masonry_and_Stone/Natural Stone/International Sources/Mexico.
Semantic interpretation of a sample of DMOZ paths revealed that:

– Terms in the headings belong to quite a small set of classes, such as 'Object' (i.e. product such as 'Car'), 'Subject' (e.g. 'Manufacturer' or 'Dealer'), 'Domain' (of competence of company, such as 'Transport' or 'Insurance'), 'Location' (e.g. 'Mexico') etc.
– Surface 'parent-child' arrangement of headings belonging to particular classes corresponds (with some ambiguity) to 'deep' ontological relations.

The idea of closed loop between IE and OL bootstrapped with web directory headings was first formulated in [4]: by matching headings (mostly corresponding to generic names of products, services, or domains of competence of companies) with full texts of pages, we can obtain content of these fulltext and use it for data extraction.

The reason why we use UNSPSC is that we would like to join this taxonomy and list of products with content of company websites to gain valuable information about verbs that usually occur in one sentence with some product category from the taxonomy. We build a tool that collects these verbs from given web pages. Presented text mining tool is based on combination of catalogue and fulltext search engine. Our approach is exploits redundancy of data on large data repositories like World Wide Web. We are exploiting the knowledge stored in hand classified web directories like DMOZ and we use their ability to provide web sites relevant to term we have chosen. The problem that had been already discussed in [10] is that the first website page does mostly does not contain much or even any text. When it does, it hardly ever describes the product or the offered services. This led us to use the fulltext search engines with restriction to particular website to ensure that we discover all term occurrences in content of whole company's website. As UNSPSC is freely available in standard ontology format from Protégé[3] website, it contains 16.000 unique products and has unambiguous structure, it is suitable for use in this field.

In this paper, we first describe the reason why UNSPSC was chosen, and why we use directories as source for our data. In next section we introduce our method to identify verbs related to products and in third section we describe experiments and the results. At the end of the paper related work and our future plans are discussed.

2 Proposed method description

2.1 Finding UNSPSC leaves in DMOZ directory

As suggested in [4] use of UNSPSC could be good technique how to overcome web directories problem with their structure and overlapping categories which describe more products. UNSPSC contain 16 000 specialized terms each describing particular

[3] http://protege.stanford.edu/

product category which can hardly be further divided. On the other hand this raise problem that UNSPSC tree leaves (product categories) varies from the directory headings in commonly used directory structure including DMOZ and Google directory. At current time there aren't any tools to automate the process of assigning right UNSPSC category to relevant DMOZ category so it must be done manually by choosing product from taxonomy and then finding appropriate category in directory. There are either a lot of categories describing our term or none. In the first case we focus on Business branch where we expect that the manufacturers and the company offering our products will be stated. The latter case – where no category is found – is worse and we have to find similar category, or find similar product. These two issues disallow this part of our work to be done automatically. In our test we found 7 nodes corresponding to the same number of products from UNSPSC from "Material handling" field.

2.2 Obtaining verbs from relevant web sites

We take advantage of human-classified web page links stored in web directories. As stated above their structure is not always valid taxonomy. Subheadings are often not specialization of headings; some of them are even not concepts (names of entities) but properties that implicitly restrict the extension of a preceding concept in the hierarchy. This is reason why we make use of UNSPSC classification in our paper. We would like to obtain so called „indicator verbs" that characterize particular term (product category in our case) in UNSPSC. Particular terms will be then generalized and may mine verbs that are indicative for the upper level of these terms. The trial was only made on one category and several terms, which limits the representativeness of results. Only several common verbs were obtained and they had to be classified manually, as we don't have any other categories to be compared with results from this. Next paragraph describes the text mining tool that collects data from selected directory category.

Table 1. Task sequence decomposition

1) Input: URL of DMOZ directory containing companies that manufacture desired product. Output: List of URL of companies. 2) Input: URL of company website Output: List of web pages containing the target term. 3) Input: Web page containing the term Output: File with extracted sentences containing the term 4) Input: Sentence with term. Output: Extracted verb.

Table 1 depicts sub-tasks of the tool. The input data for this tool are the *URL of directory in DMOZ* containing links relevant to chosen term and the *product category* chosen from UNSPSC. When we have chosen the right category the script can be run. The first part uses *link extractor* to obtain all company's web sites URLs. The list of extracted links is stored in file for further processing. Every URL from the list is then inserted into Yahoo Search Engine with the term we are currently exploiting and the parameter "site" is added. This ensures that the particular term is only searched on the selected web site. This process is repeated until all URLs from the list have been processed. We only store first 10 links from every domain, but it is only matter of setting of script and here we see a possibility of extracting more data. Up to 100 links from every company URL can be stored.

Now we have several hundreds links (depending on number of links on list) to sites where our desired term occurs in the page full text. Next part task is to extract every sentence from this set of links where the terms occur. The task is carried out by means of regular expressions and finding occurrence of the term in set of documents. As the sentences are discovered and saved into file we need to carry out some syntactical analysis to discriminate verbs from other lexical units. It is done by Adwait Ratnaparkhi's Java based Maximum Entropy POS Tagger (MXPOST) [6]. The extracted verbs are then compared with each other to find similar verbs, and number of occurrences is counted. Using WordNet[4] database and its ability to discover word stem from any word form we assure that neither text parser nor MxPost made mistake by during assigning verbs. If mistakes were made WordNet discovers them and it also provides lemma for each word inserted, which makes storing of verbs much easier. The Table 2 lists first 10 verbs given by our script for term "hoists" where word in the 9th row (i.e. "products") was incorrectly labeled by MxPost tagger as verb.

[4] http://wordnet.princeton.edu/

Table 2. Example of extracted verbs and their lemmas

word	lemma	next occurrences
include	*include*	include, includes, includes, includes, includes, includes, include, include, include, include, included, included, include, include, include, include, include, included, include
announced	*announce*	announced, announced, announced, announced, announce, announce, announced, announced, announced, announced
are	*be*	are, are
providing	*provide*	providing, providing, providing, providing, providing, providing, providing
feature	*feature*	featured, features, features, features, features, features, features, features, features, feature, feature
following	*follow*	following, following, following, following, following, following, following
leading	*lead*	leading, leading, leading, leading, leading, leading, leading, leading, leading
products	products is not a verb	products, products, products, products, products, products
including	*include*	including, including, including, including, including, including

We store all extracted verbs as matrix in relational database, where the discovered verbs represents rows and the desired terms represents columns. The intersection of row and column is the number of verb occurrences in all web pages of the directory category. We can use these sets of verbs and number of their occurrences for further examination on how the verbs characterize some broader term and if a small ontology could be build for every term. The goal of effort is to build separate lists of verbs such that some only characterize specific product types, while some other characterize whole product areas. We started with node *Handling materials and products* from this category.

3 Experiments

In extracting selected information we currently face the problem of how to set proper amount of web sites to be analyzed and what is the right amount of words taken in surroundings of the word. As stated above in this particular experiment, 10 web pages from one company website and one sentence from each web page containing examined word is taken. Our research shown that this can be sufficient amount of data for extraction of verbs for some common product types, but other product types from the

same category of UNSPSC suffer from lack of data for extraction because they don't have appropriate category (node) in DMOZ.

There are two possible ways to overcome this problem. The first one is to concede the given constrains and allow our script to crawl more pages from one website and also allow to extract more sentences from one page. The second approach we have on mind is take advantage of some news resources like Google News service as there might appear verbs that characterize some product category. But from previous experience [3] we know, that the language used in non-official texts could contain misguiding verbs that have loose connection to term's denotation because of author's will to attract the readers attention by lot of ambiguous verbs. What is necessary is to restrict the domain of search, e.g. technical innovation, or technical news.

Using the above described tool we have built a database containing 303 verbs for 7 product categories from *handling material* category. These are only words that have appropriate category in DMOZ and therefore our approach could be used for their extraction. These verbs occurred 7300 times near the selected terms.

Our goal is to find some method that would enable us to categorize verbs as either:

- *common* for most products.
- *characterizing* one branch of products
- *specific* for small group of products, or even only *one product*.

Even from seven product categories – as expected – some verbs are obvious to be entirely neutral and do not characterize the products at all. According to three methods described later, verbs *be, have, provide* and *use* are common for all sentences describing any product. Then we have verbs describing activities connected with manufacturing of any types of products e.g. *design, require, offer, make, contact, manufacture, develop, supply,* etc. More specific for our branch might be verbs describing activities related to manipulating with material. They are *handle, lift, install* and *move*.

We experimented with three different measures that could separate specific verbs from more general ones. First and second are normalizations of frequencies to eliminate the influence of very frequent verbs. Normalization based on proportions of product categories in collection is the first, **Croft's normalization** using elimination of high-frequency terms with a specific constant is the second and **TF/IDF** [8] which relies on indirect relation between verb occurrences in its importance for product category is the last. We also tried **Lift measure [**2] but it didn't provide satisfactory results for aggregate values. We plan to use it for individual product category in future as it measures how many times more often occurs one verb with one term together than expected if they where statistically independent.

We tried these three methods to class verbs to their corresponding groups of verbs. All methods provided quite similar results. The first is *normalization* described by formula (1), where *Fij* is normalized frequency, f_{ij} is the frequency of verb *j* in product category *i*, V_{tj} is sum of all occurrences of product category *i* in collection and V is total number of collected verbs. Then V_{tj} / V represents how many per cent has product category *i* in collection. We recalculate whole matrix to get numbers ranging from 0 to 43 representing the normalized frequencies showing that the verbs with high value (30-43 in our case) are independent on the product category and thus they can be considered as common one. Verbs with values from 10 to 30 are not so often

and they could be used as branch descriptors. The rest are with frequency lower than 10 are out of our interest for this moment.

$$F_{ij} = f_{ij} * (V_{tj} / V) . \quad (1)$$

Croft's normalization (2) moderates the effect of high-frequency verbs, where cf_{ij} is Croft's normalized frequency, f_{ij} is the frequency of verb j in product category i, m_i is the maximum frequency of any verb in product category i, K is a constant between 0 and 1 that is adjusted for the collection. K should be set to higher value (higher than 0.5) for collections with short documents. We used 0,3 as there are no different between 0.3 and 0.5 in our table. With this formula we get sum values for every verb ranging from 2.1 (7 product category × 0.3 for zero occurrences) for no occurrences of verb in our database to 8.58 for the most often verbs. Verbs with number above 5 normalized occurrences are significant for us as the common indicator while verbs between 3 and 5 normalized occurrences could be taken as the products representing verbs. The rest, with 3 and lower occurrences is for us as in previous method uninteresting.

$$cf = K + (1 - K) * f_{ij} / m_{ij} . \quad (2)$$

TF/IDF (term frequency / inverse document frequency) (3), where w_{ij} is a weight of verb in product category i, f_{ij} is the frequency of verb j in product category i, N is number of all verbs in collection and n is sum of verb j occurring in all product categories. TF/IDF is technique that gives verb a high rank in a document if the verb appears frequently in a document or the verb does not appear frequently in other product categories. In other words a verb that occurs in a few product categories is likely to be a better discriminator than a verb that appears in most or all categories. As a result in this test we got values from 0 to 1350. Where as usual, the highest values between 1000 and 1350 are verbs that occur independently on selected product category and we consider them as common verbs. We are much more interested in verbs with value starting around 300 and ending at 1000. As stated above, these could be used as identifiers of the product category.

$$w_{ij} = f_{ij} * \log_2(N / n) \quad (3)$$

In our trial we only examined 7 product categories from one UNSPSC node and hence we are not able to classify verbs into four categories as we suggested in part 1. We only classified them on common and specialized verbs. The first 15 results with values from each of described method are shown in Table 3.

The reason why this approach cannot be automatically run are mainly the non-corresponding items from product taxonomy to categories in widely-spread product catalogues. Our plans and intentions for the development of this tool are stated in future work section.

Table 3. Comparison of three methods

	lemma	Per cent	lemma	croft	lemma	TFIDF
1	have	43,01	have	8,58	have	1 318,40
2	provide	40,38	provide	7,41	provide	1 164,76
3	design	39,36	design	7,14	design	1 119,10
4	use	37,29	use	6,38	use	1 028,17
5	lift	26,47	require	5,32	require	802,81
6	require	26,43	handle	4,70	lift	703,11
7	handle	19,81	lift	4,70	handle	676,10
8	mount	17,75	offer	4,68	offer	648,62
9	operate	17,66	allow	4,31	allow	596,96
10	truck	17,61	include	4,30	contact	587,38
11	allow	17,25	please	4,29	move	582,57
12	contact	16,37	make	4,18	please	582,57
13	offer	15,99	contact	4,15	include	572,89
14	meet	15,91	need	4,06	meet	538,52
15	include	15,49	install	4,06	make	538,52

4 Related Work

The idea of combination information extraction with ontology learning has been described by Maedche in [5]. The idea of using identified words to extract more words was in [7] called *mutual bootstrapping*. This paper follows up with work [3] as it brought to this field use of universal product taxonomy and web directories and firstly suggested UNSPSC as possible way to obtain relevant data from given branch for a given product category. While Brin [1] uses fulltext search engines to obtain data from arbitrary sources we only use search engines for obtaining full text from the websites we have previously identified by another method, because our data are less structured and can be mistaken easily by ambiguous meanings of terms.

5 Future Work

As described in this paper, there are currently some limitations of this approach; they are mainly caused by lack of data to be mined from websites for some specialized terms. We proposed some techniques how to overcome these limitations. One of them is relaxing restrictions of fulltext search engines and the second is searching in all subdirectories for given terms in all whole DMOZ branch tree structure. All our plans for future stem from our effort to obtain as much data as possible and also better automation of whole process. We recently discovered a tool that could help us to use fulltext search on selected nodes from DMOZ web directory. As soon as we obtain verbs for more branches we could try to classify the verbs into four categories as

proposed in section 3 and use them for creating ontologies with relations labeled with extracted verbs.

6 Acknowledgements

The authors would like to thank to Martin Labský and Martin Kavalec for their comments and help.

The research has been partially supported by the grant no. 201/03/1318 of the Grant Agency of the Czech Republic „Intelligent analysis of the WWW content and structure“.

7 References

1. Brin,S.: Extracting Patterns and Relations from the World Wide Web. In WebDB Workshop at EDBT'98
2. Brin S, Motwani R., Ullman J, Tsur S.: Dynamic itemset counting and implication rules for market basket data. In SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, pages 255-264, Tucson, Arizona, USA, May 1997.
3. Kavalec M., Maedche A., Svátek V.: Discovery of Lexical Entries for Non-Taxonomic Relations in Ontology Learning. In: SOFSEM – Theory and Practice of Computer Science, Springer LNCS 2932, 2004
4. Kavalec M., Svátek V.: Information Extraction and Ontology Learning Guided by Web Directory. In: ECAI Workshop on NLP and ML for Ontology engineering (OLT-02). Lyon, 2002.
5. Maedche A., Neumann G., Staab S.: Bootstrapping an Ontology Based Information Extraction System. Studies in Fuzziness and Soft Computing, editor Kacprzyk J., Intelligent exploration of the web, Springer 2002/01/01
6. Ratnaparkhi A.: Adwait Ratnaparkhi's Research Interests, [online], http://www.cis.upenn.edu/~adwait/statnlp.html
7. Riloff E., Jones R..: Learning Dictionaries for Information Extraction by Mulit-Level Bootstrapping, Proceedings of the Sixteenth National Conference on Artificial Intelligence, P 474-479, 1999
8. Salton G., Buckley C.: Term-weighting approaches in automatic text retrieval. Information Processing and Management, 24 (5):513--523, 1988.
9. Salton, G. and Buckley, C. (1988d). : Term-weighting approaches in automatic text retrieval. Information Processing and Management, 24:513-523.
10. Svátek V., Berka P., Kavalec M., Kosek J., Vávra V.: Discovering company descriptions on the web by multiway analysis. In: New Trends in Intelligent Information Processing and Web Mining (IIPWM'03), Zakopane 2003. Springer-Verlag, 'Advances in Soft Computing' series, 2003
11. Uschold M. , Jasper R.: *A Framework for Understanding and Classifying Ontology Applications*. In: Proceedings of the IJCAI-99 Workshop on Ontologies and Problem-Solving Methods: Lessons Learned and Future Trends.

Relational Data Mining and GUHA

Tomáš Karban

Department of Software Engineering, Faculty of Mathematics and Physics,
Charles University, Malostranské nám. 25, 118 00 Praha 1, Czech Republic
tomas.karban@matfyz.cz

Abstract. This paper presents an extension of GUHA method for relational data mining of association rules. Because ILP methods are well established in the area of relational data mining, a feature comparison with GUHA is presented. Both methods suffer from the explosion of the hypotheses space. This paper shows heuristic approach for GUHA method to deal with it, as well as other methods helping with the relational data mining experience.

1 Introduction

Most data mining methods have been developed for data in the traditional single-table form: rows represent observations (objects) and columns represent variables (properties). However, real-world data are usually stored in relational databases and consist of many related tables. Data preparation methods then convert data stored in an arbitrary form to a single-table representation (e.g. through joins and aggregation). This step simplifies the data inherently and it is manually driven, so its success depends on the creativity of a KDD person.

Relational data mining studies methods for KDD that can use more than one data table directly, without transforming the data into a single table first and then looking for patterns in such an engineered table. In many scenarios, leaving the data in relational form can save a lot of information that can be later expressed in the form of relational patterns. Single-table patterns are simply less expressive. Relational data mining techniques have been mainly developed within the area of inductive logic programming (ILP). ILP was initially concerned with the synthesis of logic programs from examples and background knowledge. Recent development has expanded ILP to consider more data mining tasks, such as classification or association analysis.

On the other hand, GUHA method is an approach of generating and verifying the hypotheses. Deep theoretical background is in [1]. For example, GUHA procedure 4ft-Miner mines for association rules from a single table, while other GUHA procedures (see [4, 7, 8]) mine for other types of patterns. Its main principle is to generate all possible hypotheses based on user task setting, verify them and output the valid ones. In a typical effective implementation, it loads the database into bit strings (i.e. bitmap indexes), and verifies the hypotheses by processing the bit strings.

We can quite naturally extend this approach, so it allows us to search for association rules generalized to more than one data table. We can expect to find patterns that are more complex. Unfortunately, the hypotheses space grows rapidly and the results (valid hypotheses) are more numerous. A new system called Rel-Miner covering methods and ideas presented in this paper is under development.

This paper presents the extension of GUHA method for relational data mining and compares this approach with ILP, which is well established in this area. In the section 2 of this paper, we present the GUHA method, starting from the single-table case and

© K. Richta, V. Snášel, J. Pokorný (Eds.): Dateso 2005, pp. 103–112, ISBN 80-01-03204-3.
VŠB – Technical University of Ostrava, Dept. of Computer Science, 2005.

extending it to relational case. The section 3 covers the ILP approach (mainly relational association rules), compares it to the GUHA and puts both concepts to a universal frame. In the fourth section, we provide more insight to heuristics and optimizations for relational version of GUHA. Section 5 is the conclusion.

2 GUHA Method

2.1 Single-Table Association Rules

An association rule is commonly understood as an expression of the form of X→Y, where X and Y are sets of items. The intuitive meaning is that transactions (e.g. supermarket baskets) containing set X of items tend to contain set Y of items as well. Two measures of intensity of association rule are used, *confidence* and *support*. The goal is to find all association rules of the form X→Y such that the support and confidence of X→Y are above the user-defined thresholds *minsup* and *minconf*. The basic algorithm for mining of association rules is APRIORI [2].

The papers [6, 7] draw an attention to an alternative approach for mining association rules based on representation of each possible value of each attribute by a single string of bits. The association rules have the form $\varphi \approx \psi$ and it is possible to mine for conditional association rules in the form $\varphi \approx \psi / \chi$ as well. Here φ, ψ and χ are conjunctions of boolean attributes automatically derived from many-valued attributes in various ways (called *antecedent*, *succedent* and *condition* respectively). The symbol $\approx$ is called a 4ft-quantifier. The association rule $\varphi \approx \psi$ means that boolean attributes φ and ψ are associated in the sense of the 4ft-quantifier $\approx$. A conditional association rule $\varphi \approx \psi / \chi$ means that φ and ψ are associated (in the sense of $\approx$) if the condition given by χ is satisfied (i.e. associated on a subset of data specified by χ).

$\mathcal{M}$	ψ	$\neg\psi$	
φ	a	b	r
$\neg\varphi$	c	d	s
	k	l	n

Fig. 1. A contingency table of the association rule $\varphi \approx \psi$ in data matrix $\mathcal{M}$

The 4ft-quantifier is formally a boolean condition concerning the four-fold contingency table with frequencies a, b, c, d. For every quantifier, we can also use a *natural value of hypothesis*. It is a real number based on the same expression. See the example below.

Among usual 4ft-quantifiers there is a founded implication, which is very similar to "classic" association rule meaning with parameters *minsup* and *minconf* (in the sense [2]). We define it by the following expression over the four-fold table:

$$\frac{a}{a+b} \geq p \wedge a \geq Base .$$

There are parameters $0 < p \leq 1$ and $Base \geq 0$. You can interpret a hypothesis $\varphi \Rightarrow_{p;Base} \psi$ as "φ implies ψ on the level of 100p percent and there are at least *Base* objects satisfying both φ and ψ". The natural value of this hypothesis is the expression $\frac{a}{a+b}$.

Other quantifiers describe double founded implication, founded equivalence, above average occurrence or tests of statistical hypotheses (chi-square test of independence, lower critical implication, etc.). You can find more information about this approach in [6, 7]. The project LISp-Miner [4, 5] (namely the 4ft-Miner procedure) is the implementation of this method.

Let us give an example of the association rule with founded implication quantifier:

$$\text{Smoking(> 20 cigs.)} \;\&\; \text{PhysicalActivity(high)} \Rightarrow_{85\%} \text{RespirationTroubles(yes)}$$

We can read in plain English that 85% of observed patients, who smoke daily more than 20 cigarettes and have a high physical activity, suffer from respiration troubles.

The basic principle behind an effective implementation is usage of the bit strings. We start with a bit string of heavy smokers and bit string of people with high physical activity (these are created directly from database values). We construct the antecedent by ANDing the two bit strings (bit by bit). Note that all bit strings have the same length, which is the total number of patients (objects of this particular database). The succedent is simply a bit string of people with respiration troubles. We compute the frequencies of the four-fold contingency table as the number of bits 1 in the following bit strings: (antecedent & succedent), (antecedent & ¬succedent), (¬antecedent & succedent) and finally (¬antecedent & ¬succedent).

Note that if the database contains objects with missing information (i.e. some null values), we must also maintain a bit string of these null values (as if this was another category) and use them for computation. More details about missing information handling are out of scope of this paper.

In general, we create bit strings out of every category (i.e. possible value) of every attribute. Then we combine some of these bit strings by very fast bitwise operations (namely AND, OR and NOT) into bit strings of antecedent, succedent and possibly condition. After that, we compute a contingency table by counting the number of bits with value 1 and check the value of quantifier. We do this for every hypothesis in the whole hypotheses space specified by the user as a task setting.

2.2 Relational Association Rules

At the beginning, it is important to note that even when we have more than one data table, we still consider one data table as "the main" which stores the basic information about every single object (the same as before). Additionally, we take additional tables with the 1:N relation to the main table. This situation is in database world usually called "master-detail".

Let us start with a motivation example (see also [6]). We have a database of bank clients. The first table stores basic client information like age, gender, marital status, number of children, and the quality (status) of a running loan. The second table stores money transactions on clients' accounts. Every transaction has a source and destination account number and amount.

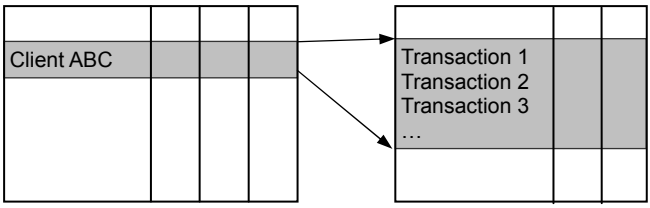


Fig. 2. A simple database schema with 1:N relation

The goal is to get relational association rules that will tell us interesting patterns about the loan quality with respect to both mentioned data tables. The target could be:

MaritalStatus(divorced) & Children(3) & SingleIncome(yes) & AvgIncome(< 1500) $\Rightarrow_{76\%}$ LoanQuality(bad)

In plain English, observed divorced people with single income less than $1500 a month and three children have the probability of 76% of having bad quality loan (troubles with repayments). Here, we used the attributes MaritalStatus and Children directly from master data table. Attributes SingleIncome and AvgIncome were derived from the transactions in the following way:

TransactionAmount(> 500) $\Rightarrow_{93\%}$ SourceAccount(acc345) / Client(ABC)
AVG(SELECT SUM(TransactionAmount) WHERE (TransactionAmount > 0) GROUP BY YearMonth)

We call the attributes derived from detail tables *virtual*. The former attribute is in the form of a founded implication on transaction data, the account number acc345 is a fixed constant for the client ABC. The attribute value for this specific client is 93%. This value is then "virtually added" to the master data table and can be discretized (e.g. yes = more than 90%) to a boolean attribute SingleIncome.

The attribute AvgIncome is a kind of SQL aggregation; for every client it counts the monthly average of the sum of all income transactions.

2.3 Adaptation for Relational Data Mining

In the single-table case, the whole KDD process can be described very easily and precisely by the CRISP-DM methodology [3]. It has separate steps of data preparation and modeling (i.e. data mining). In the data preparation step, data must be discretized before they are used in modeling step. This stays the same even for the relational case; unfortunately, the discretization is needed in the modeling step as well. In the previous example, we have seen the conversion of the strength of implication (93%) into a boolean value, as well as making some categories out of the average income.

We may want the user to prepare such discretization in advance, but the number of such virtual attributes can be very large. From this point of view, we suggest a semi-interactive mining that lets the data mining engine suggest the discretization automatically (based on the distribution of values) with the possibility that the user will change (and fix for future use) some discretization later as he/she sees the attribute used in results.

In other words, the before-distinct steps of data preparation, modeling and evaluation blend. That may introduce new challenges and problems in the whole context of KDD.

At this point, note that the attribute SingleIncome in the previous example was created as an association rule effective on the set of transactions that belong to the specified client. In theory, we can run the search for the association rules recursively, and use their validity as a new attribute at higher level. In practice, we doubt that it is useful to do it for depth more than one, as it leads to result hypotheses, which can hardly be expressed in plain English and meaningfully used in practice. Furthermore, the general recursive approach would be extremely computationally intensive. That means we practically accept only star-schema of database, with one table being master and the rest being detail tables.

2.4 Types of virtual attributes

At this point, we can summarize all methods and possibilities, how we can construct a virtual attribute. We can define a set of aggregation functions that can be applied to attributes in a detail table. The generator can try to run all aggregation functions on all attributes. However, in practice there are semantic limits on usage of certain aggregations on particular attributes. For example, there is no point computing the sum of values that denote the current amount of goods in a warehouse in every month. Unfortunately, the user must specify these limits. Among the usual aggregates, we may consider sum, average value, mean value, standard deviation, minimum, maximum, first and third quartile, count and many others. More advanced aggregates may be integration or linear regression (e.g. for time series). It may prove useful to allow the user to provide user-defined aggregate functions as well.

In the previous example, we have seen the usage of an arbitrary association rule as a virtual attribute (see also [6]). It can describe (among others) implication, equivalence, statistical independence and even relations on general contingency tables [8].

The last category of virtual attributes is existential quantifiers. Within this category, it is necessary to use constants bound to a single object, or global constants specified by user. Note that we have seen (in the example above) the literal SourceAccount(acc345), which used a constant acc345 bound to a single bank client (other clients may have other sources of income). The existential quantifier can be used for example to define a boolean attribute PaysTaxes, such that there exists a transaction with target account number equal to government account (which is a constant defined by user).

3 Other Methods of Relational Data Mining

3.1 Inductive Logic Programming

The most widely adopted method for relational data mining is inductive logic programming (ILP). A summary of the methods and taxonomy is given in [11, 16]. In [15] the author notes the following. While approaches such as ILP inherently suffer from a large search space, integrating aggregation operators further increases the hypothesis space. Moreover, the search space is less well-behaved because of the problem of non-monotonicity, which renders traditional pruning methods in ILP inapplicable. Clearly, we must give up the idea of performing a systematic search for a good hypothesis and the search must be more heuristic.

As we are proposing the extension of GUHA method, which mines for association rules, we focus on the part of ILP that concerns relational association rules as well.

3.2 WARMR

The paper [12] presents an algorithm WARMR, which is an extension to APRIORI algorithm [2] for mining of relational association rules. You can find the implementation of WARMR in Aleph [17] or ACE [18].

APRIORI algorithm is a levelwise search through the lattice of itemsets. It works in two phases. First, we search for so-called frequent itemsets; we examine every transaction and count the number of transactions containing a given itemset (as a subset). In the second phase, we create association rules from frequent itemsets that meet minimum specified confidence and support (see paragraph 2.1).

This two-phase approach is the same for WARMR. The notion of itemsets is generalized to *atomsets*, which are sets of logical atoms constructed from database relations. We count the number of examples that are covered by a given atomset. This is implemented in Prolog such that an atomset is converted to an existentially qualified conjunction and then run as a Prolog query. If it succeeds, the atomset covers the specified example (a counter is increased); if it fails, the atomset does not cover the example.

Now having the set of frequent k-atomsets (atomsets containing exactly k atoms), we construct candidate (k+1)-atomsets. This is analogous to APRIORI, although new step is introduced to prune atomsets. This extra step involves running a theorem prover to verify whether the atomsets are contradictory or redundant with respect to a user-specified clausal theory. This theory provides the user with a powerful tool to specify taxonomies, mutual exclusion relations, and other types of background information known to hold in the database. It contributes to the efficiency of the algorithm as well as to the quality of the output.

The example of association rule over multiple relations can be the following:

likes(KID, dogs) & has(KID, fish) ⇒ prefers(KID, dogs, fish); conf. 65%, supp. 20%

This association rule states that 65% of kids, who like dogs and have fish, prefer dogs to fish; furthermore, 20% of all kids like dogs, have fish and prefer dogs to fish. There can be variables instead of constants in association rules as well, like:

likes(KID, dogs) & has(KID, A) ⇒ prefers(KID, dogs, A)

3.3 Comparison of GUHA and WARMR

We can separate all algorithms and methods of relational data mining to two basic categories, according to the first part of [14]. The methods using aggregation to bring the information from detail tables to the master table belong to the first category. These methods use the aggregates as a description of the whole set of data that corresponds to the examined object from the master table.

The second category comprises of algorithms that handle sets by looking at properties of their individual elements. For example, a part of the pattern can be of the form "there exists an x in the set such that P(x) holds", where P can be a relatively complicated condition. Most ILP systems follow this approach.

We can call methods of the first category *aggregating methods* and methods of the second category *selective methods*. To make this distinction clearer, see the two simple concepts in the following the example:

- people whose average account balance is above $10.000
- people who pay telephone by encashment
  (i.e. there exists a certain type of transaction)

Clearly, WARMR algorithm is using an existentially qualified conjunction to count the number of examples covered by a given atomset. That puts it into the category of selective methods. On the other hand, ideas presented in this paper as an extension to GUHA method cover both these categories. Aggregations create virtual attributes easily and are very suitable for subsequent processing. Furthermore, using association rules as a basis for virtual attributes is a more advanced concept than simple aggregation, but it can be placed into aggregating category as well, as association rules describe the whole set of detail data related to currently examined object.

It is very easy to cover the existentially qualified conjunction case as well. In fact, the principle is very similar to association rules because they have conjunctions in common. If we create a conjunction of literals (out of attributes in detail data table), we can create a corresponding bit string and look for the bit with value 1 (in the segments belonging to individual objects of master table). If we find the bit with value 1, the existential quantifier is satisfied. Note that it is not necessary to count the number of 1 as in the case of association rules, so we can implement it substantially faster.

Also, note that there is a major difference between virtual attributes created as aggregations and association rules. While aggregations are computed from the detail data directly (e.g. by calling SQL queries or running more complex user-defined functions), creating virtual attributes out of association rules implies discretization of detail data (if necessary), creating bit strings of discrete categories, running a kind of single-table data mining engine (GUHA-based) and finally discretization of the results to allow further use as a virtual attribute in the master table.

In the paper [9], authors also note a practical difference between aggregating and selective methods. While the former approach is more suitable to highly non-determinate domains (usually in business), the latter is geared more towards structurally complex domains, such as molecular biology or language learning. This holds in comparison of WARMR and proposed Rel-Miner as well. While Rel-Miner is supposed to work with simple database schema (master-detail) containing many-valued categories and even real numbers, WARMR works with arbitrarily complex data structures with only “simple data”, searching for frequent structural patterns.

Continuing this differentiation, WARMR produces association rules spanning data tables to an arbitrary depth, bound together by the unique identification of the basic object. It is also capable of working with recursive tables. The output is a structural pattern that holds frequently, looking to individual sub-databases created from the original one by selecting rows regarding a single object at a time from all tables. On the contrary, Rel-Miner produces association rules that you can always comprehend the same way as in the single-table case, only “enhanced” by virtual attributes, which were created at run time from detail tables.

4 Making Rel-Miner Feasible

4.1 Complexity of relational hypotheses

As we already mentioned, the hypotheses space for the relational association rules is enormous. The total number of hypotheses grows with combinatorial numbers considering the number of attributes of the master table. While the number of possible aggregations is linear to the number of attributes in detail tables, the number of virtual attributes made of association rules suffers the combinatorial growth.

We may want to limit this growth considerably by limiting the number of virtual attributes (made of association rules in particular) that can be used simultaneously. The rationale is that every such an attribute strongly contributes to the complexity of the result. The process of data mining must keep in mind that it should deliver novel, potentially useful and ultimately understandable patterns. Using complex association rules as a virtual attributes makes the interpretation actually very difficult, so there is no point in pushing too hard with it.

4.2 Reordering of Hypotheses Evaluation

Another useful concept is a suitable reordering of the execution. The idea is to verify the hypotheses starting with the simple ones and going through more and more complex ones. Even if we admit the impossibility to process the entire hypotheses space, we want to be able to get the reasonable sample of results. Evaluation of the complexity of a hypothesis is inherently a heuristic function and the system should allow the user to fine-tune it to his/her needs.

This reordering implies that in the case of early stopping of the computation at any time, we are guaranteed to have finished the evaluation of all simpler hypotheses, which makes the partial result still somehow usable.

Unfortunately, this concept makes the process of verification rather inefficient. In the paper [13], the authors recommend using of the query-packs. That means to evaluate hypotheses in such order that similar hypotheses (having many attributes in conjunction in common) are evaluated together. It saves computing of the common part repeatedly. Analogous idea is in literal and cedent traces that were implemented in LISp-Miner project (see [7]).

We suggest a compromise solution for Rel-Miner, where the whole hypotheses space is divided into jobs. Every job is restricted in the hypothesis complexity (see previous paragraph) and is “local” in terms of cedent traces. Individual jobs can be evaluated optimally, even though there is some overhead among the jobs.

4.3 Distributed Computing

Thinking about the jobs, we can naturally make Rel-Miner distributed to many computers/processors. It is not difficult to design a “job manager” that will assign individual jobs to different computers. The only difficulty is with an excessive network communication; if we take into account the expected size of data in a bit string representation (see [7]), we can establish a bit string cache at every computer, so it will fetch every bit string at most once and store it for later use.

4.4 Limiting the Amount of Output

Together with the hypotheses space explosion, we can expect increased number of valid hypotheses that will go to output. We must keep in mind that our data mining is supposed to provide potentially useful results. Hence, the number of hypotheses (as results) that the user can review is limited. It is certainly possible to employ post-processing methods, such as filtering and sorting (by user-provided criteria). The possibility to limit the total size of output is nonetheless convenient. This limitation must preserve the most important hypotheses, based on their natural value.

4.5 Importance of Post-Processing

To continue with notes from the previous paragraphs, it is important to point out that the post-processing step (i.e. the evaluation according to CRISP-DM methodology [3]) is more important than in a single-table case. We also mentioned that blending the steps of data preparation, modeling and evaluation makes the overall usability worse.

We suggest creating a visual browser tool that will display the hypotheses lattice as a dynamic graph. In this graph, nodes are individual hypotheses and edges connect nodes that have something in common. The strength of the edge (displayed as length and/or thickness) denotes the measure of similarity between the connected hypotheses. The size (and/or color) of the node denotes its natural value. Unfortunately, making of such a tool is a task for a separate and generally independent research.

4.6 Possibility of User Interaction during Computation

With respect to possibly "endless" task run, it is necessary to allow viewing of interim results. During browsing of the results, the user should be able to make slight modifications to the task setting easily. When the user sees for instance an unreasonable hypothesis, because it describes semantically incorrect aggregation or an unsuitable combination of attributes, he/she can prevent the generator to work further on it (recall that although "local", more complex hypotheses are left for future computations). On the contrary, boosting the priority of some hypothesis can mitigate its complexity, so the hypotheses in the close neighborhood will be computed sooner. Research on this topic is again out of scope of this paper.

4.7 Hypotheses in Natural Language

The paper [9] shows, how to formulate mechanically association rules to reasonable sentences in natural language. You can make a limited language model, which mainly consists of formulation patterns independent of the application domain. Afterward, you supply domain vocabulary that is closely related to attributes in the database. Another area of research is to extend this method for use in relational data mining.

5 Conclusion

We have presented the extension to GUHA method for relational data mining. This approach is different from the ILP approach in many aspects, namely the target application domain. Association rules produced by Rel-Miner can contain aggregations, inner association rules (valid on detail data tables) or existential attributes. This makes them more general compared to association rules produced by WARMR algorithm. On the other hand, Rel-Miner is meant to run on a simple "master-detail" database schema, not going to depth more than 1. ILP methods in general are capable

of mining in arbitrarily complex database structures, focusing more on the structure of the database itself, than on complex computations with attribute values.

Both methods suffer from an explosion of the hypotheses space and both have unique ways to deal with it. Among the most important, Rel-Miner uses heuristics to prune the space and chooses the verification of hypotheses in the order from simple to more complex.

Reference

1. Hájek, P. – Havránek, T.: *Mechanizing Hypothesis Formation – Mathematical Foundations for a General Theory*. Springer-Verlag, 1978
2. Aggraval, R. et al.: *Fast Discovery of Association Rules*. In Fayyad, U.M. et al.: Advances in Knowledge Discovery and Data Mining, pp. 307-328, AAAI Press / MIT Press, 1996
3. CRISP-DM Methodology – http://www.crisp-dm.org/
4. Šimůnek, M.: *Academic KDD Project LISp-Miner*. In Abraham, A. – Franke, K. – Köppen, M. (eds.): Intelligent Systems Desing and Applications (Advances in Soft Computing series), ISBN 3-540-40426-0, Springer-Verlag, 2003, pp. 263–272
5. Academic KDD software system – LISp-Miner: http://lispminer.vse.cz/
6. Rauch, J.: *Interesting Association Rules and Multi-relational Association Rules*. In Communications of Institute of Information and Computing Machinery, Taiwan, Vol. 5, No. 2, May 2002, pp. 77-82
7. Rauch J. – Šimůnek, M.: *An Alternative Approach to Mining Association Rules*. In Lin, T.Y. – Ohsuga, S. – Liau, C.J. – Tsumoto, S. (eds.): Data Mining: Foundations, Methods, and Applications, Springer-Verlag, 2005
8. Rauch, J. – Šimůnek, M. – Lín, V.: *Mining for Patterns Based on Contingency Tables by KL-Miner – First Experience*. In Lin, T.Y. – Ohsuga, S. – Liau, C.J. – Hu, X. (eds.): Foundations and Novel Approaches in Data Mining, Springer-Verlag, 2005
9. Strossa, P. – Černý, Z. – Rauch, J.: *Reporting Data Mining Results in a Natural Language*. In Lin, T.Y. – Ohsuga, S. – Liau, C.J. – Tsumoto, S. (eds.): Data Mining: Foundations, Methods, and Applications, Springer-Verlag, 2005
10. Džeroski, S. – Lavrač, N. (eds.): *Relational Data Mining*, Springer 2001, ISBN: 3-540-42289-7
11. Džeroski, S.: *Multi-Relational Data Mining: An Introduction*. In ACM SIGKDD Explorations Newsletter, Vol. 5, Issue 1, 2003, pp. 1-16
12. Dehaspe, L. – De Raedt, L.: *Mining Association Rules in Multiple Relations*. In Proceedings of the 7th International Workshop on Inductive Logic Programming, Volume 1297, LNAI, pp. 125-132, Springer-Verlag, 1997.
13. Blockeel, H. – Dehaspe, L. – Demoen, B. – Janssens, G. – Ramon, J. – Vandecasteele, H.: *Improving the Efficiency of Inductive Logic Programming Through the Use of Query Packs*. In Journal of Artificial Intelligence Research, volume 16, 2002, pp. 135-166
14. Blockeel, H. – Bruynooghe, M.: *Aggregation Versus Selection Bias, and Relational Neural Networks*. In IJCAI-2003 Workshop on Learning Statistical Models from Relational Data, SRL-2003, Acapulco, Mexico
15. Assche, A. van – Vens, C. – Blockeel, H. – Džeroski, S.: *A Random Forest Approach to Relational Learning*. In2 Dietterich, T. – Getoor, L. – Murphy, K. (eds.): ICML 2004 Workshop on Statistical Relational Learning and its Connections to Other Fields, pp. 110-116
16. Blockeel, H. – Sebag, M.: *Scalability and Efficiency in Multi-Relational Data Mining*. In ACM SIGKDD Explorations Newsletter, Vol. 5, Issue 1, 2003, pp. 17-30
17. A Learning Engine for Proposing Hypotheses (Aleph), ILP system http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph_toc.html
18. The ACE Datamining system, http://www.cs.kuleuven.ac.be/~dtai/ACE/

Testing Dimension Reduction Methods for Text Retrieval

Pavel Moravec

Department of Computer Science, VŠB – Technical University of Ostrava
17. listopadu 15, 708 33 Ostrava-Poruba, Czech Republic
pavel.moravec@vsb.cz

Abstract. In this paper, we compare performance of several dimension reduction techniques, namely LSI, random projections and FastMap. The qualitative comparison is based on rank lists and evaluated on a subset of TREC 5 collection and corresponding TREC 8 ad-hoc queries. Moreover, projection times and intrinsic dimensionality were measured to present a common baseline for methods' usability.

Key words: vector model, LSI, information retrieval, random projection, FastMap, ranked lists, TREC, intrinsic dimensionality, curse of dimensionality

1 Introduction

The *information retrieval* [14, 2] deals among other things with storage and retrieval of multimedia data that can be usually represented as vectors in multidimensional space. This is especially suitable for *text retrieval*, where we store a *collection* (or *corpus*) of texts. There are several models used in text retrieval, from which we will use the *vector model* [12] providing qualitatively better results than the *Boolean model* [14], which combines word matching with Boolean operators.

In the vector model, we have to solve several problems. The ones addressed in this paper are problems with the ability to index given collection, search efficiency and result set quality.

Latent semantic indexing (*LSI*) adds an important step to the indexing process. In addition to recording which terms a document contains, the method examines the document collection as a whole, to see which other documents contain some of those same terms. LSI considers documents that have many terms in common to be semantically close, and ones with few words in common to be semantically distant. However it is not suitable for huge collections and is computationally expensive, so other methods of dimension reduction were proposed. We test two of them – *Random projection*, which projects document vectors into a subspace using a randomly generated matrix, and *FastMap*, a pivot-based method based loosely on *Multi-Dimensional Scaling*. Since both of them were created for Euclidean spaces, they may not supply good results for a

different distance functions. In our case, we need to evaluate, how these methods behave when using *cosine measure*, common in text retrieval.

The rest of this paper is organised as follows. In the second section, we describe classic vector model and its problems, which may be addressed by dimension reduction. The third section explains used dimension reduction methods. In the fourth section, we briefly describe qualitative measures used for evaluation of our tests and in the fifth the projection properties. In the sixth section, we supply results of tests on a subset of TREC 5 collection. In conclusions we give ideas for future research.

2 Vector model

In vector model, a document D_j is represented as a vector d_j of term weights, which record the extent of importance of the term for the document.

To portrait the vector model, we usually use an $n \times m$ *term-by-document matrix* A, having n rows – term vectors $t_1 \dots t_n$ (where n is the total number of terms in collection) and m columns – document vectors $d_1, \dots d_m$, where m is the size of collection (or *corpus*) C.

Term weights can be calculated in many different ways: $w_{ij} \in \{0, 1\}$; as a membership grade to a fuzzy set; or as a product of functions of term frequency both in a document and in the whole collection [13] (usually $tf.idf$ – count of term occurrences in the document multiplied by a logarithm of the inverse portion of documents containing the term). The normalisation of document vectors is sometimes applied during index generation phase to make the calculation in the retrieval phase faster.

A query Q is represented as an n-dimensional vector q in the same vector space as the document vectors. There are several ways how to search for relevant documents. Generally, we can compute some L_n metrics to represent the similarity of query and document vectors. However, in text retrieval better results can be obtained by computing similarity, usually using the *cosine measure*:

$$SIM_{cos}(d_j, q) = \frac{d_j \bullet q}{||d_j||.||q||} = \frac{\sum_{i=1}^{n} (w_{i,j} \cdot q_i)}{\sqrt{\sum_{i=1}^{n} w_{i,j}^2 \cdot \sum_{i=1}^{n} q_i^2}}$$

As one can see, we do not only obtain documents which are considered relevant, but according to their similarity (or distance) to the query vector, we can order them and obtain a rank for every document in the answer set. If we need a metrics instead of similarity measure, we can use the deviation metric $d_{dev}(x, y) = \arccos(SIM_{cos}(x, y))$.

We can define a *threshold* t, too. All documents closer than t will be considered relevant, whilst the rest will be irrelevant. However, the choice of t is not exact and its value is usually determined experimentally.

The main problem of the vector model is that the document vectors have a big dimension (e.g. 150,000) and are quite sparse (i.e. most co-ordinates are zero). If we store them as classical vectors, the storage volume is huge – consider size of a term-by-document matrix consisting of 100,000 terms and 200,000 documents.

We can use existing compression schemes for the term-by-document matrix representation to decrease memory usage, but then the access time is much longer and we are limited by the fact, that we cannot access either the term or the document vectors quickly. Another way is to use combined storage with both row and column compression, but updating would still pose a problem.

The second problem is the so-called *"curse of dimensionality"*, which causes classical indexing structures like M-trees, A-trees, iDistance, etc. (see [5]), to perform in the same way or even worse than sequential scan in high dimensions. This is caused by the distribution of document vectors, which prevents partitioning into meaningful regions.

Third, the synonyms of terms and other semantically related words are not taken into account.

The first two problems can be addressed for queries containing only a few words by *inverted list*, which is in fact a compressed storage of term vectors. Only term vectors for terms contained in a query Q are loaded and processed, computing rank for all documents containing at least one of the terms at once. However, the inverted list is not efficient when searching for similar documents, because significant part of index must be processed.

3 Dimension reduction methods

We used three methods of dimension reduction - latent semantic indexing, random projection, and FastMap, which are briefly described bellow.

3.1 Latent semantic indexing

LSI [3] is an algebraic extension of classical vector model. Its benefits rely on discovering *latent semantics* hidden in the term-by-document matrix A. Informally, LSI discovers significant groups of terms (called *concepts*) and represents the documents as linear combinations of the concepts. Moreover, the concepts are ordered according to their significance in the collection, which allows us to consider only the first k concepts important (the remaining ones are interpreted as "noise" and discarded). To name the advantages, LSI helps solve problems with synonymy and homonymy. Furthermore, LSI is often referred to as more successful in recall when compared to vector model [3], which was proved for pure (only one topic per document) and style-free collections [11].

Formally, we decompose the term-by-document matrix A by *singular value decomposition* (SVD), calculating singular values and singular vectors of A. SVD is especially suitable in its variant for sparse matrices.

Theorem 1 (Singular value decomposition [3]). *Let A is an $n \times m$ rank-r matrix and values $\sigma_1, \ldots, \sigma_r$ are calculated from eigenvalues of matrix AA^T*

as $\sigma_i = \sqrt{\lambda_i}$. *Then there exist 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 term-by-document matrix A. The left and right singular vectors (i.e. U and V matrices) are not sparse. 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.

Definition 1 ([3]). *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 *concept-by-document matrix* $D_k = \Sigma_k V_k^T$ is used in LSI as the representation of document collection. The document vectors (columns in D_k) are now represented as points in k-dimensional space (the *pseudodocument-space*). For an illustration of rank-k SVD see Figure 1.

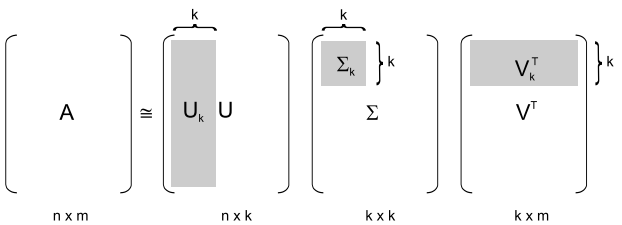


Fig. 1. rank-k SVD

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), it is known to be dependent on the number of topics in collection, however its exact value cannot be simply determined.

The LSI is hard to compute with complexity $O(mn^2)$ for dense and $O(mnc)$ for sparse matrices having on the average c nonzero values per column [11]. Once computed, it reflects only the decomposition of original term-by-document matrix. If several hundreds of documents or terms have to be added to existing decomposition (*folding-in*), the decomposition may become inaccurate. Because the recalculation of LSI is expensive, so it is impossible to recalculate LSI every time documents and terms are inserted. The *SVD-Updating* [3] is a partial solution, but since the error slightly increases with inserted documents and terms, If the updates happen frequently, the recalculation of SVD may be needed soon or later.

3.2 Approximate LSI calculation

Several approximate methods for faster SVD calculation were offered, such as application of Monte-Carlo method [8] and using random projection (see section 3.3) of document vectors into suitable l-dimensional subspace before LSI calculation for resulting k dimensions [11].

We used the latter method, applying LSI on a matrix with reduced document vectors created by random projection. This method has a complexity of $O(ml(l+c))$.

3.3 Random projection

Random projection is a fast method of dimension reduction. Unlike LSI method, it does not require expensive computation of decomposition. Instead, it uses a randomly-generated projection matrix to reduce dimension of vector space. Vector from original space with dimension n is multiplied with projection matrix to obtain a vector in reduced space of dimension l, where $l << n$.

Results of dimensionality reduction by random projection are of course worse than in case of LSI and we do not obtain latent semantics. If the reduced dimension is high enough and random values building projection matrix have a zero-mean unit-variance distribution such as $N(0,1)$, the Euclidean distances and angles between vectors are well-preserved.

The minimal "safe" dimension can be obtained from Johnson-Lindenstrauss lemma, however the currently known bound is still quite high and experiments showed that even smaller dimensions can be used [4]. Interestingly, the resulting dimension does not depend on original one, only on number of original points. With current best known bound, the lemma looks as follows:

Theorem 2 (Johnson-Lindenstrauss [1]). *For every set P of m points in $\mathbb{R}^n$, given $\varepsilon > 0$, $\beta > 0$ and $l > 0, l \geq l_0 = \frac{4+2\beta}{\varepsilon^2/2-\varepsilon^3/3} \log m$, there exists with probability at least $1 - n^{-\beta}$ mapping $f : \mathbb{R}^n \to \mathbb{R}^l$, such that for all $u, v \in P$*

$$(1-\varepsilon)||u-v||^2 \leq ||f(u)-f(v)||^2 \leq (1+\varepsilon)||u-v||^2$$

Since we JL lemma considers only Euclidean distances, we don't have yet any bounds for cosine measure. Papadimitriou et al showed in [11] that a bound can be found for cosine measure, too. In that case

$$f(v_i).f(v_j) \leq (1-\varepsilon)v_i.v_j + \varepsilon(v_i^2 + v_j^2).$$

Are the lengths of all $v_k \leq 1$, changes the inner product at most by 2ε. Again, real-life data indicate that the bound is still too high and smaller dimensions can be used.

When calculating the Euclidean distances, we need to apply a scaling factor $\sqrt{n/l}$ first, to obtain correct results because less coordinates are being used.

Classical implementations of random projection used orthogonalisation, normalisation and a dense projection matrix with Gaussian distribution. Achlioptas showed that orthogonalisation and normalisation can be skipped. He also proposed yet another powerful simplification – instead of using real coefficients of $N(0,1)$ distribution, he offered two possible distributions for elements r_{ij} of projection matrix R [1]:

$$r_{ij} = \sqrt{3}.\begin{cases} -1 \text{ with probability } \frac{1}{6} \\ 0 \quad \text{with probability } \frac{2}{3} \\ +1 \text{ with probability } \frac{1}{6}. \end{cases} \qquad r_{ij} = \begin{cases} -1 \text{ with probability } \frac{1}{2} \\ +1 \text{ with probability } \frac{1}{2}. \end{cases}$$

The $\sqrt{3}$ component does not have to be stored in projection matrix. It can be used together with scaling factor after calculation of projected vector coordinate. If we are calculating cosine measure, it can be even discarded and instead of multiplication, we can use addition and subtraction.

We used this method in our tests, since previous results [10] indicated almost the same performance as in the case of classic random projection. The complexity of random projection is $O(mcl)$.

3.4 FastMap

FastMap [7] 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.

The complexity of FastMap is $O(mck)$ for sparse and $O(mnk)$ for dense matrices.

4 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 [14]. 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.

5 Projection properties

5.1 Intrinsic dimensionality

The search in a collection of high-dimensional document vectors is negatively affected by a phenomenon called the *curse of dimensionality* [6], which causes

almost all regions to be overlapped by nearly every "reasonable" query region; so that searching deteriorates to sequential scan over all the classes. To judge the indexability of given dataset (in a metric space), we can use the concept of *intrinsic dimensionality* [6], defined as $\rho = \frac{\mu^2}{2\sigma^2}$, where μ and σ^2 are the mean and the variance of the dataset's *distance distribution*. In other words, the intrinsic dimensionality is low if there exist tight clusters of objects. Conversely, if all pairs of the indexed objects are almost equally distant, the intrinsic dimensionality is high (i.e. the mean is high and/or the variance is low), which means that the dataset is poorly intrinsically structured.

5.2 Projection stress

Sometimes, we need to verify, how well are the distances between objects preserved in the reduced dimension. To do so, we usually calculate the *stress* of projection f as

$$stress = \sqrt{\frac{(\sum_{i,j=1}^{m}(d'(f(x_i), f(x_j)) - d(x_i, x_j))^2}{\sum_{i,j=1}^{m} d^2(x_i, x_j)}},$$

where d is the distance function in original and d' in projected space. The lower the stress, the better. If $stress = 0$, then the projection did not change the distances at all.

However, the stress function must not be overrated – even if the distances are not well-preserved, they might have been scaled by some factor, making the only difference in the choice of similarity threshold.

6 Experimental results

For testing of our approach, we used a subset of TREC collection [16], consisting of 16,889 Los Angeles Times articles (years 1989 and 1990) assessed in TREC-8 ad-hoc queries. We indexed this collection, removing well-known stop-words and terms appearing in more than 25% of documents, thus obtaining 49,689 terms.

We calculated random projection into dimensions $l \in \{100, 250, 500, 1000\}$; both classic and approximate LSI (with random projection into $l = 1000$) for $k \in \{100, 250\}$. For FastMap, we used for every value of k suggested 5 iterations to choose "most distant" points. Additionally we calculated FastMap for $k = 100$ and 3 iterations (which yielded slightly worse results). Classic LSI was included to provide a baseline, since its improvement of recall is well-known.

The reduction and query projection times are shown in Table 1 [1].

6.1 Analytical results

We calculated stress and intrinsic dimension for each projection method for deviation metrics.

[1] Since the LSI was calculated on a different computer, the LSI calculation times are only approximate

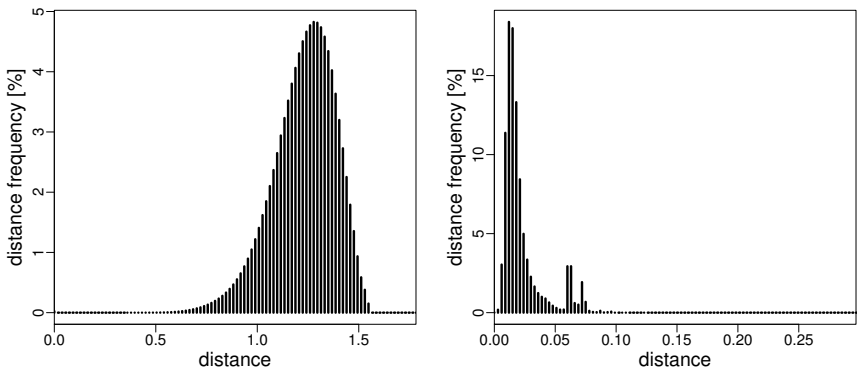
Table 1. Times of (a) dimension reduction and (b) projection of 45 TREC queries [s]

	Reduction method		
k	LSI	FastMap	RP
50	–	771	5.43
100	> 5400	910	11.98
250	> 14400	2193	26.81
500	–	–	50.51
1000	–	–	93.44

(a)

	Reduction method			
k	LSI	FastMap	RP	RP+LSI
50	0.05	0.14	0.02	0.05
100	0.12	0.28	0.03	0.12
250	0.35	1.00	0.07	0.35
500	–	–	0.13	0.78
1000	–	–	0.25	–

(b)



(a) (b)

Fig. 2. Distance distribution histograms for Deviation metrics and (a) vector model, (b) FastMap

The stress, summarised in Table 2a is quite low for both LSI and random projection, however in case of FastMap are the deviations not well-preserved. From the look at distance distribution histograms of original and FastMap reduced space in Figure 2 one can observe that the distances are highly reduced. The question, if the change affects only the dissimilarity threshold will be partly solved in the next section.

In Table 2b, we can observe high intrinsic dimensions for both LSI variants and especially for random projection, whilst the intrinsic dimension for FastMap is surprisingly low. Additional tests on real data structures are required for FastMap, to verify the indexability of reduced data. In case of LSI, we recently offered a modified σ-LSI model [15], which trades the precision for better indexing with Metric Access Methods, namely M-trees.

Table 2. (a) Stress and (b) intrinsic dimensionality of reduced datasets

	Reduction method			
k	LSI	FastMap	RP	RP+LSI
50	0.210	0.978	0.296	0.247
100	0.224	0.978	0.284	0.259
250	0.242	0.980	0.282	0.270
500	–	–	0.279	0.275
1000	–	–	0.278	–

(a)

	Reduction method			
k	LSI	FastMap	RP	RP+LSI
50	25.1	0.2	53.3	46.8
100	51.1	0.5	100.2	93.9
250	121.1	0.9	217.1	206.4
500	–	–	343.7	329.7
1000	–	–	489.3	–
VM	⟵ 31.8 ⟶			

(b)

6.2 Query Evaluation

Firstly, we used rank lists and measured interpolated average precision of the above mentioned TREC Queries at the 11 standard recall levels. Results are summarised in Figure 3. We can see that while classic LSI provides even better results than vector model due to latent semantics, other reduction techniques try with a different success to reach the results of vector model. In our case, we got results close to vector model for random projection with l=1000 and FastMap with k=250.

Since the important part of precision-recall curve is close to the 100% recall, we also calculated the mean average precision for all relevant documents in rank lists. The relative results against vector model (100%) are shown in Table 3.

Table 3. Mean average precision of different reduction methods

	Reduction method			
k	LSI	FastMap	RP	RP+LSI
50	128%	31%	13%	85%
100	155%	58%	24%	98%
250	112%	80%	37%	79%
500	–	–	59%	77%
1000	–	–	74%	–

7 Conclusion

In this paper, we have compared three well-known dimension reduction methods from the view of indexability, distance preservation and results on real-live text data (using cosine measure as similarity function). Whilst the LSI is known to provide latent semantics, it is computationally expensive and in case we only need to battle the "curse of dimensionality" by reducing the dimension, FastMap

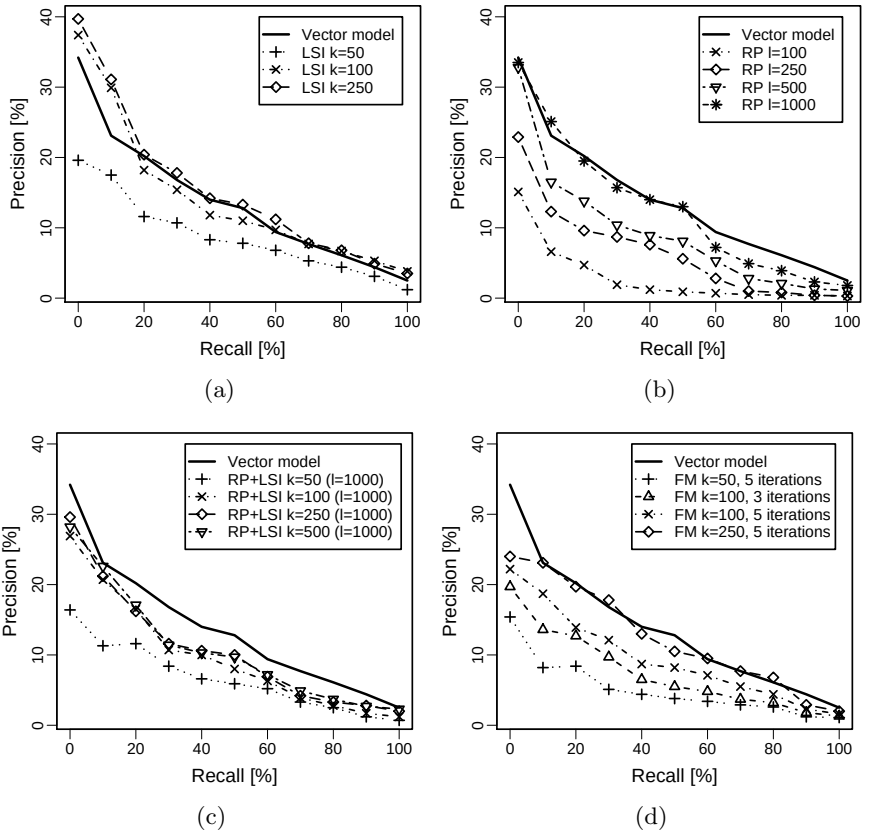


Fig. 3. Precision at the 11 standard recall levels: (a) LSI, (b) random projection, (c) approximate LSI calculation and (d) FastMap

or random projection may suffice. As expected, LSI was the slowest, but most exact method, followed by FastMap, which is faster but less accurate, and Random projections which are fast, but accurate only in high dimensions and have high intrinsic dimensionality.

There are some other newly-proposed methods, which may be interesting for future testing, e.g. the SparseMap [9]. Additionally, faster pivot selection technique based on text corpus properties may be considered. Finally, testing FastMap with deviation metrics on metric structures should answer the question of projected data indexability.

References

1. D. Achlioptas. Database-friendly random projections. In *Symposium on Principles of Database Systems*, 2001.
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. E. Bingham and H. Mannila. Random projection in dimensionality reduction: applications to image and text data. In *Knowledge Discovery and Data Mining*, pages 245–250, 2001.
5. C. Böhm, S. Berchtold, and D. Keim. Searching in High-Dimensional Spaces – Index Structures for Improving the Performance of Multimedia Databases. *ACM Computing Surveys*, 33(3):322–373, 2001.
6. E. Chávez and G. Navarro. A probabilistic spell for the curse of dimensionality. In *Proc. 3rd Workshop on Algorithm Engineering and Experiments (ALENEX'01), LNCS 2153*. Springer-Verlag, 2001.
7. 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):163–174, 1995.
8. A. Frieze, R. Kannan, and S. Vempala. Fast Monte-Carlo Algorithms for Finding Low Rank Approximations. In *Proceedings of 1998 FOCS*, pages 370–378, 1998.
9. 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):530–549, 2003.
10. P. Moravec, M. Krátký, and V. Snášel. Random Projections for Dimension Reduction in Information Retrieval Systems. In *Proceedings of IMAMM'03 Conference*, 2003.
11. C. H. Papadimitriou, H. Tamaki, P. Raghavan, and S. Vempala. Latent semantic indexing: A probabilistic analysis. In *Proocedings of the ACM Conference on Principles of Database Systems (PODS)*, pages 159–168, 1998.
12. G. Salton. *The SMART Retrieval System – Experiments in Automatic Document Processing*. Prentice Hall Inc., Englewood Clifs, 1971.
13. G. Salton and C. Buckley. Term weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.
14. G. Salton and G. McGill. *Introduction to Modern Information Retrieval*. McGraw-ill, 1983.
15. T. Skopal and P. Moravec. Modified LSI Model for Efficient Search by Metric Access Methods. In *Proceedings of ECIR'05 Conference*, Santiago de Compostela, Spain, March 2005.
16. E. M. Voorhees and D. Harman. Overview of the sixth text REtrieval conference (TREC-6). *Information Processing and Management*, 36(1):3–35, 2000.

Query Optimization by Genetic Algorithms

Suhail S. J. Owais, Pavel Krömer, and Václav Snášel

Department of Computer Science, VŠB – Technical University of Ostrava
17. listopadu 15, 708 33 Ostrava-Poruba, Czech Republic
vaclav.snasel@vsb.cz

Abstract. This study investigated the use of Genetic algorithms in Information retrieval in the area of optimizing a Boolean query. A query with Boolean logical operators was used in information retrieval. For Genetic algorithms, encoding chromosomes was done from Boolean query; where it was represented in the form of tree prefix with indexing for all terms and all Boolean logical operators. Information retrieval effectiveness measures precision and recall used as a fitness function in our work. Other Genetic algorithms operators were used as single point crossover on Boolean logical operators, and mutation operator was used to exchange one of the Boolean operators and, or, and xor with any other one. The goal is to retrieve most relevant documents with less number of nonrelevant documents with respect to user query in Information retrieval system using genetic programming.

1 Introduction

Information retrieval system is used to retrieve documents that depend on or relevant to the user input query. The growth in the number of documents in the internet made it necessary to use the best knowledge or methods in retrieving the most relevant documents to the user query. For this, Information Retrieval has become a fact of life for most internet users.

Information retrieval systems deal with data bases which is composed of information items documents that may consist of textual, pictorial or vocal information. Such systems process user queries trying to allow the user to access the relevant information in an appropriate time interval. Where the art of searching will be in the databases or hypertext networked databases such as internet or intranet for text, sound, images or data, see [1]. Thus an information system has at its heart a collection of data about reality [4].

Most of the information retrieval systems are based on the Boolean queries where the query terms are joined by the logical operators AND and OR. The similarity between a query and documents is measure by different retrieval strategies that are based on the more frequent terms found in both the document and the query. The more relevant document is deemed to be the query request. The most frequently used measures of retrieval effectiveness are **precision** *the percentage of the retrieved documents that are relevant* and **recall** *the percentage of the relevant documents that are retrieved.*

Information retrieval is concerned with collection and organization of texts, responding to the requests of internet users for the information seeking text, retrieving the most relevant documents from a collection of documents; and with retrieving some of non-relevant as possible. Information retrieval is involved in:

- Representation,
- Storage,
- Searching,
- Finding documents or texts or images those are relevant to some requirements for information desired by a user.

Genetic Algorithms (GA) represents one of the artificial intelligence algorithms that are attractive paradigm to improve performance in information retrieval systems. Retrieving necessary documents from a collection of such documents will be achieved using a query to select the most relevant documents. For implementation of genetic algorithms will be on queries. A form of genetic algorithm started to be applied in information retrieval systems in order to optimize the query by genetic algorithms

2 Genetic Algorithms (GA)

Genetic algorithms (GA) first described by John Holland in 1960s and further developed by Holland and his students and colleagues at the University of Michigan in the 1960s and 1970s [10]. GA used Darwinian Evolution to extract nature optimization strategies that uses them successfully and transform them for application in mathematical optimization theory to find the global optimum in defined phase space [5, 3].

GA is used to find approximate solutions to difficult problems through a set of methods or techniques *inheritance or crossover, mutation, natural selection, and fitness function.* Such methods are principles of evolutionary biology applied to computer science. GAs are useful for:

- Solving difficult problems.
- Modelling the natural system that inspired design.

Applying genetic algorithms over a population of individuals or chromosomes shows that several operators are utilized. They are as follows (see Figure 1):

- **Fitness operator**, metrics to measure scheduler performance for each chromosome in the problem, and calculate the values for each chromosome.
- **Selection operator**, select two chromosomes with the highest quality values from the population, that couple to produce two offspring.
- **Crossover operator**, exchanges two subparts of the selected chromosomes, the position of the subparts selected randomly.
- **Mutation operator**, randomly changes the allele value in some location.

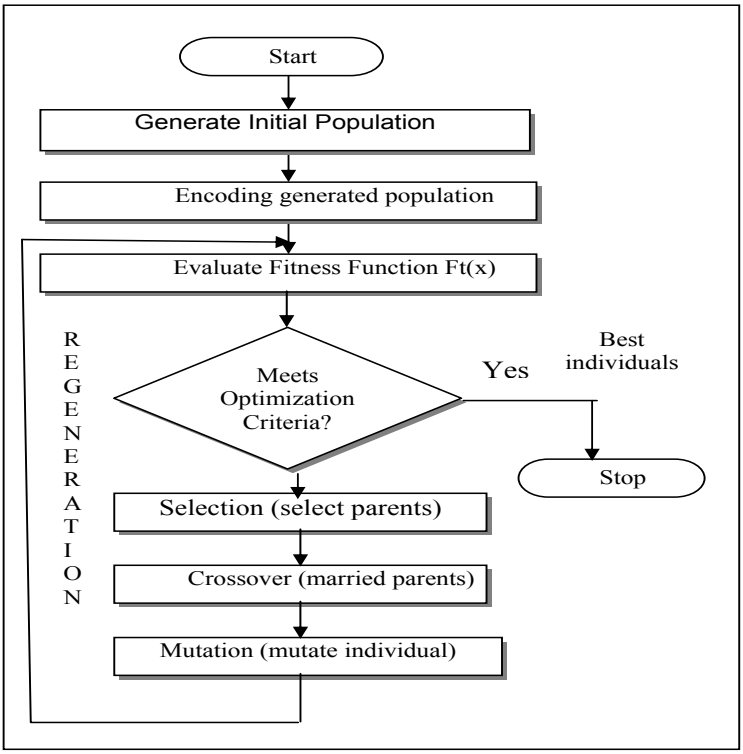


Fig. 1. Flowchart for Genetic Algorithm

3 Information Retrieval and Genetic Algorithm

This section will present the implementation of information retrieval using genetic algorithms (for SQL we can see [11, 8, 6, 2]). The GA is generally used to solve optimization problems [5]. GA starts on an initial population with fixed size of chromosomes "*P-chromosomes*". Each individual are coded according to chromosome length, where genes are allocated in each position in a chromosome with different data types, and each gene values called allele. In information retrieval, query for relevant documents are representing for each individual or chromosome, and each document described by set of terms. The description d_i for document D_i, where $i = 1 \ldots l$, the set of terms for D_i are T_j, where $j = 1 \ldots n$, thus $d_i = (w_{1_i}, w_{2_i}, \ldots, w_{n_i})$. The value for each term will be 1 if this term exists in the document or 0 if not (Note: about another weights for terms was mention in paper [9]), this indicate that the indexing function that is maps a given index term t and a given document d is

$$F : D \times T \rightarrow [0, 1].$$

Defining a query will be combination from set of terms and set of Boolean operators and, or, xor, not. The query set Q defined as set of queries for documents, define the query processing mechanism by which documents can be evaluated in terms of their relevance to a given query [7].

Ih this work, we develop genetic program for implementing GA with variable length of chromosomes and mixture symbolic of information, like real values and Boolean queries values.

4 Chromosome Encoding

In order to really understand the principle of chromosome encoding, consider a Boolean query with a series of terms $w_1, w_2, \ldots, w_n$, and set of Boolean operators from and, or, Xor , and Not. Examples of queries in infix form are:

$(w_2 \text{ or } w_6) \text{ and} (w_9 \text{ and } w_3)$
and
$(w_3 \text{ and } w_4) \text{ xor}((w_5 \text{ and } w_6) \text{ or } w_8)$

In the previous queries ordinary Boolean operators are used, and they are in infix form, they will be encoded to be chromosomes for genetic programming but in prefix form such as

$\text{and}(\text{or } w_2 w_6)(\text{and } w_9 w_3)$
and
$\text{xor}(\text{and } w_3 w_4)(\text{or}(\text{and } w_5 w_6) w_8)$

and also they will be represented in a tree form as the chromosomes shown in Figure 2.

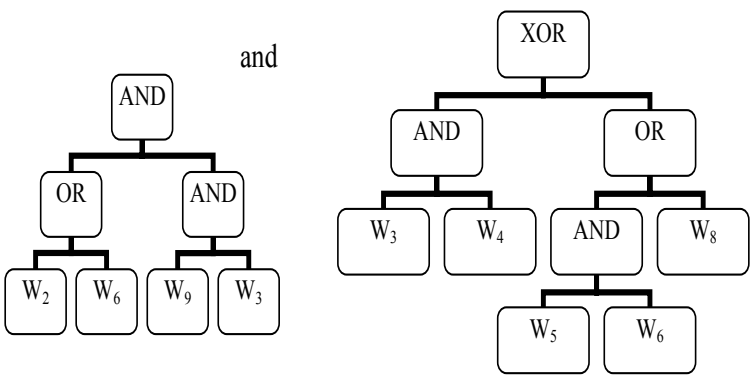


Fig. 2. Flowchart for Genetic Algorithm

5 Evaluation and Fitness Function

Evaluation of the information retrieval system is done by measuring its effectiveness. This is best measured by two statistics precision and recall, maximizing precision is subject to a constraint on the minimal recall accepted. Recall and precision are often perceived as being inversely related, i.e., complementary and competitive [7]. For any given set of documents called Collection of Documents, there is a subset of documents that are relevant to such query called Relevant documents, and a subset of documents that are retrieved called Retrieved Documents. Demonstration for precision and recall are shown in Figure 3 with respect to all documents collected. Where precision and recall are defined as:

$$Recall = \frac{RelevantRetrieved}{Relevant} \qquad Precision = \frac{RelevantRetrieved}{Retrieved}$$

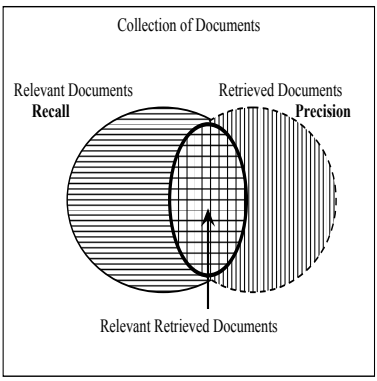


Fig. 3. Retrieved and relevant documents

Fitness function for our work will be considered as functions for precision and recall. One function will trade for the other function because both precision and recall are inversely related. And they will deal with the ranked documents. The shortcoming of using recall and precision as fitness functions is that if no relevant documents are retrieved by a chromosome, then its fitness is zero. This will lead to loss of all genes for this chromosome.

Computing recall and precision values for each query in our genetic programming as illustrated in equations 1 and 2, the first fitness function $RecallFitnessE_1$ measures recall (equ-1), and the second fitness function $PrecisionFitnessE_2$ measures Precision (equ-2). Where r_d is the relevance of document d (1 for relevant and 0 for nonrelevant), f_d is the retrieved document d (1 for retrieval and 0 for nonretrieval), and α and β are arbitrary weights. Where α and β are added specially to precision fitness function [7].

$$ReallFitnessE_1 = \frac{\sum_d [r_d \times f_d]}{\sum_d [r_d]}$$

$$PrecisionFitnessE_2 = \frac{\alpha \sum_d [r_d \times f_d]}{\sum_d [r_d]} + \frac{\beta \sum_d [r_d \times f_d]}{\sum_d [f_d]}$$

5.1 Selection operators

Processing genetic algorithm operators will be done in each generation over the best two chromosomes. From the population of chromosomes, the best two chromosomes depending on the highest fitness values for precision or recall measures will be selected. These two chromosomes will be called parent1 and parent2. These two parents will be used to produce two new offsprings.

5.2 Crossover or Recombination operators

In generating two new offsprings from the existing population, offsprings must have some inheritance from the two parents. Crossover will do that by exchanging subtree from parent1 with subtree from parent2. Positions for subtree1 and subtree2 will be selected randomly, and the position must be Boolean logical operator nodes in the tree; such as and, or, xor, and not. This will produce offspring1 and offspring2.

Single point crossover was used in our genetic programming. Index identifier was assigned for each term and each Boolean operator in a prefix form for each query that was encoded in a tree, see Figure 4.

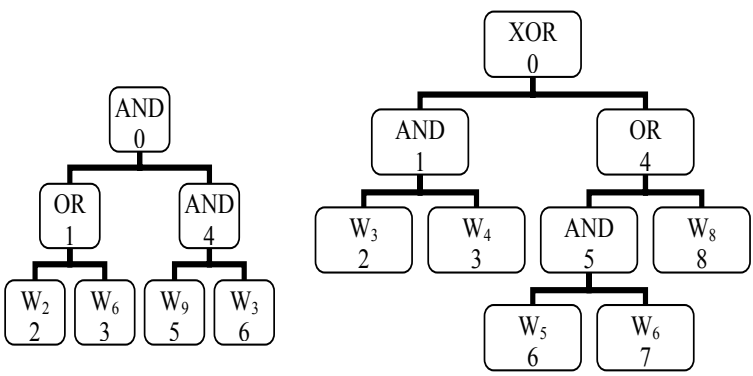


Fig. 4. Trees in Postfix form with indexing

For parent1, the selected subtree1 random number must not exceed the number of nodes of parent1. And same must be done for selecting subtree2 random number for parent2 must not exceed the number of nodes of parent2.

For example, if we have two random numbers **1** and **4** for subtree1 and subtree2 respectively, and we implement single point crossover process on parent1 and parent2 for trees shown in Figure 3, the new generated offsprings shown in see Figure 5, after exchange sub trees.

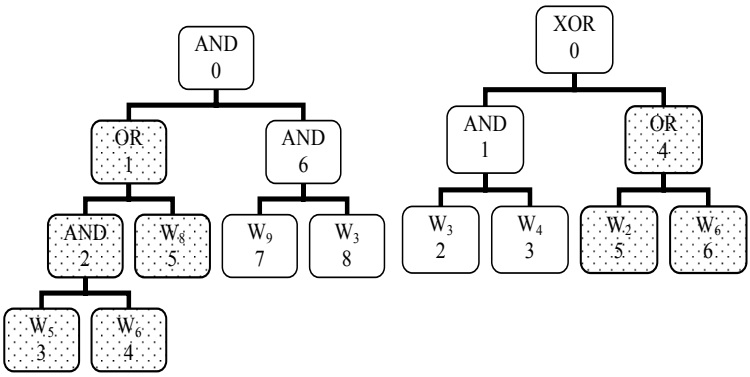


Fig. 5. New offsprings after single point crossover.

5.3 Mutation operators

Mutation, random perturbation in the chromosome representation, is necessary to assure that the current generation is connected to the entire search space, and it is necessary to introduce new genetic material into a population that has stabilized level [7]. In our genetic programming, mutation operator may change one Boolean logical operator by other one. That is for mutation will be for and, or and Xor Boolean operators.

For each new offspring, the selected random number is less than one. If the selected random number is less than mutation value, another selected random number will be drown and checking if it is on Boolean operator in the offspring, then exchange will process on it, (i.e. and will be or or xor). Example is shown in Figure 5, where mutation was not process on offspring1 but done on offspring2, where randomly if we select 1 to denote for the subtree position on offspring2, and from offspring2 we see that it is a Boolean logical operator and, and randomly it was changed to or, the new two offsprings are shown in Figure 5.

5.4 Updating Population

Fitness values will be computed for the two generated new offsprings, and after that the best chromosomes will survive for the next generation. Offsprings will be replaced by the worst chromosomes in the population, for this the new population

size will not be changed, replacing two worst chromosomes by two offsprings if the new offsprings fitness values exceed any of the chromosomes in the population.

The new population was ready to start next generation. And this will be repeated until number of generations or until we get the best solution for our problem.

5.5 Experiments

We developed our genetic program over a testing environment to see the influence of genetic algorithms on the information retrieval. The environment had a population from set of Boolean queries. For our program an input data used Boolean model of a collection of documents and set of Boolean queries as an initial population, and for execution the genetic program we used:

- Different Collections of documents with variant number of words and documents.
- Two sets of queries that represent as tree prefix form used as two different initial populations with fixed size in number of chromosomes (8 individuals).
- The results will be for a one requesting user query
- An initial values was fixed in all experiments are
  - Mutation value is 0.2
  - α is 0.25
  - β is 1.0
- Fixed number of generations are 50 generations

5.6 Limitations of current version

- At this time genetic operators are applied only for Boolean logical operators and, or and xor. This causes following: if queries in initial population do not include all the terms we have in the user query, they cannot come into existence.
- In our implementation, precision fitness value can be greater than 1.0, where the maximal value of precision is 1.25 that is coming from (α and β), thus we cannot interpret it as probability.

5.7 Experiment Tests

Our tests was done using user query

w_8 or w_2,

and we used two different initial populations Q_1 and Q_2; and they are shown in table 1, and table 2 respectively.

Table 1. Initial population Q_1.

No.	Query
1.	$(w_1 3 \text{ and } w_8) \text{ and}(w_{10} \text{ or } w_4)$
2.	$(w_1 \text{ and}(w_8 \text{ and } w_2)) \text{ or}(w_4 \text{ or } w_2)$
3.	$(w_1 \text{ or } w_2) \text{ and}((w_5 \text{ or } w_4) \text{ and}(w_3 \text{ and } w_6))$
4.	$(w_9 \text{ and } w_{14})$
5.	$(w_{14} \text{ and } w_1)$
6.	$(w_2 \text{ or } w_6) \text{ or}(w_8 \text{ and } w_{13})$
7.	$(w_3 \text{ and } w_4) \text{ or}((w_{12} \text{ xor } w_{15}) \text{ and } w_8)$
8.	$(w_1 \text{ or } w_5)$

Table 2. Initial population Q_2.

No.	Query
1.	$(w_{13} \text{ and } w_8) \text{ and}(w_{10} \text{ or } w_4)$
2.	$(w_1 \text{ and}(w_8 \text{ and } w_2)) \text{ or}(w_4 \text{ or } w_2)$
3.	$(w_1 \text{ or } w_2) \text{ and}((w_5 \text{ or } w_4) \text{ and}(w_3 \text{ and } w_6))$
4.	$(w_9 \text{ and } w_{14})$
5.	$(w_{14} \text{ and } w_1)$
6.	$(w_2 \text{ xor } w_8) \text{ or}(w_8 \text{ and } w_{13})$
7.	$(w_3 \text{ and } w_4) \text{ or}((w_2 \text{ or } w_8) \text{ and } w_8)$
8.	$(w_1 \text{ or } w_5)$

We mention that in initial population Q_1 there is in one query contain a sub expression

$w_8 \text{ and } w_2$,

and in initial population Q_2 there are in three different queries contains three different sub expressions

$w_8 \text{ and } w_2$, $w_8 \text{ or } w_2$, $w_8 \text{ xor } w_2$.

Our testing of execution for genetic programming was done independently from other execution results. Three cases were studied. The results of our tests with different collections (different number of documents and different number of words/terms). The three cases descriptions are:

- Test case 1: collection of 10 documents, 30 words in collection, maximal number of different words in a document is 10. Results are shown in table 3.
- Test case 2: collection of 200 documents, 50 words in collection, maximal number of different words in a document is 50. Results are shown in table 4.
- Test case 3: collection of 5000 documents, 2000 words in collection, maximal number of different words in a document is 500. Results are shown in table 5.

Table 3. Results of test case 1

No.	Selection for Parents depends on	Initial population	Result	Fitness value for Precision Recall
1	Precision	Q_1	(((w_8 or w_2) or (w_{13} and w_8)) or ((w_8 xor w_2) or (w_{13} and w_8)))	1.250000 1.000000
			(((w_8 or w_2) or (w_8 or w_2)) or (w_8 and w_2))	1.250000 1.000000
	Recall		((w_{13} or w_8) or (w_6 or w_2))	1.083333 1.000000
			((w_{13} and w_8) or ((w_6 or w_2) or ((w_{13} or w_8) or (w_6 or w_2))))	1.083333 1.000000
2	Precision	Q_2	((w_8 xor w_2) or (((w_{13} and w_8) or (w_8 xor w_2)) or (w_8 xor w_2)) or (w_8 xor w_2)))	1.250000 1.000000
			((w_8 or w_2) or ((w_{13} and w_8) and (w_8 or w_2)))	1.250000 1.000000
	Recall		(((w_8 xor w_2) or (w_8 xor w_2)) or (w_8 xor w_2))	1.250000 1.000000
			((w_{13} and w_8) or (w_8 xor w_2))	1.250000 1.000000

Table 4. Results of test case 2

No.	Selection for Parents depends on	Initial population	Result	Fitness value for Precision Recall
3	Precision	Q_1	((w_8 or w_2) or ((w_2 and w_4) or ((w_8 or w_2) and w_1)))	1.250000 1.000000
			((w_2 and w_4) or (((w_2 and w_4) xor (((w_2 and w_4) or (((w_4 or w_{10}) and (w_8 and w_{13})) or (w_8 and w_{13}))) and (w_8 and w_{13})) or (w_8 and w_{13}))) and (w_8 and w_{13})))	1.169580 0.678322
	Recall		(((w_2 or w_4) or (((w_2 or w_4) or (w_6 or w_2)) or (w_6 and w_2))) or (w_6 and w_2))	1.074290 0.930070
			(((w_2 or w_4) or (((w_2 or w_4) or (((w_{13} and w_8) or (w_6 or w_2)) and (w_8 xor w_2))) or (w_6 or w_2))) or (((w_{13} and w_8) or (w_6 or w_2)) or (w_8 xor w_2)))	1.101190 1.000000
4	Precision	Q_2	((w_{13} and w_8) or (w_8 or w_2))	1.250000 1.000000
			((w_8 or (w_8 or w_2)) or ((w_{13} and w_8) or (w_8 xor w_2)))	1.250000 1.000000
	Recall		(((w_{13} xor w_8) or (w_8 or w_2)) or (w_4 and w_3))	1.138199 1.000000
			((w_8 or w_2) or (w_8 or w_2))	1.250000 1.000000

Table 5. Results of test case 3

No.	Selection for Parents depends on	Initial population	Result	Fitness value for Precision Recall
5	Precision	Q_1	((w_{13} and w_8) and (w_{13} and w_8))	1.045644 0.182575
			(((((w_6 and w_2) or ((w_6 and w_2) or (w_8 and w_{13}))) and (((w_6 and w_2) or (w_8 and w_{13})) or (w_8 and w_{13}))) or ((w_6 or w_2) or (w_8 and w_{13}))) and (((w_6 and w_2) or (w_8 and w_{13})) or (w_8 and w_{13})))	1.084435 0.337739
	Recall		(w_8 or w_2)	1.250000 1.000000
			((w_{13} or w_8) or (w_6 or w_2))	0.913958 1.000000
6	Precision	Q_2	((w_8 and w_2) or (w_8 xor w_2))	1.250000 1.000000
			(w_8 or (w_8 or (w_8 or w_2)))	1.250000 1.000000
	Recall		(((((w_{13} and w_8) or (w_8 or (w_8 or w_2))) or (w_8 or (w_8 or w_2))) or (w_8 or w_2))	1.250000 1.000000
			(((w_{13} or w_8) and (w_8 xor w_2)) or ((w_{13} or w_8) or (w_8 xor w_2)))	1.025921 1.000000

6 Conclusions

The results of this study suggests that the final population composed of individuals having the same strength (quality) will have the same precision and recall values. The best individual result was randomly chosen as best. We conclude that the quality of initial population was important to have best result of genetic programming process, and the less quality of initial population caused worse results. This could be seen when we chose parents based on precision, and therefore recall was very small for the large collection.

We concluded that in order to get good results, we choose parents depending on the recall fitness values than the precision fitness values, but that will increase the number of Boolean logical operators for the final best results.

For the future work we suggest to use less number of prefix tree by identifying the Boolean logical operators only without identify index for terms. For selecting the best individual with less number of Boolean operators and less number of terms instead of random selection if we got the final population with same strength. Modifying our system to work with different Boolean operators (of, adj, ... see in [4]), and extend this for fuzzy logic.

References

1. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. Addison Wesley, New York, 1999.
2. Freytag, Johann Christoph: A Rule-Based View of Query Optimization. Proceedings of ACM-SIGMOD, 1987, pp. 173-180.
3. Goldberg, David E.: Genetic Algorithms in Search, Optimization and Machine Learning. Reading, Massachusetts: Addison-Wesley, 1989.
4. Korfhage Robert R.: Information Storage and Retrieval. John Wiley & Sons, Inc. 1997.
5. Melanie M.: An Introduction to Genetic Algorithms. A Bradford Book The MIT Press 1999.
6. Kim, Won: On Optimizing an SQL-like Nested Query. ACM Transactions on Database Systems 7, 3 (September 1982), pp. 443-469.
7. Kraft, D.H., Bordogna, G., and Pasi, G.: Fuzzy Set Techniques in Information Retrieval, in Bezdek, J.C., Didier, D. and Prade, H. (eds.), Fuzzy Sets in Approximate Reasoning and Information Systems, vol. 3, The Handbook of Fuzzy Sets Series, Norwell, MA: Kluwer Academic Publishers, 1999.
8. McGoveran, David: "Evaluating Optimizers." Database Programming and Design. January 1990, pp. 38-49.
9. Salton, G. and Buckley, C.: Terms-Weighting approach in automatic text retrieval. Information Processing and management, 1988 24(5):513-523.
10. Suhail S. J. Owais.: Timetabling of Lectures in the Information Technology College at Al al-Bayt University Using Genetic Algorithms. Master thesis, Al al-Bayt University 2003, Jordan. (in Arabic).
11. Yao, S. Bing: "Optimization of Query Algorithms." ACM Transactions on Database Systems 4, 2 (June 1979), pp. 133-155.

Author Index