Charles University in Prague, MFF, Department of Software Engineering
Czech Technical University in Prague, FEE, Department of Computer Science
VŠB–TU Ostrava, FEECS, Department of Computer Science
Czech Society for Cybernetics and Informatics

Proceedings of the Dateso 2010 Workshop

DATESO 2010

Databases, Texts, Specifications, and Objects

http://www.cs.vsb.cz/dateso/2010/
http://www.ceur-ws.org/Vol-567/

ARG
AMPHORA RESEARCH GROUP

Sponsored by

MIR Labs

ČSKI

http://www.mirlabs.org/ http://www.cski.cz/

April 21 – 23, 2010
Štědronín - Plazy

Technical editor:
Pavel Moravec, pavel.moravec@vsb.cz
VŠB – Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Computer Science

Page count:	198
Publication:	310th
Impression:	150
Edition:	1st
First published:	2010

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

Published by MATFYZPRESS publishing house of Faculty of Mathematics and Physics
Charles University in Prague, Sokolovská 83, 186 75 Praha 8, Czech Republic
as its 310th publication.

Preface

DATESO 2010, the international workshop on current trends on Databases, Information Retrieval, Algebraic Specification and Object Oriented Programming, was held on April 21 – 23, 2010 in in pension Fousek, Štědronín - Plazy.

The 10$^{\mathrm{th}}$ jubilee year was organized by Department of Software Engineering MFF UK Praha, Department of Computer Science and Engineering FEL ČVUT Praha, Department of Computer Science VŠB-Technical University Ostrava, and the Working group on Computer Science and Society of Czech Society for Cybernetics and Informatics. The DATESO workshops aim for strengthening connections between these various areas of Computer science. The proceedings of DATESO 2010 are also available at DATESO Web site: `http://www.cs.vsb.cz/dateso/2010/` and CEUR Workshop Proceeding site: `http://www.ceur-ws.org/Vol-567/` (ISSN 1613-0073). The Program Committee selected 15 papers (11 full papers and 4 posters) from 26 submissions, based on two independent reviews.

The workshop program also included 2 invited lectures: Database Trends and Directions: Current Challenges and Opportunities by George Feuerlicht, and Content-based Retrieval of Compressed Images by Gerald Schaefer.

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 for preparation of workshop and its proceedings as well as the Czech Society for Cybernetics and Informatics and MIR Labs for their support of publishing this issue.

Special thanks go to P. Moravec who, as copy editor of DATESO Proceedings, prepared this volume.

April, 2010 J. Pokorný, V. Snášel, K. Richta (Eds.)

Table of Contents

Parametrised Hausdorff Distance as a Non-Metric Similarity Model for Tandem Mass Spectrometry*

Jiří Novák and David Hoksza

Department of Software Engineering, Faculty of Mathematics and Physics,
Charles University in Prague,
Malostranské nám. 25, 118 00, Prague 1, Czech Republic
{novak, hoksza}@ksi.mff.cuni.cz

Abstract. Tandem mass spectrometry is a widely used method for protein and peptide sequences identification. Since the mass spectra contain up to 80% of noise and many other inaccuracies, there still exists a need for more accurate algorithms for mass spectra interpretation.
The sizes of protein databases grow rapidly and the methods for indexing these databases in order to interpret mass spectra become very popular. The parametrised Hausdorff distance, suitable for non-metric search, is presented in this paper. It models the similarity among tandem mass spectra very well and it is able to match the spectrum to correct peptide sequence in many cases without any post-processing scoring system.

Keywords: tandem mass spectrometry, metric access methods, peptide identification, bioinformatics

1 Introduction

Tandem mass spectrometry [8] is a fast and popular method for determining protein sequences from an experimentally prepared protein sample. Protein sequences identified by mass spectrometry are used in many fields of biological research especially in methods for protein structure and function prediction [18].

Definition 1. *Protein sequence is a linear sequence (of amino acids) over alphabet α of 20 letters, where α contains all letters from English alphabet except* $\{B, J, O, U, X, Z\}$[1].

Mass spectrometry does not determine sequences directly but the collection of data to be interpreted is obtained from tandem mass spectrometer. Each protein molecule in the sample is digested into peptides (short pieces of proteins)

* This research has been supported in part by Czech Science Foundation (GAČR) project Nr. 201/09/0683 and by institutional research plan number MSM0021620838.

[1] The omitted letters may sometimes represent more than one amino acid if there is no chance to differentiate them.

by an enzyme before mass analysis. The most common and cheap enzyme is trypsin and it digests protein after each[2] amino acid K (lysine) and R (arginine) if they are not followed by P (proline) [17].

Each peptide gets a charge z in a mass spectrometer and it becomes peptide ion[3]. Peptide ions are separated by their ratio mass m (also called precursor mass) and charge z, and then they are splitted to many peptide fragment ions. The dataset obtained from the tandem mass spectrometer is a list of mass spectra (one spectrum for each detected peptide ion). Precursor mass and charge can be provided as an additional information for each spectrum corresponding to a peptide ion. The process of assigning a corresponding peptide sequence to an experimental spectrum is denoted as mass spectrum interpretation.

Definition 2. *Mass spectrum is represented by a list of peaks. Each peak corresponds to a peptide fragment ion and it is a pair of numbers m/z and intensity of occurrence, where m denotes mass in Daltons*[4] *and z charge.*

An experimentally obtained mass spectrum (acquired by division of peptide ions in mass spectrometer) usually contains many noise peaks (up to 80%), which correspond to ions with very complicated and upredictable chemical structure. The intensity may help to differentiate between more and less significant peaks in such spectrum. Spectra generated from database of protein sequences (see section 2) can be denoted as a hypothetical or theoretical. The intensity cannot be determined from sequences for peaks in a hypothetical spectrum. The hypothetical spectra do not contain intensity which usually does not cause a problem, since the m/z ratio provides the main information for mass spectra interpretation.

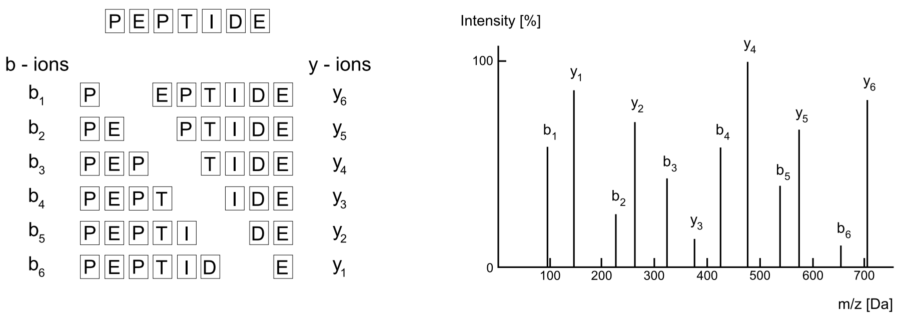


Fig. 1. Mass spectrum of sequence *PEPTIDE*.

[2] The digestion process is not perfect in practice so there can be some missed cleavage sites.

[3] Neutral molecules are not captured by mass spectrometer.

[4] Dalton (Da) is a unit of atom relative mass.

There are several types of fragment ions in a mass spectrum, which are fundamental for correct peptide sequence identification. The most frequently occurring are y-ions and b-ions[5] (Fig. 1). A ion serie is created by each type of ions. The completeness of y-ions or b-ions series determines the quality of the interpretation because the difference between two neighboring peaks in one serie corresponds to the mass of an amino acid. For example, missing of y_3 and b_4 in Fig. 1 causes loosing the information on the order of the letters T and I. The letters can be determined from the difference of m/z between y_2 and y_4 (or b_3 and b_5), but more candidate pairs of amino acids having similar aggregate m/z value can be selected from 20^2 possible amino acids pairs.

Modifications of amino acids are also a common problem when mass spectra are interpreted because other chemical groups can be attached to the amino acids in proteins. This usually happens during sample preparation for the mass analysis or in the mass spectrometer. The most common modifications are e.g. carbamidation of cysteine C (+57.01 Da) or oxidation of methionine M (+16 Da)[6]. The database UNIMOD [26] gathers discovered protein modifications for mass spectrometry. At the time of writing this paper, there were more than 620 known modifications.

2 State of the Art

Tandem mass spectra interpretation employs two basic approaches. *Ab initio*, the first approach, is based on direct mass spectra interpretation using graph algorithms and it is usually called as *De Novo* peptide sequencing [4]. This approach is highly influenced by occurrence of complete ion series because missed y-ions or b-ions can cause that there are many paths in graph and it is difficult to assign correct peptide sequence to the spectrum. The quality of identification using this approach is about 30% [9].

The other approach is based on search in the database [21] of already known or predicted[7] protein sequences. The hypothetical spectra of peptides are generated from database of protein sequences and compared with an experimental spectrum. A combined approach, Sequence Tag, was presented in [14]. First, a short amino acid sequence (tag) is determined by hand or by graph algorithm and then a database is searched. The most common tools for peptide identification based on searching in databases are SEQUEST [23], MASCOT [12], ProteinProspector [19], OMSSA [6], etc.

The number of data in protein databases grows exponentially every year [7] and sequential scan of the whole database becomes too slow. Modeling an index is not a trivial problem due to the noise, modifications and inaccuracies in mass spectra.

[5] Ion types are defined by the positions where splitting occurs.

[6] Special types of modifications are posttranslation modifications (PTM), which arise additionally after translation of DNA to proteins.

[7] It is possible to use raw translation of DNA sequences to protein sequences, so unknown protein sequences can be determined.

The naive method is based on indexing and querying mass spectra by their precursor mass using B-tree [2]. But there can be complications if the experimental spectrum contains modifications because m/z values of peaks and also precursor mass are shifted. The lengths of peptide sequences are usually about a few tens of amino acids. Looking for peptides with modifications can cause selection of many peptides from the database because a wide interval for precursor mass tolerance must be set up.

Several more sophisticated approaches were presented. One of them uses a suffix tree [25] for preprocessing the protein sequence database and a graph algorithm is used to preprocess tandem mass spectrum [11]. Then the suffix tree is searched against spectrum graph for candidate peptides. The correct peptide sequence is determined by a scoring function (such as HMM [27], dynamic programming [9], SEQUEST-like scoring [23], etc.).

Another method is based on using a self-organizing map (SOM) which is a type of neural network [15]. The hypothetical spectra are converted to high-dimensional vectors (Ex. 1) and then SOM is trained. The experimental spectrum is then used for a range query on SOM and the peptide candidate set is obtained and a scoring function is applied.

Example 1. Let the range of m/z values in the mass spectrum be 0-2,000 Da and let it be divided in subintervals of 0.1 Da. Each mass spectrum is then represented by a 20,000 dimensional boolean feature vector having ones at places corresponding to intervals for which m/z value in the spectrum exists.

There are also database approaches based on the properties of metric space [28]. One of them uses locality sensitive hashing in Euclidean space to preprocess peptides in the database followed by range query [5]. Another method is based on using cosine similarity and MVP-tree [20]. Using variants of cosine similarity (1) and representation of mass spectra as a high-dimensional vector (Ex. 1) is common idea in mass spectrometry literature [1].

Cosine of an angle is not a metric (see section 4) but it can be turned into metric by using arccos function. The approach based on MVP-tree [20] uses two alternatives of cosine distance. The first is called fuzzy cosine distance and it is generalization of (1). The other is called tandem cosine distance and it is combination of the fuzzy cosine distance and the precursor mass filter. Comparison of our method with this approach is presented in section 5.3.

$$\cos(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x}\boldsymbol{y}}{\|\boldsymbol{x}\| \, \|\boldsymbol{y}\|} \tag{1}$$

3 Original Idea and Improvements

3.1 Original Idea

The Hausdorff distance d_H (2) and logarithmic distance d_L (3) were proposed in [16] for tandem mass spectra interpretation. These distances describe the similarity among tandem mass spectra better than e.g. Euclidean d_E or maximum distance.

The advantage of using Hausdorff distance is that components on different positions in two vectors can be compared. The main idea of using logarithmic distance is that two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ are closer considering peptide identification if there are great differences in a small number of their components than if there are small differences in a large number of their components (Ex. 2).

$$d_H(\boldsymbol{x},\boldsymbol{y}) = \max(h(\boldsymbol{x},\boldsymbol{y}), h(\boldsymbol{y},\boldsymbol{x})),\quad h(\boldsymbol{x},\boldsymbol{y}) = \max_{x_i \in \boldsymbol{x}} \left\{ \min_{y_j \in \boldsymbol{y}} \{d_E(x_i, y_j)\} \right\} \tag{2}$$

$$d_L(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{k} \begin{cases} \log|x_i - y_i|, & |x_i - y_i| > 1 \\ 0, & otherwise \end{cases} \tag{3}$$

Example 2. Lets assume vectors of m/z values $\boldsymbol{x} = \{148, 263, 376, 477, 574, 703\}$, $\boldsymbol{y_1} = \{148, 263, 476, 477, 574, 703\}$ and $\boldsymbol{y_2} = \{140, 270, 370, 477, 570, 710\}$. The Euclidean distance between vectors $\boldsymbol{x}$ and $\boldsymbol{y_1}$ is $d_E(\boldsymbol{x},\boldsymbol{y_1}) = 100$ and the distance between vectors $\boldsymbol{x}$ and $\boldsymbol{y_2}$ is $d_E(\boldsymbol{x},\boldsymbol{y_2}) \doteq 14.6$ but the vectors $\boldsymbol{x}$ and $\boldsymbol{y_1}$ are closer considering peptide identification. The missing number 376 in $\boldsymbol{y_1}$ means that corresponding peak in the mass spectrum is missing. On the other hand the superfluous number 476 in $\boldsymbol{y_1}$ refers to the occurrence of similar 477. The replacement of values 376 and 476 can be observed as a consequence of these inaccuracies.

The vectors of m/z values were splitted by a sliding window to many shorter vectors of constant size in order to increase quality of identification. For example a sorted vector of 12 m/z values can be generated for sequence *PEPTIDE*, these numbers correspond to y and b-ions (Fig. 1). The $(l-1)*2-dim+1 = 10$ vectors must be indexed for one peptide sequence of length $l = 7$ and for vectors of dimension $dim = 3$. The short vectors were indexed by M-tree. The number of correctly assigned peptide sequences to the mass spectra was about 50-60% by using Hausdorff or logarithmic distance.

3.2 Improvements

Functions such as n^{th} root or logarithm [16] are suitable for the purpose of modeling similarity between mass spectra because these can significantly decrease an error caused by outliers.

The proposed parametrised Hausdorff distance d_{HP} (5) combine the characteristics of the n^{th} root function and Hausdorff distance, $\boldsymbol{x}$ and $\boldsymbol{y}$ are vectors of m/z values, d_E is Euclidean distance, n is index of the root and m is power modifier. The Hausdorff distance allows comparison of vectors with different sizes, which is valuable for peptide sequence identification because the mass spectra (hypothetical or experimentally obtained) have different number of peaks.

$$h(\boldsymbol{x},\boldsymbol{y}) = \frac{\sum_{x_i \in \boldsymbol{x}} \sqrt[n]{\left(\min_{y_j \in \boldsymbol{y}} \{d_E(x_i, y_j)\}\right)}}{|\boldsymbol{x}|} \tag{4}$$

$$d_{HP}(\boldsymbol{x},\boldsymbol{y}) = (\max(h(\boldsymbol{x},\boldsymbol{y}), h(\boldsymbol{y},\boldsymbol{x})))^m \tag{5}$$

Using d_{HP} noticeably increases accuracy even if no pre-processing nor post-processing algorithms are employed. Typical pre-processing algorithm is a heuristic which selects the most suitable peaks for peptide sequence identification from an experimental spectrum. The post-processing algorithm is usually represented by a scoring function which selects the best peptide sequence corresponding to an experimental spectrum from the peptide sequence candidate set obtained by an index structure.

Another improvement is significant reduction of the number of vectors that are generated from protein sequences. Only one vector of m/z values is necessary for peptide sequence representation which makes this method more usable. This was not possible in the previous version of the algorithm, since d_H and d_L required splitting in order to achieve sufficient quality of identification.

The time complexity for d_{HP} computation is $O(n^2)$ but since the lists of peaks are implicitly sorted by m/z ratio so an improvement is used and complexity $O(n)$ is achieved. The asymmetric part (see Alg. 1) of the Hausdorff distance can be computed using two nested loops. The inner loop can be broken if the minimum difference between components in two vectors is found. The position of minimum is stored and it is used as starting value for inner cycle in the next outer cycle (Alg. 1, line 3), *errTol* is mass error tolerance, *root(x,n)* is $\sqrt[n]{x}$, *power(x,m)* is x^m and *abs* computes the difference between two m/z values (in the Euclidean distance d_E).

Alg. 1. Parametrised Hausdorff Distance

```
1   float ComputeAsymmetric(sortedVector X,sortedVector Y,float errTol,float n) {
2     float sum = 0; int mem_j = 0;
3     for(int i=0;i<X.size();i++) {
4       /* position of component with minimum difference in the inner cycle
5       is >= than position in previous inner cycle */
6       min = abs(X[i]-Y[mem_j]);
7       for(int j=mem_j+1;j<Y.size();j++) {
8         if (abs(X[i]-Y[j]) < min) {
9           min = abs(X[i]-Y[j]);
10          mem_j = j;
11        }
12        /* minimum difference is achieved and better result cannot be found */
13        else break;
14      }
15      sum += (min>errTol)?root(min-errTol,n):0;
16    }
17    return sum/X.size();
18  }
19
20  float Compute(sortedVector X,sortedVector Y,float errTol,float n,float m) {
21    float left = computeAsymmetric(X,Y,errTol,n);
22    float right = computeAsymmetric(Y,X,errTol,n);
23    if (left > right) return power(left,m);
24    return power(right,m);
25  }
```

4 Metric Access Methods (MAMs)

The metric is a function that satisfies reflexivity, symmetry, non-negativity and triangle inequality [28]. Function which partially corrupts the triangle inequality is called a semimetric and the search process is denoted as non-metric [24]. The MAMs [28] were designed for fast search in databases modeled in metric spaces. The triangle inequality is crucial for organizing objects into metric regions and for pruning those regions while searching. MAM used in our experiments is a metric tree (M-tree) [3] but it can be replaced with any other MAM. MAMs support using range and k-NN (k-nearest neighbor) queries.

5 Experiments

The dataset from *Keller et al.* [10] was used in our experiments. The spectra were obtained by mixing 18 proteins together.[8] These spectra were identified by SEQUEST [23] and the results were manually checked. The spectra with charge 1^+ and 2^+, digested by trypsin and with corresponding peptide sequences contained in attached protein sequences file were selected. This file was used as a database *Keller1* containing 103 protein sequences (7,391 peptide sequences). The database *Keller2* is an extension of *Keller1* where protein sequences from MSDB (Mass Spectrometry Protein Sequence Database) [13] were added. The *Keller2* contains 10,000 protein sequences (649,481 peptide sequences). The databases and the query set from [20] were also used for comparison with cosine similarity (see section 5.3).

Following qualities were measured. The quality of identification is a ratio of correctly assigned peptide sequences to the mass spectra to all spectra from the query set (without differentiating the position of the correct peptide sequence in the obtained set). The distance computations ratio is the average number of runs of Alg. 1 per one mass spectrum to the sequential access. Since the real time is directly proportional to the distance computations ratio, we mostly use the ratio in the following text. The triangle inequality ratio is an empirically determined number of triplets of vectors satisfying the triangle inequality. The distance distribution histogram (DDH) [24] shows distribution of distances between any two vectors in the database. The distances on the x axis are normalised in order to be able to compare histograms with different values of n or m (e.g. Fig. 2b). The normalisation is possible because maximum mass of generated peptides is limited. The distance frequency is the number of pairs of vectors in the distance $d \pm \delta$ in the database, where δ is an error caused by rounding.

All experiments were carried out on a 1.6 GHz processor AMD TURION TL52 with 2 GB RAM and OS Windows XP SP2. Following settings were used unless otherwise specified - digestion enzyme: trypsin, maximum missed cleavage sites: 1, mass error tolerance: 0.4 Da, y and b-ions were generated in hypothetical spectra, 100 peaks with highest intensity were selected from experimental spectra, mass range of generated peptides: 500-5,000 Da.

[8] The 119 spectra from the first run on mixture A were used.

5.1 Index of the Root

First experiments concerned the influence of the index of n^{th} root function ($n = \{1, 2, 5, 10, 20, 50, 100\}$) on the quality of peptide sequence identification and the suitability of the parametrised Hausdorff distance for use with MAMs. Settings: $m = 1$ (modifier is off), DDHs measured on *Keller1*, quality of identification measured on *Keller2* (sequential access was employed for *Keller2*).

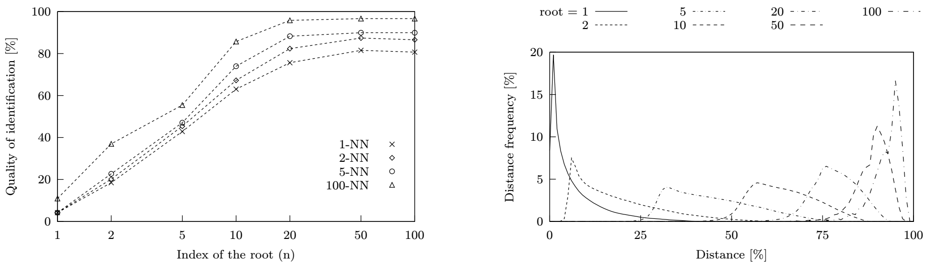


Fig. 2. Index of the root - a) quality of identification, b) DDHs.

The quality of identification increases with increasing n and the distance models the similarity among tandem mass spectra very well. The correct peptide sequences were assigned to more than 80% of experimentally obtained mass spectra as a result of 1-NN query for $n = 50$ (Fig. 2a). The number of correctly assigned sequences was about 90% for 5-NN query and more than 96% for 100-NN query. We need a 669-NN query for achieving 100% quality of identification. The selectivity is about 0.1% in such a case. The average time for the identification of one mass spectrum was about 14.4 seconds.

The triangle inequality ratio is about 17% for $n = 1$ and about 99% for $n = 2$ and higher. A disadvantage is that intrinsic dimensionality [24] gets higher with increasing n hence the distance computations ratio increases. For high n, the difference between MAMs and sequential access blends. The intrinsic dimensionality is indicated by DDH (Fig. 2b).

5.2 The Power Modifier

We tried to solve the problem of high intrinsic dimensionality by using power modifier m (5) due to poor MAMs usability. The power is monotonous function and it does not change the order of the results. The index performance (Fig. 4) was tested on M-tree with database *Keller2.*

The DDH improves with increasing modifier (Fig. 3a). Modifiers were tested for $n = 50$ (see Table 1). The DDH with $m = 1$ (modifier is off) is shown for comparison. The triangle inequality ratio gets worse with increasing power modifier (Fig. 3b). The experiments were executed for different n (see section 5.1). The quality of identification gets better with increasing triangle inequality ratio (Fig. 4a) but the distance computations ratio gets worse (Fig. 4b).

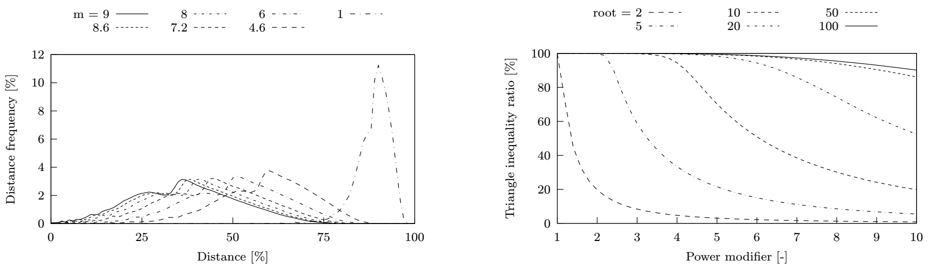


Fig. 3. The power modifier - a) corrections of DDH, b) triangle inequality ratio.

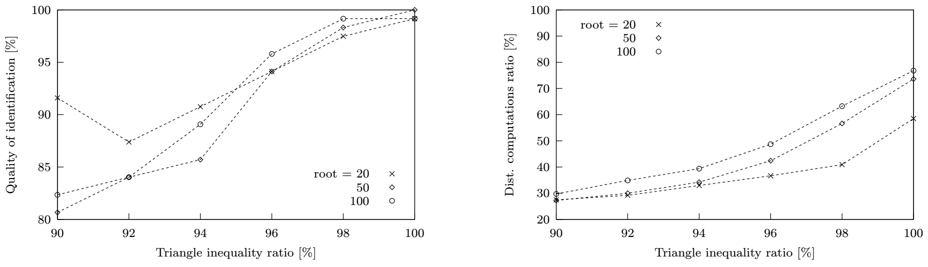


Fig. 4. The power modifier - a) quality of identification, b) dist. computations ratio.

Triangle inequality ratio [%]	90	92	94	96	98	100
$n = 20$	6.6	6.4	6	5.6	5.2	4
$n = 50$	9	8.6	8	7.2	6	4.6
$n = 100$	10	9.2	8.6	7.6	6.2	4.8

Table 1. Empirically determined modifiers m.

5.3 Comparison with the Cosine Similarity

Parametrised Hausdorff distance was compared with fuzzy cosine distance and tandem cosine distance described in [20]. Datasets described in Table 1 in the cited paper were used for the comparison. The database I contains 92,768 hypothetical spectra from the genome of Escherichia Coli K12 and 7 proteins mixture from Sashimi proteomics repository [22]. The database II has 654,276 spectra and it is an extension of database I containing hypothetical spectra from human genome. The query set contains 49 experimentally obtained spectra and comes from the 7 proteins mixture from Sashimi proteomics repository. The following settings were used: $n = 1000$, $m = 4$, 0 missed cleavage sites, error 1.0 Da, y and b ions were generated in hypothetical spectra, 100 peaks with highest intensity were selected from experimental spectra, peptide mass range 0-5,000 Da. We used 13-NN query on M-tree and the triangle inequality ratio was 99.9%.

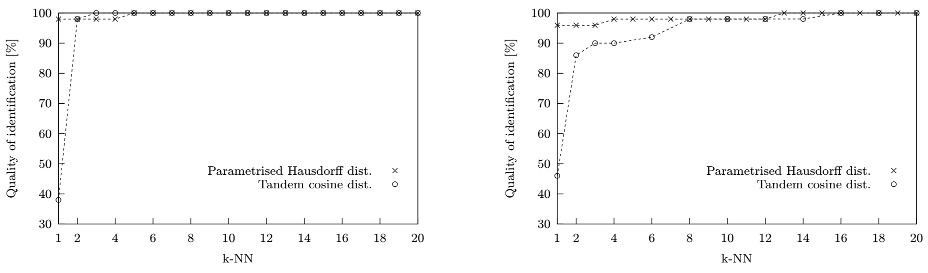


Fig. 5. Quality of identification - a) database I, b) database II.

The parametrised Hausdorff distance returns peptide sequence corresponding to the experimental spectrum as a result of 1-NN query in 98% on database I and in 95.9% on database II. The quality of identification eliminates the need of a scoring system (see section 2). But in fact the quality decreases with increasing database size and the scoring system cannot be completely removed from a real-world application.

The parametrised Hausdorff distance has better quality of identification than tandem cosine distance (Fig. 5)[9]. But in fact the tandem cosine distance has the distance computations ratio less than 0.3% for both databases and parametrised Hausdorff distance has the computations ratio 62.3% for the database I and 50.7% for the database II. Although the computations ratio of our method decreases with increasing database size, it is still slower than the tandem cosine distance. Fuzzy cosine distance has the distance computations ratio about 95%. Tandem cosine distance's computation ratio is a consequence of combination fuzzy cosine distance and precursor mass filter [20]. The precursor mass filter can be restrictive criterion if the peptide modifications are searched. Typical precursor mass tolerance is about ± 2 Da. This tolerance must be extended for searching peptides with modifications. Precursor mass of modified peptides can differ by more than a few tens to hundreds Daltons.

5.4 Non-Metric Search and k-NN Queries

An interesting characteristic can be observed when non-metric search is used. We examined the performance of the M-tree (*Keller2*) using $n = 50$ and $m = 9$ which corresponds to 90% triangle inequality ratio. The k in k-NN query was increased and the quality of identification grew. The results were not distributed uniformly over the interval of k items but the correct peptide sequences were found as the top hits in many cases (Fig. 6a). This is a consequence of non-metricity and it cannot happen if the distance is metric or if the sequential access is used. The distance computations ratio and average time of identification per one spectrum grew with increasing k in k-NN query (Fig. 6b). Average time was about 15.2 seconds for sequential access.

[9] The results for tandem cosine distance were taken from the supplement of [20].

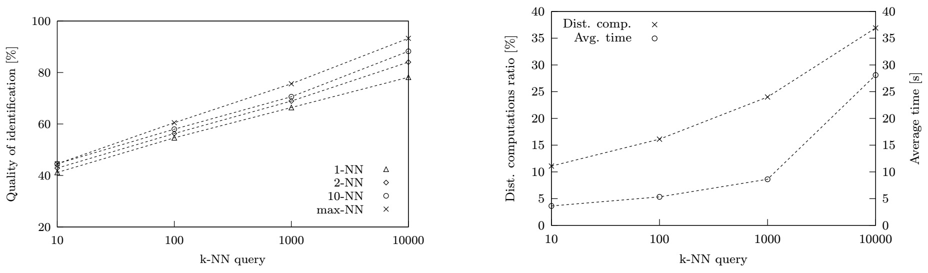


Fig. 6. Non-metric search and k-NN queries - a) quality of identification, b) distance computations ratio and average time.

6 Conclusions and Future Work

The parametrised Hausdorff distance for interpretation tandem mass spectra of peptides was proposed. It was compared with cosine distance which is widely discussed in mass spectrometry literature. The fuzzy and tandem cosine distance were used in this paper. Tandem cosine distance shows worse results in terms of quality identification than our algorithm. The fuzzy approach is moreover slower in terms of distance computations ratio. Higher speed of tandem cosine distance is a consequence of including the precursor mass filter. On the other hand, embedding of precursor mass filter can be problematic when modeling of the similarity of spectra corresponding to modified peptides is desired. Development of more precise semimetrics can also reduce the need of complicated scoring algorithms. The design of better modifier functions for parametrised Hausdorff distance opens possibilities for further research. Finally, the abilities of k-NN query for non-metric search were presented.

References

1. Z.B. Alfassi. On the normalization of a mass spectrum for comparison of two spectra. *Journal of the American Society for Mass Spectrometry*, vol. 15, issue 3, pp. 385-387. 2004.
2. R. Bayer and E.M. McCreight. Organization and Maintenance of Large Ordered Indices. *Acta Inf.*, vol. 1, pp. 173-189. 1972.
3. P. Ciaccia, M. Patella and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. *Proc. of 23rd Int. Conf. on VLDB*, pp. 426-435. 1997.
4. V. Dančík, T.A. Addona, K.R. Clauser, J.E. Vath and P.A. Pevzner. De Novo Peptide Sequencing via Tandem Mass Spectrometry. *Journal of Computational Biology*, vol. 6, no. 3, pp. 327-342. 1999.
5. D. Dutta and T. Chen. Speeding up tandem mass spectrometry database search: metric embeddings and fast near neighbor search. *Bioinformatics Oxford Journal*, vol. 23, no. 5, pp. 612-618. 2007.

6. L.Y. Geer et al. Open Mass Spectrometry Search Algorithm. *Journal of Proteome Research*, vol. 3, pp. 958-964. 2004.
7. D. Hoksza and T. Skopal. Index-based approach to similarity search in protein and nucleotide databases. *CEUR Proc. Dateso 2007*, vol. 235, pp. 67-80. 2007.
8. D.F. Hunt et al. Protein sequencing by tandem mass spectrometry. *Proc. Nati. Acad. Sci. USA*, vol. 83, pp. 6233-6237. 1986.
9. N.C. Jones and P.A. Pevzner. *An Introduction to Bioinformatics Algorithms.* MIT Press, Cambridge, Massachusetts. 2004.
10. A. Keller et al. Experimental Protein Mixture for Validating Tandem Mass Spectral Analysis. *Journal of Integrative Biology*, vol. 6, no. 2, pp. 207-212. 2002.
11. B. Lu and T. Chen. A suffix tree approach to the interpretation of tandem mass spectra: applications to peptides of non-specific digestion and post-translational modifications. *Bioinformatics Oxford Journal*, vol. 19 (Suppl. 2), pp. 113-121. 2003.
12. MASCOT. `http://www.matrixscience.com/`.
13. Mass Spectrometry Protein Sequence Database (MSDB).
`http://www.proteomics.leeds.ac.uk/bioinf/msdb.html`.
14. E. Mortz et al. Sequence tag identification of intact proteins by matching tandem mass spectral data against sequence data bases. *Proc. Natl. Acad. Sci. USA*, vol. 93, pp. 8264-8267. 1996.
15. K. Ning, H.K. Ng and H.W. Leong. PepSOM: An Algorithm for Peptide Identification by Tandem Mass Spectrometry based on SOM. *Genome Informatics*, vol. 17, pp. 194-205. 2006.
16. J. Novák and D. Hoksza. An Application of the Metric Access Methods to the Mass Spectrometry Data. *IEEE CIBCB 2009.* Nashville, TN, USA. ISBN 978-1-4244-2756-7, pp. 220-227.
17. J.V. Olsen, S. Ong and M. Mann. Trypsin Cleaves Exclusively C-terminal to Arginine and Lysine Residues. *Molecular and Cellular Proteomics*, vol. 3, pp. 608-614. 2004.
18. G.A. Petsko and D. Ringe. *Protein Structure and Function (Primers in Biology).* New Science Press Ltd, London, UK. 2004.
19. ProteinProspector. `http://prospector.ucsf.edu/`.
20. S.R. Ramakrishnan et al. A fast coarse filtering method for peptide identification by mass spectrometry. *Bioinformatics Oxford Journal*, vol. 22, no. 12, pp. 1524-1531. 2006.
21. R.G. Sadygov, D. Cociorva and J.R. Yates III. Large-scale database searching using tandem mass spectra: Looking up the answer in the back of the book. *Nature Methods*, vol. 1, no. 3, pp. 195-202. 2004.
22. Sashimi proteomics repository.
`http://sashimi.sourceforge.net/repository.html`.
23. SEQUEST. `http://fields.scripps.edu/sequest/`.
24. T. Skopal. Unified Framework for Fast Exact and Approximate Search in Dissimilarity Spaces. *ACM Transactions on Database Systems (TODS)*, vol. 32, issue 4. 2007.
25. E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, vol. 14, pp. 249-260. 1995.
26. UNIMOD. `http://www.unimod.org/`.
27. Y. Wan, A. Yang and T. Chen. PepHMM: A Hidden Markov Model Based Scoring Function for Mass Spectrometry Database Search. *Anal. Chem.*, vol. 78, pp. 432-437. 2006.
28. P. Zezula, G. Amato, V. Dohnal and M. Batko. *Similarity Search: The Metric Space Approach (Advances in Database Systems).* Springer, New York, USA. 2006.

Two-step Modified SOM for Parallel Calculation[*]

Petr Gajdoš and Pavel Moravec

Department of Computer Science, FEECS, VŠB – Technical University of Ostrava,
17. listopadu 15, 708 33 Ostrava-Poruba, Czech Republic
{petr.gajdos, pavel.moravec}@vsb.cz

Abstract. This paper presents a simple modification of classic Kohonen network (SOM), which allows parallel processing of input data vectors or partitioning the problem in case of insufficient memory for all vectors from the training set. The algorithm pre-selects potential centroids of data clusters and uses them as weight vectors in the final SOM network. We have demonstrated the usage of this algorithm on images as well as on two well-known datasets representing handwritten digits.

Keywords: SOM, Kohonen Network, parallel calculation, handwritten digits

1 Introduction

With the massive boom of GPU-based calculations, massive parallelism, memory considerations, simplicity of algorithms and CPU-GPU interaction have yet again to play an important role. In this paper, we present a simple modification of classic Kohonen's self-organizing maps (*SOM*), which allows us to dynamically scale the computation to fully utilize the GPU-based approach.

There were some attempts to introduce parallelism in Kohonen networks [4,6,5,7,8], however we needed an approach which is simple and easy to implement. Moreover, it should work both with and without the bulk-loading algorithm [2].

In this paper, we present such approach, which divides the training set into several subsets and calculates the weights in multi-step approach. Calculated weights with nonzero number of hits serve as input vectors of SOM network in the following step. Presently, we use a two-step approach, however more steps could be used if necessary.

The paper is organized as follows: in second chapter we mention classic SOM networks and describe the basic variant we have used. In third chapter we describe our approach and provide the calculation algorithm. The fourth chapter introduces experimental data we have used and presents the results of comparison of results provided by our method with classic SOM calculation.

2 Kohonen self-organizing neural network

In following paragraphs, we will shortly describe the Kohonen self-organizing neural networks (self-organizing maps – *SOM*). The first self-organizing networks were pro-

[*] This work was partially supported by SGS, VŠB-Technical University of Ostrava, No. SP/2010196 grant – Machine Intelligence

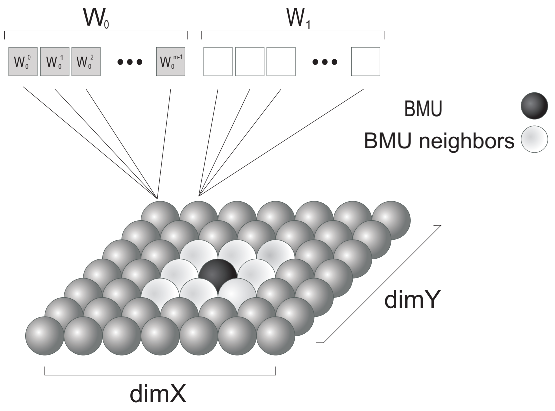


Fig. 1. Kohonen network structure

posed in the beginning of 70's by Malsburg and his successor Willshaw. SOM was proposed by Teuvo Kohonen in in the early 1980s and has been improved by his team since. The summary of this method can be found in [3].

The self-organizing map is one of the common approaches on how to represent and visualize data and how to map the original dimensionality and structure of the input space onto another – usually lower-dimensional – structure in the output space.

The basic idea of SOM is based on the human brain, which uses internal 2D or 3D representation of information. We can imagine the input data to be transformed to vectors, which are recorded in neural network. Most neurons in cortex are organized in 2D. Only the adjacent neurons are interconnected.

Besides of the input layer is in SOM only the output (competitive) layer. The number of inputs is equal to the dimension of input space. Every input is connected with each neuron in the grid, which is also an output (each neuron in grid is a component in output vector). With growing number of output neurons, the quality coverage of input space grows, but so does computation time.

SOM can be used as a classification or clustering tool that can find clusters of input data which are more closer to each other.

All experiments and examples in this paper respect following specification of the SOM (see also the Figure 1):

- The SOM is initialized as a network of fixed topology. The variables $dimX$ and $dimY$ are dimensions of such 2-dimensional topology.
- V^m represents an m-dimensional input vector.
- W^m represents an m-dimensional weight vector.
- The number of neurons is defined as $N = dimX * dimY$ and every neuron $n \in < 0, N-1 >$ has its weight vector W_n^m
- The neighborhood radius r is initialized to the value $min(dimX, dimY)/2$ and will be systematically reduced to a unit distance.
- All weights vectors are updated after particular input vector is processed.

– The number of epochs e is know at the beginning.

The Kohonen algorithm is defined as follows:

1. **Network initialization**
   All weights are preset to a random or pre-calculated value. The learning factor η, $0 < \eta < 1$, which determines the speed of weight adaptation is set to a value slightly less than 1 and monotonically decreases to zero during learning process. So the weight adaptation is fastest in the beginning, being quite slow in the end.
2. **Learning of input vector**
   Introduce k training input vectors $V_1, V_2, \ldots, V_k$, which are introduced in random order.
3. **Distance calculation**
   An neighborhood is defined around each neuron whose weights are going to change, if the neuron is selected in competition. Size, shape and the degree of influence of the neighborhood are parameters of the network and the last two decrease during the learning algorithm.
4. **Choice of closest neuron**
   We select the closest neuron for introduced input.
5. **Weight adjustment**
   The weights of closest neuron and its neighborhood will be adapted as follows:
   $$W_{ij}(t+1) = W_{ij}(t) + \eta(t)h(v,t)(V_i - W_{ij}(t)),$$
   where $i = 1, 2, \ldots, dimX$ a $j = 1, 2, \ldots, dimY$ and the radius r of neuron's local neighborhood is determined by adaptation function $h(v)$.
6. **Go back to point 2 until the number of epochs e is reached.**

To obtain the best organization of neurons to clusters, a big neighborhood and a big influence of introduced input are chosen in the beginning. Then the primary clusters arise and the neighborhood and learning factor are reduced. Also the $\eta \to 0$, so the changes become less significant with each iteration.

3 Proposed method

The main steps of SOM computation have been already described above. Following text is focused on description of proposed method, that in the end leads to results similar to the classic SOM (See also Figure 2 for illustration of our approach). We named the method Global-Merged SOM, which suggests, that the computation is divided into parts and then merged to obtain the expected result. Following steps describe the whole process of GM-SOM:

1. **Input set split**
   The set of input vectors is divided into a given number of parts. The precision of proposed method increases with the number of parts, however, it has own disadvantages related to larger set of vectors in the final phase of process. Thus the number of parts will be usually determined from the number of input vectors. Generally,

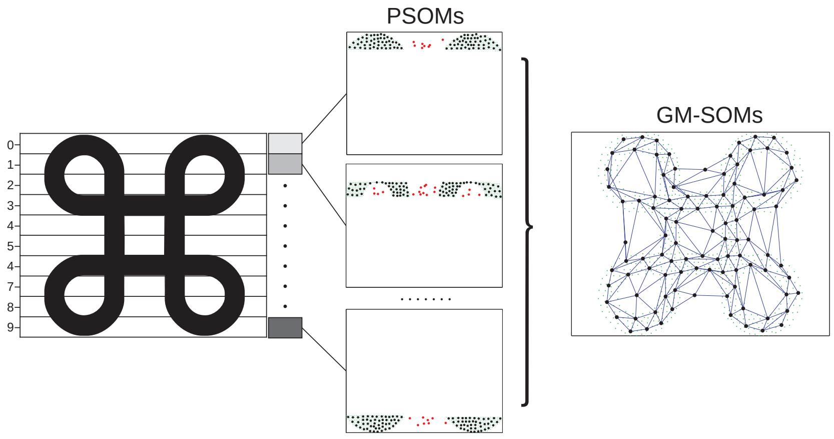


Fig. 2. GM-SOM: An Illustrative schema of the proposed method. All input vectors are divided into ten parts in this case.

$k \gg N * p$, where k is the number of input vector, N is the number of neurons and p is the number of parts. The mapping of input vectors into individual parts does not affect final result. This will be later demonstrated by the experiments, where all the input vectors were either split sequentially (images) or randomly (sets of handwritten digits).

2. **In parts computation**
   Classic SOM method is applied on every part. For simplicity sake, an acronym *PSOM* will be used from now on to indicate SOM, which is computed in a given part. All PSOMs start with the same setting (the first distribution of weights vectors, number of neurons, etc.) Such division speeds up parallel computation of PSOMs on GPU. Moreover, the number of epochs can be lower in comparison with the number of epochs in case of processing of input set by one SOM. This is represented by a factor f, which is going to be equal to $\frac{1}{3}$ in our experiments.
3. **Merging of parts**
   Weight vectors, that where computed in every part and correspond with neurons with at least one hit, represent input vectors in the final phase of GM-SOM. The unused neurons and their weight vectors have red color in Figure 2. Again, a new SOM with the same setting is computed and output weights vectors make the final result of proposed method.

The main difference between the proposed algorithm and well known batch SOM algorithms is, that individual parts are fully independent on each other and they update different PSOMs. Moreover, different SOM algorithms can be applied on PSOM of a given part, which makes proposed algorithm more variable. Next advantage can be seen in different settings of PSOMs. Thus more dense neuron network can be used in case of larger input set. The last advantage consists in a possibility of incremental updating of GM-SOM. Any additional set of input vectors will be processed by a new PSOM in a separate part and the final SOM will be re-learnt.

4 Experiments

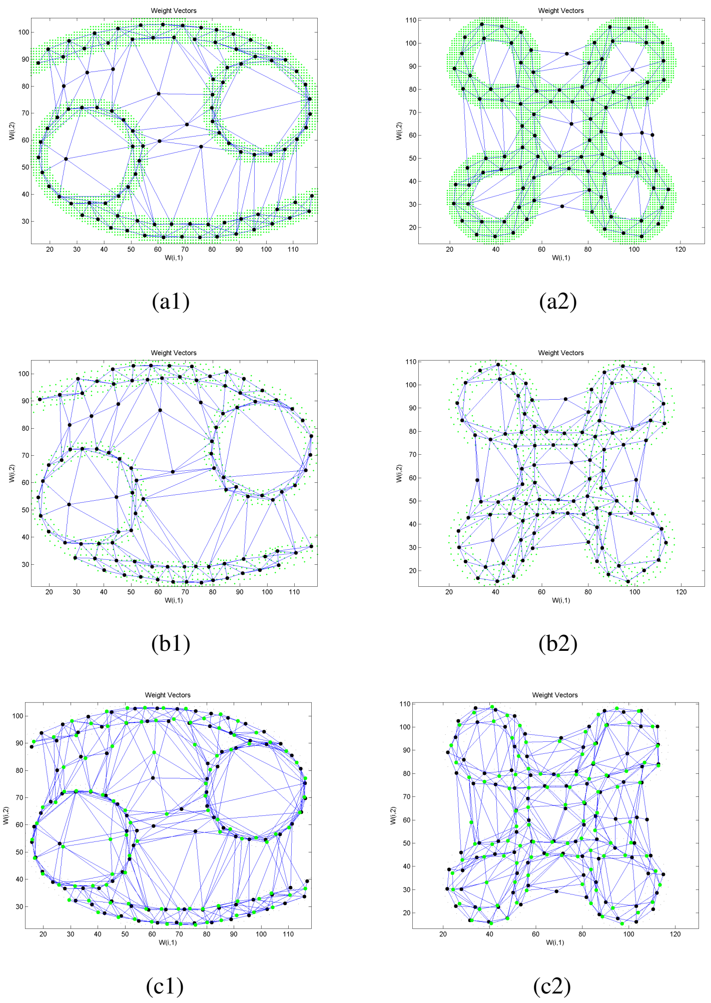


Fig. 3. Black-and-white symbols – (a) Original, (b) Per-partes, (c) Overlaid results

Our approach has been tested both on a generic set of black-and-white images as well as on two well-known datasets used for machine learning which were obtained from UCI repository [1].

In this section, we present examples of such results for a 10×10 hexagonal SOM with 300 iterations for both the original SOM network and the final calculation. Each partial SOM in our approach used 100 iterations ($f = \frac{1}{3}$). Moreover, where reasonable, a 20×20 hexagonal SOM will be used.

In the first set of experiments, we have generated a collection of black-and-white images, based on simple geometric shapes and symbols. Coordinates of black pixels were considered the input vectors for this experiment. Given a sufficient number of iterations, the weight vectors of Kohonen self-organizing neural network are known to spread through the image and cover the black areas.

We have compared the original method with our approach on a wide selection of shapes, ranging from simple convex shapes to complicated symbols consisting of several parts. Each symbol was tested several times to reduce the impact of random initiation of the network weights. Our approach typically provided results which were marginally better than the classic SOM. Some of the results are shown in Figure 3.

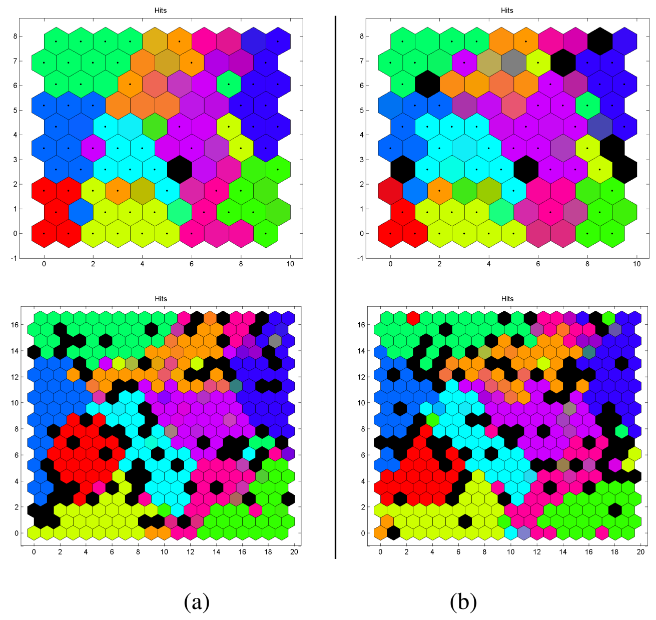
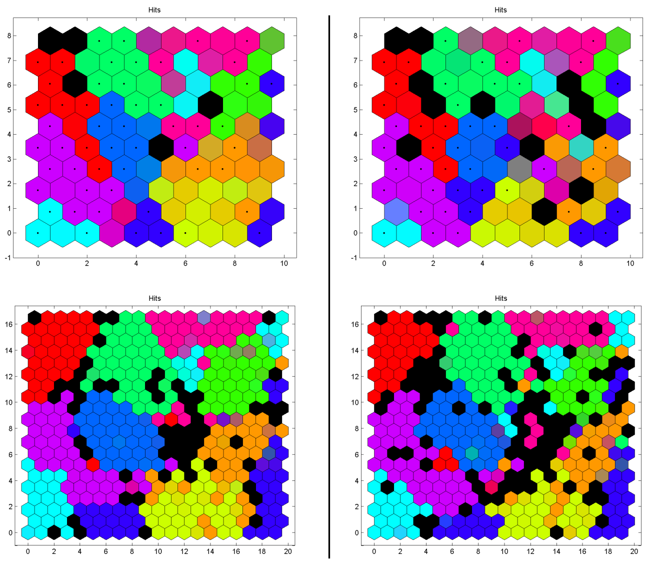


Fig. 4. OCR of handwritten digits: (a) SOM hits in case of classic approach, (b) SOM hits of proposed method. Black cells represent neurons without any hits.

For the second experiment, we have used *Optical Recognition of Handwritten Digits Data Set* from UCI repository, containing 43 sets of hand-written digits from different people. 30 sets of digits contributed to the training set and different 13 to the test set. The digits were scanned as 32×32 bitmaps, which were then divided into 4×4 blocks. The number of on pixels of each block has been recorded (to reduce the dimension and reduce the impact of small distortions) resulting in 64 features with values ranging from 0 to 16. The results for all existing classes in original and our approach are shown in Figure 6. Figure 4 shows the combined result based on SOM hits for the test set in both 10×10 and 20×20 networks. The results are again comparable for both methods.



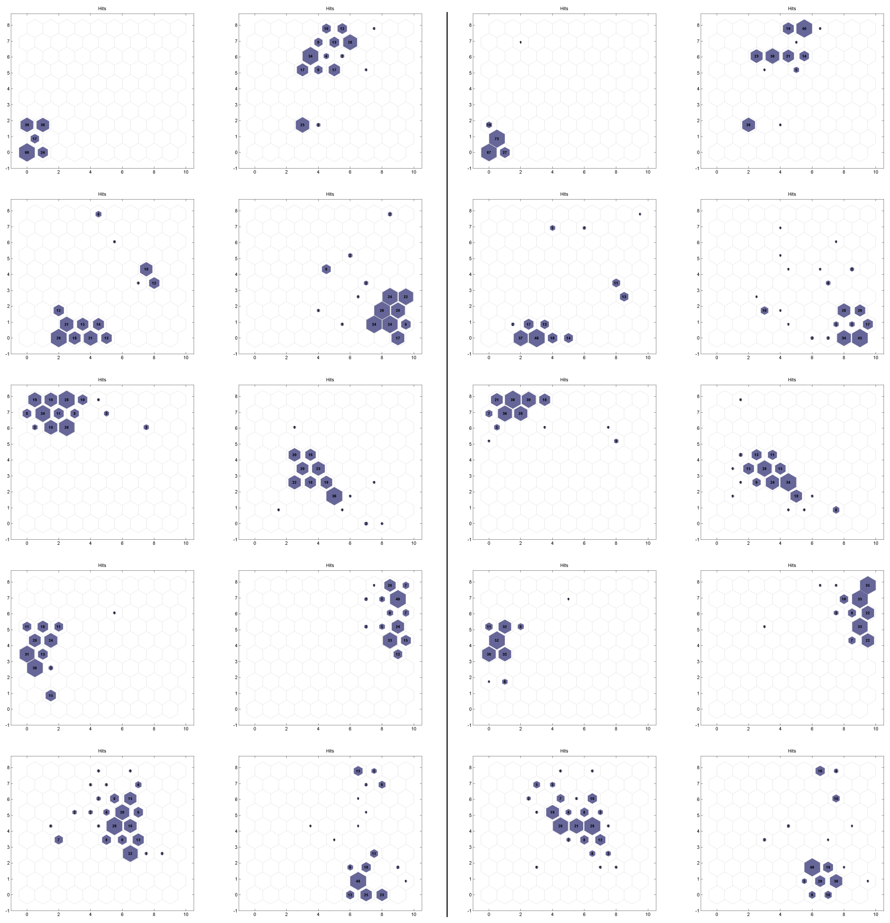
(a) (b)

Fig. 5. Pen digits: (a) SOM hits in case of classic approach, (b) SOM hits of proposed method. Black cells represent neurons without any hits.

The third experiment was conducted on *Pen-Based Recognition of Handwritten Digits Data Set*. The data set consists of 250 samples from 44 writers with 30 writers were used for training and the remaining digits written by the other 14 writers were used for testing. The recorded points were re-sampled to 8 x, y coordinates. The combined result based on SOM hits for the test set in both 10×10 and 20×20 networks for original and our approach is shown in Figure 5.

5 Conclusion

The need of parallel computation of SOM drove us to a new method, that has been presented in this paper. Although it has some common features with well known SOM batch or hierarchical algorithms, it is not one of them, as it has its unique properties.



(a) Classic SOM (b) GM-SOM

Fig. 6. OCR of handwritten digits: Particular classes of neurons, that indicate digits 0–9 are illustrated separately.

Firstly, the proposed algorithm can utilize the power of batch processing in all inner parts (PSOMs). Moreover, all PSOMs can have different number of neurons in their networks, which could be found in hierarchical algorithms. Lastly, our method excludes neurons, which do not cover any input vectors in the intermediate phase of GM-SOM.

All experiments suggest, that the results are very close to results provided by classic SOM algorithm. We would like to test the proposed algorithm on huge data collections in the near future.

References

1. A. Asuncion and D. Newman. UCI machine learning repository, 2007.
2. J. Fort, P. Letremy, and M. Cottrel. Advantages and drawbacks of the batch kohonen algorithm. In *10th-European-Symposium on Artificial Neural Networks, Esann'2002 Proceedings.*, pages 223–230, 2002.
3. T. Kohonen. *Self-Organizing Maps*. Springer Verlag, Berlin, second (extended) edition, 1997.
4. R. Mann and S. Haykin. A parallel implementation of Kohonen's feature maps on the warp systolic computer. In *Proc. IJCNN-90-WASH-DC, Int. Joint Conf. on Neural Networks*, volume II, pages 84–87, Hillsdale, NJ, 1990. Lawrence Erlbaum.
5. T. Nordström. Designing parallel computers for self organizing maps. In *Forth Swedish Workshop on Computer System Architecture*, 1992.
6. S. Openshaw and I. Turton. A parallel kohonen algorithm for the classification of large spatial datasets. *Computers & Geosciences*, 22(9):1019–1026, November 1996.
7. I. Valova, D. Szer, N. Gueorguieva, and A. Buer. A parallel growing architecture for self-organizing maps with unsupervised learning. *Neurocomputing*, 68:177 – 195, 2005.
8. L. Wei-gang. A Study of Parallel Self-Organizing Map. In *Proceedings of the International Joint Conference on Neural Networks*, 1999.

Answering Metric Skyline Queries by PM-tree

Tomáš Skopal and Jakub Lokoč

Department of Software Engineering, FMP
Charles University in Prague, Czech Republic
{skopal, lokoc}@ksi.mff.cuni.cz

Abstract. The task of similarity search in multimedia databases is usually accomplished by range or k nearest neighbor queries. However, the expressing power of these "single-example" queries fails when the user's delicate query intent is not available as a single example. Recently, the well-known skyline operator was reused in metric similarity search as a "multi-example" query type. When applied on a multi-dimensional database (i.e., on a multi-attribute table), the traditional skyline operator selects all database objects that are not dominated by other objects. The metric skyline query adopts the skyline operator such that the multiple attributes are represented by distances (similarities) to multiple query examples. The metric skyline is supposed to constitute a set of representative database objects which are as similar to all the examples as possible and, simultaneously, are semantically distinct. In this paper we propose a technique of processing the metric skyline query by use of PM-tree, while we show that our technique significantly outperforms the original M-tree based implementation in both time and space costs.

1 Introduction

As the volumes of complex unstructured data collections grow almost exponentially in time, the attention to content-based similarity search steadily increases. The concept of numeric similarity between two data entities is one of the approaches used for querying unstructured data, where a similarity function serves as a multi-valued relevance of data objects to a query (example) object. The content-based similarity search paradigm has been successfully employed in areas like multimedia databases, time series retrieval, bioinformatic and medical databases, data mining, and others. At the same time, the "similarity-centric" view on such data demands specific alternative techniques for modeling, indexing and retrieval, which dramatically differ from the traditional approaches to management of structured data (e.g., B-trees in relational databases).

In the rest of the section we introduce into the fundamentals of similarity search and briefly summarize the paper contributions.

1.1 Similarity search

Given a collection $\mathcal{C}$ of unstructured data entities (e.g., multimedia objects, like images), to query the collection we need to establish a model consisting of the object *universe* $\mathbb{U}$, a *transformation* function (a feature extraction method, resp.) $t : \mathcal{C} \rightarrow \mathbb{U}$, and a *similarity* function $\delta : \mathbb{U} \times \mathbb{U} \rightarrow \mathcal{R}$. The transformation t turns the collection $\mathcal{C}$ of original data entities into a *database of descriptors* $\mathbb{S} \subset \mathbb{U}$. In most cases the similarity function δ is expected to be a *metric* distance, because metric properties can be effectively used to index the database $\mathbb{S}$ for efficient (fast) query processing, as discussed later in Section 1.2.

Single-example queries. The portfolio of available similarity query types consists of mostly single-example queries. The *range* query and the *k nearest neighbor* (kNN) query represent the two most popular similarity query types. Using a range query (Q, r_Q) we ask for all objects $O_i \in \mathbb{S}$ the distances of which to a single query object Q are at most r_Q. On the other hand, a kNN query (Q, k) selects the k database objects closest to Q.

Besides range and kNN queries, there exist some less frequently used query types, like *reverse (k)NN queries* [15], returning those database objects having the query object Q within their (k) nearest neighbor(s).

Multi-example queries. Although the single-example queries are frequently used nowadays, their expressive power may become unsatisfactory in the future due to increasing complexity and quantity of available data. The acquirement of a query (example) object is the user's "ad-hoc" responsibility. However, when just a single example should represent the user's delicate intent on the subject of retrieval, finding such an example could be a hard task. Hence, instead of querying by a single example, an easier way for the user could be a specification of several query examples which jointly describe the query intent. Such a multi-example approach allows the user to set the number of query examples and to weigh the contribution of individual examples. Moreover, obtaining multiple examples, where each example corresponds to a partial query intent, is much easier task than finding a single "holy-grail" example.

In this paper we deal with metric skyline query (detailed in the Section 2), which represents a "native" multi-example query type.

1.2 Metric access methods

When the similarity function δ is a distance metric, the *metric access methods* (MAMs) can be used for efficient (fast) similarity query processing [16, 11, 2]. The principle behind all MAMs is the utilization of metric postulates (positiveness, symmetry and triangle inequality), which allow to partition the data space into equivalence classes of close (similar) data objects. The classes are embedded within a data structure which is stored in an *index file*, while the index is later used to quickly answer range, kNN, or other similarity queries. In particular, when issued a similarity query, the MAMs exclude many non-relevant equivalence classes from the search (based on metric properties of δ), so only several candidate classes of objects have to be exhaustively (sequentially) searched. In consequence, searching a small number of candidate classes turns out in reduced cost of the query. The number of *distance computations* $\delta(\cdot, \cdot)$ is considered as the major component of the overall costs when indexing or querying a database. Some other cost components (like *I/O costs*, internal *CPU costs*) could be taken into consideration when the computational complexity of δ is low.

In the following we briefly describe the M-tree and the PM-tree, two MAMs used further in the paper for implementation of metric skyline queries.

M-tree. The *M-tree* [5] is a dynamic metric access method that provides good performance in database environments. The M-tree index is a hierarchical structure, where some of the data objects are selected as centers (references or local *pivots*) of ball-shaped regions, and the remaining objects are partitioned among the regions in order to build up a balanced and compact hierarchy, see Figure 1.

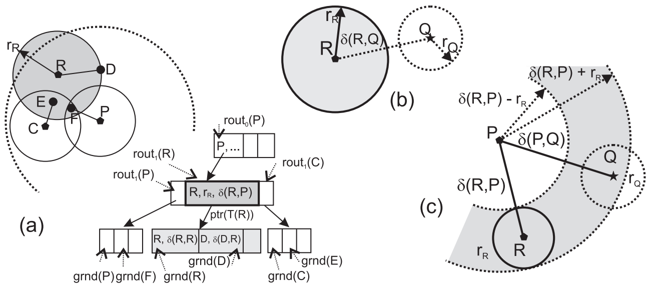


Fig. 1. (a) M-tree (b) Basic filtering (c) Parent filtering.

Each region (subtree) is indexed recursively in a B-tree-like (bottom-up) way of construction. The inner nodes of M-tree store *routing entries*

$$rout_M(R) = [R, r_R, \delta(R, \mathrm{Par}(R)), ptr(T(R))]$$

where R is a data object representing the center of the respective ball region, r_R is a *covering radius* of the ball, $\delta(R, Par(R))$ is so-called *to-parent* distance (the distance from R to the object P of the parent routing entry), and finally $ptr(T(R))$ is a pointer to the entry's subtree $T(R)$. In order to correctly bound the data in $T(R)$'s leaves, the routing entry must satisfy the *nesting condition*: $\forall O_i \in T(R), r_R \geq \delta(R, O_i)$. The data is stored in the leaves of M-tree. Each leaf contains *ground entries*

$$grnd_M(D) = [D, id(D), \delta(D, \mathrm{Par}(D))]$$

where D is the data object itself (externally identified by $id(D)$), and $\delta(D, Par(D))$ is, again, the to-parent distance. See an example of entries in Figure 1a.

The queries are implemented by traversing the tree, starting from the root. Those nodes are accessed, the parent regions of which are overlapped by the query region, e.g., by a range query ball (Q, r_Q). The check for region-and-query overlap requires an explicit distance computation $\delta(R, Q)$ (called *basic filtering*). In particular, if $\delta(R, Q) \leq r_Q + r_R$, the data ball (R, r_R) overlaps the query

(Q, r_Q), thus the child node has to be accessed, see Figure 1b. If not, the respective subtree is filtered from further processing. Moreover, each node in the tree contains the distances from the routing/ground entries to the center of its parent routing entry (the to-parent distances). Hence, some of the M-tree branches can be filtered without the need of a distance computation, thus avoiding the "more expensive" basic overlap check. In particular, if $|\delta(P, Q) - \delta(P, R)| > r_Q + r_R$, the data ball R cannot overlap the query ball (called *parent filtering*), thus the child node has not to be re-checked by basic filtering, see Figure 1c. Note $\delta(P, Q)$ was already computed at the unsuccessful parent's basic filtering.

PM-tree. The idea of PM-tree [12, 14] is to enhance the hierarchy of M-tree by an information related to a static set of p global pivots $P_i \in \mathcal{P} \subset \mathbb{U}$. In a PM-tree's routing entry, the original M-tree-inherited ball region is further cut off by a set of *rings* (centered in the global pivots), so the region volume becomes more compact – see Figure 2a. Similarly, the PM-tree ground entries are enhanced by distances to the pivots, which are interpreted as rings as well. Each ring stored in a routing/ground entry represents a distance range (bounding the underlying data) with respect to a particular pivot. A routing entry in PM-tree inner node is defined as:

$$rout_{PM}(R) = [R, r_R, \delta(R, \text{Par}(R)), ptr(T(R)), \text{HR}],$$

where the new HR attribute is an array of p_{hr} intervals ($p_{hr} \leq p$), where the t-th interval HR_{P_t} is the smallest interval covering distances between the pivot P_t and each of the objects stored in leaves of $T(R)$, i.e. $\text{HR}_{P_t} = \langle \text{HR}^{min}_{P_t}, \text{HR}^{max}_{P_t} \rangle$, $\text{HR}^{min}_{P_t} = min\{\delta(O_j, P_t)\}$, $\text{HR}^{max}_{P_t} = max\{\delta(O_j, P_t)\}$, $\forall O_j \in T(R)$. The interval HR_{P_t} together with pivot P_t define a ring region (P_t,HR_{P_t}); a ball region (P_t,$\text{HR}^{max}_{P_t}$) reduced by a "hole" (P_t,$\text{HR}^{min}_{P_t}$). A ground entry in PM-tree leaf is defined as:

$$grnd_{PM}(D) = [D, id(D), \delta(D, \text{Par}(D)), \text{PD}],$$

where the new PD attribute stands for an array of p_{pd} pivot distances ($p_{pd} \leq p$) where the t-th distance $\text{PD}_{P_t} = \delta(R, P_t)$.

The combination of all the p entry's ranges produces a p-dimensional minimum bounding rectangle (MBR), hence, the global pivots actually map the metric regions/data into a "pivot space" of dimensionality p (see Figure 2b).

When issuing a range or kNN query, the query object is mapped into the pivot space – this requires p extra distance computations $\delta(Q, P_i), \forall P_i \in \mathcal{P}$. The mapped query ball (Q, r_Q) forms a hyper-cube $\langle \delta(Q, P_1) - r_Q, \delta(Q, P_1) + r_Q \rangle \times \cdots \times \langle \delta(Q, P_p) - r_Q, \delta(Q, P_p) + r_Q \rangle$ in the pivot space that is repeatedly utilized to check for an overlap with routing/ground entry's MBRs (see Figures 2a,b). If they do not overlap, the entry is filtered out without any distance computation, otherwise, the M-tree's filtering steps (parent & basic filtering) are applied.

Note the MBRs overlap check does not require an explicit distance computation, so the PM-tree usually achieves significantly lower query costs when compared with M-tree – up to an order of magnitude (see [12–14]).

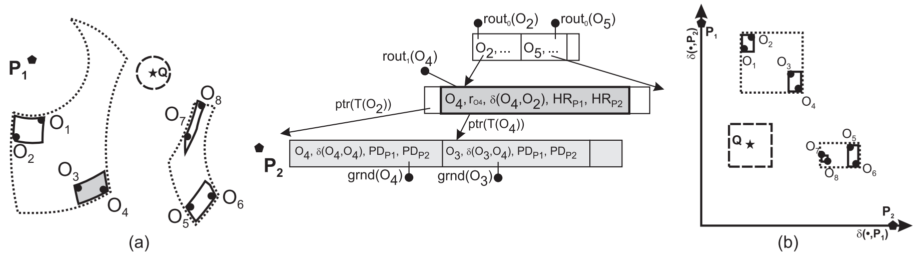


Fig. 2. (a) PM-tree employing 2 pivots (P_1, P_2). (b) Projection of PM-tree into the "pivot space".

1.3 Paper contributions

In this paper we introduce metric skyline processing by use of PM-tree. We follow the pioneer work [3] where the concept of metric skyline query was introduced, and its implementation utilizing M-tree was proposed. In Section 2 the metric skyline query and its original implementation is discussed, while in Section 3 we propose our original PM-tree implementation of metric skyline processing. In experimental results (Section 4) we show that PM-tree based metric skyline processing outperforms the original M-tree implementation not only in terms of distance computation costs, but also in terms of I/O costs, internal CPU costs and internal space costs.

2 Metric skyline queries

In relational databases, the multi-criterial retrieval is popular in situations where a query exactly specifying the desired attribute ranges cannot be effectively issued. Instead, there is a need for a simplified query concept which selects the desired database objects by some aggregation technique. Besides the top-k queries [6], a popular multi-criterial technique is the *skyline operator* [1].

2.1 The Skyline Operator

The traditional skyline operator is an advanced retrieval technique that selects objects from a multidimensional database that are "the best" from the global point of view. The only assumption on the database is that the attribute domains (dimensions) are linearly ordered, such that the lower (or higher) value of an attribute is, the better the object is (in that attribute). In the rest of the paper we use the convention that a lower value in an attribute is better.

The skyline operator selects all objects from the database (the *skyline set*), that are not *dominated* by any other object. An object O_1 dominates another object O_2 if at least one of O_1's attribute values is lower than the same attribute in O_2, and the other attribute values in O_1 are lower or equal to the corresponding attribute values in O_2. Hence, O_1 is the *dominating* object, while O_2 is the *dominated* object. In Figure 3a see an example of skyline set consisting of 5 objects, dominating the remaining 6 objects.

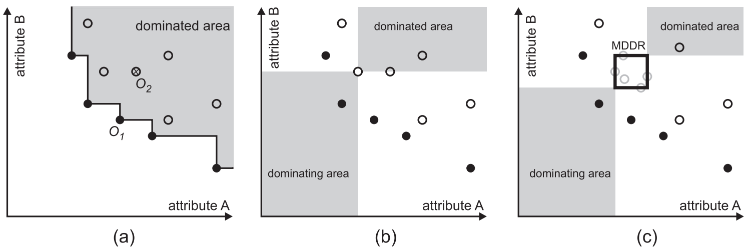


Fig. 3. (a) Skyline set and the dominated objects. A dominating-dominated (a) object and (b) rectangle (MDDR).

Skyline Processing. There exist many approaches to the efficient implementation of the skyline operator, while we outline two of them – the *Sort-First Skyline* algorithm [1] and the *branch-and-bound* algorithm which will be useful further in the paper.

In the *Sort-First Skyline* algorithm, the database objects O_i are just ordered ascendentally based on the L_1 norm on attributes (coordinates) of O_i, i.e., $||O_i||_{L_1} = O_i^1 + O_i^2 + \cdots + O_i^n$. Then, following the L_1 order, the sorted database is passed such that each visited object O_i is checked whether it is dominated by the already determined skyline objects. If O_i is not dominated, it is added to the skyline set (empty at the beginning), otherwise, O_i is ignored. After the one-pass database traversal is finished, the skyline set is complete. The algorithm is correct because of the L_1-norm ordering. Suppose an object O_i is being processed (see Figure 3b). Because every object possibly dominating O_i lies in the dominating area, its L_1 norm must be lower than that of O_i. However, such an object has already been visited (and possibly added to the skyline set) because of the ordered database traversal. Thus, O_i can be either safely added to the skyline set or filtered out.

The *branch-and-bound* approach employs a *spatial access method* (SAM), e.g. the R-tree [8]. The database is indexed by the SAM, while for the skyline processing a memory-resident *priority heap* is additionally utilized. The heap priority is defined, again, as the L_1 norm, however, besides the database objects themselves, the heap may contain also minimum bounding rectangles (MBRs, natively maintained by, e.g., R-tree). For future use outside the scope of SAM, we call MBRs as *minimum dominating-dominated rectangles* (MDDRs). The MDDRs serve as spatial rectangular approximations of the underlying database objects (or nested MDDRs), while they can be effectively used for filtering. The order of an MDDR within the heap is defined by the L_1 norm of its minimal corner (the point of MDDR with minimal values in all dimensions), which is the maximal lower bound to L_1 norm of any object inside the MDDR.

2.2 Metric Skyline Queries

The spatial skyline queries were generalized recently to support an arbitrary metric distance δ (i.e., not just Euclidean), constituting thus the *metric skyline queries* (MSQ) [3, 4].

Generally speaking, the metric skyline model just adds an abstract transformation step before the usual skyline processing. The step consists of transformation of a database in a metric space into database in m-dimensional vector space through a set $\mathcal{Q}$ of $m = |\mathcal{Q}|$ query examples. In the second step, the traditional skyline operator is performed on the transformed database. In particular, a database object O_i in the metric space is transformed into a vector V_i, where its j-th coordinate is defined as the distance from j-th query to O_i, i.e., $V_i = \langle \delta(Q_1, O_i), \delta(Q_2, O_i), \dots, \delta(Q_m, O_i) \rangle, Q_j \in \mathcal{Q}$.

Motivation. The motivation for MSQ can be seen in the insufficient expressive power of range and kNN queries, as mentioned in Section 1.1. Besides the possibility of employing multiple query examples, the metric skyline query has also another unique property, the absence of query extent, i.e., the query is defined just by the set $\mathcal{Q}$. This property could be seen as both advantage and disadvantage. The advantage is that metric skyline query returns all distinct objects from the database that are as similar to the query examples as possible. Hence, we obtain all such objects; we are freed from tuning the precision and recall proportion. Unfortunately, the disadvantage of MSQ is the skyline set (answer set) size. If $m = |\mathcal{Q}| = 1$ we obtain a regular 1-NN query. However, with increasing m the skyline size usually grows substantially, while a skyline set size exceeding several percent of the database is usually useless for an end-user. Thus, to be discriminative enough, the metric skyline query should employ only a few query examples (say, 2–5).

M-tree Based Implementation. The above described straightforward two-step abstraction is not suitable for implementation of MSQ. An explicit transformation of the original database $\mathbb{S}$ into a metric space would require expensive static preprocessing of the database, consisting of $|\mathcal{Q}| \cdot |\mathbb{S}|$ distance computations, extra storage costs, etc. Remember, the main cost component in similarity search by MAMs is the number of distance computations, so any MSQ algorithm should be designed to avoid computing as many distances as possible.

The authors of metric skyline queries proposed a native MSQ processing by M-tree [3, 4], where the transformation step was applied only on a part of the database that cannot be skipped during the processing. Basically, the M-tree based metric skyline algorithm was inspired by the traditional skyline processing by R-tree and the priority heap $\mathcal{H}$ under L_1 norm (as described in Section 2.1).

In the following we have re-formulated the original description in [3, 4] to the more abstract MDDR formalism, due to its easier extensibility to our original contribution in Section 3. The modification of R-tree based skyline processing to the metric case resides in an "on-the-fly" derivation of MDDRs, which cover the transformed data objects. Instead of "native" R-tree MDDRs (MBRs, resp.), we distinguish two types of derived MDDRs in M-tree, as follows:

(1) The *Par-MDDR* (parent MDDR) of a routing/ground entry $entry(R, r_R, \cdots)$, constructed by use of the parent routing entry $rout(P, \cdots)$ as $\text{MDDR}_{Par} = \langle LB^{Q_1}_{Par}, UB^{Q_1}_{Par}\rangle \times \cdots \times \langle LB^{Q_m}_{Par}, UB^{Q_m}_{Par}\rangle$, where $LB^{Q_i}_{Par}$ is a lower-bound distance from Q_i to the region (R, r_R) (through its parent P), while $UB^{Q_i}_{Par}$ is an upper-bound distance from Q_i to (R, r_R). Thus, $LB^{Q_i}_{Par} = max(\delta(Q_i, P) - (\delta(P, R) + r_R), (\delta(P, R) - r_R) - \delta(Q_i, P), 0)$, and $UB^{Q_i}_{Par} = \delta(Q_i, P) + \delta(P, R) + r_R$.

(2) The *B-MDDR* (basic MDDR), constructed directly from a routing/ground entry as $\text{MDDR}_B = \langle \delta(Q_1, R) - r_R, \delta(Q_1, R) + r_R\rangle \times \cdots \times \langle \delta(Q_m, R) - r_R, \delta(Q_m, R) + r_R\rangle$. In consequence, B-MDDR of ground entry is a single point.

Obviously, we have chosen the terms "Par-MDDR" and "B-MDDR" due to the analogy with parent- and basic filtering used when processing a range or kNN query in M-tree. The Par-MDDR of a routing/ground entry can be derived without an explicit distance computation; the $\delta(Q_i, P)$ distances were already computed during the top-down M-tree traversal. The derivation of B-MDDR is more expensive, it requires m computations of $\delta(R, Q_i), \forall Q_i \in \mathcal{Q}$.

An MDDR M_1 dominates all objects inside an MDDR M_2 if the L_1 norm of M_1's *maximal corner* is lower than the L_1 norm of M_2's *minimal corner*, where a max/min corner is the point with max/min values in all dimensions of an MDDR. For an example of Par-MDDR and B-MDDR, see Figure 4.

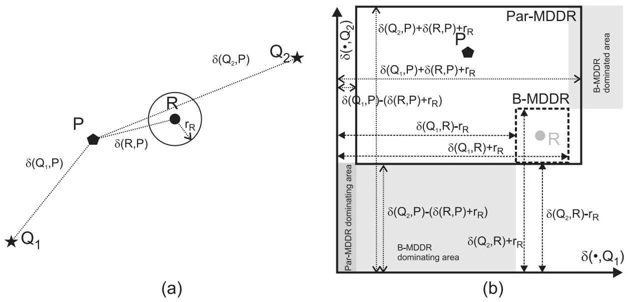


Fig. 4. (a) Metric space with M-tree regions (b) Transformed vector space with MDDRs

The MSQ algorithm starts by inserting routing entries from the M-tree root into the heap $\mathcal{H}$. The heap keeps order given by L_1 norm applied on the entries' B-MDDRs' minimal corners. Then a loop follows until the heap gets empty:

(1) An entry $entry(R, \ldots)$ with the lowest L_1 value of its B-MDDR is popped from the heap.
(2) If the entry is a ground entry, it is added to the set of skyline objects. All entries on the heap which are dominated by this new skyline object are removed. Jump to Step 1.
(3) If the entry is a routing entry, the entry's child node is fetched. The Par-MDDRs of the child node's entries are checked for dominance by the set of already determined skyline objects, while the dominated ones (and the respective subtrees, in case of routing entries) are filtered from further processing.

(4) The B-MDDRs of the non-filtered child entries are derived. Those entries not dominated by the already retrieved skyline set are inserted into the heap. Jump to Step 1.

Discussion. Unfortunately, in the original contribution [3, 4] the cost analysis and also the experiments were focused solely on measuring the number of dominance checks, i.e., how many times B-MDDRs and Par-MDDRs were checked for dominance by a skyline object. The authors completely ignored the number of distance computations (the crucial performance factor for any MAM), but also the heap size and the number of operations on heap, spent by running the metric skyline algorithm on M-tree.

As we present later in experimental evaluation, the M-tree based algorithm, as proposed in [3, 4], is extremely inefficient in terms of the heap size and the number of operations on the heap. In fact, the maximal heap size could reach the size of the database, making such an implementation inapplicable in database environments. In the following section we introduce our PM-tree based method, which not only decreases the number of distance computations spent for metric skyline processing, but also drastically decreases the maximal heap size and the number of operations on the heap.

3 PM-tree based metric skyline

The M-tree based approach to metric skyline processing can be extended to a PM-tree based implementation. In the following we introduce an algorithm that makes use of the PM-tree's extensions over the M-tree – the pivot set $\mathcal{P}$ and the respective ring regions maintained by routing/ground entries in PM-tree nodes (for PM-tree details see Section 1.2).

First of all, when a metric skyline query is started, a *query-to-pivot* matrix of pair-wise distances between the PM-tree pivots $P_i \in \mathcal{P}$ and query examples $Q_i \in \mathcal{Q}$ is computed. The PM-tree based algorithm (see Section 3.4) then utilizes the following three filtering concepts (Sections 3.1–3.3).

3.1 Deriving Piv-MDDRs

Besides the M-tree's B-MDDRs and Par-MDDRs derived from a routing/ground $entry(R, \cdots, \mathrm{HR/PD})$, an additional MDDR can be derived from the set of rings HR/PD maintained by the entry, called *Piv-MDDR* (pivot MDDR). The Piv-MDDR can be derived using the query-to-pivot matrix, as
$\mathrm{MDDR}_{Piv} = \langle LB^{Q_1}_{Piv}, UB^{Q_1}_{Piv}\rangle \times \cdots \times \langle LB^{Q_m}_{Piv}, UB^{Q_m}_{Piv}\rangle$, where
$\mathrm{LB}^{Q_i}_{Piv} = max_{P_j \in \mathcal{P}}\{\delta(P_j, Q_i) - \mathrm{HR}^{max}_{P_j}, \mathrm{HR}^{min}_{P_j} - \delta(P_j, Q_i), 0\}$, and
$\mathrm{UB}^{Q_i}_{Piv} = min_{P_j \in \mathcal{P}}\{\delta(P_j, Q_i) + \mathrm{HR}^{max}_{P_j}\}$.

Similarly as the M-tree's Par-MDDR, the derivation of Piv-MDDR requires no extra distance computation, however, Piv-MDDRs are much more compact than Par-MDDRs. This results in more effective filtering of routing/ground entries by skyline objects or some dominating MDDRs. Moreover, the Piv-MDDR is often even more compact than the direct B-MDDR, because the PM-tree's rings reduce the volume of the original M-tree's sphere. In Figure 5 see an example of Piv-MDDR, Par-MDDR and B-MDDR, when 2-pivot PM-tree and 2 query examples are used.

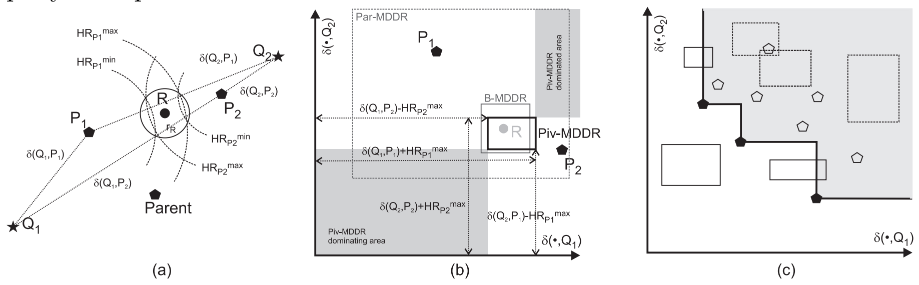


Fig. 5. A PM-tree routing entry in (a) metric space and (b) mapped to Piv-, Par-, and B-MDDR. (c) A pivot skyline.

3.2 Pivot-Skyline filtering

If the pivots P_i come from the database (i.e., $P_i \in \mathcal{P} \subset \mathbb{S}$), the MDDRs that are about to be inserted into the heap can be checked for a dominance by the pivots. Since the query-to-pivot matrix is computed at the beginning of every metric skyline query processing, the transformation of the pivots into the "query space" requires no additional distance computations. Moreover, to reduce the number of pivots used for dominance checking, we can determine the so-called *pivot skyline* – those pivot objects, which constitute a metric skyline within the pivot set $\mathcal{P}$ itself, see an example in Figure 5c.

The filtering by use of pivot skyline is beneficial in the early phase of the metric skyline processing, when the set of determined skyline objects is still empty. In the experiments we show that such an early phase is the dominant phase of the entire skyline processing – 80-90% of the total distance computations is performed before the first skyline object is found. Hence, pruning the heap by use of the pivot skyline greatly helps to reduce the heap size and, consequently, the number of operations on the heap. **Note:** As the number of determined skyline objects grows, the objects in the pivot skyline become dominated by the "regular" skyline objects. Hence, in order to effectively use the pivots for dominance checking, we keep just those pivots in the pivot skyline, that are not dominated by the already determined skyline objects. Thus, at the moment when all skyline objects are known, the pivot skyline becomes empty.

3.3 Deferred heap processing

In the original M-tree algorithm, the priority heap contains just L_1-ordered B-MDDRs (together with the associated routing/ground entries). When an entry is to be inserted into the heap, its B-MDDR must be determined, see Steps 3,4 of the algorithm in Section 2.2. We call this approach a *non-deferred heap processing.*

However, the non-deferred heap processing is not optimal in terms of the number of distance computations. In order to save some distance computations, we propose the *deferred heap processing* for the metric skyline, inspired by the Hjaltason's & Samet's incremental nearest neighbor algorithm, which is optimal in the number of distance computations [9]. The modified heap is generalized such that it may contain not only B-MDDRs of routing/ground entries, but also the intersections of their Piv-MDDR and Par-MDDR (denoted as Piv-MDDR $\cap$ Par-MDDR). The deferred heap processing then deals with two situations:

(1) An entry equipped by B-MDDR is popped from the heap. Then,
(a) If the entry is a ground entry, it becomes a skyline object.
(b) If the entry is a routing entry, its child node is fetched, while for every entry in the child node the Piv-MDDR $\cap$ Par-MDDR is checked for a dominance by the skyline set. Every not-dominated child entry is equipped by its Piv-MDDR $\cap$ Par-MDDR and inserted into the heap.
(2) An entry equipped by Piv-MDDR $\cap$ Par-MDDR is popped from the heap and checked for a dominance by the skyline set. If not dominated, the entry's B-MDDR is determined and, if still not dominated, inserted back into the heap.

Listing 1 *(Algorithm of PM-tree based metric skyline query)*

```
MSQuery()
{
Input: PM-tree $\mathcal{PM}$, query points $\mathcal{Q}$, type ('M-tree', 'PM-tree',
                                    'PM-tree+PSF', 'PM-tree+PSF+DEF')
Output: Result $\mathcal{MSS}$ containing skyline points

  if (type is not 'M-tree')
    P2Q_DM = evaluate the query-to-pivot matrix
    // pivots must be DB objects
    PSL = evaluate pivot skyline (using P2Q_DM)
  Insert all routing entries + their Piv-MDDR $\cap$ B-MDDR from the
                                    PM-tree root into the heap $\mathcal{H}$
  while ($\mathcal{H}$ is not empty)
    currentEntry = pop entry from the heap $\mathcal{H}$
    if (currentEntry is not equipped by 'B-MDDR')
      FilterAndInsert(currentEntry, currentEntry, type, true)
    else if (currentEntry is of type 'ground entry' and is equipped by
                                                        'B-MDDR')
      Insert currentEntry into $\mathcal{MSS}$
      $\mathcal{H}$.FilterDominatedObjectsBy(currentEntry.MDDR)
      PSL.FilterDominatedObjectsBy(currentEntry.MDDR)
    else
      $N$ = fetch child node of currentEntry
      for each childEntry in $N$
        FilterAndInsert(childEntry, currentEntry, type, false)
}

FilterAndInsert(newEntry, parentEntry, type, deferred)
{
  if (not deferred)
    Equip newEntry by its Par-MDDR
    if (type is not 'M-tree')
      Update newEntry.MDDR by intersection with newEntry's Piv-MDDR
  if (Filter(newEntry, type))
    return
  if (type = 'PM-tree+PSF+DEF' and not deferred)
    Insert newEntry into $\mathcal{H}$
    return
  Equip newEntry by its B-MDDR
  if (Filter(newEntry, type))
    return
  Insert newEntry into $\mathcal{H}$
}

Filter(newEntry, type)
{
  for each $O_i$ in $\mathcal{MSS}$
    if (newEntry.MDDR is dominated by $O_i$)
      return true
  if (type is 'M-tree' or 'PM-tree')
    return false
  for each $O_i$ in PSL
    if (newEntry.MDDR is dominated by $O_i$)
      return true
  return false
}
```

3.4 The algorithm

In Listing 1 the algorithm for metric skyline query is presented, including the original M-tree variant as well as the proposed PM-tree extensions.

The input attribute type allows to set the MSQ variant as follows: type = 'M-tree' is the original M-tree based algorithm, type = 'PM-tree' is the basic PM-tree based algorithm using the Piv-MDDR filtering (as described in Section 3.1), type = 'PM-tree+PSF' additionally uses the pivot-skyline filtering (as described in Section 3.2), and type = 'PM-tree+PSF+DEF' additionally uses the deferred heap processing (as described in Section 3.3).

4 Experimental evaluation

We performed an extensive experimentation with the three new variants of the PM-tree based metric skyline processing, comparing them against the original M-tree based method. Instead of the number of dominance checks (as included in the original contribution [3, 4]), we have observed other 4 measures of costs spent by the MSQ processing – the number of distance computations, the number of operations on the heap, the maximal allocated size of the heap, and finally the I/O costs.

In addition to the absolute numbers presented in the figures below, we also relate the number of distance computations spent by (P)M-tree MSQ processing to the costs of MSQ processed by simple *sequential search*, which takes $|\mathcal{Q}| \cdot |\mathbb{S}|$ distance computations for every query.

4.1 The testbed

We have used two databases, a subset of the *CoPhIR* database [7] of MPEG7 image features extracted from images downloaded from `flickr.com`, and a synthetic database of polygons. The CoPhIR database, consisting of one million feature vectors, was projected into two subdatabases, the *CoPhIR_12* database, consisting of 12-dimensional color layout descriptors, and the *CoPhIR_76* database, consisting of 76-dimensional descriptors (12-dimensional color layout and 64-dimensional color structure). As a distance function the Euclidean (L_2) distance was employed.

The *Polygons* database was a synthetic randomly generated set of 250,000 2D polygons, each polygon consisting of 5–15 vertices. The Polygons should serve as a non-vectorial analogy to clustered points. The first vertex of a polygon was generated at random. The next one was generated randomly, but the distance from the preceding vertex was limited to 10% of max. distance in the space. We used the Hausdorff distance [10] for measuring the distance between two polygons, so here a polygon could be interpreted as a cloud of points.

4.2 Experiment settings

The query costs were always averaged for 200 metric skyline queries, while the query examples followed the distribution of database objects. As the parameters we observed various database sizes, the (P)M-tree node capacities, the number of query examples, and the number of PM-tree leaf pivots. The (P)M-tree node capacities ranged from 20 to 40 routing/ground entries, the index sizes took 200MB–2GB, the P(M)-tree heights were 3–5 (4–6 levels). The minimal (P)M-tree node utilization was set to 20% of node capacity. The number of PM-tree leaf pivots ranged from 30 to 1000, while the number of inner pivots ranged from 15 to 500. Unless otherwise stated, the number of MSQ query examples was 2, the (P)M-tree node size was 20, the number of leaf pivots was 1000 for CoPhIR and 300 for Polygons (the number of inner pivots was half the number of leaf pivots).

4.3 The results

In the first set of experiments, the number of PM-tree leaf pivots was increasing, see Figure 6. When considering Polygons database, the M-tree's MSQ got to 17% of distance computations needed by simple sequential search on the Polygons database. However, for the highest number of pivots the PM-tree's MSQ reduced the M-tree costs by another 35%. The heap size required by PM-tree reached only up to one third of the heap size required by the M-tree. The impact of pivot-skyline filtering (the +PSF(+DEF) variants) on the maximal heap size was significant.

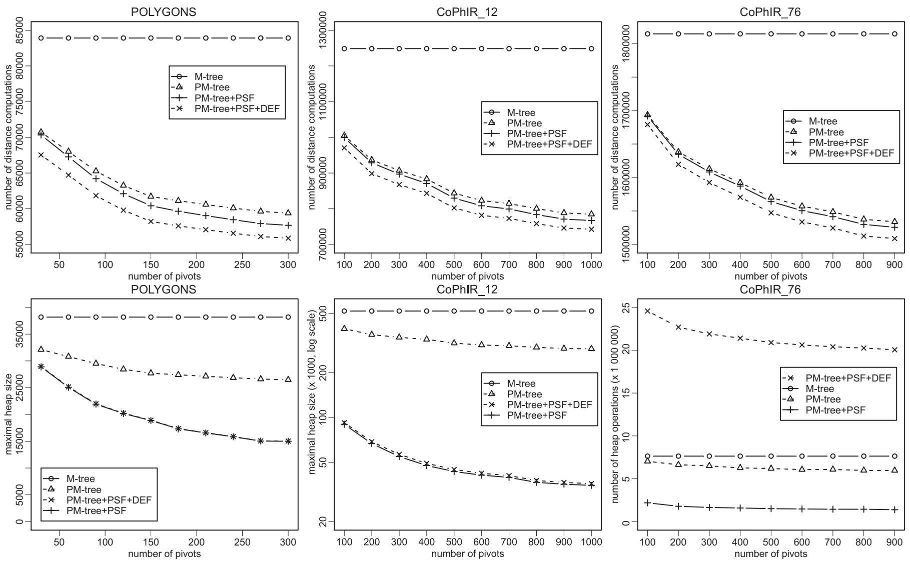


Fig. 6. Increasing number of pivots: distance computations, maximal heap size

The same situation is presented for the Cophir_12 database. The results are even better as for Polygons – the number of distance computations for PM-tree+PSF+DEF variant was reduced to 60% of M-tree costs, while the maximal heap size was reduced down to 8% of the heap size required by M-tree (note the log.scale in the figure).

Finally, the same situation is presented for the high-dimensional Cophir_76 database. Because of the high dimensionality, the M-tree performance was poor – it got to 91% distance computations required by simple sequential search. The PM-tree performed better, achieving 75% of the sequential search's distance computations. The PM-tree+PSF+DEF variant performs poorly when looking at the number of heap operations, due to the deferred heap processing, i.e., repeated insertions of MDDRs into the heap (see Section 3.3). On the other hand, the +DEF variant steadily achieves the lowest distance computation costs (as expected). The PM-tree+PSF variant performs the best, achieving 25% of the heap operations spent by M-tree.

The second set of experiments focused on the increasing database size. In Figure 7 the results for Cophir_76 database are presented. The trend of increasing distance computations is obvious for all MSQ processing methods. However, the situation is dramatically different for the number of heap operations and the maximal heap size, where the PM-tree+PSF beats the M-tree by a factor of 17 in heap operations, and by a factor of 7 in the maximal heap size. On the other hand, PM-tree+PSF+DEF suffers from a high number of heap operations.

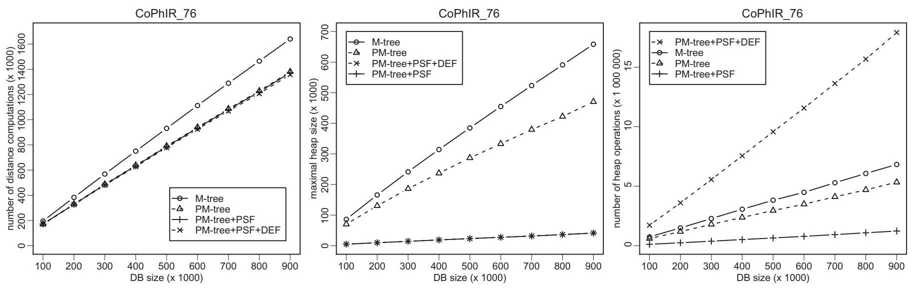


Fig. 7. Increasing size of Cophir_76 database: (a) Distance computations (b) Maximal heap size (c) Heap operations

In the third set of experiments, the results for increasing number of query examples used in metric skyline queries are presented on the Cophir_12 database, see Figure 8. Because the number of skyline objects grows substantially with the increasing number of query examples (retrieving 50, 400, 1750, 4570 skyline objects for 2-, 3-, 4-, and 5-example MSQs), the overall MSQ costs grow substantially as well. Nevertheless, the PM-tree MSQ processing is still much cheaper than the M-tree in the heap size and operations, even for 5 query examples. However, note that for 5 query examples the distance computations of all the methods come close to the costs of simple sequential search.

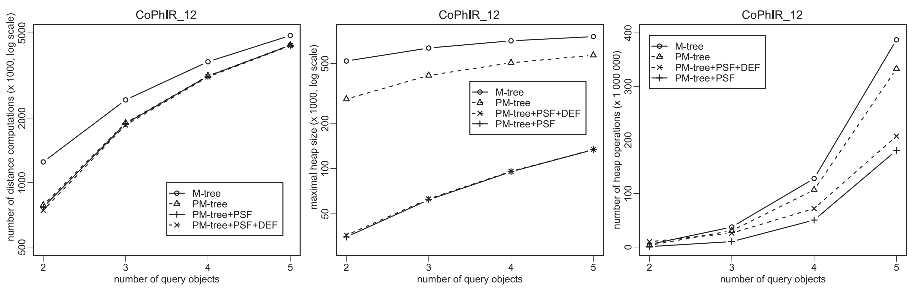


Fig. 8. Increasing number of query examples: (a) Distance computations (b) Maximal heap size (c) Heap operations

Although the I/O costs do not represent a dominant performance component in similarity search[1], in the last experiment we present the I/O costs as a supplementary result (CoPhIR_12, 2 query examples). In particular, in Figure 9a we give the numbers of logical seeks[2] spent by skyline processing (the seek operation is the most expensive one when fetching a page/PM-tree node from the disk). The PM-tree based MSQ processing spent just 64% of seek operations required by the M-tree. As for the distance computation costs, also the I/O costs were decreasing with increasing number of pivots.

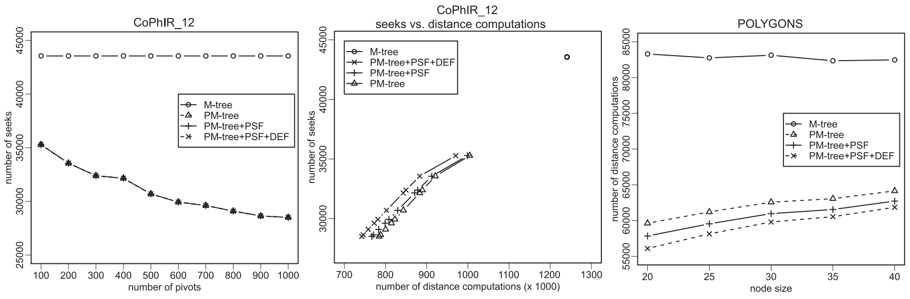


Fig. 9. Increasing number of pivots: (a) I/O costs (b) I/O costs vs. distance computations. (c) Increasing size of (P)M-tree nodes.

In Figure 9b the I/O costs vs. computation costs are shown. As in the first chart, the pairs ⟨I/O costs, distance computations⟩ were obtained for different numbers of pivots employed by PM-tree. Since the (P)M-tree indices consisted of 79,584 nodes, note that the I/O costs correspond to fetching 55% of all the index nodes for M-tree and 35% for PM-tree (1000 pivots). Also note there is linear correlation between the distance computations and I/O costs. 55%. Finally, the Figure 9c shows the performance of (P)M-tree depending on the node size.

4.4 Summary

The experimentation with M-tree and PM-tree based metric skyline processing has shown that the PM-tree outperforms the M-tree implementation up to 2 times in the number of distance computations, almost 20 times in the number of heap operations and the maximal heap size, and almost 2 times in the I/O costs. The results for maximal heap size are exceptionally important, because a large size of the heap (which is a main-memory structure) would prevent from processing of metric skyline queries on very large databases.

5 Conclusions

In this paper we have proposed a PM-tree based implementation of metric skyline query, a recently proposed multi-example query concept suitable for advanced

[1] A single distance computation is generally supposed to be much more expensive than a single I/O operation.

[2] We did not consider any node caching in this experiment.

similarity search in multimedia databases. We have shown that the PM-tree based implementation of metric skylines significantly outperforms the existing M-tree based implementation in all observed costs – the time, space, and I/O costs.

Acknowledgments. This research was supported by Czech Science Foundation (GAČR) Project 201/09/0683 and by institutional research plan MSM0021620838.

References

1. S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proceedings of the 17th International Conference on Data Engineering*, pages 421–430, Washington, DC, USA, 2001. IEEE Computer Society.
2. E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.
3. L. Chen and X. Lian. Dynamic skyline queries in metric spaces. In *EDBT '08: Proceedings of the 11th international conference on Extending database technology*, pages 333–343, New York, NY, USA, 2008. ACM.
4. L. Chen and X. Lian. Efficient processing of metric skyline queries. *IEEE Trans. on Knowl. and Data Eng.*, 21(3):351–365, 2009.
5. P. Ciaccia, M. Patella, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *VLDB'97*, pages 426–435, 1997.
6. R. Fagin. Combining fuzzy information from multiple systems. *J. Comput. Syst. Sci.*, 58(1):83–99, 1999.
7. F. Falchi, C. Lucchese, R. Perego, and F. Rabitti. CoPhIR: COntent-based Photo Image Retrieval [http://cophir.isti.cnr.it/CoPhIR.pdf], 2008.
8. A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In B. Yormark, editor, *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984*, pages 47–57. ACM Press, 1984.
9. G. Hjaltason and H. Samet. Incremental similarity search in multimedia databases, computer science dept. tr-4199, univ. of maryland, college park, 2000.
10. D. Huttenlocher, G. Klanderman, and W. Rucklidge. Comparing images using the hausdorff distance. *IEEE Patt. Anal. and Mach. Intell.*, 15(9):850–863, 1993.
11. H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, 2006.
12. T. Skopal. Pivoting M-tree: A Metric Access Method for Efficient Similarity Search. In *Proceedings of the 4th annual workshop DATESO, Desná, Czech Republic, ISBN 80-248-0457-3, also available at CEUR, Volume 98, ISSN 1613-0073,* `http://www.ceur-ws.org/Vol-98`, pages 21–31, 2004.
13. T. Skopal. Unified framework for fast exact and approximate search in dissimilarity spaces. *ACM Transactions on Database Systems*, 32(4):1–46, 2007.
14. T. Skopal, J. Pokorný, and V. Snášel. Nearest Neighbours Search using the PM-tree. In *DASFAA '05, Beijing, China*, pages 803–815. LNCS 3453, Springer, 2005.
15. Y. Tao, D. Papadias, and X. Lian. Reverse knn search in arbitrary dimensionality. In *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*, pages 744–755. VLDB Endowment, 2004.
16. P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search: The Metric Space Approach (Advances in Database Systems)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

A Framework for Efficient Design, Maintaining, and Evolution of a System of XML Applications[⋆]

Martin Nečaský and Irena Mlýnková

Department of Software Engineering, Charles University in Prague, Czech Republic
{necasky,mlynkova}@ksi.mff.cuni.cz

Abstract. The today's applications usually form a system of sub-applications, each being responsible for a particular functionality. Hence, the design and maintenance of such a complex system is not a simple task. In addition, the user requirements can change and the affected parts need to be identified and evolved. Similarly, new components or even whole system may need to be integrated.
In this paper we describe a framework that enables one to face the described issues. For this purpose we exploit verified technologies, such as conceptual modeling, data semantics, matching algorithms etc. Using a set of examples we show that our approach enables one to design, maintain, and evolve a system of applications efficiently and precisely. We depict the features on an XML system represented by a set of web services that exchange XML data. However, the concepts are general and can easily be extended for any kind of data format.

1 Introduction

The current applications are often based on two concepts. First, they do not form a monolithic piece of software, but they are usually composed of a set of simpler sub-applications, each being responsible for a particular execution part. Second, such sub-applications usually exploit a set of web technologies so that they can be distributed and communicate with each other. Hence, we usually speak of a complex *system of applications* which involves a huge amount of data formats being exchanged and processed by its *components*.

Considering such a complex system, there occurs a number of related issues. First, the data formats need to be designed. With regard to the existing data design techniques we need a kind of conceptual model, general enough to cover any of the formats. Second, we need to design the formats correctly, so that they cover all the required information, but avoid redundancies. Hence, we cannot design the particular formats independently. And, last but not least, when the system is designed and implemented, there occurs the problem of evolution. The requirements of users can change which can lead to changes in several data formats and consequently the respective components that process and exchange

⋆ Supported by the Czech Science Foundation (GAČR), grants no. 201/09/P364 and P202/10/0573.

the data. It can also influence storage and manipulation strategies of the data formats. And, similarly, new applications may need to be incorporated, or even whole systems may be mutually integrated.

The aim of this paper is to describe a framework that faces the described issues and identifies related problems that have not been solved yet by researchers. The proposed framework covers the whole life cycle of a system of applications from design and maintenance to evolution. Its main advantages are as follows:

– It involves a conceptual level of both the data and the business processes which enable one to describe the user requirements easily and precisely.
– It creates and preserves the relations between applications and their data. Hence, any evolution change can be propagated to all affected components.
– The system is open, thus new applications can be semi-automatically incorporated or it can be integrated with a whole other system.
– The additional information are exploited in various parts of the system, such as storage strategies or matching of components during integration.

Probably the most common example of the described system is represented by the principle of *Service Oriented Architecture* (SOA) and its most common implementation – *web services* [16]. A web service (WS) is a software system designed to support machine-to-machine interaction over the Internet using message exchanging. Most of the functionality of a WS is based on XML [8] – its interface is described in WSDL [11], the data is exchanged using SOAP [9] messages, etc. For simplicity we will consider only such type of system – so-called *XML system of applications* that exchange and process data in XML format. However, the framework can easily be extended for various other formats.

The paper is structured as follows: Section 2 provides a running example of an XML system. Section 3 describes basic decomposition of an XML system and Section 4 describes advanced components that form the framework. Section 5 provides conclusions.

2 Running Example

Let us consider a real-life purchasing application, in particular a simple business process of purchasing goods. The diagram of the business process modeled in BPMN [13] is depicted in Figure 1. The process starts with receiving a purchase order from a customer. Firstly, the trader checks the provided credit card details and rejects the purchase when the check fails. Otherwise, the trader arranges delivery of the purchased goods and sends an invoice back to the customer.

The business process is implemented by a publicly available WS *PurchaseWS* depicted in Figure 2. The WS provides an operation *ProcessPurchase* that receives a purchase order as the input from a customer. When a customer's credit card cannot be validated the WS sends a rejection message back to the customer. Otherwise, the customer receives an invoice with the delivery details as a response from the WS.

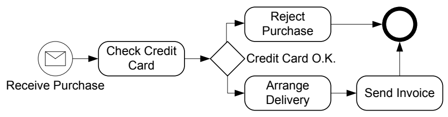


Fig. 1. Purchasing Business Process Diagram

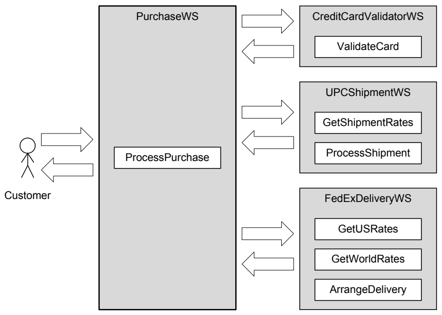


Fig. 2. *PurchaseWS* WS

Figure 2 also depicts third-party WSs exploited in the implementation of *PurchaseWS*. *CreditCardValidator* WS is exploited for validating credit cards of customers. WSs of various logistics companies are exploited. To arrange a shipment, shipment offers are retrieved from these services and the cheapest is selected. The figure depicts WSs of the UPS[1] and FedEx[2] companies.

3 Basic System Decomposition

The full architecture of our framework is depicted in Figure 3. In this section we describe so-called *run-time parts*, i.e. parts that form the run time of an XML system. In the following section we describe extensions we need to design, maintain and evolve such XML system efficiently.

The run-time parts of the system are *XML Schema* part, *Web Services* part, *Database* part, and *Semantics* part. We denote them *XS*, *WS*, *DB*, and *SEMANTICS*, respectively.

The ***XS component*** is a mandatory part of each XML system. It covers *XML schemas* that specify XML formats applied in the system and *integrity constraints* that enhance XML schemas with advanced conditions that cannot be expressed with XML schema languages. XML schemas can be expressed in languages such as, e.g., XML Schema [5, 6], or Relax NG [25]; for expressing XML constraints, XML pattern languages, e.g. Schematron [19], can be applied.

Example 1. The running example in Section 2 exploits several XML formats. For example, there is a format for XML messages with purchase orders sent by customers to *PurchaseWS* or two formats for XML messages with delivery information sent by *PurchaseWS* to *UPCShipmentWS* and *FedExDeliveryWS*. There are also advanced integrity constraints – for example, an integrity constraint specifying that "*check sum equals to the sum of prices of individual items*".

[1] `http://www.ups.com/`
[2] `http://fedex.com/`

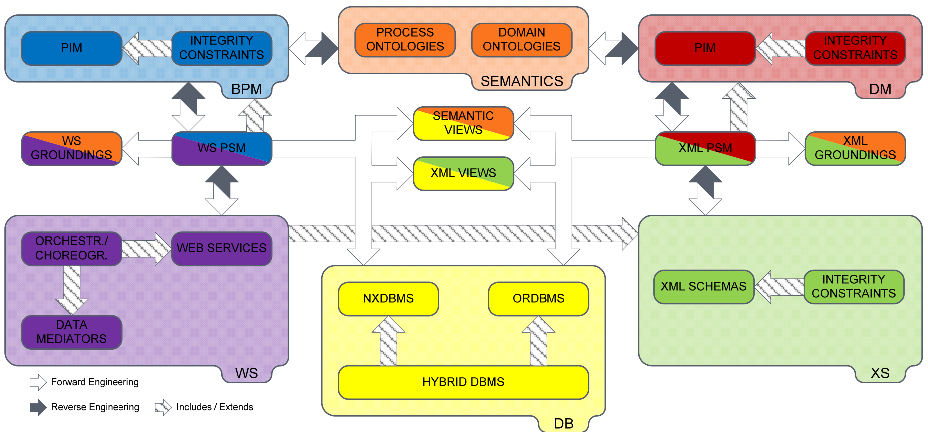


Fig. 3. System Architecture

The ***WS part*** covers business processes implemented by the XML system. A WS is comprehended as a standalone software component that implements a business process. *Data mediators* can then implement transformation of XML messages between WSs, and *orchestration/choreography* allows for composition of individual WSs into complex services ones. Transformations can be specified as XSLT [1] scripts. To describe orchestration/choreography, languages such as WSBPEL [10] can be applied.

Example 2. In our sample system we specify WS interfaces as WSDL descriptions. Moreover, *PurchaseWS* exploits the two external *UPCShipmentWS* and *FedExDeliveryWS* and also two trader's internal WSs *InventoryWS* and *AccountingWS* for checking the amount of a particular product on the stock and issuing invoices, respectively. In other words, *PurchaseWS* orchestrates these WSs. Since the input XML formats of both *UPCShipmentWS* and *FedExDeliveryWS* differ from the output XML format of *PurchaseWS*, it is necessary to incorporate XSLT data mediators that transform the messages respectively.

The ***DB (Database) part*** covers databases that persist XML data involved in the system. The XML documents can be stored centrally in a single database or distributed across different databases. *Native XML databases* allow for storing XML messages in their native form. *Object-relational databases* require decomposition of XML messages into object-relational tables.

Example 3. The data in the exchanged XML messages in our sample XML system are stored in a relational database. When a purchase order arrives from a customer, it is shredded into records of relational tables. The same is for other information, e.g. delivery details, invoices etc. Conversely, because of legal constraints, each trader needs to store the invoices from suppliers as they come instead of converting them to normalized relational tables. Therefore, the trader exploits a native XML database in this case.

The ***SEMANTICS part*** exploits ontologies to express semantics of the data domain and business processes in a machine-readable way. *Domain ontologies* specify semantics of the data domain, e.g., in OWL [3]. *Process ontologies* specify semantics of business processes, e.g., in OWL-S [2] or WSMO [7]. This enables one to exploit various advanced semantic techniques to, e.g., dynamically discover *PurchaseWS* or mediate other business processes to the business process implemented by the XML system.

Example 4. For customers equipped with solutions based on the Semantic Web technologies, a trader can provide additional semantics part of the XML system. The trader exploits a standardized BMO[3] business ontology to specify the important concepts, e.g. customer, product, purchase etc., and the purchasing business process at the semantic level.

When two or more run-time parts are present in the system, we need them to work together. This internal integration is covered by *integration parts* called *XML VIEWS*, *SEMANTIC VIEWS*, *XML GROUNDING*, and *WS GROUNDING* depicted in Figure 3 as double-colored rounded boxes.

XML VIEWS cover integration of the XML and DB parts. They transform data from its database representation to XML representation and vice versa. An XML view can be expressed in XML query languages such as SQL/XML [18].

Example 5. In our example we need XML views expressed in SQL/XML to transform the data stored in the relational database (e.g. purchase orders and delivery data) to XML formats specified by the XML schemas in the XS part and vice versa. These views allow to access the data in their various XML representations even if it is logically stored in relational tables.

SEMANTIC VIEWS cover integration of SEMANTICS and DB parts. They transform data from its database representation to the ontological representation, e.g. RDF [4] triples, and vice versa. This allows for expressing the semantics of the data in a machine-readable way and, at the same time, database-supported semantic reasoning and querying. A SEMANTIC view can be expressed in an XML query language, such as SQL/XML, as RDF triples can be represented in XML.

Example 6. Our sample system provides semantics-enabled customers with the purchase, invoice, and delivery data represented in the ontological representation conforming to the ontologies from the SEMANTICS part. This is achieved by semantic views that convert, e.g., purchase order data in relational tables to RDF triples and vice versa.

XML GROUNDING integrates SEMANTICS and XS parts. It comprises mutual mappings of XML schemas to domain ontologies and as XSLT scripts that specify data transformation between XML and ontological representation. It is similar to SEMANTIC VIEWS part but instead of DB there is the XS part.

[3] `http://www.bpiresearch.com/Resources/RE_OSSOnt/re_ossont.htm`

Example 7. When a semantics-enabled customer receives an XML message with delivery information, (s)he needs to know the semantics of parts of the XML message, e.g. delivery date, packaging etc., in the terms of the domain ontology. This is expressed by the XML grounding that provides mapping of parts of the XML schema for the XML message to the domain ontology.

WS GROUNDING integrates SEMANTICS and WS parts. It enhances XML GROUNDING by adding mappings of WS operations, orchestration, and choreography to process ontologies in order to specify semantics of, e.g., inputs/outputs of WSs.

Example 8. Similarly to XML grounding, it is necessary to provide mapping of the WSDL description of *PurchaseWS* to the process ontology that specifies the semantics of our purchasing business process.

4 System Extensions

The components described in the previous section (or, in simpler cases, their various subsets) form the XML system and are present at run time. They need to be designed, implemented, and maintained. In addition, since an XML system usually evolves, the components need to be modified or even whole new XML components need to be integrated. Our framework involves techniques and tools that enable one to manage the whole life cycle of an XML system in a user-friendly and effective way.

Complex Design As we have outlined, an XML system involves XML schemas, WSDL schemas, database schemas, integration scripts etc. They implement different and often limited viewpoints of various participants (i.e. users, other systems, standardization efforts etc.). Therefore the primary component of the extended system is a family of conceptual models [27], related integrity constraints [31], and a complex design tool [12] that supports them. In Figure 3 they are represented by the two *design-phase parts* of the architecture – *DM* and *BPM.*

The ***DM (Domain Model) part*** covers a conceptual model of the data domain. It involves *platform-independent model* (*PIM*) which provides conceptual diagrams of the domain and integrity constraints. PIM models important aspects of the data domain (i.e. concepts and relationships) regardless its representation at logical levels, e.g. XML or DB, using classical UML [15] class diagrams. Integrity constraints extend PIM with information that cannot be expressed with the PIM modeling language. For this purpose (OCL) [14] is exploited.

Example 9. Figure 4 depicts a PIM diagram[4] of our sample problem domain. Concepts are expressed as classes, e.g. *Customer* or *Order*. Relationships between concepts are expressed as associations.

[4] It was modeled in XCase [12], a modelling tool that implements basic features described in this paper.

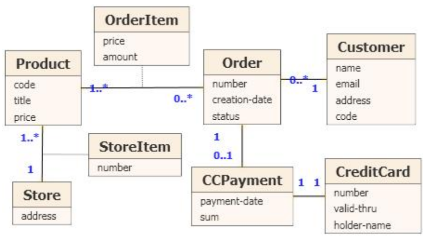


Fig. 4. DM PIM

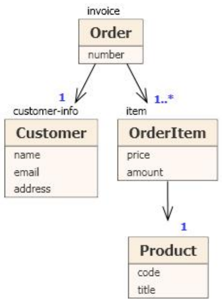


Fig. 5. DM PSM

The ***BPM (Business Process Model) part*** covers a conceptual model of the business processes. It involves PIM which provides conceptual diagrams of the business processes and integrity constraints. PIM models activities, events, and messages participating in business processes independently of their implementation in WS part in BPMN [13].

Integrity constraints extend PIM with advanced constraints specific for individual WSs (e.g. pre-conditions and post-conditions of activities and events or constraints on exchanged messages). Again OCL can be applied.

Example 10. A sample business process PIM diagram is depicted in Figure 1. Each message, e.g. purchase order, credit-card check, invoice etc., specified by the business process represents part of the data domain. This part is modeled as a PIM diagram which is part of the whole PIM diagram from Figure 4. For example, the PIM diagram for invoice messages contains *Order*, *ProductItem*, *Product*, and *Customer* classes.

Note that the existing modeling languages [15, 13] consider data modeling and business-process modeling separately. In our framework we interconnect these two areas and model them uniformly at PIM level. Hence, PIM gives an overall picture of the data domain and business processes independently of their implementation; the interconnection enables one to describe the required applications more precisely.

We survey methods for conceptual modeling techniques in [26], where we show that current methods allow modeling XML formats only at the PSM level. In [27] we introduced a conceptual model for XML that allows for modeling XML formats also at the PIM level.

Regarding business process modeling, there are languages such as BPMN [13]. However, we are missing methods for modeling data in current BPM PIM modeling languages, i.e. methods interconnecting BPM and DM PIMs. These languages must therefore be further extended. Probably the first step towards this aim is paper [21], where the authors deal with transformations of BPM to UML using XSLT.

Forward Engineering Manual coding of all components of the XML system (e.g. XML and WSDL schemas, XSLT scripts, database schemas etc.) consumes a lot of effort and is error-prone. Hence, our framework is based on a family of conceptual models [27] and respective technologies. Apart from PIM, it involves so-called *platform-specific model* (*PSM*) represented by the XML PSM integration part, where each diagram takes part of the PIM diagram(s) and specifies how it is represented in a particular XML format. The diagram can also be comprehended as a mapping between PIM and XML schemas. Similarly, our framework involves techniques for specification of implementation of the business processes by WSs. This is represented by the WS PSM integration part and comprises WS PSM diagrams. Again each diagram specifies implementation of parts of a business process.[5]

The translation of PSMs to respective representations is done semi-automatically. In particular it involves translation of:

– XML PSM to XML schemas (*DM-to-XS*),
– XML PSM to database schemas (*DM-to-DB*) and XML VIEWS,
– WS PSM to WSDL descriptions, XSLT data mediation scripts, BPEL orchestrations, and WS-CDL choreographies (*BPM-to-WS*), and
– PIM to ontologies (*DM&BPM-to-SEMANTICS*) and SEMANTIC VIEWS.

Forward engineering is depicted in Figure 3 by white-filled arrows.

Example 11. A sample XML PSM diagram is depicted in Figure 5. It models how invoices are implemented in XML, i.e. how instances of classes from the DM PIM diagram in Figure 4, e.g. *Order*, *Customer*, or *Product*, are represented. From the XML PSM diagram, an XML schema, XML view, and XML grounding for this particular XML format are derived. Similarly, forward engineering of BPM PIM to WSs specification can be solved via BPM PSM.

Consequently, the manual coding of WS, XML, DB, and SEMANTICS parts components is significantly reduced to design of XML PSM and WS PSM diagrams which is more user-friendly and natural. The user does not need to bother with syntactic details, specifics of particular format etc. What is more, the common PIM diagram also formally interrelates components of WS, XML, DB, and SEMANTICS run-time parts. Such information is further exploited in the following sections.

In [26], we also study techniques of translating conceptual diagrams in various conceptual modeling languages to XML schemas. There are also methods for translating BPMN diagrams to BPEL scripts [34]. The translation is done only automatically. However, designers need a possibility to influence the translation process which is missing in the current literature.

In [29], we study derivation of an optimal native XML database schema from a set of XML PSM diagrams and their DM PIM diagram. In [33], the authors

[5] Note that in the same way the system can be extended with PSM of relational data (e.g. ER diagrams [32]), classes and objects (e.g. UML [15]), etc.

study methods of derivation of an optimal hybrid database schema for a given XML schema. These methods should be further extended for deriving a hybrid database schema for a set of XML PSM diagrams.

Reverse Engineering When a new XML component needs to be incorporated into the XML system, it is usually necessary to integrate it with other components manually. In simple cases it is possible, however in complex situations it can be a very hard task. For this purpose our framework involves (semi-)automatic techniques for reverse engineering of:

– XML schemas to XML PSM diagrams
– WS components (i.e. WSDL schemas, BPEL, and WS-CDL scripts) to WS PSM diagrams
– external domain ontologies to DM PIM, and
– external process ontologies to BPM PIM.

Example 12. Suppose that the trader wants the XML system to support also managing supplies from the suppliers. The suppliers provide WSs for managing supplies; however, the interfaces are different. This requires to integrate the WSDL descriptions and XML schemas of the suppliers with the trader's XML system. Instead of doing this manually, we enable one to map the XML schemas to the DM PIM diagram through XML PSM diagrams (semi-)automatically derived from the XML schemas. WSDL descriptions can be also mapped in a similar way to a BPM PIM diagram specifying the supply management business process from the trader's point of view. Then, all other components can be derived automatically using the forward engineering procedures.

Having the (semi-)automatic strategy, we significantly reduce the manual work when incorporating third-party components, e.g. XML schemas, WSDL descriptions, or ontologies of standardization organizations, business partners etc., into an existing XML system.

As we have described in [28] the reverse engineering approach cannot be purely automatic since in several cases there can be multiple options of a suitable mapping. However, using verified strategies, such as similarity matching [23], evaluation of semantics etc., our approach enables one to reduce the options to reasonable amount. In addition, it even provides several metrics that enable one to evaluate quality of the options from distinct points of view.

Note that similarly we can support integration of whole XML systems at PIM level. Again, if a given system does not involve our DM and BPM extensions, they can be reverse engineered. Then the PIM integration specifications are directly translated to XSLT data mediators and BPEL business process mediators.

Evolution and Versioning Management As mentioned before, sooner or later user requirements can change and, hence, the respective data need to be modified. The problem is that such modifications can affect multiple components of the system, such as, e.g., XML schemas, WSDL descriptions, database schemas etc. And not only can such modifications be demanding, but, in complex systems, they can also be very hard to identify.

For the purpose of complex evolution management we exploit the previously described features described. Similarly to design phase, we assume that most users express their modifications in PIM since again (s)he does not have to bother with specific features of particular formats. Such changes are then propagated to all related run-time parts by exploiting the forward-engineering methods – we speak about *downwards propagation*. On the other hand, when a change needs to be done in a run-time part, it can be propagated to PIM by exploiting the reverse-engineering methods – we speak about *upwards propagation* – and then to all the related system parts again using downward propagation [30]. To perform the respective modifications, our framework involves techniques for (semi-)automatic derivation of XSLT scripts.

Example 13. In our sample scenario, we may need to represent names of customers in purchase orders as a pair first name and surname instead of a single value name. This requires to modify the XML schema for purchase orders. It may need to be propagated to the existing XML messages to preserve validity against the evolved XML schema as well as to the corresponding XML PSM diagram to preserve consistency with the XML schema. However, this also requires to change the DM PIM diagram or mapping from the XML PSM diagram to the PIM diagram. When the DM PIM diagram is changed, the change must be propagated downward to the other XML schemas in the system.

Similarly to the case of reverse engineering, also in case of evolution management the key advantage of our approach is reduction of manual work when the XML system evolves.

As we have studied in [30], the amount of approaches to XML evolution is surprisingly low and the approaches are trivial. They only deal with separate aspects, such as propagation of modification of XML schema level to XML documents or vice versa [22], several papers also deal with modifications of a kind of abstraction of the XML schema – either visualization [20] or UML diagram [17], i.e. a kind of PSM. However, none of them views the problem from the point of view of multiple applications sharing common domain.

Run-Time Support During the run time of the system we need further system components, such as storage strategies and respective query operations, platform for running WSs, support for semantic operations etc. All these components can also benefit from the design-time components and exploit the complex information on the whole system at run time.

For example, in most XML systems the XML data that are processed and exchanged by its components usually need to be persistently stored and retrieved. In general, there seems to be no generally optimal storage strategy. Since requirements of various XML applications significantly differ, for each type of processing of XML data the respective appropriate approach should be used [24]. And it is even often further optimized in a specific way. However, manual optimization of, e.g., database schemas with respect to the expected data retrieval and manipulation is complicated task. The more information are taken into account, the better, however the more complicated the search for optimum becomes.

As we have already described, our framework involves complex information on multiple applications that process the XML data, data mediators, schema versions etc. Consequently, the respective storage strategies can be found more precisely and in more broader context of multiple application views. In addition, they can be adjusted to centralized or distributed architecture.

5 Conclusion

The aim of this paper was a description of a framework that enables one to simplify, clarify, and streamline the design, maintenance, and evolution of a complex system of applications. Due to space limitations we have described a general architecture of an XML system, the most common issues that need to be solved during its life cycle and, in particular, how they can be simplified using the described framework, i.e. extension of the system.

Our current work naturally covers the full implementation of the key components of the framework. Some of them have already been covered by XCase [12] which we currently extend with modeling of business processes and storage level. Our future work will focus mainly on support of non-XML data models such as ER model, UML, etc. and application of the system in real-world use cases.

References

1. *XSL Transformations (XSLT) Version 1.0.* W3C, 1999. `http://www.w3.org/TR/xslt`.
2. *OWL-S: Semantic Markup for Web Services.* W3C, 2004. `http://www.w3.org/Submission/OWL-S/`.
3. *OWL Web Ontology Language.* W3C, 2004. `http://www.w3.org/TR/owl-features/`.
4. *RDF/XML Syntax Specification (Revised).* W3C, 2004. `http://www.w3.org/TR/rdf-syntax-grammar/`.
5. *XML Schema Part 1: Structures (Second Edition).* W3C, 2004. `http://www.w3.org/TR/xmlschema-1/`.
6. *XML Schema Part 2: Datatypes (Second Edition).* W3C, 2004. `http://www.w3.org/TR/xmlschema-2/`.
7. *Web Service Modeling Ontology (WSMO).* W3C, 2005. `http://www.w3.org/Submission/WSMO/`.
8. *Extensible Markup Language (XML) 1.0 (Fourth Edition).* W3C, 2006. `http://www.w3.org/XML/`.
9. *SOAP Version 1.2 Part 0: Primer.* W3C, 2007. `http://www.w3.org/TR/soap12-part0/`.
10. *Web Services Business Process Execution Language (WSBPEL) TC.* OASIS, 2007. `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel`.
11. *Web Services Description Language (WSDL) Version 2.0 Part 0: Primer.* W3C, 2007. `http://www.w3.org/TR/wsdl20-primer/`.

12. *XCase – A Tool for XML Data Modeling.* 2008. `http://kocour.ms.mff.cuni.cz/~necasky/xcase/`.
13. *Documents Associated with Business Process Modeling Notation (BPMN) 1.2.* OMG, 2009. `http://www.omg.org/spec/BPMN/1.2/`.
14. *Object Constraint Language Specification, version 2.0.* OMG, 2009. `http://www.omg.org/technology/documents/formal/ocl.htm`.
15. *Unified Modeling Language.* OMG, 2009. `http://www.uml.org/`.
16. *Web Services Activity.* W3C, 2009. `http://www.w3.org/2002/ws/`.
17. E. Dominguez, J. Lloret, A. L. Rubio, and M. A. Zapata. Evolving XML Schemas and Documents Using UML Class Diagrams. In *DEXA'05*, pages 343–352, Berlin, Heidelberg, 2005. Springer.
18. ISO/IEC 9075-14:2003. *Part 14: XML-Related Specifications (SQL/XML).* Int. Organization for Standardization, 2006.
19. R. Jelliffe. *The Schematron – An XML Structure Validation Language using Patterns in Trees.* 2001. `http://xml.ascc.net/resource/schematron/`.
20. M. Klettke. Conceptual XML Schema Evolution – the CoDEX Approach for Design and Redesign. In *BTW Workshops*, pages 53–63. Aachen, 2007.
21. O. Macek and K. Richta. The BPM to UML Activity Diagram Transformation Using XSLT. In *DATESO'09*, volume 471, pages 119–129, Spindleruv Mlyn, Czech Republic, 2009. CEUR-WS.
22. M. Mesiti, R. Celle, M. A. Sorrenti, and G. Guerrini. X-Evolution: A System for XML Schema Evolution and Document Adaptation. In *EDBT'06*, pages 1143–1146, Berlin, Heidelberg, 2006. Springer.
23. I. Mlynkova. Similarity of XML Schema Definitions. In *DocEng'08*, pages 187–190, 2008.
24. I. Mlynkova. Standing on the Shoulders of Ants: Towards More Efficient XML-to-Relational Mapping Strategies. In *XANTEC'08*, pages 279–283, Turin, Italy, 2008. IEEE.
25. M. Murata. *RELAX (Regular Language Description for XML).* 2002. `http://www.xml.gr.jp/relax/`.
26. M. Necasky. Conceptual Modeling for XML: A Survey. In *DATESO'06*, volume 176, pages 40—53, Cerna Ricka, Czech Republic, 2006. CEUR-WS.
27. M. Necasky. *Conceptual Modeling for XML.* IOS Press, Heidelberg, Netherlands, 2008.
28. M. Necasky. Reverse Engineering of XML Schemas to Conceptual Diagrams. In *APCCM'09*, pages 117—128, Wellington, New Zealand, 2009. CRPIT.
29. M. Necasky and T. Knap. Reconstruction of Normalized XML Documents. In *Innovations'08*, pages 213—217, Al Ain, UAE, 2008. IEEE.
30. M. Necasky and I. Mlynkova. On Different Perspectives of XML Schema Evolution. In *FlexDBIST'09*, Linz, Austria, 2009. IEEE.
31. M. Necasky and K. Opocenska. Designing and Maintaining XML Integrity Constraints. In *MoViX'09*, Linz, Austria, 2009. IEEE.
32. Ch. Peter. Entity-Relationship Modeling: Historical Events, Future Trends, and Lessons Learned. pages 296–310, 2002.
33. L. Stromback, M. Asberg, and D. Hall. HShreX — a Tool for Design and Evaluation of Hybrid XML Storage. In *FlexDBIST'09*, Linz, Austria, 2009. IEEE.
34. S. A. White. Using BPMN to Model a BPEL Process. IBM Corp., USA, 2005. `http://bpmn.org/Documents/Mappingf`.

Reconstructing Social Networks from Emails

Marcel Kvassay, Michal Laclavík, and Štefan Dlugolinský

Institute of Informatics, Slovak Academy of Sciences
Dúbravská cesta 9, 845 07 Bratislava 45, Slovak Republic
marcel.kvassay@savba.sk

Abstract. The article provides a brief overview of Social Network Analysis (SNA) and its potential for exploiting the wealth of information buried in the email archives of business and private entities. Within the scope of the COMMIUS[1] project, we built a proof-of-concept prototype in Java, which used the spreading activation algorithm to reconstruct various aspects of multidimensional social network from emails. Two different variants of the spreading activation algorithm are discussed and compared.

1 Introduction

1.1 Social Network Analysis

History. The development of Social Network Analysis (SNA) is instructively mapped out by Freeman in [1] and, more briefly, by Fararo in [2]. Fig. 1 depicts the area at the intersection of sociology and psychology where SNA emerged as an alternative to traditional sociology. This area can be broadly classified as social psychology (SP), though it overlaps with other fields, such as anthropology or ethnology.

Psychology typically focuses on the individual. It studies mental functions – perception, cognition, attention, emotion, motivation, etc. – and their role in individual and social behavior. Sociology, in contrast, focuses on collective formations and analyzes human social activity starting from the micro level (agency and interaction) to the macro level (systems and social structures). In the area where sociology overlaps with psychology, the focus is on small groups. According to Wikipedia,[2] there are differences between social psychology as practised by psychologists (SP_P), and by sociologists (SP_S). Psychologists retain their individual focus and study how the thoughts, feelings, and behaviors of individuals are influenced by other members of the group. Sociologists focus more on the group itself and study group dynamics, crowd-phenomena, etc., often in the context of larger social structures (race, class, gender). Based on this distinction, the origins of SNA can be traced specifically to social psychology as practiced by sociologists and anthropologists (SP_S).

[1] http://www.commius.eu/

[2] http://en.wikipedia.org/wiki/Social_psychology

Early forms of SNA, such as sociometry, appeared in the 1930s but for various reasons did not catch on. SNA was eventually accepted as a separate discipline in the 1970s, and has been on the rise ever since. With the spread of internet and cheap computing power, it penetrated mainstream sociology to such an extent that Wikipedia[3] now considers it "a key technique in modern sociology."

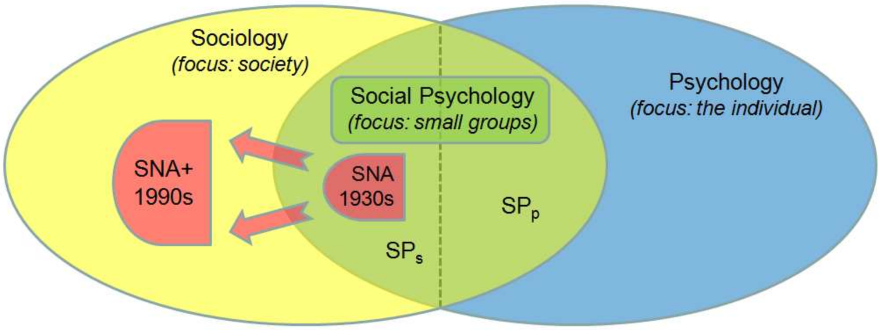


Fig. 1. SNA originated in the 1930s at the intersection of sociology and psychology, in the area shared by social psychology (SP), anthropology and ethnology. With the spread of internet and cheap computing power, it entered mainstream sociology (SNA+)

According to Freeman [1], SNA has four defining features:

1. Structural intuition (that patterning of social ties influences actors);
2. Systematic collection of empirical data;
3. Use of rigorous mathematical and computational models;
4. Graphic imagery.

Methods. To the use of statistics, which had a long tradition in sociology, SNA added new methods grounded in algebra and graph theory. SNA also enriched these disciplines with new concepts and techniques, such as block-modeling or the notions of bridge, centrality, structural balance, etc. More recently the focus has shifted to computational sociology and multi-agent simulations of social phenomena.

1.2 Social Networks in Emails

Email has become the most widespread Internet application. It is a tool supporting not only communication but also cooperation, task management, archiving, or information and knowledge management. Furthermore Email is a source of information on personal, enterprise or community network of an individual or an organization. Email communication analysis allows extraction of social networks with further connection to people, organizations, locations, topics or time.

Social Networks included in email archives are becoming increasingly valuable assets in organizations, enterprises and communities, though to date they have been little explored. While social networks in social network site such as Facebook are owned by

[3] http://en.wikipedia.org/wiki/Social_network_analysis

third parties, email social network data are owned by individual or organization including many useful connections hidden in emails. On personal archives, Xobni[4] exploits social networks to help the user manage contacts and attachments, but at the enterprise or community level, social networks can be exploited to improve email search, manage customers and suppliers, prioritise emails or improve inference mechanisms when connected with other detected semantic information from the email.

Social networks within email communication have been studied to some extent. For example, communication on the Apache Web Server mailing lists and its relation to CVS activity was studied in [6]. This work also introduces the problem of identifying email users' aliases. Extracting social networks and contact information from email and the Web and combining this information is discussed in [7]. Similarly, new email clients (e.g. Postbox) or plug-ins (Xobni) try to connect email social networks with web social networks like LinkedIn or Facebook. We have also performed some experiments on extraction of social networks from large email archives and network transformations using a semantic model [4]. Another research effort [8] exploits social networks to identify relations and tests proposed approaches on the Enron corpus.

To conclude, there is much research work done on social networks in the area of web social network applications, but email social networks are a bit different since in the email you can discover the level of interactions (number of messages exchanged, time, relation to content and possibly discovered semantics), and the influence of these differences on better information and knowledge management still needs to be explored. We would like to use similar approach as IBM Galaxy [10] in Nepomuk[5] project, where concept of multidimensional social network was introduced. In this paper we show initial results of exploiting email social network in order to support better understanding of email content as well as allowing applications such as partner or supplier search within organization or community.

1.3 COMMIUS[6] project

COMMIUS project is part of the 7th Framework Programme of the European Commission. Its acronym stands for "Community-based Interoperability Utility for Small and Medium Enterprises." The consortium comprises partners from Austria, Germany, Greece, Great Britain, Italy, Slovakia and Spain. Their objective is to provide SMEs (Small and Medium Enterprises) with "a zero, or very low-cost, entry into interoperability, based on non-proprietary protocols." To this end, a flexible architecture based on the open-source software was designed and implemented in Java.

The COMMIUS system is aimed primarily at companies that already conduct part of their business through email, i.e. send and receive orders, invoices, questionnaires, forms, etc. These documents are often manually retyped so as to enter them in the company's order-management or accounting system. Such companies would directly benefit from COMMIUS, since it is designed to automate these tasks. COMMIUS

[4] http://www.xobni.com/
[5] http://nepomuk.semanticdesktop.org/
[6] http://www.commius.eu/

scans the incoming emails, recognizes certain entities and documents (suppliers, customers, orders, invoices, telephone numbers, etc.), and proposes appropriate actions to the user. In case of an incoming order, for instance, the appropriate actions could be to check whether the requested items can actually be supplied, to send a confirmation to the customer, or to invoice and ship the order. These recommendations are presented to the user as HTML links that COMMIUS inserts into each incoming email. The user is then free to accept the recommendation (by clicking on the link) or proceed differently. A more detailed description of COMMIUS can be found in [3] and [4].

Social Networks in COMMIUS. The core of COMMIUS functionality deals with the business interoperability: detection of orders, invoices, payments, etc. in the incoming emails, and their semi-automated inclusion in the company's order-management and accounting system. Social network functionality is an add-on to this core. Its main purpose is to smoothen the process of COMMIUS adoption by new users, for example by pre-populating their product, customer and supplier databases based on the information extracted from their emails. On this basis, more advanced functions can be built, e.g. search for potential business partners.

Integration with the COMMIUS core. At the moment of installing COMMIUS, the user's email archives will be processed and the results stored in the form of multidimensional social network graph. After the installation, each incoming email will be added to this graph. Social network queries will search in the graph so that users get response in reasonable time. It should be noted that social network functions provide probabilistic results and would be offered to the user as recommendations only.

2 Implementation

We implemented our "social network extractor" in Java on top of the open-source graphical library JUNG.[7] The novelty of our approach is in the application of the spreading activation algorithm to two principal tasks:

1. reconstructing the social network from emails;
2. efficiently searching the social network.

The prototype implementation described below is a work in progress. So far only the initial version of the prototype has been tested on the first type of task; details are provided in section *3. Evaluation*.

2.1 Initial Version: Simple Cumulative Scorer

In the initial version, the information extraction module (IE) passes on a collection of strings (objects) matched by regular expressions. The strings acquire a "type" according to the regular expression that matched them. In this way, they are categorized as email addresses, personal names, names of organizations, telephone numbers, etc. The

[7] http://jung.sourceforge.net/

email message itself is represented by a “message ID” object to which the other objects are connected in a star-like fashion. If the same string is found in several messages, it is connected to all of them. If the same string is found multiple times in the same message, it has multiple links to that particular message ID, which is our way of recording the strength of the bond. The resulting network graph can be represented as a three-tier structure or tripartite graph (Fig. 2).

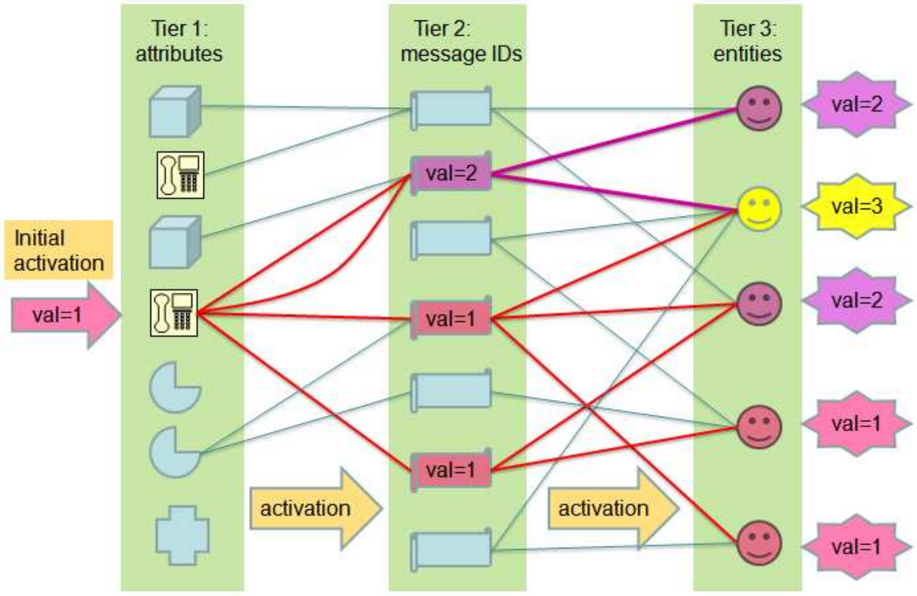


Fig. 2. Spread of activation as implemented in the initial version of the prototype. The activation starts from the left by assigning the initial activation value (val=1) to one attribute instance in Tier 1. In the next step, this initial value flows towards Tier 2 via the four red-colored links. There it accumulates in the three messages in which this attribute instance was found. One of them is connected via the double parallel link (which means it contained two occurrences of this attribute instance), so it accumulates a higher value (val=2) than the other two messages. In the final step, the activation values flow separately from each activated message (with val>0) towards Tier 3, where they further accumulate in the primary entities found in these messages. The primary entity with the highest accumulated value (val=3) is declared the “owner” of this attribute instance. The whole process is repeated for all the attribute instances in Tier 1

Our Simple Cumulative Scorer is inspired by the spreading activation algorithm in the sense that it separately “activates” each attribute instance with a uniform value of 1 and cumulatively spreads this activation (in the breadth-first manner) via the message IDs to the primary entities that will “own” the attribute. Each node can only fire once. It is an extremely simple implementation – we omitted such features of the standard spreading activation algorithm as attenuation, activation threshold or limit on the maximum activated value –nevertheless it allowed us to establish the utility of spreading activation in social networks.

In the three-tier structure depicted on Fig. 2, the middle tier (message IDs) links the attributes in Tier 1 to their “primary entities” in Tier 3. In general, the “primary entity” can be any of the objects identified by the Information Extractor (IE) if it can “own” (or be composed of) some other objects (attributes) likewise identified by the IE. In this sense, the “date” as a complex data type can be the primary entity with respect to year, month and day of which it is composed (provided the IE identifies these as separate objects), but it can itself be the attribute of a more complex data type, such as “event,” “conference,” etc. The number of such scenarios is limited only by the capabilities of the Information Extractor. In our case – since we are trying to reconstruct the social network – the primary entities are persons and organizations represented either by their email addresses or by their proper names. The Information Extractor collects both the email addresses and proper names from all the parts of the email message (the headers as well as the body). For our test task, we have chosen the telephone numbers as the sample attribute that we wish to assign correctly to persons and organizations.

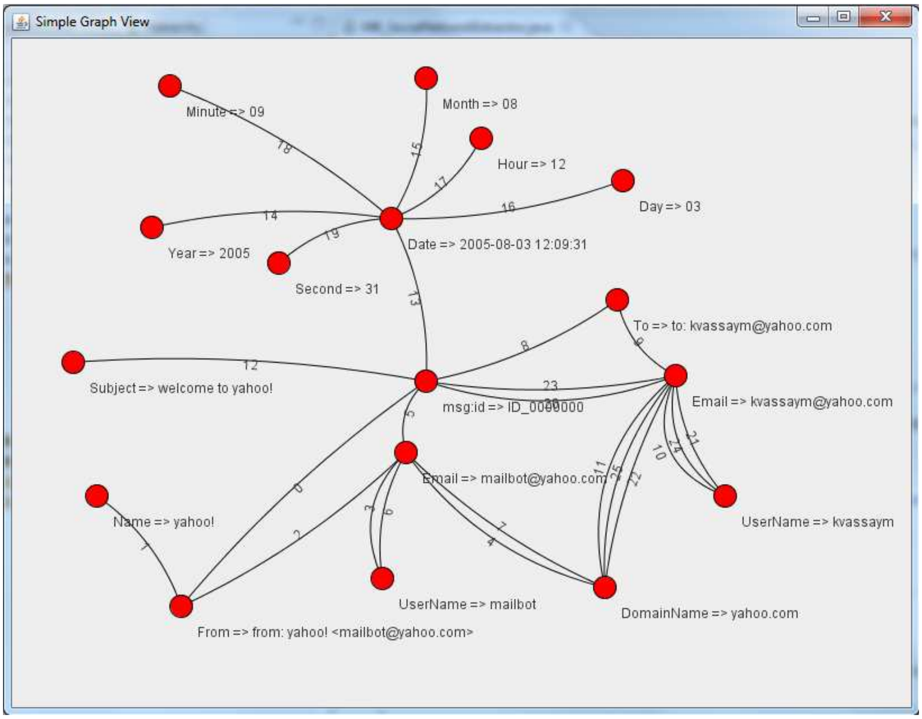


Fig. 3. Hierarchical graph of objects extracted by the enhanced information extractor (IE) from one email. Nodes representing sentences, paragraphs and blocks of the message are omitted. Graph was built from our personal email. Due to privacy reasons we cannot show the graph built from the enterprise emails of our Commius partners

2.2 Cumulative Edge Scorer with Attenuation

The improved version of the prototype relies on the enhanced output from the information extractor (IE). There are two major improvements:

1. IE produces a hierarchical tree of complex objects as shown on Fig. 3;
2. Objects are not linked directly to message IDs, but to the nodes representing the sentences, paragraphs and blocks of the message in which they were found. In this way the information about the physical proximity of the objects in the original email is preserved for further analysis as shown on Fig. 4.

In this richer and deeper tree-like structure, it makes sense to use a more sophisticated variant of spreading activation with attenuation and activation threshold. The structure can still be visualized as a multipartite graph, but the objects of each data type now require a separate partition (Fig. 4). This applies to candidate attributes as well as to primary entities. When partitioned in this way, the objects in each partition (i.e. the objects of the same data type) still have no connections among themselves, only to objects in other partitions, which is advantageous from the point of view of computational complexity.

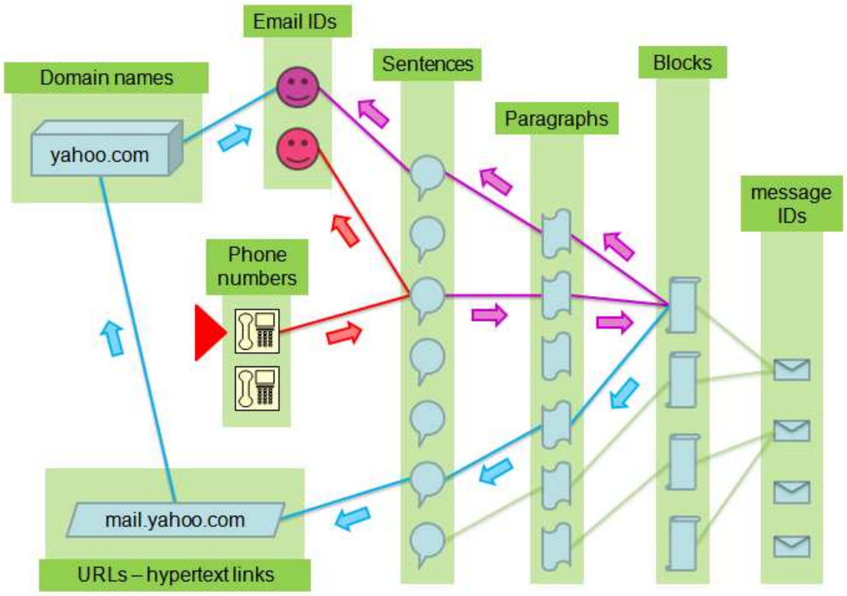


Fig. 4. A more complex variant of spreading activation in a multipartite graph. Activation again starts at one attribute instance (phone number) indicated by the red arrow (center left) and flows towards candidate primary entities (email IDs) by a variety of ways. Since the activation gets attenuated each time it passes through an edge, the shortest path (indicated in red) will carry over the greatest increment. But the longer paths (such as those indicated in purple and blue) can be so numerous that – depending on the value of attenuation and other parameters – their accumulated contribution ultimately prevails

Even in the enhanced version of our prototype, the spreading activation always starts from a single node. This allowed us to keep the simple and elegant breadth-first variant of the algorithm, in which each node can only fire once. Applications that start the initial activation from more than one node may need more sophisticated implementation of spreading activation.

3 Evaluation

We created a test email archive consisting of 28 representative sample messages supplied by our partners in COMMIUS. We then run the prototype with the task to assign telephone numbers to people and organizations (represented by their proper names) that were found in the emails.

Though our primary goal was to evaluate the spreading activation algorithm, we could not completely insulate it from the effects caused by the Information Extractor. These are seen primarily in the figures for the recall. Out of 17 unique relevant telephone numbers in the test emails, the IE identified 13. These 13 numbers were then passed on to the spreading activation algorithm. As can be seen from Table 1, the initial version of the spreading activation algorithm correctly assigned 8 of them, which resulted in the precision of 61.5% and the recall of 47%.

Table 1. Quantitative evaluation of the initial version of the spreading activation algorithm

Total relevant	Total found	Correctly assigned	Wrongly assigned	Recall [%]	Precision [%]
17	13	8	5	47	61.5

At present, we are still in the early stages of experimenting with the attenuated variant of the spreading activation. The first tentative results that we obtained are presented in Table 2.

Table 2. Quantitative evaluation of the attenuated variant of spreading activation

Total relevant	Total found	Correctly assigned	Wrongly assigned	Recall [%]	Precision [%]
17	13	10	3	58.8	76.9

As expected, the enhanced version of the prototype gave significantly better results (the precision of 76.9% and the recall of 58.8%), though we still need to investigate the remaining problems. Similarity in the tabular way of presenting the data actually masks deep differences between the two implementations.

In the initial version, the Information Extractor identified 32 candidate names of persons and organizations against which the phone numbers had to be matched. The candidate names were found purely by matching against regular expressions.

In the next version, it was decided to enhance the capabilities of the Information Extractor by adding gazetteers. Regular expressions were adapted so as to increase the recall and return a much larger number of candidate names, which would be subsequently filtered by the gazetteers. However, at the time of our experiment the gazetteers were not yet ready, so the enhanced version of the prototype had to match the phone numbers against a much larger set of 152 candidate names. That it still outperformed the initial version is therefore doubly significant and promising.

Moreover, the 3 phone numbers that were wrongly assigned had very low frequencies (one or two occurrences in the test corpus), so the basic assumptions of the algorithm were not met. Nevertheless, our initial experimentation with the prototypes was very useful and provided us with important hints for future work, which we discuss in the next section.

4 Conclusions and Future Work

This article is a report of a work in progress, and our experimentation with the prototypes still continues. The most obvious need is to further test the enhanced prototype on a larger set of representative emails. Here the main challenge is to get access to relevant and representative email sets. Though the email correspondence grows at an accelerating rate, we have learned that not all emails were created equal. They fall into distinct groups which differ significantly with respect to the ways and kinds of information that can be mined from them. Each application needs to be tested on the emails that are representative of the area in which it will be actually deployed.

The second lesson was that there were several alternative ways in which the Information Extractor could present the data extracted from the emails, and each had its pros and cons. There is a scope for deeper theoretical analysis here – either to find a general "canonical" structure suitable for most purposes or, alternatively, an easy way of transforming the data from one form to another depending on the task.

In general, graph and data transformations may be necessary in order to filter out the irrelevant information. Certain algorithms may require it for correct functioning; in others it will help to reduce the complexity of computation.

The "Social Network Extractor" component that we developed is able to process either mailboxes in mbox format or directories with email (.eml) messages, and thus extract multidimensional social network information contained in the email archive. In such a graph or network it is possible to see and exploit the links among objects such as people, time, email addresses, subjects, URLs, contact details or recipients.

The preliminary results of the extraction of social networks from email archives show that it is possible to deliver Xobni-like functionality in the enterprise or organizational context. Our approach is based on the concept of spreading activation similar to IBM Galaxy [10].

We have shown inferring relations between people and phone numbers on a small set of emails using a simple algorithm. The success rate (precision) of the experiment is 76.9%. In future, we would like to infer the relations such as those between customers and services, suppliers, products and transactions, organizations and people, people and address details, and others.

The extracted graph data from the email archives together with a well defined and tuned spreading activation algorithm can deliver the data needed for the adaptation of Commius or other enterprise systems. Such data can also be used to fill in the enterprise system database upon installation and thus help it to offer full functionality from the beginning. For example, we can populate a system database with a list of potential suppliers, organizations, contacts and their expertise.

Acknowledgements

This work is partially supported by projects Commius FP7-213876, APVV DO7RP-0005-08, AIIA APVV-0216-07, VEGA 2/0184/10 and VEGA 2/0211/09. We would also like to thank the anonymous reviewers, based on whose comments we have significantly reworked and enhanced the paper.

References

1. Freeman, L.: The Development of Social Network Analysis. Empirical Press, Vancouver (2006) (URL: http://aris.ss.uci.edu/~lin/book.pdf)
2. Fararo, T.J.: “Theoretical Sociology in the 20th Century.” Journal of Social Structure 2. (2001) (URL: http://www.cmu.edu/joss/content/articles/volume2/Fararo.html)
3. Laclavík, M., Šeleng, M., Gatial, E., Hluchý, L.: Future email services and applications. In: CEUR-WS: Proceedings of the poster and demonstration paper track of the 1st Future Internet Symposium (FIS´08), Vol. 399. Telecon Res. Center, Vienna (2008) 33-35. ISSN 1613-0073. (URL: http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-399).
4. Laclavík, M., Šeleng, M., Ciglan, M., Hluchý, L.: Supporting Collaboration by Large Scale Email Analysis. In: Cracow´08 Grid Workshop: Proceedings. Academic Computer Centre CYFRONET AGH, Kraków (2009) 382-387. ISBN 978-83-61433-00-2
5. Balzert, S., Burkhart, T., Kalaboukas, K., Carpenter, M., Laclavik, M., Marin, C., Mehandjiev, N., Sonnhalter, K., Ziemann, J.: Appendix to D2.1.2: State of the Art in Interoperability Technology, Commius project deliverable (2009)
6. Bird, C., Gourley, A., Devanbu, P., Gertz, M., Swaminathan, A.: Mining Email Social Networks. In: MSR ’06: Proceedings of the 2006 International Workshop on Mining Software Repositories. ACM, New York (2006) 137–143.
7. Culotta, A., Bekkerman, R., McCallum, A.: Extracting Social Networks and Contact Information from Email and the Web. In: CEAS ’04: Proceedings of the First Conference on Email and Anti-Spam, 2004. http://www.ceas.cc/papers-2004/176.pdf
8. Diehl, C. P., Namata, G., Getoor, L.: Relationship Identification for Social Network Discovery. In: The AAAI 2008 Workshop on Enhanced Messaging (2008)
10. Judge, J., Sogrin, M., Troussov, A.: Galaxy: IBM Ontological Network Miner. In: Proceedings of the 1st Conference on Social Semantic Web, Volume P-113 of Lecture Notes in Informatics (LNI) series (ISSN 16175468, ISBN 9783-88579207-9). (2007)

Efficient Implementation of XPath Processor on Multi-Core CPUs

Martin Kruliš and Jakub Yaghob

Department of Software Engineering
Faculty of Mathematics and Physics
Charles University in Prague
Malostranské nám. 25, Prague, Czech Republic
{krulis, yaghob}@ksi.mff.cuni.cz

Abstract. Current XPath processors use direct approach to query evaluation which is quite inefficient in some cases and usually implemented serially. This may be a problem in case of processing complex queries on large documents. We propose algorithms and XML indexing techniques which are more efficient and which can utilize standard parallel templates. Our implementation is highly scalable and outperforms common XML libraries.

Key words: XML, XPath, parallel, multi-threaded, multi-core

1 Introduction

The course of processor developement has changed significantly in the past few years. Pursuit of core frequency is not as important as before. Multiple processor cores are integrated on a single chip, making parallel architectures available to common users. This trend requires programmers to focus on problem parallelization instead of classic linear optimization. Common tools for querying XML documents do not exploit computational power of multi-core CPUs. Even though this is not a problem in case of small documents and simple queries, processing complex queries on large amount of data may benefit from parallelization.

In this paper, we will focus solely on processing single query on a single document, which we expect to be loaded into main memory in DOM [7] and indexed. This restriction seems very strong, however, mainstream computers of the day can accomodate XML documents with hundreds of millions of elements in RAM. Furthermore, we assume the following about the processed XML documents:

- Documents are shallow [18]. Element nesting depth does not exceed $\mathcal{O}(\log N)$, where N denotes number of elements.
- Strings (element names, contents, etc.) are short. We expect that all strings have $\mathcal{O}(1)$ length. This is quite strong assumption, however, we are focusing on processing the element structure, not on string algorithms.
- Documents contain only small number of element and attribute identifiers respectively to total number of elements and attributes.

Most XML documents have the properties described above, so our implementation will process them just fine. Documents that do not conform to our assumptions will not be considered in this paper for the sake of scope.

XPath specification [6] contains a lot of details, many of which are uninteresting from our perspective. We will focus on processing location paths, since they present the most challenging part of XPath. Common implementations [10][11] take direct approach which can lead to inefficient algorithms. We demonstrate the problem on an example. Consider simple location path:

`/descendant::a/following::b[`P`]`

Let us assume that all elements `b` are in 'following' relation with any element `a`. The `following::b` step then yields the same set of nodes for every element `a` produced by previous step. Therefore, predicate P will be evaluated for each node `b` $\mathcal{O}(N_a)$ times, where N_a denotes total number of elements `a`.

Straightforward implementation leads to an algorithm which has its time complexity exponentially dependent on the level of predicate nesting. Fortunately, this problem can be avoided by introducing vector operations and min-context rules [2]. When each predicate is processed for whole vector of contexts instead of traditional recursive evaluation, the exponential phantom menace is disposed of and evaluation time remains polynomial in both size of the document and query nesting depth.

The paper is organized as follows. Section 2 revises related work. Section 3 describes XML representation and indexing techniques. Section 4 inspects proposed algorithms in detail. Section 5 suggests possible parallelism exploitation. Section 6 presents experimental results and Section 7 concludes.

2 Related Work

Efficient implementation of XPath has been addressed in several papers. One of the most significant works was presented by Gottlob et al. [2]. It proposes algorithms for processing XPath queries in polynomial time. XPath also relies on efficient evaluation of location axes which was addressed in [16]. The topic of XML processing parallelization is relatively new. First studies focused on XML parsing [14][15] and processing in a general way. Successful approach is to employ work stealing [3] in order to reduce load balancing problems.

To our best knowledge, the only work directly related to parallel processing of XPath queries was presented by IBM [4]. The paper proposes data, query and hybrid partition strategies, how to divide workload among multiple threads. The partitioned queries are then processed by Xalan. This technique can be improved [5] by introducing metrics for processing costs. These metrics determine the best points for partitioning.

We take different approach. Our strategy is to design algorithms which will utilize parallel templates and ideas presented by Intel TBB [12]. These templates implement solutions for common patterns in parallelism such as parallel-for or parallel-scan algorithms.

3 XML Index

XPath processor expects XML document to be loaded in main memory in abstract tree structure (for example DOM). This structure provides basic operations for traversing the document such as finding children, parent, or attributes of given node. Even though these operations are sufficient to traverse the whole tree and process XPath query, they are quite ineffective for complex operations.

When processing XPath query, we are often required to retrieve all nodes with specific name or of specific type that are in relation (defined by used axis) with initial node-set. This task can be accelerated significantly when proper index is built. We have used the following two indices in our implementation: the Left-right-depth index and the element index.

The *Left-right-depth* index (also denoted LRD) is often used by many XML algorithms and data structures [2][16][17] to determine the relation of two nodes in constant time. The index is based on tagging nodes of DOM structure with three integers - *left*, *right*, and *depth* value. We have tagged only element nodes and attributes as other nodes are irrelevant for XPath evaluation. We denote $l(v)$, $r(v)$, and $d(v)$ the left-value, right-value, and depth of the node v respectively.

The *element index* is formed by hash table where element names are keys. Each name holds a list of references to the DOM structure pointing to all elements with this name as depicted on Figure 1.

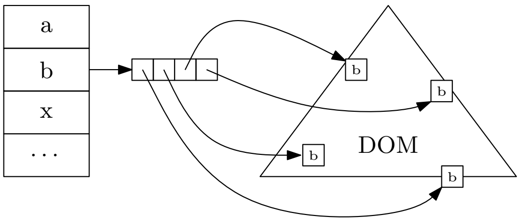


Fig. 1. Element index lists all elements of given name.

Links in the reference list are ordered by the position (i.e. the left-value) of the nodes to which they point to. This ordering is very useful, as shown later.

3.1 XPath Axes Implementation

Let us have a look on each axis and how it can benefit from these indices. In following descriptions, we expect to have context node c and location step `axis::x` where the `axis` will vary. In every case, we extract the reference list from element index using `x` as key, so we have list of all `x` elements at our disposal.

Implementation of the *descendant* axis is quite simple. The index reference list is ordered by left-values of the nodes. Furthermore, we know that every descendant of context element c has its left-value in range $(l(c), r(c)\rangle$. Such nodes are definitely stored in continuous subrange of the reference list. A binary-search algorithm can be applied to find both beginning and end of the subrange.

The *following* axis uses the same approach. If we have an element c, all following nodes have their left-values greater than $r(c)$. We simply find the first one and then take all nodes from its position to the end of the list.

The *preceding* axis is a little more complicated than the previous two. In order to determine whether node u precedes context c, we need to compare $r(u)$ with $l(c)$. Unfortunately, the reference list is not ordered by right-values. As the right-value is always greater or equal left-value (of the same node), we will use all nodes with left-value lesser than $l(c)$ as *candidates*. These candidates contain two types of nodes – predecessors and ancestors of c. We can test right-value of each candidate and filter out the predecessors.

We might create second element index where references will be ordered by right-values and implement evaluation of the *preceding* axis analogically to the *following* axis. However, we did not use it in our implementation.

The *ancestor* axis cannot be effectively accelerated by this index. On the other hand, we expect that XML documents are shallow, therefore the best way to find all ancestors of context node is to follow parental links in XML tree structure and filter nodes one by one.

Techniques described for *descendant*, *following*, and *preceding* axes could be extended also to *child*, *following-sibling*, and *preceding-sibling* relations. In order to do that, an element index must be also built for every node with children. Such index contains only children elements of associated node. The implementation would be analogical.

4 Algorithms

Before we describe optimization and implementation details of our XPath processor, we will revise recursive algorithm for location path evaluation. A location path consists of sequence of location steps. Each step takes node-set produced by previous step (called *initial node-set*) and generates another node-set which is used by successive step. First step uses the context node as a singleton and last step produces node-set which is also the result of the whole location path.

The recursive algorithm evaluates location steps as follows. First, an intermediate node-set S_i is generated for every node v_i in the initial set by application of location axis on v_i and node test to filter out nodes of specific type or name[1]. Each set S_i is filtered by predicates and S'_i ($S'_i \subseteq S_i$) is produced. Finally, all S'_i sets are united and the union is yielded as the result of the location step.

When a node-set is filtered by a predicate, the predicate is executed recursively for every node in the set using the node, its position and size of the set as context values. Result of the predicate is converted into Boolean value, and if *true*, the tested node is included in the result. A location step may have more than one predicate. In that case, predicates are applied one by one, each producing another node-set which is handed over to successive predicate.

[1] In our case, we focus only on name filters.

As mentioned before, the recursive algorithm is not very efficient. Now, we describe the most important optimizations used in our implementation.

4.1 Minimal Context Optimizations

First, we define *context flags* for each XPath expression (and subexpression):

- flag n for context node,
- flag p for position,
- and flag s for size.

These flags indicate, which parts of evaluation context are required by the expression. Expression E is tagged with set of flags denoted $\mathit{cf}(E) \subseteq \{n, p, s\}$. When a flag is present in the set $\mathit{cf}(E)$, corresponding part of context is required for evaluation of E. Formally, we define *cf* as shown in Table 1.

expression E	context flags $\mathit{cf}(E)$
arithmetic operators $(+, -, *, \mathrm{div}, \mathrm{mod})$	$\mathit{cf}(\mathit{operand}_1) \cup \mathit{cf}(\mathit{operand}_2)$
comparisons $(<, \leq, >, \geq, =, \neq)$	$\mathit{cf}(\mathit{operand}_1) \cup \mathit{cf}(\mathit{operand}_2)$
logical operators (and, or)	$\mathit{cf}(\mathit{operand}_1) \cup \mathit{cf}(\mathit{operand}_2)$
absolute location path	$\{\}$
relative location path	$\{n\}$
location step	$\{n\}$
[*predicate*]	$\mathit{cf}(\mathit{predicate})$
union $(\vert)$	$\mathit{cf}(\mathit{operand}_1) \cup \mathit{cf}(\mathit{operand}_2)$
position()	$\{p\}$
last()	$\{s\}$
literals and functions without arguments	$\{\}$
functions with arguments	$\bigcup \mathit{cf}(a), a \in \mathit{arguments}$

Table 1. Formal definition of context flags

Even though there are eight (2^3) types of expressions depending on which of n, p, s flags they have, we will streamline the situation by recognizing only three *context classes*:

- *Context-free* class represents expressions with empty *cf* set. These expressions are either constant (like literals) or contain absolute location paths.
- Expressions in *context-node* class requires only context node, thus their $\mathit{cf}(E) = \{n\}$. Typical representants are relative location paths and location steps.
- Finally, we place all remaining expressions in *full-context* class.

We call the context-free class the *weakest*, because empty context flag set is always a subset of any other class. Analogically, the full-context class will be the *strongest*. Expression with non-empty $cf \subseteq \{p, s\}$ is also considered to be *stronger* than context-node expression, as the context position and size is always generated together with nodes, hence node flag is in fact implicitly included.

When evaluating location step, many nodes can be filtered by predicates multiple times if they appear in more than one intermediate (S_i) set. This may be unavoidable since the node can be at different positions or the S_i sets can have different sizes, thus they create a different context for the predicate evaluation. However, if there are no predicates which require position or size context value, the filtering procedure can be optimized.

Formally, if all predicates of a location step are from context-node or context-free class, we may unite all S_i sets before they are filtered by predicates. Then the union is filtered instead, thus predicates are executed for each node at most once. Furthermore, if there are some predicates from context-free class, we need not evaluate them for every node, as it is sufficient to execute them just once. If they resolve *true*, they can be removed from predicate list since they do not restrict the result set in any way. Otherwise (if they turn *false*), we need not resolve the location step nor the remaining part of the location path any more as it will never yield any nodes.

4.2 Vectorization of Axes

Previous optimization can be used to improve processing of axes in location steps. When axis is processed, single node is taken as input and a set of all conforming nodes is returned. If there are no full-context predicates in the location step, the intermediate node-sets (S_i) are united right after the axis part of the step (with the name filter) is resolved. Knowing this, we can optimize axis processing so it will not retrieve intermediate set for each node, but rather use entire initial node-set as input and yield the union of node-sets S_i. In following algorithms we expect that a node-set is represented by an array of nodes where nodes are ordered by their left-values (i.e. in the document order).

Descendant axis utilizes following observations: descendants of a single node are stored in continuous subrange of element index (as described in Section 3) and descendant sets of two following nodes have empty intersection. Furthermore, if we have initial nodes u and v where v is descendant of u, descendants of v are also descendants of u, therefore we need to process only node u when retrieving descendants. The algorithm will work as follows.

```
**for** $i = 1$ to $|S|$ **do**
  **if** $i = 1$ *or not*($S[i]$ descendant of *last*) **then**
    *last* $\leftarrow S[i]$
    append descendants of $S[i]$ to the result
  **end if**
**end for**
```

Following and preceding axes can be accelerated greatly by this optimization. First, we have to find initial node with the lowest right-value (for the following axis) or the greatest left-value (for the preceding axis). Then we use this node as an input for simple version of axis resolution algorithm described in Section 3. Finding the initial node is trivial in case of the preceding axis, since the node with greatest left-value is always at the end of the set. In case of the following axis, situation is slightly more complicated. Node with the smallest right-value is either first node in the initial set or its descendant:

```
$i \leftarrow 1$
**while** $i < |S|$ *and* $r(S[i]) > r(S[i+1])$ **do**
  $i \leftarrow i + 1$
**end while**
return following nodes of $S[i]$
```

Almost every other axis may be accelerated as well using similar approach. We omit the description of the algorithms for the sake of scope. Complete overview can be found in [1].

4.3 Predicate Caches

Previous optimizations reduce necessary amount of work in many situations. However, the recursive algorithm still suffers from exponential time complexity. We shall avoid this problem using caches for predicate expressions. Problematic expressions (which are likely to be evaluated multiple times with the same context) will be *covered* by caches. If a cache covers an expression, all results of this expression are stored in cache (when first computed). When the expression is evaluated, the result is first looked up in the cache, thus each expression is executed for each context at most once.

The cache will reflect context class of the covered expression (e.g. expression from context-node class must be covered by context-node cache). The class of the cache defines which part of the context is used for indexing. Therefore, context-free cache stores single value, context-node cache is indexed by nodes and context-full cache utilizes context node, position and size. Full-context cache may consume up to $\mathcal{O}(N^4)$ memory (where N is size of the document); however, we may apply limits for cache size and make it forget records. Forgetful cache will not save us from exponential time complexity, but it still improves performance significantly.

Following rules are applied for caches:

- Literal values must not be covered by a cache directly.
- When an expression is covered by a cache, all its subexpressions from the same or stronger class are also considered to be covered. This rule is transitive and does not apply to predicates (a predicate cannot be covered transitively). All expressions from weaker classes must be covered by another cache of their own.

- If location step has a full-context predicate, all predicates of that step must be covered by caches. Otherwise, no predicate of that step needs to be covered, but these rules are applied recursively on nested predicates. Furthermore, when a predicate must be covered by cache, a cache with Boolean values is always used, considering the predicates are implicitly wrapped by $boolean()$ function.

The cache is implemented by concurrent hash-map from TBB library [13]. This hash-map works as hash table with concurrent access with fine grade locking. Locking is implemented by atomic operations and each position in the table can be locked individually. Therefore, collisions on locks will be rare. Unfortunately, the results indicate that locking on caches has significant impact on efficiency.

5 Parallelization

We have described data structures and algorithms which should allow us to exploit parallelism using standard templates. These templates (parallel-for, parallel-reduce, parallel-scan, etc.) and data structures (concurrent vector, concurrent hash-map, etc.) are described in literature [12]. We omit their description for the sake of scope.

The implementation presents many opportunities to apply these templates and concurrent data structures. We describe only the most interesting ones. More detailed description is provided in related work [1].

5.1 Predicate Filtering

We expect to have list of predicates $\mathcal{P}_i$ $(i = 1 \ldots \pi)$ and a node-set S being filtered. The filtering is processed as follows.

- First, all context-free predicates are resolved. Predicates resolved as *true* are immediately removed. When a single predicate is resolved as *false*, the whole filtering operation yields empty node-set since every node is exterminated by such predicate.
- Remaining predicates are grouped into *segments*. Full-context predicates must be positioned as first predicate in segment and there can be only one full-context predicate in each segment. Other predicates (from the context-node class) are grouped together to form as large segments as possible. For example, predicates[2] $nnfnfnn$ are grouped as $(nn)(fn)(fnn)$.
- Node-set S is filtered by first segment producing S', then S' is filtered by second segment and so on. Last segment produces the result of whole filtering operation.

[2] Where n denotes context-node predicate and f denotes full-context predicate.

The node-set is filtered by segment in two phases. In the first phase, all nodes from initial set are filtered by all predicates in the segment (the predicate order is respected). When a predicate is resolved as *false*, corresponding node in initial set is replaced by `null` value. In the second phase, non-`null` values are copied into filtered set.

The first phase utilizes parallel-for pattern. All nodes from the set are processed concurrently. Each task affects only its nodes, so no explicit synchronization is required. The second phase copies only valid nodes and must respect the ordering of the nodes. We use parallel-scan, where first pass (the pre-scan) only calculates new offsets for nodes (after `null`s are removed) and the second pass (final scan) copies the nodes.

5.2 Node-sets Merging

Some situations of XPath processing require merging node-sets (namely location steps with full-context predicates and the union operator). In both situations, a concurrent hash-map is used. All nodes are inserted into the hash-map (which ensures uniqueness of each node), then retrieved into an array and sorted. All three steps are performed in parallel fashion. Nodes that are inserted into hash-map are processed by parallel-for. The parallel-scan copies nodes into an array where first pass computes offsets and second pass copies the nodes. Finally, the array is sorted by parallel sort [12].

In case of union operator, we employ another parallel-for which processes operands of the union, so we have top level loop which operates over operands (and resulting node-sets) and nested loops which insert nodes into hash-map.

5.3 Processing Axes

There are hardly any opportunities for parallelism when an axis is processed for a single node. The only thing which can be done concurrently is copying sequence of nodes (using parallel-for). However, even when a location step is processed without optimizations, we may still process each node from initial set (i.e. resolve the axis and filter it by predicates) concurrently.

Vectorized processing of axes presents more opportunities for parallelism. In many cases (e.g. descendant axis, child axis, ...), node-sets produced by initial nodes are disjoint. Therefore, we need not use concurrent hash-map (which requires locking) to merge them. Instead, we use parallel-scan to assemble merged sets in correct order. The parallel scan is used as usual – first pass computes offsets for nodes and the second pass copies the nodes.

6 Practical Tests

Practical experiments were designed for two things. First, we would like to determine scalability of the solution and to measure speedup on multiple cores. Second, we compare our solution with existing libraries for XPath processing.

The comparison is relevant only for our implementation restricted to single core as both libxml and Xalan are single-threaded.

6.1 Methodology, Testing Data, Hardware Specifications

Performed tests will focus solely on execution speed. We will measure time required to evaluate a query using system real-time clock. Other operations such as loading XML data or processing results will not concern us. Real-time clock will better reflect the practical characteristics of the implementation and cover both application and kernel time (thus include context switching, thread synchronization, etc.).

Each query is executed 10×. *Raw average* is computed as arithmetic average of all 10 times. Then, each measured time which is greater than raw average multiplied by 1.25 (i.e. with 25% tolerance) is excluded. Final time is computed as arithmetic average of remaining values.

It is still possible that all ten measured values are distorted due to some long lasting activity running on the system at the same time as the tests. Therefore, each test-set was repeated three times in different daytime. These three results were closely compared and if one of the values was obviously tainted, the test was repeated. A result is considered tainted if it deviates from the other two by more than 25%.

Document used for testing was generated by xmlgen, a XML document generator developed under XMark [8] project. This document simulates an auction website (a real e-commerce application) and contains over 3 million elements.

Queries evaluated on the document are taken from XPathMark performance tests [9]. These queries are especially designed to determine speed of tested XPath implementation. Queries that were not compatible with our subset of XPath were omitted. Unfortunately, some of the results are not presented here due to lack of space. Complete data and results may be found in [1].

All test were performed on Dell M905 server with four six-core AMD Opterons 8431 (i.e. 24 cores) clocked at 2.4 GHz. Server was equipped with 96 GB of RAM organized in 4-node NUMA. A RedHat Enterprise Linux (version 5.4) was used as operating system.

6.2 Results

Table 2 summarizes times (in ms) measured for our implementation, libxml and Xalan libraries. Columns $\mathcal{C}_t$ show times required by our implementation on t threads. Symbol ∞ represents times that were completely out of scale[3].

The results suggest that our implementation is highly scalable. The best speedup was observed at query B_6 which runs 18.8× faster on 24 cores than on single core. Unfortunately, some queries (A_2, A_3, C_2, D_2, and E_4) have shown poor speedup or even slow-down. This is mostly caused by the structure of these queries. When intermediate results between location steps are too small,

[3] Times greater than 2 millions ms.

	$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_4$	$\mathcal{C}_8$	$\mathcal{C}_{16}$	$\mathcal{C}_{24}$	libxml	Xalan
A_2	2.531	3.317	2.603	2.381	2.488	2.157	284.1	1054
A_3	2.816	3.520	2.562	2.43	2.366	2.127	312.5	305.8
B_5	629.9	337.1	208.9	135.3	85.38	80.94	∞	∞
B_6	5542	2809	1470	755.6	400	295.2	∞	∞
B_9	17830	9257	4769	2687	1654	1422	∞	∞
B_{10}	8955	4663	2396	1354	835.8	707.7	∞	∞
C_1	65.33	38.15	21.1	17.09	19.4	19.09	516.8	57.04
C_2	176.4	131.7	137.2	162.7	159.3	135.1	118.6	88.45
D_2	0.925	2.216	1.095	1.531	2.377	1.754	3922	45910
E_4	360.1	306.8	322.4	372.6	384.2	338.9	2891	369.9
E_5	6325	3304	1724	907.5	490.1	369.1	∞	∞

Table 2. Times in ms for XPathMark [9] tests

the query cannot benefit from data parallelism very much. Almost all of these queries (except for C_2 and E_4) are resolved very quickly on single core, therefore we need not parallelize them at all. Queries A_2, A_3, and D_2 are even slower on two cores than on single core. This is caused by locking overhead of caches. Finally, we have observed some fluctuations between times on 8, 16, and 24 cores which are most likely caused by effects of memory access on NUMA systems and thread distribution among real processors.

In comparison with known XPath processors [10][11], our implementation outperforms them in almost every test. The most significant difference is in queries B_5, B_6, B_9, B_{10}, and E_5 which all contain following or preceding axis. However, queries C_1 and C_2 are resolved slightly slower in our implementation, which is most likely caused by better optimizations of comparisons in libxml and Xalan.

7 Conclusions

This paper presents algorithms and indexing data structures for XPath processing which can be easily adapted by standard parallelization templates. Our experimental results demonstrate that proposed algorithms scale very well for long lasting queries. These algorithms are beneficial also for sequential processing as they outperform libxml and Xalan libraries in almost every test even on single core.

Our implementation uses some data structures which require locking. These structures are most likely responsible for poor speedup of some tested queries. Furthermore, our implementation does not analyze executed query nor data to determine whether particular part of the query should be parallelized or not. We use simple greedy method to partition workload and hope for the best.

We will focus on removing locking completely and try to design metrics for query evaluation cost in our future work.

References

1. M. Kruliš, J. Yaghob. Algorithms for Parallel Searching in XML Datasets, 2009. `http://www.ksi.mff.cuni.cz/~krulis/?page=study/master_thesis`
2. G. Gottlob, C. Koch, R. Pichler. Efficient algorithms for processing XPath queries. ACM Trans. Database Syst., 30(2):444491. ACM, New York, NY, USA, 2005.
3. Lu, W. and Gannon, D. Parallel XML Processing by Work Stealing SOCP '07: Proceedings of the 2007 workshop on Service-oriented computing performance: aspects, issues, and approaches
4. Bordawekar, R. and Lim, L. and Shmueli, O. Parallelization of XPath queries using multi-core processors: challenges and experiences, EDBT '09: Proceedings of the 12th International Conference on Extending Database Technology
5. Bordawekar, R. and Lim, L. and Kementsietsidis, A. and Kok, B.W.L. To Parallelize or Not to Parallelize: XPath Queries on Multi-core Systems
6. Clark, S. DeRose, et al. XML Path Language (XPath) Version 1.0. W3C Recommendation, 16:1999, 1999.
7. XML Document Object Model. `http://www.w3.org/DOM/`
8. XMark - benchmark for various XML technologies. `http://www.xmlbenchmark.org/`
9. XPathMark - benchmark for XPath 1.0. `http://sole.dimi.uniud.it/~massimo.franceschet/xpathmark/`
10. Libxml2 - The XML Library for GNOME. `http://xmlsoft.org/`
11. Xalan for C++ - XSLT and XPath library developed by Appache. `http://xml.apache.org/xalan-c/`
12. J. Reinders. Intel threading building blocks. OReilly & Associates, Inc., Sebastopol, CA, USA, 2007.
13. Intel Threading Building Blocks – an open source library for parallel programming. `http://www.threadingbuildingblocks.org/`.
14. Pan, Y. and Lu, W. and Zhang, Y. and Chiu, K. A static load-balancing scheme for parallel xml parsing on multicore cpus. IEEE International Symposium on Cluster Computing and the Grid, Rio de Janeiro, 2007
15. Wu, Yu, Qi Zhang, Zhiqiang Yu and Jianhui Li. "A Hybrid Parallel Processing for XML Parsing and Schema Validation." Presented at Balisage: The Markup Conference 2008, Montral, Canada, August 12 - 15, 2008. In Proceedings of Balisage: The Markup Conference 2008. Balisage Series on Markup Technologies, vol. 1 (2008). doi:10.4242/BalisageVol1.Wu01.
16. T. Grust. Accelerating XPath location steps. pages 109120. ACM, New York, NY, USA, 2002.
17. M. Kratky, J. Pokorny, and V. Snasel. Implementation of XPath axes in the multi-dimensional approach to indexing XML data. Current Trends in Database Technology-Edbt 2004 Workshops: EDBT 2004 Workshops, PhD, DataX, PIM, P2P&DB, and Clustweb, Heraklion, Crete, Greece, March 14-18, 2004: Revised Selected Papers, page 219. Springer, 2004.
18. I. Mlýnková, K. Toman, and J. Pokorný. Statistical Analysis of Real XML Data Collections.
    In COMAD06: Proc. of the 13th Int. Conf. on Management of Data, pages 2031, New Delhi, India, 2006. Tata McGraw-Hill Publishing Company Limited.

Fast Fibonacci Encoding Algorithm[⋆]

Jiří Walder, Michal Krátký, and Jan Platoš

Department of Computer Science
VŠB–Technical University of Ostrava
17. listopadu 15, Ostrava–Poruba, Czech Republic
{jiri.walder,michal.kratky,jan.platos}@vsb.cz

Abstract. Data compression has been widely applied in many data processing areas. Compression methods use variable-length codes with the shorter codes assigned to symbols or groups of symbols that appear in the data frequently. Fibonacci code, as a representative of these codes, is often utilized for the compression of small numbers. Time consumption of encoding as well as decoding algorithms is important for some applications in the data processing area. In this case, efficiency of these algorithms is extremely important. There are some works related to the fast decoding of variable-length codes. In this paper, we introduce the Fast Fibonacci encoding algorithm; our approach is up-to 4.6× more efficient than the conventional bit-oriented algorithm.

1 Introduction

Data compression has been widely applied in many data processing areas. Various compression algorithms have been developed for processing text documents, images, video, etc. In particular, data compression is of the foremost importance and has been well researched as it is presented in excellent surveys [13, 18].

Various codes have been applied for data compression [14]. In contrast with fixed-length codes, statistical methods use variable-length codes, with the shorter codes assigned to symbols or groups of symbols that have a higher probability of occurrence. People who design and implement variable-length codes have to deal with these two problems: (1) assigning codes that can be decoded unambiguously and (2) assigning codes with the minimum average size.

In some applications, a prefix code is required to code a set of integers whose length is not known in advance. The prefix code is a variable-length code that satisfies the prefix property. As we know, the binary representation of integers does not satisfy this condition. In other words, the size n of the set of integers has to be known in advance for the binary representation since it determines the code size as $1 + \lfloor \log_2 n \rfloor$. Fibonacci coding is distinguished as a suitable coding for a compression of small numbers [13].

[⋆] Work is partially supported by Grants of GACR No. P202/10/0573 and SGS, VŠB–Technical University of Ostrava, No. SP/2010138.

The time consumption of decompression is more critical than the time of compression; therefore, efficient decompression algorithms were studied in many works for the decompression of data structures [15, 6, 16] or text files [12, 3]. In the case of physical implementation of database systems, retrieval of compressed data structure's pages may be more efficient than retrieval of uncompressed pages due to the fact that the cost of decompression is lower than the cost of page accessing in the secondary storage [16, 2].

Since fast decoding algorithms have not yet been known, variable-length codes have not been used to compression of data structures, and, in generally, in the data processing area. The first effort of the fast decoding algorithm for Fibonacci codes of order ≥ 2 has been proposed in [7, 8]. We studied fast decoding algorithms of various variable-length codes in our previous work [17]. The fast encoding algorithms for the Fibonacci code yet not has been studied. In the case of data structures, pages are decompressed during every reading from the secondary storage into the main memory or items of a page are decompressed during every access to the page. If insert or update operations are considered, data compression becomes more significant.

In this article, we present Fast encoding algorithm of the Fibonacci of order 2 code. In Section 2, we describe the conventional Fibonacci of order 2 encoding algorithm. In Section 3, we provide a theoretical background of the fast encoding algorithm based on Fibonacci right shift and Encoding-Interval table. In Section 4, experimental results are presented and the proposed algorithm is compared to the conventional approach. In the last section, we conclude this paper and outline future works.

2 Fibonacci Coding

In this section, the theoretical background of Fibonacci of order 2 is briefly described. This universal code introduced by Apostolico and Fraenkel in [1] is based on the Fibonacci numbers [10]. The sequence of Fibonacci numbers is defined as follows:

$$F_i = F_{i-1} + F_{i-2} \text{ , for } i \geq 1,$$
$$\text{where } F_{-1} = F_0 = 1.$$

Definition 1. *(Fibonacci binary encoding and computation of its value)*

Let $F(n) = a_0 a_1 a_2 \ldots a_p$ be the Fibonacci binary encoding of a positive integer n. The value of the Fibonacci binary encoding, denoted $V(F(n))$, is defined as follows:

$$V(F(n)) = n = \sum_{i=0}^{p} a_i F_i \ (a_i \in \{0,1\}, 0 \leq i \leq p)$$

In the Fibonacci binary encoding, each bit represents a Fibonacci number F_i. Such a number has the property of not containing any sequence of two consecutive 1-bits [1]. This property is utilized for the construction of the Fibonacci code $\mathcal{F}(n)$ of number n. Fibonacci code $\mathcal{F}(n)$ maps n onto a binary string so that the string ends with a sequence of two consecutive 1-bits. The Fibonacci codes for some integers are shown in Table 1.

Table 1. Examples of Fibonacci codes for some integers

n	$F(n)$							$\mathcal{F}(n)$
1	1							11
2	0	1						011
3	0	0	1					0011
4	1	0	1					1011
5	0	0	0	1				00011
6	1	0	0	1				10011
7	0	1	0	1				01011
8	0	0	0	0	1			000011
16	0	0	1	0	0	1		0010011
32	0	0	1	0	1	0	1	00101011
i	0	1	2	3	4	5	6	
F_i	1	2	3	5	8	13	21	

In Algorithm 1, we see how the conventional bit-oriented algorithm encodes a positive integer n into a Fibonacci code. This algorithm outputs the encoded number into Fn and its length into LFn. Due to the fact that bits of Fibonacci coding are encoded in the reverse order we must use the temporary Fn variable. Bits in the variable are right-shifted in each inner loop.

3 Fast Fibonacci Encoding Algorithm

The main issue of the conventional encoding algorithm is handling encoded numbers in the bit-by-bit manner. To design a fast encoding algorithm, encoded numbers are separated into segments larger than one bit. Similar principles have been utilized in fast decoding algorithms [17, 8, 9]. The separation of encoded numbers utilizes the Fibonacci right shift operation introduced in [17, 9]. Individual segments are then encoded by the precomputed *Encoding-Interval table.* When all individual segments are encoded, they are put together into the complete code. The Fibonacci right shift and Encoding-Interval table are presented in Section 3.1. The fast algorithm is explained in Section 3.2.

3.1 Fibonacci Shift Operation and Encoding-Interval Table

The Fibonacci shift operation introduced in [17, 9] is required for the bit manipulation in fast encoding and decoding algorithms. In this paper, we introduce an efficient computation of this operation.

```
input  : n, a positive integer
output: Fn, encoded number by Fibonacci code of order 2 with LFn length
p ← 0 ;
while $F_p$ ≤ n do p ←p +1;
p ← p − 1;
Fn ← 1;
LFn ← 1;
while p ≥ 0 do
    Fn ← Fn << 1;
    if $F_p$ ≤ n then
        Fn ← Fn | 1;
        n ← n − $F_p$;
    end
    LFn ←LFn +1;
    p ← p − 1;
end
```

Algorithm 1: Conventional encoding algorithm for the Fibonacci code of order 2

Definition 2. *Fibonacci shift operation*

Let $F(n) = a_0 a_1 a_2 \dots a_p$ be a Fibonacci binary encoding, k be an integer, $k \geq 0$. The k-th Fibonacci left shift $F(n) <<_F k$ is defined as follows:

$$F(n) <<_F k = \overbrace{00\dots0}^{k} a_0 a_1 a_2 \dots a_p$$

Fibonacci right shift is defined as is follows:

$$F(n) >>_F k = a_k a_{k+1} a_{k+2} \dots a_p$$

We utilize k-Fibonacci right shifts for the separation of numbers into segments. We do not need all k-Fibonacci right shifts, we only need multiplies of the segment size S. If we consider 32 bit-length integers than the length of the largest code is $L(F(2^{32})) = 46$; therefore, $k \in \{0, 8, 16, 24, 32, 40\}$ for $S = 8$ and $k \in \{0, 16, 32\}$ for $S = 16$.

The conventional approach to the calculation of Fibonacci right shift works in the following steps:

1. Compute $F(n)$ for the n value according to Definition 1.
2. Binary-shift the bits of $F(n)$ by k: $F(n) >> k$.
3. Compute the value of the shifted number $F(n) >> k$ according to Definition 1.

The Fibonacci right shift operation is time consuming since the Fibonacci value computation in Steps 1 and 3 requires a summarization of Fibonacci numbers for 1-bits; therefore, we utilize the Encoding-Interval table for the fast computation.

The Encoding-Interval table allows separating of numbers into segments with the k-th Fibonacci right shift and then direct translation of segments into Fibonacci codes. The size of the Encoding-Interval table depends on the size of the segment S. The segment size is usually one byte, i.e. $S = 8$. When we use larger segments the length of the Encoding-Interval table grows; on the other hand, the encoding becomes faster. There are $F_S - 1$ codes which can fit into one segment; it means $F_8 - 1 = 54$ codes which can fit into the 8 bit-length segment and $F_{16} - 1 = 2,583$ codes for the 16 bit-length segment.

Therefore, the total size of the Encoding-Interval table is:

$$table\_length = (F_S - 1) \times \left\lceil \frac{2^b}{S} \right\rceil$$

where b is the number of bits of the largest code.

Each line of the Encoding-Interval table is then built for all $F(n)$ codes which can fit into one segment and for all k-th Fibonacci right shifts. Each line includes the following values (see Table 3 for some examples):

- $F(n)$ – the Fibonacci code stored in the segment.
- $L(F(n))$ – the bit-length of the Fibonacci code
- k – the parameter k of the Fibonacci right shift operation
- n – the value stored in the actual segment $n = V(F(n))$
- $\langle n_{\min}, n_{\max} \rangle$ – an interval of numbers before the shift operation; this number is formally defined as follows: $\forall x \in \langle n_{\min}, n_{\max} \rangle : V(F(x) \;>>_F k) = n.$

This table can be used for the computation of the k-th Fibonacci right shift. For each x value we need to pass through the table to find the correct line where $x \in \langle n_{\min}, n_{\max} \rangle$. In this line we can directly read the shifted value $V(F(x)) >>_F k = n$ and also the corresponding Fibonacci code $F(n)$. Obviously, we need to pass only lines with correct k-th Fibonacci right shift.

To be able to compute a shifted value as fast as possible, we must consider the properties of the Fibonacci code. Let F_i denote the i-th term of the Fibonacci sequence then we can express each Fibonacci number by

$$F_i = \left\| b\phi^i \right\|$$

where ϕ is the well-known golden ratio [11] and a is the coefficient of the dominating term [9]. In the case of Fibonacci of order 2 the value $\phi = \frac{1+\sqrt{5}}{2} \approx 1.6180$ and $b = \frac{3\sqrt{5}+5}{10}$ [4]. The Fibonacci sequence calculated according to this formula is shown in Table 2.

Table 2. Examples of the Fibonacci sequence calculation

i	$a\phi^i$	$\left\|a\phi^i\right\|$
0	1.17	1
1	1.89	2
2	3.07	3
3	4.96	5
4	8.02	8
5	12.98	13
6	21.01	21
7	33.99	34
8	55	55
9	89	89
10	144	144

If we utilize this property, Fibonacci right shift $F(n) >>_F k$ is approximately calculated by the following equation:

$$a_0a_1a_2\dots a_p >>_F k = \sum_{i=0}^{p} a_iF_i >>_F k = \sum_{i=0}^{p} a_i \left\|b\phi^i\right\| >>_F k$$
$$\approx \sum_{i=0}^{p} a_ib\phi^i >>_F k = \frac{\sum_{i=0}^{p} a_ib\phi^i}{\phi^k} = \sum_{i=0}^{p} a_ib\phi^{i-k} \approx \sum_{i=0}^{p} a_i \left\|b\phi^{i-k}\right\| =$$
$$\sum_{i=0}^{p} a_iF_{i-k} = \sum_{i=-k}^{p-k} a_{i-k}F_i = \sum_{i=0}^{p-k} a_{i-k}F_i = a_ka_{k+1}a_{k+1}\dots a_p.$$

The shift operation result is not calculated precisely due to the fact that we ignore twice rounds. We can simply rewrite the approximate computation of the Fibonacci right shift as follows:

$$V(F(n) >>_F k) \approx \left\|\frac{V(F(n))}{\phi^k}\right\|$$

The maximum error of the estimation is 1; by experiments we found, when the estimation misses, it is overestimated; therefore, the correct value is less by 1. To check if the estimation is correct we compare the estimation with a range in the Encoding-Interval table. If the check fails, we simply subtract the 1 value and receive the correct right shift value. In other words, this estimation decreases the linear complexity of the table scan to at most two attempts.

Example 1. First, let us consider $V(F(n)) = n = 265$; $F(n) = 001010100001$. We calculate the estimation of the 8-th Fibonacci right shift as follows:

$$V(F(265) >>_F 8) \approx \left\|\frac{265}{\phi^8}\right\| = \left\|\frac{265}{1.6180^8}\right\| = \left\|\frac{265}{46.9787}\right\| = \|5.4493\| = 5$$

Table 3 shows some lines of the Encoding-Interval table for the 8-th Fibonacci right shift. After checking the 5-th line of the table we see that the value lies in the interval $< 233; 287 >$; therefore, $V(0001) = 5$ is the correct value of the 8-th Fibonacci right shift.

Second, let us consider $V(F(n)) = n = 280$; $F(n) = 000001010001$. We calculate the estimation:

$$V(F(280) >>_F k) \approx \left\| \frac{280}{46.9787} \right\| = \|5.9601\| = 6$$

After checking the interval in the 6-th line of the Encoding-Interval table we see that the value not lies in the interval $< 288; 321 >$; therefore, the estimation is not correct and the correct value is less by 1. The correct value of the 8-th Fibonacci right shift of 280 is $V(0001) = 5$.

Table 3. Examples of the Encoding-Interval table for the 8-th and 16-th Fibonacci right shifts

n	$F(n)$	$L(F(n))$	k	n_{min}	n_{max}	n	$F(n)$	$L(F(n))$	k	n_{min}	n_{max}
1	1	1	8	55	88	1	1	1	16	2,584	4,180
2	2	2	8	89	143	2	2	2	16	4,181	6,764
3	4	3	8	144	198	3	4	3	16	6,765	9,348
4	5	3	8	199	232	4	5	3	16	9,349	10,945
5	8	4	8	233	287	5	8	4	16	10,946	13,529
6	9	4	8	288	321	6	9	4	16	13,530	15,126
7	10	4	8	322	376	7	10	4	16	15,127	17,710
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
27	73	7	8	1,275	1,308	27	73	7	16	59,898	61,494
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
46	149	8	8	2,173	2,206	46	149	8	16	102,085	103,681
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

3.2 Fast Fibonacci Encoding Algorithm

The fast encoding algorithm is shown in Algorithm 2. This algorithm utilizes the previously proposed Encoding-Interval table denoted by EIT. Due to the fact that the bits of the Fibonacci code are stored in the reverse order we need to write the bits of segments in the reverse order as well. Encoded segments or their parts are stored in a specific position of the result Fn by the $SetValueOfSegment$ function.

In Algorithm 2, n is encoded by Fibonacci coding; the result is stored in the Fn, the length is stored in the LFn. Short numbers with the code lower than F_8 are directly encoded by the Encoding-Interval table (see Lines 3–5). For larger numbers we first need to find the maximal k of the Fibonacci right shift (see Line 7). After the number is separated with k-Fibonacci right shift, it is encoded by the Encoding-Interval table and stored into the highest segment with position $k/8$ (see Lines 8–10). In Line 11, we subtract the minimal range of the encoded segment from the n. In Lines 12–20, all other segments except the lower one are separated by Fibonacci right shifts and they are encoded by the Encoding-Interval table. The length of the encoded number is increased by the segment size in Line 19. The lowest segment is encoded in Line 21. Finally, in Lines 23–24, the delimiter is put at the end of the encoded number.

```
input  : n, a positive integer
output: Fn, number encoded by Fibonacci code of order 2 with the LFn
        length
1  Fn ← 0 ;
2  k ← 8 ;
3  if n < $F_k$ then
4      LFn ← EIT[k ][Number].LFn;
5      SetValueOfSegment(Fn,0,EIT[k ][Number].Fn);
6  else
7      while n < $F_{k+8}$ do  k ← k +8;
8      n ← n $>>_F$ k;
9      LFn ← 8+ EIT[k ].LFn;
10     SetValueOfSegment(Fn,k/8,EIT[k ][n ].Fn);
11     n ← n −EIT[k ][n ].Nmin;
12     while k > 8 do
13         k ← k −8;
14         if n $\geq F_k$ then
15             n ← n $>>_F$ k;
16             n ← n − EIT[k ][n ].Nmin;
17             SetValueOfSegment(Fn,k/8,EIT[k ][n ].Fn);
18         end
19         LFn ←LFn +8;
20     end
21     SetValueOfSegment(Fn,0,EIT[k ][Number].Fn);
22 end
23 SetBit(Fn,LFn,1);
24 LFn ←LFn +1;
```

Algorithm 2: Fast encoding algorithm for the Fibonacci code of order 2

Example 2. Encoding of the number 17327 is depicted in Figure 1. The value of the Fibonacci code stored in all segments is $V(F(17327)) = 17327$. Encoding of the highest segment is depicted in Figure 2(a). After the 16-th Fibonacci right shift of the value $F(17327)$, i.e. $F(17327) >>_F 16$, we obtain the shifted Fibonacci code $F(7)$. The value after the shift is depicted in Line 3. It represents the Segment 2 value. We directly access the line 7 of the Encoding-Interval table by the 16-th Fibonacci right shift and we directly find the code $F(7) = 10$ with the length $L(F(7)) = 4$. The result is in the range $\langle 15127; 17710 \rangle$. The lower bound of the range is depicted in Line 2. The value of Segments 0 and 1 is obtained by subtracting the lower bound of the range from the $V(F(17327))$ number, i.e Segments 0 and 1 stores $V(F(17327)) - 15127 = V(F(2200))$. This encoding is carried out in Lines 8–11 of Algorithm 2.

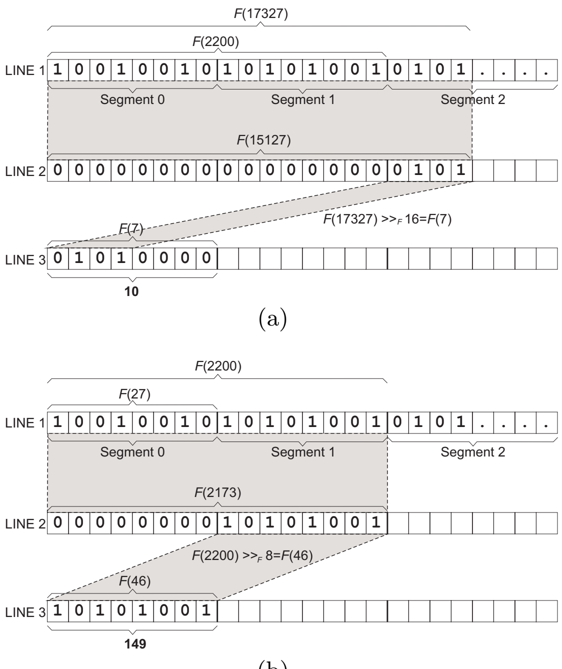


Fig. 1. Example of Fast Fibonacci encoding

Further encoding is depicted in Figure 2(b). The value of the Fibonacci code stored in Segments 0 and 1 is $V(F(2200)) = 2200$. After the 8-th Fibonacci right shift of the value $F(2200)$, i.e. $F(2200) >>_F 8$, we obtain the shifted Fibonacci code $F(46)$. The value after the shift is depicted in Line 3. It represents the separated value from Segment 1. We directly access the line 46 of the Encoding-Interval table by the 8-th Fibonacci right shift and we directly find the code $F(46) = 149$ with the length $L(F(46)) = 8$. The result is in the range $\langle 2173; 2206 \rangle$. The lower bound of the range is depicted in Line 2. The value of Segment 0 is obtained by subtracting the lower bound of the range from $V(F(2200))$ number, i.e Segment 0 includes $V(F(2200)) - 2173 = V(F(27))$. This encoding is carried out in Lines 12–20 of Algorithm 2.

The last Segment 0 with the value $V(F(27))$ is directly encoded into $F(27) = 73$ with the length $L(F(27)) = 7$ in Line 21 of Algorithm 2. We access the line 27 of the Encoding-Interval table for any shift to obtain this value, because we do not need any following subtraction.

Finally, the 1-bit delimiter is added in position $8+8+L(F(7))+1 = 8+8+4 = 12$ of the encoded number (see Lines 23–24 of Algorithm 2).

4 Experimental Results

The proposed Fast Fibonacci encoding algorithm has been tested and compared with the conventional algorithm. The algorithms' performance has been tested for various test collections. The tests were performed on a PC with dual core Intel 2.4GHz, 3 GB RAM using Windows 7 32-bit.

The test collections used in experiments have the same size: $10,000,000$ numbers. The proposed algorithm is universal and it may be applied for arbitrary numbers > 0. However, we worked with numbers $\leq 4,294,967,295$, it means the maximal value is the value of the 32 bit-length binary number. Tested collections are as follows:

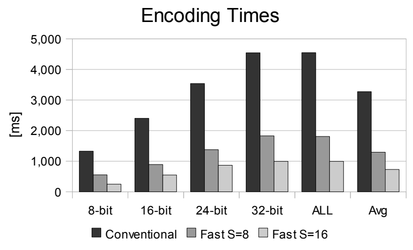
- 8-bit – a collection of random numbers ranging from 1 to 255
- 16-bit – a collection of random numbers ranging from 256 to 65,535
- 24-bit – a collection of random numbers ranging from 65,536 to 16,777,215
- 32-bit – a collection of random numbers ranging from 16,777,216 to 4,294,967,295
- ALL - a collection of random numbers ranging from 1 to 4,294,967,295

Table 4. Fast Fibonacci encoding times and speedup ratios for different random collections for conventional and fast algorithms with different segment sizes

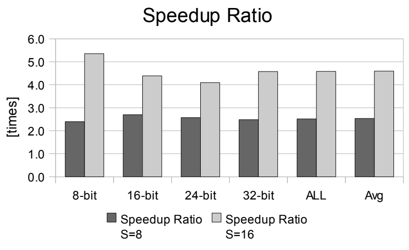
Algorithm / Collection	**Conventional**	**Fast $S = 8$**		**Fast $S = 16$**	
	Time [ms]	**Time** [ms]	**Speedup** [times]	**Time** [ms]	**Speedup** [times]
8-bit	1,327	553	2.4	248	5.4
16-bit	2,399	889	2.7	547	4.4
24-bit	3,538	1,375	2.6	865	4.1
32-bit	4,539	1,829	2.5	992	4.6
ALL	4,547	1,808	2.5	992	4.6
Avg.	**3,270**	**1,291**	**2.5**	**729**	**4.6**

We performed tests for segment sizes $S = 8$ and $S = 16$. We ran each test 10 times and calculated average values. Results of all tests are depicted in Table 4 and Figure 2. The Fast Fibonacci encoding algorithm is approximately $2.6\times$ faster than the conventional approach for the segment size $S = 8$ and $4.6\times$

faster for the segment size $S = 16$. The encoding times linearly increase with the bit-length of encoded numbers but the speedup ratio is not influenced by this increasing.



(a)



(b)

Fig. 2. (a) Encoding times for random collections and conventional and fast algorithms. (b) Speedup ratios between conventional and fast algorithms for random collections.

5 Conclusion

In this paper, the fast encoding algorithm for the Fibonacci code of order 2 is introduced. We introduced the effective implementation of Fibonacci right shift which is utilized by the encoding algorithm for separating integers into segments. The segments are directly translated into the Fibonacci code by the Encoding-Interval table. The improvement depends on the segment size used for the separation of the encoded numbers. For the segment size $S = 8$ (it means one byte), the Fast Fibonacci encoding is up-to 2.6× more efficient than the conventional algorithm. For the larger segment of two bytes in size (it means $S = 16$), the proposed algorithm is up-to 4.6× more efficient than the conventional algorithm. In our future work, we want to develop fast encoding algorithms for other universal codes like Elias-delta [5], Fibonacci code of order 3 [1], and so on.

References

1. A. Apostolico and A. Fraenkel. Robust Transmission of Unbounded Strings Using Fibonacci Representations. *IEEE Transactions on Information Theory*, 33(2):238–245, 1987.
2. R. Bača, J. Walder, M. Pawlas, and M. Krátký. Benchmarking the Compression of XML Node Streams. In *Proceedings of the BenchmarX 2010 International Workshop, DASFAA, Accepted.* Springer-Verlag, 2010.
3. T. C. Bell and I. H. Witten. *Text Compression.* Prentice Hall, 1990.

4. R. A. Dunlap. *The Golden Ratio and Fibonacci Numbers.* World Scientific Publishing Co. Pte. Ltd., 1997.
5. P. Elias. Universal Codeword Sets and Representations of the Integers. *IEEE Transactions on Information Theory*, IT-21(2):194–203, 1975.
6. J. Goldstein, R. Ramakrishnan, and U. Shaft. Compressing Relations and Indexes. In *Proceedings of the 14th International Conference on Data Engineering, ICDE 1998*, page 370, Los Alamitos, CA, USA, 1998. IEEE Computer Society.
7. S. T. Klein. Fast Decoding of Fibonacci Encoded Texts. In *Proceedings of the International Data Compression Conference, DCC'07*, page 388, Washington, DC, USA, 2007. IEEE Computer Society.
8. S. T. Klein and M. K. Ben-Nissan. Using Fibonacci Compression Codes as Alternatives to Dense Codes. In *Proceedings of the International Data Compression Conference, DCC'08*, pages 472–481, Washington, DC, USA, 2008. IEEE Computer Society.
9. S. T. Klein and M. K. Ben-Nissan. On the Usefulness of Fibonacci Compression Codes. *Accepted in Computer Journal, 2009*, 2009.
10. Leonardo of Pisa (known as Fibonacci). *Liber Abaci.* 1202.
11. M. Livio. *The Golden Ratio: The Story of Phi, the World's Most Astonishing Number.* Broadway, January 2003.
12. H. Plantinga. An Asymmetric, Semi-adaptive Text Compression Algorithm. In *Proceedings of the International Data Compression Conference, DCC 1994.* IEEE Computer Society, 1994.
13. D. Salomon. *Data Compression The Complete Reference.* Third Edition, Springer–Verlag, New York, 2004.
14. D. Salomon. *Variable-length Codes for Data Compression.* Springer-Verlag, 2007.
15. H. Samet. Data Structures for Quadtree Approximation and Compression. *Communications of the ACM archive*, 28(9):973–993, September 1985.
16. J. Walder, M. Krátký, and R. Bača. Benchmarking Coding Algorithms for the R-tree Compression. In *Proceedings of the Dateso 2009 Annual International Workshop on Databases, Texts, Specifications and Objects*, pages 32–43. CEUR Workshop Proceedings, Volume: 471, 2009.
17. J. Walder, M. Krátký, R. Bača, J. Platoš, and V. Snášel. Fast Decoding Algorithms for Variable-Lengths Codes. *Submitted in Information Science*, January, 2010.
18. I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes, Compressing and Indexing Documents and Images, 2nd edition.* Morgan Kaufmann, 1999.

iXUPT: Indexing XML Using Path Templates

Tomáš Bartoš and Ján Kasarda

Department of Software Engineering, Faculty of Mathematics and Physics,
Charles University in Prague, Malostranské nám. 25,
118 00 Prague, Czech Republic
`bartt4am@lab.ms.mff.cuni.cz, Jan.Kasarda@mff.cuni.cz`

Abstract. The XML format has become the standard for data exchange because it is self-describing and it stores not only information but also the relationships between data. Therefore it is used in very different areas. To find the right information in an XML file, we need to have a fast and an effective access to data. Similar to relational databases, we can create an *index* in order to speed up the querying for the information. There are several ways of indexing XML data but previous research showed that one of the most effective approaches is to index *root-to-leaf paths* in the input file. So we took the inspiration from existing path-based indexing concepts, enhanced those ideas, and created a new *native XML indexing* method derived from the combination of existing approaches in order to improve the evaluation time of regular path expressions in *XPath queries.*

Keywords: Indexing XML, path-based indexing, path templates, XPath queries, regular path expressions

1 Introduction

In past few years there has been an expansion of semi-structured data mostly stored as XML files and used for saving and exchanging information over the Internet. The simplicity is only one of the factors why the format became so popular. As more and more files occurred in this format, we wanted to access the stored data and search for the specific information according to prior criteria. For this purpose languages such as XPath [19] or XQuery [20] have been created. They allow searching for elements, attributes, or text values based on either specific values or regular expressions. If there are multiple conditions in an XPath query, we can combine them, and we get the *regular path expression* pattern.

The path expression usually matches several elements in the input XML file (the result set). The challenge is to find these elements quickly and efficiently, especially in large files with a high number of elements and with different structures. One came with an idea of *indexing* the XML data in order to quickly get the the results for any query.

No matter which indexing technique we use, if we had an XPath expression, the most problematic queries would be those with `'//'` (relative paths) or `'*'` (wildcards). These queries match numerous distinct elements and are difficult to handle compared to expressions with absolute paths only.

In this paper, our goal is to combine the best concepts of existing indexing methods and enhance them in order to improve the evaluation time of XPath queries. To achieve this, we make contributions to these areas:

– The previous research showed that indexing paths is one of the effective ways of indexing XML documents, so we use this approach and we create *a new indexing method* based on indexing paths (Section 2).
– Additionally, we combine it with one of the numbering schemes in order to accelerate the evaluation of XPath regular path expressions (see Section 3).
– Finally we compare the new concept with existing solutions in terms of time complexity while evaluating sample XPath queries (Section 4).

2 XML Indexing

There are various distinct approaches of indexing XML files. The main difference between them is that each method focuses on the specific topic such as decreasing the number of I/O operations [9], converting the XML format into tables in a relational DBMS to leverage the database engine ([8], [5], [16], [17] or GRIPP [6]), or relying on the simplicity of numbering schemes and joining elements (XISS [11] or twig joins [18]). The indexes might be based on a known data structure, e.g. the Patricia tries [12] are used in [7] or [9], but more often they use custom structures.

The method of labeling nodes in the tree with numbers might help with discovering ancestor-descendant (A-D) relationships. Dietz's numbering scheme [1] inspired us to design our numbering scheme based on intervals that evaluates A-D relationships in constant time. We found also motivation in XISS for the decomposition of XPath expressions, producing the intermediate results (called candidates in our approach), and element joins.

Specializing on paths, the DataGuide [2] handles raw paths and provided the basis for future path indexing methods such as XDG (Extended DataGuide [4]) or Index Fabric [7].

The interesting work of mapping all root-to-leaf paths into multi-dimensional points (MDX [3] or UB-trees [10]) influenced us on designing our indexing method. These concepts try to avoid using structural joins because of their time complexity compared to indexing paths. We understand the inefficiency of element-based approaches but we also see the potential slowdown for the multi-dimensional mapping methods when evaluating queries with multiple wildcards and relative paths. In this case, the domain used for finding matching tuples might grow faster.

The aim of the proposed indexing scheme is to follow the multi-dimensional techniques and leverage the path-based indexing with focus on grouping paths according to common characteristics, *path labels* (see Section 2.2). We expect this idea to eliminate many unsuitable paths and, as the result, to speed up the evaluation of any query (especially those problematic ones; see Section 3).

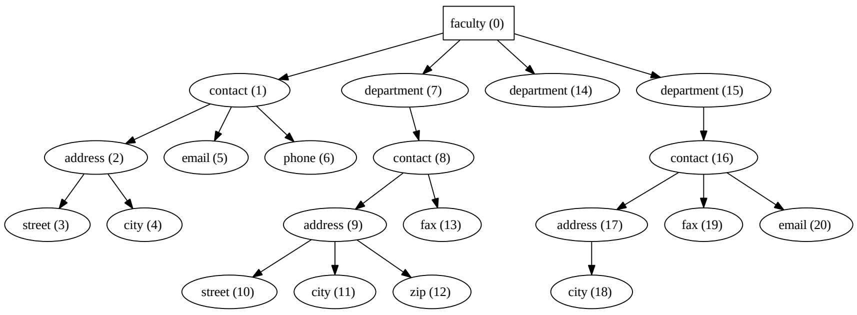


Fig. 1. The sample XML file

2.1 Graph-structured data

If we take XML files, they can be easily transformed into oriented graphs. The elements and attributes correspond to graph nodes and the edges are derived from the parent-child (or element-subelement) relationships. Generally, the graph might contain loops but if we discard the `IDREF` attributes, we will get a *tree* (see Figure 1). Our approach considers only tree structures as the input. We also exclude attributes and text values from the indexing and querying methods and focus primarily on the elements and their relationships. The support for indexing and evaluating attributes is straightforward (any attribute of an element might be considered as a specific subelement with a specific edge).

Although several various indexing methods exist, the main purpose remains the same - preprocess the source file and store information that will give fast response to almost any query. We suggest indexing paths, so we evaluate queries only on such paths that are common for as many elements from the query as possible. This method eliminates a lot of paths and elements that will not be in the result and therefore it improves the querying time.

2.2 Path-based Indexing

First of all, we assign elements in the source file the unique identification numbers (`NodeIDs`) and convert the element (tag) names into integers (`TagIDs`). We prefer numbers over strings because the comparison of integers is faster than comparing strings with variable lengths although it brings some memory overhead. We use the numbering method that assigns a node the `NodeID` when the corresponding element is visited for the first time (using the SAX parsing method). The root has the number 0, while the number of following nodes is always incremented by 1 (see the sample XML file with element names and `NodeIDs` in Figure 1).

Next, we store all root-to-leaf paths according to their *path labels*. Whenever we reach a leaf node, a new root-to-leaf path occurs, and we store the `Path reference` according to its path label. While the `Path reference` is indicated by the sequence of `NodeIDs`, the path label is determined by the sequence of `TagIDs` of all nodes on the path from the root to the current (leaf)

Table 1. Terms definition and description

Term	**Description**
`NodeID`	The unique node number given by the numbering method.
`LastNodeID`	`NodeID` of the last visited leaf node in the subtree determined by the current node. The interval (`NodeID,LastNodeID`) covers all subelements.
`TagName`	The name of an element.
`TagID`	The unique number for the `TagName`. For every new `TagName` we assign the next `TagID` (increased by 1).
`Path Label`	The sequence of `TagIDs` of the nodes on the path from the root.
`Path Prefix`	The sequence of `TagIDs` that identifies a prefix of at least one path.
`Path Template ID` (`PTID`)	The unique number for a path label. The path labels are converted into integers (`PTIDs`).
`Path Reference`	The sequence of `NodeIDs` of the nodes on the path from the root.

node. The path label is stored only once but one path label can contain several `Path references`. Moreover, we convert the path label into *Path Template ID* (`PTID`) that groups similar `Path references` together (for the detailed specification of used terms see Table 1).

Example 1. If we take the sample XML file from the Figure 1 and we visit a leaf node *fax (13)*, the `Path reference` (sequence of `NodeIDs`) will be `(0,7,8,13)`. Converting element names "*/faculty/department/contact/fax*" to `TagIDs`, we get the path label '`/0/7/1/9`'. For details about the conversion, see the Table 2(a) (`NodeTags` table) and the Section 2.3.

Table 2. Structures with data for the sample XML

(a) *NodeTags* table

TagName	**TagID**	**Path Template IDs**
faculty	0	{0,1,2,3,4,5,6,7,8,9}
contact	1	{0,1,2,3,4,5,6,7,9}
address	2	{0,1,4,5,6}
street	3	{0,4}
city	4	{1,5}
email	5	{2,9}
phone	6	{3}
department	7	{4,5,6,7,8,9}
zip	8	{6}
fax	9	{7}

(b) *Paths* table

PTID	**Path label**	**Path references**
0	0/1/2/3	{(0,1,2,3)}
1	0/1/2/4	{(0,1,2,4)}
2	0/1/5	{(0,1,5)}
3	0/1/6	{(0,1,6)}
4	0/7/1/2/3	{(0,7,8,9,10)}
5	0/7/1/2/4	{(0,7,8,9,11); (0,15,16,17,18)}
6	0/7/1/2/8	{(0,7,8,9,12)}
7	0/7/1/9	{(0,7,8,13); (0,15,16,19)}
8	0/7	{(0,14)}
9	0/7/1/5	{(0,15,16,20)}

2.3 Structures

While we parse the input XML file, we maintain several structures that store the root-to-leaf paths and other necessary information that we will later use when we evaluate queries. There are three major data structures: the `NodeTags` table, the `Paths` table, and the `PrefixTree`. While the first two are more like database tables, the last one is definitely a tree structure (inspired by a Patricia trie [12]).

NodeTags table. This table provides conversion between the `TagName` and the `TagID`. Furthermore it saves all path template IDs (`PTIDs`) where the given `TagID` appear; see the Table 2(a).

Paths table. This table contains the conversion between the `Path label` and the `PTID`. As mentioned before, a single `Path label` provides grouping for several `Path references` that are also stored here; see the Table 2(b).

Prefix tree. The `PrefixTree` covers all distinct *prefixes of path labels* from the source file. Each node in the tree has the `TagID` and the path from the root to a given node represents the path prefix. Nodes also contain all `PTIDs` which do have the selected prefix. We use this structure to find all `PTIDs` to which a specific `NodeID` might belong (for details see Section 3).

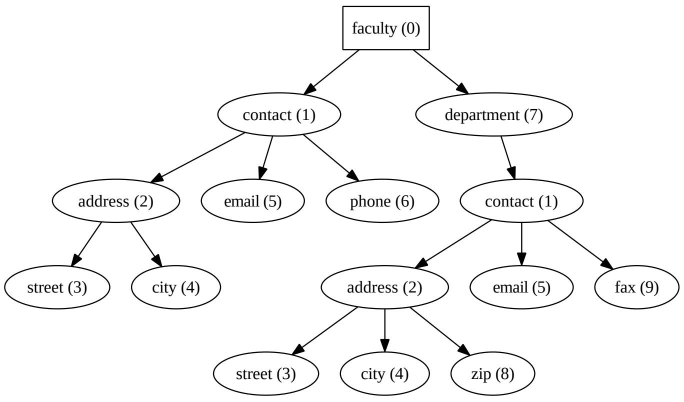


Fig. 2. *PrefixTree* for the sample XML file

Example 2. If we take the node *contact (1)* from the sample XML (see Figure 1), it belongs to almost all `PTIDs` (according to the `NodeTags` table). And using the `PrefixTree`, we find the prefix of `'/0/1'` that matches the `PTIDs` {0,1,2,3}. So the chosen node occurs only on paths with these path labels.

3 Evaluating XPath queries

Before we start evaluating an XPath query, we need to parse and prepare the query. We describe these steps in the following sections in more detail.

3.1 Parsing XPath expressions

In the beginning, we convert the string query, that describes a regular path expression, into the structure that is appropriate for the evaluation. We selected the graph structure (`XPathTree`) created by two types of nodes (`XPathNodes`): *steps* and *predicates* (see the sample queries in Figure 3).

Steps If we divide the path expression into sequence of step expressions, they will be represented by these nodes.

Predicates We express the further filter expressions by the *Predicate* nodes. While the *Step* nodes cannot be added as subnodes, each node in the tree might contain one or more *Predicate* subnodes.

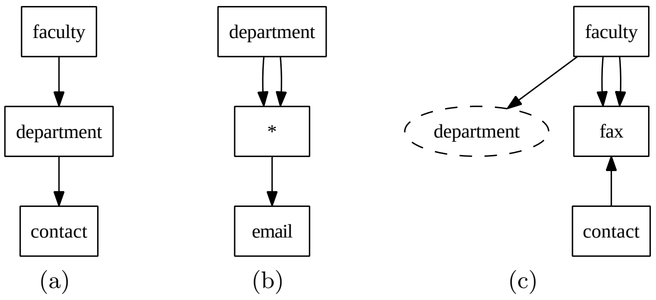


Fig. 3. The tree structure of sample queries. The *Steps* are represented by boxes, while the *Predicates* are shown as dashed ellipses. We use oriented edges to determine the forward and reverse axes (the edge leads from an ancestor to a descendant). We also put multiple edges to indicate that the node is generally a descendant (not necessarily a childnode) The corresponding XPath expressions are: (a) **faculty/department/contact**, (b) **department//*/email**, and (c) **faculty[department]//fax/ancestor::contact**

3.2 XPathTree pre-processing

After we parse the XPath expression and create the corresponding `XPathTree`, we need to convert the stored element names into `TagIDs`. While converting, we check element names whether they exist in the index. If a name does not occur in the `NodeTags` table, the result is instantly available because no nodes will be in the result set and no further evaluation is needed (we suppose all predicates to co-exist at the same time).

We also assign `XPathNodes` the integer `Order`, so we know the order in which they are visited and evaluated.

Algorithm 1 Visit procedure for evaluating a node in the XPathTree

```
Require: xnode is the current XPathNode in the XPathTree that is being visited
 1: xnode.Candidates = GetCandidates()
 2: for all (predicate in xnode.Predicates) do
 3:    Visit(predicate)  {recursive call for a predicate}
 4:    xnode.VoteForCandidateNodes(predicate.Candidates)
 5: end for
 6: if xnode.HasPredicates then
 7:    xnode.FilterCandidateNodes()
 8: end if
 9: if (xnode.IsStepNode) then
10:    MergeCandidates (lastNode, xnode)
11:    if (xnode.NextStepNode is not null) then
12:       Visit(xnode.NextStepNode)
13:    end if
14: end if
```

3.3 XPathTree evaluating

When we build and validate the `XPathTree`, we can start the evaluation. We use slightly different evaluating methods according to the type of the current `XPathNode`. When traversing the `XPathTree`, the *Step* nodes are evaluated with the top-down approach, while the *Predicate* nodes are processed in the bottom-up style (from the lowest level upwards using depth-first search). The reason for distinct methods is that all predicates must be resolved before we can continue with the next `XPathNode`.

The Algorithm 14 describes the procedure that we apply to all `XPathNodes`. There are several steps that we need to explain in more detail but the main principle is that we save candidate nodes (`NodeIDs`) for each `XPathNode` we visit. So the candidate nodes of the last *Step* node represent the result set.

1. **Save candidates**
   The first step is to save candidate nodes for the current node. We store them in the table called *candidates* (line 1). It contains `PTIDs`, where the `XPathNode` occur, with corresponding `NodeIDs`. Each `PTID` is stored only once but one `NodeID` might be saved for more `PTIDs`. It is because we focus later on merging *candidates* according to `PTIDs` rather than `NodeIDs`.
   To identify the `PTIDs`, we try to find the smallest `PTID` set that is common for as many `XPathNode` nodes as possible (using the `NodeTags` table). For a *predicate* node, this means to take `PTIDs` that are common for `XPathNodes` on the path from the last *Step* node. Because we evaluate predicates bottom-up, we create the set of `PTIDs` on the way "down". For a *Step* node we take the path from the first *Step* node. This holds only if all axis directions on the path are the same. If we have an alternating[1] `XPathNode`, it divides the

[1] Alternating XPathNode is a node that *changes* the axis direction (the incoming edge has different direction than the outgoing edge).

`XPathNodes` on the path into two groups for which the smallest `PTID` set must be computed separately. We pre-compute the minimal `PTID` set for all corresponding nodes when the first node in a specific group is visited.

When we obtain the minimal `PTID` set, we use the `Paths` table to find the candidate nodes (`NodeIDs`) according to the `Path references` for a `PTID`. If the `TagID` does not represent `'*'`, we find the positions of the `TagID` in the `Path label` identified by the current `PTID`. We search only for positions that are either *after* (forward axes) or *before* (reverse axes) the position of the last node and we take all `NodeIDs` on those positions from the `Path references`. If the `TagID` reflects `'*'` and the `XPathNode` does not have any predicates, we skip it and save the minimal and the maximal number of positions to be skipped when searching for positions in the next `XPathNode`. The numbers are determined by the current axis: $(1, 1)$ for the direct relative (parent, child), and $(1, \infty)$ for other axes (ancestor, descendant).

Example 3. If we take the Query 3 in Figure 3(c), the alternating `XPathNode` is the *fax* node. Therefore the minimal `PTID` subset can be computed only for set of nodes {faculty,fax} and {fax,contact}. The candidates for this `XPathNode` are shown in Table 3.

Table 3. Candidates table for the *fax node* in Figure 3(c)

PTID	NodeIDs
2	{5}
9	{20}

2. **Evaluate predicates and voting**
If there are any predicates for the current `XPathNode`, we need to handle them before we continue with the next `XPathNode`. Because predicates give additional filtering criteria, not all candidate nodes from the current `XPathNode`'s candidates will meet the new criteria. Therefore we use *voting* for candidate nodes (line 4). Every predicate gives a vote for all candidate nodes in the current `XPathNode`'s candidates table (no matter of their `PTIDs`) that are reachable from a `NodeID` stored in the predicate's candidates. The reachability is dedicated from the axis type and the `NodeIDs`. Because we consider only tree structures, we can use the interval (`NodeID,LastNodeID`) that is available to any `NodeID` to test whether a path between two `NodeIDs` exists. The path from a node n_1 to a node n_2 exists only if

$$(n_2.NodeID > n_1.NodeID) \text{ and } (n_2.NodeID \leq n_1.LastNodeID). \quad (1)$$

3. **Filter candidates**
Whenever we use the voting mechanism, we need to finalize the candidates. We call this action *filtering* candidates (line 7). The candidate nodes that

have not received enough *votes* will be removed. Number of votes needed for being kept equals the number of predicates that has been included in voting. After we eliminate unsuitable candidate nodes, we need to update the `PTIDs` for the next step. By updating `PTIDs` we mean find all `PTIDs` where a `NodeID` might occur. If the axis of the next `XPathNode` has the same direction, we take the `PTIDs` only from the smallest `PTID` set for the current `XPathNode`. Otherwise, we need to consider potentially all `PTIDs`. To take only correct `PTIDs`, we use the `PrefixTree` that determines only such `PTIDs` in which the given `NodeID` might occur. We cannot use only the `Paths` table because we will find `PTIDs` for any `NodeID` with the same `TagName` and that produces bigger set than we need for a specific `NodeID`. So we use the `PrefixTree` instead. The path prefix that we need for navigation in the `PrefixTree` is defined by the current `NodeID`.

4. **Merge candidates**
   If we have two *Step* `XPathNodes`, we need to merge their candidate nodes (line 10). Usually we take the candidates of the last *Step* `XPathNode` and apply the same voting mechanism on the current `XPathNode` as with predicates. After voting, we automatically filter candidates. The result for the current `XPathNode` contains previous candidate nodes that received votes.

4 Experimental results

We implemented the prototype of the *iXUPT* Index in .NET Framework 2.0 and use the laptop machine with Intel Core Duo processor (1.66GHz), 3GB main memory, and Windows XP SP2 installed to execute the experiments.

For our experiments, we use *XMark* [14] (the XML benchmark database) as the data set. *XMark* is designed to generate XML documents with multiple parts meeting various XML approaches (data-centric and document-centric). Although the generated documents are valid to a specific DTD, we do not use this schema as the hint for indexing or evaluating.

To acquire the data set, we use the tool *xmlgen* [22] which is the implementation of the *XMark*. We generated several documents with different characteristics (see the Table 4 for details).

Table 4. Characteristics of the XMark data set

Factor	Size [MB]	Nodes	#Real paths	#PTIDs
0.01	1.12	17 132	12 504	338
0.1	11.32	167 865	122 026	422
0.3	34.05	504 498	364 481	434
0.5	56.28	832 911	605 546	434
1	113.06	1 666 315	1 211 774	434

We provide also a comparison between a number of all root-to-leaf paths and a number of different `Path labels` (see Figure 4). The reason is that the number of different `Path labels` is smaller than the number of all paths (and might have an upper bound). So searching for candidate nodes on common `PTIDs` (that uniquely identify `Path labels`) enhance the speed of the evaluation.

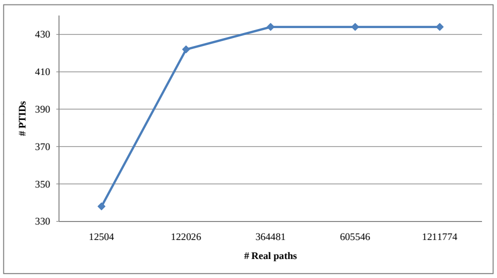
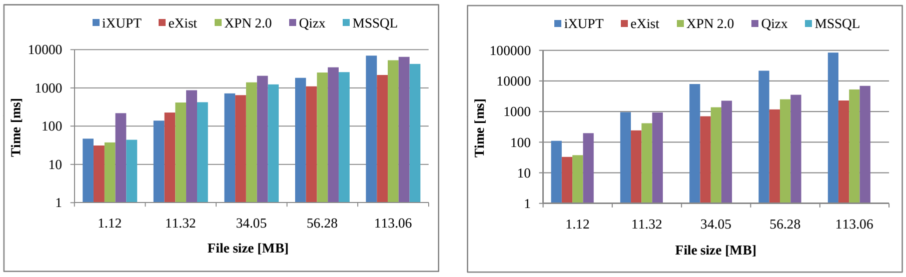


Fig. 4. The comparison between the number of real paths and the number of different *Path labels* (or PTIDs).

We choose two sample queries according to the given DTD of the generated documents that will cover as many features and functionality as possible.

1. `site/regions/*/item/location`
   The first query (Q1) is simple, there are no predicates and it uses mostly direct relatives (the child axis).
2. `//regions[europe]/ancestor::*/people//person`
   This query (Q2) is more complex, there is a predicate, a branching node, and an alternating node. Furthermore, the second *Step* node matches several elements and different axis directions are used.



(a) Q1 Results (b) Q2 Results

Fig. 5. The evaluation times for queries Q1 and Q2 in logarithm time scales.

To eliminate any negative effects of the managed code, we present the average time of 10 subsequent executions for each query. We compare the *iXUPT* prototype with several other products such as eXist [13], Qizx [23], MS SQL Server

2005 [24] (does not provide support for ancestor axis, so the query Q2 was not executed) or the built-in *XPathNavigator* in .NET Framework 2.0 (marked as XPN 2.0). The Figure 5 shows the evaluation times for both queries Q1 and Q2.

We can see the improvements of the query evaluation especially for the first query in the Figure 5(a). But we understand that the reason might be that we do not consider the time needed to create the index or that our index is fundamentally memory-based.

5 Conclusion and future work

In this paper, we have proposed a new XML indexing method based on storing root-to-leaf paths and grouping them according to common `Path labels` in order to enhance the evaluation time of regular path expressions in XPath queries. The experimental results showed that there is an improvement of the evaluation time in the category of small and medium-sized files. Although handling medium files is still comparable to other approaches, evaluating large files and complex queries did not bring the anticipated results.

For the future, there are still several issues in the current prototype that we would like to improve, such as speeding up the branch queries or optimizing the tree structure that stores the XPath query before evaluating. Next, we want to design an optimal structure for saving the *iXUPT* index to a hard drive. Furthermore, we would like to provide support for graph-oriented XML files (not only trees) which means to replace the interval-based path testing with a more general structure (such as Rho-index [15]). Finally, we aspire to use our indexing method in the environment of distributed XML processing.

6 Acknowledgments

This research has been partly supported by Czech Science Foundation (GAČR) project Nr. 201/09/0683, by institutional research plan number MSM0021620838, and by Czech Science Foundation (GAČR), grant Nr. P202/10/0573.

References

1. Paul F. Dietz: *Maintaining a Order in a linked list.* Proceedings of the 14th Annual ACM Symposium on Theory of Computing. San Francisco, California (1982) 122–127.
2. R. Goldman, J. Widom: *DataGuides: Enable query formulation and optimization in semistructured databases.* Proceedings of 23rd International Conference on Very Large Data Bases. Athens, Greece (1997) 436–445.
3. M. Krátký, R. Bača, V. Snášel: *On the Efficient Processing Regular Path Expressions of an Enormous Volumne of XML Data.* Lecture Notes in Computer Science, Vol. 4653. Springer-Verlag, Germany (2007) 1–12.
4. J.M. Bremer, M.Gertz: *An Efficient XML Node Identification and Indexing Scheme.* Technical Report CSE-2003-04. Dept. of Computer Science, University of California at Davis, (2003).

5. G.Marks, M.Roantree: *Pattern Based Processing of XPath Queries.* IDEAS 2008 - International Symposium on Database Engineering and Applications. Coimbra, Portugal (2008).
6. S. Trißl, U.Leser: *Fast and Practical Indexing and Querying of Very Large Graphs.* Proceedings of the 2007 ACM SIGMOD international conference on Management of data. Beijing, China (2007).
7. B.f. Cooper, N. Sample, M.J. Franklin, G.R. Hjaltason, M. Shadmon: *A Fast Index for Semistructured Data.* Proceedings of the 27th International Conference on Very Large Data Bases. San Francisco, USA (2001) 341–350.
8. Torsten Grust: *Accelerating XPath Location Steps.* Proceedings of the 2002 ACM SIGMOD international conference on Management of data. New York, USA (2002) 109–120.
9. D. Barashev, B. Novikov: *Indexing XML to Support Path Expressions.* Proc. of the 6th East European Conf. on Advances in Databases and Information Systems (ADBIS 2002), Vol. 2: Research Communications. Bratislava, Slovakia (2002) 1–10.
10. M. Krátký, J. Pokorný, V. Snášel: *Indexing XML Data with UB-trees.* Proc. of the 6th East European Conf. on Advances in Databases and Information Systems (ADBIS 2002), Vol. 2: Research Communications. Bratislava, Slovakia (2002) 155–164.
11. Quanzhong Li, B.Moon: *Indexing and Querying XML Data for Regular Path Expressions.* Proceedings of the 27th VLDB Conference. Roma, Italy (2001) 361–370.
12. Donald Knuth: *The Art of Computer Programming.* Volume III, Sorting and Searching, Third Edition. Addison Wesley, Reading, MA (1998).
13. Wolfgang Meier: *eXist: An Open Source Native XML Database.* Lecture Notes in Computer Science, Vol. 2593/2009. Springer Berlin, Heidelberg (2003) 169–183.
14. A. Schmidt, F. Waas, I. Manolescu, M. Kersten, R. Busse, M.J. Carey: *XMark: A Benchmark for XML Data Management.* Proceedings of the 28th VLDB Conference. Hong Kong, China (2002) 974–985.
15. S. Bartoň, P. Zezula: *Rho-index - An Index for Graph Structured Data.* 8th International Workshop of the DELOS Network of Excellence on Digital Libraries. Schloss Dagstuhl, Germany (2005) 57–64.
16. M. Yoshikawa, T.Amagasa, T. Shimura and S. Uemura: *XRel: a Path-based Approach to Storage and Retrieval of XML Documents Using Relational Databases.* ACM Transactions on Internet Technology (TOIT). New York, NY, USA (2001) 110–141.
17. Zhiyuan Chen and G.J. Korn and F. Koudas and N. Shanmugasundaram and J.D. Srivastava: *Index Structures for Matching XML Twigs Using Relational Query Processors*, Data & Knowledge Engineering. Amsterdam, The Netherlands (2007) 283–302.
18. Chen, T. and Lu, J. and Ling, T.W.: *On Boosting Holism in XML Twig Pattern Matching Using Structural Indexing Techniques*, Proceedings of the 2005 ACM SIGMOD international conference on Management of data. New York, NY, USA (2005) 455–466.
19. XML Path Language (XPath) 2.0. http://www.w3.org/TR/xpath20.
20. XQuery 1.0: An XML Query Language. http://www.w3.org/TR/xquery.
21. eXist-db Open Source Native XML Database. http://exist.sourceforge.net.
22. Albrecht Schmidt: *xmlgen - The Benchmark Data Generator.* http://www.xml-benchmark.org.
23. Qizx, a fast XML repository and search engine fully supporting XQuery. http://www.xmlmind.com/qizx.
24. Microsoft SQL Server 2005. http://www.microsoft.com/sqlserver/2005.

Reverse-engineering of XML Schemas: A Survey*

Jakub Klímek and Martin Nečaský

XML Research Group, Department of Software Engineering
Faculty of Mathematics and Physics, Charles University in Prague
Malostranské náměstí 25, 118 00 Praha 1
The Czech Republic
{klimek, necasky}@ksi.mff.cuni.cz

Abstract. As approaches to conceptual modeling of XML data become more popular, a need arises to reverse-engineer existing schemas to the conceptual models. They make the management of XML schemas easier as well as provide means for accomplishing integration of various XML data sources. Some methods for reverse-engineering of XML schemas have been proposed and in this paper, they are compared using various criteria such as used XML schema languages, level of user involvement, number of XML schemas that can be covered by the conceptual model or support for consecutive XML schema evolution. They are also evaluated according to their potential to be used as parts of a system for management, evolution and integration of XML as a whole.

Keywords: XML, schema, reverse-engineering, conceptual modeling

1 Introduction

Today, XML [27] is a technology used in a wide variety of scenarios, from a message format used by web services [7] to storage of data in databases [4]. As the number of possible usages of XML grows, so does the need of easy management of large numbers of XML data sources and their integration. There we can use conceptual modeling of XML data. It allows a domain expert to model the problem domain independently of the implementation (XML in our case) and then create corresponding XML schemas, which are used to describe a structure of XML documents.

A common situation today is that a company uses several XML formats for various purposes and has these formats described by an XML schema. To ease the process of managing those formats and schemas as they evolve in time, the company can use a conceptual model such as [1, 2, 15, 16, 19, 23], to which the schemas would be connected. A problem usually arises when there is a new

* This work was supported in part by the Czech Science Foundation (GAČR), grant number P202/10/0573.

format that should be connected to the conceptual model, as most of the conceptual models do not support this operation well enough. During the process of connecting the new format to the conceptual model, the model may need to be extended, if the format contains a concept that was not covered by the model. A special case of this problem is, when the company does not have a conceptual model at all, and wants to create it from the schemas which they already have (they extend an empty model).

Therefore, an important aspect of the approaches used for conceptual modeling of XML data is if and how we can create the model from existing XML schemas (or connect a new schema to an existing model) and once we have it, if we can use it for the management of evolution of our set of schemas. The process of creating a conceptual model from existing XML schemas is what we call *Reverse-engineering of XML schemas.*

In this paper, we compare and evaluate approaches for reverse-engineering of XML schemas according to various criteria, including their usefulness as a method that can be integrated into a system for management of evolution of XML schemas.

1.1 Outline

The rest of this paper is structured as follows. In section 2, an introduction to the frequently used techniques in this area is given. In section 3 we introduce our framework for evolution and integration of XML schemas, against which the approaches will be evaluated. Section 4 contains a description of our comparison criteria. In section 5, we describe approaches to the problem, which reverse-engineer XML schemas to various user-friendlier models. In section 6, approaches to reverse-engineering to ontologies are described. In section 7 we summarize our findings and section 8 concludes.

2 Terms

In this section we provide an introduction to basic techniques used widely in the area of reverse-engineering of XML schemas.

2.1 XML schemas

In this paper, by *XML schema language* we mean one of the XML schema languages such as DTD [27], XML Schema [28], Relax NG [6], Schematron [12] etc. We state this because sometimes an XML schema gets confused with the actual XML Schema language.

2.2 Model-Driven Architecture

Model-Driven Architecture (MDA) [17] is a general approach to modeling software systems and can be profitably applied to data modeling as well. MDA distinguishes several types of models that are used for modeling at different levels of abstraction. For this paper, two types of models are important. A *Platform-Independent Model* (PIM) allows modeling data at the conceptual level. A PIM diagram is abstracted from a representation of the data in concrete data models such as relational or XML. A *Platform-Specific Model* (PSM) models how the data is represented in a target data model. For each target data model (such as XML), we need a special PSM that is able to capture its implementation details. A PSM diagram then models a representation of the problem domain in this particular target data model, it provides a mapping between the conceptual diagram and a target data model schema.

2.3 UML class diagrams

A large number of approaches to reverse-engineering use UML class diagrams as a PIM. Basically, it consists of *classes* representing concepts, *associations* representing relations and *attributes* of classes, representing properties of the concepts. For a more detailed description see [21, 22].

2.4 Schema matching

Most of the reverse engineering approaches use some methods of schema matching. They include string comparisons, data type compatibility measurements, structural similarity measurements and linguistic resources like thesauri and dictionaries. These methods are surveyed in detail in [26, 10]. There is one major difference between XML schema matching and reverse-engineering of XML schemas to conceptual models. XML schema matching usually works with two different XML schemas (written in XML schema languages) and the goal is to find mappings of components of one schema to the components of the second schema. On the other hand, reverse-engineering of XML schemas works with one XML schema and optionally a conceptual model. The goal is either to create the conceptual model when there is none, or to find appropriate mappings of the XML schema components to the components of the model, which can be written in e.g. UML, and therefore is of a whole different type.

3 Framework for evolution and integration of XML schemas

In this section, we introduce our framework for evolution and integration of XML schemas. It comprises six levels, each representing a different view of an XML system and its evolution. The framework is depicted in Figure 1. The lowest level, called *extensional level*, represents XML documents. Its parent level, called *operational level*, represents operations over XML documents, i.e. XML queries. The level above is called *schema level* and represents XML schemas that describe the structure of the XML documents.

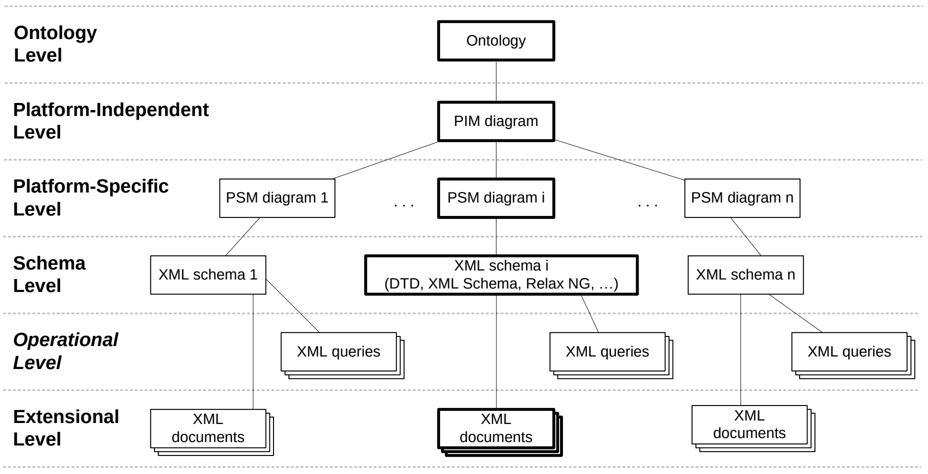


Fig. 1. Six-level XML evolution and integration framework

The platform-independent and platform-specific levels follow MDA [17] which is based on modeling the problem domain on different levels of abstraction. The *platform-independent* level represents the whole problem domain. It consists of a conceptual model that specifies the problem domain independently of its representation in the XML formats below. We call the conceptual model a *platform-independent model* (*PIM*) of the problem domain. The level below, called *platform-specific* level, represents mappings of the problem domain to particular XML formats. For each XML format it comprises a model of mapping of a selected part of the PIM to XML element and attribute declarations. We call this model *platform-specific model* (*PSM*) of the selected XML format. Recently, a number of approaches translating XML schemas to an ontology have appeared. The ontology level is the topmost in our framework.

In our framework, components in documents on individual levels can be formally binded with components in documents on the neighboring levels. These bindings can then be used for evolution of the conceptual model, XML schemas, XML documents and queries. They provide means for automatic detection of all the places on all the levels where a single change has an impact.

4 Comparison criteria

We will firstly introduce several criteria which we will later use to compare current approaches to reverse-engineering of XML schemas. In particular, we will focus on the following criteria:

1. Target model - on what level of our framework does the target model of the reverse-engineering method belong. This criterion is important, because lots of methods only visualize the XML schema in another model (e.g. UML

class diagrams) and therefore their target is on the *platform-specific* level (it is a PSM in our framework). A true conceptual model on the *platform-independent* level (a PIM in our framework) should be independent of the target implementation completely. If it is a conceptual model bent for a specific implementation, it is in fact a PSM. Recently, some approaches to mapping an XML schema directly to an ontology (the top level of our framework) have appeared. This criterion distinguishes among these three types of target models.

2. Number of schemas supported by the model - whether the method is limited to only one schema, or whether it can reverse-engineer multiple schemas to one model. This is also very important, because to be able to manage a set of XML schemas, it is not enough to have a separate model for each schema. We need all the schemas to be related to one model.
3. XML schema languages supported - DTD, XML Schema, Relax NG, Schematron, etc. As can be seen from our framework, the conceptual model can be and should be independent of the actual XML schema language used on the *schema* level, because the target data model (which a PSM should represent) is XML itself, not a specific XML schema language.
4. Mapping to an existing model - whether the method can map a schema to an already existing model or whether it can only generate a new model. This criterion is paramount for evaluating the possibility of integrating an approach to a bigger system for evolution and integration of XML schemas. If it only can create a new model for each input, it cannot be used if we already have the model and only want to add a new XML schema to the system.
5. Level of user involvement - methods can be automatic or semi-automatic (relying on human intervention). It is impossible to infer a conceptual diagram automatically, as so far only a human can determine if two objects represent the same concept. And because we need an exact and reliable match, if we want to use an approach as a part of a system for integration of XML data, we can not rely on an automatic translation (mapping) and we need the user to at least confirm a match.
6. Evolution support - whether the approach is a part of a system that also supports evolving the schema once it is integrated into the system (when the system changes). This criterion indicates, whether the method is developed by itself, or if it already is a part of a system that also helps with the evolution of the mapped schemas (e.g. by preserving the bindings between levels of our framework)

5 Approaches to mapping to user-friendly models

In this section we evaluate different approaches to reverse-engineering of XML schemas to various more user-friendly models.

5.1 Yang Weidong et al.

In [31], there is an algorithm for automatic generation of UML class diagrams (PIMs in our framework) from DTDs according to MDA using a DTD graph as a PSM. The authors also claim that they can generate UML class diagrams from XSDs, but the prototype implementation is not freely available, so it cannot be verified. The main drawback of this approach is that it only serves as an automatic translator from DTD to UML meant to make the schema more understandable to people who do not know DTD or XML Schema.

This approach does not support mapping of multiple XML schemas to the PIM and it does not preserve any mappings between the PIM and the PSM nor between the PSM and the DTD. It is automatic, limited to DTD only and it cannot map a schema to an existing model. The bright side is that it actually uses both PIM and PSM correctly.

5.2 Mikael R. Jensen et al.

In [13], another method of automatic conversion of DTDs to UML class diagrams is presented. Again, it is meant for easier browsing of XML data available on the Internet and to ease the work of a data integrator. In contrast with the previous method, the UML diagrams reflect the structure of the DTD an therefore are only on the PSM level.

This approach does not support mapping of multiple XML schemas and it does not preserve any mappings between the PSM and the DTD. It is automatic, limited to DTD only and it cannot map a schema to an existing model.

5.3 DIXSE framework

In [25], a semi-automatic method of deriving a semantic model from several DTDs is presented. By default, for each element of every DTD a new element is created in the model. This process is done automatically. If the user wants to create a more meaningful model (e.g. wants all *Address* elements to be mapped to one element of the model), a rule written in their DIXml language extending the element in the DTD must be created manually. The model is a PSM as it still preserves the DTD structure. It uses the Telos [18] metamodeling language.

This approach supports semi-automatic mapping of multiple schemas to a conceptual model, but it does not preserve any mappings between the PSM and the DTDs and it is limited to DTD only. It can map a new DTD to an existing model.

5.4 Xyleme

In [24], a project called Xyleme is described. Its focus is to provide a unified view of a large number of heterogeneous XML documents described by DTDs. This enables the user to perform queries on one unified model (called an abstract DTD), to which all the other DTDs describing the XML documents are mapped automatically. The methods for discovery of mappings are mainly language based (thesauri, discovery of synonyms, abbreviations, etc.). A semi-automatic prototype implementation called SAMAG was used to evaluate the approach. In SAMAG, a user needs to validate each syntactic relationship detected.

This approach supports semi-automatic mapping of multiple schemas to a model which is a PSM. It preserves no mapping between the PSM and the DTDs. It is limited to DTD only.

5.5 XTM - XML Tree Model

In [9], a conceptual model for XML called XTM - XML Tree Model is proposed, including an algorithm for reverse-engineering of XML Schema into XTM. It also has a strong theoretical background. However, it is still only a PSM in our framework.

This approach automatically visualizes one schema at a time, is limited to XML Schema and does not maintain any mappings between the PSM and the schemas.

5.6 Nečaský

In [20], a complex approach to the process of reverse engineering of XML schemas to a conceptual model is presented. The model follows MDA as it uses UML class diagrams as a PIM and their extension as PSMs. It is further described in [19]. Because the model has two levels, the process is divided into two parts. The first part is an automatic translation of an XML schema to a PSM. The PSMs are, however, independent of any specific XML schema language; the approach is presented using XML Schema. The second part is a semi-automatic algorithm for the reconstruction of mappings between the PSM and a PIM, but it has some drawbacks. The most problematic one is the computational cost which is up to m^n, where m is a maximum number of outgoing PIM associations from one PIM class and n is the number of PIM classes in the model. Therefore in practice, the algorithm will not work if the PIM diagram contains a bigger number of associations. Nevertheless, the algorithm uses maximum of information that we can get from a PSM and thus can offer the best results. An implementation in an experimental stage is available in the development version of XCase [14], which is a tool implementing the conceptual model and its evolution.

This approach supports semi-automatic reverse-engineering to PSMs and to a PIM. It supports multiple schemas, is independent of any specific XML schema language and it can also map to an existing conceptual model. It maintains mappings between the PIM and the PSM and therefore support further schema evolution.

6 Approaches to mapping to ontologies

Recently, a number of methods of reverse-engineering of XML schemas to ontologies have appeared. The main difference between a PIM and an ontology is that ontologies are more expressive, as the relations they capture can be more complex and they may even involve logical formulae. A frequent language for ontologies is OWL [30].

6.1 Canonic Conceptual Models (CCMs)

In [8], a method for integration of XML schemas into an ontology is presented. At first, each input DTD is semi-automatically transformed into a so called CCM - Canonic Conceptual Model. It combines the ER [5] and ORM [11] models. A default transformation is made and a user is then allowed to make adjustments where needed. The CCMs are PSMs in our framework. Then, each CCM is integrated (again semi-automatically - user has to verify/adjust) to form the final ontology. The mappings between the individual CCM components and the components of the ontology are preserved. Although the authors call it an ontology, it is in fact more of a conceptual diagram - a PIM in our framework, because it still is a CCM, only independent of the XML structure. This method only provides a unified view of the integrated data so far. No implementation was mentioned.

This approach supports semi-automatic mapping of multiple DTDs to corresponding PSMs and to a PIM. It is restricted to DTD only. It preserves mappings between PSMs and a PIM.

6.2 Xiao et al.

In [32], an algorithm is proposed to match given XML document elements to given ontology concepts to achieve an integrated view of multiple XML documents, when matched to the same ontology. It is based on automatic structural matching of a DTD tree to an ontology tree. However, a precondition is that a domain expert has provided a table of synonyms, i.e. a list of semantically matching strings from the DTD and from the ontology (which seems to be a strong precondition).

This in fact semi-automatic approach maps multiple DTDs, it is limited to DTD only and it does not preserve any mappings between the DTDs and the ontology. It can only map to an existing ontology.

6.3 Bedini et al.

In [3], a general architecture of building ontologies from XML schemas is presented. However, no specific methods are suggested, only a sequence of tasks that a tool for ontology building should follow and also a set of rules according to which concepts and their relations can be extracted from a XML Schema.

The general architecture takes the need for consecutive schema evolution into account. A semi-automatic prototype implementation called Janus is presented briefly. It requires human assistance for merging of similar concepts and it does not preserve any mappings between the schemas and the ontology.

This approach is semi-automatic, it is limited to XML schema and it does not preserve any mappings between the schemas and the ontology. It also only creates new ontologies.

6.4 DTD2OWL

In [29], a method of automatic translation of DTD to OWL ontology is suggested. In addition, this method transforms the actual XML documents into OWL individuals. The authors suggest that the whole web should be transformed this way. This method is pure translation of one DTD to one OWL ontology with no support for schema evolution nor conceptual modeling.

This approach is automatic, it is limited to DTD and it can only map one DTD to one new ontology, not preserving any mappings.

7 Summary

In this section, a brief summary of evaluated approaches and the comparison criteria is given.

	Model	**Schemas**	**Languages**	**Automatic**	**Maps to**	**Evolution**
5.1	PSM	One	DTD	Yes	New	No
5.2	PSM	One	DTD	Yes	New	No
5.3	PSM	Multiple	DTD	No	New, Existing	No
5.4	PSM	Multiple	DTD	Yes	New, Existing	No
5.5	PSM	One	XSD	Yes	New	No
5.6	PIM	Multiple	Any[a]	No	New, Existing	Yes
6.1	PIM	Multiple	DTD	No	New, Existing	Yes
6.2	O	Multiple	DTD	No	Existing	No
6.3	O	Multiple	XSD	No	New	No
6.4	O	One	DTD	Yes	New	No

[a] This method is not limited to any XML schema language. Currently implemented for XML Schema

Table 1. Overview of approaches according to various criteria

Let us review the comparison criteria used (the columns in Table 1 correspond to them):

1. Whether the target of the approach is a PIM, PSM or an ontology

2. Whether the approach is limited to only one schema or whether it can handle multiple schemas
3. Which XML schema languages can the approach handle
4. Whether the method is a pure automatic translation or whether the process is semi-automatic - a user is involved
5. Whether the method creates a new target model or whether it can use an existing one
6. Whether the method preserves mappings between the schemas and the target model, which can be used for consecutive schema evolution

The best suitable method for our intention of creating a system for management of XML schema evolution and integration is 5.6. However, it has some issues with computational complexity, which need to be resolved before its implementation.

8 Conclusion

In this paper, we have compared and evaluated several approaches for reverse-engineering of XML schemas according to given comparison criteria. Among them, only one was well suited for being a part of a larger system for evolution and integration of XML schemas. Also, this survey showed a severe lack of support for newer XML schema languages like Relax NG or Schematron in the area of conceptual modeling of XML and reverse-engineering of XML schemas.

References

1. R. Al-Kamha, D. W. Embley, and S. W. Liddle. Augmenting Traditional Conceptual Models to Accommodate XML Structural Constructs. In *Proceedings of 26th International Conference on Conceptual Modeling*, pages 518–533, Auckland, New Zealand, Nov. 2007. Springer.
2. A. Badia. Conceptual Modeling for Semistructured Data. In *Proceedings of the 3rd International Conference on Web Information Systems Engineering Workshops*, pages 170–177, Singapore, Dec. 2002. IEEE Computer Society.
3. I. Bedini, G. Gardarin, and B. Nguyen. Deriving Ontologies from XML Schema. *CoRR*, abs/1001.4901, 2010.
4. R. Bourret. XML and Databases. September 2005. `http://www.rpbourret.com/xml/XMLAndDatabases.htm`.
5. P. Chen. The Entity-Relationship Model–Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36, Mar. 1976.
6. J. Clark and M. Makoto. *RELAX NG Specification.* Oasis, December 2001. `http://www.oasis-open.org/committees/relax-ng/spec-20011203.html`.
7. D. Booth, C. K. Liu. *Web Services Description Language (WSDL) Version 2.0 Part 0: Primer.* W3C, June 2007. `http://www.w3.org/TR/wsdl20-primer/`.
8. R. dos Santos Mello and C. A. Heuser. A Bottom-Up Approach for Integration of XML Sources. In *Workshop on Information Integration on the Web*, pages 118–124, 2001.
9. J. Fong, S. K. Cheung, and H. Shiu. The XML Tree Model - toward an XML conceptual schema reversed from XML Schema Definition. *Data Knowl. Eng.*, 64(3):624–661, 2008.

10. H. Hai, Do. *Schema Matching and Mapping-based Data Integration: Architecture, Approaches and Evaluation.* VDM Verlag, Saarbrücken, Germany, Germany, 2007.
11. T. Halpin and T. Morgan. *Information Modeling and Relational Databases.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.
12. ISO. *Information Technology Document Schema Definition Languages (DSDL) Part 3: Rule-based Validation Schematron. ISO/IEC 19757-3*, feb 2005.
13. M. R. Jensen, T. H. Møller, and T. B. Pedersen. Converting XML Data to UML Diagrams For Conceptual Data Integration. In *In: 1st International Workshop on Data Integration over the Web (DIWeb) at 13th Conference on Advanced Information Systems Engineering (CAISE01)*, 2001.
14. J. Klímek, L. Kopenec, P. Loupal, and J. Malý. XCase - A Tool for Conceptual XML Data Modeling. In *Advances in Databases and Information Systems*, volume 5968/2010 of *Lecture Notes in Computer Science*, pages 96–103. Springer Berlin / Heidelberg, March 2010.
15. B. Loscio, A. Salgado, and L. Galvao. Conceptual Modeling of XML Schemas. In *Proceedings of the Fifth ACM CIKM International Workshop on Web Information and Data Management*, pages 102–105, New Orleans, USA, Nov. 2003.
16. M. Mani. Erex: A conceptual model for xml. In *Proceedings of the Second International XML Database Symposium*, pages 128–142, Toronto, Canada, Aug. 2004.
17. J. Miller and J. Mukerji. *MDA Guide Version 1.0.1.* Object Management Group, 2003. http://www.omg.org/docs/omg/03-06-01.pdf.
18. J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. Telos: representing knowledge about information systems. *ACM Trans. Inf. Syst.*, 8(4):325–362, 1990.
19. M. Nečaský. *Conceptual Modeling for XML*, volume 99 of *Dissertations in Database and Information Systems Series.* IOS Press/AKA Verlag, January 2009.
20. M. Nečaský. Reverse Engineering of XML Schemas to Conceptual Diagrams. In *Proceedings of The Sixth Asia-Pacific Conference on Conceptual Modelling*, pages 117–128, Wellington, New Zealand, January 2009. Australian Computer Society.
21. Object Management Group. *UML Infrastructure Specification 2.1.2*, nov 2007. http://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF/.
22. Object Management Group. *UML Superstructure Specification 2.1.2*, nov 2007. http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF/.
23. G. Psaila. ERX: A Conceptual Model for XML Documents. In *Proceedings of the 2000 ACM Symposium on Applied Computing*, pages 898–903, Como, Italy, Mar. 2000. ACM.
24. C. Reynaud, J.-P. Sirot, and D. Vodislav. Semantic integration of xml heterogeneous data sources. In *IDEAS '01: Proceedings of the International Database Engineering & Applications Symposium*, pages 199–208, Washington, DC, USA, 2001. IEEE Computer Society.
25. P. Rodríguez-Gianolli and J. Mylopoulos. A Semantic Approach to XML-based Data Integration. In *ER '01: Proceedings of the 20th International Conference on Conceptual Modeling*, pages 117–132, London, UK, 2001. Springer-Verlag.
26. P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches. *Journal on Data Semantics*, 4:146–171, 2005.
27. T. Bray and J. Paoli and C. M. Sperberg-McQueen and E. Maler and F. Yergeau. *Extensible Markup Language (XML) 1.0 (Fourth Edition).* W3C, September 2006. `http://www.w3.org/TR/REC-xml/`.
28. H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. *XML Schema Part 1: Structures (Second Edition).* W3C, October 2004. `http://www.w3.org/TR/xmlschema-1/`.

29. P. T. T. Thuy, Y.-K. Lee, and S. Lee. DTD2OWL: Automatic Transforming XML Documents into OWL Ontology. In *ICIS '09: Proceedings of the 2nd International Conference on Interaction Sciences*, pages 125–131, New York, NY, USA, 2009. ACM.
30. W3C OWL Working Group. *OWL 2 Web Ontology Language.* W3C, October 2009. `http://www.w3.org/TR/owl2-overview/`.
31. Y. Weidong, G. Ning, and S. Baile. Reverse Engineering XML. *Computer and Computational Sciences, International Multi-Symposiums on*, 2:447–454, 2006.
32. L. Xiao, L. Zhang, G. Huang, and B. Shi. Automatic Mapping from XML Documents to Ontologies. In *CIT '04: Proceedings of the The Fourth International Conference on Computer and Information Technology*, pages 321–325, Washington, DC, USA, 2004. IEEE Computer Society.

Evolving Quasigroups by Genetic Algorithms*

Václav Snášel[1], Jiří Dvorský[1], Eliška Ochodková[1], Pavel Krömer[1], Jan Platoš[1], and Ajith Abraham[2]

[1] Department of Computer Science, FEECS, VŠB – Technical University of Ostrava
17. listopadu 15, 708 33 Ostrava – Poruba, Czech Republic
{vaclav.snasel,jiri.dvorsky,eliska.ochodkova, pavel.kromer,jan.platos}@vsb.cz

[2] Center of Excellence for Quantifiable Quality of Service, Norwegian University of Science and Technology
O.S. Bragstads plass 2E,
N-7491 Trondheim, Norway
ajith.abraham@ieee.org

Abstract. Quasigroups are a well-known combinatorial design equivalent to more familiar Latin squares. Because all possible elements of a quasigroup occur with equal probability, it makes it an interesting tool for the application in computer security and for production of pseudorandom sequences. Most implementations of quasigroups are based on look-up table of the quasigroup, on system of distinct representatives etc. Such representations are infeasible for large quasigroups. An analytic quasigroup is a recent concept that allows usage of certain quasigroups without the need of look-up table. The concept of isotopy enables consideration of many quasigroups and genetic algorithms allow efficient search for good ones. In this paper we describe analytic quasigroup and genetic algorithms for its optimization.

1 Introduction

Random and pseudorandom sequences can be used in many applications, e.g. in modeling, simulations, and of course in cryptography. Pseudorandom sequences are the core of stream ciphers. The design goal in stream ciphers is to efficiently produce pseudorandom sequences - keystreams (i.e. sequences that possess properties common to truly random sequences and in some sense are "indistinguishable" from these sequences).

The use of quasigroups and quasigroup string transformations is a recent but successful tendency in cryptography and coding [12]. With quasigroups in the hearth of advanced cryptosystems and hash functions, a need to find good quasigroups becomes hot topic.

Quasigroups and its applications in computer security were studied e. g. in [2]. A design of pseudorandom sequence generator (PRSG) based on quasigroup operation was presented in [3]. The authors have performed an extensive analysis of

* This paper was partially supported by GACR 205/09/1079 grant.

2^{16} randomly chosen quasigroups of the orders 5, 6, 7, 8, 9 and 10 and concluded that different quasigroups produce pseudorandom sequences with different period (i.e. the number of elements after which the pseudorandom sequence starts to repeat). They have show that only a small number of quasigroups feature very large value of coefficient of period growth, a property that significantly affects the period of generated pseudorandom sequence [3]. This results encourage research of efficient methods for search for good quasigroups in the field of pseudorandom generators and cryptography.

Genetic algorithms are probably the most popular and wide spread member of the class of evolutionary algorithms (EA). EAs operate with a population of artificial individuals (chromosomes) encoding potential problem solutions. Encoded individuals are evaluated using a carefully selected objective function which assigns a fitness value to each individual. The fitness value represents the quality (relative ranking) of each individual as a solution to given problem. Competing individuals explore in a highly parallel manner problem domain towards an optimal solution [16].

2 Quasigroups

Definition 1. *A quasigroup is a pair $(Q, \circ)$, where $\circ$ is a binary operation on (finite) set Q such that for all not necessarily distinct $a, b \in Q$, the equations*

$$a \circ x = b \text{ and } y \circ a = b.$$

have unique solutions.

The fact that the solutions are unique guarantees that no element occurs twice in any row or column of the table for $(\circ)$. However, in general, the operation $(\circ)$ is neither a commutative nor an associative operation.

Quasigroups are equivalent to more familiar Latin squares. The multiplication table of a quasigroup of order q is a Latin square of order q, and conversely, as it was indicated in [4,5,18], every Latin square of order q is the multiplication table of a quasigroup of order q.

Definition 2. *Let $A = \{a_1, a_2, \ldots, a_n\}$ be a finite alphabet, a $n \times n$ Latin square L is a matrix with entries $l_{ij} \in A$, $i, j = 1, 2, \ldots, n$, such that each row and each column consists of different elements of A.*

For $i, j, k \in A$ the ordered triple $(i, j; k)$ is used to represent the occurrence of element k in cell (i, j) of the Latin square. So a Latin square may be represented by the set $\{(i, j; k)|$ entry k occurs in cell (i, j) of the Latin square $L.\}$

All reduced Latin squares of order n are enumerated for $n \leq 11$ [14]. Let L_n be the number of Latin squares of order n, and let R_n be the number of reduced Latin squares of order n. It can be easily seen that

$$L_n = n!(n-1)!R_n.$$

Number of distinct Latin squares of a given order grows exceedingly quickly with the order and there is no known easily-computable formula for the number of distinct Latin squares. The problem of classification and exact enumeration of Latin squares of order greater than 11 probably still remains unsolved. Thus, there are more than 10^{90} quasigroups of order 16 and if we take an alphabet $L = \{0 \dots 255\}$ (i.e. data are represented by 8 bits) there are at least $256!255! \dots 2! > 10^{58000}$ quasigroups.

Multiplication in quasigroups has an important property; it is proven that each element occurs exactly q times among the products of two elements of Q, q^2 times among the products of three elements of Q and, generally q^{t-1} among the products of t elements of Q. Since there are q^t possible ordered products of t elements of Q, this shows that each element occurs equally often among these q^t products (see [6]).

2.1 Isotopism of quasigroups

Definition 3. *Let $(G, \cdot)$, $(H, \circ)$ be two quasigroups. An ordered triple (π, ρ, ω) of bijections π, ρ, ω of the set G onto set H is called an* isotopism *of $(G, \cdot)$ upon $(H, \circ)$ if $\forall u, v \in G, \pi(u) \circ \rho(v) = \omega(u \cdot v)$. Quasigroups $(G, \cdot)$, $(H, \circ)$ are said to be* isotopic.

We can imagine an isotopism of quasigroups as a permutation of rows and columns of quasigroup's multiplication table.

Example 1. Consider a multiplication table for a quasigroup isotopic to the quasigroup of modular subtraction, with operation $\circ$ defined as $a \circ b = (a + n - b) \bmod n)$:

$\circ$	0	1	2	3
0	0	3	2	1
1	2	1	0	3
2	1	0	3	2
3	3	2	1	0

The table was created from table of modular subtraction. The second and the third rows were exchanged. Permutations π, ρ were identities and $\omega = [0213]$. A multiplication in this quasigroup can be illustrated by e.g. $1 \circ 0 = \omega(1) \circ 0 = 2 \circ 0 = 2$.

Starting with the quasigroup of modular subtraction, we can explore a large class of quasigroups isotopic to the quasigroup of modular subtraction [7,17]. This allows us to utilize quasigroups with very large number of elements without the necessity of their storage in program memory. The multiplication in such isotopic quasigroup is defined as follows:

$$a \circ b = \pi^{-1}((\omega(a) + n - \rho(b)) \bmod \ n). \tag{1}$$

We call the quasigroup defined by its multiplication formula and three selected permutations an *analytic quasigroup* [11,21].

The notion of analytic quasigroup enables efficient work with large quasigroups. Previos studies in this field used mostly quasigroups of small order [10], or just a small parts of certain quasigroup were utilized e.g. as a key for Message Authentication Code. Such small quasigroups are represented as look-up tables in main memory. Larger quasigroup of order 2^{256} is used by NIST's SHA-3 competition candidate, hash function Edon$\mathcal{R}$ [9].

The properties of one analytic quasigroup isotopic to the quasigroup of modular subtraction were studied in [11]. The quasigroup was created using three static functions that divided the sequence of n elements of the quasigroup into several parts. The parts were rotated in various directions and exchanged among themselves. It was shown that the investigated quasigroup has some faults in its properties.

2.2 Constructing quasigroups isotopic to the quasigroup of modular subtraction

Consider a quasigroup on the length n defined by multiplication $a \circ b = (a + n - b) \bmod n)$. Then three permutations π, ρ, ω must be chosen in order to implement isotopic quasigroup, whose multiplication will be defined by (1).

Obviously, there is $n!$ different permutations of n elements. Because three independent permutations are used to define any isotopic quasigroup, there are $n!n!n!$ possible choices of π, ρ and ω. Permutations of elements cannot be sought for an analytic quasigroup directly, because its elements are not stored in memory. Instead, the permutation needs to be implemented as a function of an element of Q. One way to achieve this goal is the use of bit permutation.

A quasigroup over a set of n elements requires $log_2(n)$ bits to express each element. Each permutation of bits in the element representation represents also a permutation of all elements of the quasigroup (if n is a power of 2). Bit permutation can be implemented easily as a function of $q \in Q$.

The bit permutation is an elegant way of implementing permutations over n elements of Q. Although it enables us to explore only a fragment $(log_2(n)!log_2(n)!\,log_2(n)!)$ of all possible permutation triples over the quasigroup of n elements, it is useful because it does not require all n elements in main memory and therefore fits into the framework of analytic quasigroups.

Bit permutations are computationaly more expensive than the static functions used to implement permutation in [11]. However, there are ongoing efforts to implement bit permutation instructions in hardware, which would improve the performance of the proposed algorithm significantly [8].

3 Genetic algorithms

Genetic algorithms (GAs) are generic and reusable population-based metaheuristic soft optimization method [16]. GAs operate with a population of chromosomes encoding potential problem solutions. Encoded individuals are evaluated using a carefully selected domain specific objective function which assigns a fitness value to each individual. The fitness value represents the quality of each candidate solution in context of the given problem. Competing individuals explore the problem domain towards an optimal solution [16].

The emulated evolution is driven by iterative application of genetic operators. Genetic operators algoritmize principles observed in natural evolution. The crossover operator defines a strategy for the exchange of genetic information between parents (sexual reproduction of haploid organisms) while the mutation operator introduces the effect of environment and randomness (random perturbation of genetic information). The basic workflow of the standard generational GA is shown in algorithm 1.

Algorithm 1: A summary of genetic algorithm

```
1 Define objective (fitness) function and problem encoding
2 Encode initial population $P$ of possible solutions as fixed length strings
3 Evaluate chromosomes in initial population using objective function
4 while Termination criteria not satisfied do
5     Apply selection operator to select parent chromosomes for
      reproduction: $sel(P_i) \rightarrow parent1$, $sel(P_i) \rightarrow parent2$
6     Apply crossover operator on parents with respect to crossover
      probability to produce new chromosomes:
      $cross(pC, parent1, parent2) \rightarrow \{offspring1, offspring2\}$
7     Apply mutation operator on offspring chromosomes with respect to
      mutation probability: $mut(pM, offspring1) \rightarrow offspring1$,
      $mut(pM, offspring2) \rightarrow offspring2$
8     Create new population from current population and offspring
      chromosomes: $migrate(offspring1, offsprig2, P_i) \rightarrow P_{i+1}$
9 end
```

Many variants of the standard generational GAs have been proposed. The differences are mostly in particular selection, crossover, mutation and replacement strategy [16].

In the next section, we present genetic algorithm for the search for good analytic quasigroups. It is an extended version of the initial GA for quasigroup evolution introduced in [21]. In this study, we introduce a new fitness function based on associativity and commutativity that is used for quasigroup optimization.

4 Genetic search for analytic quasigroups

The genetic algorithm for the search for analytic quasigroup is defined by encoding of the candidate solutions and fitness function to evaluate chromosomes.

4.1 Encoding

As noted in subsection 2.2, any analytic quasigroup isotopic to quasigroup of modular subtraction is defined by three permutations. Such permutation triple represents a problem solution and should be mapped to one GA chromosome. Permutations can be for the purpose of genetic algorithms encoded using several strategies. In this study, we use random key encoding.

Random key (RK) encoding is an encoding strategy suitable for problems involving permutation optimization [19]. In random key encoding, the permutation is represented as a string of real numbers (random keys), whose relative position changes after sorting corresponds to the permutation index. An example or random key encoding is shown in (2).

$$\Pi_5 = \begin{pmatrix} 0.2 & 0.3 & 0.1 & 0.5 & 0.4 \\ 2 & 3 & 1 & 5 & 4 \end{pmatrix} \tag{2}$$

To encode a quasigroup (isotopic to the quasigroup of modular subtraction) of the length $n = 2^l$, we use a vector of $3l$ real numbers $v = (v_1, \ldots, v_{l-1}, v_l, \ldots v_{2l-1}, v_{2l}, \ldots, v_{3l})$. The vector is interpreted as three concatenated RK encoded permutations of the length l.

This encoding allows us to use traditional implementations of genetic operators, such as n-point crossover and mutation. Crossover was implemented as mutual exchange of genes between selected parents and mutation was implemented as a replacement of gene with a uniform random number from the interval $[0, 1]$.

4.2 Fitness function

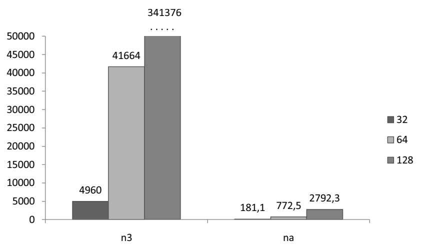
The fitness function f we propose in this work is based on commutativity and associativity in quasigroup.

$$f(n, n_a, n_c, \alpha) = \alpha \frac{n_2 - n_c}{n_2} + (1 - \alpha) \frac{n_3 - n_a}{n_3} \tag{3}$$

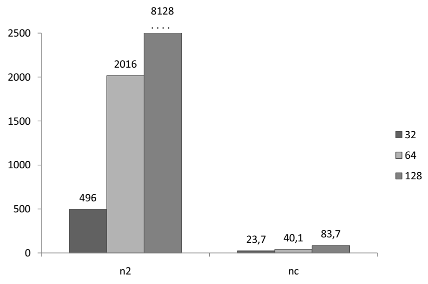
In (3), n stands for the order of the quasigroup, n_2 stands for the number of all combinations of 2 elements out of n, n_3 represents all combinations of 3 elements out of n, n_c stands for the number of element pairs that have the commutativity property and n_a represents the number of element triples that have the associativity property. The coefficient $\alpha \in [0, 1]$ is used to prioritize between commutativity and associativity. The value of fitness function is 0 for $n_c = n_2$ and $n_a = n_3$ at the same time and 1 for $n_c = 0$ and $n_a = 0$. Informally, we can say that it seeks for a quasigroup that has “low associativity“ and “low commutativity“.

5 Experimental optimization

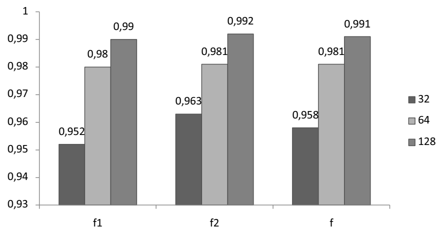
We have investigated the associativity and commutativity in randomly generated quasigroups isotopic to the quasigroup of modular subtraction of orders 32, 64 and 128 respectively. The average values of n_a, n_c and fitness in random quasigroups are shown in Figure 1.



(a) Average n_a in random quasigroups ($n3$ illustrates the number of all combinations of 3 elements out of n).



(b) Average n_c in random quasigroups($n2$ illustrates the number of all combinations of 2 elements out of n).



(c) Average fitness in random quasigroups. $f1$ corresponds to $\frac{n_2 - n_c}{n_2}$, $f2$ to $\frac{n_3 - n_a}{n_3}$ and $f = 0.5f1 + 0.5f2$

Fig. 1. Average values of n_a, n_c and fitness in random quasigroups of order 32, 64 and 128.

In the next step, we have performed genetic search for better (in terms of low associativity and low commutativity) quasigroups isotopic to the quasigroups of modular subtraction with the dimensions 32, 64 and 128 respectively. We have implemented genetic algorithm with permutation encoding and fitness function as defined above. The parameters of the algorithm (α, P_M, P_C etc.) were selected after initial tuning of the algorithm. The paramteres are summarized in Table 1.

Beacuse genetic algorithm is a stochastic method, every experiment was repeated 10 times and presented results are average values after 10 independent runs. The values of optimized fitness, n_c and n_a are shown in Table 2. A comparison of optimized values with n_a, n_c and fitness in random quasigroups isotopic

Table 1. The settings of genetic algorithm for quasigroup search

Parameter	value
Fitness function coefficient α	0.5
Population size	20
Probability of mutation P_M	0.02
Probability of recombination P_C	0.8
Selection operator	elitist
Max number of generations	1000

to the quasigroups of modular subtraction are illustrated in Figure 2. We can see that in every experiment (for every quasigroup dimension), the genetic optimization process delivered a quasigroup better than random one.

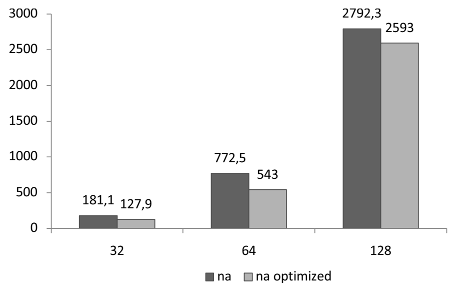
Table 2. Results for average evolved quasigroup of the dimension 32, 64 and 128 respectively.

	Property		
Dimension	n_c	n_a	fitness
32	4.3	127.9	0.983
64	5	543	0.992
128	24	2593	0.995

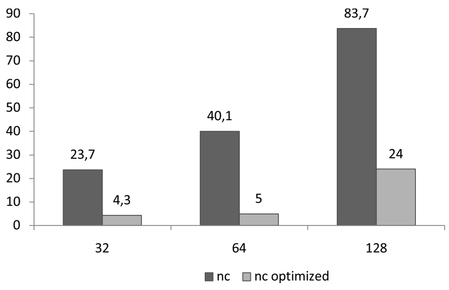
6 Conclusions

In this paper was described a genetic algorithm for optimization of an analytic quasigroup. The genetic algorithm performs a search for good bit permutations that are then used to construct analytic quasigroups with desired properties. Both, the analytic quasigroup and bit permutation, do not rely on the lookup table of the quasigroup stored in memory.

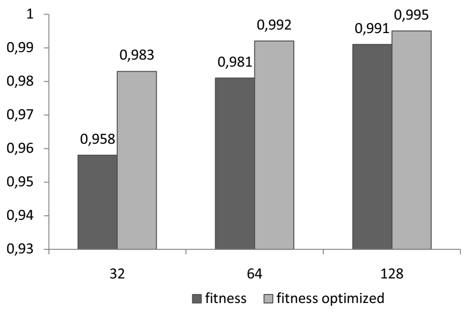
The fitness function we use in this paper is based on associativity and commutativity in quasigroups. It triggers a search for quasigroups that have “low associativity“ and “low commutativity“. In a numerical experiment, we have been able to find quasigroups with better properties than random ones have. The drawback of this method is at this point its computational expensiveness. In order to evaluate the fitness function, all combinations of 2 elements out of n and all combinations of 3 elements out of n (quasigroup dimension) have to be found and evaluated using quasigroup operation $\circ$. However, the main aim of this work was to verify that genetic algorithm can evolve quasigroups with above average properties.



(a) Average n_a in optimized quasigroups. Lower is better.



(b) Average n_c in optimized quasigroups. Lower is better.



(c) Average fitness in optimized quasigroups. Higher is better.

Fig. 2. Average values of n_a, n_c and fitness in optimized quasigroups of order 32, 64 and 128 compared with n_a, n_c and fitness in random quasigroups.

In our future work, we want to investigate the properties of generated quasigroups, study the fitness function and look for alternative fitness functions. We want to focus on efficient implementation of used genetic algorithms with the utilization of GPGPU. Moreover, we will investigate other successful computational intelligence methods for quasigroup optimization (e.g. differential evolution, ant colony optimization).

References

1. G. Marsaglia and W. W. Tsang, "Some Difficult-to-pass Tests of Randomness", Journal of Statistical Software, volume 7,number i03.
2. S. Markovski, "Quasigroup String Processing and Applications in Cryptography", Proceedings 1st Conference of Mathematics and Informatics for Industry, Thessaloniki, Greece, pp. 278–290, 2003.
3. V. Dimitrova, J. Markovski, "On quasigroup pseudo random sequence generator", Proc. of the 1-st Balkan Conference in Informatics, Y.Manolopoulos and P. Spirakis eds., Thessaloniki, pp. 393–401, Nov 2004.
4. Belousov, V. D. Osnovi teorii kvazigrup i lup (in Russian), Nauka, Moscow, 1967.

5. Dénes, J., Keedwell, A. Latin Squares and their Applications. Akadémiai Kiadó, Budapest; Academic Press, New York (1974)
6. Dénes, J., Keedwell, A. A new authentication scheme based on Latin squares. Discrete Mathematics (106/107) (1992) pp. 157–161
7. J. Dvorský, E. Ochodková, V. Snášel, Hash Functions Based on Large Quasigroups, Proceedings of Velikonoční kryptologie, Brno, 2002, pp. 1–8.
8. Y. Hilewitz, Z. J. Shi, Lee, and R. B., "Comparing fast implementations of bit permutation instructions," in *Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers*, (Pacific Grove, California, USA), pp. 1856–1863, Nov. 2004 2004.
9. D. Gligoroski, et al. EdonR cryptographic hash function. Submition to NIST's SHA-3 hash function competition, 2008,
http://csrc.nist.gov/groups/ST/hash/sha-3/index.html
10. D. Gligoroski, S. Markovski, L. Kocarev and J. Svein. The Stream Cipher Edon80. The eSTREAM Finalists, *Lecture Notes in Computer Science*, Vol. 4986. pp. 152-169 , 2008
11. V. Snášel, A. Abraham, J. Dvorský, P. Krömer, and J. Platoš, "Hash functions based on large quasigroups.," in *ICCS (1)* (G. Allen, J. Nabrzyski, E. Seidel, G. D. van Albada, J. Dongarra, and P. M. A. Sloot, eds.), vol. 5544 of *Lecture Notes in Computer Science*, pp. 521–529, Springer, 2009.
12. S. J. Knapskog, "New cryptographic primitives," in *CISIM '08: Proceedings of the 2008 7th Computer Information Systems and Industrial Management Applications*, (Washington, DC, USA), pp. 3–7, IEEE Computer Society, 2008.
13. J. Koza, "*Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems*," Technical Report STAN-CS-90-1314, Dept. of Computer Science, Stanford University, 1990.
14. B. D. McKay and I. M. Wanless. On the Number of Latin Squares. *Journal Annals of Combinatorics*, Issue Vol.ume 9 (2005), No. 3, pp. 335-344.
15. R .C. Merkle, *Secrecy, authentication, and public key systems.* Stanford Ph.D. thesis 1979, pages 13-15.
http://www.merkle.com/papers/Thesis1979.pdf
16. M. Mitchell, *An Introduction to Genetic Algorithms.* Cambridge, MA: MIT Press, 1996.
17. E. Ochodková, V. Snášel, Using Quasigroups for Secure Encoding of File System, Proceedings of the International Scientific NATO PfP/PWP Conference "Security and Information Protection 2001", May 9-11, 2001, Brno, Czech Republic, pp.175–181.
18. J. D. H. Smith, An introduction to quasigroups and their representations, Chapman & Hall/CRC, 2007.
19. L. V. Snyder and M. S. Daskin, "A random-key genetic algorithm for the generalized traveling salesman problem," *European Journal of Operational Research*, vol. 174, no. 1, pp. 38–53, 2006.
20. M. Vojvoda. Cryptanalysis of One Hash Function Based on quasigroup. In *Conference Mikulášská kryptobesídka*, pp. 23-28., Praha, 2003.
21. V. Snášel, A. Abraham, J. Dvorský, E. Ochodková, J. Platoš, P. Krömer, Searching for Quasigroups for Hash Functions with Genetic Algorithms, Proceedings of the 2009 World Congress on Nature & Biologically Inspired Computing, pp. 367 - 372 IEEE Computer Society, 2009.

Using Spectral Clustering for Finding Students' Patterns of Behavior in Social Networks

Gamila Obadi, Pavla Dráždilová, Jan Martinovič, Kateřina Slaninová, and Václav Snášel

VŠB - Technical University of Ostrava, FEECS, Department of Computer Science
17. listopadu 15, 708 33 Ostrava-Poruba, Czech Republic
jan.martinovic@vsb.cz, slaninova@opf.slu.cz

Abstract. The high dimensionality of the data generated by social networks has been a big challenge for researchers. In order to solve the problems associated with this phenomenon, a number of methods and techniques were developed. Spectral clustering is a data mining method used in many applications; in this paper we used this method to find students' behavioral patterns performed in an elearning system. In addition, a software was introduced to allow the user (tutor or researcher) to define the data dimensions and input values to obtain appropriate graphs with behavioral pattens that meet his/her needs. Behavioral patterns were compared with students' study performance and evaluation with relation to their possible usage in collaborative learning.

1 Introduction

Social networks have been attracting millions of users, where each user is represented by a huge number of variables. This popularity leads to very high dimensional data, often with sparse data sets. Researchers, dealing with such high-dimensional data collections, face many challenges due to the difficulties with visualization of these data sets and the vast increase in their computing time or memory complexity. Dimension reduction is the process of reducing the number of variables describing the data set. This process is possible, because many of the variables are correlated with each other and many of them have a variation smaller than the measurement noise and thus will be irrelevant. Researchers approach these high dimensional data sets either by finding a subset of the original variables or by mapping the multidimensional space into a space of fewer dimensions. In this work the authors are interested in finding the patterns of behavior of elearning students and the relationship between them and students academic performance. To solve the problem of higher dimensionality, spectral clustering was used. Besides, a software was developed to allow the user (researcher or elearning tutor) to define the data dimensions and input values for graph generation that meet his/her needs.

2 Finding Patterns in Social Networks

Data mining is the process of extracting patterns from data. Patterns can be defined as structures that make statements only about restricted regions of the space spanned by the variables of the data [17]. These structures might represent Web pages that are all about the same topic or people that socialize together. The discovery of pattern structures in complex networks is an essential and challenging task in many disciplines, including business, social science, engineering, and biology. Therefore researchers have a great interest in this subject and have been approached it differently through data mining methods, social network analysis, etc. This diversity is not limited to the techniques used to implement this task, but it is also applied to its applications. The authors of [18] provided an overview about the usage of frequent pattern mining techniques for discovering different types of patterns in a Web logs. While in [22] the authors applied different clustering algorithm to detect crimes patterns, and some data mining tools were used in [1] to solve the intrusion detection problem.

Social Network Analysis (SNA) is another common method for patterns discovery and has been used in previous studies. The authors in [21] used some SNA metrics to study the interaction patterns of students in Elearning online communities. Discovering patterns from email datasets is one of the topics researches are interested in as well. In [8] the authors described a multi-stage spam filter based on trust and reputation for detecting the spam behavior of the call in Voice over Internet Protocol (VoIP). Pattern discovery is also used in the investigation and control of an infectious disease epidemic. A study was conducted to explore the movements of cattle and sheep during the initial phase of the 2001 Foot and mouth disease (FMD) outbreak in UK to describe and visualize the network of these movements using social network analysis techniques [25].

Educational data mining (EDM) is a discipline witch concentrates on developing methods for exploring the data that come from an educational context, and use these methods to improve the quality of learning. To achieve this objective a number of studies have been conducted where researchers applied different data mining techniques to discover the factors that affect the academic performance of students. The authors in [4] studied the navigational behavior of the students to identify the patterns of high performance, in [10] were used decision trees to Predict Students Drop Out and in [3] authors investigated the Cheating behavior in Online Student Assessment. Romero and Ventura classified educational data mining methods into two categories - Statistics and visualization, and Web mining (Clustering, classification, outliers detection, Association rule mining and sequential pattern mining, Text mining) [28].

Clustering is a useful method to investigate students' patterns of behavior, a number of different clustering algorithms have been used to detect online students' patterns such as the TwoStep algorithm [4], the model-based clustering [16], the K-means algorithm [26, 27], etc. In this study, the authors aim to apply spectral clustering to investigate the correlation between the similarity in students behavior and their grades. Even though spectral clustering has been applied to solve many problems in signal processing, bioinformatics and information retrieval, in the field of educational data mining this method has not exploited widely yet.

3 Spectral Clustering

Clustering is the process of organizing objects into collections whose members are similar in some sense. A cluster is therefore a collection of objects which are similar to each other and are dissimilar to the objects belonging to other clusters. The task of finding clusters has been the focus of research in machine learning, information retrieval, social network analysis, etc.

Spectral clustering algorithms cluster a set of data points using the similarity matrix that is derived from the data. It uses the second eigenvector of a graph's Laplacian to define a semi-optimal cut of a weighted undirected graph in which nodes correspond to objects and edges represent the distance (similarity) between the objects. The idea of finding partitions of graphs by using the eigenvectors of their Laplacians can be traced back to 1970s to Fiedler [14], Donath [12]. Fiedler associated the second-smallest eigenvalue of the Laplacian of a graph with its connectivity and suggested partitioning by splitting vertices according to their value in the corresponding eigenvector. Thus, this eigenvalue is called Fiedler value and the corresponding vector is called the Fiedler vector. According to Fiedler, the graphs's Laplacian has the following spectral properties:

- All eigenvalues are non-negative.
- If the graph is divided into g components, there are g zero eigenvalues.
- Eigenvector components act like coordinates to represent nodes in space.
- The Fiedler vector has both positive and negative components, their sum must be zero.
- If the network is connected, but there are two groups of nodes weakly linked to each other, they can be identified from Fiedler vector. Where the positive components are assigned to one group and the negative components are assigned to the other.

Spectral clustering has been studied and applied to solve many problems. In [19] Kannan et al. developed a natural bicriteria measure for assessing the quality of clustering. Cheng et al. in [7] showed how to use spectral algorithm studied in [19]. A practical implementation of the clustering algorithm is presented in [6]. In [11] Ding et al. proposed a new graph partition method based on the min-max clustering principle: the similarity between two subgraphs (cut set) is minimized, while the similarity within each subgraph (summation of similarity between all pairs of nodes within a subgraph) is maximized. Shi and Malik [30] treated image segmentation as a graph partitioning problem and proposed a global criterion, the normalized cut, for segmenting this graph. They showed that an efficient computational technique based on a generalized eigenvalue problem can be used to optimize this criterion. A recursive algorithm was used in [9], Dasgupta et al. analyzed the second eigenvector technique of spectral partitioning on the planted partition random graph model, by constructing a recursive algorithm. Spectral clustering was used in [5] for extracting communities from the Enron graph.

3.1 Algorithm for Graph Partitioning Using Fiedler Vector

1. Find all connected components in graph.
2. Create Laplacian matrix of component $L = D - P'$. $P' = P - I$, where P is the adjacency matrix with weights, I is unity matrix and D is the diagonal matrix with $d_{ii} = \sum_j p'_{ij}$.
3. Find the eigenvector corresponding to second smallest eigenvalue of L.
4. Divide the component based on the sorted eigenvector.
5. Recurse on the obtained components (back to the step 2).

4 Social Network Analysis

Social network analysis concentrates on the importance of the relationships between the nodes. It maps and measures formal and informal relationships to understand what facilitates or obstructs the knowledge flows that connect the interacting objects, e.g., who knows whom, and who shares what information and knowledge with whom and by which communication media.

The results of social network analysis might be used to:

- Identify the individuals or groups who play central roles.
- Distinguish bottlenecks (central nodes that provide the only connection between different parts of a network), as well as isolated individuals and groups.
- Strengthen the efficiency and effectiveness of existing, formal communication channels.
- Improve innovation and learning.
- Refine strategies.

Centrality is an important concept in social network analysis. Borgatti and Everett [2] developed a unified framework for the measurement of centrality. All measures of centrality assess a node's involvement in the walk structure of a network. Measures vary along four key dimensions: type of nodal involvement assessed, type of walk considered, property of walk assessed, and choice of summary measure.

Degree centrality can be used for identifying central roles of the object. Actors who have more ties to other actors may be in advantageous positions. Degree centrality is measured as the number of edges that involve a given node [15]. A node with high degree centrality maintains contacts with numerous other network nodes. Such nodes can be seen as popular nodes with large numbers of links to others. A central node occupies a structural position that may act as a way for information exchange. In contrast, peripheral nodes maintain few or no relations and thus are located at the margins of the network. Degree centrality for a given node p_i is calculated as:

$$C^D(p_i) = \sum_{k=1}^{N} a(p_i, p_k) \tag{1}$$

where N is the number of nodes in the network, $a(p_i, p_k) = 1$ if a edge exists between p_i and p_k and $i \neq k$ else $a(p_i, p_k) = 0$.

Katz [20] recognized that an individual's centrality depends not only on how many others it is connected to (it's degree), but also on their centrality. He measured centrality of a node by the total number of paths linking it to other nodes in a network, exponentially weighted by the length of the path.

Closeness centrality measures the reciprocal of the mean geodesic distance $d(p_i, p_k)$, which is the shortest path between a node p_i and all other reachable nodes [15]. Closeness centrality can be regarded as a measure of how long it will take information to spread from a given node to other nodes in the network [23]. Closeness centrality for a given node is calculated as:

$$C^C(p_i) = \frac{N-1}{\sum_{k=1}^{N} d(p_i, p_k)} \tag{2}$$

where N is the number of nodes in the network and $i \neq k$.

In [24] authors combined existing methods on calculating exact values and approximate values of closeness centrality and presented new algorithms to rank the top-k vertices with the highest closeness centrality.

Betweenness centrality measures the extent to which a node lies on the paths linking other nodes [15]. Betweenness centrality can be regarded as a measure of the extent to which a node has control over information flowing between others [23]. A node with a high betweenness centrality has a capacity to facilitate interactions between the nodes that it links. It can be regarded as how well a node can facilitate communication to other nodes in the network. Betweenness centrality is calculated as:

$$C^B(p_i) = \sum_{j=1}^{N} \sum_{k}^{j-1} \frac{g_{jk}(p_i)}{g_{jk}} \tag{3}$$

where g_{jk} is the total number of geodesic paths linking p_j and p_k, and $g_{jk}(p_i)$ is the number of those geodesic paths that include p_i. Freeman's centrality metrics are based on analysis of a complete and bounded network which is sometimes referred to as a sociocentric network. These metrics become difficult to evaluate in networks with a large node population, because they require complete knowledge of the network topology.

5 Case Study: Finding Students' Patterns of Behavior in LMS Moodle

5.1 Data Set

A case study is conducted in order to find and visualize the patterns of behavior in a large and sparse social network. The analyzed data collections are stored in the Learning Management System (LMS) Moodle logs used to support eLearning education at Silesian University, Czech Republic.

The logs consist of records of all events performed by Moodle's users such as communication in forums and chats, reading study materials or blogs, taking tests or quizzes etc. The users of this system (students, tutors, and administrators) are members of a community which aims to provide the appropriate services and guidance to its members, to make them achieve their objectives successfully. The authors are interested in studying students' activities in the Moodle system and in discovering the latent social network created from groups of students with similar patterns of behavior.

Data anonymisation was implemented during the data preprocessing phase, and the study was only limited to investigating the events performed by students. Let us define a set of students $s \in S$, set of courses $c \in C$ and term *Event* as a combination of Event prefix $p \in P$ (e.g. course view, resource view, blog view, quiz attempt) and a course c. An event then represents an action performed by student $s \in S$ in certain course c in LMS. On the basis of this definition, we have obtained *Set of Events* $e_i \in E$, which is represent by pairs $e_i = (p_j, c_k), j \in \{1, \ldots |P|\}, k \in \{1, \ldots |C|\}$ ordered by TimeStamp.

After that we obtain set of activities $a_j \in A$. *Activity* is a sequence of events $a_j = \langle\, e_1,\, e_2,\, \ldots,\, e_n\,\rangle$, performed by certain student s in a certain course c during the optimal time period. In our previous experiments we found the 30 minutes time period to be the most effective time interval. The findings showed that in shorter time periods (5 minutes) students were performing only non-study activities, and in longer periods there was not a significant activity difference (that means activities were very similar). For detailed information see our previous work [13]. Similar conclusion was presented by Zorrilla et al. in [31].

Two matrices were obtained to represent the data: the Student matrix T ($|S| \times |A|$), where row $(t_1, t_2, \ldots, t_{|A|})$ represents a subset of activities performed by the student in the Moodle system, and the Matrix of similarity P ($|S| \times |S|$), which is derived from matrix T, and defines students' relationships using their similar activities. The similarity between two students (vectors) was defined by the Cosine measure [29].

$$p_{i,j} = \frac{\sum_{k=1}^{n} t_{ik} t_{jk}}{\sqrt{\sum_{k=1}^{n} t_{ik}^2}\sqrt{\sum_{k=1}^{n} t_{jk}^2}} \tag{4}$$

Matrices T and P are very large and sparse because of the large number of activities performed by students. Therefore the visualization of the latent ties between students with similar behavior was very hard and unfeasible. One of our goals was the reduction of that high number of activities using specification of smaller groups with characteristic activities.

Spectral clustering was used to divide the collection of students into a number of smaller groups. The spectral clustering method was then extended by further analysis of the obtained clusters. For each cluster a level was set to obtain smaller sets of objects in cluster. The most frequent activities in each cluster can describe the cluster with more details. We are interested in typical activities for each cluster. Activities, which are in all clusters, and activities which are less frequent can be omitted. Merging the reduced activity sets in each cluster defines set of typical activities for all objects in the selected group.

5.2 Experiment

The main objective of this experiment is to investigate the relationship between the similarity in students' behavior and their grades. For this purpose has been developed specialized software, which allows the user (researcher or tutor) to define the values of input data that meets his/her needs. Input data definition is based on data collection filtering and setting of parameter time period. The user can select groups of students by Course, Event Prefix (which represents a set of Events e), Step (time period) and level of similarity. To test the behavior of students accessing the resources which are assigned for the different courses, a course called MicroEconomy with 307 students and a level of similarity $= 0.5$ was analyzed. The following figures illustrate the results of the experiment. Figure 1 represents the graph before clustering, and Figure 2 illustrates the different clusters obtained (colored nodes represent different grades; yellow is A, green yellow is B, green is C, blue is D, red is E, black is F and gray is without the grade).

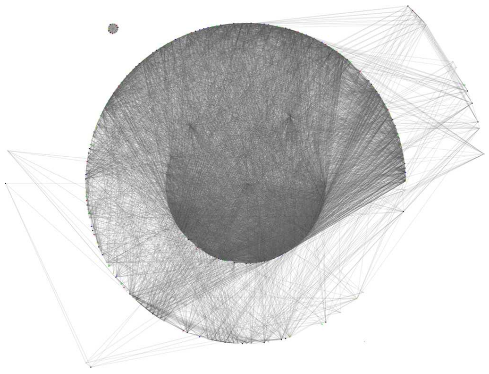


Fig. 1. Graph of Students' Activities when Accessing the Course Resources

Obviously, the graphs show that these clusters consist of students with different distribution of grades.

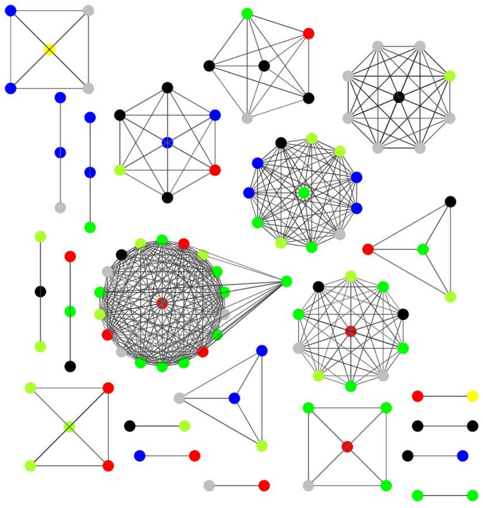


Fig. 2. Graph of Student's Clusters when Accessing the Course Resources

The same procedure was applied to the same course to explore students' behavioral patterns in the forum and again a number of clusters were detected with different grades in each one of them, see (Fig 3 and Fig 4).

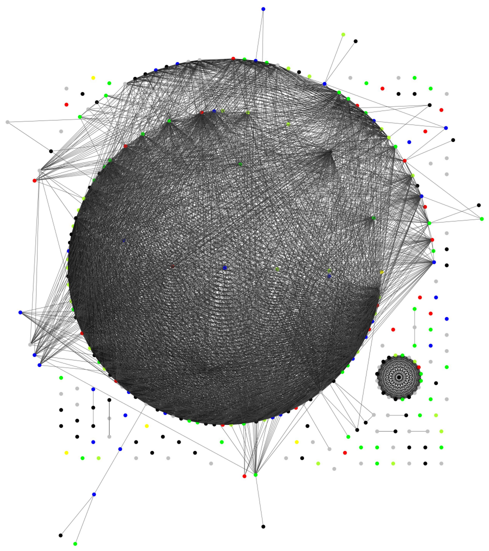


Fig. 3. Graph of Students' Activities in the Forum

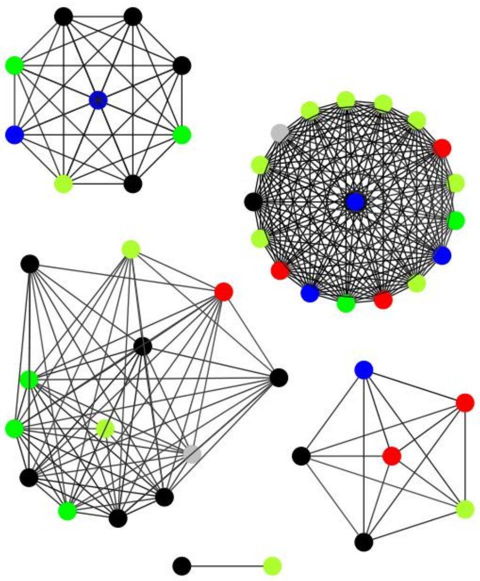


Fig. 4. Graph of Students' Clusters in the Forum

The results of this experiment showed that the similarity in students behavior does not have significant effect on their final grades. It is caused by several limitations, more detailed description is presented in conclusion.

Another test was applied to study the correlation between students' position in the network (using different types of centrality) and their academic performance. Findings from this experiment can be helpful for course tutors. The tutor can recommend to students suitable study behavior, or contact to the student with appropriate study behavior.

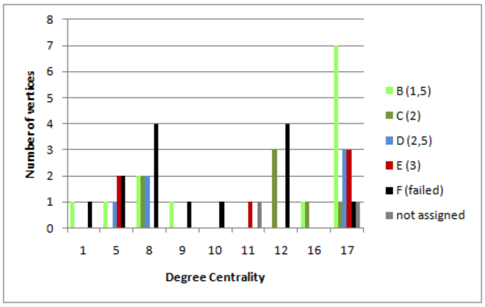


Fig. 5. Histogram of the Degree Centrality for One Course and Clustering Level 40

The findings of the study showed that both students with high academic performance and students with educational difficulties were isolated, while students with average academic performance were highly connected, see (Fig 5). This group of students participates actively in the course forum, but maybe they discuss the issues that are less important to their academic progress. This is one of the limitations of the data extracted from Moodle log files. The log files record only request transactions but they do not record the type of content on each page (especially topics discussed in forum).

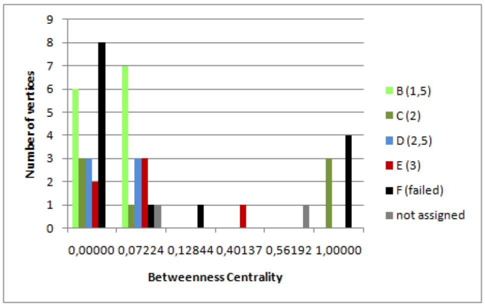


Fig. 6. Histogram of the Betweenness Centrality for One Course and Clustering Level 40

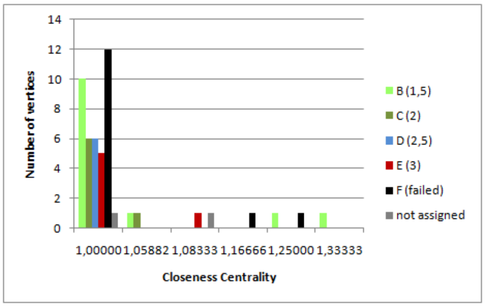


Fig. 7. Histogram of the Closeness Centrality for One Course and Clustering Level 40

6 Conclusion

In this study was presented an application of spectral clustering method to find the patterns of behavior of groups of students enrolled in the elearning system. For easier generation of the graphs described students' behavioral patterns in elearning system, with requires setting of number of input variables for clustering method or setting of the dimensions (selection of the appropriate course or students' activities in the course provided in elearning system), was developed specialized software. Moreover, authors attempted to find relations between the behavioral study patterns and the students' study performance in the selected course. The findings of the experiment did not show any relation between the similarity in students' behavior and their grades as well as relation between students' positions in generated network and their academic performance. We found significant clusters of similar behavior, but with different distribution of grades. Students with average values of grades were at the center of the network. During the implementation of this study we encountered a number of limitations. The first limitation was the small size and homogeneity of the data set, the second was that all events were given the same importance - the experiment showed that clustering could be better if we assigned different weights to the events according to their significance to the course. Nevertheless, developed software can be successfully used as a supporting tool for tutors leading groups of students in elearning systems, to discover significant behavioral patterns of their students. These information can be useful especially in large groups of students were are applied methods of collaborative learning. In our future work, we intend to apply a thorough analysis on larger data collections to explore significant patterns of behavior, and to find other factors that might affect students' grades.

7 Acknowledgement

We acknowledge the support of project SP/2010196 Machine Intelligence.

References

1. D. Barbar, J. Couto, S. Jajodia, L. Popyack, and N. Wu. Adam: Detecting intrusions by data mining. In *In Proceedings of the IEEE Workshop on Information Assurance and Security*, pages 11–16, 2001.
2. S. Borgatti and M. Everett. A graph-theoretic perspective on centrality. *Social Networks*, 28(4):466–484, October 2006.
3. G. N. Burlak, J.-A. Hernandez, A. Ochoa, and J. Munoz. The use of data mining to determine cheating in online student assessment. In *CERMA '06: Proceedings of the Electronics, Robotics and Automotive Mechanics Conference*, pages 161–166, Washington, DC, USA, 2006. IEEE Computer Society.
4. J. M. Carbo, E. Mor, and J. Minguillon. User navigational behavior in e- learning virtual environments. In *WI '05: Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 243–249, Washington, DC, USA, 2005. IEEE Computer Society.
5. A. Chapanond, M. S. Krishnamoorthy, and B. Yener. Graph theoretic and spectral analysis of enron email data. *Comput. Math. Organ. Theory*, 11(3):265–281, 2005.
6. D. Cheng, R. Kannan, S. Vempala, and G. Wang. On a recursive spectral algorithm for clustering from pairwise similarities. Technical report, 2003.
7. D. Cheng, R. Kannan, S. Vempala, and G. Wang. A divide-and-merge methodology for clustering. *ACM Trans. Database Syst.*, 31(4):1499–1525, December 2006.

8. R. Dantu and P. Kolan. Detecting spam in voip networks. In *SRUTI'05: Proceedings of the Steps to Reducing Unwanted Traffic on the Internet on Steps to Reducing Unwanted Traffic on the Internet Workshop*, pages 5–5, Berkeley, CA, USA, 2005. USENIX Association.
9. A. Dasgupta, J. Hopcroft, R. Kannan, and P. Mitra. Spectral clustering by recursive partitioning. In *ESA'06: Proceedings of the 14th conference on Annual European Symposium*, pages 256–267. Springer-Verlag, 2006.
10. G. Dekker, M. Pechenizkiy, and J. Vleeshouwersu. Predicting students drop out: A case study. In *In Proceedings of Educational Data Mining 2009*, pages 41–50, 2009.
11. C. H. Q. Ding, X. He, H. Zha, M. Gu, and H. D. Simon. A min-max cut algorithm for graph partitioning and data clustering. In *ICDM '01: Proceedings of the 2001 IEEE International Conference on Data Mining*, pages 107–114, Washington, DC, USA, 2001. IEEE Computer Society.
12. W. E. Donath and A. J. Hoffman. Lower bounds for the partitioning of graphs. *IBM J. Res. Dev.*, 17(5):420–425, 1973.
13. P. Drázdilová, K. Slaninová, J. Martinovic, G. Obadi, and V. Snásel. Creation of students' activities from learning management system and their analysis. In *CASoN*, pages 155–160, 2009.
14. M. Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23:298–305, 1973.
15. L. C. Freeman. Centrality in social networks: Conceptual clarification. *Social Networks*, 1(3):215–239, 1979.
16. E. Gaudioso, M. Montero, L. Talavera, and F. H. del Olmo. Supporting teachers in collaborative student modeling: A framework and an implementation. *Expert Systems with Applications*, 36(2, Part 1):2260 – 2265, 2009.
17. D. J. Hand, P. Smyth, and H. Mannila. *Principles of data mining.* MIT Press, Cambridge, MA, USA, 2001.
18. R. Iváncsy and I. Vajk. Frequent pattern mining in web log data. *Acta Polytechnica Hungarica*, 3(1), 2006.
19. R. Kannan, S. Vempala, and A. Vetta. On clusterings: Good, bad and spectral. *J. ACM*, 51(3):497–515, May 2004.
20. L. Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18:39–43, 1953.
21. A. Laghos and P. Zaphiris. Sociology of student-centred e-learning communities: A network analysis. In *IADIS international conference*, Dublin, Ireland, July 2006. e-Society.
22. S. V. Nath. Crime pattern detection using data mining. In *WI-IATW '06: Proceedings of the 2006 IEEE/WIC/ACM international conference on Web Intelligence and Intelligent Agent Technology*, pages 41–44, Washington, DC, USA, 2006. IEEE Computer Society.
23. M. Newman. A measure of betweenness centrality based on random walks. *Social Networks*, 27(1):39–54, January 2005.
24. K. Okamoto, W. Chen, and X.-Y. Li. Ranking of closeness centrality for large-scale social networks. In *FAW '08: Proceedings of the 2nd annual international workshop on Frontiers in Algorithmics*, pages 186–195, Berlin, Heidelberg, 2008. Springer-Verlag.
25. A. Ortiz-Pelaez, D. Pfeiffer, R. Soares-Magalhes, and F. Guitian. Use of social network analysis to characterize the pattern of animal movements in the initial phases of the 2001 foot and mouth disease (fmd) epidemic in the uk. *Preventive Veterinary Medicine*, 76(1-2):40 – 55, 2006.

26. D. Perera, J. Kay, I. Koprinska, K. Yacef, and O. R. Zaïane. Clustering and sequential pattern mining of online collaborative learning data. *IEEE Trans. on Knowl. and Data Eng.*, 21(6):759–772, 2009.
27. S. Preidys and L. Sakalauskas. Analysis of students study activities in virtual learning environments using data mining methods. *Technological and economic development of economy*, 16(1):94–108, 2010.
28. C. Romero and S. Ventura. Educational data mining: A survey from 1995 to 2005. *Expert Systems with Applications*, 33(1):135 – 146, 2007.
29. G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.
30. J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:888–905, 1997.
31. M. Zorrilla, E. Menasalvas, D. Marn, E. Mora, and J. Segovia. *Computer Aided Systems Theory EUROCAST 2005*, volume 3643/2005 of *Lecture Notes in Computer Science*, chapter Web Usage Mining Project for Improving Web-Based Learning Sites. Springer Berlin / Heidelberg, 2005.

Deferred Node-copying Scheme for XQuery Processors

Jan Kurš and Jan Vraný

Software Engineering Group, FIT ČVUT,
Kolejní 550/2, 160 00, Prague, Czech Republic
`kurs.jan@post.cz, jan.vrany@fit.cvut.cz`

Abstract. XQuery is generic, widely adopted language for querying and manipulating XML data. Many of currently available native XML databases are using XQuery as its primary query language. The XQuery specification requires each XML node to belong to exactly one XML tree. In case of the XML subtree is appended into a new XML structure, the whole subtree has to be copied. This may lead into excessive and unnecessary data copying and duplication. In this paper, we present a new XML node copying scheme that defers the node data copy operation unless necessary. We will show that this schemes significantly reduces the XML node copy operations required during the query processing.

Keywords: XML, XQuery, XQuery Processor, Smalltalk

1 Introduction

XQuery is an XML query language designed by the World Wide Web Consortium. Although widely adopted, fast and efficient implementation is still lacking. Optimization techniques for XQuery are still a subject to an active research. XQuery 1.0 and XPath 2.0 Data Model specification [1] forbids sharing of data model among multiple XML node hierarchies. Section 2.1 says:

> ...
> *Every node belongs to exactly one tree, and every tree has exactly one root node.*
> ...

If a XML node is added into a new XML tree, the naive realization of this requirement would create a new node (by copying the original one) and the copy would be placed into the new XML tree. Consider the query at figure 1 is to be evaluated and its output is to be serialized to an output file.

A whole XML subtree that matches `fn:doc("doc.xml")//authors` is never used. This may lead into excessive node copying and higher memory consumptions depending on the subtree size.

In this paper we will describe an efficient node-copying scheme that avoids unnecessary copying while preserving XQuery semantics. We will also discuss its correctness and benchmark results.

```
let $authors = element authors { fn:doc("doc.xml")//authors }
let $titles = element titles { fn:doc("doc.xml")//titles }
return element result { $titles }
```

Fig. 1. Simple document-creating query

The paper is organized as follows: section 2 give an overall description of the node-copying scheme mentioned above. Section 3 discusses experimental results based on running XMark benchmarks. Section 4 provides a brief overview of related work. Section 5 concludes by summarizing presented work.

2 Deferred Node-copying

The basic idea is simple: share existing XML nodes between node hierarchies and defer node-copy operation unless absolutely inevitable. In our implementation the XML node can belong into multiple node hierarchies, although the XQuery specification requirement mentioned in section 1 is preserved.

The deferred node copying scheme has been developed to meet two main goals:

– separate query processing logic from underlying physical data model and
– reduce memory consumption by preventing unnecessary data copying

The first requirement has software engineering origin. XQuery processors should be able to operate over various data models, not necessarily XML-based. Moreover, good separation of query processor from physical data model provides possibility to use one XQuery implementation in multiple environments – as a standalone XQuery tools or within a database management machine.

The latter goal came from practical needs. In case of large documents and complex queries, naive implementation of an XQuery may consume – in edge cases – twice more memory than actually needed.

2.1 XDM Adaptor

XDM specification defines a *sequence* to be an instance of data model. Each sequence consists of zero or more *items*. An *item* is either a *node* or *atomic value*. The specification also defines a bunch of node properties such as `dm:node-name` or `dm:parent`.

To meet our first goal we separates node from its physical data storage though an *XDM adaptor* which operates on so called *node ids*. Node id is an unique identifier of an XML node within particular physical storage. The structure of the node id is not defined – in fact node id could be anything: reference to a DOM node in memory, pointer to a database file or simple integer.

Usage of XDM adaptor give us easy and straightforward way how to access different physical data models. XDM adaptor abstracts any kind of data source and may use any kind of optimization (such as extensive caching) to access data effectively. However the physical data storage and access strategies are hidden to the rest of the XQuery processor.

2.2 Node States

In order to defer copy operation, a new node property called *node state* is introduced. Each node is in exactly one state from following three states:

Accessed State. Nodes that come from external data source are in *accessed state.*

Constructed State. Nodes that are constructed during query processing are in *accessed state.*

Hybrid State. Nodes which belongs to multiple node hierarchies are in a *hybrid state.*

2.3 Actions

During the query processing, the state of the node may change. The state diagram of the node is shown at figure 2. There are three kinds of actions:

Copy Action. The copy action is performed whenever the XML subtree is appended into a new XML hierarchy. The original subtree should be duplicated in order to meet the requirement XML node to belong into just one node hierarchy.

Change Action. The change action models any change in a data model such as setting a new parent.

"Child Read" Action. The "child read" action represents the situation when the XQuery processor accesses child nodes of given node.

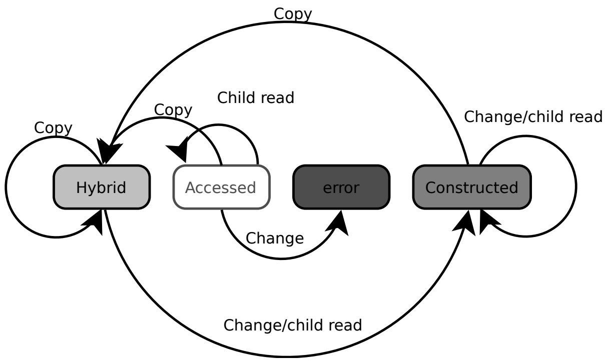


Fig. 2. Node State Transitions

Consider a document `doc.xml` (it's content is shown at figure 3) and query 1 (figure 4). During execution of the query, following actions are performed:

```
<?xml version="1.0"?>
<root>
  <elem>elem1</elem>
  <elem>elem2</elem>
</root>
```

Fig. 3. `doc.xml` contents

```
element myroot {
  attribute attr { 'value' },
  fn:doc("doc.xml")/elem[0]
}
```

Fig. 4. Example Query 1

1. The `myroot` element is created in a constructed state. Then *change actions* are issued on that node: setting the node name "myroot", adding attribute "attr" and appending a text node.
2. Afterwards, the `doc.xml` is read and two *child read actions* are performed in order to evaluate XPath expression.
3. Finally, the first `elem` (accessed) node from `doc.xml` is to be added into the `myroot` (constructed) node – the `elem` node and all its descendants should be copied.

2.4 Transitions

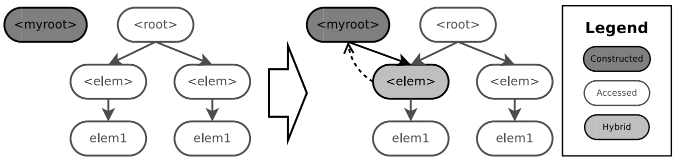


Fig. 5. Two XML trees sharing one hybrid node

Accessed Node Transitions. When a copy action of accessed node is triggered, the node state is changed from accessed to hybrid and no physical data copy is made. Changes to accessed nodes are not permitted – any change will immediately lead into an error.

Constructed Node Transitions. Copy operations on constructed nodes behaves exactly as on accessed nodes. Changes to constructed nodes are permitted.

Hybrid Node Transitions. Transitions based on actions on hybrid nodes are bit more interesting:

Copy Action. Copy action on hybrid nodes is a no-op. As a result, the same node is returned with its state unchanged.

Change Action. Whenever any of node properties (dm:parent, dm:name etc.) is to be changed the node state is changed to constructed and all node properties are copied. See the query at figure 6. When processing expression at line 5, two things happen (in that order):

1. The text node *“elem1”* (a result of `$doc/elem[0]/text()` expression) is added to the `myroot` element. States of nodes after this addition are depicted at left side of figure 7.
2. Afterwards, the text node value *“elem1“* has to be changed to the *“elem1 is the first”* because of the specification requirements. Obviously, the hybrid text node must be copied. The XML data accessible though `$doc` must remain unchanged.

Child Read Action. While appending a XML tree into a new structure, the state of a root node of the appended tree is changed to hybrid, the reference from the new structure is added to the hybrid. The rest of the appended tree (children of the root node) are unchanged – they don’t know, that their parent has changed its state to hybrid. This cause serious problems while executing XPath commands. To overcome this issue, we convert hybrid node to a constructed one during child read action. Such a behavior is illustrated at figure 8.

Data are physically copied only when hybrid node is either being changed or its children are being read.

```
1 let $doc:= doc("doc.xml")
2 return
3   element myroot {
4     element myelem {
5       { $doc/elem[0]/text() } is the first }
6     }
7   }
```

Fig. 6. Example Query 2

Serialization of Result Set. Once the query is processed, serialization of result set may not lead into XML node copying. Because query is already processed, no node kind transitions must be performed during serialization and thus no node copies must be created. Obviously, if the application wants work with the result set as with nodes in memory and wants to perform some modification on it, the result set must be copied.

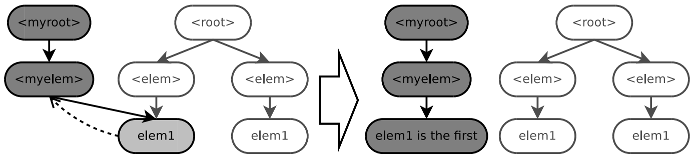


Fig. 7. Change of hybrid node into the constructed node

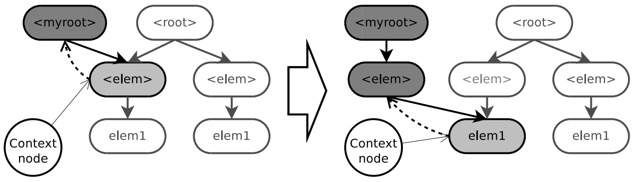


Fig. 8. Change of hybrid node while exploring the children

3 Discussion

3.1 Specification Conformance

Although deferred node-copying scheme does not require the XML nodes to belong to exactly one node hierarchy it preserves original XQuery semantics. Our claim is based on the results from the XQuery Test Suite [3].

The *axes tests* and *element constructors tests* from *Minimal Conformance - Expressions* section of XQTS Catalogue cover the node identity semantics and were used to test the correctness of deferred node-copying scheme. Our proof-of-concept implementation successfully passes all the mentioned test cases.

3.2 Benchmarks

Presented deferred node-copying scheme has been developed in order to increase XQuery processor performance by reducing number of copy operations. A natural question is whether this scheme has substantial effect in real-world applications. The table 3 shows number of copy operations for selected XMark [2] queries[1] on a file created with the XMark data generator.

Number of saved copies is dependent on a query characteristics. There are no new nodes created in a Q1 command and that is why there is no difference in results. There are text nodes appended to elements in a Q2 command. The text nodes does not need to be copied at all, only transformed to the hybrid state.

There is a subtree appended to each result item during the Q13 execution. Without the optimization, each element of a tree has to be copied, but with the optimization turned on, only a few of nodes are copied.

[1] Plus one nonstandard query marked INC. Its code is `element a {doc("file:///auctions.xml") }`. We include it as an illustration of extreme case.

Q. #	**DNC**		**IC**		Q. #	**DNC**		**IC**	
	N_h	N_c	N_h	N_c		N_h	N_c	N_h	N_c
Q1	0	0	0	0	Q2	106	0	0	106
Q3	0	44	0	44	Q4	0	0	0	0
Q5	0	0	0	0	Q6	0	0	0	0
Q7	0	0	0	0	Q8	25	25	0	50
Q9	12	25	0	39	Q10	402	1	0	1244
Q11	12	25	0	39	Q12	3	3	0	6
Q13	22	22	0	560	Q14	0	0	0	0
Q15	7	0	0	7	Q16	0	6	0	6
Q17	0	138	0	138	Q18	0	0	0	0
Q19	217	217	0	434	Q20	8	0	0	12
INC	2074	114	0	5857					

Legend:

N_h – number of hybrid nodes created
N_c – number of physically copied nodes
DNC – evaluated using deferred node-copying scheme
IC – evaluated using immediate copy as specified by the XQuery specification

Table 1. Benchmark results

4 Related Work

eXist XQuery Processor. eXist[2] is an open-source XML-native database with XQuery as its primary query language. As far as we know, eXist XQuery implementation unconditionally copies nodes whenever the node is to be added into a different node hierarchy. Our approach is different since we avoid unnecessary copy operations.

Saxon XQuery Processor. Saxon[3] is well-known, widely adopted XML tool set including XSLT 2.0, XPath 2.0 and XQuery 1.0 processor. Saxon's XQuery processor introduces concept of *virtual nodes* – a light-weigh node shallow copies that shares as many properties as possible with their origin.

Similarly to our approach, for a given virtual node some of standard XDM properties may be overridden – namely the parent property. When the Saxon

[2] http://exist.sourceforge.net/
[3] http://saxon.sourceforge.net/

XQuery processor iterates over virtual node's children, those are converted to virtual nodes.

However, presented deferred node copying scheme differs from virtual nodes approach in several aspects:

1. Creating virtual copies requires a new object to be allocated in the memory. Deferred node copying scheme shares the same object.
2. Creation of virtual copies is a part of XQuery processing logic and must be explicitly expressed, whereas our approach separates copying logic of an XDM model from the query evaluation logic.

5 Conclusion and Future Work

This paper presents a deferred XML node-copying scheme for XQuery processors that significantly reduces number of source nodes copy operations required during query processing. This scheme defers the copy operation unless absolutely inevitable. Whether the node is actually copied depends on a node state, a new property which is maintained for each node in addition to standard XDM properties. Correctness of this approach has been successfully tested by XQuery Test Suite.

The main benefits of deferred node-copying scheme are: (i) efficiency, (ii) easy to implement, (iii) independent on physical data model and (iv) independent on XQuery processing logic.

As a future plan, we plan to extend this scheme for use with various XML indexing approaches, Ctree [4] and [5] in particular.

References

1. M. N. Mary Fernández, Ashok Malhotra. Jonathan Marsh and N. Walsh. *XQuery 1.0 and XPath 2.0 Data Model (XDM).* W3C, 1st edition, 2006. `http://www.w3.org/TR/xpath-datamodel`.
2. A. Schmidt, F. Waas, M. Kersten, M. J. Carey, I. Manolescu, and R. Busse. Xmark: A benchmark for xml data management. In *In VLDB*, pages 974–985, 2002.
3. W3C XML Query Working Group. *XML Query Test Suite.* W3C, 1st edition, 2006. `http://www.w3.org/XML/Query/test-suite/`.
4. Q. Zou, S. Liu, and W. W. Chu. Ctree: a compact tree for indexing xml data. In *WIDM '04: Proceedings of the 6th annual ACM international workshop on Web information and data management*, pages 39–46, New York, NY, USA, 2004. ACM.
5. Q. Zou, S. Liu, and W. W. Chu. Using a compact tree to index and query xml data. In *CIKM '04: Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 234–235, New York, NY, USA, 2004. ACM.

Denotational Semantics of the XML-λ Query Language[⋆]

Pavel Loupal[1] and Karel Richta[2]

[1] Department of Computer Science and Engineering, FEL ČVUT
Karlovo nám. 13, 121 35 Praha 2
`loupalp@fel.cvut.cz`

[2] Department of Software Engineering MFF UK,
Malostranske nam. 25, 118 00 Praha 1
`richta@ksi.mff.cuni.cz`

Abstract. In this paper, we define formally the XML-λ Query Language, a query language for XML, that employs the functional data model. We describe its fundamental principles including the abstract syntax and denotational semantics. The paper basically aims for outlining of the language scope and capabilities.

1 Introduction

In this paper, we define formally the XML-λ Query Language, a query language for XML, that employs the functional data model. The first idea for such an attitude was published in [4]. This research brought in the key idea of a functional query processing with a wide potential that was later proven by a simple prototype implementation [6].

We can imagine two scenarios for this language; firstly, the language plays a role of a full-featured query language for XML (it has both formal syntax and semantics and there is also an existing prototype that acts as a proof-of-the-concept application). In the second scenario, the language is utilized as an intermediate language for the description of XQuery semantics. In [3] we propose a novel method for XQuery evaluation based on the transformation of XQuery queries into their XML-λ equivalents and their subsequent evaluation. As an integral part of the work, we have designed and developed a prototype of an XML-λ query processor for validating the functional approach and experimenting with it.

2 XML-λ Query Language

In this section, we describe the query language XML-λ, that is based on the simply typed lambda calculus. As a formal tool we use the approach published in

[⋆] This work has been supported by the Ministry of Education, Youth and Sports under Research Program No. MSM 6840770014 and also by the grant project of the Czech Grant Agency (GAČR) No. GA201/09/0990.

Richta's overview of semantics [5]. For listing of language syntax, we use the Extended Backus-Naur Form (EBNF) and for meaning of queries the denotational semantics [5].

2.1 Language of Terms

Typical query expression has a query part — an expression to be evaluated over data — and a constructor part that wraps a query result and forms the output. The XML-λ Query Language is based on λ-terms defined over the type system $\mathcal{T}_E$ as shown later. Lambda calculus, written also as λ-calculus, is a formal mathematical system for investigation of function definition and application. It was introduced by Alonzo Church and has been utilized in many ways. In this work, we use a variant of this formalism, the simply-typed λ-calculus, as a core for the XML-λ Query Language. We have gathered the knowledge from [7] and [1]. Our realization is enriched by usage of tuples.

The main constructs of the language are *variables*, *constants*, *tuples*, *projections*, and λ-calculus operations — *applications* and *abstractions*. The syntax is similar to λ-calculus expressions, thus the queries are structured as nested λ-expressions, i.e., $\lambda \ldots (\lambda \ldots (expression) \ldots)$. In addition, there are also typical constructs such as logical connectives, constants, comparison predicates, and a set of built-in functions.

Language of terms is defined inductively as the least set containing all terms created by the application of the following rules. Let $T, T_1, \ldots, T_n$, $n \geq 1$ be members of $\mathcal{T}_E$. Let $\mathcal{F}$ be a set of typed constants. Then:

1. *variable:* each variable of type T is a term of type T
2. *constant:* each constant (member of $\mathcal{F}$) of type T is a term of type T
3. *application:* if M is a term of type $((T_1, \ldots, T_n) \to T)$ and $N_1, \ldots, N_n$ are terms of the types $T_1, \ldots, T_n$,then $M(N_1, \ldots, N_n)$ is a term of the type T
4. λ-*abstraction:* if $x_1, \ldots, x_n$ are distinct variables of types $T_1, \ldots, T_n$ and M is a term of type T, then $\lambda x_1 : T_1, \ldots, x_n : T_1.(M)$ is a term of type $((T_1, \ldots, T_n) \to T)$
5. n-*tuple:* if $N_1, \ldots, N_n$ are terms of types $T_1, \ldots, T_n$, then $(N_1, \ldots, N_n)$ is a term of type $(T_1, \ldots, T_n)$
6. *projection:* if $(N_1, \ldots, N_n)$ is a term of type $(T_1, \ldots, T_n)$, then $N_1, \ldots, N_n$ are terms of types $T_1, \ldots, T_n$
7. *tagged term:* if N is a term of type $NAME$ and M is a term of type T then $N : M$ is a term of type $(\mathbf{E} \to T)$.

Terms can be interpreted in a standard way by means of an interpretation assigning to each constant from $\mathcal{F}$ an object of the same type, and by a semantic mapping from the language of terms to all functions and Cartesian products given by the type system $\mathcal{T}_E$. Speaking briefly, an application is evaluated as an application of of the associated function to given arguments, an abstraction 'constructs' a new function of the respective type. The tuple is a member of Cartesian product of sets of typed objects. A tagged term is interpreted as a function defined only for one $e \in \mathbf{E}$. It returns again a function.

3 Abstract Syntax

As for evaluation of a query, we do not need its complete derivation tree; such information is too complex and superfluous. Therefore, in order to diminish the domain that needs to be described without any loss of precision, we employ the *abstract syntax*. With the abstract syntax, we break up the query into logical pieces that forming an abstract syntax tree carrying all original information constitute an internal representation suitable for query evaluation. We introduce syntactic domains for the language, i.e., logical blocks a query may consist of. Subsequently, we list all production rules. These definitions are later utilized in Section 4 within the denotational semantics.

3.1 Syntactic Domains

By the term *syntactic domain*, we understand a logical part of a language. In Table 1, we list all syntactic domains of the XML-λ Query Language with their informal meaning. Notation $Q : Query$ stands for the symbol Q representing a member of the $Query$ domain.

$Q : Query$	XML-λ queries,
$O : Option$	XML-λ options – XML input attachements,
$C : Constructor$	XML-λ constructors of output results,
$E : Expression$	general expressions, yield a $BaseType$ value,
$T : Term$	sort of expression, yield a $BaseType$ value,
$F : Fragment$	sub-parts of a $Term$,
$BinOp : BinOperator$	binary logical operators,
$RelOp : RelOperator$	binary relational operators,
$N : Numeral$	numbers,
$S : String$	character strings,
$Id : Identifier$	strings conforming to the $Name$ syntactic rule in [2],
$NF : Nullary$	identifiers of nullary functions (subset of $Identifier$),
$Proj : Projection$	identifiers for projections (subset of $Identifier$).

Table 1. Syntactic domains of the XML-λ Query Language

3.2 Abstract Production Rules

The *abstract production rules* listed in Table 2 (written using EBNF) connect the terms of syntactic domains from the previous section into logical parts with suitable level of details for further processing. On the basis of these rules, we will construct the denotational semantics of the language.

4 Denotational Semantics

For description the meaning of each XML-λ query, we use *denotational semantics*. The approach is based on the idea that for each correct syntactic construct of the language we can define a respective meaning of it as a formal expression in

$$
\begin{array}{lll}
\mathit{Query} & ::= & \mathit{Options}\ \mathit{Constructor}\ \mathit{Expression} \\
\mathit{Constructor} & ::= & \mathit{ElemConstr}+\ \mid \mathit{Identifier}+ \\
\mathit{ElemConstr} & ::= & \mathit{Name}\ \mathit{AttrConstr}*\ (\mathit{Identifier} \mid \mathit{ElemConstr}) \\
\mathit{AttrConstr} & ::= & \mathit{Name}\ \mathit{Identifier} \\
\mathit{Expression} & ::= & \mathit{Fragment} \\
\mathit{Fragment} & ::= & \mathit{Nullary} \mid \mathit{Identifier} \mid \mathit{Fragment}\ \mathit{Projection} \\
 & & \mid \mathit{SubQuery} \mid \mathit{FunctionCall} \mid \mathit{Numeral} \mid \mathit{String} \mid \mathit{Boolean} \\
\mathit{Term} & ::= & \mathit{Boolean} \mid \mathit{Filter} \mid \text{'}\texttt{not}\text{'}\ \mathit{Term} \mid \mathit{Term}\ \mathit{BinOper}\ \mathit{Term} \\
\mathit{Filter} & ::= & \mathit{Fragment}\ \mathit{RelOper}\ \mathit{Fragment} \\
\mathit{SubQuery} & ::= & \mathit{Identifier}+\ \mathit{Expression} \\
\mathit{BinOper} & ::= & \text{'}\texttt{or}\text{'} \mid \text{'}\texttt{and}\text{'} \\
\mathit{RelOper} & ::= & \text{'}\texttt{<=}\text{'} \mid \text{'}\texttt{<}\text{'} \mid \text{'}\texttt{=}\text{'} \mid \text{'}\texttt{!=}\text{'} \mid \text{'}\texttt{>}\text{'} \mid \text{'}\texttt{>=}\text{'} \\
\mathit{Numeral} & ::= & \text{Digit+} \mid \text{Numeral}\ \ '.'\ \ \text{Digit+} \\
\mathit{Digit} & ::= & \text{'}\texttt{0}\text{'} \mid \text{'}\texttt{1}\text{'} \mid \text{'}\texttt{2}\text{'} \mid \text{'}\texttt{3}\text{'} \mid \text{'}\texttt{4}\text{'} \mid \text{'}\texttt{5}\text{'} \mid \text{'}\texttt{6}\text{'} \mid \text{'}\texttt{7}\text{'} \mid \text{'}\texttt{8}\text{'} \mid \text{'}\texttt{9}\text{'} \\
\mathit{Identifier} & ::= & \mathit{Name} \\
\mathit{Projection} & ::= & \mathit{Identifier} \\
\mathit{Nullary} & ::= & \mathit{Identifier}
\end{array}
$$

Table 2. Abstract production rules for the XML-λ Query Language

another, well-known, notation. We can say that the program is the denotation of its meaning. The validity of the whole approach is based on structural induction; i.e, that the meaning of more complex expressions is defined on the basis of their simpler parts. As the notation we employ the *simply typed lambda calculus*. It is a well-known and formally verified tool for such a purpose.

4.1 Prerequisites

The denotational semantics utilizes a set of functions for the definition of the language meaning. For this purpose, we formulate all necessary mathematical definitions. We start with the data types and specification of the evaluation context followed by the outline of bindings to the $\mathcal{T}_E$ type system. Then, all auxiliary and denotation functions are introduced.

Data Types. Each value computed during the process of the query evaluation is of a type from $Type$. Let E be a type from the type system $\mathcal{T}_E$, we define $Type$ as:

$$
\begin{array}{l}
Type ::= \ BaseType \mid SeqType \\
SeqType ::= \ \perp \mid BaseType \times SeqType \\
BaseType ::= \ E \mid PrimitiveType \\
PrimitiveType ::= \ Boolean \mid String \mid Number
\end{array}
$$

Primitive types, $Boolean$, $String$, and $Number$, are defined with their set of allowed values as usual. The type $SeqType$ is the type of all ordered sequences of elements of base types[3]. We do not permit sequences of sequences. The symbol $\perp$ stands for the empty sequence of types – represents an unknown type. More

[3] We suppose usual functions cons, append, null, head, and tail for sequences.

precisely, we interpret types as algebraic structures, where for each type $\tau \in Type$ there is exactly one carrier $\mathcal{V}_\tau$, whose elements are the values of the respective type τ.

Variables. An XML-λ query can use an arbitrary (countable) number of variables. We model variables as pairs $name : \tau$, where *name* refers to a variable name and τ is the data type of the variable – any member of $Type$. Syntactically, variable name is always prepended by the dollar sign. Each expression in XML-λ has a recognizable type, otherwise both the type and the value are undefined.

Query Evaluation Context. During the process of query evaluation we need to store variables inside a working space known as a context. Formally, we denote this context as the *State*. We usually understand a state as the set of all active objects and their values at a given instance. We denote the semantic domain *State* of all states as a set of all functions from the set of identifiers $Identifier$ into their values of the type $\tau \in Type$. Obviously, one particular state $\sigma : State$ represents an immediate snapshot of the evaluation process; i.e., values of all variables at a given time. We denote this particular value for the variable x as $\sigma[\![x]\!]$. Simply speaking, the state is the particular valuation of variables. We use the functor $f[x \leftarrow v]$ for the definition of a function change in one point x to the value v.

4.2 Auxiliary Functions

For the sake of readability improvement, we propose few semantic functions, denoted as *auxiliary*, that should make the denotations more legible. We introduce functions: $isIdent$ — returns *true* iff its argument denotes an variable identifier in a given *State*, $typeOf$ — returns a type of given argument (one type from the *Type* set), $valueOf$ — returns a typed value of an expression, *bool* — evaluates its argument as a Boolean value, *num* — converts its argument into a numeric value, and *str* — converts its argument into a string value.

Each expression e has a distinguished type in a state σ. The type can depend on the state because an expression can contain variables. This type is available by calling the $typeOf$ semantic function defined in Table 3.

$$typeOf \; : \; Expression \; \times \; State \; \rightarrow \; Type$$

4.3 XML Schema-Specific Functions

For utilization of the features offered by the XML-λ Framework we propose a number of functions working with information available in the type system. These functions help us to access an arbitrary data model instance. An *application* is informally used for accessing child elements of a given one. More formally, it is an evaluation of a T-object specified by its name. A *projection* is generally used for selecting certain items from a sequence. A *nullary function.* A T-nullary function returns all abstract elements from $\mathbf{E}_T$. *Root Element Access* is a shortcut for a common activity in the XML world — accessing the root element of

$$typeOf[\![e]\!](\sigma) = \begin{cases} \perp & \text{if } e \text{ is a nullary fragment} \\ Boolean & \text{if } e \in \nu_{Boolean} \\ & (e \text{ is a constant of the type } Boolean) \\ Numeral & \text{if } e \in \nu_{Numeral} \\ & (e \text{ is a constant of the type } Numeral) \\ String & \text{if } e \in \nu_{String} \\ & (e \text{ is a constant of the type } String) \\ \tau & \text{if } isIdent[\![e]\!](\sigma) \text{ and } \sigma[\![e]\!] : \tau \\ & (e \text{ is a variable of the type } \tau) \\ Boolean & \text{if } e \text{ is a relational fragment (filter)} \\ & e_1 \; RelOper \; e_2 \\ Boolean & \text{if } e \text{ is a logical expression} \\ & e_1 \; BinOper \; e_2, \text{ or } not \; e_1 \end{cases}$$

Table 3. Types of general expressions

$app_{XMLDoc} : E \rightarrow SeqType$
$proj_{XMLDoc} : SeqType \times \tau \rightarrow SeqType$
$null_{XMLDoc} : E \times T \rightarrow SeqType$
$root_{XMLDoc} : E$

4.4 Signatures of Semantic Functions

Having defined all necessary prerequisites and auxiliary functions (recalling that the *SeqType* represents any permitted type of value), we formalize semantic functions over semantic domains as

$$\begin{array}{ll} Sem_{Query} & : Query \rightarrow (XMLDoc \rightarrow SeqType) \\ Sem_{Options} & : Options \rightarrow (State \rightarrow State) \\ Sem_{Expr} & : Expression \rightarrow (State \rightarrow SeqType) \\ Sem_{Term} & : Term \rightarrow (State \rightarrow Boolean) \\ Sem_{Frag} & : Fragment \rightarrow (State \rightarrow SeqType) \\ Sem_{RelOper} & : Fragment \times RelOper \times Fragment \rightarrow (State \rightarrow Boolean) \\ Sem_{BinOper} & : Term \times BinOper \times Term \rightarrow (State \rightarrow Boolean) \end{array}$$

4.5 Semantic Equations

We start with the semantic equations for the expressions. Each expression e has a value $Sem_{Expr}[\![e]\!](\sigma)$ in a state σ. The state represents values of variables. The result is a state, where all interesting values are bound into local variables.
Resulting values are created by constructors. A constructor is a list of items which can be variable identifier or constructing expression. Resulting values can be created by element constructors. Elements can have attributes assigned by attribute constructors.
Options and Queries. The only allowed option in the language is now the specification of input XML documents. We explore a function $\mathcal{D}om(X)$ that converts input XML document X into its internal representation accessible under identification $X\#$. A query consists of query options, where input XML documents

$Sem_{Term}\llbracket B \rrbracket$	$= \lambda\sigma : State.bool\llbracket B \rrbracket$ if B is a constant of the type $Boolean$
$Sem_{Term}\llbracket F_1\ RelOp\ F_2 \rrbracket$	$= \lambda\sigma : State.Sem_{RelOper}\llbracket F_1\ RelOp\ F_2 \rrbracket\sigma$
$Sem_{Term}\llbracket '\texttt{not}'\, T \rrbracket$	$= \lambda\sigma : State.not(Sem_{Term}\llbracket T \rrbracket\sigma)$
$Sem_{BinOper}\llbracket T_1\ '\texttt{or}'\ T_2 \rrbracket$	$= \lambda\sigma : State.(Sem_{Term}\llbracket T_1 \rrbracket\sigma\ or\ Sem_{Term}\llbracket T_2 \rrbracket\sigma)$
$Sem_{BinOper}\llbracket T_1\ '\texttt{and}'\ T_2 \rrbracket$	$= \lambda\sigma : State.(Sem_{Term}\llbracket T_1 \rrbracket\sigma\ and\ Sem_{Term}\llbracket T_2 \rrbracket\sigma)$
$Sem_{Term}\llbracket T_1\ BinOper\ T_2 \rrbracket$	$= \lambda\sigma : State.Sem_{BinOper}\llbracket T_1\ BinOper\ T_2 \rrbracket\sigma$

Table 4. Semantic equations for terms, relational and binary operators

$$
\begin{aligned}
& Sem_{AttrConstr}\llbracket N\ I \rrbracket\sigma = attribute(N, Sem_{Expr}\llbracket I \rrbracket\sigma)\\
& Sem_{ElemConstr}\llbracket N A_1 ... A_n I \rrbracket\sigma =\\
& = element(N, \sigma\llbracket I \rrbracket, Sem_{AttrCons}\llbracket A_1 \rrbracket\sigma, ..., Sem_{AttrCons}\llbracket A_n \rrbracket\sigma)\\
& Sem_{ElemConstr}\llbracket N A_1 ... A_n E \rrbracket\sigma =\\
& = element(N, Sem_{Expr}\llbracket E \rrbracket\sigma, Sem_{AttrCons}\llbracket A_1 \rrbracket\sigma, ..., Sem_{AttrCons}\llbracket A_n \rrbracket\sigma)\\
& Sem_{ElemConstr}\llbracket N\ I \rrbracket\sigma = element(N, \sigma\llbracket I \rrbracket, nil)\\
& Sem_{ElemConstr}\llbracket N\ E \rrbracket\sigma = element(N, Sem_{Expr}\llbracket E \rrbracket\sigma, nil)\\
& Sem_{Cons}\llbracket E_1 E \rrbracket\sigma = append(Sem_{ElemCons}\llbracket E_1 \rrbracket\sigma, Sem_{Cons}\llbracket E \rrbracket\sigma)\\
& Sem_{Cons}\llbracket I_1 E \rrbracket\sigma = cons(\sigma\llbracket I_1 \rrbracket, Sem_{Cons}\llbracket E \rrbracket\sigma)\\
& Sem_{Cons}\llbracket\, \rrbracket\sigma = nil
\end{aligned}
$$

Table 5. The semantic equation for constructors

$Sem_{Frag}\llbracket Null \rrbracket$	$= \lambda\sigma : State.null_{XMLDoc}\llbracket Null \rrbracket$
$Sem_{Frag}\llbracket Id \rrbracket$	$= \lambda\sigma : State.\sigma\llbracket Id \rrbracket$
$Sem_{Frag}\llbracket f(E_1, ..., E_n) \rrbracket$	$= \lambda\sigma : State.f(Sem_{Expr}\llbracket E_1 \rrbracket\sigma, ..., Sem_{Expr}\llbracket E_n \rrbracket\sigma)$
$Sem_{Frag}\llbracket F\ P \rrbracket$	$= \lambda\sigma : State.(Sem_{Frag}\llbracket F \rrbracket \circ Sem_{Frag}\llbracket P \rrbracket)\sigma$
$Sem_{Frag}\llbracket (subquery)(arg) \rrbracket$	$= \lambda\sigma : State.(Sem_{Expr}\llbracket subquery \rrbracket(\sigma)(Sem_{Expr}\llbracket arg \rrbracket(\sigma)))$
$Sem_{Frag}\llbracket I_1 I_2 ... I_n E \rrbracket$	$= Sem_{Expr}\llbracket I_2 ... I_n E \rrbracket(\sigma[Sem_{Expr}\llbracket E \rrbracket\sigma \leftarrow I_1])$
$Sem_{Frag}\llbracket N \rrbracket$	$= \lambda\sigma : State.num\llbracket N \rrbracket$ if N is a constant of the type $Numeral$
$Sem_{Frag}\llbracket S \rrbracket$	$= \lambda\sigma : State.str\llbracket S \rrbracket$ if S is a constant of the type $String$
$Sem_{Frag}\llbracket B \rrbracket$	$= \lambda\sigma : State.bool\llbracket B \rrbracket$ if B is a constant of the type $Boolean$
$Sem_{Expr}\llbracket F \rrbracket\sigma$	$= Sem_{Frag}\llbracket F \rrbracket\sigma$

Table 6. Semantic equations for fragments and expressions

$$
\begin{array}{l}
Sem_{Query}[\![O\ C\ E]\!] = \\
= \ \lambda\delta : XMLDoc.(Sem_{Cons}[\![C]\!](Sem_{Expr}[\![E]\!](Sem_{Options}[\![O]\!](\lambda\sigma.\bot)(\delta))) \\
Sem_{Query}[\![Q]\!](nil) = nil \\
Sem_{Query}[\![Q]\!](cons(H,T)) = append(Sem_{Query}[\![Q]\!](H), SemQuery[\![Q]\!](T)) \\
Sem_{Options}[\![\ ]\!] = \lambda\sigma : State.\bot \\
Sem_{Options}[\![\ \texttt{xmldata(}\ X\ \texttt{)}\ Y]\!] = \lambda\sigma : State.Sem_{Options}[\![Y]\!](\sigma[\mathcal{D}om(X) \leftarrow X\#])
\end{array}
$$

Table 7. Semantic equations for options and queries

are bound to its formal names, the query expression to be evaluated, and the output construction commands. First, input files are elaborated, than an initial variable assignment takes place, followed by evaluation of expression. Finally, the output is constructed. The whole meaning of a query can be modeled as a mapping from the sequence of input XML documents into a sequence of output values of the type of $Type$.

5 Conclusions

In this paper, we have presented syntax and denotational semantics of the XML-λ Query Language, a query language for XML based on simply typed lambda calculus. We use this language within the XML-λ Framework as an intermediate form of XQuery expressions for description of its semantics. Nevertheless the language in its current version does not support all XML features, e.g. comments, processing instructions, or deals only with type information available in DTD, it can be successfully utilized for fundamental scenarios both for standalone query evaluation or as a tool for XQuery semantics description.

References

1. H. Barendregt. Lambda calculi with types. In *Handbook of Logic in Computer Science, Volumes 1 (Background: Mathematical Structures) and 2 (Background: Computational Structures), Abramsky & Gabbay & Maibaum (Eds.), Clarendon*, volume 2. Oxford University Press, 1992.
2. T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible markup language (XML) 1.0 (fourth edition), August 2006. `http://www.w3.org/TR/2006/REC-xml-20060816`.
3. P. Loupal. *XML-λ : A Functional Framework for XML*. Ph.D. Thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, February 2010. Submitted.
4. J. Pokorný. XML functionally. In B. C. Desai, Y. Kioki, and M. Toyama, editors, *Proceedings of IDEAS2000*, pages 266–274. IEEE Computer Society, 2000.
5. K. Richta and J. Velebil. *Sémantika programovacích jazyku.* Univerzita Karlova, 1997.
6. P. Šárek. Implementation of the XML lambda language. Master's thesis, Dept. of Software Engineering, Charles University, Prague, 2002.
7. J. Zlatuška. *Lambda-kalkul.* Masarykova univerzita, Brno, Česká republika, 1993.

Modeling and Verification of Priority Assignment in Real-Time Databases Using Uppaal[⋆]

Martin Kot

Center for Applied Cybernetics, Department of Computer Science, FEI,
VSB - Technical University of Ostrava,
17. listopadu 15, 708 33, Ostrava-Poruba, Czech Republic
martin.kot@vsb.cz

Abstract. Real-time database management systems (RTDBMS) are recently subject of an intensive research. Model checking algorithms and verification tools are of great concern as well. In this paper we show some possibilities of using a verification tool Uppaal on some variants of priority assignment algorithms. We present some possible models of such algorithms expressed as nets of timed automata, which are a modeling language of Uppaal.

Keywords: real-time database systems, priority assignment, timed automata, model checking, verification, verification tool, Uppaal

1 Introduction

Many real-time applications need to store some data in a database. It is possible to use traditional database management systems (DBMS). But they are not able to guarantee any bounds on a response time. This is the reason why so-called real-time database management systems (RTDBMS) emerged (e.g. [1, 6]).

Formal verification is of great interest recently and finds its way quickly from theoretical papers into a real live. It can prove that a system (or more exactly a model of a system) has a desired behavior. The difference between testing and formal verification is that during testing only some possible computations are chosen. Formal verification can prove correctness of all possible computations. A drawback of formal verification is that for models with high descriptive power are almost all problems undecidable. It is important to find a model with an appropriate descriptive power to capture a behavior of a system, yet with algorithmically decidable verification problems.

There are two main approaches to fully automated verification – equivalence checking and model checking. Using equivalence checking, two models of systems (usually model of specification and model of implementation) are compared using

⋆ Author acknowledges the support by the Czech Ministry of Education, Grant No. 1M0567.

some behavioral equivalence. In this paper we consider the other approach – so called model checking (see e.g. [4, 9]). This form of verification uses a model of a system in some formalism and a property expressed usually in the form of formula in some temporal logic. Model checking algorithm checks whether the property holds for the model of a system. There are quite many automated verification tools which implement model checking algorithms (see e.g. [11] for overview). Those tools use different modeling languages or formalisms and different logics.

The idea of our research is to explore possibilities of using existing verification tools on RTDBMS. To our best knowledge, there are only rare attempts of automated formal verification of real-time database system. In fact we know about one paper ([10]) only where authors suggested a new pessimistic protocol and verified it using Uppaal. They presented two small models covering only their protocol.

There is not any verification tool intended directly for real-time database systems. We have chosen the tool Uppaal because it is designed for real-time systems. But, it is supposed to be used on so-called reactive systems, which are quite different from database systems. So we need to find some possibilities how to deal with it. Big problem of verification tools is so called state space explosion. Uppaal is not able to manage too detailed models. On the other hand, too simple models can not catch important properties of a real system. So we need to find a suitable level of abstraction.

One of the most important and crucial parts of all database management systems allowing concurrent access to data records is concurrency control. Some of protocols used for concurrency control were modeled and verified using Uppaal in [7, 8].

In this paper, we will concentrate on other important part of real-time database systems – priority assignment. There are many parameters that can be used for determination of priority of database transaction. Some of them are criticality, deadline, amount of resources already used by transaction etc. There were several algorithms presented for this task. In this paper we will consider two of them presented in [1] – First Come First Serve (Section 4) and Earliest Deadline (Section 5). The nature of this paper should be mainly proof of concept. There are not any new informations found about described algorithms or any errors in them discovered. But some general possibilities of modeling using nets of timed automata are shown which can be used for verification and comparison of, e.g., newly designed algorithms in the future.

Before we will discuss concrete models of algorithms, we will shortly describe the tool Uppaal in the Section 2 and talk some general possibilities and assumptions over in Section 3.

2 Verification tool Uppaal

Uppaal ([3]) is a verification tool for real-time systems. It is jointly developed by Uppsala University and Aalborg University. It is designed to verify systems that can be modeled as networks of timed automata extended with some further features such as integer variables, structured data types, user defined functions, channel synchronization and so on.

A timed automaton is a finite-state automaton extended with clock variables. A dense-time model, where clock variables have real number values and all clocks progress synchronously, is used. In Uppaal, several such automata working in parallel form a network of timed automata. An automaton has locations and edges. Each location has an optional name and invariant. An invariant is a conjunction of side-effect free expressions of the form $x < e$ or $x \leq e$ where x is a clock variable and e evaluates to an integer. Each automaton has exactly one initial location.

Particular automata in the network synchronize using channels and values can be passed between them using shared (global) variables. A state of the system is defined by the locations of all automata and the values of clocks and discrete variables. The state can be changed in two ways - passing of time (increasing values of all clocks by the same amount) and firing an edge of some automaton (possibly synchronizing with another automaton or other automata). Some locations may be marked as committed. If at least one automaton is in a committed location, time passing is not possible, and the next change of the state must involve an outgoing edge of at least one of the committed locations.

Each edge may have a guard, a synchronization and an assignment. Guard is a side-effect free expression that evaluates to a boolean. The guard must be satisfied when the edge is fired. It can contain not only clocks, constants and logical and comparison operators but also integer and boolean variables and (side-effect free) calls of user defined functions. Synchronization label is of the form *Expr*! or *Expr*? where *Expr* evaluates to a channel. An edge with $c!$ synchronizes with another edge (of another automaton in the network) with label $c?$. Both edges have to satisfy all firing conditions before synchronization. Sometimes we say that automaton firing an edge labeled by $c!$ sends a message c to the automaton firing an edge labeled by $c?$. There are urgent channels as well (synchronization through such a channel have to be done in the same time instant when it is enabled) and broadcast channels (any number of $c?$ labeled edges are synchronized with one $c!$ labeled edge). An assignment is a comma separated list of expressions with a side-effect. It is used to reset clocks and set values of variables.

Uppaal has some other useful features. Templates are automata with parameters. These parameters are substituted with given arguments in the process declaration. This enables easy construction of several alike automata. Moreover, we can use bounded integer variables (with defined minimal and maximal value), arrays and user defined functions. These are defined in declaration sections. There is one global declaration section where channels, constants, user data types etc. are specified. Each automaton template has own declaration section, where local clocks, variables and functions are specified. And finally, there is a system declaration section, where global variables are declared and automata are created using templates.

Uppaal's query language for requirement specification is based on CTL (Computational Tree Logic, [5]). We do not present any formulas in this paper hence details about query laguage are omitted due to space restrictions.

The simulation and formal verification are possible in Uppaal. The simulation can be random or user assisted. It is more suitable for the user of the tool to see if a model is behaving like he want and like it corresponds to the real system. Formal verification should confirm that the system has desired properties expressed using the query language. There are many options and settings for verification algorithm in Uppaal. For example we can change representation of reachable states in memory or the order of search in the state space (breadth first, depth first, random depth first search). Some of the options lead to less memory consumption, some of them speed up the verification. But improvement in one of these two characteristic leads to a degradation of the other usually. For more exact definitions of modeling and query languages and verification possibilities of Uppaal see [3].

3 General comments and assumptions

In real-time database systems we consider usually transaction processing. Each transaction incoming to a system is assigned a priority. Resources are than apportioned according to priorities to transactions that are processed concurrently. The number of concurrently processed transactions in system is usually bounded – it is controlled by overload management policy. Incoming transactions have usually a deadline. This can be hard (transaction exceeding deadline becomes nearly useless and can be aborted for the sake of other transactions meeting their deadlines) or soft (the value of transaction exceeding deadline decreases, the priority can be lowered and transaction is processed in the time when there are not any transaction possibly meeting deadlines). In this paper we consider hard deadlines.

Scheduling and computation time assignment is strongly connected with priorities. Hence we will discuss this also in the following sections. Other aspects as, e.g., concurrency control will be omitted in order that the suggested models are still manageable by Uppaal.

There are many possibilities how to model transaction arrival. For example, there can be special automaton serving as generator of transactions. The models described in this paper are designed for comparison of two different algorithms. Hence we have decided to define incoming transactions statically as an array `inc_trans` which elements are structures of `release_time` (representing incoming time of transaction since beginning), `deadline_time`(deadline since beginning), `operations` (number of database operations), `received_time` (initially equals zero, it represents computation time already used by this transaction). For simplicity we do not consider exact database records and we even consider that all database operations need the same computational time given by constant `OP_TIME`. Both model can be simulated over this array to compare them (number of aborted transactions etc.) We can also automatically check queries expressed as formulae of temporal logic and become some other useful informations about modeled algorithms.

4 First Come First Serve

This policy assigns the highest priority to the transaction with the earliest release time. Often, release time equals arrival time. This algorithm is not very suitable for real-time database systems because it does not make use of deadline information. It can give more computation time to older transaction instead a newer transaction with more urgent deadline.

We consider that all computation time is assigned to the oldest transaction in a system until it finishes or reaches its deadline. Hence we will need just one copy of automaton depicted on Figure 1 atop. This automaton represents successively all transaction processed by a modeled system.

The state `Inactive` represents situation when there is not any processed transaction. Constant `TRANSACTIONS` contains the overall number of transactions defined in the input array, variable `act_trans` counts processed transactions. Clock variable `time` represents time from the beginning while clock variable `op_time` measures the time of performance of one database operation. If actual transaction ends successfully (number of performed operations `op_done` reaches the number of operations specified for this transaction), the automaton gets through the state `Done` to `Inactive` and it is prepared for representation of next transaction. If the transaction reaches its deadline before successful finish, the abort is modeled by the state `Abort`, it is counted and the automaton goes to the state `Inactive` once again.

The state `Waiting` is intended for the situation when there are more transactions in the input sequence but release time (representing time of arrival in the real system) is not passed yet. If all transactions from the input sequence are processed, automaton goes to the state `End` and a run is deadlocked. It is the only possible deadlock situation of this model (this has been checked using verification possibility of Uppaal).

5 Earliest Deadline

Earliest Deadline is algorithm which gives the highest priority to transaction with the earliest deadline. A disadvantage of this policy is that it can give high priority and hence a big amount of resources to a transaction which is about to miss its deadline anyway.

Assigned priorities can be used in several different ways for distribution of resources. One way is that a transaction with the highest priority gets all resources until it finishes or exceeds its deadline and it is aborted. To show some other general modeling possibilities, we have chosen some other way. We consider several concurrently processed transactions and scheduler distributes computation time between all of them. Transactions with higher priority get more time but no transaction is skipped. All automata representing concurrent transactions are instances of the same template depicted on Figure 1 down. The number of those instances can be set by a constant PAR_TRANS.

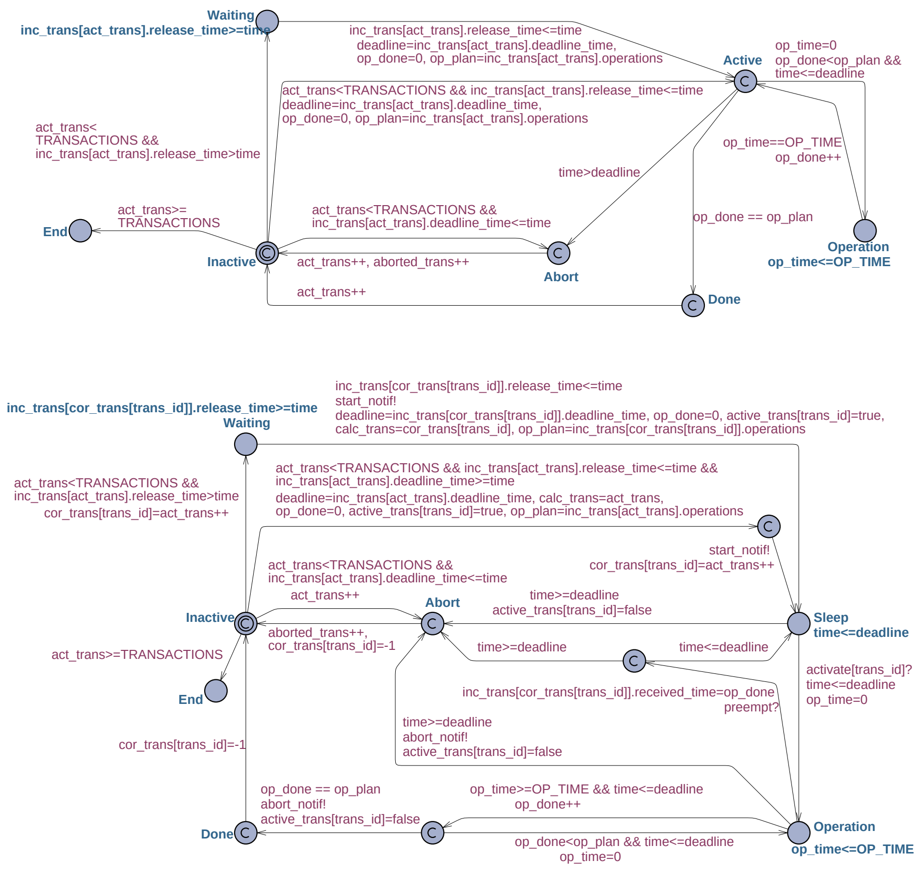

Waiting
inc_trans[act_trans].release_time>=time
inc_trans[act_trans].release_time<=time
deadline=inc_trans[act_trans].deadline_time,
op_done=0, op_plan=inc_trans[act_trans].operations
Active
op_time=0
op_done<op_plan &&
time<=deadline
act_trans<TRANSACTIONS && inc_trans[act_trans].release_time<=time
deadline=inc_trans[act_trans].deadline_time,
op_done=0, op_plan=inc_trans[act_trans].operations
act_trans<
TRANSACTIONS &&
inc_trans[act_trans].release_time>time
op_time==OP_TIME
op_done++
time>deadline
op_done == op_plan
act_trans>=
TRANSACTIONS
End
act_trans<TRANSACTIONS &&
inc_trans[act_trans].deadline_time<=time
Inactive
act_trans++, aborted_trans++
Abort
Operation
op_time<=OP_TIME
act_trans++
Done
inc_trans[cor_trans[trans_id]].release_time<=time
start_notif!
deadline=inc_trans[cor_trans[trans_id]].deadline_time, op_done=0, active_trans[trans_id]=true,
calc_trans=cor_trans[trans_id], op_plan=inc_trans[cor_trans[trans_id]].operations
inc_trans[cor_trans[trans_id]].release_time>=time
Waiting
act_trans<TRANSACTIONS && inc_trans[act_trans].release_time<=time &&
inc_trans[act_trans].deadline_time>=time
act_trans<TRANSACTIONS &&
inc_trans[act_trans].release_time>time
cor_trans[trans_id]=act_trans++
deadline=inc_trans[act_trans].deadline_time, calc_trans=act_trans,
op_done=0, active_trans[trans_id]=true, op_plan=inc_trans[act_trans].operations
start_notif!
cor_trans[trans_id]=act_trans++
act_trans<TRANSACTIONS &&
inc_trans[act_trans].deadline_time<=time
act_trans++
Abort
time>=deadline
active_trans[trans_id]=false
Inactive
Sleep
time<=deadline
aborted_trans++,
cor_trans[trans_id]=-1
time>=deadline
time<=deadline
act_trans>=TRANSACTIONS
activate[trans_id]?
time<=deadline
op_time=0
End
inc_trans[cor_trans[trans_id]].received_time=op_done
preempt?
time>=deadline
abort_notif!
active_trans[trans_id]=false
cor_trans[trans_id]=-1
op_done == op_plan
abort_notif!
active_trans[trans_id]=false
op_time>=OP_TIME && time<=deadline
op_done++
Done
Operation
op_done<op_plan && time<=deadline
op_time=0
op_time<=OP_TIME

Fig. 1. Transaction automata for FCFS algorithm (atop) and Earliest Deadline algorithm (down)

Distribution of computation time is controlled by Scheduler Automaton depicted on Figure 2 atop. Priorities to transactions are assigned using Priority Assignment Automaton depicted on Figure 2 down.

The basic behavior of Transaction automata is the same as in the case of First Come First Serve algorithm. The main modification is the state Sleep and its adjacent edges. It represents the situation when this transaction is processed but actually has not assigned resources. An indication of assigned resources is received from Scheduler automaton through the channel activate[trans_id]. trans_id is unique identifier of each Transaction automaton and activate is

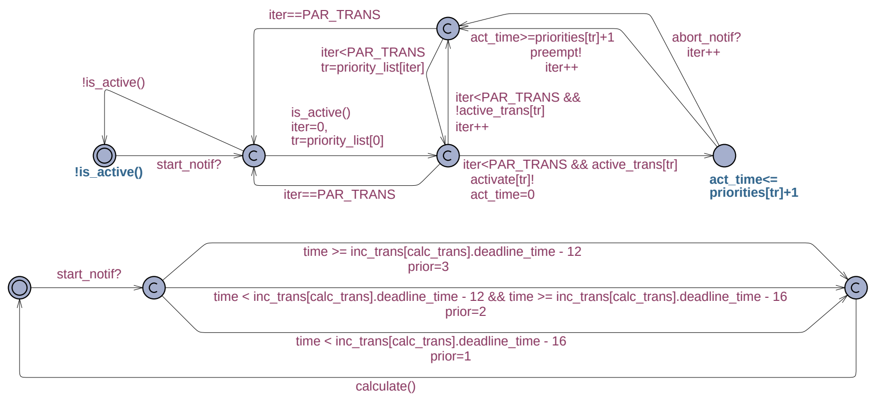


Fig. 2. Earliest Deadline algorithm - Scheduler automaton (atop) and Priority Assignment automaton (down)

array of channels. Taking the resources away is announced through the channel `preempt`. There are two more channels – `abort_notif` informs Scheduler automaton that this transaction is aborted and resources are free and `start_notif` is broadcast channel that informs Scheduler about active transaction and simultaneously asks Priority Assignment to compute priority for new transaction (identification of this transaction is shared using global variable `calc_trans`).

There are two axillary arrays. `active_trans` contains for each Transaction automaton a flag if it actually represents some transaction, `cor_trans` contains for each Transaction automaton identification of actually represented transaction from input array.

Scheduler automaton waits in the initial state until it is notified about new transaction. Then it takes iteratively identifications of Transaction automata representing transaction according the priority from the array `priority_list` which is maintained sorted by Priority Assignment automaton. Each transaction automaton is activated for the time corresponding to its priority (this can be chosen in different ways, in this paper it is directly priority, just for technical reasons increased by one). After all active Transaction automata take a turn, the whole process is repeated again from the automaton representing transaction with the highest priority. Function `is_active` is axillary, it returns true if there is at least one `true` in the array `active_trans`.

Priority Assignment automaton assigns priorities $1, 2$ or 3. The highest priority goes to transactions which have at most 12 time units until deadline, priority 2 to transactions with 12 to 16 time units before deadline and priority 1 to all other. Those values were chosen without any real meaning. It should be clear how to add more values of priority and change intervals for them. Function `calculate()` sorts identifications of Transaction automata in the array `priority_list` according to priorities of transactions they represent.

6 Conclusion

In the previous sections, several timed automata were shown. They form models of two variants of algorithms for priority assignment used in (real-time) database management systems. Of course, this were not the only possible models. The purpose was to show that some important aspects of the real-time database system, such as a priority assignment, can be modeled using such a relatively simple model as nets of timed automata are. The models can be extended in many different ways to capture more behavior of those policies and thus allow many properties to be described as a formula in the logic of Uppaal and then checked using its verification algorithms. Some properties even can not be expressed using Uppaal's modification of CTL. Automata can be modified to bypass this imperfection, but it can demand a special modification for each query and it also can increase reachable state space. Another possible solution to this problem is to try some other verification tool with other query language which can be our future work.

References

1. Abbott, R. K., Garcia-Molina, H.: Scheduling real-time transactions: a performance evaluation. ACM Transactions on Database Systems (TODS), Volume 17 , Issue 3, pages 513 – 560, ACM, September 1992.
2. Alur, R., Dill, D.L.: Automata for modeling real-time systems. Proc. of Int. Colloquium on Algorithms, Languages, and Programming, volume 443 of LNCS, pages 322-335, 1990.
3. Behrmann, G., David, A., Larsen, K. G.: A Tutorial on Uppaal. Available on-line at http://www.it.uu.se/research/group/darts/papers/texts/new-tutorial.pdf (March 15, 2010)
4. Berard, B., Bidoit, M., Petit, A., Laroussinie, F., Petrucci, L., Schnoebelen, P.: Systems and Software Verification, Model-Checking Techniques and Tools. ISBN 978-3540415237, Springer, 2001.
5. Henzinger, T.A.: Symbolic model checking for real-time systems. Information and computation, 111:193-244, 1994.
6. Kao, B., Garcia-Molina, H.: An Overview of Real-Time Database Systems. Advances in Real-Time Systems, pages 463-486, Prentice-Hall, Inc., 1995.
7. Kot, M.: Modeling selected real-time database concurrency control protocols in Uppaal. Innovations in Systems and Software Engineering, Volume 5, Number 2, pages 129-138, ISSN 1614-5046 (Print), ISSN 1614-5054 (Online), Springer, June 2009.
8. Kot, M.: Modeling Real-Time Database Concurrency Control Protocol Two-Phase-Locking in Uppaal. Proceedings of the International Multiconference on Computer Science and Information Technology, Volume 3, pages 673-678, ISBN 978-83-60810-14-9, ISSN 1896-7094, IEEE Computer Society Press, 2008.
9. McMillan, K. L.: Symbolic Model Checking. ISBN 978-0792393801, Springer, 1993.
10. Nyström, D., Nolin, M., Tesanovic, A., Norström, Ch., Hansson, J.: Pessimistic Concurrency-Control and Versioning to Support Database Pointers in Real-Time Databases. Proc. of the 16^{th} Euromicro Conference on Real-Time Systems, pages 261-270, IEEE Computer Society, 2004.
11. ParaDiSe (Parallel & Distributed Systems Laboratory): Yahoda verification tools database. Available on-line at http://anna.fi.muni.cz/yahoda/ (March 15, 2010)

Testing Quasigroup Identities using Product of Sequence[⋆]

Eliška Ochodková[1], Jiří Dvorský[1], Václav Snášel[1] and Ajith Abraham[2]

[1] Department of Computer Science
FEECS, VŠB – Technical University of Ostrava
17. listopadu 15, 708 33 Ostrava – Poruba, Czech Republic
{eliska.ochodkova, jiri.dvorsky, vaclav.snasel,}@vsb.cz

[2] Center of Excellence for Quantifiable,
Quality of Service,
Norwegian University of Science and Technology
O.S. Bragstads plass 2E,
N-7491 Trondheim, Norway
ajith.abraham@ieee.org

Abstract. Non-associative quasigroups are well known combinatorial designs with many different applications. Many cryptographic algorithms based on quasigroups primitives have been published. There are several classifications of quasigroups based on their algebraic properties. In this paper we propose a new classification of quasigroups based upon strings (product elements) obtained by a product of a sequence. It is shown in this paper that the more various results of the product elements, the less associative quasigroup.

1 Introduction

Almost all known constructions of cryptographic algorithms have made use of associative algebraic structures such as groups and fields. There is a possibility to use non-associative quasigroups [7] , well known combinatorial designs with a lot of theoretical results concerning them, too. Many cryptographic algorithms based on quasigroups primitives have been published. Proposed cryptographic algorithms are used for ciphering [15], for constructing pseudorandom generators [9], hash functions [12], for zero knowledge protocols [2], etc. Majority of published algorithms can be seen as rather simple experimental algorithms. As a representative of the ambitious proposals include the stream cipher Edon80 [5] published as an eSTREAM[3] candidate, and the NIST's SHA-3[4] competition candidate, hash function Edon$\mathcal{R}$ [4].

If a quasigroup is a base of some cryptographic primitive, it is necessary to examine whether its algebraic properties, structure or other features possess a

[3] http://www.ecrypt.eu.org/stream/
[4] http://csrc.nist.gov/groups/ST/hash/sha-3/

[⋆] This paper was partially supported by GACR 205/09/1079 grant.

security risk to the whole cryptographic algorithm. From all existing quasigroups of a given order we have to select those, which do not have various identities (as associativity is) and in which various identities appears rarely, or rather not at all. Properties of small quasigroups (e.g. of order 4), represented as a look-up table only, may be examined by the exhaustive search. But examination of identities of the quasigroups of a large order, e.g. 2^{16}, may not be easy.

Testing of all possible identities at once may be expensive, both in terms of time and in terms of space. Therefore we have focused on associativity only. If associativity holds, then for each element $a, b, c \in Q : a \circ (b \circ c) = (a \circ b) \circ c$. The situation differs when we work with non-group (i.e. non-associative) structure: $a \circ (b \circ c) \neq (a \circ b) \circ c$. We have made experiments with powers a^k of all elements $a \in Q$, where $k = 2, 3, \dots, n, n = |Q|$, obtained by a product of a sequence. Obtained results were evaluated and compared to the number of associative triples identified for each quasigroup used in experiments. Tested set of quasigroups was the subset of all distinct quasigroups of order 8. For better representation of the results, we have used their visualization.

The paper is organized as follows. Motivation of our work is introduced in Section 2, some necessary concepts are given here too. Concept of a product of sequence, experiments and their results are described in Section 3. Finally, Section 4 comprise conclusion and some ideas of future works.

2 Preliminaries

2.1 Basic Concepts

Definition 1. *Let $A = \{a_1, a_2, \dots, a_n\}$ be a finite alphabet, $n \times n$* Latin square *L of order n is a matrix with entries $l_{ij} \in A$, $i, j = 1, 2, \dots, n$, such that each row and each column consists of different elements of A.*

The numbers of all LSs of order ≤ 11 are known [14]. Number of distinct Latin squares[5] of a given order grows exceedingly quickly with the order. Latin squares are equivalent to quasigroups. The multiplication table of a quasigroup of order n is a Latin square of order n, and conversely every Latin square of order n is the multiplication table of a quasigroup of order n [3].

Definition 2. *A* quasigroup *is a pair $(Q, \circ)$, where $\circ$ is a binary operation on (finite) set Q such that for all not necessarily distinct $a, b \in Q$, the equations $a \circ x = b$ and $y \circ a = b$. have unique solutions. We say that quasigroup $(Q, \circ)$ is of order n if $|Q| = n$.*

In general, the operation $\circ$ is neither a commutative nor an associative operation. Every quasigroup satisfying the associative law has an identity element and is, hence, a group. There is, for example, 576 distinct quasigroups of order 4, but only 16 are associative. So non-associative quasigroups dominate heavily.

[5] We abbreviate 'Latin square' to LS.

Isotopism. Various methods of generating a practically unlimited number of quasigroups of a (theoretically) arbitrary order are known and shown in various publications. One common way of creating quasigroups is through isotopism [3].

Definition 3. *Let $(Q_1, \cdot)$ and $(Q_2, \circ)$ be two quasigroups with $|Q_1| = |Q_2|$. An ordered triple (α, β, γ) of one-to-one mappings α, β, γ of the set Q_1 onto the set Q_2 is called an* isotopism *of Q_1 upon Q_2 if $\alpha(x) \circ \beta(y) = \gamma(x \cdot y)$ for all $x, y \in Q_1$.*

One can prove that the set of all isotopisms of a quasigroup of order n forms a group of order $(n!)^3$. It should be noted that the mapping γ permutes the elements in the table of operations in a quasigroup Q_1, while α and β operate on the elements of the row and column borders of this table, respectively.

2.2 Motivation

Design of many of the existing algorithms is based on *quasigroup string transformations* [7, 11]. The following concepts are taken from [7].

Consider an alphabet (i.e. a finite set) Q, and denote by Q^+ the set of all nonempty words (i.e. finite strings) formed by the elements of Q. Let $(Q, \circ)$ is a quasigroup. Let $q = q_1 q_2 \ldots q_n \in Q^+, q_i \in Q$ and $l \in Q$ is a fixed element called leader. For each $l \in Q$ we define two functions $e_{l_\circ}$ and $d_{l_\circ}$: $Q^+ \to Q^+$ as follows:

$$e_{l_\circ}(q) = b_1 b_2 \ldots b_n \iff b_1 = l \circ q_1,\ b_2 = b_1 \circ q_2,\ \ldots,\ b_n = b_{n-1} \circ q_n \tag{1}$$

i.e. $b_{i+1} = b_i \circ q_{i+1}$ for each $i = 0, 1, \ldots, n-1$, where $b_0 = l$, and

$$d_{l_\circ}(q) = c_1 c_2 \ldots c_n \iff c_1 = l \circ q_1,\ c_2 = q_1 \circ q_2,\ \ldots,\ c_n = q_{n-1} \circ q_n \tag{2}$$

i.e. $c_{i+1} = q_i \circ q_{i+1}$ for each $i = 0, 1, \ldots, n-1$, where $q_0 = l$.

The functions $e_{l_\circ}$ and $d_{l_\circ}$ are called $e-$ and *$d-$transformation* of Q^+ based on the operation $\circ$ with leader l. In general, several quasigroup operations on the set Q can be used for defining quasigroup transformations. Let, $\circ_1, \circ_2, \ldots, \circ_k$ be such a sequence of (not necessarily distinct) quasigroup transformations. We may also choose leaders $l_1, l_2, \ldots l_k \in Q$ (not necessarily distinct), and then the compositions $\bullet$ of mappings

$$E_k = E_{l_1 l_2 \ldots l_k} = e_{l_1} \bullet e_{l_2} \bullet \ldots \bullet e_{l_k} \tag{3}$$

and

$$D_k = D_{l_1 l_2 \ldots l_k} = d_{l_1} \bullet d_{l_2} \bullet \ldots \bullet d_{l_k} \tag{4}$$

are said to be $E-$ and $D-$transformations of Q^+ respectively. In the last notation, we use e_{l_1} for the clarity, but formally we should use $e_{l_{1_{\circ_1}}}$.

The experiments with the length of a period of a string generated by e-transformations are mentioned in [6] and in [10]. Quasigroups are divided into two groups, to *linear* and *exponential* quasigroups. What algebraic properties must quasigroups of order 4 have to be linear resp. exponential? The quasigroups are of a small order (order 4), it is therefore impossible to say whether (besides identities) it is their structure, which affects the resulting period of the transformed string. Quasigroups of larger order are more convenient for analogical tests described in Sec. 3.

3 Experiment with Product of Sequence

Let $\circ$ be the binary operation. Consider the finite sequence A of elements $a_1, \ldots, a_n$, $a_i \in A, i = 1, 2, \ldots, n, n \geq 2$. What does mean a product of this sequence? Clearly, for $n = 2$ we have $a_1 \circ a_2$, by juxtaposition $a_1 a_2$. For $n = 3$ a product of the sequence a_1, a_2, a_3 is defined as a set consisting of product elements $a_1(a_2 a_3)$ and $(a_1 a_2)a_3$. The product is denoted as $\{a_1 a_2 a_3\}$ and symbol $a_1 a_2 a_3$ means any product element. Generally, we can define a product of a sequence of n elements of the set A as follows [1].

Definition 4. *The product of a sequence $a_1, a_2, \ldots, a_n$ of elements $a_i \in A, i = 1, 2, \ldots, n$ is the set $\{a_1 a_2 \ldots a_n\}$ defined by:*

– *for $n = 2$ the set $\{a_1 a_2\}$ consist of only one element $a_1 a_2$,*
– *for $n \geq 2$ the set $\{a_1 a_2 \ldots a_n\}$ is defined as*

$$\{a_1 a_2 \ldots a_n\} = \{a_1\}\{a_2 \ldots a_n\} \cup \{a_1 a_2\}\{a_3 \ldots a_n\} \cup \ldots \cup \{a_1 \ldots a_{n-1}\} \cup \{a_n\}.$$

The n elements can be joined, without changing their order, in $\frac{(2n-2)!}{n!(n-1)!}$ ways. For e.g. $n = 1, 2 \ldots, 10$ we obtain $1, 1, 2, 5, 14, 42,\ 132, 429,\ 1430, 4862$ ways of joining n elements. These numbers are called *Catalan numbers* [8]. The mth Catalan number, for $m \geq 0$ is given by:

$$C_m = \frac{1}{m+1}\binom{2m}{m} = \frac{(2m-2)!}{m!(m-1)!}.$$

If the operation $\circ$ on the set A does not hold an associativity law, we can generally obtain distinct values $a_1 a_2 \ldots a_n$ (not one common value) for all C_m ($m = n - 1$, because Catalan numbers are numbered from 0) possible product elements of the product set $\{a_1 a_2 \ldots a_n\}$ of the sequence $a_1, a_2, \ldots, a_n$.

3.1 Experiment

We have tested product $\{q_1 q_2 \ldots q_k\}$ of the sequence $q_1, q_2, \ldots, q_k$, where all $q_i \in Q, i = 1, 2, \ldots, k$ are equal. So, if all elements are equal, each element is denoted as a and we will compute a product $\{\underbrace{aa \ldots a}_{k}\}$ of the sequence $\underbrace{a, a, \ldots, a}_{k}$. This product consists of all C_{k-1} product elements. Questions is, how many distinct values $q_1 q_2 \ldots q_k = \underbrace{aa \ldots a}_{k} = a^k$ for all $a \in Q$ we obtain. In the ideal case we can obtain all possible values as a result; the set of possible values has only max. n values from Q (of order n) for all powers a^k.

Better information about the identities in the given quasigroup gain from the evaluation of particular product elements by the e-transformation defined in Eq. (2). Therefore all strings $b_1 \ldots b_8$, see Fig. 1, obtained during evaluation of product elements of a^k (for $k = 8, a^8 = b_8$, Fig. 1) were stored. The experiment:

– Generate a quasigroups Q of order n.

- For each element $a \in Q$ and for each $k, 2 \leq k \leq n$ create the product $\{aa \ldots a\}$ of the sequence $a, a, \ldots, a$, evaluate all C_{k-1} product elements a^k.
- Store strings $b_1 \ldots b_k$ and compute the number of their occurrence during evaluation of all product elements a^k.

$$\begin{tikzcd}
 & a \arrow[d] & a \arrow[d] & \cdots & a \arrow[d] & a \arrow[d] \\
a \arrow[ur] & b_1 \arrow[ur] & b_2 \arrow[ur] & \cdots \arrow[ur] & b_7 \arrow[ur] & b_8 = a^8
\end{tikzcd}$$

Fig. 1. e-transformation used for valuing the product elements $a^k, k = 8$

3.2 Quasigroups used in tests

Quasigroups were represented by corresponding Latin squares. We decided to use a subset of quasigroups of order 8. We have tested:

- all $n! = 40320$ distinct quasigroups isotopic to additive group $(\mathbb{Z}_8, +)$ when only permutation α was not an identity permutation,
- a set of one million randomly generated quasigroups,
- a set of special quasigroups that consist of e.g. additive group $(\mathbb{Z}_8, +)$, of six well described quasigroups published in [13], etc.

3.3 Ideal results

Results are shown on the highest (8th) power of element a. There are $C_7 = 429$ distinct ways how to obtain it.

- Ideally, for each $a \in Q$, i.e. for each $a = 0, 1, \ldots, 7$, we obtain all 8 possible values of $a^8 \in Q$.
- For each $a \in Q$ we obtain all 429 distinct strings $b_1 \ldots b_8$.
- Finally, for each quasigroup $(Q, \circ)$ we ideally obtain all together $429 \times 8 = 3432$ distinct strings $b_1 \ldots b_8$ for all $a \in Q$.

3.4 Experimental results

Results of experiments are shown on the set of five chosen quasigroups represented by their corresponding LSs. The first quasigroup is randomly generated quasigroup No. 24 represented by L_{24}. The second quasigroup, obtained by non-affine isotopy [13], is represented by corresponding LS L_{103}. The third quasigroup is quasigroup 104 obtained by complete mapping [13] and represented by LS L_{104}. The fourth quasigroup is quasigroup No. 106, from [13], is represented by LS L_{106}. The last quasigroup $(Q_1, \circ)$ with No. 107 is represented by corresponding LS L_{107} (this quasigroup is the additive group $(\mathbb{Z}_8, +)$).

Numbers of distinct values a^8 for each $a \in Q$ for five chosen quasigroups are shown in Table 1. Only quasigroups No. 24 and 104 have ideal results. Conversely, quasigroup's No. 107 results are always the same; a^8 is always 0.

Results of the process evaluating the strings $b_1 \dots b_8$: the best results have quasigroups No. 24 and 104. Number of all distinct strings is higher comparing the remaining three quasigroups. This fact is evident from Table 2 (sums of distinct strings for each quasigroup and all $a \in Q$ are shown). The higher number of associative triples, the lower the sum of all strings. Results were also visualized, Sec. 3.5. The greater number of subsquares of different brightness in the image corresponds with the greater number of distinct strings $b_1 \dots b_8$ for each a^8, see Figs. 2 and 3.

Table 1. Number of obtained distinct values of a^8 for each $a \in Q$

	L_{24}	L_{103}	L_{104}	L_{106}	L_{107}
0^8	8	8	8	8	$1(0^8 = 0)$
1^8	8	8	8	1	$1(1^8 = 0)$
2^8	8	8	8	2	$1(2^8 = 0)$
3^8	8	$1(3^8 = 3)$	8	3	$1(3^8 = 0)$
4^8	8	8	8	4	$1(4^8 = 0)$
5^8	8	8	8	2	$1(5^8 = 0)$
6^8	8	8	8	2	$1(6^8 = 0)$
7^8	8	8	8	2	$1(7^8 = 0)$

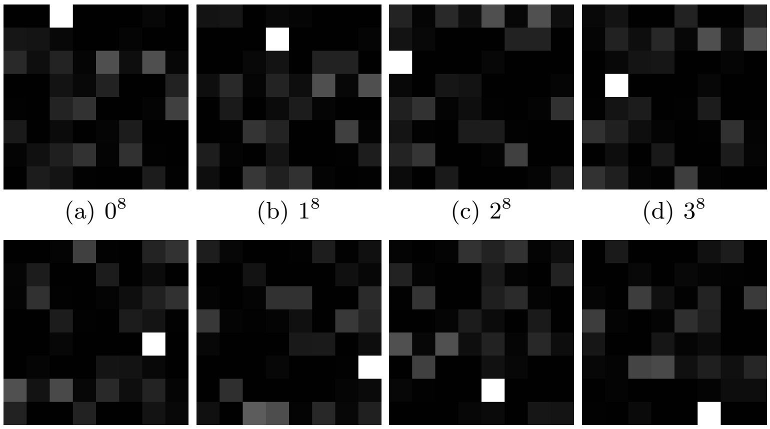
Table 2. Number of obtained strings $b_1, \dots, b_8$ for all $a \in Q$

	L_{24}	L_{103}	L_{104}	L_{106}	L_{107}
number of strings	2426	2019	2666	664	307
number of AT	72	70	60	304	512

3.5 Visualization of strings $b_1, \dots, b_n$ valuation

We have focused only on the 8th power of quasigroups elements. For each quasigroup $(Q, \circ)$ and for each $a^8, a \in Q$, we have generated 512×512 pixels images where each subsquare (64×64 pixels) represents one element l_{ij} of tested quasigroup represented by corresponding Latin square $L, l_{ij} \in L$. The more visits of particular element, the brighter subsquare. The brightness of the subsquares is calculated relatively to the number $C_7 \times ir = 429 \times 6 = 2574$, where $ir = 6$

is number of strings from a^2 to a^8 when computing a^8. The greater the sum of distinct strings $b_1 \dots b_n$, the greater the number of subsquares of different brightness in the image.


(a) 0^8 (b) 1^8 (c) 2^8 (d) 3^8

(e) 4^8 (f) 5^8 (g) 6^8 (h) 7^8

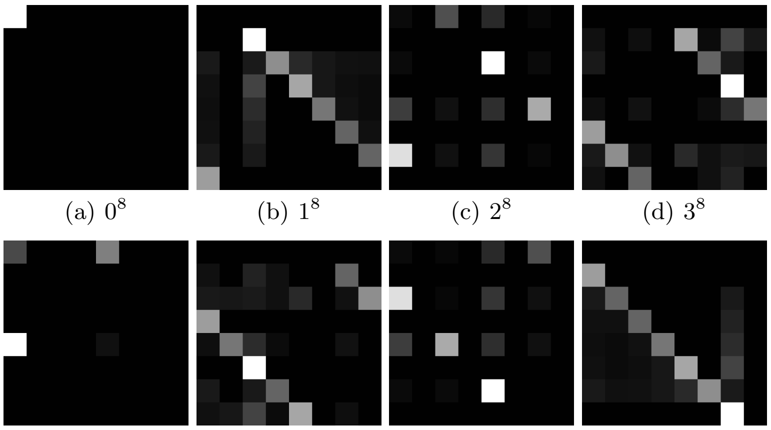
Fig. 2. Quasigroup No. 104

4 Conclusion

Our goal is to find a new way of testing the properties of large quasigroups and to explore the interpretation of experimental results. We have reported a new classification of quasigroups based upon strings (product elements) obtained by a product of a sequence. As is shown, the more various results of the product elements, the less associative quasigroup. More precisely, values of all possible product elements from the product set of a sequence of elements from a given quasigroup were examined and relationships between experiment results and associativity of tested quasigroup have been tested. Testing of quasigroup's identities through the product of a sequence is an appropriate method with good results. Experiments will be repeated with quasigroups of larger order. Several consecutive applications of a quasigroup transformations on the sequences will be tested, too.

References

1. O. Borůvka. *Foundations of the theory of groupoids and groups.* Wiley, 1976.
2. J. Dénes, and T. Dénes. Non-associative algebraic system in cryptology. Protection against "meet in the middle" attack. *Q. and Related Systems* 8 (2001): 7–14.



(a) 0^8 (b) 1^8 (c) 2^8 (d) 3^8

(e) 4^8 (f) 5^8 (g) 6^8 (h) 7^8

Fig. 3. Quasigroup No. 107

3. J. Dénes, and A. Keedwell. *Latin Squares and their Applications.* New York: Akadémiai Kiadó, Budapest, Academic Press, 1974.
4. D. Gligoroski, et al. EdonR cryptographic hash function. NIST's SHA-3 hash function competition, 2008, http://csrc.nist.gov/groups/ST/hash/sha-3/index.html
5. D. Gligoroski, S. Markovski, L. Kocarev, and J. Svein. The Stream Cipher Edon80. The eSTREAM Finalists, *LNCS* 4986 (2008): 152–169.
6. D. Gligoroski. One-Way Functions and One-Way Permutations Based on Quasigroup String Transformations. *Cryptology ePrint Archive.* Report 2005/352.
7. D. Gligoroski, and S. Markovski. Cryptographic potentials of quasigroup transformations. Talk at EIDMA Cryptography Working Group, Utrecht, 2003.
8. P. Hilton, and J. Pedersen. Catalan Numbers, Their Generalization, and Their Uses. *Journal The Mathematical Intelligencer,* 13, no. 2 (1991): 64–75.
9. C. Kościelny. NLPN Sequences over GF(q). *Quasigroups an Related Systems* 4 (1997): 89–102.
10. S. Markovski, D. Gligoroski, and J. Markovski. Classification of quasigroups by random walk on torus. *J. of Appl. Math. and Comp.* 19, no. 1-2 (2005): 57–75.
11. S. Markovski, D. Gligoroski, and L. Kocarev. Unbiased Random Sequences from Quasigroup String Transformations. in *12th International Workshop FSE,* Paris, LNCS 3557 (2005): 163.
12. S. Markovski, D. Gligoroski, and V. Bakeva. Quasigroup and Hash Functions. *Disc. Math. and Appl.*, In Proceedings of the 6th ICDMA, Bansko, 2001.
13. K. A. Meyer. A new message authentication code based on the non-associativity of quasigroups. Ph.D Thesis, 2006, http://orion.math.iastate.edu/dept/thesisarchive/PHD/KMeyerPhDSp06.pdf
14. B. D. McKay, and I. M. Wanless. On the Number of Latin Squares. *Journal Annals of Combinatorics* 9, no. 3 (2005): 335–344.
15. E. Ochodková, and V. Snášel. Cryptographic Algorithms with Uniform Statistics. In *NATO Regional Conference on Military Communications and Informations Systems* Zegrze, Poland: 165–172, 2001.
16. K. Toyoda. On axioms of linear functions. *P*roc. Imp. Acad. Tokyo, 17 (1941): 221–227.

Database Trends and Directions: Current Challenges and Opportunities

George Feuerlicht[1,2]

[1] Department of Information Technology, University of Economics, Prague
W. Churchill Sq. 4, Prague, Czech Republic

[2] Faculty of Engineering and Information Technology University of Technology, Sydney
P.O. Box 123 Broadway, Sydney, NSW 2007, Australia
`jiri@it.uts.edu.au`

Abstract. Database management has undergone more than four decades of evolution producing vast range of research and extensive array of technology solutions. The database research community and software industry has responded to numerous challenges resulting from changes in user requirements and opportunities presented by hardware advances. The relational database approach as represented by SQL databases has been particularly successful and one of the most durable paradigms in computing. Most recent database challenges include internet-scale databases – databases that manage hundreds of millions of users and cloud databases that use novel techniques for managing massive amounts of data. In this paper we review the evolution of database management systems over the last four decades and then focus on the most recent database developments discussing research and implementation challenges presented by modern database applications.

Keywords: Relational Databases, Object-Relational Databases, NoSQL Databases

1 Introduction

Databases, in particular relational databases, are a ubiquitous part of today's computing environment. Database management systems support a wide variety of applications, from business to scientific and more recently various types of internet and electronic commerce applications. Database management systems (DBMS) are a core technology in most organizations today and run mission-critical applications that banks, hospitals, airlines, and most other types of organizations rely on for their day to day operation. Over the last three decades relational DBMS technology has proven to be highly adaptable and has evolved to accommodate new application requirements and the ever-increasing size and complexity of data. But, there are indications that some of the recently emerging data-intensive applications (e.g. internet searches) cannot be satisfactorily addressed using existing DBMS technology, and some experts argue that significant innovation is needed (a new database paradigm) to overcome the limitations of the current generation of database technology.

The combination of inexpensive and high capacity storage and the prevalence of digital devices (digital cameras, sound recorders, video recorders, mobile phones, RFID readers, and various types of sensors) is creating a deluge of digital information. According to a recent article in the Economist [1] the amount of data collected by various sensors, computers, and devices is growing at a compound annual rate of 60%. A 2008 study by International Data Corporation (IDC) predicted that over a thousand exabytes of digital data will be generated in 2010 [2]. Scientific applications in astronomy, earth sciences, etc. (e-science) tend to produce massive amounts of data; well-documented examples include the Large Hadron Collider at CERN [3] that generates 40 terabytes of data every second. Storing and analyzing such volumes of data represents an insurmountable challenge for the current generation of database technology. Another relatively recent development that may require a revision of current database paradigms are internet-scale applications (e.g. search engines, social networking applications, cloud computing services, etc.) that typically process petabytes of data, use thousands of servers, and serve millions of users that demand sub-second access to information. Companies like Google, Facebook, Amazon, and eBay manipulate petabytes of data every day. For example, Facebook handles 20 petabytes of data, managing 20 billion photographs in 4 different resolutions, growing by 2 billion photographs per month. The Facebook database is serving 600,000 photographs per second for a user base of 300 million active users [4]. Google manages vast amounts of semi-structured data: billions of URLs with associated internet content, crawl metadata, geographic objects (roads, satellite images, etc.), and hundreds of terabytes of satellite image data, with hundreds of millions of users and thousands of queries per second [5]. The scale and level of functionality required for such "big data" applications has not been anticipated by commercially available DBMSs, and almost invariably internet companies were forced to develop their own database solutions. But, even more traditional database applications manage increasingly large volumes of data; for example the retail chain WalMart handles more than one million transactions per hour, and manages databases with more than 2.5 petabytes of data.

It is estimated that structured data constitutes only about 5% of the total volume of generated data, with the rest of this "digital universe" in semi-structured or unstructured form, making it more difficult to manage and to extract meaningful information from it. This massive increase in the volume and complexity of data is challenging available database management techniques and technologies, forcing a re-evaluation of the direction of database research. Some fundamental questions arise, including what constitutes a database application. Can applications that search petabytes of unstructured data (e.g. Web pages) using thousands of servers working in parallel be classified as database applications?

In this paper we firstly review the past achievements of database research and technology solutions (section 2), and then discuss the research challenges and opportunities created by new types of database applications (section 3). The final sections (section 4) are our conclusions.

2 Evolution of Database Technology

While the origin of commercial database management systems can be traced to hierarchical and CODASYL (Conference on Data Systems Languages) databases of 1960s and 1970s it was the emergence of relational DBMS during the 1980s that started a revolution in data management. The simplicity and elegance of the relational model proposed by E.F. Codd in 1970 [6] resulted in unprecedented volume of research activity and the emergence of highly successful relational DBMS (RDBMS) implementations. Relational databases are a rare example of a theoretical model preceding and guiding the implementation of technologies. Codd is often credited with turning the previously *black art* of data management into an engineering discipline providing a blueprint for the design and implementation of databases and the foundation of modern database technology. The basic idea of the relational model is to represent data as two-dimensional tables with well-defined properties and to use of a high-level query language for data access. This remarkably simple set of ideas based on the underlying relational theory had a major impact on the development of database technology over the following two decades. Relational databases solved two major interrelated problems of the earlier database approaches. The first achievement was to de-couple the database from application programs by providing effective support for data independence. Second, and equally important achievement of the relational approach was to free database application developers from the burden of programming navigational access to database records by introducing a non-procedural query language.

A number of different relational languages were proposed following Codd's original description of the relational model, notably a language called QUEL (Ingres DBMS) developed at University of California at Berkeley, and IBM's Structured Query Language (SQL) developed at the IBM San Jose Research Laboratory. The next major milestone in the evolution of relational databases was the acceptance by ANSI (American National Standards Institute) of a subset of IBM's SQL as the first version of the standard relational database language - SQL86. Although SQL86 lacked many important features of the relational model as originally proposed by Codd, including key aspects of the model such as referential integrity and domains, it quickly became universally accepted as the database language for relational DBMS systems. The shortcomings in SQL86 were largely rectified in the subsequent releases of the SQL standard (SQL89, SQL92) and SQL has evolved from a relatively simple language into a comprehensive database language implemented in all significant RDBMS products today. Many of the enhancements incorporated into SQL over the last two decades were integral features of the relational model omitted from the earlier standard specifications, other features, such as triggers, role-based security, and stored procedures were retrofitted into the standard as a result of their widespread use in commercial products.

Given the computing environment of the 70s and early 80s, relational databases were initially used for relatively simple business applications running on large mainframe computers; data used in such traditional business applications (e.g. financial and banking) can be structured into tables and stored in a relational database with relative ease. The main concern of early RDBMS implementations was to ensure adequate performance, in particular for online transaction processing (OLTP) applications. Initially, relational DBMSs had inferior performance when compared with earlier DBMS approaches as SQL uses *expensive* join operations and relies on a query optimizer to determine how to access data records instead of using faster pointer-based navigational access implemented in hierarchical and CODASYL databases. For that reason the main use of relational DBMSs was initially confined to decision support applications that did not involve users waiting for query results online. However, as computer hardware became more powerful and optimization techniques improved, relational systems became the technology of choice in most application environments, including those with stringent response time requirements.

Relational DBMSs proved to be extraordinarily successful in taking advantage of new computing platforms, architectures and environments. The first significant demonstration of the adaptability of relational databases was the extension of the relational model to cover distributed database environments. The origin of distributed relational database was IBM's research project System R* (continuation of the Project R) which addressed distributed database issues including distributed query optimization, distributed transactions, and catalog management. Following on from the System R* database researchers solved most of the problems that concern running applications transparently across multiple databases. Most commercial RDBMSs incorporate a whole range of distributed database features, including reliable (two-phase commit) distributed transactions, optimized distributed queries, and advanced replication facilities. Similarly, relational DBMSs were among the first technologies to support applications with large number of users in distributed client/server environments. This was largely due to the non-procedural nature of SQL, which made it possible for database queries to be packaged and send over a computer network as messages from a client application to a database server. This type of client/server interaction later supplemented with remotely executed database stored procedures using RPC calls (Remote Procedure Calls) enabled the implementation of scalable client/server database applications.

Relational DBMSs were quick to take advantage of the new multiprocessor architectures and provide support for parallel execution of SQL queries. Query decomposition, necessary for parallel execution is made possible by the declarative nature of the SQL language enabling queries to be decomposed into well-defined sub-queries that run in parallel across multiple processors. Parallel SQL was implemented for shared memory, shared-disk, and shared-nothing parallel architectures with excellent performance and scalability. Both distributed and parallel databases benefit from the theoretical underpinning provided by the relational model. As a result of such developments relational databases became the fastest and most scalable commercially available DMBS systems.

2.1 Objects and Databases

RDBMs have shown remarkable ability to take advantage of new computing platforms and continuously improve functionality, performance and scalability to a point where relational databases became the dominant database technology in 1990s, supporting mission-critical environments with tens of thousands of users. However, by mid 90s it became quite clear that the simple data structures and a limited set of data types that characterize SQL92 relational DBMSs constitute a significant drawback when implementing new types of applications that use complex data. Modern database applications are characterized by four categories of requirements:

(1) need to store and manage large multimedia data objects – images, sound clips, videos, maps, etc.
(2) requirement for database data types to mirror application-level data types, including the ability for users to define their own data types as needed by specific applications
(3) representation of complex relationships, including composition and aggregation, e.g. multi-level component assemblies used in CAD (Computer-Aided Design) and similar applications
(4) need for seamless integration with object-oriented programming languages; with Java in particular

Such requirements are particularly evident in applications that use multimedia data, GIS (Geographical Information Systems), e-science and web applications. Web applications typically contain a whole range of multimedia data types such as textual information, images, video and audio clips, and fragments of program code. Many modern applications require specialized data types, for example GIS applications involve spatial data types (e.g. points, lines, polygons, etc.) and spatial operations (e.g. distance, area, etc.). The initial solution adopted in relational databases to accommodate non-traditional data (e.g. multimedia, GIS, etc.) was to allow the storage of large objects (LOBs) as columns in database tables. However, using this approach multimedia data is treated as unstructured large granularity objects – the data type of the object is not explicitly recognized by the database type system and only very limited processing of the object data is supported.

In addition to the need to store large and complex objects in the database, there is another important requirement that motivated the introduction of object support at the database level. Most modern applications are developed using object-oriented programming languages (i.e. Java, C++, C#) and close integration of the database language SQL with object-oriented programming languages reduces *impedance mismatch* (i.e. differences between the type systems, error handling, etc.) with corresponding improvements in programmer productivity. This requirement, while not new gained urgency with the emergence of Java as a *de facto* standard programming language for internet applications, making it imperative to ensure that Java objects can be easily mapped into database objects.

While there was a wide agreement within the database research community about the need to support objects at the database level, there was a considerable divergence of opinion about how this should be achieved. Two competing approaches emerged: the *revolutionary* approach, seeking to develop a completely new fully object-oriented database solution [7], and the *evolutionary* approach which took the path of adding object features to SQL. In early 90s a number of database management systems were developed ground-up as pure object DBMS (ODBMS) systems with the goal to address the limitations of relational databases by adopting a completely new database model with support for objects with unique identifiers, methods, inheritance, encapsulation, polymorphism and other features commonly associated with object systems. The basic idea was to build on top of object-oriented programming languages and provide persistence for application objects achieving homogeneous programming environment with close correspondence between application objects and objects stored in the database. This (revolutionary) approach popularized by the Object Database Management Group (ODMG) resulted in the proposal for a new database model and Object Query Language (OQL). As the commercial ODBMS products appeared on the market and attempted to capture market share from the established relational DBMSs, many regarded object-oriented databases as the next generation of database technology destined to supersede relational databases in much the same way as relational technology superseded earlier databases approaches. However, this radical attempt to break with the past has been largely unsuccessful as ODBMSs have not been able to match RDBMS technology in a number of important aspects, including reliability, scalability and level of standardization. Even more importantly, while popular in some niche application areas (e.g. CAD/CAM), object databases have not been able to address the wider requirements of mainstream corporate applications.

As a response to ODBMS enthusiasts a number of influential database researches formed a Committee for Advanced DBMS Function with the objective to define the requirements for the next generation database systems, and published the Third-Generation Database Systems Manifesto [8] as a blueprint for future database development. While recognizing the limitations of relational databases, this important effort argued that the next generation database systems should subsume the existing (second generation) DBMSs and preserve the benefits of relational databases, in particular non-procedural access and data independence. The essential point of difference from the advocates of object-oriented databases was the insistence on natural evolution from the existing relational DBMSs technology, and the implementation of object identity, abstract data types, inheritance, and other object features as relational database extensions.

The evolutionary approach resulted in a new breed of hybrid Object-Relational database technology. In retrospect, Object-Relational databases to a very large extent achieved the original objective of the Third-Generation Database Systems Manifesto, to preserve the benefits of relational database and at the same time to take advantage of object features. However, bringing object features into SQL did not turn out to be an easy task, and the evolutionary approach has struck numerous challenges and produced a number of changes in direction. At a superficial level there seems to be a good match between relations and objects, more specifically the concepts of relational rows and object instances. But, at closer inspection there are deep conflicts between the two models. For example, encapsulation, a key feature of object systems is difficult to reconcile with a database query language, as encapsulated data cannot be queried directly and requires access via methods, imposing unacceptable performance overheads. Various attempts at the unification of relations and objects using concepts such as ADTs (Abstract Data Types) have been proposed and discussed extensively by the ISO WG3 (Working Group 3), the working group responsible for database language standardization, but failed to gain the necessary wide support. After more than five years of intensive work by the WG3 Working Group on Database Languages some of the early ambitious attempts to incorporate object-orientation into the SQL standard were significantly scaled down. The resulting SQL:1999 standard is a very pragmatic solution, which addresses the main limitation of relational databases by enabling data type extensibility, providing the basis for a rich database type system [9]. The mainstream database vendors (Oracle, IBM, and others) have strongly endorsed the object-relational approach and actively participated in the development of SQL:1999, and most leading DBMS products support object-relational features of SQL:1999.

2.2 XML and Databases

Another challenge to the dominance of relational databases that echoed the efforts to incorporate object support into databases in the 90s arose approximately a decade later with the emergence of XML. XML became the *de facto* standard formatting language for semi-structured data and has been widely adopted in e-business applications as a standard data interchange language and a core standard for Web Services and related technologies. The availability of a standard XML query language XQuery [10], and a standard schema definition language XML Schema [11], and numerous other XML tools and languages (XPath, SAX, DOM, XQL, etc.) provided a basis for the implementation of XML DBMSs. This lead to the development of a number of research prototypes, e.g. Lore [12], XTABLES [13], SilkRoute [14], and some commercial products, e.g. Tamino [15].

Native XML Databases (NXD) that store XML documents in their native format and use XML query languages for retrieval were regarded by some as a new generation of DBMS technology destined to supersede relational DBMS. However, similar to ODBMS, NXD did not replace relational DBMS and remain a solution in niche application domains, mainly in document and content management applications. A detailed analysis of the benefits and limitations of NXD databases is available in [16].

As an alternative to the Native XML Database approach, ORDBMSs (e.g. Oracle) provide a repository functionality called XML native type to store XML documents in the database without any conversion, and support updates, queries, indexing, and views on this data type. Another option available in ORDBMS is to convert XML data into a relational or object-relational form (so called *shredding*) and store the resulting rows of data in corresponding typed tables. The SQL/XML specification [17] that is a part of the SQL:2003 standard defines the XML data type and provides mapping rules between XML schemas and SQL structures, as well as functions that support manipulation of XML data within SQL queries.

3 Database Research Directions

As per our discussion in the previous section (section 2), database research and associated standardization activities have successfully guided the development of database technology over the last four decades and SQL relational databases remain the dominant database technology today. This effort to innovate relational databases to address the needs of new applications is continuing today. Recent examples of database innovation include the development of streaming SQL technology that is used to process rapidly flowing data ("data in flight") minimizing latency in Web 2.0 applications [18], and database appliances that simplify DBMS deployment on cloud computing platforms [19]. It is also evident from the above discussion that the relational database approach has proven to be extraordinarily durable, and has adapted to new hardware architectures as well as new application requirements, successfully subsuming both object-oriented and XML paradigms. Database research has played an important role in solving key research problems and facilitating rapid technology transfer making DBMS technology one of the most successful efforts in computer science [20]. However, it is equally evident that database research is facing major new challenges due to *explosion of data*, novel usage scenarios, and a major shift in computing architectures. A recent meeting of leading database experts characterized the present situation as a "turning point in database research" and identified a number of trends that necessitate re-evaluation of research directions, and at the same time present new research opportunities. "The Claremont Report on Database Research" [21] identified the following trends:

(1) Big Data (applications that process very large volumes of data, e.g. Web search, e-science, etc)
(2) Data analysis as a profit center (increasing number of companies where the main business is data)
(3) Ubiquity of structured and unstructured data (mainly originating from various Web sources)
(4) Developer demands (as adoption of open source relational DBMS accelerates, developers demand more intuitive programming models)
(5) Architectural shifts in computing (emergence of cloud computing services brings about a fundamental change in software architecture towards parallel clusters of computers; shift away from increasing CPU clock speed to increasing the number of processor cores)

The report goes on to identify the following research opportunities:

(1) Re-design of architecture of database engines, to overcome the limitations of current relational databases (RDBMS provide poor price/performance for many popular applications, including text indexing, serving web pages, and media delivery)
(2) Declarative Programming for Emerging Platforms (support for data independence, declarative programming and cost-based optimization for new programming models, e.g. MapReduce)
(3) The Interplay of Structured and Unstructured Data (managing a rich collection of structured, semistructured and unstructured data, spread over many repositories in the enterprise and on the Web)
(4) Cloud Data Services (improving manageability of cloud databases, Federated cloud architectures, etc.)
(5) Mobile Applications and Virtual Worlds (manage massive amounts of diverse user created data, and provide real-time services)

The report notes that a number of additional research areas were not included as these are the subject of ongoing investigation, and includes a summary of past reports in the appendix. Setting agenda for database research is a challenging task, and forty years of database research and development illustrates both the successes and failures of such efforts. Database management is a very pragmatic field and many promising research ideas were discarded as they did not provide any practical benefits, or turned out not to be a database problem (e.g. deductive databases [22], expert databases [23], etc.).

4 Conclusions

The Claremont Report on Database Research made an interesting observation noting that the database research community has doubled in size over the last decade (as measured by the number of publications and number of database related conference sessions), but at the same time there was a perception that the quality of reviews (and consequently, the quality of publications) has been decreasing over time. Notwithstanding this massive research effort most significant recent innovations came out of research labs of various companies (e.g. Google, Facebook, etc.), who are facing urgent challenges of unprecedented size and complexity of data, and millions of users running many thousands of transactions per second. These developments have not been fully anticipated by the database research community and interestingly not even by the traditional database vendors whose products offerings were dwarfed by the scale and complexity of new application domains.

Recent rise of the NoSQL movement whose proponents regard the existing relational DBMSs as inefficient, complex and expensive, and favor open source non-relational solutions, demonstrates this point. For example, the MapReduce programming model developed by Google [24], and its open source clone Hadoop [25] initially used to simplify the construction of inverted indexes, has been applied to text processing and numerous other tasks that require parallel computation over a very large set of data. MapReduce is designed to automatically parallelize and execute a program on a large cluster of commodity machines (typically, tens of thousands of machines), managing data partitioning, task scheduling, inter-machine communication, and recovery from machine failures. The combination of Hadoop with Hypertable [26] (an open source version of Google BigTable [26]), enables the concurrent execution of programs on tens of thousands of machines processing petabytes of data on a daily basis [27].

These types of applications were traditionally the domain of parallel databases, and a number of commercial database machines (e.g. Teradata, Oracle Exadata, etc.) have been available for some time with proven performance characteristics for processing very large data volumes. The comparison of MapReduce and parallel databases has been the subject of a recent publication [28], concluding that "using MapReduce to perform tasks that are best suited for DBMSs yields less than satisfactory results", and that MapReduce resembles more Extract-Transform-Load (ETL) system rather than a DBMS, and therefore is a complimentary technology rather than competing with DBMS. But, these conclusions are being hotly disputed by MapReduce proponents who claim that the scalability benefits of this approach will eventually "relegate relational DBMS to the status of legacy technology". Database vendors are taking different approach to adopting the open source version of MapReduce (Hadoop), some implementing this technology in their products (e.g. Teradata, and IBM), while others (e.g. Microsoft) adopting a more cautious attitude [29].

Other vendors, notably Netezza have developed massively parallel database machines (Data Warehouse Appliances) that perform data filtering directly on the disk so that only the relevant portions of the data are propagated to the SQL database, gaining significant performance improvements over more traditional parallel database architectures. Additional functionality such as data analytics functions can also be implemented directly in hardware, achieving further performance gains [30]. Such approaches are using standard SQL database technology and are betting on further advances in computer hardware (larger and less expensive computer memory and more powerful CPUs as predicted by Moore's Law), and innovative, massively parallel database architectures to overcome the challenges of *big data*.

A key to understanding present and likely future database developments is a firm view of what constitutes the database paradigm, i.e. defining the scope of database research problems. Many of the recent challenges (in particular those faced by internet companies such as Google, Facebook, etc.), concern situations where three key elements that normally constitute a database environment are not present. While these applications are clearly data-intensive, there is no database (data is not loaded into a database), there is no database schema (data is mostly semi-structured and sparse), and there is no support for database queries (application involve mainly text search over semi-structured data). It is therefore difficult to regard such applications as database applications. Learning from history we can observe that a similar situation arose in the 1990s with Object-Oriented databases and later with XML databases, and conclude that there is little benefit in applying database solutions to problems that do not fit the database paradigm.

ACKNOWLEDGMENT
This research has been supported by GAČR (Grant Agency, Czech Republic) grant No. P403/10/0092 - Advanced Principles and Models for Enterprise ICT Management, University of Economics, Prague IGA (Internal Grant Agency) grant No. IG406040 – Cloud Computing Adoption and Governance, and the Research Centre for Human Centered Technology Design at the University of Technology, Sydney.

References

1. *Data, data everywhere - A special report on managing information*, in *The Economist*. 2010.
2. Gantz, J.F., *The Diverse and Exploding Digital Universe*. 2008, IDC. IDC. http://www.emc.com/collateral/analyst-reports/diverse-exploding-digital-universe.pdf
3. CERN. *The Large Hadron Collider*. 2010 14 MArch 2010]; CERN]. Available from: http://public.web.cern.ch/public/en/LHC/LHC-en.html.

4. Rothschild, J. *High Performance at Massive Scale – Lessons learned at Facebook*. CNS 2009 Lecture Series Archives 2009 [cited 2010 15 March 2010]; Centre for Networked Systems Lecture]. Available from: http://cns.ucsd.edu/lecturearchive09.shtml.
5. Dean, J. *Designs, Lessons and Advice from Building Large Distributed Systems*. 2009 [cited 2010 18 March 2010]; Available from: http://www.odbms.org/download/dean-keynote-ladis2009.pdf.
6. Codd, E.F., *A relational model of data for large shared data banks.* Commun. ACM, 1970. **13**(6): p. 377-387.
7. Atkinson, M., et al. *The object-oriented database system manifesto*. 1989: Citeseer.
8. Stonebraker, M., et al., *Third-generation database system manifesto.* SIGMOD Rec., 1990. **19**(3): p. 31-44.
9. Eisenberg, A. and J. Melton, *SQL: 1999, formerly known as SQL3.* ACM SIGMOD Record, 1999. **28**(1): p. 138.
10. Chamberlin, D. *XQuery: A query language for XML*. 2003: ACM New York, NY, USA.
11. W3C. *XML Schema*. 2010 [cited 2010 18 March 2010]; XML Schema Standard Sepcification]. Available from: http://www.w3.org/XML/Schema.
12. Widom, J., *Data management for XML: Research directions.* Bulletin of the Technical Committee on: p. 44.
13. Funderburk, J., et al., *XTABLES: Bridging relational technology and XML.* IBM Systems Journal, 2002. **41**(4): p. 616-641.
14. Fernández, M., W. Tan, and D. Suciu, *SilkRoute: trading between relations and XML.* Computer Networks, 2000. **33**(1-6): p. 723-745.
15. Schöning, H. *Tamino-a DBMS designed for XML*. 2001: IEEE Computer Society.
16. Bourret, R. *XML and Databases*. 2005 [cited 2010 18 March 2010]; Available from: http://ece.ut.ac.ir/DBRG/seminars/AdvancedDB/2006/Sanamrad-Hoseininasab/Refrences/6.pdf.
17. Eisenberg, A., et al., *SQL: 2003 has been published.* ACM SIGMOD Record, 2004. **33**(1): p. 119-126.
18. Hyde, J., *Data in flight.* Commun. ACM. **53**(1): p. 48-52.
19. Aboulnaga, A., et al., *Deploying Database Appliances in the Cloud.* Data Engineering, 2009. **32**(1): p. 13.
20. Silberschatz, A., M. Stonebraker, and J. Ullman, *Database research: Achievements and opportunities into the 21st century.* SIGMOD RECORD, 1996. **25**(1): p. 52–63.
21. Agrawal, R., et al., *The Claremont report on database research.* SIGMOD Rec., 2008. **37**(3): p. 9-19.
22. Ramakrishnan, R. and J. Ullman, *A survey of deductive database systems.* The journal of logic programming, 1995. **23**(2): p. 125-149.
23. Kerschberg, L., *Expert database systems: Knowledge/data management environments for intelligent information systems.* Information Systems, 1990. **15**(1): p. 151-160.

24. Dean, J. and S. Ghemawat, *MapReduce: Simplified data processing on large clusters.*
25. Borthakur, D., *The hadoop distributed file system: Architecture and design.* Hadoop Project Website, 2007.
26. Chang, F., et al. *Bigtable: A distributed storage system for structured data*. in *OSDI '06*. 2006.
27. Lai, E., *No to SQL? Anti-database movement gains steam*, in *Computerworld*. 2009. http://www.computerworld.com/s/article/9135086/No_to_SQL_Anti_database_movement_gains_steam_
28. Stonebraker, M., et al., *MapReduce and parallel DBMSs: friends or foes?* Commun. ACM. **53**(1): p. 64-71.
29. Lai, E., *Big three database vendors diverge on Hadoop*, in *Computerworld*. 2009, IDG. http://www.computerworld.com/s/article/9142406/Big_three_database_vendors_diverge_on_Hadoop
30. Morgan, T.P. *Netezza to bake analytics into appliances*. 2010 [cited 2010 18 March 2010]; Available from: http://www.theregister.co.uk/2010/02/24/netezza_data_analytics/.

Content-based Retrieval of Compressed Images

Gerald Schaefer

Department of Computer Science
Loughborough University
Loughborough, U.K.
gerald.schaefer@ieee.org

Abstract. Content-based image retrieval allows search for pictures in large image databases without keyword or text annotations. Much progress has been made in deriving useful image features with most of these features being extracted from (uncompressed) pixel data. However, the vast majority of images today are stored in compressed form due to limitations in terms of storage and bandwidth resources. In this paper, we therefore investigate a different approach, namely that of compressed-domain image retrieval, and present some compressed-domain image retrieval techniques that we have developed over the past years. In particular, a method for retrieving images compressed by vector quantisation, that uses codebook information as image features, is presented. Retrieval of losslessly compressed images obtained using lossless JPEG, can be retrieved using information derived from the Huffman coding tables of the compressed files. Finally, CVPIC, a 4-th criterion image compression technique is introduced and it is demonstrated that compressed-domain image retrieval based on CVPIC is not only able to match the performance of common retrieval techniques on uncompressed images, but even clearly outperforms these.

Keywords: content-based image retrieval (CBIR), image compression, compressed-domain image retrieval, vector quantisation, lossless JPEG, CVPIC

1 Introduction

With the recent explosion in availability of digital imagery the need for content-based image retrieval (CBIR) is ever increasing. While many methods have been suggested in the literature, only few take into account the fact that - due to limited resources such as disk space and bandwidth - virtually all images are stored in compressed form. In order to process them for CBIR they first need to be uncompressed and the features calculated in the pixel domain. The desire for techniques that operate directly in the compressed domain providing, so-called midstream content access, is therefore evident [18].

In this paper, we introduce several techniques that perform compressed-domain image retrieval. In general there are two approaches. The first is based on existing compression techniques and tries to extract useful information from the compressed data streams produced by these. The second approach is to develop so-called 4-th criterion compression algorithms where the data in compressed form is directly visually meaningful and can hence be exploited for image retrieval. We will cover both approaches in this paper.

2 Content-based image retrieval

Since textual annotations are not available for most images, searching for particular pictures becomes an inherently difficult task. Luckily, a lot of research has been conducted over the last two decades leading to various approaches for content-based image retrieval [30, 2]. Content-based image retrieval (CBIR) does not rely on textual attributes but allows search based on features that are directly extracted from the images [30]. This however is, not surprisingly, rather challenging and often relies on the notion of visual similarity between images or image regions.

While humans are capable of effortlessly matching similar images or objects, machine vision research still has a long way to go before it will reach a similar performance for computers. Currently, many retrieval approaches are based on low-level features such as colour, texture, and shape features, leaving a 'semantic gap' to the high-level understanding of users [30].

3 Image compression and compressed-domain CBIR

Despite continuous advances in technology both storage space and bandwidth are still limited. In terms of the storage and transmission of images (e.g. through the Internet) this means that images have to be stored in compressed form. However, to achieve this compression some of the original image information needs to be sacrificed; that is, the compressed image will differ from the original image. Consequently, image descriptors obtained from compressed images will also be somewhat different from those derived from their uncompressed counterparts. In one of our studies [28], we investigated the effect image compression has on the performance of several popular CBIR techniques [33, 4, 32, 17, 6, 1]. We found that the resulting drop in retrieval performance is small yet not negligible [28].

Although most images exist only in compressed form, almost all CBIR techniques operate in the pixel domain. In contrast, compressed domain techniques operate directly on the compressed data without the need for decompression [13]. Compressed domain CBIR can be performed either based on existing compression formats such as JPEG [19] or vector quantisation [25], or employing so-called 4-th criterion compression techniques where the compressed information is directly visually meaningful [18].

4 Compressed-domain CBIR based on vector quantisation

4.1 Vector quantisation

Vector quantisation (VQ [5]) represents a mapping that assigns to each input vector a codebook vector achieving compression by setting the size of the codebook small relative to the possible gamut of input vectors. In particular, in terms of VQ image compression, an image is divided into a set of L-dimensional vectors I by splitting it into image blocks where each block forms a vector. A codebook C with N entries is then found. There are many ways how this can be achieved. In our approach we start with one codebook entry (the mean of the distribution) and then iteratively add new entries by identifying and splitting the cluster which has the largest variance. After the desired codebook size has been reached we apply the LBG algorithm [12] to optimise the generated codebook. Once a codebook is defined the input vectors can be mapped to vectors (codewords) of the codebook according to a nearest neighbour rule:

$$I_i \rightarrow C_j \text{ iff } d(I_i, C_j) \leq d(I_i, C_k) \; \forall \; C_k \in C \tag{1}$$

The respective image block can then be represented by an index to the closest codeword only.

4.2 CBIR through VQ codebook matching

Even though information is lost due to the compression, image retrieval based on VQ data not only provides information on the colour content, as do colour histograms for example, but also on the spatial information (encompassing textural and shape attributes) of the image, which is due to the image being divided into blocks and the blocks coded as a whole.

In our algorithm we use block sizes of 4×4 pixels thus giving vectors of length 48 for colour images. In contrast to previous methods which use a universal codebook for all images, and then base their retrieval technique on histograms [10] or binarised histograms [9] of codebook indices, codebooks were generated on a per image basis ensuring that image quality is high, even for a small number of codes. Also, this not only makes image distribution easier (the there is no need for codebook negotiation between encoder and decoder) but also guarantees that the information stored by the codevectors is optimally adapted for each image. Indeed, this is the key feature that is used in our technique. Because prototype codes encompass precise information about an image, images can be compared by using the content stored in their respective codebooks.

There are several ways to compare 2 L-dimensional point sets C_A and C_B. One choice would be to use the Hausdorff distance [8]. However, as this is based on a max-min operator, the original Hausdorff distance can become highly dependent on outliers and so is statistically not very robust. A better way to compare two VQ codebooks would therefore be to use a variant of the Hausdorff distance which shows more robustness. In our approach, we use a Modified Hausdorff distance $\mathrm{HD}_{\mathrm{mod}}$ defined as

$$\mathrm{HD}_{\mathrm{mod}} = \max(\mathrm{hd}_{\mathrm{mod}}(C_A, C_B), \mathrm{hd}_{\mathrm{mod}}(C_B, C_A)) \tag{2}$$

with

$$\mathrm{hd}_{\mathrm{mod}}(C_A, C_B) = \frac{1}{N} \sum_{i=1}^{N} \min_j \|C_A(i) - C_B(j)\| \tag{3}$$

where $\|.\|$ denotes some underlying norm, in our case the L_2 norm. Rather than taking the maximum of the minima as in the original Hausdorff distance we use the average of the minimum which makes the distance measure less sensitive to outliers [3, 26].

After calculating the distances to all images in the database, the images can be ranked in order of their similarity to a given query image.

4.3 Experimental results

We performed VQ image retrieval on the MPEG-7 Common colour dataset [15]. This database consists of 5466 images and a set of 50 queries with predefined ground truth images. We compressed the images with codebooks of size 64, and performed image retrieval based on the Modified Hausdorff distances between the VQ codebooks. We use the MPEG-7 Normalised Modified Retrieval Rank (NMRR) [15] as the standard performance measure for this data set. The NMRR is defined as

$$\mathrm{NMRR} = \frac{\mathrm{MRR}(q)}{K + 1/2 - N_G/2} \tag{4}$$

where $\mathrm{MRR}(q) = \mu(q) - 1/2 - N_G(q)/2$ and $\mu(q) = \sum_{i=1}^{N_G(q)} r_i / N_G(q)$. $N_G(q)$ is the number of ground truth images for the qth query image and r_i denotes the retrieved rank. For K we use the MPEG-7 recommendation $K = \min(4N_G(q), 2\max_q(N_G(q)))$.

The results achieved give an average NMRR of 0.1196. In comparison, image retrieval based on colour histograms [33] results in an average NMRR of 0.1075. The slight drop in performance can be explained with the fact that we are essentially compressing the already severely (JPEG) compressed images of the MPEG-7 dataset again and hence part of the information stored in the VQ codebooks can be attributed to compression artefacts rather than to image content.

5 Compressed-domain CBIR based on lossless JPEG

5.1 Lossless JPEG compression

Predictive image coders work on the basis that images tend to change slowly over most areas of an image. Consequently, most neighbouring pixels will have similar values. A pixel at location (i, j) is predicted, based on the values of its neighbouring pixels as

$$P'_{(i,j)} = \sum_{k<i,l<j} \omega_{(k,l)} P_{(k,l)} \tag{5}$$

where P' represents the prediction, P are the actual pixel values of the neighbouring pixels, and ω describe weights used for the prediction. In this paper we adapt one of the predictors of the lossless JPEG [34] scheme, in particular, the JPEG-7 predictor where pixels are predicted as the the average of their top and left neighbours (i.e. $\omega_{(i-1,j)} = \omega_{(i,j-1)} = 0.5$).

Once a pixel has been predicted it is encoded as the difference between its actual value and its prediction:

$$D_{(i,j)} = P_{(i,j)} - P'_{(i,j)} \tag{6}$$

This has the advantage that now differences close to 0 are much more likely than higher differences. Consequently, an entropy encoding stage, which assigns shorter codewords to more frequent codes and longer codewords to rarer events, is then applied. In our framework we use a Huffman coder [7] for performing the entropy coding. Huffman coders are optimal in the sense that they allow encoding data using the minimal number of bits (with the restriction that each codeword has an integer number of bits).

The losslessly compressed image then comprises two parts: the Huffman table, and the differences now represented as indices into the Huffman table.

5.2 CBIR in the losslessly compressed domain

Difference histogram In order to find a way to index the compressed images directly in the encoded domain, we first reverse the entropy coding stage. This is also being done by all other methods that operate in the compressed domain where entropy coding is part of the compression algorithm [13]. After this, we naturally end up with the difference data $D_{(i,j)}$ for each pixel. We now want to make explicit what this data actually means. The prediction of each pixel is essentially a statistical description of its neighbourhood. By calculating the difference to the actual pixel value, the resulting descriptor $D_{(i,j)}$ represents the change of the pixel compared to its neighbourhood. Texture can be defined as a property that pixels exhibit in comparison to their neighbourhood. Therefore, the differences between the predictions and the actual pixel values also define a description of the textural properties of the image.

Hence, we propose to use the difference data directly as a description of the image content. Building histograms of the differences seems to provide a good choice. However, one has to be aware that the distribution of the prediction differences is not uniform. Differences close to 0 are much more likely than higher values. To rectify this we first apply a non-linear transformation to the predictor differences:

$$D'_{(i,j)} = \begin{cases} -M - \log(-D_{(i,j)}) & \text{if } D_{(i,j)} < 0 \\ 0 & \text{if } D_{(i,j)} = 0 \\ M + \log(D_{(i,j)}) & \text{if } D_{(i,j)} > 0 \end{cases} \tag{7}$$

where $M = \log(1/255)$, i.e. the transformed value of the smallest prediction difference possible.

After this transformation we build a uniformly quantised histogram of the D's. Once histograms H_i are built, they can be compared using histogram intersection, as described in [33]

$$d(H_1, H_2) = \sum_k \min\{H_1(k), H_2(k)\} \tag{8}$$

where H_1 and H_2 are the histograms of the D' coefficients. Image retrieval is performed by calculating the distances between a query image and all images in the database, and returning the closest matches.

Codebook matching As we have mentioned above, all algorithms to date that operate in the compressed domain of images need to reverse the entropy coding as a first step. We will now introduce a technique that, based on the predictive coding framework outlined in Section 5.1, allows for image indexing directly in the compressed image data without the need to undo the entropy coding. In particular, we will use the Huffman codebook itself as the index.

The Huffman codebook contains one codeword for each possible difference in the interval $[-255; 255]$. Shorter codewords are assigned to events that are more probable. Consequently, the length of a Huffman codeword is indirectly proportional to its frequency in the image. That is, the codebook contains approximately the same information as a histogram of the data! Hence to compare two images, one can compare their codebooks. To do this we calculate the cumulative difference of codeword lengths

$$d(B_1, B_2) = \sum_{-255<k<255} |B_1(k) - B_2(k)| \tag{9}$$

where B_1 and B_2 are the Huffman codebooks of two images, and $|.|$ is the L_1 norm. Codewords that are not present in a codebook are assigned the maximum length that is to be found in the respective codebook before the comparison.

5.3 Experimental Results

We took 80 images of the VisTex [14] image set, a collection of colour texture images from MIT, and extracted from each of the 512×512 pixel images two 256×256 non-overlapping regions. One of each was assigned model while the others represent the query images. In order to acquire the following results, each query image was compared to each model image, and as we know which one is the corresponding picture the rank in which the correct image is retrieved can be recorded. As performance measure, we use the match percentile [33] defined as

$$MP = \frac{N - R}{N - 1} \tag{10}$$

or rather the average match percentile over all query images as a measure of goodness for assessing retrieval performance. Here N is the number of model images in the database, and R is the rank, i.e. the position of the correct match in the retrieval list.

We encoded all images using the JPEG-7 predictor and Huffman compression as explained in Section 5.1. As we deal with RGB images, each channel was coded separately. Image retrieval was then performed by computing the difference histograms ($35 \times 35 \times 35$ bins) as defined in Section 5.2 and calculating the histogram intersection (Equation (8)) of each query image to each of the models, before the returned models were ranked according to their distance to the query. The average match percentile achieved over the whole dataset is 98.20 with 88.75% of the correct images retrieved in 1st rank.

In order to compare this performance, we also applied the rotation invariant version of the LBP operator [16] to the images. LBP has been shown to represent a powerful texture classification technique that outperforms most other standard texture algorithms [16]. The average match percentile, based on the resulting $36 \times 36 \times 36$ LBP histograms is 98.50 (92.50% 1st rank retrievals). Hence we see that our proposed algorithm performs comparable to current state-of-the-art techniques.

Finally, we also evaluated the performance of our codebook matching algorithm from Section 5.2 on the VisTex dataset. The result is an average match percentile of 96.36 with 67.50% first rank retrievals. We see confirmed what we suspected, namely that the performance drops due to the inexactness of the representation, and also due to the lack of measurement of correlation between the channels (which for 3-dimensional histogram is preserved). However, a match percentile of more than 96 is still a very good basis to reject most of the images and leave only those that are close to the query.

6 Compressed-domain CBIR based on CVPIC

6.1 Colour Visual Pattern Image Coding

Colour Visual Pattern Image Coding (CVPIC) divides an image into small 4×4 pixel blocks and then matches each block to one of a pre-defined classes of patterns (14 edge patterns shown in Figure 1, plus a uniform block, i.e. a block without an edge), followed by quantisation of the colour information. The compressed data stream contains direct information about colour and shape information of the image, and we have introduced various techniques for performing compressed domain CBIR in the CVPIC domain [21, 20, 22, 24, 23]. In here, we focus on the approach presented in [23].

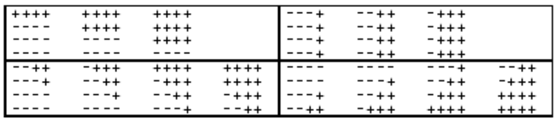


Fig. 1. The 14 edge patterns used in CVPIC [27].

6.2 CBIR based on CVPIC data

The data that is readily available in CVPIC compressed images is the colour information of each of the 4×4 image blocks, and information on the spatial characteristics of each block, in particular on whether a given block is identified as a uniform block (a block with no or little variation) or a pattern block (a block where an edge or gradient has been detected). Furthermore, each pattern block is assigned to one of 14 universally predefined classes according to the orientation and position of the edge within the block. We make direct use of this information to derive an image retrieval algorithm that utilises both colour and shape information. The colour information is summarised similar to colour coherence vectors introduced in [17] and the border/interior pixel approach in [31] which both show that dividing the pixels of an image into those that are part of a uniform area and those that are not can improve retrieval performance.

In essence we create two colour histograms, one for uniform blocks and one for non-uniform (pattern) blocks. Shape descriptors are often calculated as statistical summaries of local edge information such as in [11] where the edge orientation and magnitude is determined at each pixel location and an edge histogram calculated. Exploiting the CVPIC image structure, an effective shape descriptor can be determined very efficiently. Since each (pattern) block contains exactly one (pre-calculated) edge and there are 14 different patterns we simply build a 1×14 histogram of the edge indices. CVPIC image retrieval based on both colour and shape features can then be performed by calculating the combined difference (a weighted L_1 norm) between all three histograms.

Table 1. Retrieval results obtained on the UCID dataset in terms of modified average match percentile [29].

	MP
Colour histograms [33]	90.47
Colour coherence vectors [17]	91.03
Border/interior pixel histograms [31]	91.27
Colour correlograms [6]	89.96
CVPIC	94.24

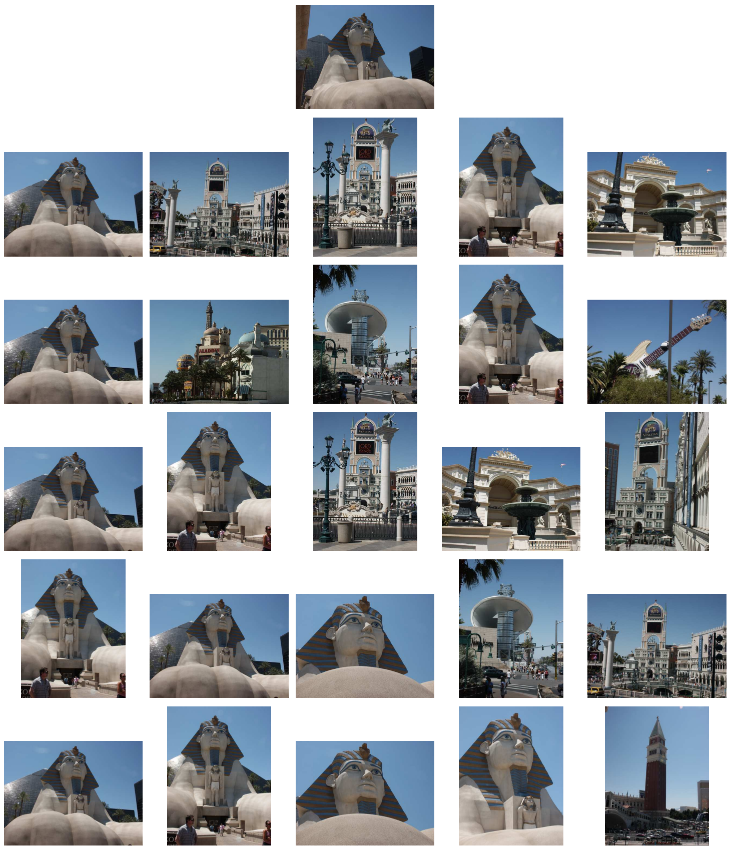


Fig. 2. Sample query together with top five ranked images returned by (from top to bottom) colour histograms, colour coherence vectors, border/interior pixel histograms, colour correlograms, and CVPIC retrieval.

6.3 Experimental results

Results on the UCID dataset [29], shown in Table 1 in terms of match percentile, confirm that this approach is not only capable of performing efficient and effective compressed domain image retrieval but that our algorithm also outperforms various popular CBIR techniques, even when these are run on uncompressed images. An example query with the top five retrieved images obtained from several pixel domain CBIR methods and our CVPIC techniques is shown in Figure 2.

7 Conclusions

Compressed-domain image retrieval provides an interesting alternative to common image retrieval algorithms as it provides the advantage that image features are extracted directly from the compressed data stream of images. In this paper we have presented several compressed domain techniques that allow efficient and effective querying of large image databases. In particular, we have presented techniques based on vector quantisation, lossless JPEG compression, and CVPIC, a 4-th criterion compression algorithm. Experimental results have shown that the introduced techniques are able to match or even exceed the performance of common pixel-based retrieval algorithms.

References

1. L. Cinque, S. Levialdi, and A. Pellicano. Color-based image retrieval using spatial-chromatic histograms. In *IEEE Int. Conf. Multimedia Computing and Systems*, pages 969–973, 1999.
2. R. Datta, D. Joshi, J. Li, and J. Z. Wang. Image retrieval: Ideas, influences, and trends of the new age. *ACM Computing Surveys*, 40(2):1–60, 2008.
3. M-P. Dubiusson and A.K. Jain. A modified Hausdorff distance for object matching. In *12th IEEE Int. Conference on Pattern Recognition*, pages 566–568, 1994.
4. C. Faloutsos, W. Equitz, M. Flickner, W. Niblack, D. Petkovic, and R. Barber. Efficient and effective querying by image content. *journal of Intelligent Information Retrieval*, 3(3/4):231–262, 1994.
5. R.M. Gray. Vector quantization. *IEEE ASSP Magazine*, 1(2):4–29, 1984.
6. J. Huang, S.R. Kumar, M. Mitra, W-J. Zhu, and R. Zabih. Image indexing using color correlograms. In *IEEE Int. Conference Computer Vision and Pattern Recognition*, pages 762–768, 1997.
7. D.A. Huffman. A method for the construction of minimum redundancy codes. *Proceedings of the Institute of Radio Engineers*, 40:1098–1101, 1952.
8. D.P. Huttenlocher, G.A. Klanderman, and W.J. Rucklidge. Comparing images using the Hausdorff distance. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 15(9):850–863, 1993.
9. F. Idris and S. Panchanathan. Storage and retrieval of compressed images. *IEEE Trans. Consumer Electronics*, 41(3):937–941, 1995.
10. F. Idris and S. Panchanathan. Image and video indexing using vector quantization. *Machine Vision and Applications*, 10:43–50, 1997.
11. A.K. Jain and A. Vailaya. Image retrieval using color and shape. *Pattern Recognition*, 29(8):1233–1244, 1996.
12. Y. Linde, A. Buzo, and R.M. Gray. An algorithm for vector quantizer design. *IEEE Trans. Communications*, 28:84–95, 1980.
13. M. Mandal, F. Idris, and S. Panchanathan. A critical evaluation of image and video indexing techniques in the compressed domain. *Image and Vision Computing*, 17(7):513–529, 1999.
14. MIT. VisTex Vision Texture database. `http://www-white.media.mit.edu/vismod/imagery/VisionTexture/vistex.html`.
15. Moving Picture Experts Group. Description of core experiments for MPEG-7 color/texture descriptors. Technical Report ISO/IEC JTC1/SC29/WG11/ N2929, 1999.
16. T. Ojala, M. Pietikäinen, and T. Menpää. Gray scale and rotation invariant texture classification with local binary patterns. In *6th European Conference on Computer Vision*, pages 404–420, 2000.

17. G. Pass and R. Zabih. Histogram refinement for content-based image retrieval. In *3rd IEEE Workshop on Applications of Computer Vision*, pages 96–102, 1996.
18. R.W. Picard. Content access for image/video coding: The fourth criterion. Technical Report 195, MIT Media Lab, 1994.
19. G. Schaefer. JPEG image retrieval by simple operators. In *2nd International Workshop on Content-Based Multimedia Indexing*, pages 207–214, 2001.
20. G. Schaefer. Mistream content access of visual pattern coded imagery. In *4th Int. Workshop on Multimedia Data and Document Engineering*, 2004.
21. G. Schaefer and S. Lieutaud. Colour and shape based image retrieval for CVPIC coded images. In *Int. Conference on Imaging Science, Systems, and Technology*, pages 456–461, 2004.
22. G. Schaefer and S. Lieutaud. CVPIC based uniform/non-uniform colour histograms for compressed domain image retrieval. In *7th Int. Conference on VISual Information Systems*, pages 344–348, 2004.
23. G. Schaefer and S. Lieutaud. CVPIC compressed domain image retrieval by colour and shape. In *Int. Conference on Image Analysis and Recognition*, volume 3211 of *Springer Lecture Notes on Computer Science*, pages 778–786, 2004.
24. G. Schaefer, S. Lieutaud, and G. Qiu. CVPIC image retrieval based on block colour co-occurance matrix and pattern histogram. In *IEEE Int. Conference on Image Processing*, pages 413–416, 2004.
25. G. Schaefer and W. Naumienko. Midstream content access by VQ codebook matching. In *Int. Conference on Imaging Science, Systems, and Technology*, pages 170–176, 2003.
26. G. Schaefer, G. Qiu, and G. Finlayson. Retrieval of palettised colour images. In *Storage and Retrieval for Image and Video Databases VIII*, volume 3972 of *Proceedings of SPIE*, pages 483–493, 2000.
27. G. Schaefer, G. Qiu, and M.R. Luo. Visual pattern based colour image compression. In *Visual Communication and Image Processing '99*, volume 3653 of *Proceedings of SPIE*, pages 989–997, 1999.
28. G. Schaefer and S. Ruszala. Effective and efficient browsing of image databases. *Int. Journal of Imaging Systems and Technology*, 18:137–145, 2008.
29. G. Schaefer and M. Stich. UCID - An Uncompressed Colour Image Database. In *Storage and Retrieval Methods and Applications for Multimedia 2004*, volume 5307 of *Proceedings of SPIE*, pages 472–480, 2004.
30. A.W.M. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of the early years. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 22(12):1249–1380, 2000.
31. R.O. Stehling, M.A. Nascimento, and A.X. Falcao. A compact and efficient image retrieval approach based on border/interior pixel classification. In *Proc. 11th Int. Conf. on Information and Knowledge Management*, pages 102–109, 2002.
32. M. Stricker and M. Orengo. Similarity of color images. In *Conf. on Storage and Retrieval for Image and Video Databases III*, volume 2420 of *Proceedings of SPIE*, pages 381–392, 1995.
33. M.J. Swain and D.H. Ballard. Color indexing. *Int. Journal of Computer Vision*, 7(11):11–32, 1991.
34. G.K. Wallace. The JPEG still picture compression standard. *Communications of the ACM*, 34(4):30–44, 1991.

Author Index