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

Proceedings of the Dateso 2009 Workshop

DATESO
Databases, Texts
Specifications, and Objects
2009

http://www.cs.vsb.cz/dateso/2009/
http://www.ceur-ws.org/

April 15 – 17, 2009
Špindlerův Mlýn – Patejdlova bouda

DATESO 2009
© K. Richta, J. Pokorný, V. Snášel, editors

Technical editor:
Pavel Strnad, `strnap1@fel.cvut.cz`

Page count:	178
Impression:	50
Edition:	1st
First published:	2009

This proceedings was typeset by PDFLaTeX.
Cover design by Markéta Růžičková
Printed and bound in Prague, Czech Republic by Česká technika - nakladatelství ČVUT.

Published by Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science and Engineering

Preface

DATESO 2009, the international workshop on current trends on Databases, Information Retrieval, Algebraic Specification and Object Oriented Programming, was held on April 15 – 17, 2009 in Špindlerův Mlýn – Patejdlova bouda. This was the 9th annual workshop organized by Department of Computer Science and Engineering FEL ČVUT Praha, and Department of Software Engineering MFF UK Praha, and Department of Computer Science VŠB-Technical University Ostrava, and working group on Informatics 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 2009 are also available at DATESO Web site `http://www.cs.vsb.cz/dateso/2009/`, and CEUR Workshop Proceeding pages `http://www.ceurws.org/` (ISSN 1613-0073).

The Program Committee selected 14 papers from 21 submissions, based on two independent reviews.

We wish to express our sincere thanks to all the authors who submitted papers, the members of the Program Committee, who reviewed them on the basis of originality, technical quality, and presentation. We are also thankful to the Organizing Committee for preparation of workshop and its proceedings. Special thanks belong to Czech Society for Cybernetics and Informatics for its support of publishing this issue.

March, 2009 K. Richta, J. Pokorný, V. Snášel (Eds.)

Table of Contents

Speeding Up Shortest Path Search in Public Transport Networks [⋆]

Vladislav Martínek and Michal Žemlička

Charles University, Faculty of Mathematics and Physics
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
VladaM@seznam.cz, Michal.Zemlicka@mff.cuni.cz

Abstract. The searching for the shortest path in public transport networks can take more time than is acceptable for given situation. We have therefore searched for methods that speed up the given calculation. The approach, when the calculation is not performed on the original network but on the simplified one, seems to be very promising. The path found in the simplified network can be easily mapped to a corresponding path in the original network. In the case of the Prague public transport the simplified network has several times less nodes and the computation is speeded up correspondingly.

Keywords: shortest path search in public transport networks, network simplification

1 Introduction

Searching for the optimal connection between two various places is a frequent task solved in public transport networks. The path duration is an important criterion of the searched connection. From the view of graph theory the problem can be seen as a shortest path search. It is possible to keep in mind other criteria derived from user preferences or restrictions when choosing the connection. Traditional approaches to the shortest path search (recent optimizations compared in [1] expect searching in a static network). Most of these approaches are not applicable on dynamic network without additional modifications[1]. The method introduced in [2] is performing data reduction to simplify train network. Such reduction is generally an NP-hard problem. Fortunately on real data the problem is usually solvable within acceptable time [3]. Our paper introduces similar approach to the graph reduction aimed at the urban public transport and at the practical use of this reduced data in mobile devices application.

[⋆] This paper was partially supported by the Program "Information Society" under project 1ET100300517 and by the Czech Science Foundation by the grant number 201/09/0983.

[1] For example a bi-directional search would be difficult to introduce in public transport networks, if we do not know the arrival time.

1.1 Basic Solution Approaches

Two basic approaches can be considered when solving the problem. The first approach is to find the path between all vertices and then only return results on query. The situation is complicated in mass transport networks by the fact, that the edge value is determined according to the actual time. The precomputation of the results would mean to find the shortest paths between all vertices for certain time interval. This approach is suitable when the number of queries is relatively high and there is sufficient memory and computation power for reaction to data changes in appropriate time. The hardware requirements may be impossible to satisfy for the large networks as described in [4].

The second approach is to find the shortest path directly according to the given parameters. This approach is suitable, if the number of queries is relatively low and if it is possible to find the answer in acceptable time. The advantage against the previous case is that the variable value of the edges does not mean serious complication in our case.

Algorithms used for direct computing of the shortest path derived from the algorithm published by Dijkstra in [5]. Their complexity is typically superlinear with respect to the number of vertices and edges. If we succeed in reducing the size of input graph, the computation speed will be increased significantly.

1.2 Scheduled and Real Traffic

Timetables determine prescript departure times of individual connections which may vary from actual times of departure. There typically occur two types of irregularities. The first ones may occur relatively frequently and may be relatively small. They may be caused by the current density of traffic, weather, road conditions or other relatively predictable effects. One of the user's preferences could be a requirement for reliability of the connection.

The second irregularity type is caused by extraordinary events of a larger impact. They cannot be predicted, and cause relatively large irregularities from the timetables. Typically can cause temporary interference of the carrier into the timetables. These temporary changes in the timetables would be difficult to handle in the case of precomputed results.

To choose the connection so it meets all user preferences is the case of the search algorithm. This paper is focused on the reduction of input data. For the selection of a connection we will consider only one criterion – the path length (duration).

2 Mass Transport Network Representation

Mass transport system can be seen as an oriented multigraph with valued edges.

Pseudoline is a representation of the certain line of the mass transport system or a walk. Every line of the mass transport has its own timetable and route. In order to separate various line directions, it is suitable to represent them as separate pseudolines.

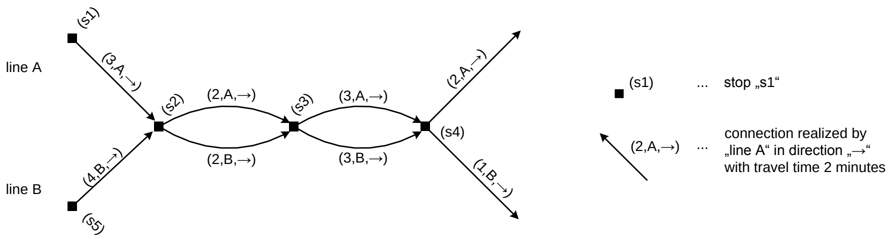


Fig. 1. Graph example

Vertex represents the street refuge of the stop or a platform of the station. Every stop can have several various refuges. In order to count time between refuges in a walk, it is appropriate to represent them as separate vertices.

Edge represents direct connection between two stops, or more precisely refuges. The connection is realized by one of the pseudolines, this information is marked inside the edge structure. If several direct connections are available between two vertices, then every single one of them is represented by the separate edge.

Edge value is expected travel time of the pseudoline between the stops connected by the edge. Travel time may vary according to the time of a day[2].

Waiting time is expected time spent on waiting for the service arrival. This value is added to the edge value in case of transfer between services. It is determined on the basis of actual time and valid timetable of the given line.

There could be various exceptions in the timetables – for example the service has a variable route or is avoiding some stops in certain moments. This situation can be handled by creation of new pseudolines for each type of the exception. The exceptions can be excluded from the original pseudoline and delegated into the new pseudoline. Several new pseudolines can be created on the basis of one line. New pseudoline will be marked in the same way as the original pseudoline for the user.

When searching for the shortest path in mass transport network, it is necessary to count the edge value and also the waiting time in the path length. The waiting time is counted only in the case of transferring between the services or getting in a service. In order to detect the transfers it is necessary to remember the pseudoline, which has been used to get to the vertex. If a new edge is added to the current path and this edge is realized by other pseudoline than the one

[2] For example a "shorted" travel time – B and a normal travel time – A is set for the trams in weekdays according to the departure time from the stop: 0:00-6:59 - B, 7:00-18:59 - A, 19:00-23:59 - B.

used to get to the last vertex in the current path, then the waiting time is to be added to the value of new edge.

The value of the edge realized by the walk should be derived from walk time. Value of the walk edge can markedly vary according to the user preferences.

Recognition of the resulting path Classical algorithms for searching for the shortest path between two vertices in the graph terminates the search at the moment the target is processed. In each vertex there is stored a pointer to the predecessor, from which the vertex was reached. In the case of a multigraph this information is insufficient to recognize the resulting path and it is necessary to remember also the edge leading from the predecessor to the vertex.

Correctness The resulting path found in the reduced graphs must be equal to the path found in the original graph for the equal search parameters. Introducing pseudolines does not change the results presented to the user. Each of the pseudolines holds the identification of the real line, which is presented to the user. The pseudoline is equal to the subset of services of the original line. Transfers between pseudolines are equal to the transfers between original lines. However, the transfers between pseudolines identified by the same line could be redundant[3]. The redundant transfer can occur in the real situation as well, without influence to the resulting path length.

3 Graph Reduction

To reduce the size of input data, we will try to reduce certain edges and vertices in the input graph using appropriate adjustments. Reduction of edges is achieved by a replacement of several original edges with a single new one. The new edge will fully represent all of the original edges when searching for the shortest path. The reduction of vertices occurs so that after reduction of edges some isolated vertices remain in the graph, which are not reasonable to hold for the shortest path search. When the shortest path is found in the reduced graph, it is important to be able to reconstruct the appropriate path in the original graph easily. If the mapping of the path to the original network is too complicated, the advantage of searching in reduced graph could be eliminated. Moreover, the edge values must be maintained, otherwise the condition of the shortest path could be violated.

The following adjustments are intended to be used in the way described. The separate usage of adjustments is possible, but with weaker effect. The result of their use in an opposite order is unsure. The results compared with the previous adjustment are listed in the table below each of them.

3.1 Edge Aggregation

The first adjustment is based on the following heuristics: To get from one stop to the next one in the shortest possible time, it is necessary to get on the service,

[3] An additional transfer may occur if the pseudoline represents the shortened route of the certain line. This can be easily solved by postprocessing of the path found.

which will arrive there first. If several services take the same time to run between those stops, the service with the shortest waiting time should be taken. To find out how long to wait for each connection, we are forced to view timetables of all these connections.

The above situation is in the graph indicated by the two vertices which are connected by several edges with the same value. These edges differ only by pseudoline. We will create a new pseudoline which is a combination of all pseudolines of the edges mentioned above. These edges can be all replaced by a single new edge. The new edge connects the same two vertices as the original edges does and has the same value as the original edges, but it is labeled by a new pseudoline:

Merge-line is a new pseudoline, which is created as a combination of several original pseudolines. The merge-line has its own timetable, which is a combination of the timetables of original pseudolines. The merge-line leaves the stop every time, when one of the original pseudolines leaves the stop. In order to map the path found in new graph to the original graph, it is necessary to remember from which pseudolines the merge-line was created.

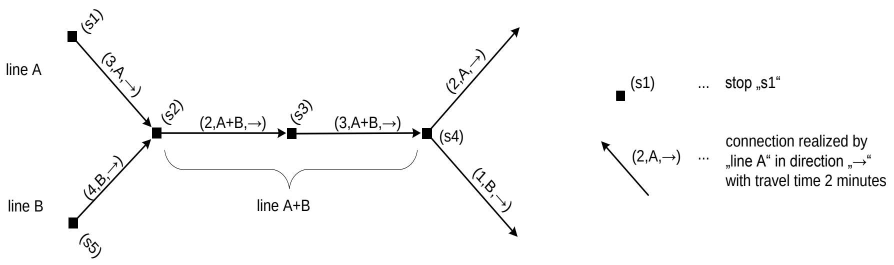


Fig. 2. Aggregation of the edges with the same travel time

Mapping the Path to the Original Network The vertices of the new graph directly correspond to the vertices of the original graph. The problem occurs in the case of edges, where the information about the original pseudoline used to travel between stops is lost due to creation of merge-line. To be able to determine the edge in the original graph, it is necessary to remember the identity of the original pseudoline that is used to travel through the aggregated edge[4]. When searching the new graph, the transfer detection should be changed as followed:

[4] There can be more such pseudolines; we will therefore remember a list of applicable pseudolines.

Getting in While getting in a pseudoline, it is necessary to detect which of the original pseudolines from the merge-line is just used. First, the departure time of the next service is found – in the aggregated timetable. This is the departure time of merge-line. Now the original pseudoline needs to be found. At this point it is necessary to view the original timetables to determine which original pseudolines are leaving at the found time[5].

Transfer between aggregated pseudolines If the identification of the merge-line on the arrival edge and on the leaving edge is equal, then this is not a transfer. In this case, the waiting time is not counted in.

Real transfer If the identification of the merge-line on the arrival edge and on the leaving edge are not equal, then this could be a transfer. This is not a real transfer if the pseudoline of arrival edge **is contained** in the list of original pseudolines of the merge-line on the leaving edge. In other case, this is a real transfer and the waiting time must be counted in.

Correctness The vertices in the new graph are equal to the vertices in the original one. Every aggregated edge represents the edge in the original graph with equal value. The changed transfer mechanism above ensures, that the transfers between merge-lines are equal to the transfers between original pseudolines. The pseudolines in the original graph can be labeled by the identification of merge-line into which it is aggregated. The path found in the reduced graph will be created by merge-lines corresponding the labels on pseudolines, which creates the path in the original graph.

	# of nodes	# of edges	memory usage	search time
before	1096	8473	429 721B	4s
after	1096	2927	1 182 226B	1.70s
decrement	0%	65%	-175%	58%

Table 1. Comparison of computation over original data on Prague Public Transport Network and after simple network compaction

3.2 Path Aggregation

The latter adjustment builds on the results of the first one. Based on a direct connection between the two stops longer stretch of several consecutive stops. The condition is that only one certain line runs in this sequence of stops, and no other line is connecting or leaving this sequence. Such a sequence of stops can be aggregated into a sort of "pipelines".

[5] This situation is simpler then in the original graph. The departure time needs to be found only once – in the aggregated timetable. Then this time is searched in the original timetables for the match.

Pipeline is an aggregation of a number of services that ensures the same travel time between all the stops in the sequence. In order to guarantee this property, the order of services entering the pipeline must be the same like the order of the services leaving the pipeline. For trams or trolley-buses this property is guaranteed. The problem might occur on buses.

Node is a stop, where the passenger can take a relevant transfer to another service. The transfer is considered to be relevant, if the service can get the passenger to other stop that he was not before. The example of not relevant transfer is a transfer to the same line, but opposite direction. This typically gets the passenger to the stop where he was[6]. By excluding these loops the resulting path will always get shorter.

Starting with a graph where the first adjustment has been already made simplifies the initial situation. In the following, we assume the first adjustment is made. The second adjustment may be done in two phases.

The first phase Pipeline is the edge of the new graph. Vertices, which are outside the pipeline, will form the vertices of the new graph – nodes. Vertices, which are intside the pipeline must hold the following two conditions:

1. The vertex may not have more than two different adjacent vertices[7].
2. Edges that go into the vertex must also go outside. Corresponding input and output edges must bear the same identification of merge-line.

The second phase The nodes are connected by edges representing the pipelines: An edge representing a pipeline between two nodes corresponds to a sequence of vertices in the original graph. This sequence must be of the same merge-line and can not be interrupted by any other node what ensures that there will be no transfer inside the sequence. This sequence is replaced by the new single graph edge. The value of new edge is set to the sum of the values of edges in the original sequence. Merge-line of the new edge is the merge-line of the original edges.

Connection of the path search There will be a reduction of vertices in the new graph. This means that if the shortest path search starts in a vertex, which is not a node in the new graph, it is necessary to find the path to the nearest nodes. This path can be found in the original graph quite easily because of the properties of vertices within pipeline which means that the path can lead up to a maximum of two directions. After a very short search two peripheral nodes will be encountered. Both peripheral nodes will be taken as starting points for search in the new graph. For starting points the initial value of path length estimation will be set to the length of the path from the original vertex to the nearest node – to the starting point. Similarly, if the target is not node.

[6] The opposite direction of the link can get the passenger to the stop, where he was not yet; therefore the nodes should be chosen carefully.

[7] The vertices available in the opposite direction of the oriented edge are also considered to be adjacent here.

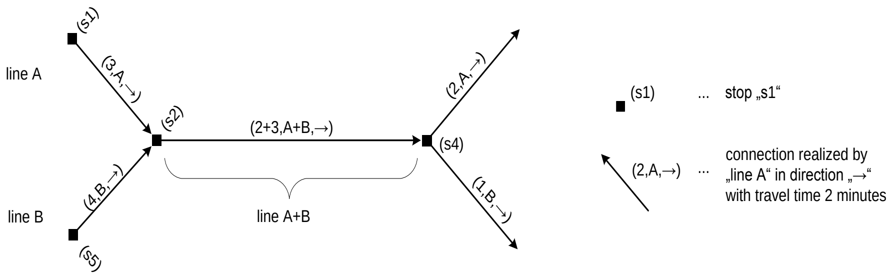


Fig. 3. Creation of pipeline between the nodes.

Mapping the path to the original network Since the latter adjustment is based on the first one, we need the same procedure used for detection of transfers as the first adjustment. The path found in this new graph is made up of nodes and pipelines.

To overcome the reduction of vertices to nodes we can return back to the vertices. To get detailed path in the 2nd level graph it is sufficient to search for path between vertices corresponidng to the nodes being neighbours in the 3rd level graph.

Finally, we need to add the initial segments into the resulting path which we used to get from initial vertex to the initial nodes.

Correctness The sequences of vertices and edges in the graph after first reduction were replaced by the single edge. The value of this edge is equal to the sum of values of the original edges. The mechanism of choice of the vertices inside the pipeline ensures, that the transfer in the vertex inside the pipeline is not relevant for searching the shortest path. Therefore the value of pipeline is equal to the value of the part of any shortest path leading through the sequence of vertices and edges creating the pipeline. The mechanism for detection of transfers is equal to the mechanism in previous reduction step.

	# of nodes	# of edges	memory usage	search time
before	1096	2927	1 182 226B	1.70 s
after	549	1987	1 225 810B	1.03 s
decrement	50%	32%	-4%	39%

Table 2. Comparison of computation over modified data on Prague Public Transport Network and after advanced network compaction

4 Prague Public Transport

As we have only data set the Prague public transport, the example is based on part of this data. The time necessary for the graph search on a sample mobile device (hardware details are mentioned below) is included in the tables in the text. For benchmarking purposes the terminating condition was excluded from the search algorithm. So the values in Tab. 2 are maximal – represent searching entire graph.

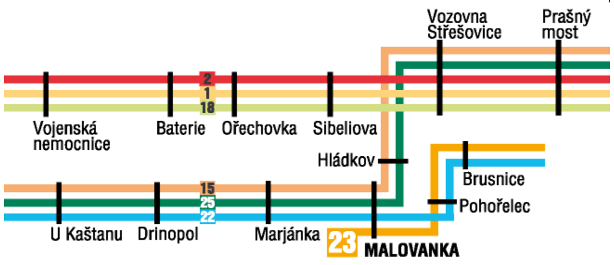


Fig. 4. Sample cutout of Prague public transport (taken from [6])

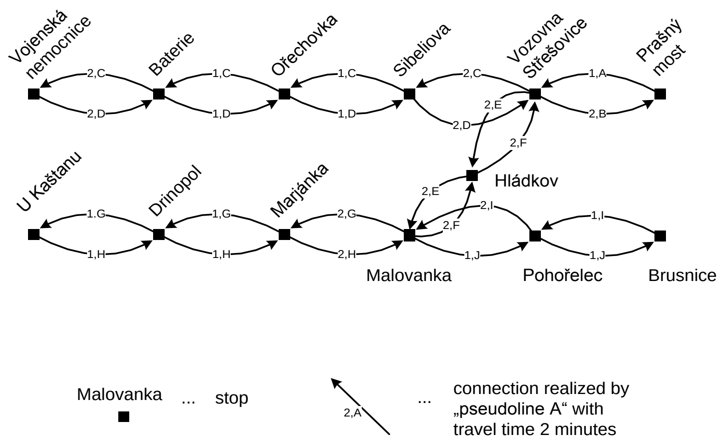


Fig. 5. Sample of Prague public transport after first adjustment.

In our sample case the number of vertices decreased more then two times and number of edges almost seven times. The density of individual lines crossing

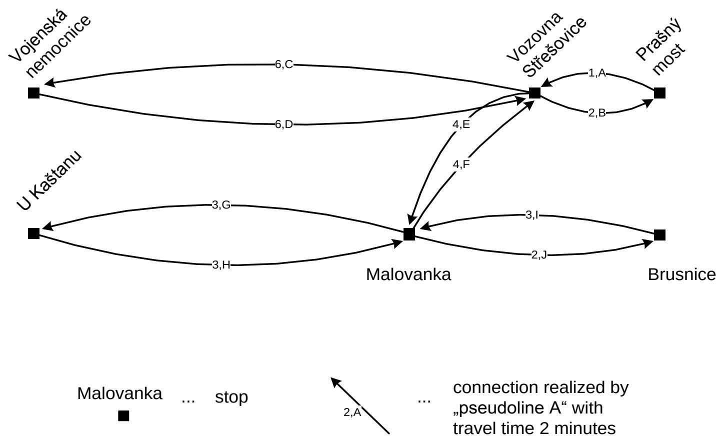


Fig. 6. Sample cutout of Prague public transport after both adjustments.

and irregular placement of street refuges prevent from such significant reduction in full-scale.

	# of nodes	# of edges	memory usage	search time
original graph	1096	8473	429 721B	4s
first adjustment	1096	2927	1 182 226B	1.70s
second adjustment	549	1987	1 225 810B	1.03s
decrement	50%	77%	-185%	74%

Table 3. Comparison of computation over original data on Prague Public Transport Network and after simple network compaction

5 More Opportunities for Graph Reduction

The advantage of the adjustments referred to in this paper is that they do not alter the substance of problem but they only reduce the size of input data. It does not prevent the application of other techniques for reducing the search complexity.

5.1 Highway Hierarchy

One of interesting processes, which could build on referred adjustments is “highway hierarchy” [7]. This method could be used in two ways. We could further reduce the public transport network from the second adjustment to achieve a further acceleration. To bring a noticeable improvement, we must be able to find areas in the graph, with relatively dense traffic inside. These areas should be connected together in a relatively small number of vertices. This approach brings a problem, how to find areas compliant to the "highway hierarchy". In the case of urban public transport we consider finding of compliant areas to bring the significant improvement very difficult. The other approach is to link urban public transport and national or interstate transport networks together using "highway hierarchy".

6 Results

6.1 Memory Consumption

There are three levels of the graph – the original graph, graph after the first and after both adjustments. For the search itself we need to hold the two highest levels in memory. The initial segments are searched in the second level graph when the initial or target vertex is not a node. Otherwise we start directly in the third level. The path between nodes is searched in the third level graph. It is not necessary to hold the original graph in main memory. The size of the graph levels is decreasing. So the memory consumed by the graphs themselves would not exceed the double of original value.

The data of the original timetables are needed for the changed transfer detection in the modified graph. The aggregated timetables are needed to determine the waiting time. The resulting memory consumption depends on the representation of timetables. If the time tables are maintained only for the initial stop of the pseudoline, the number of merged time tables will depend on the number of pipeline. Each pipeline has its own merge-line and its own timetable.

In the current implementation all three levels of the graph are kept in the main memory. This reflects to the referred memory consumption. Our aim in future is to choose an appropriate representation of time tables and to minimize the memory needed to store the additional structures.

6.2 The Hardware

Our reference hardware is HTC X7500 having Intel XScale 624MHz processor and 128 megabytes RAM, 65 megabytes of free. Portable devices, on which the current implementation is mostly targeted, used to be equipped by secondary Flash-type memory. The writing to this kind of memory is usually several times slower then the reading. The advantage of the mentioned adjustments is that they do not require frequent writing into the secondary memory. Another specific feature is that the access to various locations in the memory is not as complicated as for example for hard drives. For this reason it is possible to hold the original graph in the secondary memory without slowdown noticeable to the user.

7 Conclusions

The methods introduced here speed up searching for optimal path between two given points in public transport network. They are intended to be used on mobile hardware where the original computation took several seconds what is for many users not acceptable. Currently the computation takes in the worst case only one second what is acceptable for most users. We are currently considering other options of improvement of the search algorithm.

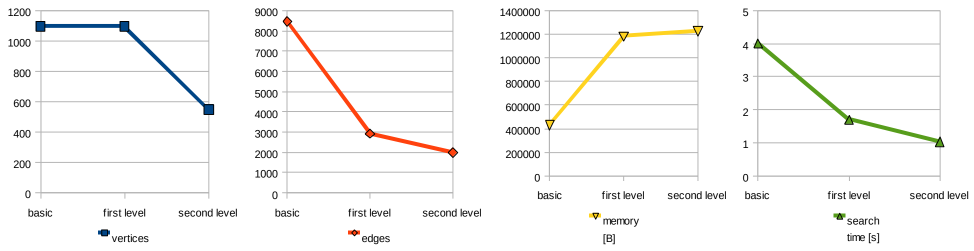


Fig. 7. Shift in the number of vertices, edges, memory consumption and computing speed.

References

1. Sanders, P., Schultes, D.: Engineering fast route planning algorithms. In Demetrescu, C., ed.: WEA. Volume 4525 of Lecture Notes in Computer Science, Springer (2007) 23–36
2. Weihe, K.: Covering trains by stations or the power of data reduction. In Battiti, R., Bertossi, A.A., eds.: Proceedings of "Algorithms and Experiments" (ALEX98). (1998) 1–8
3. Liebers, A., Weihe, K.: Recognizing bundles in time table graphs - a structural approach. In Näher, S., Wagner, D., eds.: Algorithm Engineering. Volume 1982 of Lecture Notes in Computer Science., Springer (2000) 87–98
4. Demetrescu, C., Italiano, G.F.: Experimental analysis of dynamic all pairs shortest path algorithms. ACM Transactions on Algorithms **2**(4) (2006) 578–601
5. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1** (1959) 269–271
6. Dopravní podnik hlavního města Prahy: (Užitečná dopravní schémata) `http://www.dpp.cz/uzitecna-dopravni-schemata/`.
7. Delling, D., Sanders, P., Schultes, D., Wagner, D.: Highway Hierarchies Star. Technical report, ARRIVAL Project (2006) work presented at 9th DIMACS Challenge on Shortest Paths.

From Web Pages to Web Communities

Miloš Kudělka[1], Václav Snášel[1], Zdeněk Horák[1], and Aboul Ella Hassanien[2]

[1] VSB Technical University Ostrava, Czech Republic
milos.kudelka@inflex.cz, {vaclav.snasel, zdenek.horak.st4}@vsb.cz
[2] Faculty of Computer and Information, Information Technology Department, Cairo University, Egypt
abo@cba.edu.kw

Abstract. In this paper we are looking for a relationship between the intent of Web pages, their architecture and the communities who take part in their usage and creation. From our point of view, the Web page is entity carrying information about these communities and this paper describes techniques, which can be used to extract mentioned information as well as tools usable in analysis of these information. Information about communities could be used in several ways thanks to our approach. Finally we present an experiment which illustrates the benefits of our approach.

Keywords: Web community, Web site, Web pattern, genre

1 Introduction

Metaphor: A Web page is like a family house. Each of its parts has its sense, determined by a purpose which it serves. Every part can be named so that everybody imagines approximately the same thing under that name (living room, bathroom, lobby, bedroom, kitchen, balcony). In order that the inhabitants may orientate well in the house, certain rules are kept. From the point of view of these rules, all houses are similar. That is why it is usually not a problem e.g. for first time visitors to orientate in the house. We can describe the house quite precisely thanks to names. If we add information about a more detailed location such as sizes, colors, equipment and further details to the description, then the future visitor can get an almost perfect notion of what he will see in the house when he comes in for the first time. We can also approach the description of a building other than a family house (school, supermarket, office etc.). Also in this case the same applies for visitors and it is usually not a problem to orientate (of course it does not always have to be the case, as well as bad Web pages there are also bad buildings).

In the case of buildings, we can naturally define three groups of people, which are somehow involved in the course of events. The first group are the people defining the intent and the purpose (those who pay and later expect some profit), the second one are those who construct the building (and are getting paid for it) and the third group are "users" of the building. These groups fade into another and change as society and technology evolve.

As we describe in the subsequent text, the presented metaphor can - up to certain point - serve as an inspiration to seize the Web pages content and also the whole Web environment.

This text is organized as follows. In the second section we describe the Web page from the view of groups of people sharing the Web page existence. The third section describes tools and techniques required for our experiment. In particular our own Pattrio method, which is designed to detect Design patterns within Web pages, and FCA used for clustering. In the fourth section we describe experiment dealing with Web site description. The last section contains paper recapitulation and focuses on possible directions of further research.

2 From Web pages to Web communities

Every single Web page (or group of Web pages) can be perceived from three different points of view. When considering the individual points of view we were inspired by specialists on Web design ([29]) and on the communication of humans with computers ([6]). These points of view represent the views of three different groups of communities who take part in the formation of the Web page (fig. 1).

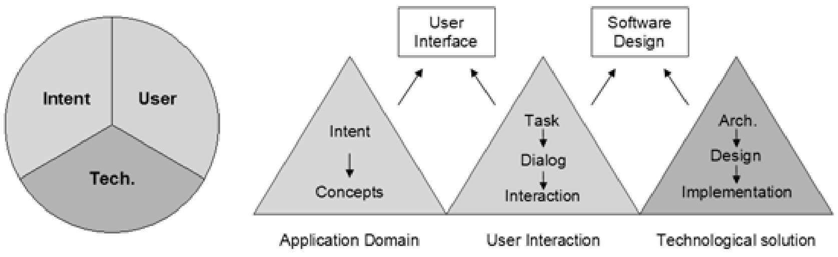


Fig. 1. Views of three different groups

(1) The first group are those whose intention is that the user finds what he expects on the Web page. The intention which the Web page is supposed to fulfill is consequently represented by this group. For the sake of clarity, we can say that this group is often represented by Web site owners. (2) The second group are developers responsible for the creation of the Web page. They are therefore consequently responsible for fulfilling the goals of the two remaining groups. (3) The third group are users who work with the Web page. This group consequently represents how the Web page should appear outwardly to the user. It is important that this performance satisfies a particular need of the user.

As an example we can mention blogs. The first community are the companies, which offer an environment and technological background for blog authors and to

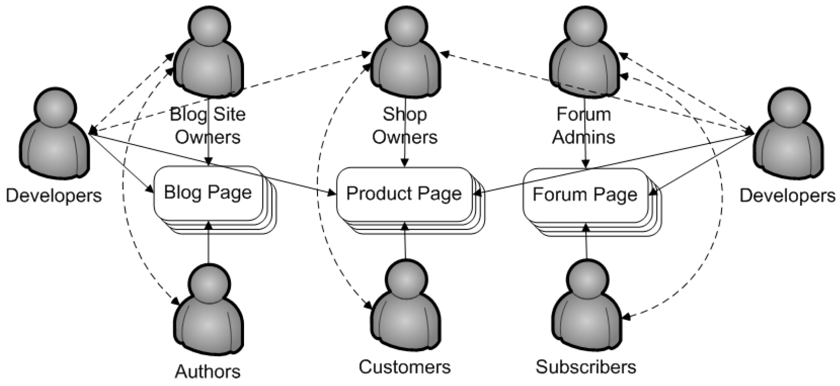


Fig. 2. Social network around Web pages

some extent they also define the formal aspects of blogs. The second community are the developers who implement the task given by the previous group. The visible attribute of this group is that they – to a certain degree – share their techniques and policies. The third group consists of blog authors (in the sense of content creation). They influence the previous two groups retroactively. The second example can be the product pages - the intention of the e-shop is to sell items (concretely to have Web pages where you can find and buy the products), the intention of the developers is to satisfy the e-shop owners as well as the Web page visitors. The intention of the visitors is to buy products, so they expect clearly stated and well-defined functionality. From this point of view, the web pages are elements around which the social networks are formed (fig. 2). For further details and references, please see [1] and [15] (which considers also the aspect of network evolution).

Under the term *Web community* we usually think of a group of related Web pages, sharing some common interests (see [28], [20], [21]). As a Web community we may also consider Web site or groups of Web sites, on which people with common interests interact. It is apparent, that all three aforementioned groups participate in the Web page life cycle. The evolution of a page is directly or indirectly controlled by these groups. As a consequence, we can understand the Web page as a projection of interaction among these three groups. The analysis of the page content may uncover significant information, which can be used to assign the Web page to a Web community.

3 Tools and techniques

Our aim is to automatically discover such information about Web pages, that comes out of intentions of particular groups. Using these information we can find the relations between the communities and describe them (on the technical level). The key element for Web page description is the name of the object, which represents the intention of the page. It can be "Home page", "Blog" or "Product Page". In the detailed description we can distinguish, for example, between "Discussion", "Article" or "Technical Features". We can also use more general description, such as "Something to Read" or "Menu" (see [14]).

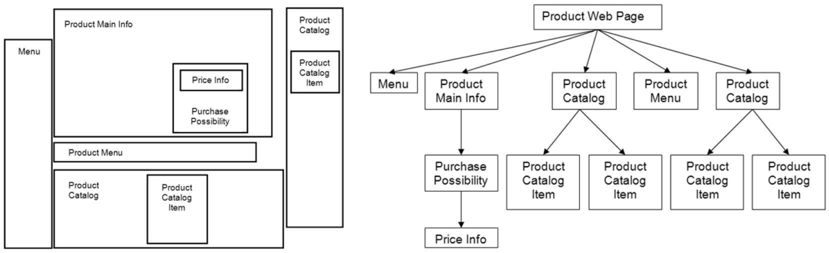


Fig. 3. Product page scheme (a), (b)

The first group of intentions represents so-called Genre (see [5]). The second group is very close to Web Design Patterns [30]. Figure 3 contains schematically depicted product Web page with hierarchy of solved tasks (each task represents one particular intention). The ability to discover aforementioned elements (Genres and Web design patterns) is required to obtain the Web page description (and consequently also the intentions represented by mentioned communities).

Genre is a taxonomy that incorporates the style, form and content of a document which is orthogonal to topic, with fuzzy classification to multiple genres [4]. In the same paper are described existing classifications. Regarding these classifications there are many approaches on genre identification methods. The goal of paper [11] is to analyze home page genres (personal home page, corporate home page or organization home page). In paper [7] authors have proposed a flexible approach for Web page genre categorization. Flexibility means that the approach assigns a document to all predefined genres with different weights. In [9] paper, there is described a set of experiments to examine the effect of various attributes of Web genre on the automatic identification of the genre of Web pages. Four different genres are used in the data set (FAQ, News, E-Shopping and Personal Home Pages).

3.1 Pattrio method

Design patterns describe proven experience of repeated problem solving in the area of software solution design. While the design patterns have been proven in real projects, their usage increases the solution quality and reduces the time of their implementation. Good examples are also the so called Web design patterns, which are patterns for design related to the Web. Even in this area, the patterns are getting quite common (they are collected and published in the form of printed or Internet catalogues, e.g. see [29], [30]).

We have designed our own Pattrio method used for the detection of Web design pattern instance in web pages. In Pattrio method we work with 24 patterns (mostly e-commerce and social domain). Pattrio method is based on analysis of technical (architectural) and semantical attributes of solutions of the same tasks in the environment of Web, for details see [13], [14].

Detection algorithm In the context of our approach, there are elements with semantic contents (words or simple phrases and data types) and elements with importance for the structure of the web page where the Web pattern instance can be found (technical elements). The rules are the way that individual elements take part in the Web pattern display. While defining these rules, we have been inspired by the Gestalt principles (see [27]). We are using four rules based on these principles. The first one (proximity) defines the acceptable measurable distances of individual elements from each other. The second one (closure) defines the way of creating of independent closed segments containing the elements. One or more segments then create the Web pattern instance on the web page. The third one (similarity) defines that the Web pattern includes more related similar segments. The forth one (continuity) defines that the Web pattern contains more various segments that together create the Web pattern instance. The relations among Web patterns can be on various levels similar as classes in OOP (especially simple association and aggregation).

The basic algorithm for detection of Web patterns then implements the preprocessing of the code of the HTML page (only selected elements are preserved e.g. block elements as table, div, lines, etc.), segmentation and evaluation of rules and associations. The result for the page is the score of Web patterns that are present on the page. The score then says what is the relevance of expecting the Web pattern instance on the page for the user.

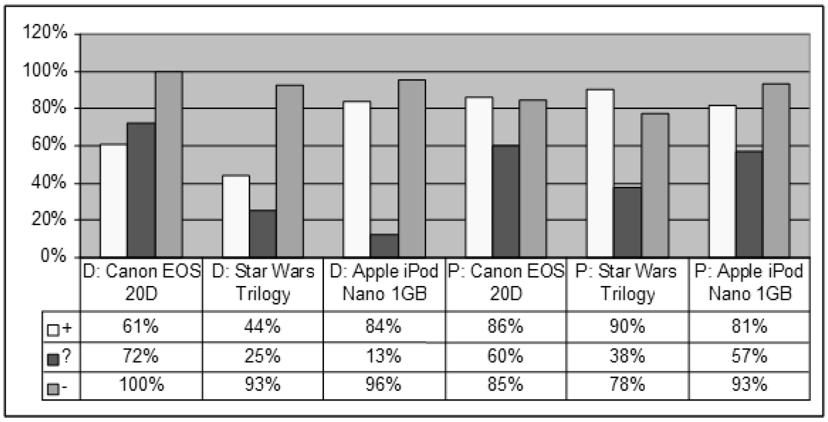
The accuracy of our method is about 80% (see [12]). Figure 4 shows the accuracy of Pattrio method for tree selected products (Apple iPod Nano 1GB, Canon EOS 20D, Star Wars Trilogy film) and for the *Discussion* pattern and the *Purchase possibility* pattern. We used only the first 100 pages for each product. We manually and using Pattrio method evaluated the pages using a three-degree scale:

+ Page contains required pattern.
? Unable to evaluate results.
- Page do not contain required pattern.

Then we compared these evaluations. For example the first value 61% expresses the method accuracy for the pages with Canon EOS 20D product where there was a discussion.

3.2 Formal Concept Analysis

As one of the suitable tools for analyzing this kind of data we consider Formal concept analysis. When preprocessing Web pages we often cannot clearly state



	D: Canon EOS 20D	D: Star Wars Trilogy	D: Apple iPod Nano 1GB	P: Canon EOS 20D	P: Star Wars Trilogy	P: Apple iPod Nano 1GB
□+	61%	44%	84%	86%	90%	81%
■?	72%	25%	13%	60%	38%	57%
▨-	100%	93%	96%	85%	78%	93%

Fig. 4. Accuracy of Pattrio method for detection of *Discussion* and *Purchase Possibility* patterns - percentage of agreement between human and Pattrio method evaluation on sets of Web pages returned for different search queries

the presence of an object in the page content. We are able to describe the amount of its presence at some scale and this information can be captured using fuzzy methods and analyzed using a fuzzy extension of Formal Concept Analysis ([3]). But since we are dealing with a large volume of data ([8]) and a very imprecise environment, we should consider several practical issues, which have to be solved prior the first applications. Methods of matrix decomposition have succeeded in reducing the dimensions of input data (see [26] for application connected with Formal concept analysis and [18], [17] for overview).

Formal concept analysis (shortly FCA, introduced by **Rudolf Wille** in 1980) is well known method for object-attribute data analysis. The input data for FCA we call **formal context** C, which can be described as $C = (G, M, I)$ – a triplet consisting of a set of objects G and set of attributes M, with I as relation of G and M. The elements of G are defined as objects and the elements of M as attributes of the context.

For a set $A \subseteq G$ of objects we define $A^{'}$ as the set of attributes common to the objects in A. Correspondingly, for a set $B \subseteq M$ of attributes we define $B^{'}$ as the set of objects which have all attributes in B. A **formal concept** of the context (G, M, I) is a pair (A, B) with $A \subseteq G$, $B \subseteq M$, $A^{'} = B$ and $B^{'} = A$. $\mathcal{B}(G, M, I)$ denotes the set of all concepts of context (G, M, I) and forms a complete lattice (so called **Gallois lattice**). For more details, see [10].

4 Experiment

For the need of our experiment we have implemented a Web application with user interface connected to the API of different search engines (google.com, msn.com, yahoo.com and the Czech search engine jyxo.cz above all). Users from a group of students and teachers of high schools and our university were using this application for more than one year to search for ordinary information. We have not influenced the process of searching in any way. The purpose of this part of experiment was to view the World Wide Web using the perspective of normal users (as the search engines play key role in World Wide Web navigation). In the end we have obtained dataset with more than 115,000 Web pages. After clean up, 77,850 unique Czech pages remained. For every single Web page we have performed the detection of sixteen objects. The page did not have to contain any object, as well as it may have contained 16 objects (Price information, Purchase possibility, Special offer, Hire sale, Second hand, Discussion and comments, Review and opinion, Technical features, News, Enquire, Login, Something to read, Link group, Price per item, Date per item, Unit per item). We have used such preprocessed dataset for following experiment.

In the experiment we have tried to visualize the structure and relations of Web sites (and as a result also Web communities) referring to one specific topic. As an input we have used the list of domains created in the previous experiment. Only Web sites with more than 20 pages in the dataset have been taken into consideration. Each domain is accompanied by detected objects. This list is transformed into a binary matrix and considered as a formal context. Using methods of FCA we have computed a concept lattice which can be seen on figure 5. The resulting matrix has 516 rows (objects) and 16 columns (attributes), computed concept lattice contains 378 concepts.

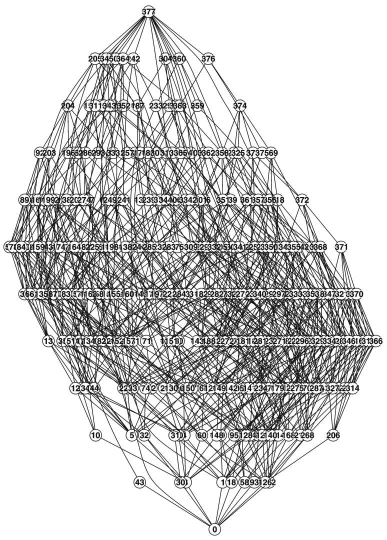


Fig. 5. Lattice calculated from whole dataset

From the computed lattice we have selected a sub-lattice containing 18 Web sites dealing with cell phones. Only 5 attributes have been selected and the visualization was created in a slightly different manner (see figure 6 and attached legend). Each node of the graph corresponds to one formal concept. To increase the visualization value, the attributes are represented by icons and the set of objects (Web sites) is depicted using small filled/empty squares in the lower part. It can be easily seen that using created visualization we can think of dividing the whole set of Web sites into two groups - the first one contains sites where users are enabled to buy cell phones and the second one where the users are allowed to have a discussion. The illustrated division is in the soft sense only — one may think of concept nr. 8 as being part of the shopping group also. Web sites presented in higher levels of lattice are considered in more specific context. Deeper insight gives you more detailed information about Web site structures and relations.

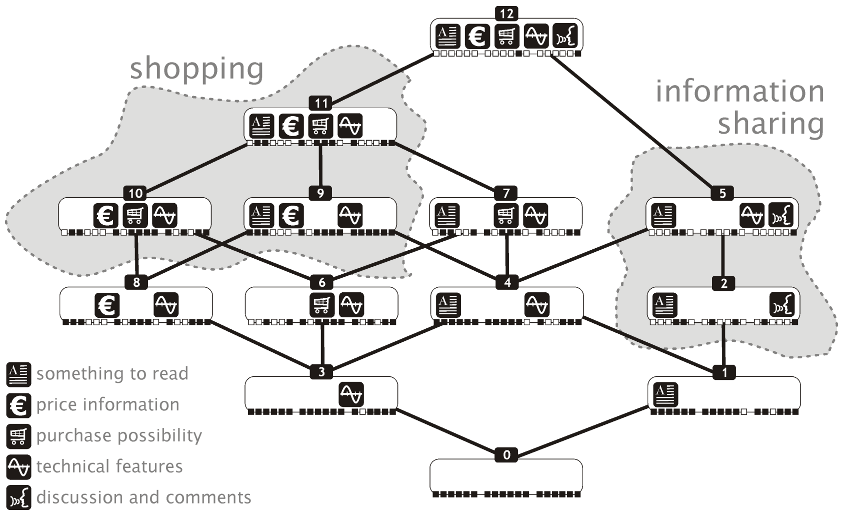


Fig. 6. Part of lattice

The concept lattice forms a graph, which can be interpreted as an expression of relation between different Web sites. As a consequence, it describes the relation between different Web communities, because behind the shopping-related domains we can see the group of users interested in buying cell phones and behind the information–sharing pages we see the community of users interested only in the technical aspects, features of cell phones and their discussing.

5 Conclusions and future work

In this paper we have described three kinds of social groups which take part in Web page creation and usage. We distinguish these groups using their relation to the Web page - whether they define the intent of the page, whether they create the page or whether they use the page. By using this analysis we can follow the evolution of the communities and observe the expectancies, rules and behavior they share. Obtained information can be surely used to improve the searching process. From this point of view, Web 2.0 is only a result of the existence and interaction of these social groups.

Our experiment shows that if we focus ourselves on Web sites and the Web page content they provide, we can ask interesting questions. These questions may bear upon the Web sites' similarity and the similarity of social groups involved with these pages. For us this shows the direction of further research in which we will investigate answers to these questions in more detail.

References

1. L. Adamic, E. Adar: How to search a social network, Journal Social Networks, vol. 27, pp. 187–203 (2005)
2. Ch. Alexander: A Pattern Language: Towns, Buildings, Construction, Oxford University Press, New York (1977)
3. R. Belohlavek, V. Vychodil: What is a fuzzy concept lattice, Proceedings of the CLA, 3rd Int. Conference on Concept Lattices and Their Applications, pp. 34–45 (2005)
4. E. S. Boese: Stereotyping the web: Genre classification of Web documents, Colorado State University (2005)
5. E. S. Boese, A. E. Howe: Effects of web document evolution on genre classification, Proceedings of the 14th ACM international conference on Information and knowledge management, pp. 632–639 (2005)
6. J. O. Borchers: Interaction Design Patterns: Twelve Theses, Workshop, The Hague, vol. 2 (2000)
7. J. Chaker, O. Habib: Genre Categorization of Web Pages, Proceedings of the Seventh IEEE International Conference on Data Mining Workshops, pp. 455–464 (2007)
8. R. J. Cole, P. W. Eklund: Scalability in Formal Concept Analysis, Computational Intelligence, vol. 15, pp. 11–27 (1999)
9. L.Dong, C. Watters, J. Duffy, M. Shepherd: An Examination of Genre Attributes for Web Page Classification, Proceedings of the Proceedings of the 41st Annual Hawaii International Conference on System Sciences, pp. 133–143 (2008)
10. B. Ganter, R. Wille: Formal Concept Analysis: Mathematical Foundations, Springer-Verlag, New York (1997)
11. A. Kennedy, M. Shepherd: Automatic Identification of Home Pages on the Web, Proceedings of the 38th Hawaii International Conference on System Sciences (2005)
12. J. Kocibova, K. Klos, O. Lehecka, M. Kudelka, V. Snasel: Web Page Analysis: Experiments Based on Discussion and Purchase Web Patterns, Web Intelligence and Intelligent Agent Technology Workshops, pp. 221–225 (2007)

13. M. Kudelka, V. Snasel, O. Lehecka, E. El-Qawasmeh: Semantic Analysis of Web Pages Using Web Patterns, Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence, pp. 329–333 (2006)
14. M. Kudelka, V. Snasel, O. Lehecka, E. El-Qawasmeh, J. Pokorny: Web Pages Reordering and Clustering Based on Web Patterns, SOFSEM 2008, pp. 731–742 (2008)
15. R. Kumar, J. Novak, A. Tomkins: Structure and Evolution of Online Social Networks, Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 611–617 (2006)
16. D. Lee, O. R. Jeong, S. Lee: Opinion mining of customer feedback data on the web, Proceedings of the 2nd international conference on Ubiquitous information management and communication, pp. 230–235 (2008)
17. D. Lee, H. Seung.: Learning the parts of objects by non-negative matrix factorization, Nature, vol. 401, pp. 788–791 (1999)
18. T. Letsche, M. W. Berry, S. T. Dumais.: Computational methods for intelligent information access, Proceedings of the 1995 ACM/IEEE Supercomputing Conference (1995)
19. H. Y. Limanto, N. N. Giang, V. T. Trung, J. Zhang, Q. He, N. Q. Huy: An information extraction engine for web discussion forums, International World Wide Web Conference, pp. 978–979 (2005)
20. T. Murata: Discovery of User Communities from Web Audience Measurement Data, Web Intelligence 2004, pp. 673–676 (2004)
21. T. Murata, K. Takeichi: Discovering and Visualizing Network Communities, Web Intelligence/IAT Workshops 2007, pp. 217–220 (2007)
22. Z. Nie, J. R. Wen, W. Y. Ma: Object-level Vertical Search, Third Biennial Conference on Innovative Data Systems Research, pp. 235–246 (2007)
23. Z. Pawlak: Rough Sets: Theoretical Aspects of Reasoning about Data, Kluwer Academic Publishing (1991)
24. M. A. Rosso: User-based identification of Web genres. JASIST (JASIS) 59(7), pp. 1053–1072 (2008)
25. S. Schmidt, H. Stoyan: Web-based Extraction of Technical Features of Products, Beiträge der 35. Jahrestagung der Gesellschaft für Informatik, pp. 256–261 (2005)
26. V. Snasel, M. Polovincak, H. M. Dahwa, Z. Horak: On concept lattices and implication bases from reduced contexts, Supplementary Proceedings of the 16th International Conference on Conceptual Structures, ICCS 2008, pp. 83–90 (2008)
27. J. Tidwell: Designing Interfaces: Patterns for Effective Interaction Design, O'Reilly, pp. 0–596 (2005)
28. M. Toyoda, M. Kitsuregawa: Creating a Web community chart for navigating related communities, Hypertext 2001, pp. 103–112 (2001)
29. D. K. Van Duyne, J. A. Landay, J. I. Hong: The Design of Sites: Patterns, Principles, and Processes for Crafting a Customer-Centered Web Experience, Addison-Wesley Professional (2003)
30. M. Van Welie: Pattern Library for Interaction Design. www.welie.com, (last access 2008-08-07)
31. S. Zheng, R. Song, J. R. Wen: Template-independent news extraction based on visual consistency, In Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, pp. 1507–1513 (2007)

Compression of the Stream Array Data Structure *

Radim Bača and Martin Pawlas

Department of Computer Science, Technical University of Ostrava
Czech Republic
{radim.baca,martin.pawlas}@vsb.cz

Abstract. In recent years, many approaches to XML twig pattern query (TPQ) processing have been developed. Some algorithms are supported by a stream abstract data type.
Stream is an abstract data type usually implemented using inverted list or special purpose data structure. In this article, we focus on an efficient implementation of a stream ADT. We utilize features of a stream ADT in order to implement compressed stream array and compare it with regular stream array.

Key words: Stream ADT, XML

1 Introduction

In recent years, many approaches to XML twig pattern query (TPQ) processing have been developed. Indexing techniques for a XML document structure have been studied extensively and works such as [11, 10, 8, 1, 3, 9, 4, 5] have outlined basic principles of streaming scheme approaches. Node of an XML tree is labeled by a labeling scheme [11, 10] and stored in a stream array. Streaming methods usually use the XML node tag as a key for one stream. Labels retrieved for each query node tag are then merged by some type of XML join algorithm such as structural join [1] or holistic join [3].

We can use also relational databases in order to store and query labeled XML tree, however relational query processor join operation is not designed for this purpose. Due to this fact, XML joins outperform significantly relational query processors [11].

XML joins are based on a stream abstract data type which usually implemented using inverted list or special purpose data structure. In this article, we focus on an efficient implementation of a stream ADT. We utilize features of a stream ADT in order to implement compressed stream array and compare it with regular stream array. We utilize fast fibonacci encoding and decoding algorithms in order to achieve maximal efficiency of the result data structure. Moreover, our compressed stream array data structure allows us to store variable length labels such as Dewey order without storage overhead.

In Section 2, we describe XML model. Section 3 introduce the stream abstract data type and outline persistent stream array and its compression. In Section 4, we describe different compression techniques applied on a block of a stream array. Section 5 describes some experimental results.

* Work is partially supported by Grants of GACR No. 201/09/0990.

2 XML model

An XML document can be modeled as a rooted, ordered, labeled tree, where every node of the tree corresponds to an element or an attribute of the document and edges connect elements, or elements and attributes, having a parent-child relationship. We call such representation of an XML document an *XML tree*. We can see an example of the XML tree in Figure 1. We use the term 'node' to define a node of an XML tree which represents an element or an attribute.

The labeling scheme associates every node in the XML tree with a label. These labels allow to determine structural relationship between nodes. Figures 1(a) and 1(b) show the XML tree labeled by *containment labeling scheme* [11] and *dewey order* [10], respectively.

The containment labeling scheme creates labels according to the document order. We can use simple counter, which is incremented every time we visit a start or end tag of an element. The first and the second number of a node label represent a value of the counter when the start tag and the end tag are visited, respectively. In the case of dewey order every number in the label corresponds to one ancestor node.

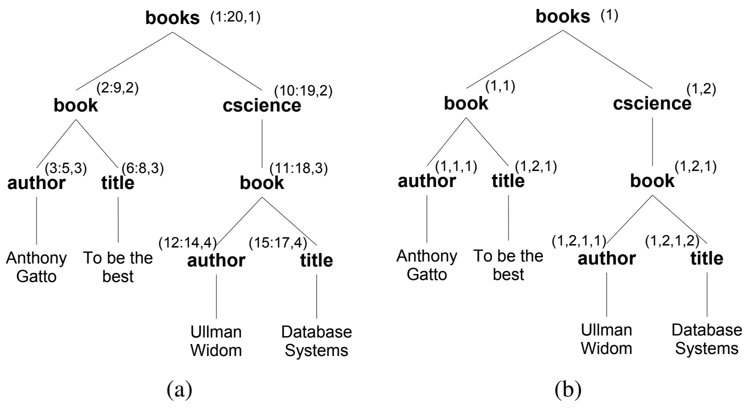


Fig. 1. (a) Containment labeling scheme (b) Dewey order labeling scheme

3 Stream ADT

Holistic approaches use an abstract data type (ADT) called a *stream*. A stream is an ordered set of node labels with the same *schema node label*. There are many options for creating schema node labels (also known as *streaming schemes*). A cursor pointing to the first node label is assigned to each stream. We distinguish the following operations of a T stream: *head(T)* – returns the node label to the cursor's position, *eof(T)* – returns true iff the cursor is at the end of T, *advance(T)* – moves the cursor to the next node label. Implementation of the stream ADT usually contains additional operations: *openStream(T)* – open the stream T for reading, *closeStream(T)* - close the stream.

The Stream ADT is often implemented by an inverted list. In this article we describe simple data structure called *stream array*, which implement stream ADT. We test different compression techniques in order to decrease number of disk accesses. It also allows us to store variable length vectors efficiently.

3.1 Persistent stream array

Persistent stream array is a data structure, which uses common architecture, where data are stored in blocks on secondary storage and main memory cache keeps blocks read from the secondary storage. In Figure 2 we can see an overview of such architecture. Cache uses the least recently used (LRU) schema for a selection of cache blocks [7].

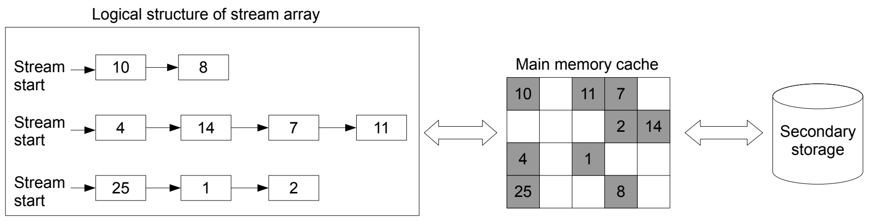


Fig. 2. Overview of a persistent architecture

Each block consists of an array of tuples (node labels) and from a pointer to the next block in a stream. Pointers enable dynamic character of the data structure. We can easily insert or remove tuples from the blocks without time-consuming shift of all items in a data structure. Blocks do not have to be fully utilized, therefore we also keep the number of tuples stored in each block.

Insert and delete operations We briefly describe the insert and delete operations of the stream array in order to see how the data structure is created. In Algorithm 1 we can observe how a label is inserted. $B.next$ is a pointer to the next block in the stream. We try to keep higher utilization of blocks by using similar split technique used by B$^+$tree [6], where we create three 66% full blocks of two full block if possible.

Delete operation is very similar to insert. We process merge of blocks in a case that their utilization is bellow a threshold. However, this operation is out of scope of this article.

3.2 Compressed stream array

There are two reasons for a stream array compression. The first advantage is that we can decrease the size of the data file and therefore decrease number of disk accesses. Of

Algorithm 1: Insert l_T label into the stream array

```
1  Find the block $B$ where the $l_T$ label belongs;
2  if $B$ is full then
3      if block $B.next$ is full $\vee$ $B.next$ does not exist then
4          Create three blocks from $B$ and $B.next$ (if $B.next$ exist);
5          Find the right block and insert $l_T$;
6      else
7          Shift some items from $B$ to the $B.next$;
8          Insert $l_T$ into $B$;
9      end
10 else
11     Insert $l_T$ into $B$;
12 end
```

course, there is an extra time spend on a compression and decompression of data. The compression and decompression time should be lower or equal to time saved having less disk accesses. As a result compression algorithm should be fast and should have good compression ratio. We describe different compression algorithms in Section 4.

The second advantage is that we can store variable length tuples. Tuples in a regular stream block are stored in a array with fixed items' size. The items' size has to be equal to the longest label stored in the stream array and we waste quite a lot of space in this way. Compressed stream block do not use array of items in the block but the byte array where the tuples are encoded.

The stream array has a specific feature which enables efficient compression. We never access items in one block randomly during the stream read. Random access to a tuple in the block may occur only during the stream open operation, but the stream open is not processed very often. Therefore, we can keep the block items encoded in the byte array and remember only the actual cursor position in the byte array. The cursor is created during the stream open and it also contains one tuple, where we store encoded label of the current cursor position. Each label is encoded only once during the *advance(T)* operation. The *head(T)* operation only returns the encoded tuple assigned to cursor. Using this schema we keep data compressed even in the main memory and have to have only one decompressed tuple assigned to each opened stream.

4 Block Compression

In following chapters we will describe compression algorithms implemented during our tests and also we will show examples of these algorithms.

4.1 Variable length tuple

This compression is only based on fact that we can store variable length tuple. It is done by saving dimension length with each tuple.

Example 41 *Let us have these two tuples:* $\langle 1, 2\rangle$ *and* $\langle 1, 2, 3, 7\rangle$. *When using this compression they will occupy 6×4 B + 2 B for dimension length for these two tuples. If we use regular stream array without supporting variable tuple length we will have to align first tuple, so it will look like* $\langle 1, 2, 0, 0\rangle$ *and these two tuples will occupy 8×4 B.*

4.2 Fibonacci coding

This kind of compression is based on Fibonacci coding of number. Because each dimension of tuple contain only non negative number we can use Fibonacci coding.

Example 42 *Let us have a tuple* $\langle 1, 2, 3, 7\rangle$. *After encoding the tuple will be stored as a sequence of bits* 11011001101011, *which occupy 2 B instead of original 24 B (each dimension is 4 B length).*

The problem for this compression technique might be when tuple contains large numbers and then compression of the tuple will take more time, because the number is encoded bit-by-bit. Due to this fact we used the the fast Fibonacci decompression algorithm, which is described in more details in [2]. This decompression algorithm is faster because it is working with whole bytes.

4.3 Compression based on reference item

Tuples in a stream array are sorted and we can use this feature to compress a tuple with knowledge of his ancestor.

Common prefix compression Common prefix compression is based on idea of Run Length Encoding (RLE). Usually ancestor of actual compressing tuple is very similar and therefore we do not have to store every dimension.

Example 43 *Let us have these tuples:* $\langle 1, 2, 3, 7, 9, 7\rangle$, $\langle 1, 2, 3, 7, 5, 6, 7\rangle$, $\langle 1, 2, 3, 7, 7, 0, 0, 7\rangle$. *First tuple in the block cannot be compressed, because there is no ancestor. Second tuple have to store only 3 dimensions and third one have to store last 4 dimensions. The result after compression looks like:* $0 - \langle 1, 2, 3, 7, 9, 7\rangle$, $4 - \langle 5, 6, 7\rangle$, $4 - \langle 7, 0, 0, 7\rangle$, *where the first number says how many dimensions are common. In this example we saved 28 B (original size is 23×4 B, compressed size is (13+3)×4 B).*

Fibonacci coding with reference item The Fibonacci code is designed for a small numbers. However, numbers in the case of containment labeling scheme grows rapidly. In this case, the Fibonacci code becomes inefficient and compression does not work appropriately. In order to keep the numbers small we subtract each tuple with its previous tuple.

Example 44 *Let us have these two tuples:* $\langle 1000, 200, 300, 7\rangle$ *and* $\langle 1005, 220, 100, 7\rangle$. *From this example we see that we can subtract first 2 dimensions. After subtraction we will have* $\langle 1000, 200, 300, 7\rangle$ *and* $\langle 5, 20, 100, 7\rangle$, *which are encoded faster and also occupy less space.*

5 Experimental results

In our experiments[2], we used XMARK [1] data collection and we generated labels for two different labeling schemes: containment labeling scheme with fixed size of labels and dewey order labeling scheme with variable dimension length. We tested scalability of the compression schemes on different collection sizes. We provide test for XMARK collections containing approximately 200k, 400k, 600k, 800k and 1000k labels. Each collection contains 512 streams.

The stream array and all compression algorithms were implemented in C++. We created one persistent stream array for each collection of labels. We provide set of tests, where we simulate real work with the stream array and measure the influence of the compression. For each test we randomly selected 100 streams and read them until the end. Test is processed with a cold cache. During tests we measured file size, query time and Disk Access Cost (DAC). Query time include time interval needed for opening of each randomly selected stream and his reading until the end. DAC is equal to the number of disk accesses during the query processing.

5.1 Fixed dimension length

In Figure 3(a) we can see that file size is same for the block without compression and for the block which support storing variable tuple dimensionality. There is small difference, but it is only because of supporting variable length of dimension. As we can see in Figure 3(a) the regular Fibonacci compression can save us about 25 %. Due to the fact, that the labels values are very close, we can achieve significantly better results in the case of Fibonacci compression using reference tuple. This kind of compression can save about 50 % compared to the regular stream array. Common prefix compression saved us only about 10 %.

Even thought that the compression ratio is good, the query time for compressed stream array is a little bit worse than for regular stream array as you can see in Figure 3(b). Disk access cost that we save using the compression is not sufficient in this case and it is less than the time spend on decompression.

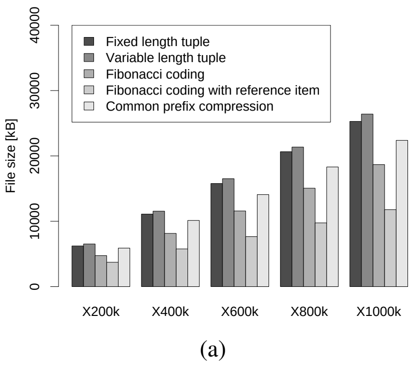
5.2 Variable dimension length

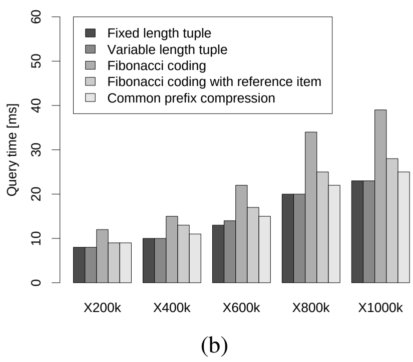
If collection data contains tuples with variable dimension length we can save from 55 % (only by using block which support variable dimension length) up to 85 % (for Fibonacci compression with reference item) of file size when comparing to regular stream array.

The query time of the compressed stream array is always smaller for every implemented compression technique than query time of regular stream array as you can see in Figure 4(b). The Fibonacci compression has the best result for this data collection, with or without reference tuple. The results are comparable because the labels' numbers do not grow so quickly in the case of dewey order labeling scheme.

[2] The experiments were executed on an Intel® Celeron ®D 356 - 3.33 Ghz, 512 kB L2 cache; 3 GB 533 MHz DDR2 SDRAM; Windows Vista.

[1] `http://monetdb.cwi.nl/xml/`





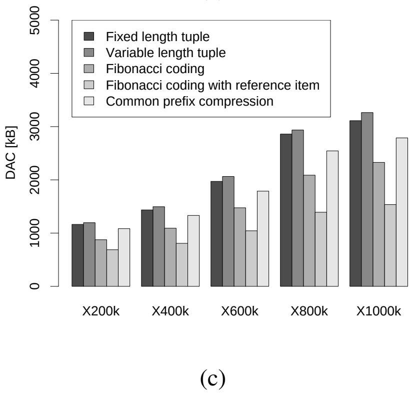


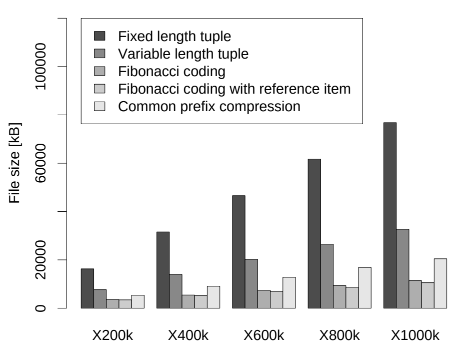
Fig. 3. Results (a) Compress ratio (b) Query time (c) DAC for fixed dimension length tuples

6 Conclusion

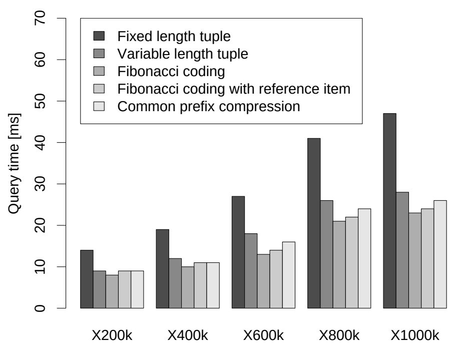
In this article we evaluate the persistent stream array compression. The persistent stream array is designed to implement the stream ADT which support an XML indexing approaches. We tested two most common types of labeling schemes of XML trees: containment labeling scheme and dewey order labeling scheme. We performed series of experiments with different compression techniques. The compression of Containment labeling scheme is feasible only if we want to decrease the size of data file. The data decompression time is always higher than the time saved on a DAC, therefore, the query processing using a compressed stream array is less efficient than the regular stream array. On the other hand, compressed stream array storing the dewey order labels perform significantly better than the regular stream array. The best query time is achieved with the compression technique utilizing the fast fibonacci coding.

References

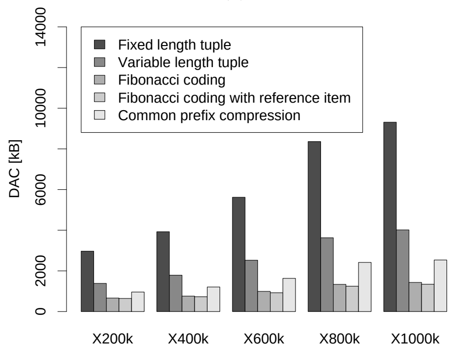
1. S. Al-Khalifa, H. V. Jagadish, and N. Koudas. Structural Joins: A Primitive for Efficient XML Query Pattern Matching. In *Proceedings of ICDE 2002*. IEEE CS, 2002.



(a)



(b)



(c)

Fig. 4. Results (a) Compress ratio (b) Query time (c) DAC for variable dimension length tuples

2. R. Baca, V. Snasel, J. Platos, M. Kratky, and E. El-Qawasmeh. The Fast Fibonacci Decompression Algorithm. *Arxiv preprint arXiv:0712.0811*, 2007.
3. N. Bruno, D. Srivastava, and N. Koudas. Holistic Twig Joins: Optimal XML Pattern Matching. In *Proceedings of ACM SIGMOD 2002*, pages 310–321. ACM Press, 2002.
4. S. Chen, H.-G. Li, J. Tatemura, W.-P. Hsiung, D. Agrawal, and K. S. Candan. Twig2Stack: Bottom-up Processing of Generalized-tree-pattern Queries Over XML documents. In *Proceedings of VLDB 2006*, pages 283–294, 2006.
5. Z. Chen, G. Korn, F. Koudas, N. Shanmugasundaram, and J. Srivastava. Index Structures for Matching XML Twigs Using Relational Query Processors. In *Proceedings of ICDE 2005*, pages 1273–1273. IEEE CS, 2005.
6. D. Comer. Ubiquitous b-tree. In *ACM Computing Surveys*, pages 121–137. ACM Press, June, 1979.
7. H. Garcia-Molina, J. Ullman, and J. Widom. *Database systems: the complete book*. Prentice Hall, 2002.
8. T. Grust, M. van Keulen, and J. Teubner. Staircase Join: Teach a Relational DBMS to Watch Its (Axis) Steps. In *Proceedings of VLDB 2003*, pages 524–535, 2003.
9. H. Jiang, H. Lu, W. Wang, and B. Ooi. XR-Tree: Indexing XML Data for Efficient. In *Proceedings of ICDE, 2003, India*. IEEE, 2003.

10. I. Tatarinov and at al. Storing and Querying Ordered XML Using a Relational Database System. In *Proceedings of ACM SIGMOD 2002*, pages 204–215, New York, USA, 2002.
11. C. Zhang, J. Naughton, D. DeWitt, Q. Luo, and G. Lohman. On Supporting Containment Queries in Relational Database Management Systems. In *Proceedings of ACM SIGMOD 2001*, pages 425–436, 2001.

Benchmarking Coding Algorithms for the R-tree Compression*

Jiří Walder, Michal Krátký, and Radim Bača

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

Abstract. Multi-dimensional data structures have been widely applied in many data management fields. Spatial data indexing is their natural application, however there are many applications in different domain fields. When a compression of these data structures is considered, we follow two objectives. The first objective is a smaller index file, the second one is a reduction of the query processing time. In this paper, we apply a compression scheme to fit these objectives. This compression scheme handles compressed nodes in a secondary storage. If a page must be retrieved then this page is decompressed into the tree cache. Since this compression scheme is transparent from the tree operations point of view, we can apply various compression algorithms to pages of a tree. Obviously, there are compression algorithms suitable for various data collections, therefore, this issue is very important. In our paper, we compare the performance of Golomb, Elias-delta and Elias-gamma coding with the previously introduced Fast Fibonacci algorithm.

Keywords: multi-dimensional data structures, R-tree, compression scheme, Golomb, Elias-delta, and Elias-gamma coding, Fast Fibonacci algorithm

1 Introduction

Multidimensional data structures [21] have been widely applied in many data management fields. Spatial data indexing is their natural application, however there are many applications in different domain fields. In the case of spatial data, structures often store two- and three-dimensional objects. In the case of multimedia data, spaces with dimensionality up to 100,000 appear.

Many multidimensional data structures have been developed in the past, e.g. the quadtree family [21], LSD-tree [11], R-tree [10], R$^+$-tree [23], R*-tree [4], and Hilbert R-tree [13]. In the case of R-tree, tuples are clustered in a tree's page using *MBBs* (*Minimal Bounding Boxes*). If we consider a multidimensional tuple collection, redundancy appears. Consequently, a compression may be used for the nodes efficient storage and retrieval.

* Work is partially supported by Grant of GACR No. 201/09/0990.

Although some works applying a compression inside a DBMS have been developed, a real-time compression of multidimensional data structures is not often a research interest. Obviously, a smaller index file means lower *DAC* (*Disk Access Cost*) when a query is processed [19]. Consequently, lower DAC may mean the lower processing time.

There are a lot of works applying a compression inside a DBMS. In [24], authors depict RLE for compression of sorted columns to have few distinct values. In [7], authors propose the SBC-tree (String B-tree for Compressed sequences) for indexing and searching RLE-compressed sequences of arbitrary length. Work [1] demonstrates that the fast dictionary-based methods can be applied to order-preserving compression. In [5], authors introduce the IQ-tree, an index compression technique for high-dimensional data spaces. They present a page scheduling strategy for nearest neighbor algorithms that, based in a cost model, can avoid many random seeks. Work [6] introduces a tree-based structure called PCR-tree to manage principle components. In [26], authors introduce the xS-tree that uses lossy compression of bounding regions. Original works written about compressions of multidimensional data structures describe the compression of quad-tree [8, 22]. Work [8] suggested an algorithm to save at least 66% of the computer storage required by regular quadtrees. The first work [9], which concerns compressing R-tree pages, uses the relative representation of MBB to increase the fanout of the R-tree page. A bulk-loading algorithm, which is a variation of *STR* [16], and a lossy compression based on the coordinate quantization are presented there. Other works in this field are focused on improving the effectiveness of the main memory indexes. Those cache-conscious indexes suppose that they can store most of the index in the main memory. Such a work is *CR-tree* [14], which uses a type of MBB representation similar to [9]. Let the irrelevant page be the page whose MBB does not intersect a query box. These works apply the lossy compression, therefore an improved compression ratio is achieved when a filtration of irrelevant pages must be processed during a query processing.

In this paper, we utilize a compression scheme for R-tree introduced in [2]. Pages of a tree are stored in a secondary storage and decompressed in a tree's cache. We achieved a lower DAC and the pages are not always decompressed when an operation is required. In this paper, we compare the Fast Fibonacci coding [3, 2] with three other coding algorithms: Golomb, Elias-Gamma, and Elias-Delta codings [20, 17].

In Section 2, we briefly describe the R-tree and its variants. In Section 3, the above depicted compression scheme is presented. In Section 4, we describe various coding techniques: Fast Fibonacci, Golomb, Elias-gamma, and Elias-delta. Experimental results are shown in Section 5. Finally, we conclude with a summary of results and discussion about future work.

2 R-tree

R-trees [10] support point and range queries, and some forms of spatial joins. Another interesting query, supported to some extent by R-trees, is the k nearest neighbors (k-*NN* query. R-tree can be thought of as an extension of B-trees in a multi-dimensional space. It corresponds to a hierarchy of nested n-dimensional MBBs (see [10] for detail). R-tree performance is usually measured with respect to the retrieval cost (in terms of DAC) of queries.

Variants of R-trees differ in the way they perform the split algorithm. The well-known R-tree variants include R*-trees and R$^+$-trees. In [18], we can find a more detailed description as well as depiction of other R-tree variants.

It is not usually efficient to insert a large amount of data into an R-tree using the standard insert operation [10, 4]. The split algorithm is rather an expensive operation, therefore, the insertion of many items may take quite a long time. Moreover, this algorithm is executed many times during the insertion. The query performance is greatly influenced by utilization of the R-tree. A common utilization rate of an R-tree created with a standard insert algorithm is around 55%. On the other hand, the utilization rate of the R-tree created with the bulk-loading method, rises up to 95% [4].

Several bulk-loading methods [12, 15, 16] have been developed. All bulk-loading methods first order input items. Method [16] utilizes one-dimensional space-filling curve criterion for such ordering. This method is very simple and allows to order input items very fast. The result R-tree preserves suitable features for the most common data.

3 A Compression Scheme for Tree-like Data Structures

In this section, we describe a basic compression scheme which can be utilized for most paged tree data structures [2]. Pages are stored in a secondary storage and retrieved when the tree requires a page. This basic strategy is widely used by many indexing data structures such as B-trees, R-trees, and many others. They utilize cache for fast access to pages as well, since the access to the secondary storage can be more than 20 times slower compared to access to the main memory. We try to decrease the amount of DAC to a secondary storage while significantly decreasing the size of a tree file in the secondary storage.

In Figure 1, we can observe the basic idea of compression scheme used in this paper. If a tree data structure wants to retrieve a page, the compressed page is transfered from the secondary storage to the tree's cache and it is decompressed there. An important issue of our compression scheme is that the pages are only compressed in the secondary storage.

When the compression scheme is taken into consideration, the tree insert algorithm only needs to be slightly modified. Query algorithms are not affected at all because page decompression is processed only between cache and secondary storage and the tree can utilize decompressed pages for searching without knowing that they have been previously compressed.

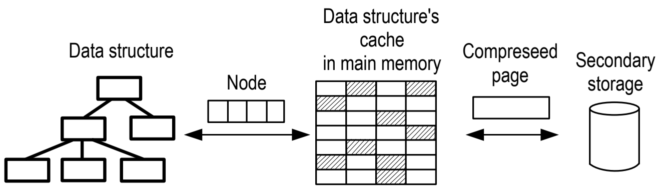


Fig. 1. Transfer of compressed pages between the secondary storage and tree's cache.

The goal of R-tree and its variants is to cluster the most similar tuples into a single page. The term 'similar tuples' means that the tuples are close to each other in a multi-dimensional space according to L_2 metric. This feature can be utilized to compress R-tree pages by a fitting compression algorithm. An important issue of this scheme is that we can apply various compression algorithms to a single R-tree. In Section 4, we show an algorithm for the R-tree compression, other compression algorithms can be found in [27].

Using this compression scheme we reduce the R-tree index size as well as DAC during a query processing. We require a decompression algorithm to be as fast as possible, otherwise the decompression time would not exceed the time saved for a lower DAC.

4 Compression Algorithm

Since tuples of a tree's page are closely located to one another in a multidimensional space, we can suppose that coordinates of these tuples are 'similar'. This means that the coordinates in each dimension are the same or their differences are rather of small values. Consequently, this feature provides increased potential for a compression.

We implemented different bit-length number coding techniques: Golomb, Elias-gamma and Elias-delta. These coding algorithms are compared with the previously published Fast Fibonacci coding [3, 2]. We utilize these coding techniques in a compression algorithm based on coding of differences between similar tuple coordinates.

4.1 Golomb, Elias-gamma and Elias-delta and Fast Fibonacci Coding

Small values are possible to code with various coding techniques. We have implemented three simple techniques for the coding of values. These techniques are as follows: Golomb, Elias-gamma, and Elias-delta [20, 17]. The algorithms used for coding are shown in Algorithms 1, 2, and 3. All codes for numbers 1-12 are depicted in Table 1.

In Algorithms 1, 2, and 3 the compressed values are read bit by bit. Retrieving the bit from the compressed memory is a time consuming operation. In [3], Fast Fibonacci decompression was introduced. This algorithm processed data without retrieving every single bit from a compressed memory. The proposed Fibonacci decompression method is based on a precomputed mapping table. This table enables converting of compressed memory segments directly into decompressed values.

Table 1. Numbers for various coding techniques

Number	**Golomb**			**Elias**		**Fibonacci**
	M=4	**M=8**	**M=16**	**gamma**	**delta**	
1	000	0000	00000	1	1	11
2	001	0001	00001	010	0100	011
3	010	0010	00010	011	0101	0011
4	011	0011	00011	00100	01100	1011
5	1000	0100	00100	00101	01101	00011
6	1001	0101	00101	00110	01110	10011
7	1010	0110	00110	00111	01111	01011
8	1011	0111	00111	0001000	00100000	000011
9	11000	10000	01000	0001001	00100001	100011
10	11001	10001	01001	0001010	00100010	010011
11	11010	10010	01010	0001011	00100011	001011
12	11011	10011	01011	0001100	00100100	101011

4.2 Difference-based Compressions

Difference-based compression algorithm for the R-tree was introduced in [2], however difference-based compression algorithms are well known [27, 20]. This algorithm is shown in Algorithm 4. This algorithm simply computes XOR differences between coordinates of the first tuple and values of other tuples. After that we add all difference numbers into the `mCodingBuffer` buffer, all numbers are coded by the `Encode` function. In this paper, we compare Fast Fibonacci, Golomb, Elias-Gamma, and Elias-Delta for coding of numbers. In Figure 2, we can see some encoded values for these coding techniques. Differences for the page P are output in the page P_{XOR}. The difference numbers in the page P_{XOR} are then coded by the `Encode` function.

5 Experimental Results

In our test[1], we have used the compression scheme depicted in Section 3 and coding algorithms described in Section 4. In this section, we compare the query

[1] The experiments were executed on a PC with 1.8 Ghz AMD Opteron 865, 2 MB L2 cache; 2 GB of DDR333; Windows 2008 Server.

```
input  : Golomb code bit stream and Golomb code parameter $parameterM$
output: Decoded number $n$
Bits ←Log(parameterM)/ Log(2);
TreshNumber ←Pow(2,Bits +1)−parameterM ;
PowerTwo ←Floor(Bits)==Bits;
qpart ← 0;
rpart ← 0;
bit ←stream.GetNextBit();
while bit do
    qpart ++;
    bit ←stream.GetNextBit();
end
if PowerTwo then
    for $x$ ← 0 to Bits do
        bit ←stream.GetNextBit();
        rpart ←rpart <<1|bit ;
    end
else
    for $x$ ← 0 to Bits do
        bit ←stream.GetNextBit();
        rpart ←rpart <<1|bit ;
    end
    if rpart >=TreshNumber then
        bit ←stream.GetNextBit();
        rpart ←rpart <<1|bit ;
        rpart ←rpart-TreshNumber ;
    end
end
n ← qpart *parameterM +rpart ;
```

Algorithm 1: Golomb decoding algorithm

$$P = \begin{pmatrix} 4 & 0 & 6624 & 6625 & 1526 \\ 42 & 0 & 6624 & 6725 & 1535 \\ 9 & 0 & 6624 & 6626 & 6631 \\ 11 & 0 & 6624 & 6632 & 6633 \\ 29 & 0 & 6624 & 6650 & 6675 \end{pmatrix} \qquad P_{XOR} = \begin{pmatrix} 4 & 0 & 6624 & 6625 & 1526 \\ 46 & 0 & 0 & 932 & 9 \\ 13 & 0 & 0 & 3 & 7185 \\ 15 & 0 & 0 & 9 & 7199 \\ 25 & 0 & 0 & 27 & 8165 \end{pmatrix}$$

Fig. 2. The example of the page (P) and computed page difference (P_{XOR})

performance of a compressed as well as uncompressed data structures. We test both real and synthetic data sets. In all experiments, we turn off the OS's disk read cache to prevent the OS from file caching and the cache of data structures was 1,000 inner and leaf nodes. The page size of all data structures is 2,048B. To compare the performance of the compressed and uncompressed R-tree we observe the following features:

```
input  : Elias-gamma code bit stream
output: Decoded number $n$
Bits $\leftarrow 1$;
n $\leftarrow 0$;
bit $\leftarrow$ stream.GetNextBit();
while not bit do
    Bits ++;
    bit $\leftarrow$ stream.GetNextBit();
end
repeat
    Bits −−;
    n $\leftarrow$ n |bit << Bits ;
    if Bits $> 0$ then
        bit $\leftarrow$ stream.GetNextBit();
    end
until Bits $== 0$;
```

Algorithm 2: Elias-gamma decoding algorithm

– the query processing time and DAC, see Section 5.1
– R-tree index size, see Section 5.2
– an influence of various space dimensionalities, see Section 5.3
– an influence of various query selectivities, see Section 5.4

We perform experiments on synthetic as well as real data sets. In the case of synthetic data sets, we generate collections of 500,000 points for dimensionalities: 2, 4, and 6 in an integer domain of the $\langle 0, 2 \times 10^6 \rangle$ range with the uniform distribution of values. In the case of real data sets, we test TIGER 2D spatial data collections of 500,000 (TIG05) and 2 million (TIG20) points [25]. These data collections only include unique tuples. In this way, the compression scheme performance is not influenced by identical tuples. We process series of query experiments where one experiment consists of 50 randomly generated queries. Consequently, each presented result is the summary result of all these queries. Query boxes covering 0.1%, 0.2%, 0.3%, 0.4%, and 0.5% of the data space were randomly generated. In other words, the query selectivity is changed in this way.

5.1 Processing Query Time and DAC

In Tables 2 and 3, query processing performance is presented for both real and random data collections. In this experiment, selectivity is 0.2%. In the case of random data, the best query time was achieved by the Fast Fibonacci algorithm. In the case of other coding algorithms, the query time is little worse than in the case of the uncompressed R-tree. The Elias-delta decoding algorithm is 30% slower than Fast Fibonacci. The Golomb and Elias-gamma algorithms are 60% slower than Fast Fibonacci. In the case of real collections, Elias-delta coding outperforms the Fast Fibonacci coding. In the case of TIG05, Elias-delta saves

```
input  : Elias-delta code bit stream
output: Decoded number $n$
Bits ← 1;
n ← 0;
x ← 0;
bit ←stream.GetNextBit();
while not bit do
    Bits ++;
    bit ←stream.GetNextBit();
end
Bits −−;
x ←x |1 <<Bits ;
while Bits > 0 do
    Bits −−;
    bit ←stream.GetNextBit();
    x ←x |bit <<Bits ;
end
x −−;
n ←n |1 <<x ;
while x > 0 do
    x −−;
    bit ←stream.GetNextBit();
    n ←n |bit <<x ;
end
```

Algorithm 3: Elias-delta decoding algorithm

6% of the query processing time in a comparison to the Fast Fibonacci algorithm and 19% of the query processing time in a comparison to the uncompressed R-tree.

In Table 4, we propose the query processing time in more detail for both Elias-delta and Fast Fibonacci. These results are related to the TIG20 collection. Obviously, time spent on reading of pages in the secondary storage is lower in the case of Elias-delta, however the decompression time is lower for the Fast Fibonacci algorithm. Overall query processing time is better for Fast Fibonacci. Elias-delta reads values in the bit-by-bit way, on the other hand Fast Fibonacci works with bytes. In the future, we can focus on a development of similar byte-based reading for other coding algorithms, especially for the Elias-delta algorithm. Elias-delta achieves the lowest DAC for both real and random data collections.

5.2 Index Sizes

An important issue of the compression is a reduction of the R-tree index size. In Figure 3(f), we can see the index sizes for the real collection. The best compression ratio was achieved by the Elias-delta encoding. In this case, we save more than 60% of the index size.

```
Input  : stream, an instance of cStream
output: Compressed R-tree node
mCodingBuffer.Clear ();
stream.Write (mCount);
for $i \leftarrow 0$ to mDimension do
    value ← mTuples [$i$].GetInt (0);
    stream.Write (value);
    for $j \leftarrow 1$ to mCount do
        int tmpValue ← mTuples [$j$].GetInt ($i$);
        int diff ← value XOR tmpValue ;
        mCodingBuffer.Add (diff);
    end
    Encode (mCodingBuffer);
    stream.Write (mCodingBuffer);
end
```

Algorithm 4: Difference-based compression of an R-tree leaf page

	Normal	**Golomb M=4**	**Golomb M=8**	**Golomb M=16**	**Elias gamma**	**Elias delta**	**Fast Fibonacci**
Processing Time [s]	60.9	87.3	87.2	89.9	86.6	68.9	**52.1**
DAC All Nodes [MB]	21,131	8,477	8,634	8,797	7,806	7,176	**7,175**
DAC Leaf Nodes [MB]	10,691	4,333	4,413	4,491	3,988	3,674	**3,673**

Table 2. Results for the random data collection, 500K tuples, dimension: 6

	Normal	**Golomb M=4**	**Golomb M=8**	**Golomb M=16**	**Elias gamma**	**Elias delta**	**Fast Fibonacci**
Processing Time [s]	3.9	3.96	3.77	4.1	3.95	**3.17**	3.41
DAC All Nodes [MB]	4,497	2,148	2,073	2,039	2,391	**1,831**	1,899
DAC Leaf Nodes [MB]	2,185	1,016	979	959	1,137	**854**	893

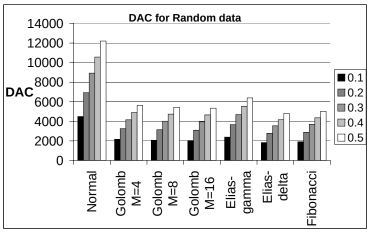
Table 3. Results for the bulk-loaded real data collection, 500K tuples, dimension: 2

Time [s]	**Regular R-tree**	**Elias-delta**	**Fast Fibonacci**
Read	13.47	**5.86**	6.17
Decompression	-	10	**5.87**
Overall	22.99	21.55	**18.94**
DAC [MB]	103,251	**38,338**	40,805

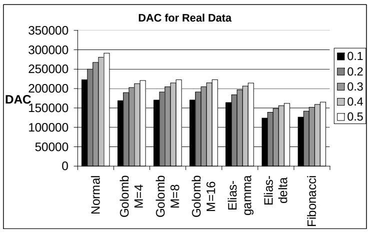
Table 4. Analysis of the query processing time

5.3 Influence of the Space Dimension

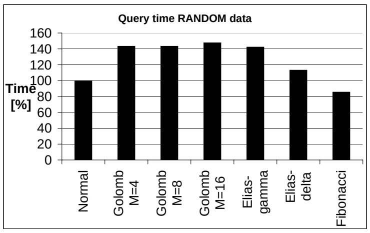
We compare DAC for randomly generated data collections with dimensionalities 2, 4, and 6 (see Figure 3e). We save more than 60% of DAC in the case of Elias-delta and dimension 2. The compression ratio weakens with increasing space dimension. The space is bigger with increasing dimension, tuples are further from one another, therefore, less redundancy appears in tuples. The dimension modification has no impact on the performance of various coding techniques.



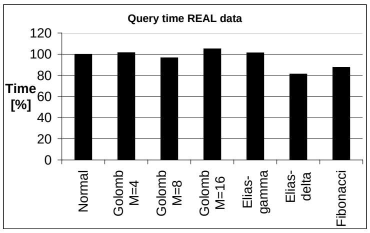
(a)



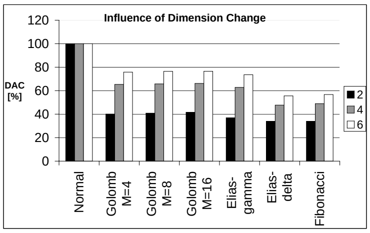
(b)



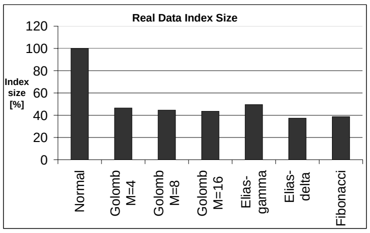
(c)



(d)



(e)



(f)

Fig. 3. DAC for various selectivities and bulk-loaded (a) random and (b) real data collections
Query processing time for bulk-loaded (c) random and (d) real data collections
(e) Selectivity influence comparison (f) Index size comparison

5.4 Influence of the Query Selectivity

In this experiment, we choose the following selectivities: 0.1%, 0.2%, 0.3%, 0.4%, and 0.5%. DAC and query processing times are put forward in Figures 3(a)-(d). The results are presented for both random and real data collections. Obviously, Elias-delta outperform other coding algorithms. The other codings produce approximately the same DAC. The selectivity modification has no impact on the performance of various coding techniques.

6 Conclusion

In this paper, we test a lossless compression scheme for the R*-tree data structure. We compare the following coding techniques: Golomb, Elias-Gamma, and Elias-Delta, with the previously published Fast Fibonacci coding. All coding algorithms improve DAC compare to the regular R-tree. In the case of real data collections, Elias-delta and Fast Fibonacci techniques achieve the best results. The Elias-delta algorithm saves 5% DAC of Fast Fibonacci. All other algorithms are less efficient that the Fast Fibonacci algorithm. When real data sets are concerned, the compression methods save at least 60% of the index size required by a regular R-tree.

The best compression ratio was achieved by the Elias-delta codding. On the other hand, decompression time for Fast Fibonacci is lower than in the case of Elias-delta. Elias-delta reads values in the bit-by-bit way, however Fast Fibonacci works with bytes. In our future work, we want to focus on a development of similar byte-based reading for other coding algorithms, especially for the Elias-delta algorithm.

References

1. G. Antoshenkov. Dictionary-based Order-preserving String Compression. *VLDB Journal – The International Journal on Very Large Data Bases*, 6(1):26–39, 1997.
2. R. Bača, M. Krátký, and V. Snášel. A compression scheme for multi-dimensional data structures. *Submitted to Information Systems*, 2009.
3. R. Bača, V. Snášel, J. Platoš, M. Krátký, and E. El-Qawasmeh. The fast fibonacci decompression algorithm. In *arXiv:0712.0811v2*, `http://arxiv.org/abs/0712.0811`, 2007.
4. N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD 1990)*, pages 322–331. ACM Press, 1990.
5. S. Berchtold, C. Böhm, H.-P. Kriegel, J. Sander, and H. Jagadish. Independent Quantization: An Index Compression Technique for High-Dimensional Data Spaces. In *Proceedings of the 16th International Conference on Data Engineering (ICDE 2000)*, pages 577 – 588. IEEE Computer Society, 2000.
6. J. Cui, S. Zhou, and S. Zhao. PCR-Tree: A Compression-Based Index Structure for Similarity Searching in High-Dimensional Image Databases. In *Proceedings of the Fourth International conference on Fuzzy Systems and Knowledge Discovery (FSKD 2007)*, pages 395–400. IEEE Computer Society, 2007.
7. M. Eltabakh, W.-K. Hon, R. Shah, W. G. Aref, and J. Vitter. The SBC-Tree: An Index for Run-Length Compressed Sequences. In *Proceedings of the 11th International Conference on Extending Database Technology (EDBT 2008)*. ACM Press, 2008.

8. I. Gargantini. An Effective Way to Represent Quadtrees. *Communications of the ACM*, 25:905–910, 1982.
9. J. Goldstein, R. Ramakrishnan, and U. Shaft. Compressing Relations and Indexes. In *Proceedings of IEEE Conference on Data Engineering (ICDE 1998)*, pages 370–379, Los Alamitos, USA, 1998. IEEE Computer Society.
10. A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of ACM International Conference on Management of Data (SIGMOD 1984)*, pages 47–57. ACM Press, June 1984.
11. A. Henrich, H. W. Six, and P. Widmayer. The lsd tree: spatial access to multidimensional and non-point objects. In *VLDB '89: Proceedings of the 15th international conference on Very large data bases*, pages 45–53, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.
12. I. Kamel and C. Faloutsos. On packing R-trees. In *Proceedings of the Second International Conference on Information and Knowledge Management (CIKM 1993)*, pages 490–499. ACM Press, 1993.
13. I. Kamel and C. Faloutsos. Hilbert r-tree: An improved r-tree using fractals. In *In Proceedings of VLDB 1984*, pages 500–509, 1994.
14. K. Kim, S. K. Cha, and K. Kwon. Optimizing Multidimensional Index Trees for Main Memory Access. In *Proceedings of ACM International Conference on Management of Data (SIGMOD 2001)*, pages 139–150, New York, USA, 2001. ACM Press.
15. L.Arge, K.H.Hinrichs, J.Vahrenhold, and J.S.Vitter. Efficient Bulk Operations on Dynamic R-Trees. *Algorithmica*, pages 104–128, 2004.
16. S. Leutenegger, M. Lopez, and J. Edgington. STR: A Simple and Efficient Algorithm for R-Tree Packing. In *Proceedings of 13th International Conference on Data Engineering (ICDE 1997)*, pages 497–506. IEEE CS Press, 1997.
17. D. J. C. Mackay. *Information Theory, Inference & Learning Algorithms.* Cambridge University Press, June 2002.
18. Y. Manolopoulos, A. Nanopoulos, A. N. Papadopoulos, and Y. Theodoridis. *R-Trees: Theory and Applications.* Springer, 2005.
19. Y. Manolopoulos, Y. Theodoridis, and V. J. Tsotras. *Advanced Database Indexing.* Kluwer Academic Publisher, 2001.
20. D. Salomon. *Data Compression The Complete Reference.* Third Edition, Springer–Verlag, New York, 2004.
21. H. Samet. *Foundations of Multidimensional and Metric Data Structures.* Morgan Kaufmann, 2006.
22. H. Samet. Data Structures for Quadtree Approximation and Compression. *Communications of the ACM archive*, 28(9):973–993, September 1985.
23. T. Sellis, N. Roussopoulos, and C. Faloutsos. The r$^+$-tree: A dynamic index for multidimensional objects. In *In Proceedings of VLDB 1987*, pages 507–518, 1987.
24. M. Stonebraker, D. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, and S. Madden. C-store: A Column Oriented DBMS. In *Proceedings of the International Conference on Very Large Data Bases, VLDB 2005.*
25. U.S. Department of Commerce, U.S. Census Bureau, Geography Division. TIGER/Line Files, 2006 Second Edition, Alabama, Autauga County, 2006, `http://www.census.gov/geo/www/tiger/`.
26. C. Wang and X. S. Wang. Indexing Very High-dimensional Sparse and Quasi-sparse Vectors for Similarity Searches. *VLDB Journal – The International Journal on Very Large Data Bases*, 9(4):344–361, 2001.
27. I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes, Compressing and Indexing Documents and Images, 2nd edition.* Morgan Kaufmann, 1999.

Translation of Ontology Retrieval Problem into Relational Queries

Jaroslav Pokorný[1], Jana Pribolová[2], and Peter Vojtáš[1]

[1] Department of Software Engineering,
Charles University, Prague, Czech Republic
{Pokorny, Vojtas}@ksi.mff.cuni.cz
[2] Institute of Computer Science,
Faculty of Science, University of P. J. Šafárik,
{Jana.Pribolova}@upjs.sk

Abstract. Ontology as a knowledge base can provide different reasoning tasks, e.g. to check consistency of the ontology or to check whether a resource is instance of a concept or not. In this paper we want to focus on retrieval problem. There already exist systems resolving this problem, but they are not effective within large datasets. Our idea is to transform ontology into a relational database. We present particular algorithms of this transformation both on the scheme level and SQL level with special handling of functional roles and definitions. This enables to query such database by usual tools of SQL, i.e. to solve the retrieval problem.

Keywords: ontology, translation, relational database, SQL

1 Introduction

OWL enables the creation of ontologies and provides extensive semantics for Web data. This language is heavily influenced by description logics (DL, see [1]). Research on DL reasoners consists of the solving reasoner problems such as satisfiability, consistency or retrieving instances of the concept. Relational database is an excellent system for storing and querying data, but their inferencing capabilities are limited just to querying. In this paper we describe the method to extend relational database to store ontology.

In present some research groups are interesting in topic of ontology storing and maintenance. Some of them are trying to limit expressive power of language to speed up reasoner tasks. But some of them are trying to fuse two or more kind of systems.

One of the prominent directions in this area is blending ontologies and logic databases [5, 3]. They create so called *description logic programs* which are description logic expressions and logic programs mixed together.

Another direction of the research area is combining ontologies and relational databases. First experiments focus on XML document translation into corresponding relational tables [9]. Inspired by the XML storing in relational databases there are some research projects concerned with ontology storing in the relational databases. One of the first projects is published in [4]. However, many others occurred, e.g., the system HAWK [10] and its ancestor the system DLDB [7], as well as the projects described in [2, 6].

In Section 2 we write about a knowledge base represented through DL and relational data model. Further, in subsection 2.1 we deal with potential and valid domains as well as with ranges. The valid domains are important input parameters of the algorithm to construct relational scheme which we present in Section 2.2. Subsection 2.3 explains creating the relational database. Section 3 concentrates on ontology implementation in an SQL environment. This enables to query such database by usual tools of SQL, i.e. to solve the retrieval problem. In fact, it means creating tables and views (Subsection 3.1) in the SQL language. Then we explain how to insert data in the tables (Subsection 3.2). Section 4 concludes the paper.

2 Knowledge Base in Relational Data Model

Basics of DL, ontologies and all used symbols are described in [8], here we refer to unexplained notions. Our language consists of names for atomic concepts $A, B \in \mathbf{NC}$, names for roles $R \in \mathbf{NR}$. Roles are either functional (**NFR**) or non-functional (**NNR**). Concept constructions we usually denote by C, D and we understand them as nonterminal symbols.

The main idea of the knowledge base representation with relational data model is shown in Figure 1. For every concept or role we create unary or binary relation, respectively, except for functional roles. For every functional role we create only one attribute in a relation or one attribute in more than one relation depending on so called valid domains (see Section 2.1). The ABox assertions are translated by inserting some tuples into relations. TBox assertions of the type $\sqsubseteq$ are translated into integrity constrains and equalities into special relations called T_C^{view}. In practice, such a relation is represented by view in SQL.

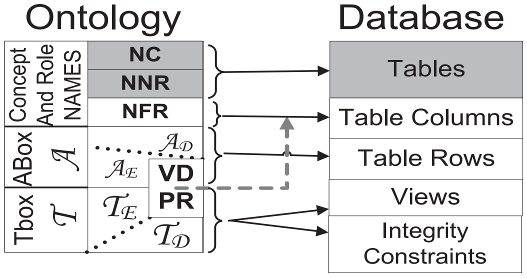


Fig. 1. Translation mapping the ontology elements to database structures.

Note that ontology as a relational database offers the users some kind of inferencing. The main feature is to retrieve all instances of the concept. Moreover such a database supports query construction. In DL, an equivalent of the query is a concept defined as equality. However, in practice ontology-based systems are applied for more complicated queries than concepts can be. The W3C standard for such a set of queries is the query language called SPARQL [13]. A lot of such queries are supported by our system.

Section 2.2 presents the solution to find out all instances of the concept, not only instances that are explicitly known, i.e. those expressed by $C(a)$.

2.1 Domains and Ranges of Roles

In DL the knowledge base comprises TBox and ABox. We understand both of them, ABox $\mathcal{A}$ about and TBox $\mathcal{T}$, as sets of assertions (about individuals or concepts, respectively). Each set of assertions can be divided into two categories. One, denoted by subscript $\mathcal{E}$, includes *extensional assertions*, the second one comprehends as set of additionally *deduced assertions* and it is denoted by subscript $\mathcal{D}$. The set $\mathcal{T}_\mathcal{E}$ consists of acyclic definitions $A := C$ and axioms $C \equiv D$ and $C \sqsubseteq D$. The set $\mathcal{T}_\mathcal{D}$ is derived with respect to (symmetric, transitive) properties of the assertions $\equiv$ and $\sqsubseteq$. For all TBox sets the following holds:

- $\mathcal{T} = \mathcal{T}_\mathcal{E} \cup \mathcal{T}_\mathcal{D}$,
- $\mathcal{T}_\mathcal{E} \cap \mathcal{T}_\mathcal{D} = \emptyset$.

ABox $\mathcal{A}_\mathcal{E}$ consists of statements of the form $B(a)$ and $R(a, b)$. Let us also mention that $\mathcal{A} = \mathcal{A}_\mathcal{E} \cup \mathcal{A}_\mathcal{D}^\mathcal{T}$. The set $\mathcal{A}_\mathcal{D}^\mathcal{T}$ depends on the TBox $\mathcal{T}$, because we derive assertions on basis of $\mathcal{A}_\mathcal{E}$ and TBox assertions. If it is evident which $\mathcal{T}$ inducted $\mathcal{A}_\mathcal{D}^\mathcal{T}$, we omit superscript $\mathcal{T}$.

Example 1. Let us have a knowledge base $\mathcal{O}$ with concepts shown in Figure 2. The concept *Student* is defined as follows:

$$Student := Person \sqcap \exists takesCourse.Course$$

The set of assertions $\mathcal{T}$ includes previous definition of concept *Student* and also subsumption assertions shown in Figure 2 with solid line. Also there are some roles in $\mathcal{O}$, i.e. **NFR** $= \{hasName, hasAddress\}$ and **NNR** $= \{takesCourse\}$.

Note that the name of the concept *GraduateCourse* is abridged to *GCourse*. The deduced assertions are shown in Figure 2 with dotted arrows.

Also ABox consists of the extensional assertions:

$$\begin{array}{llll}
\mathcal{A}_\mathcal{E} = \{ & Person(S_2), & GCourse(C_2), & hasName(P_1, Name_b), \\
 & Student(S_1), & takesCourse(S_1, C_1), & hasName(C_1, Name_c), \\
 & Publication(P_1), & takesCourse(S_2, C_1), & hasName(C_2, Name_d), \\
 & Article(P_1), & hasName(S_1, Name_a) & hasAddress(S_1, Address_a), \\
 & Course(C_1), & & \}
\end{array}$$

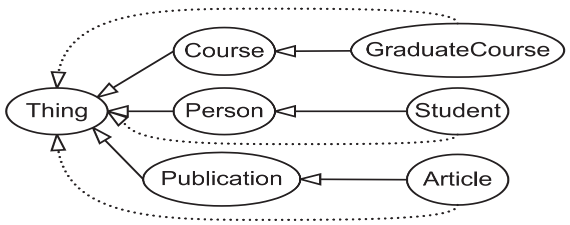


Fig. 2. IS-A hierarchy of the knowledge base $\mathcal{O}$.

and also of the deduced ones:

$$\mathcal{A}_{\mathcal{D}}^{\mathcal{T}} = \{Person(S_1), Course(C_2), Student(S_2)\}$$

As we can see the assertion *Publication*(P_1) belongs to the set of extensional assertions. Though it can be deduced also, because the assertion *Article* $\sqsubseteq$ *Publication* is in the TBox $\mathcal{T}$ and *Article*(P_1) is an extensional assertion.

In DL there are two kinds of the equality meanings. The equality whose left-hand side is atomic concept means the definition. In this paper we denote definition equality as the symbol $:=$ instead of $\equiv$ to distinguish definition equalities from the rest. We assume, there are no cycles in definitions.

In OWL language there is a chance to define domain and range of a role but it is not a necessary condition. Emphasize that our approach tries to reduce the number of join operations within the query construction. The main idea is to encode each functional property as an attribute of the relation, that represents domain of the role, not as standalone relation. In case without defined domain (range) of the functional roles we need to find concepts of instances that are related to another through the role. Sometimes domain or range are defined as union of concepts. This case is similar to the previous one. Therefore we define so called potential and valid domains as follows.

Definition 1. *A concept $B \in \mathbf{NC}$ is said to be a potential domain for role $R \in \mathbf{NR}$ with respect to the set assertions $\mathcal{A}$ if there is $R(a, b) \in \mathcal{A}$ such that $B(a) \in \mathcal{A}$. The set of potential domains for role R with respect to $\mathcal{A}$ is denoted $\mathbf{PD}_R^{\mathcal{A}}$.*

Example 2. Let us illustrate the Definition 1 on Example 1. The role *hasName* has the following potential domains with respect to $\mathcal{A}_{\mathcal{E}}$:

$$\mathbf{PD}_{hasName}^{\mathcal{A}_{\mathcal{E}}} = \{Publication, Article, Course, GCourse, Student\}$$

The role *hasAddress* has the following potential domain with respect to $\mathcal{A}_{\mathcal{E}}$:

$$\mathbf{PD}_{hasAddress}^{\mathcal{A}_{\mathcal{E}}} = \{Student\}$$

In OOP (Object-Oriented Programming) if the ancestor class has defined a function, the descendant class can use it. We map the concept into meaning of OOP class and functional role into OOP function. That means that we do not need translate the same functional role for ancestor concept and for descendant concept separately. Therefore let us define valid domains – concepts for which we consider the functional role to translate into relational scheme.

Definition 2. *A valid domain for role $R \in \mathbf{NFR}$ with respect to $\mathcal{A}$ is a potential domain $A \in \mathbf{PD}_R^{\mathcal{A}}$ with property that does not exists B, $B \in \mathbf{PD}_R^{\mathcal{A}}$ so that $A \sqsubseteq B \in \mathcal{T}$ and $A \equiv B \notin \mathcal{T}$, as well as $B \sqsubseteq A \notin \mathcal{T}$. The set of valid domains for role R is denoted as $\mathbf{VD}_R^{\mathcal{A},\mathcal{T}}$. If we are interested in valid domains with respect to whole TBox assertions, e.g. $\mathcal{T}$, we can omit the superscript $\mathcal{T}$.*

Note that if ABox is changed it is necessary to revise the potential and valid domains again. This process prevents ontology deformation of the original modeling intent.

Example 3. With assistance of the previous examples we can present the valid domains of the role *hasName* with respect to $\mathcal{A}_{\mathcal{E}}$:

$$\mathbf{VD}_{hasName}^{\mathcal{A}_{\mathcal{E}}} = \{Publication, Course, Student\}.$$

The concept *GCourse* does not belong to the set $\mathbf{VD}_R^{\mathcal{A}_{\mathcal{E}}}$ because there exists $Course \in \mathbf{VD}_R^{\mathcal{A}_{\mathcal{E}}}$ so that $GCourse \sqsubseteq Course$ and neither $Course \sqsubseteq GCourse$ nor $Course \equiv GCourse$ is in $\mathcal{T}$.

It is useful to define "inverse" function that can find for any concept A all roles for which the concept is valid domain.

Definition 3. *The role $R \in \mathbf{NFR}$ is said to be a role defined on the concept B with respect to the set of assertions $\mathcal{A}$ if $B \in \mathbf{VD}_R^{\mathcal{A},\mathcal{T}}$. We denote the set of all roles defined on the concept B as $isIn\mathbf{VD}_B^{\mathcal{A},\mathcal{T}}$. Similarly as in Definition 2 if we use all assertions of $\mathcal{T}$, we can omit the superscript and denote the set of all roles defined on the concept B as $isIn\mathbf{VD}_B^{\mathcal{A}}$.*

Example 4. In the running example of this paper an interesting point is to compute $isIn\mathbf{VD}_{Student}^{\mathcal{A}_{\mathcal{E}}}$:

$$isIn\mathbf{VD}_{Student}^{\mathcal{A}_{\mathcal{E}}} = \{hasName, hasAddress\}$$

Our solution requires valid domains to encode functional properties. To keep some integrity constraints it is useful to define also range of the roles.

Definition 4. *A potential range for role $R \in \mathbf{NR}$ (with respect to $\mathcal{A}$) is concept B for which there exists $R(a,b) \in \mathcal{A}$ and $B(b) \in \mathcal{A}$. We denote the set of potential ranges of role R as $\mathbf{PR}_R$.*

Example 5. We compute potential ranges for all roles as follows:

$$\mathbf{PR}_{hasName}^{\mathcal{A}} = \{String\},$$

$$\mathbf{PR}_{takesCourse}^{\mathcal{A}} = \{Course\}.$$

Computation of valid ranges is unnecessary because we use ranges only to keep integrity.

2.2 Construction of a Relational Scheme

The first part of ontology translation into relational database consists of creating a relational scheme.

Algorithm 1 Let $\mathcal{O}$ be a knowledge base with TBox $\mathcal{T}$ and ABox $\mathcal{A}$. $\mathcal{T}$, $\mathcal{A}$, concept's names, and role's name are translated into relational database $\mathcal{D} = (\mathbf{R}, \mathbf{I})$. Here $\mathbf{R}$ denotes a relational database scheme consisting of basic relational schemes and view definitions using relational algebra expressions (RA expressions). Second, $\mathbf{I}$ denotes a set of integrity constraints. The translation is done by induction as described below.

First part of translation depends only on the language, the second part depends also on ABox and the last depends on the TBox too.

Note that names of attributes are motivated by RDF *(subject, predicate, object)* and *resource* terminology.

The construction is based on the following steps:

First translation steps are based solely on the description logic language

1. For all $A \in \mathbf{NC}$ we add to $\mathbf{R}$ new relation T_A with scheme $T_A(\underline{resource})$.
2. For all $R \in \mathbf{NNR}$ we add to $\mathbf{R}$ a relation scheme $T_R(\underline{subject, object})$.

Following translation steps depend on the ABox (and deduced valid domains)

3. For all $A \in \mathbf{NC}$ for which $isIn\mathbf{VD}_A^{\mathcal{A}\varepsilon} = \{R_1, R_2, \ldots, R_n\} \subseteq \mathbf{NFR}$, $n \geq 1$ we modify $T_A(resource) \in \mathbf{R}$ to relation $T_A^{mod} \in \mathbf{R}$ with scheme
$$T_A^{mod}(\underline{resource}, R_1.object, \ldots, R_n.object)$$
If $R \in \mathbf{NFR}$ and $\mathbf{VD}_R^{\mathcal{A}\varepsilon} = \emptyset$, then we add to $\mathbf{R}$ a new relational scheme $T_R(\underline{subject}, object)$.

The following translations depend on the TBox. First we deal with definitions:

4. For all $A \in \mathbf{NC}$ such that there is a concept construction C with $A := C \in \mathcal{T}$ we add to $\mathbf{R}$ a new relation T_A^{view} with scheme $T_A^{view}(\underline{resource})$ and view definitions so that (S_D and S_E are defined in step 5):
   - If $C := D \sqcap E$ then
   $$T_A^{view} = T_A \cup (S_D \cap S_E)$$
   - If $C := \exists R.D$ and $R \in \mathbf{NNR}$ or $\mathbf{VD}_R^{\mathcal{A}\varepsilon} = \emptyset$ then
   $$T_A^{view} = T_A \cup (T_R(subject, object) [T_R.object = S_D.resource] S_D(resource))[T_R.subject].$$
   - If $C = \exists R.D$ and $R \in \mathbf{NFR}$ and $\mathbf{VD}_R^{\mathcal{A}\varepsilon} \neq \emptyset, n > 0$ then
   $$T_A^{view} = T_A \cup (S_R^{rec}(subject, object) [S_R^{rec}.object = S_D.resource] S_D(resource))[S_R^{rec}.subject]$$

where S_R^{rec} is the RA expression for reconstruction of the role R from appropriate columns of T_B^{mod} tables

$$S_R^{rec}(subject, object) = \bigcup_{B \in \mathbf{VD}_R^{\mathcal{A}\mathcal{E}}} \left(T_B^{mod}[resource, R.object]\right)$$

Here we assume that this is a lossless encoding of all ABox information about R.

5. For a non-atomic concept construction C such that there is in $\mathcal{T}$ no definition with right hand side C and C is a sub construction of a concept definition in $\mathcal{T}$, then we create a new RA expression S_C with the only attribute resource so that:
   - If $C = D \sqcap E$ then
     $$S_C = (S_D \cap S_E)$$
   - If $C = \exists R.D$ and $R \in \mathbf{NNR}$ or $\mathbf{VD}_R^{\mathcal{A}\mathcal{E}} = \emptyset$ then
     $S_C = (T_R(subject, object)[T_R.object = S_D.resource]S_D(resource))$ $[T_R.subject]$
   - If $C = \exists R.D$ and $R \in \mathbf{NFR}$ and $\mathbf{VD}_R^{\mathcal{A}\mathcal{E}} \neq \emptyset$ then
     $S_C = (S_R^{rec}(subject, object)[S_R^{rec}.object = S_D.resource]S_D(resource))$ $[S_R^{rec}.subject]$
6. To transform axioms in $\mathcal{T}$, we add the following integrity constraint to $\mathbf{I}$:
   - if $C \equiv D \in \mathcal{T}$ and C, D are non atomic concept constructions then $S_C = S_D \in \mathbf{I}$,
   - if $C \sqsubseteq D \in \mathcal{T}$ then $S_C \subseteq S_D \in \mathbf{I}$.

The following interesting observations result from the previous algorithm.

1. For all $R \in \mathbf{NNR}$ and for all $R \in \mathbf{NFR}$ for which $\mathbf{VD}_R^{\mathcal{A}\mathcal{E}} = \emptyset$:
   - $T_R[subject] \subseteq \bigcup_{B \in \mathbf{PD}_R^{\mathcal{A}\mathcal{E}}} T_B$,
   - $T_R[object] \subseteq \bigcup_{B \in \mathbf{PR}_R} T_B$.
2. For all $B \in \mathbf{NC}$ for which $isIn\mathbf{VD}_B^{\mathcal{A}\mathcal{E}} \neq \emptyset$ and for all $R \in isIn\mathbf{VD}_B^{\mathcal{A}\mathcal{E}}$:
   $$T_B^{mod}[R.object] \subseteq \bigcup_{A \in \mathbf{PR}_R} T_A.$$

These assertions check "integrity" of the translation of role assertions. In more detail, the mentioned assertions take care to preserve the subject in area of role domains and the object in area of the role ranges.

The previous algorithm is illustrated on the following example.

Example 6. According to previous examples and applying Algorithm 1 we receive the following database scheme:

$$\begin{array}{rl} T_{Publication}^{mod}\ (\underline{resource}, hasName.object), & T_{Article}\ (\underline{resource}), \\ T_{Course}^{mod}\ (\underline{resource}, hasName.object), & T_{GCourse}\ (\underline{resource}), \\ T_{Student}^{mod}\ (\underline{resource}, hasName.object), & T_{Person}\ (\underline{resource}), \\ T_{takesCourse}\ (\underline{subject, object}). & \end{array}$$

and
$\mathbf{I} = \{T_{Article}[resource] \subseteq T_{Publication}[resource],$
$\quad T_{GCourse}[resource] \subseteq T_{Course}[resource],$
$\quad T_{Student}[resource] \subseteq T_{Person}[resource]\}.$
Also relation defined as follows belongs to the database scheme:
$T^{view}_{Student} = T_{Student}[resource] \cup T_{Person} \cap$
$\quad \big(T_{takesCourse}[T_{takesCourse}.object = T_{Course}.resource]$
$\quad \big)[T_{takesCourse}.subject]$
and the following assertions hold for given $\mathcal{D}$:

- $T_{takesCourse}[subject] \subseteq T_{Person} \cup T_{Student}$
- $T_{takesCourse}[object] \subseteq T_{Course}$

Note that, the attributes of the relations representing non-functional roles called *subject* and *object* have the same domain. Also note that all instances of a concept $B \in \mathbf{NC}$, so that $B := D$, are stored in T^{view}_{B}.

2.3 Construction of a Relational Database

In previous section we have created a relational scheme. Now we present the algorithm to insert the data in the database relations.

Algorithm 2 Suppose that $\mathcal{T}$, **NC** and **NR** are translated into database $\mathcal{D}$. ABox $\mathcal{A}$ is transferred into $\mathcal{D}$ by induction as follows:

1. If $B(a) \in \mathcal{A}$ and also $B \in \mathbf{NC}$, then $\langle a \rangle \in T_B$.
2. If $R(a,b) \in \mathcal{A}$ and $R \in \mathbf{NNR}$, then $\langle a,b \rangle \in T_R$.
3. If $R(a,b) \in \mathcal{A}$ and $R \in \mathbf{NFR}$, then one of the following items:
   (a) if $\mathbf{VD}^{\mathcal{A}_\mathcal{E}}_R = \emptyset$ then
   $$\langle a,b \rangle \in T_R,$$
   (b) if there exists $A \in \mathbf{VD}^{\mathcal{A}_\mathcal{E}}_R$ so that $A(a) \in \mathcal{A}$, then
   $$\langle a,b \rangle \in T^{mod}_A[T_A.resource, R.object],$$
   (c) if there exists $A \in \mathbf{PD}_R \setminus \mathbf{VD}^{\mathcal{A}_\mathcal{E}}_R$ so that $A(a) \in \mathcal{A}$, then there exists a maximal sequence $A = B_1, B_2, \ldots, B_n \in \mathbf{NC}$ so that $B_n \in \mathbf{VD}^{\mathcal{A}_\mathcal{E}}_R$ and $B_i \sqsubseteq B_{i+1} \in \mathcal{T}_\mathcal{D}$ for $i = 1, \ldots, n-1$ Then
   $$\langle a,b \rangle \in T^{mod}_{B_n}[B.resource, R.object].$$

Note that the last step of algorithm the information $A(a)$ is not lost and moreover information $B_n(a)$ is a consequence of TBox axioms.

It is important to remember that the valid domains are built with respect to the set of extensional assertions $\mathcal{A}_\mathcal{E}$ (Algorithm 2, Steps 3b and 3c). On the other hand, we consider all assertions from ABox $\mathcal{A}$ to insert them in the proper tables (Step 1).

Example 7. After applying Algorithm 2 we obtain:

$$\begin{aligned} T^{mod}_{Publication} &= \{\langle P_1, Name_b \rangle\}, & T_{Person} &= \{\langle S_1 \rangle, \langle S_2 \rangle\}, \\ T^{mod}_{Course} &= \{\langle C_1, Name_c \rangle, \langle C_2, Name_d \rangle\}, & T_{Article} &= \{\langle P_1 \rangle\}, \\ T^{mod}_{Student} &= \{\langle S_1, Name_a \rangle\}, & T^{view}_{Student} &= \{\langle S_2 \rangle\}. \end{aligned}$$

3 Ontology Implementation in an SQL Environment

We designed and implemented translation of a description logic knowledge base into an SQL environment, which works in accordance with Algorithms 1, 2. In fact, we create the database scheme which consists of `CREATE TABLE` and `CREATE VIEW` statements of the SQL language.

3.1 Create Statements

First we create tables representing concepts as it is stated in Step 1 of Algorithm 1. A special case is the top concept. For technical reasons, we assign a numerical identifier to every URI which represents the associated instance.

```
for all $A \in \mathbf{NC}$ do:
  if ($A = \top$) then
     CREATE TABLE $T_\top$
       ( resource INT NOT NULL PRIMARY KEY,
         uri VARCHAR NOT NULL );
  else
     CREATE TABLE $T_A$ ( resource INT NOT NULL PRIMARY KEY );
```

Next step is the Step 2 of Algorithm 1. Therefore we create tables for non-functional roles in this way:

```
for all $R \in \mathbf{NNR}$ do:
  CREATE TABLE $T_R$ (
    subject INT NOT NULL,
    object INT NOT NULL,
    PRIMARY KEY(subject, objects) );
```

After that we deal with functional properties – Step 3 of the Algorithm 1:

```
for all $R \in \mathbf{NFR}$ do:
  for all $A \in \mathbf{VD}_{\mathbf{R}}^{\mathcal{A}\varepsilon}$ do: ALTER TABLE $T_A$ ADD COLUMN $R\_object$ INT;
```

Let us mention that in DL we do not distinguish different kinds of roles. However, in OWL there are two kinds of roles. Practically a role represents a relationship type (in OWL so called object property) or an attribute type (in OWL data type property). In case of relationship roles there is relationship between two instances represented by URIs. On the other hand, attribute roles represents relationship between instance represented by URI and literal value. It may appear that the problem can became if in the column representing the object functional role there is a literal value, or URI in the case of the data type role. But practically the role can be either object type or data type, not both types simultaneously. So it could not happen the mentioned collision.

For all atomic concepts that are equivalent to other concept, we also create view in the database as it is defined in Algorithm 1 in Step 4. Note, that in [8] we have proved that to any concept defined via other atomic and non-atomic concepts we can construct an SQL view whose definition contains only `INTERSECTION` operations, `SELECT`s and `TABLE R` expressions. Each `SELECT` uses only join conditions. To achieve this it is necessary to normalize concept definition on the relational algebra level. We omit details of this procedure here. After this comment Step 4 looks like:

```
for all $A \in \mathbf{NC}$: $C \equiv D$ do:
  String[] elements = getExpressionsOf($A$)
  if (elements.length > 1) then
    for i=2 to elements.length do:
      elements[1] += " INTERSECT " + elements[i]
  CREATE VIEW View$_A$ AS elements[1];
```

where the function `getExpressionsOf` is defined as follows:

```
String[] getExpressionsOf(Concept $A$){
  String[] result;
  for all $D_i$ : $A := \prod_{i \geq 1} D_i$ do
    if ($D_i \in \mathbf{NC}$) then
      result[i] = "SELECT resource FROM $T_{D_i}$;"
    else if ($D_i = \exists R.E$ and $E \in \mathbf{NC}$) then
      result[i] = "SELECT $T_R$.subject FROM returnRole(R)
                   JOIN $T_E$ ON $T_R$.object = $T_E$.resource;"
    else if ($D_i = \exists R.E$) then
      result[i] = "SELECT $T_R$.subject
                   FROM " + returnRole(R) + " AS $T_R$
                   JOIN " + getExpressionsOf($E$) + " AS $T_E$
                   ON $T_R$.object = $T_E$.resource;"
  return result;
}
```

and the function `returnRole` is:

```
String returnRole(Role $R$){
  String result;
  if ($R \in \mathbf{NNR}$) then result = "$T_R$"
  else
    int i = 0;
    for all $A \in \mathbf{VD}_R^{\mathcal{A}\varepsilon}$ do:
      i++;
      if (i = 1) then result = "SELECT resource,$R\_object$ FROM $T_A$"
      else
        result =+ "UNION SELECT resource,$R\_object$ FROM $T_A$"
  return result;
}
```

At the end of the Algorithm 1 we have do one more thing – to add some integrity constraints generated in Step 6.

```
for all $t$: $t = A \sqsubseteq B$ do:
  ALTER TABLE $T_A$ ADD FOREIGN KEY (resource) REFERENCES $T_B$;
```

The equality of two tables implementing $\equiv$ leads to cyclic referential integrity in relational database schema. Therefore, we omit it from further consideration.

3.2 Insert the Data

Let us to suppose that the database scheme in SQL is created. Now we will deal with data – instances as it is described in Algorithm 2. We will show how to insert them into created tables.

```
for all $A \in \mathbf{NC} \setminus \{\top\}$ do:
  for all $A(a) \in \mathcal{A}$ do:
    if ($a \in T_\top$) then
      id = (SELECT resource FROM $T_\top$ WHERE uri = '$a$')
    else
      INSERT INTO $T_\top$(resource, uri) VALUES ( generateId($a$),'$a$');
    INSERT INTO $T_A$(resource) VALUES ( generateId($a$));
```

This code fragment implements Step 1. It uses the function `int generateId( String URI)` which generates a numerical identifier to use it within condition in join operation instead of string identifier.

Step 2 of the Algorithm 2 is implemented in this way:

```
for all $R \in \mathbf{NNR}$ do:
  for all $R(a,b) \in \mathcal{A}$ do:
    INSERT INTO $T_R$ VALUES (getId($a$),getId($b$));
```

Finally, Step 3 of the Algorithm is interpreted as follows:

```
for all $R \in \mathbf{NFR}$ do:
  for all $R(a,b) \in \mathcal{A}$ do:
    if($\mathbf{VD}_R^{\mathcal{A}\varepsilon} = \emptyset$ then
      INSERTINTO $T_R$ VALUES (getId($a$), getId($b$))
    else
    if($\exists A \in \mathbf{VD}_R^{\mathcal{A}\varepsilon}$ and $A(a) \in \mathcal{A}$) then
      UPDATE $T_A$ SET $R\_object$ = getId($b$) WHERE resource = getId($a$)
    else
      find $B \in \mathbf{VD}_R^{\mathcal{A}\varepsilon}$ : $A \sqsubseteq B$
      UPDATE $T_B$ SET $R\_object$ = getId($b$) WHERE resource = getId($a$)
```

The function `int getId(String URI)` returns the numerical identifier assigned to a given `URI`.

4 Conclusion

The paper describes an approach to mapping ontology into relational database. The process of mapping is autonomous that means that there is no need of human interaction. We present algorithms of translation ontology into relational scheme and relational data model. In this paper we focused on implementations of the mentioned algorithms – transformation of algorithm's steps in SQL statements.

We work with so called $\mathcal{EL}$ description logic which contains top, intersect and full existential quantification constructor. We would like to extend the logic with additional concept constructors inspired by relational database operators.

The important area of our research relates to valid domains. We want to do a research about valid domains and the assumption that valid domains preserve ISA-hierarchy.

Acknowledgment

This paper was supported by Slovak project VEGA 1/0131/09, Slovak project VVGS/UPJ Š/45/09-10, Czech project GAČR 201/09/0990 and Czech project IS 1ET100300517.

References

1. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook. Theory, implementation, and application.* Cambridge University Press, 2003 United Kingdom.
2. J. Dokulil, J. Tykal, J. Yaghob and F. Zavoral. Semantic Web Repository and Interfaces. *International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies.* In Proc. of UBICOMM 2007, IEEE Computer Society, 2007, pp. 223–228.
3. T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. *Combining Answer Set Programming with Description Logics for the Semantic Web.* Artificial Intelligence Vol. 172, Issues 12–13, 2008, pp. 1495–1539.
4. A. Gali, C. X. Chen, K. T. Claypool, and R. Uceda-Sosa. *From Ontology to Relational Databases.* Springer Verlag, LNCS 3289, 2004, pp. 278–289.
5. B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. *Description Logic Programs: Combining Logic Programs with Description Logic.* In Proc. of WW2003, Hungary, 2003, pp. 48–57.
6. N. Kottmann and T. Studer: *Improving semantic query answering.* DEXA 2007, LNSC 4653, Springer, 2007, pp. 671 – 679.
7. Z. Pan and J. Heflin: *DLDB: Extending relational databases to support semantic web queries.* In Workshop on Practical and Scaleable Semantic Web Systems, ISWC 2003, 2003, pp. 109–113.
8. J. Pokorný, J. Pribolová, and P. Vojtáš. *Ontology Engineering Relationally.* Technical Report 2009-2, Dep. of Software Engineering, Faculty of Mathematics and Physics, Charles University, 2009, 20 p.
9. J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. Relational databases for querying xml documents: Limitations and opportunities. In Proceedings of 25 VLDB Conference, 1999, pp. 302–314.
10. HAWK. http://swat.cse.lehigh.edu/projects/index.html#hawk
11. MySQL. `http://www.mysql.com/`.
12. Sesame. `http://www.openrdf.org/`.
13. SPARQL. `http://www.w3.org/TR/rdf-sparql-query/`.
14. SWAT Projects - the Lehigh University Benchmark (LUBM). `http://swat.cse.lehigh.edu/projects/lubm/`.

Various aspects of user preference learning and recommender systems[⋆]

Alan Eckhardt

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

Abstract. In this paper, we describe area of recommender systems, with focus on user preference learning problem. We describe such system and identify some interesting problems. We will compare how well different approaches cope with some of the problems. This paper may serve as an introduction to the area of user preference learning with a hint on some interesting problems that have not been solved yet.

Keywords: user preference learning, data mining, recommender systems

1 Introduction

User preference learning is an important part of any recommender system. We will work with a scenario of user searching for some object (we will refer to user as "she" for not having to distinguish between he and she). Recommendation may help user to find what she is looking for more quickly and efficiently, because she has not to crawl through hundreds of products but sees the recommended products on only one or two pages. Of course, these recommended products may not be an exhaustive list, but they are a hint for user.

In Section 2 some important related work is studied, providing also an introduction to the problematic of user modeling. Then, in Section 3, we describe how user preferences are modeled in our approach. In Section 4 is described a typical scenario of recommendation cycle for user. We also describe how our user model is constructed and ways for estimating usefulness of a user model. In Section 5 are listed some interesting problems associated with learning of user preferences. Finally, in Section 6 are conclusive remarks and more importantly areas for future work in this field are proposed.

1.1 Example

In the whole paper, we will refer to a set of "objects". These objects are supposed to be of interest for user, probably she wants to buy one. In our traditional example, user is

[⋆] This paper was supported by Czech projects MSM 0021620838 and 1ET 100300517.

buying a notebook. She has some preferences of notebooks, e.g. the maximal price she is willing to pay, the preferred manufacturer or the size of the display.

This example is suitable for our approach because notebooks have well defined attributes that describes the product completely. More about this is in Section 4.1.

1.2 Notation

We will work with a set of objects X. Set X can be also viewed as a set of identifiers of objects (id), which will be often referred to as o. Every object has attributes $A_1, ..., A_N$ with domains $D_{A_1}, ..., D_{A_N}$. If we want to specify the value of an attribute A_i for an object o_j, we will use notation $A_i(o_j)$. We will use $X^i(a)$ when denoting a set of objects for which attribute A_i has the value a. When the attribute will be clear from context (which will be most of the times), we will use only $X(a)$.

2 Related work

User preference modeling was very nicely described in [4] and also in a more general view in [10]. In Figure 1 (which was taken from [4]) are various components of preference modeling. Model is how user preferences are understood – for our purposes Total order of outcomes (or objects) will be most suitable. That means we can create a list of all objects ordered according to user preferences. Language is a way for user to express her preferences. It may be a rating of an object, as $V(o)$ in Figure 1, or a query to the system etc. Language is explored in Section 4.1.

The most interesting part for us is Interpretation, where the information from user is somehow transformed into Model, e.g. the total order of outcomes. However, because of intuition, we will slightly change the notation – we will refer to the method for creating the total order as "user model" or "user preference model". Interpretation may be also viewed as learning phase, where a user preference model is constructed.

In the following two sections, two alternative ways of user modeling are described along with their possible interpretations. First, qualitative models are based on comparing two objects between them, and second, quantitative models are based on evaluation of a single object with a scoring function.

2.1 Qualitative approaches

Preference relations are the most used and studied qualitative approach. There is a huge amount of related work in the field of preference relations. Preference relations represent preferences as a relation between two objects, it is usually assumed that this relation creates some pre-order on X. There are typically three relations, P are strict preferences, I represents indistinguishability of objects or equality of preference and R is union of P and I meaning that it represents non-strict preferences. For example $P(o_1, o_2)$ means that o_1 is strictly preferred to o_2, $I(o_1, o_2)$ on the other hand means that o_1 and o_2 have the same preference and finally $R(o_1, o_2)$ means that o_1 is preferred or equal to o_2.

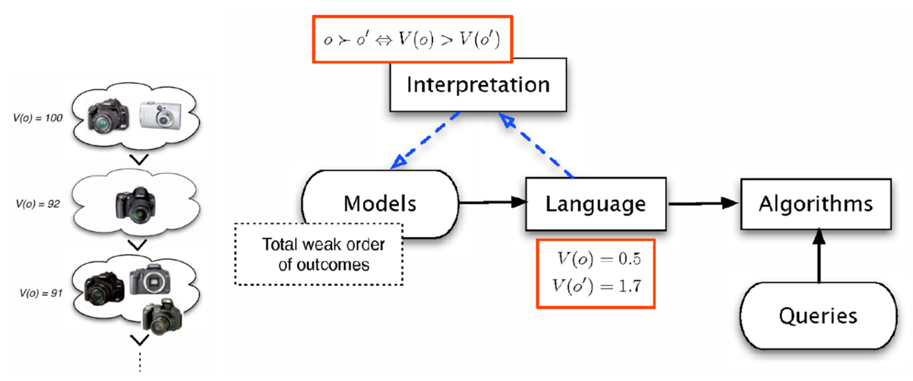


Fig. 1. Preference model components.

Preference relations in database systems and their integration into SQL by preference queries was studied by Chomicki in [8], [9]. Also Kießling contributed to this field with [25].

A different approach was suggested by Kießling in [19], [24]. This approach is based on the idea of preference relations but it uses relations over attribute values rather than relations over whole objects. This is more like our approach based on fuzzy logic. However Kießling does not use scoring functions but uses special predicates POS, NEG etc. to represent relation between two attribute values. An example from [24] is from the area of cars: POS(transmission, automatic) and NEG(make, Ferrari) meaning that automatic transmission is preferred to any other type and any maker is preferred to Ferrari.

As for learning of preference relations, a great contribution is from Fürnkranz and Hüllermeier [17], [20].

2.2 Quantitative approaches

The other approach, also adopted by us, is quantitative. It sorts objects by a score defined by a scoring function. This approach is arguably less expressive than the qualitative one – it can not express e.g. a cycle in preferences. There are also some very interesting works in this area.

Content based models Content based models uses attributes of object for construction of scoring function. For example Fagin in [16] proposed a way of combining numerous fuzzy inputs. Another classical work is from Agrawal [6].

Collaborative filtering Besides content based models, such as the one presented in Section 3, there is another widely used user model that is based on the preferences of other users. Collaborative filtering was proposed in early 90's in [18] and further developed. One of the well-known systems using collaborative filtering is GroupLens [27].

Collaborative filtering is based on the idea of similarity of users. When we want to know how user u_1 will like object o_1, one way is to look how other people liked o_1. `Amazon.com` succeeds in describing this approach in one sentence "Customers Who Bought This Item Also Bought...".

The better way is to restrict only to those users that are similar to u_1. The similarity may be computed in various way, the most common is the similarity of ratings of objects other than o_1. Other possibility is to compute the similarity of user profiles – e.g. find managers, from 25 to 30, divorced, with interest in psychology and computer science. There is a hidden assumption that similarity in profile imply similarity in preferences, which may not be always true.

3 User model based on fuzzy logic

In this section, we describe user model we are using. This model is based on a scoring function that assigns every object a score that represents the rating of that object. User rating of an object is a fuzzy subset of X, i.e. a function $R(o) : X \rightarrow [0,1]$, where 0 means least preferred and 1 means most preferred. Our scoring function is divided into two steps.

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

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

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

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

where f_A are fuzzy sets for normalization of attribute A.

Another totally different approach was proposed in [15]. It uses the training dataset as partitioning of normalized space $[0,1]^N$. For example, if we have an object with normalized values $[0.4, 0.2, 0.5]$ with rating 3, every other object with better attribute values (e.g. $[0.5, 0.4, 0.7]$) is supposed to have rating at least 3. In this way, we can find the highest lower bound on any object with unknown rating. In [15] was also proposed a method for interpolation of ratings between the objects with known ratings and even using the ideal (non-existent) virtual object with normalized values $[1, ..., 1]$ with rating 6.

In other words, we can say that the pareto front is constructed in the first step. Pareto front is a set of objects that are not dominated by any other object. We say that object o_1 dominates object o_2 iff $\forall i = 1, ..., N : f_i(o_1) > f_i(o_2)$, i.e. o_1 is better in every attribute than o_2. In the second step, we choose the best object from the pareto front.

4 A recommender system

A recommender system tries to help user to find the object she is looking for. It is necessary for user to transfer some information about her preferences to the system. It is convenient for user to describe her preferences in an intuitive and simple way. The more complex user interface is, the more structured information the system gets but much less users will use it (according to [26], as little as 9 out of 260 people provided a feedback to their system). This fact also penalizes preference relations – user is supposed to compare two objects, but the number of couples is quadratic to the number of objects.

There is an example how a recommender system may work in Figure 2. System presents user with a set of objects S_0. User rates some of these objects and this information is sent back to the system as feedback U_0. User model is constructed from user's ratings and a personalized set S_1 is sent to user. Again, user rates some objects (U_1) and system updates its user model and sends S_2 and this cycle may go on until user is satisfied or bored. In Section 4.1 various possible types of feedback from user are studied. In Section 4.3 and 4.2 the construction and update of user model is described.

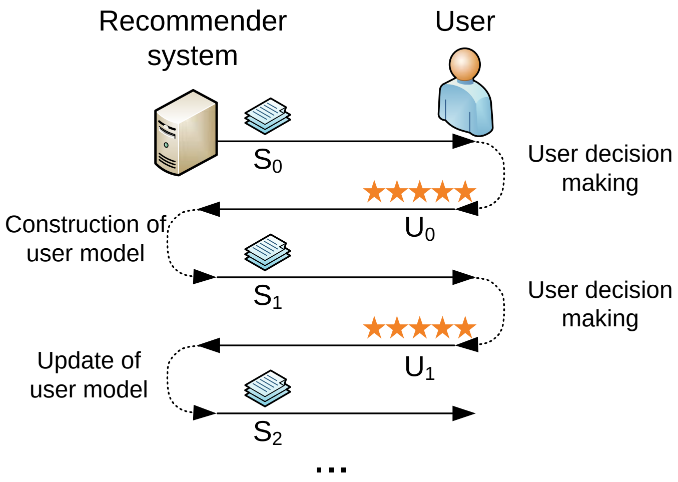


Fig. 2. A use of a recommender system in steps.

The process of recommendation is in Figure 3. User does some actions with the system, which are processed by various components of the system. You can see that some inputs may be processed by multiple components. There are three examples in the figure – analysis of user behaviour, collaborative filtering, analysis of ratings and direct query. Each of these components then creates user model which is used for prediction of preference of all objects. Some models, like collaborative filtering, use the information about other users or other additional information. All these models are then combined together to provide most precise recommendation for user. Furthermore, when the system identifies some of the situation discussed in 5, it can favour the model that behaves best in this situation or the other way round. Collaborative filtering is not good when there is a small number of users, so if that is the case, it can be disfavoured.

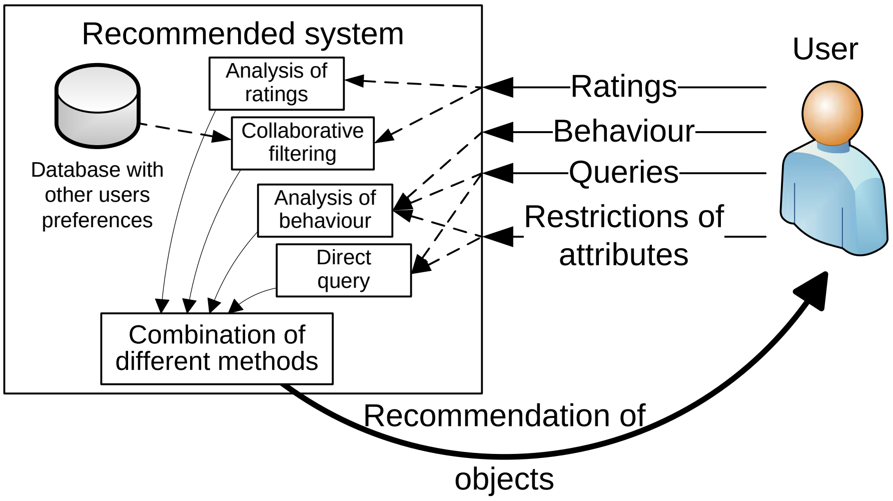

Recommended system
Analysis of ratings
Collaborative filtering
Database with other users preferences
Analysis of behaviour
Direct query
Combination of different methods
User
Ratings
Behaviour
Queries
Restrictions of attributes
Recommendation of objects

Fig. 3. Structure of a recommender system.

4.1 Input from user

From the information user provides to the system, her user model is built. User model should be capable to determine which objects user will like or to what degree an object will be preferred. The construction of user model is of most interest for us. We are working with user ratings. These ratings user associates to a small number of objects. This is key aspect of user model construction – it can not be expected that user will rate hundreds of objects. When doing experiments, we often limit the size of training set to 40 objects.

Other approaches may expect different forms of information from user other than ratings. For example for preference relations, comparisons between two objects is the expected input. The full-text query issued by user may be also viewed as a source of information about what user wants – document retrieval uses queries as its only information from user.

Datasets There exists publicly available datasets of user ratings such as Netflix [3]. In these datasets exist users with even thousands of ratings. Unfortunately, most of these datasets have a very small number of attributes.

- Books [5] - have author, title, year of publication and publisher. It contains 433 670 ratings with non-zero rating from 77 805 users.
- Jokes [23] - have no attributes, except the text of the joke itself. It contains 4.1 million of ratings by 73 421 users.
- Films [2], [3] - have many attributes (from IMDB [1]) but they are complicated. One film can have many actors, many producers, many directors etc. The normal attributes such as length of a film are not determined easily, because they differ across countries, editions or releases. Movielens contains 10 million ratings from 71 567 users. Netflix contains over 100 million ratings.

All this is data from users who were using a system for a long time, several years in most cases.

Behaviour analysis User behaviour interpretation was studied in [21] and [22]. Because user tends not to give very much information about herself, the interpretation of her behaviour may provide a useful information that supports the explicit actions she had done (such as ratings of objects). There are many events that can be monitored, such as the time spent on a web page with details of an object, clicking on a page, scrolling down a document, filtering the content of the page, issuing a query etc. These actions are then interpreted as if they were motivated by her preferences, e.g. the longer user stays on a page, the more preferred is the object on that page. When user browses the shop by categories or restricts values of some attribute, it is a clear statement what she likes. For example when user narrows her search to notebooks with display size 14", we can deduce that 14" is the best size of display. This information helps when creating local preferences (in Section 4.2). The order in which such restrictions are applied may represent importance of attributes. Typical counter-example against interpretation of behaviour is when user is going for a coffee, leaving the browser open on that page, or a user searching for an object for a friend (see also Section 5 for this issue).

In following sections, we outline some methods for creating local and global preferences, which are contributions we made to this area in the past.

4.2 Learning local preferences

The acquisition of local preferences differs for different types of attributes. For nominal attributes, we use a method based on representative value that is computed from user ratings [11, 13].

For numerical attributes, the problem is more complicated – there is usually only one object with value a. A typical example is price – there is often only one notebook with price e.g. 941$. We can stick with traditional methods such as linear regression, where the input is formed from the ratings of objects and their prices. However linear regression may be affected by the distribution of data, but we want a function that corresponds to real values across the whole domain, not only there where are most values. Because of this, we proposed a way of using representants for numerical domain in [11].

Lately, we identified another problem with numerical domains. It may happen that a value of another attribute, often nominal, affects the normalization of a numerical attribute. For example, when flying with British Airways, you may prefer lower price, because the comfort is fine in economy class, but with Aeroflot, you may prefer higher price because the overall comfort is worse. This phenomenon is related to ceteris paribus preferences [28] and CP-nets [7].

4.3 Learning global preferences

Method called "Statistical" was described in [11] and in [12]. It is based on the evaluation of distribution of ratings across attribute domain D_A and from this distribution it is possible to derive weight that A plays in the decision process of user. Then the weighted average with these weights is used as aggregation function.

When constructing scoring function called "Instances" that uses the objects from training set as lower bounds (from [15]), there is little to do. It does not need any further transformation or analysis. This is unfortunately balanced by a more complicated computation during evaluation of new objects.

4.4 How to measure usefulness of a recommender system

In this section we identify several ways of measuring usefulness of a recommender system. As in data-mining, we adopt the idea of training (Tr) and testing sets (Ts). User model is constructed from objects in training set and then its performance is measured on testing set. In the following, the real preference of an object o will be denoted as $R(o)$ and preference user model estimates as $\widehat{R}(o)$.

RMSE RMSE stands for root mean squared error. It is widely used as error measure in data-mining community. It is computed as $\sqrt{\sum_{o \in Ts} (\widehat{R}(o) - R(o))^2 / |Ts|}$.

When considering user preferences, we introduced a modified RMSE, weighted RMSE. It associates more weight to the preferred objects than to the non-preferred. The formula is $\sqrt{\sum_{o \in Ts} R(o) * (\widehat{R}(o) - R(o))^2 / \sum_{o \in Ts} R(o)}$. The less is RMSE, the better the system performs.

Tau coefficient Tau coefficient is known in economy. It is used to measure the similarity of the ordered lists of objects. It is assumed that both lists contain the same set of objects. For our purpose, we compare the ordering of objects by real user preferences $R(o)$ with the ordering user model would make $\widehat{R}(o)$.

Tau coefficient is based on concordant and discordant pairs. A pair (o_1, o_2), (p_1, p_2) is concordant, if $sgn(R(o_1) - R(o_2)) = sgn(\widehat{R}(p_1) - \widehat{R}(p_2))$, where sgn is signum function. Then the coefficient is computed as $\tau = \frac{n_c - n_d}{\frac{1}{2}n(n-1)}$ where n_c is number of concordant pairs, n_d is the number of discordant pairs and n is the number of objects in lists. The higher Tau coefficient is, the better for the system.

We can apply weighting scheme to Tau coefficient, too. Objects with higher real preferences will matter more than objects with low preference, when comparing the two orderings.

ROC curves ROC curve is a method for capturing the performance of a system under different conditions. In terms of classification of positive and negative examples, it tells how the recall (the ratio between the number of correctly classified positive objects and the total number of positive objects) of the system increases if we relax the ratio of correctly classified negative objects and the number of all negative objects.

The problem here is that we do not have a simple positive-negative scenario. Ratings have typically the domain {1,2,3,4,5}. The possible solution is to consider several cuts in this domain and measure at each of these cuts. We can take as positive objects with rating 5 and the rest as negative. Then, objects with ratings 5 and 4 will be considered positive and the rest as negative etc. In this way, we will get 5 different ROC curves.

Amount of information required from user Another important characteristic of a recommender system is amount of information that is required from user. Possible approaches are described in Section 4.1. Often, the higher precision of a system is outweighed by a larger investment from user. But the intuition says that few users will be willing to perform complicated tasks, no matter the increase of helpfulness of the system. The less information the system needs from user, the better for user.

Improvement in user ease of work with the system The main task of recommender systems is to help the user. All the above mentioned measures are good because they can be computed analytically, but they cannot capture the real improvement for user. This improvement can be only given by the real user working with the various recommender systems and comparing them between them. There are some problems with this comparison

- Every man perceives the ease of work differently. There should be a large number of people testing the systems to be compared to evaluate overall suitability.
- The system has to be implemented including the user interface. Different methods work with different inputs from the user and they have to be able to monitor these inputs.
- People testing the system has to have a motivation to work with the system. If they are not, they would not be critical or they would apply some criteria irrelevant in the real usage of the system. This is most difficult to achieve.

5 Interesting problems of recommender systems

In this section, some interesting problems are described. All these problems relate to user preference learning and can be viewed as the main contribution of this paper, besides the introductory part.

Cold start When a system is based on user ratings, it has serious troubles in the beginning of its existence. This problem is most pronounced for collaborative filtering-based system, because they need a lot of rating from a lot of users for making correct predictions. For content based recommendation, this is not much of a problem because the accuracy of recommendation does not depend on the number of users of the system. The following issue however affect both approaches.

New user When a new user starts using the system, it is hard to recommend something because the system has little information about her. This issue may be overcome with a default user profile, but this solution is not personalized enough.

There is a possibility to learn something quickly about a new user – it is by choosing a good set S_0 in Figure 2. When there are very different objects in S_0, we can learn immediately what user certainly does not like and what area is of her interest. When S_0 is constructed randomly, there is a higher possibility that there would not be any object the area of user's interest. Suitable S_0 may be found for example using clustering.

Identifying context for user When a user works with the system, it is assumed that she searches for some object she wants. But there are other possible motives such as scanning the area or finding some object for a friend. This context of work deeply influences user behaviour and may severely damage existing user model.

Small rating scale Typical rating is expressed on the scale $\{1,2,3,4,5\}$. When having thousands of objects, this scale is far too small for capturing the ordering among the best objects. We proposed a method Phases for overcoming this obstacle in [14], but it is not a complete analyze and there are many more aspects of this problem. Phases are suited for a single session, for long-term usage there should be another way of differentiating among the objects with rating 5.

This problem could be of course solved by enlarging the scale of ratings to e.g. $\{1,...,100\}$ but its benefit is questionable. Will user really feel the difference between ratings 64 and 65? Will user be able to apply it consistently?

Negative preferences In our model, 0 means the lowest preference. When a manufacturer ASUS has the preference 0, it penalizes the notebook, but in the overall score it may be outweighed by other attributes which are more preferred by user. Negative preferences are typically more stronger than this – a notebook made by ASUS is strongly disregarded. The strength may be expressed by the weight of the attribute, but it is no solution, because for other manufacturers, this attribute may not have such a big weight. Our task is to model a better way of penalizing objects with a highly non preferred value and the way of expressing such strong negative preference.

Time dimension Main problem with acquisition of user preferences is the small size of training data. When user is using the system for a longer period, the training data may get bigger. But here comes a new problem – user preferences are typically unstable. What user preferred a month ago may not be good today. But the high rating user provided a month ago is the same high rating provided today.

There is a need to find a method for penalization of older ratings or promotion of newer ones. The system needs to guess whether to apply the penalization at all – maybe the preferences stay the same when buying a house, but they may change quickly for a movie.

6 Conclusion

We have described a recommender system and identified several interesting issues and problems that occur during user model construction. The main contribution is the suggestion of several possible ways how to measure helpfulness of a recommender system. The experimental evaluation is often lacking in more theoretical focused papers. Also the list of some problems or typical situations in recommender systems may be inspiring for searching of their solutions. We hope that this analysis would be helpful to anyone new in recommender systems and user modeling field.

6.1 Issues for future work

All problems in Section 5 are worth studying, but we would like to address the two last problems in near future – that of Negative preferences and Time dimension.

References

1. The internet movie database. *http://www.imdb.com/.*
2. *Movielens dataset. http://www.grouplens.org/node/73.*
3. *Netflix dataset. http://www.netflixprize.com.*
4. *Preference handling an introductory tutorial. http://www.cs.bgu.ac.il/ brafman/tutorial.pdf.*
5. Improving Recommendation Lists Through Topic Diversification, *2005.*
6. *R. Agrawal and E. L. Wimmers. A framework for expressing and combining preferences.* SIGMOD Rec., *29(2):297–306, 2000.*
7. *C. Boutilier, R. I. Brafman, H. H. Hoos, and D. Poole. Cp-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements.* Journal of Artificial Intelligence Research, *21:2004, 2004.*
8. *J. Chomicki. Preference queries.* CoRR, *cs.DB/0207093, 2002.*
9. *J. Chomicki. Preference formulas in relational queries.* ACM Trans. Database Syst., *28(4):427–466, 2003.*
10. *J. Doyle. Prospects for preferences.* Computational Intelligence, *20:111–136(26), May 2004.*
11. *A. Eckhardt. Inductive models of user preferences for semantic web. In J. Pokorný, V. Snášel, and K. Richta, editors,* DATESO 2007, *volume 235 of* CEUR Workshop Proceedings, *pages 108–119. Matfyz Press, Praha, 2007.*

12. *A. Eckhardt, T. Horváth, D. Maruščák, R. Novotný, and P. Vojtáš. Uncertainty issues in automating process connecting web and user. In P. C. G. da Costa, editor,* URSW '07 Uncertainty Reasoning for the Semantic Web - Volume 3*, pages 97–108. The 6th International Semantic Web Conference, 2007.*
13. *A. Eckhardt, T. Horváth, and P. Vojtáš. Learning different user profile annotated rules for fuzzy preference top-k querying. In H. Prade and V. Subrahmanian, editors,* International Conference on Scalable Uncertainty Management*, volume 4772 of* Lecture Notes In Computer Science*, pages 116–130, Washington DC, USA, 2007. Springer Berlin / Heidelberg.*
14. *A. Eckhardt, T. Horváth, and P. Vojtáš. PHASES: A user profile learning approach for web search. In T. Lin, L. Haas, R. Motwani, A. Broder, and H. Ho, editors,* 2007 IEEE/WIC/ACM International Conference on Web Intelligence - WI 2007*, pages 780–783. IEEE, 2007.*
15. *A. Eckhardt and P. Vojtáš. Considering data-mining techniques in user preference learning. In* 2008 International Workshop on Web Information Retrieval Support Systems*, 2008.*
16. *R. Fagin. Combining fuzzy information from multiple systems. pages 216–226, 1996.*
17. *J. Fürnkranz and E. Hüllermeier. Pairwise preference learning and ranking. In* In Proc. ECML-03, Cavtat-Dubrovnik*, pages 145–156. Springer-Verlag, 2003.*
18. *D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry.* Commun. ACM*, 35(12):61–70, 1992.*
19. *S. Holland, M. Ester, and W. Kiessling. Preference mining: A novel approach on mining user preferences for personalized applications. In* Knowledge Discovery in Databases: PKDD 2003*, pages 204–216. Springer Berlin / Heidelberg, 2003.*
20. *E. Hüllermeier and J. Fürnkranz. Learning preference models from data: On the problem of label ranking and its variants. In G. D. Riccia, D. Dubois, R. Kruse, and H. Lenz, editors,* Preferences and Similarities*, pages 283–304. Springer-Verlag, 2008.*
21. *T. Joachims. Optimizing search engines using clickthrough data. In* KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142, New York, NY, USA, 2002. ACM Press.*
22. *T. Joachims, L. Granka, B. Pan, H. Hembrooke, and G. Gay. Accurately interpreting clickthrough data as implicit feedback. In* SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 154–161, New York, NY, USA, 2005. ACM.*
23. *G. K. Eigentaste: A constant time collaborative filtering algorithm.* Information Retrieval*, 4:133–151(19), July 2001.*
24. *W. Kiessling. Foundations of preferences in database systems. In* VLDB '02: Proceedings of the 28th international conference on Very Large Data Bases*, pages 311–322. VLDB Endowment, 2002.*
25. *W. Kiessling and G. Köstler. Preference sql: design, implementation, experiences. In* VLDB '02: Proceedings of the 28th international conference on Very Large Data Bases*, pages 990–1001. VLDB Endowment, 2002.*
26. *S. E. Middleton, N. Shadbolt, and D. D. Roure. Capturing interest through inference and visualization: Ontological user profiling in recommender systems. In* K-CAP2003*, 2003.*
27. *A. M. Rashid, I. Albert, D. Cosley, S. K. Lam, S. M. McNee, J. A. Konstan, and J. Riedl. Getting to know you: learning new user preferences in recommender systems. In* IUI '02: Proceedings of the 7th international conference on Intelligent user interfaces*, pages 127–134, New York, NY, USA, 2002. ACM.*
28. *G. H. Wright. The logic of preference reconsidered. In* Theory and Decision*, volume 3, pages 140–169, 1972.*

Using Top Trees For Easy Programming of Tree Algorithms

Michal Vajbar

Department of Software Engineering, Faculty of Mathematics And Physics, Charles University in Prague, Malostranské nám. 25, 118 00, Praha 1, Czech Republic
`michal.vajbar@mff.cuni.cz`

Abstract. Top trees are a dynamic self-adjusting data structure that can be used by any tree algorithm. Actually, an arbitrary number of different tree algorithms can use a single structure. In our previous work, we have implemented top trees, but the usage still requires a detail knowledge of the structure which is quite complex. In this paper, we introduce Top Tree Friendly Language (TFL) and Top Tree Query Language (TQL). The TFL is a special programming language which combines declarative and procedural approaches that results in simpler and faster algorithm designing. The query language TQL provides an easy top trees administration. The implementation of top trees, the programming language TFL and the query language TQL together form a complex solution for using top trees.

Keywords: data structure, top trees, programming language, query language

1 Introduction

There exist many algorithms that work over tree graphs. Any of them can use different data structures to represent the same tree. If we need to run several of these algorithms over one forest together each one could use its own representation of the forest. Moreover, if the forest dynamically changes over the time by edge insertions and deletions all these changes have to be reflected into the data structures of all algorithms. This is not efficient. Thus several data structures have been proposed to solve this problem. Any of them is good at some properties, but each has a weak point. It appears that top trees provide the most acceptable trade-off. In this paper, we present a complex solution that allows easy using of top trees. The solution is built on our implementation of the structure [7] and it brings Top Tree Friendly Language (TFL) and Top Tree Query Language (TQL). The TFL is a special programming language which combines declarative and procedural approaches that results in simpler and faster algorithm designing. The query language TQL provides easy top trees administration.

The paper is organized as follows. In Section 2 we introduce top trees. Used implementation of top trees is shortly presented in Section 3. Section 4 introduces the TFL programming language and shows how to use it. In Section 5 we present the query language TQL. Final remarks are made in Section 6.

2 Top Trees

Top trees are a dynamic self-adjusting data structure that was proposed by Alstrup et al. [1]. A top tree R is an ordinary binary tree with a root. It is used to represent a tree graph T with defined information that some tree algorithm works with. The structure R consists of clusters. The edges of the T form basic clusters and two clusters connected through a common vertex create other cluster. So the R records a way of clusters connecting into one root cluster that represents whole T. Each cluster holds information of appropriate part of the T. A manner how the information is computed during joining and splitting of clusters characterizes used algorithm. That is a simplified idea. In the rest of this section, the structure is described more precisely.

2.1 Formal Definition and Properties

The structure is defined over a pair consisting of a tree T and a set ∂T of at most two vertices from T that are called *external boundary vertices*. Let $(T, \partial T)$ is the pair, then any subtree C of T has a set $\partial_{(T,\partial T)} C$ of at most two *boundary vertices* from C. Each of them is either from ∂C or incident to an edge from $T \setminus C$. The subtree in undirected graph means any connected subgraph. The subtree C is called a *cluster* of $(T, \partial T)$ if it has at least one edge and at most two boundary vertices. Then T is also cluster and $\partial_{(T,\partial T)} T = \partial T$. If A is a subtree of C then $\partial_{(C,\partial_{(T,\partial T)} C)} A = \partial_{(T,\partial T)} A$. This means that A is a cluster of $(C, \partial_{(T,\partial T)} C)$ if and only if A is a cluster of $(T, \partial T)$. We will use ∂ as a shortcut for $\partial_{(T,\partial T)}$ and also for all subtrees of T.

Definition: A top tree R over $(T, \partial T)$ is a binary tree such that:

1. The nodes of R are the clusters of $(T, \partial T)$.
2. The leaves of R are the edges of T.
3. Two clusters are called *neighbours* if they intersect in a single vertex. Their union is a *parent cluster* (Fig. 1).
4. The root of R is T itself.

Top tree R represents whole T. If T consists of a single vertex then it has an empty top tree. The edges of T are basic building blocks of the clusters and the vertices represent endpoints of the clusters. That is why the cluster is consisted of at least one edge. The neighbouring clusters are edge-disjunct with one common vertex (Fig. 1).

Let C is a cluster. A vertex v is an *internal vertex* of C when $v \in C$ and $v \notin \partial C$. If C has two boundary vertices a and b then C is called a *path cluster* and the path $a \ldots b$ is called the *cluster path* of C. If C has one boundary vertex a then C is called a *point cluster.*

Top tree nodes contain pointers to their sons and parents. Each node represents a cluster and there is a set of at most two boundary vertices associated with the cluster. The leaf nodes represent the edges. The non-leaf node holds information how it is joined from its sons (Fig. 1). According to this information it is possible to construct any cluster.

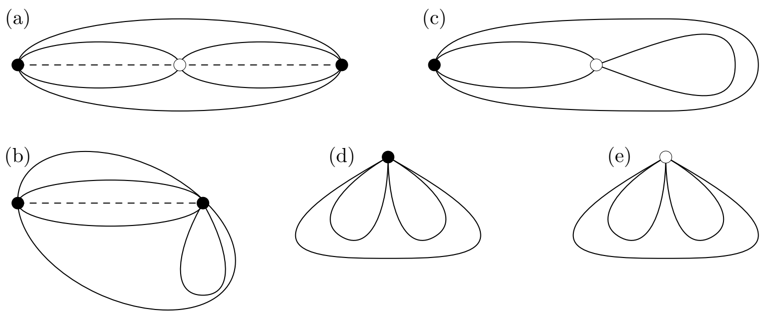


Fig. 1. The cases of joining two neighbouring clusters into the parent cluster (assumed from [1]). The • are the boundary vertices of the parent cluster. The ∘ are the boundary vertices of children clusters that did not become the boundary vertices of the parent. The dashed line presents the cluster path of the parent cluster. Moreover, there exist symmetric variants for (b) and (c).

2.2 Supported Operations

When we work with a forest (a set of trees) then each tree of the forest is represented by one top tree. Given vertices v and w, the whole forest is controlled by following operations:

link(v, w) – If v and w occur in different trees then a new edge (v, w) is created.
cut(v, w) – If the graph contains the edge (v, w) then the edge is removed.
expose(v, w) – If v and w occur in the same tree then the top tree is rebuilt so that v and w become the external boundaries and the path $v \dots w$ becomes the root path (Fig. 1a, 1b). It is possible to call expose with one vertex to create the root cluster with only one external boundary (Fig. 1c, 1d).
select(R) – A special operation for nonlocal algorithms. The R is a top tree. This operation is described in the Section 2.4.

A number of clusters can be changed by the operations above. These changes are executed by *local operations*:

create$() = e$ – Creates a top tree with only one cluster e consisted of one edge.
join$(A, B) = C$ – A and B are the neighbouring root clusters of top trees R_A and R_B. This operation creates a new cluster $C = A \cup B$ which represents a root cluster of the top tree R_C consisted of R_A and R_B.
split$(C) = (A, B)$ – The C is the root cluster of a top tree R_C. The *split* removes C and divides the top tree R_C into top trees R_A and R_B. The clusters A and B are created from the C as the root clusters of the new top trees.
destroy$()$ – This operation removes the cluster represented by one edge.

The operations *link*, *cut* and *expose* invoke a sequence of the local operations. Each local operation always changes just one cluster. The root cluster may represent only a part of the tree T during the rebuilding, but in the end it represents whole T again.

2.3 Usage of Top Trees

Algorithms working over top trees associate information that they need with clusters or vertices. The local operations describe how to deal with the information during the changes. Then to describe the algorithm it is enough to define the manner of the local operations.

To keep the structure in a consistent state it is possible to change information only in the root cluster or in the external boundary vertex. So the top tree must be rebuilt by the *expose* operation when we want to change information saved in the cluster specified by a path $v \ldots w$ or by a vertex v. Then the path becomes the root path and specified vertices become external boundary vertices. The structure does not allow to change any information outside of the root cluster.

2.4 Types of Algorithms

Tree algorithms can be classified as local or nonlocal. The *local algorithms* deal with local properties of the tree. The local property is defined in the following way: if the property holds for an edge or a vertex in the whole tree T then it holds for the same edge or the vertex in any subtree of T. An example can be the searching for the heaviest edge or the finding out the distance between two vertices. The local properties can be computed easily in bottom-up manner in the top tree. The *nonlocal algorithms* are different. They deal with nonlocal properties: this kind of property can be held by one edge (or a vertex) in the whole tree and by another one in any subtree. For example the searching for the center or the median of the tree. Evidently, the using of top trees for nonlocal algorithms is more difficult than for local algorithms.

Top trees have to support a fourth operation called *select* to enable nonlocal algorithms running. This operation is very complex so only basic idea is showed here. The detailed description can be found in [6]. The *select* gets a top tree R and it picks one of the children clusters of the root cluster. It is important to realize that the R represents the whole tree T. The operation works recursively and it is similar to the binary searching. One child of the root cluster is selected and this child becomes the root for next iteration. The top tree is rebuilt to prepare the new root after the selection (details in [6]). So the new root still represents the whole tree T. The finding finishes when a basic cluster that represents one edge is found. This cluster is the intersection of all selected clusters.

3 Used Implementation

We have developed an implementation of top trees in our previous work [7]. Our implementation results from Tarjan and Werneck's work [5, 8]. It combines tree decomposition used in Sleator and Tarjan's *ST-trees* [4] with compression and raking used in Frederickson's *topology trees* [2]. These principles ensure the representation of a tree graph T and bring the great idea of clustering. For more details see [7].

In our implementation, all four operations are supported in $O(\log n)$ amortized time where n is a number of vertices. This time analysis is proved in [5] for *link*, *cut* and *expose*. For *select* it is proved in [7].

The data structure and both languages TFL and TQL were developed in Java language. Source codes with examples and documentation are available on author's homepage[1].

4 Top Tree Friendly Language (TFL)

In our previous work [7], we have implemented top trees, but the usage still requires a detail knowledge of the structure which is quite complex. So we have decided to develop a declarative programming language that makes the structure available to users with only a basic knowledge.

The declarativity saves users from technical details of the implementation and it allows to focus on the designing of a tree algorithm. So hence it came the name of the language - Top Tree Friendly Language (for short *TFL*).

We did not try to develop the almighty programming language. The aim was a language that simplifies the designing of algorithms. The TFL allows to write a simple and short source code quickly and then easily verify its functionality with the TQL (Sec. 5).

4.1 Basics of TFL

The source code of the language consists of several blocks that form two sections. In the first section, there is declared a name of the algorithm and information stored in vertices and clusters. A description of the designed algorithm occurs in the second section of the script. A line or block comment can be used anywhere.

In the declarative section, it is necessary to keep the order of three obligatory blocks - firstly the algorithm name in the *algorithm* block, then the declaration of the vertex in the *vertex* block and finally the declaration of the cluster in the *cluster* block. An example can be seen in Fig. 2.

The declaration of the vertex and the cluster contains a specification of variables to store information. The variables are described by a data type. All supported types are depicted in the section 4.3. In the *cluster* block, there it is possible to use a special array. The array holds information of the specified data type for two keys - the boundary vertices of the relevant cluster. The *cluster* is the only block where the arrays can be declared. This construction resulted from the need of some algorithms that require to hold different information for both boundary vertices in each cluster.

In the second section, there is described a behaviour of the algorithm. Two levels of the blocks are used here whereas the only one level occurs in the declarative section. The root blocks form the first level. They are *var*, *create*, *destroy*, *join*, *split* and *selectQuestion*. These blocks are not obligatory, but their occurrence has to correspond to the mentioned order. They describe the algorithm

[1] http://siret.ms.mff.cuni.cz/vajbar

```
/* The declaration of the algorithm name and saved information: */
algorithm { algorithmName } // 1. the name of the algorithm
vertex { // 2. the declaration of vertex information
  type1 variable1, variable2;
  type2 variable3; }
cluster { // 3. the declaration of cluster information
  type3 variable4;
  array(type2) variable5, variable6; }

/* The description of the algorithm: */
...
```

Fig. 2. The structure of the TFL source code.

during the local operations as their names imply. In the root blocks, there can be used only the blocks of the second level called auxiliary blocks. The *var* and the *selectQuestion* are exceptions without the local blocks.

The auxiliary blocks contain the procedural tools of the TQL language that are depicted in the section 4.4. They can be read as the sequences of the commands that are executed if some condition holds (e.g. the cluster represents a path). The blocks are explained in more detail in the following subsection.

4.2 Algorithm Description

The conception of the blocks corresponds to the train of thought during the algorithm designing. Thanks to blocks the process of the designing is divided into the smaller problems which can be solved more simply. That brings a declarative way of the programming, so an user takes care of the designing only. There is no need to take care of the technique how the result is achieved. Moreover, only basic procedural tools are needed to describe the work of the algorithm in the blocks.

The root block *var*. The *var* is a special kind of the root block. There are declared global variables that can be used in any other root block. The syntax of the declaration is the same as in the *vertex* block. Arrays are not allowed there.

The root blocks *create* and *destroy*. These blocks describe the algorithm during the local operations of the corresponding names. It is necessary to distinguish if the cluster represents a path or if it is a point cluster. So there are two auxiliary blocks:

`path` – the current base cluster is a *path cluster*
`point` – the current base cluster is a *point cluster*

A content of the block is executed if the condition above holds. These auxiliary blocks can occur in any order but each at most once.

The root blocks *join* and *split*. The blocks describe the algorithm during the appropriate local operations. There have to be considered the types of the parent and both children clusters. According to these types the behaviour of the algorithm can be described by following auxiliary blocks:

`path_child` – current descendant of the parent cluster is a *path cluster*
`point_child` – current descendant of the parent cluster is a *point cluster*
`path_parent` – parent cluster is a *path cluster*
`point_parent` – parent cluster is a *point cluster*
`path_and_path` – clusters represent the variant Fig.1a
`path_and_point` – clusters represent the variant Fig.1b
`point_and_path` – clusters represent a symmetry to the variant Fig.1b
`point_and_point` – clusters represent the variant Fig.1e
`lpoint_over_rpoint` – clusters represent the variant Fig.1c
`rpoint_over_lpoint` – clusters rep. a symmetry to the variant Fig.1c
`lpoint_and_rpoint` – clusters represent the variant Fig.1d

A content of the block is executed if the condition above holds. These auxiliary blocks form three groups: **_child*, **_parent* and the variants from the figure 1. In the *join* block, the groups have to occur in the mentioned order. In the *split*, the order of **_child* and **_parent* is switched. Within the scope of the groups the blocks can occur in any order.

The mentioned orders correspond to a succession that should be observed during the algorithm designing. When two neighbours are connected by the *join* then typically the data from them are prepared at first (**_child*) and then the data of the parent cluster are computed (**_parent*). If the algorithm needs more information about the clusters then the blocks from third group can be used. The procedure is analogical for the *split*. Firstly the data from the parent cluster are prepared (**_parent*) and then the data for the children are computed (**_child*). Eventually, the blocks from the third group can be used of course.

The root block *selectQuestion*. The last of the root blocks describes the way how the decision during one step of the *select* operation proceeds. The *selectQuestion* can be seen as the fifth local operation. Its content is slightly different from the other root blocks. There are no auxiliary blocks and the syntax follows:

```
selectQuestion {
  /* any code */
  if (condition) {
    /* any code */
    select a;
  }
  else {
    /* any code */
    select b;
  }
}
```

The command `select (a|b);` specifies the cluster that is selected. The a (b) denotes the left (right) child. The usage of the cluster names can be found in Section 4.5.

4.3 Supported Data Types

The language supports four data types - *integer*, *real*, *string* and *boolean*. It should be sufficient for the most of tree algorithms. The *string* represents the sequences of characters. The values must be surrounded by the quotation marks. The *boolean* type was included to allow working with logical values `true` and `false`.

The integral numbers are represented by the *integer* data type and the real numbers by the *real*. The language does not enable a casting between these two data types. The both numeral types support positive and negative infinity - `IpINF` and `ImINF` for the *integer* and `RpINF` and `RmINF` for the *real*. This differentiation between the types of the infinities results from the using of the auxiliary variables. It is described in the Section 4.5. The work with the infinities abides by the standard IEEE 754 [3].

The variables declared in the vertices and the clusters are initialized by the default values according to the data types. A null string `""` is the default for the *string* and the `false` for the *boolean*. The *integer* is initialized by integral zero `0` and the *real* by decimal zero `0.0`.

4.4 Constructions of Programming Language

By the procedural tools we try to cover usual needs of the algorithm designing. We have implemented a lot of tree algorithms over the top trees and it has revealed that there are only two essential commands. The first is the assignment of a value into a variable and the second is *if-else* construction. It seems to be very little, but it is not. There is no need to support more complicated constructions like *for*-loops or *while*-loops and the need of *switch*-block can be substituted by the *if-else* very easily.

The syntax of the *if-else* is the same as in other programming languages. The construction can contain any number of *elseif* parts and the *else* part is not necessary:

```
if (condition1) {
  /* any code */
} elseif (condition2) {
  /* any code */
} else {
  /* any code */
}
```

The language TFL supports basic arithmetic operations `+`, `-`, `*`, `/`. The division is integral for the *integer*. Numbers can be compared by `==`, `!=`, `<`, `>`, `<=` and `>=`. The *integer* and the *real* type cannot be combined anywhere. There is an operator `&` for *string* concatenation. The language also offers the logical AND `&&` and the OR `||`. The priorities of the operators are ordinary and the order can be specified by parentheses `(`, `)`. The summary of operators with their priorities:

Priority	Operator type	Operator
1	unary minus	`-`
2	multiplication, division	`* /`
3	enumeration, subtraction, concatenation	`+ - &`
4	comparison	`== != < > <= >=`
5	logical operators	`&& ||`
6	assignments	`= += -= *= /= &=`

4.5 Using of Variables

There are two kinds of variables in the TFL. The first kind includes the variables declared in the *vertex* and the *cluster* block. The second kind are auxiliary variables. The names have to match the regular expression `[a-zA-Z][a-zA-Z0-9_]*`. To make accessing to clusters and vertices easier the labels were set up:

`c`	current cluster
`a`	left child of the current cluster
`b`	right child of the current cluster
`child`	mark for `a` and `b` in **_child* blocks
`left`	left boundary vertex
`right`	right boundary vertex
`common`	common vertex of the children
`border`	the only boundary vertex of the point cluster

The usage of the names employs dot notation similarly to the object-oriented programming languages:

`variable`	global (declared in *var*) or auxiliary variable
`cluster.variable`	auxiliary variable or declared in *cluster*
`cluster.array[vertex]`	value attached to vertex in cluster array
`cluster.vertex.variable`	auxiliary or declared variable of the vertex
`vertex.variable`	abbreviation for `c.vertex.variable`

Auxiliary variables. In the root block (excepting the *var*), the auxiliary variables can be used. They do not need to be declared. The TFL learns the type from the first value that is assigned to the variable. Therefore each auxiliary variable must be initialized by some value. This is the reason for using one type of the infinity for the *integer* and another one for the *real*. The creation of auxiliary arrays is not allowed.

4.6 Example of Usage

The usage of the language is very easy as can be seen from the following example - the finding out the length of a path. The path is specified by two vertices and the technique was described by Alstrup et al. [1]. Each cluster holds its length. The algorithm needs to implement only the *join*. If a point cluster is created then its length is zero. The length of a path cluster is computed as the sum of the children lengths. The produced source code is very short and simple:

```
/****** The length of the path ******/

/*** Declaration of stored information ***/
algorithm {lengthOfWay}    // name of the algorithm
vertex { integer name; }   // we store name of the vertex in vertices
cluster { integer length; } // we store length of the cluster in clusters

/*** Description of the algorithm ***/
join {
  path_child { // if a or b is the path, then we remember its length
    child.l = child.length;
  } point_child { // if a or b is the point, then its length is zero
    child.l = 0;
  } path_parent { // path cluster length is the sum of its children
    c.length = a.l + b.l;
  } point_parent { // point cluster length is zero
    c.length = 0;
  }
}
```

5 Top Tree Query Language (TQL)

The TQL is a simple query language that was developed to control top trees. The language allows the adding of the vertices, joining and splitting of the edges or working with information stored in the top trees. There are supported the same data types as in the TFL.

To enable an easy manipulation with nodes of the forest one statement stored in the nodes must be unique. The user can choose that unique identifier which is preferred. This identifier cannot be the *boolean*, because it can determine only two values.

5.1 Adding of Nodes

When a new node is created then some values are saved into the statements declared in the *vertex* block of the TFL. To make it simpler there is a *node* command:

```
node (param1, param2, ...) (param3=value1, param4=value2, ...);
node;
```

So there can be declared the order of some parameters and default values of another parameters. The order of declarative and definition part is arbitrary and it is not necessary to use both of them. The default values have to match the data types and the unique identifier cannot be used. It is possible to call this command at any time. When the *node* is called without the parameters then it displays the current order and the default values settings.

A new node is created in the following manner:

```
unique_name (value1, value2, ...) (param1=value3, param2=value4, ...);
```

The enumeration part must agree with the declaration of the *node*. In the assignment part, the default values can be overwritten and other parameters defined. The parts can occur in any order. Any part can be omitted, but at least one has to be applied.

5.2 Joining and Splitting of Edges

Analogous to the nodes there is a command to specify the order of parameters and the settings of default values for edges that were declared in the *cluster* block of TFL:

```
edge (param1, ...) (param2=val1, param3[L]=val2, param3[R]=val3, ...);
edge;
```

The usage is the same as for the *node*, but moreover there can be used an array. To access the entries of the array there are marks `L` and `R` for the left and the right boundary vertex.

The command for the edge joining enables to specify the position of the new edge with regard to the position of other edges:

```
u {a} -- v {b} (value1, value2, ...) (param1=value3, ...);
```

When edges are organized in circular order around u and v, then the new edge (u, v) is the right successor of the (u, a) and the (v, b). This position specification can be omitted. For the enumeration and the assignment parts there are the same rules as for the nodes creation. A node can be created during the joining - all its parameters have to be set by default values and no position specification can be used.

The removing of the edges is very easy:

```
u ## v;
```

5.3 Reading Information and Another Commands

Sometimes it is important to read the information stored in the vertex v or in the cluster with boundary vertices u, v:

```
info(v);
info(u,v);
```

In the TQL language, there is a lot of embedded functions that were prepared for the most frequently used operations (e.g. value changing, . . .). In addition, the language enables to program custom functions as a plug-in in the Java language. The details can be seen in our previous work [6].

6 Final Remarks

We have developed a complex solution that allows using top trees for easy programming of tree algorithms. The solution is built on our implementation of top trees [7] and it is formed by Top Tree Friendly Language (TFL) and Top Tree Query Language (TQL).

The TFL is a declarative programming language that makes the structure available to users with only a basic knowledge of top trees. The declarativity saves users from technical details of the implementation and it allows to focus on the designing of tree algorithms. The TQL is a simple query language that was developed to control top trees. The language allows the adding of the vertices, joining and splitting of the edges or working with information stored in the top trees. In addition, the language TQL can be extended by custom functions.

Both languages form an useful tool that enables to write a simple and short source code quickly and then easily verify its functionality. So the tool makes the algorithm designing easier and more comfortable.

6.1 Related Work

To the best of our knowledge, there exists no similar tool to compare with. Top trees are relatively young data structure. Excepting the work of Alstrup et al. [1] and Tarjan and Werneck's works [5, 8], we did not find any other information about top trees.

References

1. S. Alstrup, J. Holm, K. D. Lichtenberg, and M. Thorup. Maintaining information in fully dynamic trees with top trees. *ACM Trans. Algorithms*, 1(2):243–264, 2005.
2. G. N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM Journal of Computing*, 14(4):781–798, 1985.
3. IEEE-Task-P754. *ANSI/IEEE 754-1985, Standard for Binary Floating-Point Arithmetic*. IEEE, New York, aug 12 1985. A preliminary draft was published in the January 1980 issue of IEEE Computer, together with several companion articles. Available from the IEEE Service Center, Piscataway, NJ, USA.
4. D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.
5. R. E. Tarjan and R. F. Werneck. Self-adjusting top trees. In *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 813–822, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.
6. M. Vajbar. Modelling dynamic trees. Master's thesis, Department of Software Engineering, Charles University in Prague, 2008.
7. M. Vajbar and K. Toman. Implementation of self-adjusting top trees. Technical Report No 2008/3, Charles University, Prague, Czech Republic, 2008.
8. R. F. Werneck. *Design and analysis of data structures for dynamic trees*. PhD thesis, Princeton University, Princeton, NJ, USA, 2006. Adviser-Robert E. Tarjan.

Dimension Reduction Methods for Iris Recognition

Pavel Moravec and Václav Snášel

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

Abstract. In this paper, we compare performance of several dimension reduction techniques, namely LSI, FastMap, and SDD in Iris recognition. We compare the quality of these methods from both the visual impact, and quality of generated "eigenirises".

Keywords: SVD, FastMap, information retrieval, SDD, iris recognition

1 Introduction

Methods of human identification using biometric features like fingerprint, hand geometry, face, voice and iris are widely studied.

A human eye iris has its unique structure given by pigmentation spots, furrows and other tiny features which are stable throughout life. It is possible to scan an iris without physical contact in spite of wearing contact lenses or eyeglasses. The iris is hard to forge which makes the iris a suitable object for the identification of people. Iris recognition seems to be more reliable than other biometric techniques like face recognition[3]. Iris biometrics systems for both private and public use have been designed and deployed commercially by NCR, Oki, IriScan, BT, US Sandia Labs, and others.

In this paper, we use the Petland's approach to image retrieval: image vectors of complete images of the size width $\times$ height of the image [13] build the feature vectors.

To fight the high dimension of image vector, we can extract several *features* which represent the image and concatenate them into a *feature vector*. The feature extraction methods can use different aspects of images as the features, typically the color features (histograms), shape features (moments, contours, templates), texture features and others (e.g. eigenvectors). Such methods are either using a heuristics based on the known properties of the image collection, or are fully automatic and may use the original image vectors as an input.

In this paper we will concentrate on the last category – other feature extraction methods which use known dimension reduction techniques and clustering for automatic feature extraction.

Singular value decomposition (*SVD*) was already successfully used for automatic feature extraction. In case of face collection (such as our test data), the *base vectors* can be interpreted as images, describing some common characteristics of several faces. These base vectors are often called eigenfaces. For a detailed description of eigenfaces, see [13].

However SVD is not suitable for huge collections and is computationally expensive, so other methods of dimension reduction were proposed. We test two of them – *Semi-Discrete decomposition*.

Recently, Toeplitz matrix minimal Eigenvalues are also playing a role towards image description and feature extraction [10]. This approach presents a method for the reduction of image feature points as it deals with the geometric relation between the points rather than their geometric position [11]. This can reduce the characteristic or feature points number from n to at least $\frac{n}{10}$ decreasing the computation level and hence the task time.

The rest of this paper is organized as follows. The second section explains used dimension reduction methods. In the third section, we briefly describe qualitative measures used for evaluation of our tests. In the next section, we supply results of tests for several methods on ORL face collection. In conclusions we give ideas for future research.

2 Dimension Reduction Methods

We used three methods of dimension reduction for our comparison – Singular Value Decomposition, Semi-Discrete Decomposition and FastMap, which are briefly described bellow.

2.1 Singular Value Decomposition

SVD [2] is an algebraic extension of classical vector model. It is similar to the *Principal components analysis* (*PCA*) method, which was originally used for the generation of eigenfaces. Informally, SVD discovers significant properties and represents the images as linear combinations of the base vectors. Moreover, the base vectors are ordered according to their significance for the reconstructed image, which allows us to consider only the first k base vectors as important (the remaining ones are interpreted as "noise" and discarded). Furthermore, SVD is often referred to as more successful in recall when compared to querying whole image vectors [2].

Formally, we decompose the matrix of images A by *singular value decomposition* (*SVD*), calculating singular values and singular vectors of A.

We have matrix A, which is an $n \times m$ rank-r matrix and values $\sigma_1, \ldots, \sigma_r$ are calculated from eigenvalues (λ_i) of matrix AA^T as $\sigma_i = \sqrt{\lambda_i}$. Based on them, we can calculate column-orthonormal matrices $U = (u_1, \ldots, u_r)$ and $V = (v_1, \ldots, v_r)$, where $U^T U = I_n$ a $V^T V = I_m$, and a diagonal matrix $\Sigma = diag(\sigma_1, \ldots, \sigma_r)$, where $\sigma_i > 0, \sigma_i \geq \sigma_{i+1}$.

The decomposition

$$A = U \Sigma V^T$$

is called *singular decomposition* of matrix A and the numbers $\sigma_1, \ldots, \sigma_r$ are *singular values* of the matrix A. Columns of U (or V) are called *left* (or *right*) singular vectors of matrix A.

Now we have a decomposition of the original matrix of images A. We get r nonzero singular numbers, where r is the rank of the original matrix A. Because the singular values usually fall quickly, we can take only k greatest singular values with the corresponding singular vector coordinates and create a *k-reduced singular decomposition* of A.

Let us have k $(0 < k < r)$ and singular value decomposition of A

$$A = U\Sigma V^T \approx A_k = (U_k U_0)\begin{pmatrix}\Sigma_k & 0\\ 0 & \Sigma_0\end{pmatrix}\begin{pmatrix}V_k^T\\ V_0^T\end{pmatrix}$$

We call $A_k = U_k \Sigma_k V_k^T$ a k-reduced singular value decomposition (rank-k SVD).

Instead of the A_k matrix, a matrix of image vectors in reduced space $D_k = \Sigma_k V_k^T$ is used in SVD as the representation of image collection. The image vectors (columns in D_k) are now represented as points in k-dimensional space (the *feature-space*). For an illustration of rank-k SVD see Figure 1.

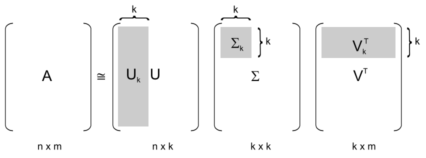


Fig. 1. rank-k SVD

Rank-k SVD is the best rank-k approximation of the original matrix A. This means that any other decomposition will increase the approximation error, calculated as a sum of squares (*Frobenius norm*) of error matrix $B = A - A_k$. However, it does not implicate that we could not obtain better precision and recall values with a different approximation.

To execute a query Q in the reduced space, we create a reduced query vector $q_k = U_k^T q$ (another approach is to use a matrix $D'_k = V_k^T$ instead of D_k, and $q'_k = \Sigma_k^{-1} U_k^T q$). Instead of A against q, the matrix D_k against q_k (or q'_k) is evaluated.

Once computed, SVD reflects only the decomposition of original matrix of images. If several hundreds of images have to be added to existing decomposition (*folding-in*), the decomposition may become inaccurate. Because the recalculation of SVD is expensive, so it is impossible to recalculate SVD every time images are inserted. The *SVD-Updating* [2] is a partial solution, but since the error slightly increases with inserted images. If the updates happen frequently, the recalculation of SVD may be needed soon or later.

2.2 SDD Method

Semidiscrete decomposition (*SDD*) is one of other LSI methods, proposed recently for text retrieval in [8]. As mentioned earlier, the rank-k SVD method (called *truncated SVD* by authors of semidiscrete decomposition) produces dense matrices U and V, so

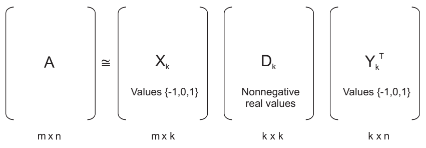


Fig. 2. "rank-k" SDD

the resulting required storage may be even larger than the one needed by the original term-by-document matrix A.

To improve the required storage size and query time, the semidiscrete decomposition was defined as

$$A \approx A_k = X_k D_k Y_k^T,$$

where each coordinate of X_k and Y_k is constrained to have entries from the set $\varphi = \{-1, 0, 1\}$, and the matrix D_k is a diagonal matrix with positive coordinates.

The SDD does not reproduce A exactly, even if $k = n$, but it uses very little storage with respect to the observed accuracy of the approximation. A rank-k SDD (although from mathematical standpoint it is a sum on rank-1 matrices) requires the storage of $k(m + n)$ values from the set $\{-1, 0, 1\}$ and k scalars. The scalars need to be only single precision because the algorithm is self-correcting. The SDD approximation is formed iteratively.

The optimal choice of the triplets (x_i, d_i, y_i) for given k can be determined using greedy algorithm, based on the residual $R_k = A - A_{k-1}$ (where A_0 is a zero matrix).

2.3 FastMap

FastMap [6] is a pivot-based technique of dimension reduction, suitable for Euclidean spaces.

In first step, it chooses two points (feature vectors) from the matrix A, which should be most distant for calculated reduced dimension. Because it would be expensive to calculate distances between all points, it uses following heuristics (all chosen points are image vectors from matrix A):

1. A random point c_0 is chosen.
2. The point b_i having maximal distance $\delta(c_i, b_i)$ from c_i is chosen, and based on it we select the point a_i with maximal distance $\delta(b_i, a_i)$
3. We iteratively repeat step 2 with $c_{i+1} = a_i$ (authors suggest 5 iterations).
4. Points $a = a_i$ and $b = b_i$ in the last iteration are pivots for the next reduction step.

In second step (having the two pivots a, b), we use the cosine law to calculate position of each point on line joining a and b. The coordinate x_i of point p_i is calculated as

$$x_i = \frac{\delta^2(a_i, p_i) + \delta^2(a_i, b_i) - \delta^2(b_i, p_i)}{2\delta(a_i, b_i)}$$

and the distance function for next reduction step is modified to

$$\delta'^2(p'_i, p'_j) = \delta^2(p_i, p_j) - (x_i - x_j)^2$$

The pivots in original and reduced space are recorded and when we need to process a query, it is projected using the second step of projection algorithm only. Once projected, we can again use the original distance function in reduced space.

3 Qualitative Measures of Retrieval Methods

Since we need an universal evaluation of any retrieval method, we use some measures to determine quality of such method. In case of Information Retrieval we usually use two such measures - *precision* and *recall*. Both are calculated from the number of objects relevant to the query Rel – determined by some other method, e.g. by manual annotation of given collection and the number of retrieved objects Ret. Based on these numbers we define precision (P) as a fraction of retrieved relevant objects in all retrieved objects and recall (R) as a fraction of retrieved relevant objects in all relevant objects. Formally:

$$P = \frac{|Rel \cap Ret|}{|Ret|} \text{ and } R = \frac{|Rel \cap Ret|}{|Rel|}$$

So we can say that recall and precision denote, respectively, completeness of retrieval and purity of retrieval. Unfortunately, it was observed that with the increase of recall, the precision usually decreases [12]. This means that when it is necessary to retrieve more relevant objects, a higher percentage of irrelevant objects will be probably obtained, too.

For the overall comparison of precision and recall across different methods on a given collection, we usually use the technique of rank lists [1], where we first sort the distances from smallest to greatest and then go down through the list and calculate maximal precision for recall closest to each of the 11 standard recall levels (0.0, 0.1, 0.2, . . . , 0.9, 1.0). If we are unable to calculate precision on i-th recall level, we take the maximal precision for the recalls between $i - 1$-th and $i + 1$-th level. From all levels, we calculate mean average, which is a single-value characteristics of overall precision-recall ratio.

4 Experimental Results

For testing of the different methods, we used iris collection consisting of 384 irises. The iris were scanned by TOPCON optical device connected to the CCD Sony camera. The acquired digitized image is RGB of size 576 $\times$ 768 pixels. Only the red (R) component of the RGB image has been used in our experiments because it appears to be more reliable than recognition based on green or blue components or converting the irises to grayscale first. It is in accord with [4], where near-infrared wavelengths are used anyway.

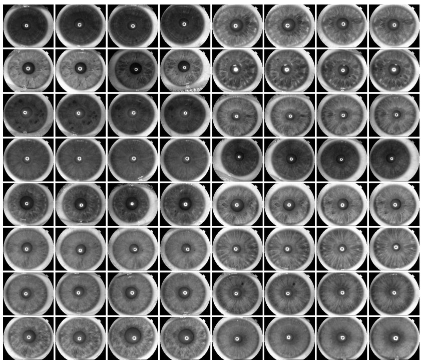


Fig. 3. Several irises from the collection

We have excluded one of the three irises for each eye for further querying (so that the query iris would not be included in the collection and skew the query results), which led to a collection of 256 irises of 64 people.

An example of several irises from the collection is shown in Figure 3, the first 12 query vectors are shown in Figure 8. We did not isolate the central part and eyelids to provide comparable results with[9].

4.1 Generated "Eigenirises" and Reconstructed Images

Many of tested methods were able to generate a set of base images, which could be considered to be "eigenirises" as is the case of PCA, SVD and several other methods. We are going to provide examples of both factors (base vectors) – "eigenirises" and reconstructed images which can be obtained from regenerated A_k. We calculated results for all methods in several dimensions, for the demonstration images we will use $k = 64$. We do not provide these images for FastMap, where it is not possible (we could have provided the images used as pivots in each step of FastMap process).

With SVD, we obtain factors with different generality, the most general being among the first. The first few are shown in figure 4. The eigenirises with higher index bring more details to reconstructed images.

The reconstructed images for rank-64 SVD method are somewhat blurred, but generally still recognizable, which can bee observed in figure 5.

The SDD method differs slightly from previous methods, since each factor contains only values $\{-1, 0, 1\}$. Gray in the factors shown in figure 6 represents 0; -1 and 1 are represented with black and white respectively.

The images in figure 7 are reconstructed least exactly from all methods (although consistently), but this is to be expected due to the three-valued encoding of base vectors. One may note a general loss of fine details, which is unfortunate, since it means that the query process would be highly affected and the retrieval results poor.

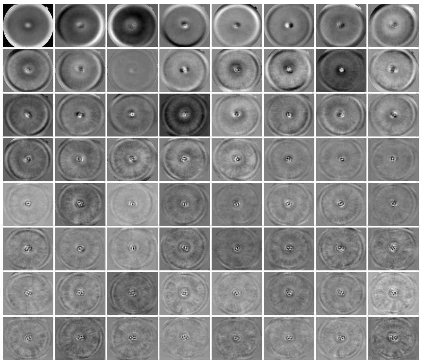


Fig. 4. First 64 eigenirises (out of possible 256) for SVD method

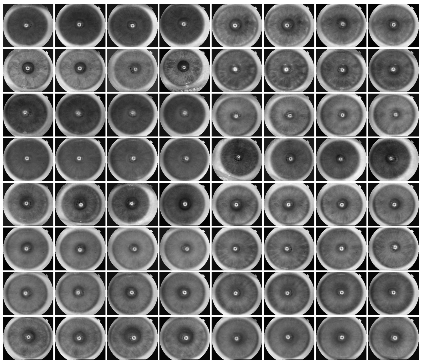


Fig. 5. Reconstructed images for SVD method

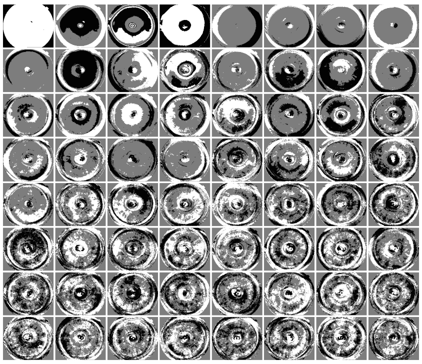


Fig. 6. First 18 base vectors (out of 100) for SDD method

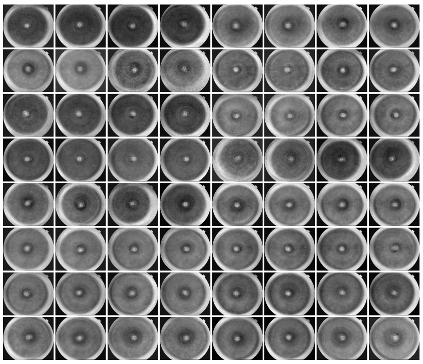


Fig. 7. Reconstructed images for SDD method

4.2 Query Evaluation

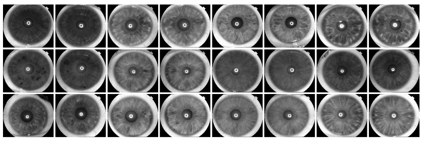


Fig. 8. Several irises used for querying

First, we calculated the mean average precision (MAP) for all relevant images in rank lists. The relative MAPs (against original matrix A – 100%) are shown in Table 1.

One would suspect, that querying in original dimension would provide better results that any of the dimension reduction methods. In Table 2 we show the number of queries, where the first returned iris was of the same person (out of 128).

5 Conclusion

In this paper, we have compared several dimension reduction methods on real-live image data (using L_2 metrics). Whilst the SVD is known to provide quality results, it is computationally expensive and in case we only need to beat the "curse of dimensionality" by reducing the dimension, FastMap may suffice.

There are some other newly-proposed methods, which may be interesting for future testing, e.g. the SparseMap [7]. Additionally, faster pivot selection technique for FastMap may be considered. We may also benefit from the use of Toeplitz matrices and their minimal eigenvalues relation.

Table 1. Mean average precision of iris comparison (VSM: 49%)

	Reduction method		
k	*FastMap*	**SVD**	SDD
4	25%	14%	3%
8	33%	31%	3%
16	39%	44%	4%
32	42%	46%	4%
64	45%	48%	5%
128	47%	49%	5%

Table 2. Number of queries, where the person was successfully identified (VSM: 83)

	Reduction method		
k	*FastMap*	**SVD**	SDD
4	32	37	2
8	47	56	3
16	54	75	3
32	60	81	5
64	64	82	4
128	96	84	5

What we currently need is a better iris segmentation, i.e. removing the central piece (in our case in light reflection), eyelids, and identifying the exact iris position, such as methods described in [5].

References

1. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison Wesley, New York, 1999.
2. M. Berry, S. Dumais, and T. Letsche. Computational Methods for Intelligent Information Access. In *Proceedings of the 1995 ACM/IEEE Supercomputing Conference*, San Diego, California, USA, 1995.
3. J. Daugman. Statistical richness of visual phase information: Update on recognizing persons by iris patterns. *International Journal of Computer Vision*, 45(1):25–38, 2001.
4. J. Daugman. The importance of being random: statistical principles of iris recognition. *Pattern Recognition*, 36(2):279–291, 2003.
5. J. Daugman. New methods in iris recognition. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 37(5):1167–1175, 2007.
6. C. Faloutsos and K. Lin. FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets. *ACM SIGMOD Record*, 24(2):163–174, 1995.
7. G. R. Hjaltason and H. Samet. Properties of embedding methods for similarity searching in metric spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5):530–549, 2003.
8. T. G. Kolda and D. P. O'Leary. Computation and uses of the semidiscrete matrix decomposition. In *ACM Transactions on Information Processing*, 2000.

9. P. Praks, L. Machala, and V. Snášel. Iris Recognition Using the SVD-Free Latent Semantic Indexing. In *MDM/KDD2004 – Fifth International Workshop on Multimedia Data Mining "Mining Integrated Media and Complex Data" in conjunction with KDD'2004 - The 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; Section 2. Multimedia Data Mining: Techniques and Applications*, pages 67–71, Seattle, WA, USA, 2004.
10. K. Saeed. *Image Analysis for Object Recognition.* Bialystok Technical University Press, Bialystok, Poland, 2004.
11. K. Saeed and M. K. Nammous. A speech-and-speaker identification system: Feature extraction, description, and classification of speech-signal image. *IEEE Trans. on Industrial Electronics*, 54(2):887–897, 2007.
12. G. Salton and G. McGill. *Introduction to Modern Information Retrieval.* McGraw-ill, New York, USA, 1983.
13. M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

Five-Level Multi-Application Schema Evolution*

Martin Nečaský, Irena Mlýnková

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

Abstract. Schema evolution has recently gained much interest in both research and practice. However, most of the existing works deal with separate aspects of the problem such as evolution of XML schemas or evolution of conceptual schemas. In addition, all of them view the problem only from the perspective of a single application.
In this paper we show that schema evolution has several different levels at which it can be performed and that are highly related. Secondly, we show that schema evolution is not the problem of a single application, but multiple applications having the same problem domain can influence each other as well. In particular we deal with five levels – extensional, operational, logical, platform-specific and platform-independent. We describe the particular levels, how they can be modified and the respective propagation of the modifications to other levels and applications. We also show which of the situations have already been discussed and solved in the existing works as well as which of them still remain open.

1 Introduction

Since XML [6] has become a de-facto standard for data representation and manipulation, there exists a huge amount of applications having their data represented in XML. Naturally, each of the applications can work only with correct data. We distinguish two levels of such correctness – well formedness and validity. An XML document is *well-formed* if it meets the well-formedness constraints stated in XML specification [6] such as, e.g., that the tags ensure the correct tree structure. An XML document is *valid* if it is associated with a legal XML schema (i.e. a schema that conforms to the specification of the selected language, such as, e.g., DTD [6], XML Schema [18, 3], etc.), and complies with the constraints expressed in it.

On the other hand, since most of the XML applications are usually dynamic, sooner or later the structure of the data needs to be changed. At the same time, we still need to be able to work with the old as well as new data without any loss. In relation to this topic, we usually speak about so-called *schema evolution*, i.e. a situation that a schema of the data is updated and we need to apply

* This work was supported in part by the Czech Science Foundation (GAČR), grants number 201/09/P364 and 201/09/0990.

these updates on its existing instances to revalidate them. In case of XML data, having a set of *XML schema transformations*, we can transform an XML schema S^x to an XML schema S^y. A natural requirement for such transformation is *consistency*, i.e. transforming a legal schema S^x to a legal XML schema S^y. The problem of XML schema evolution deals with applying a set of respective *XML data transformations* on XML documents valid against S^x so that they become valid against S^y.

Currently there exist several works dealing with this topic. However, most of the existing works deal with separate aspects of the problem such as evolution of XML schemas or evolution of conceptual schemas. In addition, all of them view the problem only from the perspective of a single application. Hence, in this paper we study the problem from a more general perspective. We show that schema evolution has several different levels at which it can be performed and that are highly related. Secondly, we show that schema evolution is not the problem of a single application, but multiple applications having the same problem domain can influence each other as well. In particular we deal with five levels – extensional, operational, logical, platform-specific and platform-independent. We describe the particular levels, how they can be modified and the respective propagation of the modifications to other levels and applications. We also show which of the situations have already been discussed and solved in the existing works as well as which of them still remain open.

The paper is structured as follows: Section 2 introduces the given problem using a motivating example. Section 3 overviews existing related works. Section 4 describes the proposed evolution model in detail and Section 5 provides an overview of respective schema transformations. Section 6 overviews some real examples. And, finally, Section 7 provides conclusions and outlines possible future work.

2 Motivation

To depict the issue we are focusing on, we first provide a real-world example. We consider a company that produces products and sells them to customers. An information system of the company is composed of various applications utilized by different users for different purposes. Each application applies different XML formats that represent different user views on the data. On the other hand, since the applications work with the same data, e.g. customers, products, etc., the XML formats represent the same problem domain. If a change in one XML format results into a change in our interpretation of the domain, it can require to make corresponding changes in other XML formats as well. We demonstrate some examples in the rest of this section.

We consider two particular XML formats each described by an XML schema expressed in the XML Schema language. A first XML schema `PurchaseRequest.xsd` describes an XML format used by customers to send purchase requests to the company. A second XML schema `SalesReport.xsd` describes a format representing reports on sales of products in regions. The XML schemas are depicted in Figure 1.

```
<xs:schema xmlns:xs=”w3.org/2001/XMLSchema”>
 <xs:element name=”purchase-request” type=”Purchase”/>
 <xs:complexType name=”Purchase”>
  <xs:sequence>
   <xs:choice>
    <xs:element name=”messenger” type=”Messenger”
                minOccurs="0"/>
    <xs:element name=”van” type=”Van” minOccurs="0"/>
   </xs:choice>
   <xs:element name=”item” type=”Item”
               maxOccurs=”unbounded” />
  </xs:sequence>
  <xs:attribute name=”issue-date” type=”xs:string”/>
  <xs:attribute name=”registration” type=”xs:string”/>
 </xs:complexType>
 <xs:complexType name=”Messenger”>
  <xs:attribute name=”code” type=”xs:string”/>
 </xs:compleType>
 <xs:complexType name=”Van”>
  <xs:attribute name=”plate-no” type=”xs:string”/>
 </xs:compleType>
 <xs:complexType name=”Item”>
  <xs:sequence>
   <xs:element name=”amount” type=”xs:string”/>
   <xs:element name=”price” type=”xs:string”/>
  </xs:sequence>
  <xs:attribute name=”code” type=”xs:string”/>
 </xs:compleType>
</xs:schema>
```

```
<xs:schema xmlns:xs=”w3.org/2001/XMLSchema”>
 <xs:element name=”sales-report” type=”Product”/>
 <xs:complexType name=”Product”>
  <xs:sequence>
   <xs:element name=”name” type=”xs:string”/>
   <xs:element name=”region” type=”Region”
               minOccurs="0"
               maxOccurs=”unbounded”/>
  </xs:sequence>
  <xs:attribute name=”code” type=”xs:string”/>
 </xs:complexType>
 <xs:complexType name=”Region”>
  <xs:sequence>
   <xs:element name=”customer” type=”Customer”
               minOccurs="0"
               maxOccurs=”unbounded”/>
  </xs:sequence>
  <xs:attribute name=”code” type=”xs:string”/>
 </xs:complexType>
 <xs:complexType name=”Customer”>
  <xs:sequence>
   <xs:element name=”name” type=”xs:string”/>
   <xs:element name=”email” type=”xs:string”/>
  </xs:sequence>
  <xs:attribute name=”registration”
                type=”xs:string”/>
 </xs:complexType>
</xs:schema>
```

Fig. 1. XML schemas for purchase requests and sales reports

In our interpretation of the domain, there are customers each having a registration number and name. Both XML schemas somehow represent customers. While `PurchaseRequest.xsd` represents only their registration numbers, `SalesReport.xsd` represents registration numbers as well as names.

We need customers to specify their names in purchase requests. Since `PurchaseRequest.xsd` does not consider names we have to modify it. We decide to add XML element `name` to root XML element `purchase-request`. This transformation does not change our interpretation of the domain since we have already considered names of customers. It does not therefore influence `SalesReport.xsd`.

Further, sales managers want to structure names of customers in sales reports to first and family names. `SalesReport.xsd` has only XML element `name` in XML element `customer`. Therefore, we replace `name` with `first-name` and `family-name`. This transformation however changes our interpretation of the problem domain since we have considered a name as one unstructured value. This can therefore influence other XML schemas as well. In a concrete, we need to structure XML element `name` in `PurchaseRequest.xsd` in a similar way.

As the examples demonstrate, a transformation of an XML schema can influence other XML schemas in the system as well. This is because we need to keep the XML schemas consistent with our interpretation of the problem domain. However, managing the consistency at the XML schema level is an error–prone and time–consuming task since it must be handled manually. Therefore, a more sophisticated method for XML schema evolution would be useful in practice.

3 Related Work

We can divide current approaches to XML schema evolution into three groups depending on the level where transformations can be specified by the designer. These groups are depicted in Figure 2 in columns (a), (b) and (c) respectively.

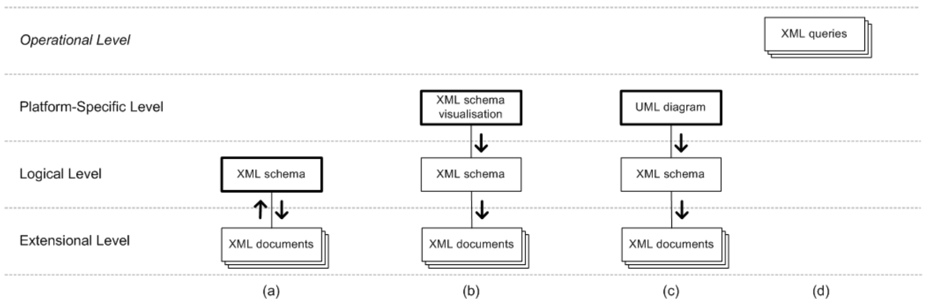


Fig. 2. Current XML Evolution Approaches

Approaches in the first group consider transformations at so-called *logical*, i.e. XML schema level. They differ mainly in the selected XML schema language, i.e. DTD [16, 2, 9] or XML Schema [17, 12]. In general, the transformations can be variously classified. For instance, paper [17] proposes three classes: *Migratory* transformations deal with movements of elements/attributes to other parts of the schema or transformation of elements to attributes and vice versa. *Structural* transformations involve adding/removal of elements/attributes. Finally, *sedentary* transformations involve renaming of elements/attributes and modifications of simple data types. In all the proposed systems the transformations are then automatically propagated to the respective XML documents, i.e. to so-called *extensional* level, to ensure that the XML documents are valid against the evolved XML schema. There also exists an opposite approach that enables to evolve XML documents and propagate the transformations to their XML schema [5].

Approaches in the second and third group are similar to the first one but they do not consider transformations specified at the logical level but its abstraction. In particular, approaches in the second group consider a visualization of the XML schema [11], whereas approaches in the third group consider a UML class diagram that models the XML schema [10]. From our point of view, we comprehend both the cases as so-called *platform-specific* model, since it directly models the components of the XML schema but not the data abstracted from its representation in XML. Transformations made at platform-specific level are then propagated to corresponding transformations in the XML schema and from here to XML documents. The main advantage is that it is easier for the designer to specify transformations at the abstraction level than at the XML schema level because (s)he can concentrate more on the transformation itself rather than technical XML schema details.

Another open problem related to schema evolution is adaptation the respective operations, in particular XML queries. We speak about so-called *operational* level (see Figure 2 by column (d)). Unfortunately, there seems to exist only a single paper [14] dealing with this topic. The authors provide several use cases when the queries need to be corrected with regard to the modification of the data and, finally, state several recommendations how to write queries that do not need to be adapted for evolving schema.

In general, the problem of current approaches is that they do not consider several important issues. Firstly, they consider evolution of a single XML format. However, as we demonstrated in Section 2, there are usually multiple XML formats providing different views of the domain. Hence, a transformation of an XML format can influence other XML formats as well. Secondly, queries over XML documents are ignored. Since queries depend on the structure of the XML documents, transformation of an XML schema can influence them and they should therefore be considered as a part of the evolution system as well.

4 Five Level Schema Evolution

In this paper, we introduce a new approach to XML evolution. While the current approaches consider evolution on two, resp. three levels as depicted in Figure 2, we consider five levels. The introduced levels are depicted in Figure 3.

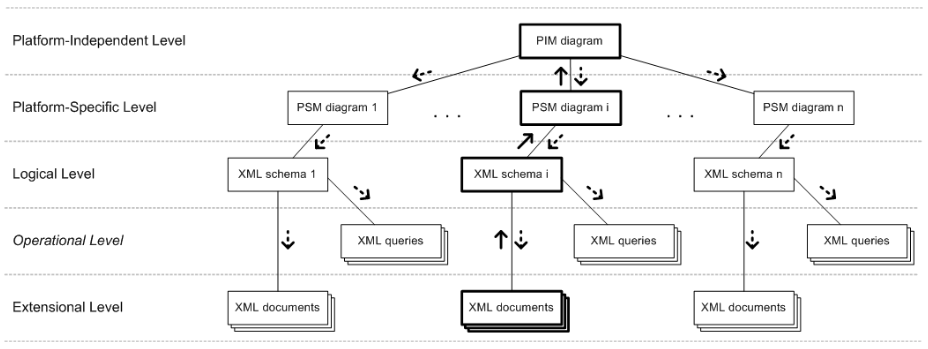


Fig. 3. Five level XML evolution architecture

The *extensional level* contains for each XML format a set of XML documents conforming to this format. The *logical level* contains an XML schema that describes structure of the XML format. The *operational level* contains queries related to the XML format. If we consider only these levels, it is hard to determine whether a transformation of an XML format influences other XML formats since it must be determined manually. The problem is that we have no layer that interrelates the XML formats. Therefore, we add two other levels, i.e. platform–specific and platform–independent level, that provide such interrelation.

We adopted the terms *platform–independent* and *platform–specific* from the Model–Driven Architecture (MDA) terminology which considers modeling data at different levels of abstraction. Even though MDA considers more levels, we adopt only the two mentioned ones. The platform–independent level contains a conceptual diagram of the problem domain. It provides a description of the problem domain abstracted from the logical level. The platform–specific level interrelates the platform–independent and logical level. It provides a mapping of each XML diagram to the conceptual diagram. This mapping enables automatic transformation propagation since a transformation of an XML schema can be propagated to the conceptual diagram and from here to other XML schemas.

Figure 3 depicts in bold rectangles that a transformation can occur at any level L except the operational level. From L, the transformation is firstly propagated to the upper levels. This propagation is therefore called *upward propagation* and is depicted in Figure 3 by solid arrows. After the upward propagation, the transformation is propagated back to the lower levels as depicted in Figure 3 by dashed arrows. This propagation is therefore called *downward propagation.*

The upward propagation means that a transformation specified at L implies a corresponding transformation at the upper level $L - 1$. This propagation does not occur in every case. There are transformations whose impact depends on a designer. There are also transformations that have no impact at all. The upward propagation can end up at any level from where the downward propagation continues. If the upward propagation ends up at the platform–specific or lower level, the downward propagation continues only in the scope of the corresponding XML format. If the propagation ends up at the platform–independent level, our interpretation of the problem domain has changed and a downward propagation to other XML formats can be therefore required.

We describe possible transformations and their propagation in detail in Section 5. Before this, we describe the separate levels in the rest of this section.

Platform–Independent Level The platform–independent level contains a conceptual diagram of the problem domain. It describes the domain independently of the considered XML formats. To design such a diagram we use a conceptual modeling language which is called *Platform–Independent Model (PIM)* in the MDA terminology. The diagram is then called *PIM diagram.* As PIM, we consider the well-known UML class model. We consider only basic modeling constructs, i.e. classes for modeling concepts and binary associations for modeling relationships between the concepts.

Example 1. Figure 4 shows a sample PIM diagram modeling the domain of the company introduced in Section 2. We designed the diagram in a tool called XCase [1] that we have developed for conceptual modeling of XML schemas. For example, there is a class *Customer* modeling customers. It has attributes *registration*, *name*, *email* and *phone* modeling relevant customer characteristics. A sample association is the one connecting *Customer* and *Purchase.* It models that customers make purchases.

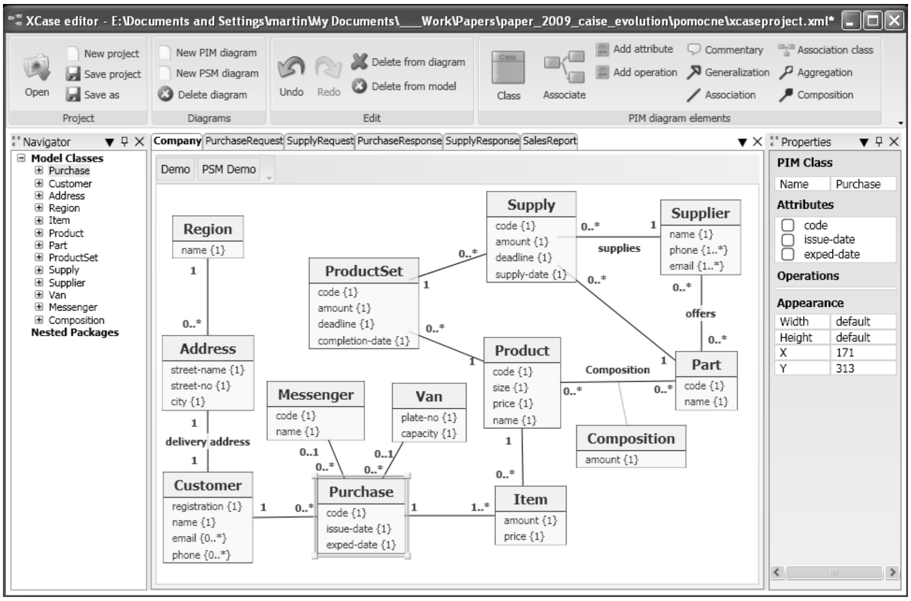


Fig. 4. Company PIM diagram

Platform–Specific Level The platform–specific level contains for each XML format a diagram that models the XML format in terms of the PIM diagram. In other words, it serves as a mapping between the corresponding XML schema and the PIM diagram. For this, we use a modeling language which is called *Platform–Specific Model (PSM)* in the MDA terminology. The resulting diagram is called *PSM diagram*. As PSM, we consider the UML class model extended with some constructs covering XML–specific features. In this paper, we introduce only some of the constructs. For their full description, we refer to [15].

A PSM diagram contains classes from the PIM diagram and organizes them into the hierarchical structure of the modeled XML format by associations. There is a formal background that maps associations in PSM diagrams to associations in a PIM diagram. However, we omit it in this paper. A label displayed above a class is called *element label* and specifies a name of an XML element that represents instances of the class in the XML format. Attributes of a class model XML attributes. They can be separated to so called *attribute container* displayed by a box beneath the class. In that case they model XML elements. We can also model variants in the content of a PSM class by so called *content choice*. It is displayed by a circle with an inner | and models that only one of its components can be instanciated for each its instance.

Example 2. Figure 5 shows two sample PSM diagrams modeling the XML formats for purchase requests and sales reports from Section 2. Both model the respective XML format in terms of the PIM diagram depicted in Figure 4. The PSM diagrams were designed in the XCase tool [1].

To explain the PSM constructs, consider the diagram on the left. It models that purchases are represented in the corresponding XML format as roots since *Purchase* is a root class. Because the root PSM class *Purchase* has an element label *purchase-request*, a purchase is represented as an XML element `purchase-request`. The associations going from *Purchase* model that a purchase has a customer, delivery information and list of items as children whose representation is modeled by the children of *Purchase*. The diagram utilizes an attribute container to specify that attributes *amount* and *price* of *Item* model XML elements. It also utilizes a content choice to specify that each purchase contains a messenger or van, that delivers the purchase, but not both.

The PSM diagram provides a mapping of `PurchaseRequest.xsd` to the PIM diagram. Each PSM class maps a complex type in the XML schema to a corresponding PIM class. The same is for associations and attributes.

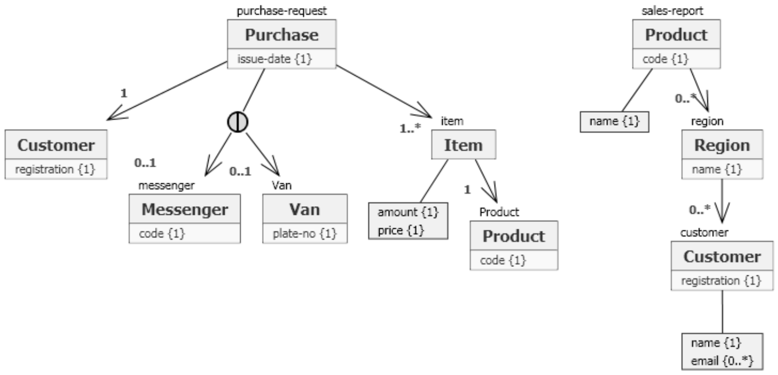


Fig. 5. Purchase and Sales Report PSM diagrams

Logical Level The logical level contains for each considered XML format its XML schema. An XML schema describes a syntactical structure, i.e. what XML elements and attributes can be used in respective XML documents. Even there are various languages for expressing XML schemas, we consider the XML Schema language [18, 3] in this paper. Self-descriptive examples of XSDs are depicted in Figure 1 in Section 2 .

XML Schema provides several constructs. We consider only the basic ones in this paper. Namely, simple data types (`simpleType`), complex data types (`complexType`), elements (`element`) and attributes (`attribute`). Since we are restricted in space, we do not describe the constructions in a more detail and refer to [18] for their explanation.

Extensional Level The extensional level contains for each considered XML format a set of XML documents. XML documents are composed of XML elements that can contain text values and/or nested XML elements. An XML element can also have XML attributes. Figure 6 shows two self-descriptive XML documents valid against the XML schemas depicted in Figure 1.

```
<purchase-request issue-date="30/11/2009"
                  registration="C828111">
 <messenger code="M190" />
 <item code="P2345">
  <amount>4</amount>
  <price>92</price>
 </item>
 <item code="P9982">
  <amount>1</amount>
  <price>10299</price>
 </item>
</purchase-request>
```

```
<sales-report code="P2345">
 <name>Product 2345</name>
 <region name="cz">
  <customer registration="C828111">
   <name>Martin Necasky</name>
   <email>martin@somewhere.cz</email>
  </customer>
  <customer registration="C802112">
   <name>Irena Mlynkova</name>
   <email>irena@somewhere.cz</email>
  </customer>
 </region>
 <!-- other regions -->
</sales-report>
```

Fig. 6. Sample XML documents conforming to XML formats for purchase requests and sales reports

Operational Level The operational level contains for each XML format queries that are evaluated on the XML documents of that format. There are several XML query languages such as, e.g. XSLT [7] or XQuery [4]. They all apply XPath [8] as a common factor to navigate in the structure of XML documents. Therefore, XPath expressions must be considered during evolution since a change in the structure of an XML format can have an impact on them. We do not describe XPath in detail here since we are restricted in space. A sample XPath expression is a path `/sales-report//customer/name` that targets names of customers in XML documents with sales reports.

5 Transformations and their Propagation

Having the above described levels, we can now specify the set of transformations a user may make at each of them. Similarly to the existing works [12] we can distinguish atomic transformations and high-level transformations (i.e. sequences of atomic transformations) as well as sedentary, structural and migratory ones. However, since this classification is too general, we further adopt general types of transformations similar to [14]:

- Structural:
  - *Adding* – adds a new item
  - *Removal* – removes a new item
- Sedentary:
  - *Extension* – adds a new item that does not change structure
  - *Renaming* – renames an item
  - *Renumbering* – changes the cardinality of an item

  - *Retyping* – changes the data type of an item
  - *Resetting* – changes the value of an item
  - *Mapping* – maps an item to an item from another level
  - *Unmapping* – removes a mapping between levels
- Migratory:
  - *Moving* – moves an item
  - *Reordering* – changes the order of a set of items
  - *Transformation* – transforms an item to an item of a different type

Note that not all types of the transformations exist at all five levels we are dealing with. For instance, retyping does not occur at the platform–indepdendent level, since we restrict ourselves only to classes, attributes and associations.

As we have described, propagation of transformations can be either upwards or downwards. Hence, the transformations at particular levels need to be propagated to all neighboring levels. With our general view of the data, we do not need to propagate the modifications to all the other levels if not necessary. In general the propagation needs to be done if the XML data become invalid. However, in all the other cases it remains user's decision if the respective propagations should be done. In the following text we describe only the necessary propagations, however also user-required propagations are discussed in important cases. For simplicity, we will deal only with atomic operations and thus omit all migratory ones that are high-level. For the same reason we omit trivial transformations, such as, e.g., renaming. (The full description can be found in [13].)

5.1 Platform–Independent Level

As we can see in Table 1, most of the operations at the platform–independent level are of the type of adding or removal that can be applied on all PIM items. Furthermore we can apply renumbering on the cardinalities. The operations can be propagated only to the platform–specific level within the downwards propagation.

Class	**Operations**	**PSM Propagation**
Adding	add class, attribute or association	
Removal	remove class, attribute or association	delete mapping
Renumbering	change class cardinality	

Table 1. PIM transformations and their propagation

Adding a new PIM item does not need to automatically trigger propagation to the platform–specific level. However, if such propagation is required, a user must state the required mapping to the respective PSM items since there usually exist multiple options. On the other hand, deleting of a PIM item invokes deleting of the respective mapping. However, a user may require also optional deleting of the respective item as well. Similarly, changing cardinality may invoke changing cardinality of the respective PSM item depending on user's requirements.

5.2 Platform–Specific Level

As stated in Table 2 at the platform–specific level there are several sedentary transformations, in particular changing cardinalities or data types and adding/deleting mapping to a PIM item. The rest of the transformations are structural since they create or delete respective PSM items.

Class	Operations	Logical Propagation	PIM Propagation
Mapping	map a PSM item to a PIM item		create a PIM item (O)
Unmapping	delete mapping		
Adding	add class, attribute, association, content choice or content container	add simple type, complex type, element, attribute, choice operator or sequence operator	
Removal	remove class, attribute, association, content choice or content container	remove simple type, complex type, element, attribute, choice operator or sequence operator	
Renumbering	change class cardinality	change element cardinality	
Retyping	change data type	change simple type	

Table 2. PSM transformations and their propagation

Except for the mapping transformations, all the others need to be propagated to the logical level. (Note that for the sake of simplicity we do not deal with XML schema specific aspects such as possibilities of creating a global or local element or equivalent content model. The detailed description of propagation of operations with particular schema items can be found in [13].) On the other hand, except for mapping a PSM item to a PIM item which requires creating of the respective PIM item if it does not exist yet, no other transformations need to be propagated from the platform–specific to platform–independent level if not otherwise specified by a user.

5.3 Logical Level

Table 3 provides an overview of logical-level transformations. Similarly to the previous case, for the sake of simplicity we omit all the possible specifications of an equivalent XML schema construct. To ensure consistency of the transformations we also state the following invariant that needs to hold during the whole evolution process: A globally defined item can be deleted only if all the respective references have already been deleted.

As we can see, most of the transformations belong to structural class since they add or delete XML schema items; the only sedentary transformations are changing a simple type or a cardinality and adding simple and complex types.

Class	**Operations**	**Extensional Propagation**	**PSM Propagation**
Adding	add element (+ simple or complex type), attribute (+ simple type), choice operator or sequence operator	add element or attribute (O)	add a class, attribute, association, content choice or content container (O)
Extension	add simple type or complex type		
Removal	remove simple type, complex type, element, attribute, choice operator or sequence operator	remove element or attribute (O)	delete class, attribute or content choice (O)
Renumbering	change element cardinality	add or remove element (O)	change class cardinality
Retyping	change simple type	change data value (O)	change data type

Table 3. Logical transformations and their propagation

As for the propagation to extensional level, all the transformations need to be propagated only in case the validity of XML documents is violated, i.e. all the propagations are optional (O). For instance, adding an XML schema element may cause adding an XML element only in case the XML schema element is compulsory. On the other hand, in case of logical-to-PSM propagation, we always need to propagate changes in data types and cardinalities. However, if we add or remove an XML schema item, the respective propagation depends on its type and can cause adding/removing of respective item(s) or nothing. For instance, if we create a complex type, the operation has no influence at platform–specific level, however if we create an element with a complex type, we need to create the respective PSM class. And, finally, note that creating a simple or complex type that is not associated with respective element or attribute has no influence in both directions of propagation.

5.4 Extensional Level

As we can see in Table 4 transformations over XML documents involve structural transformations of adding/removing an element/attribute and a sedentary transformation of changing a data value.

Depending on the XML schema, propagation of all the transformations is optional an depends on violation of validity. In addition, some of the transformations have several options how they can be propagated. For instance, creating

Class	**Operations**	**Logical Propagation**
Adding	add element or attribute	create element or attribute, change cardinality (O)
Removal	remove element or attribute	remove element or attribute, change cardinality (O)
Value Change	change data value of element or attribute	change simple data type (O)

Table 4. Extensional transformations and their propagation

an element may cause creating an element in the XML schema, changing cardinality or even nothing.

5.5 Operational Level

Last but not least, let us briefly discuss the operational level. In particular, considering only the XML data, we may restrict the transformation set to XML queries. The question is how can be this level influenced by the other levels and, conversely, whether the operational level influences other ones.

The answer for the latter question is much easier since we can hardly assume that if a user modifies a query, such modification should anyhow influence the data. However, any of the changes at the logical level can influence the operational level, since the queries may become irrelevant with regard to the data. As discussed in [14], such situation occurs in case the queries do not follow the rules the authors provide. For instance, if a query involves a simple XPath [8] path and any of the elements on the path is deleted, also the path needs to be respectively updated.

6 Propagation Examples

In this section we demonstrate transformation propagation on the transformations introduced in Section 2.

The first transformation was an addition of new child XML element `name` to `purchase-request` in the XML format for purchase requests. We initiate the transformation at the extensional level by modifying an XML document. Since we have added a new element to an XML document, the transformation must be propagated to the logical level and the corresponding XML schema (depicted in Figure 1 on the left) must be extended with the corresponding element declaration. It must be further propagated to the corresponding PSM diagram (depicted in Figure 5 on the left) where we add a new attribute *name* to the class *Customer*. Finally, the transformation is propagated to the PIM diagram. Since customer names are already modeled by attribute *name* of *Customer* we map the *name* PSM attribute to the exisitng *name* PIM attribute and no transformation is needed. Therefore, our interpretation of the domain has not been changed by the initial requirement and no downward propagation is required.

The second transformation was a replacement of XML element `name` with new elements `first-name` and `family-name` in the XML format for sales reports. We initiate this transformation in `SalesReport.xsd`, i.e. at the logical level. Again, the transformation is propagated upwardly first. In the corresponding PSM diagram (depicted in Figure 5 on the right), it means to replace the attribute *name* of *Customer* with new attributes *first-name* and *family-name*. Since these attributes have no equivalents in the PIM diagram, our interpretation of the domain has changed and the PIM diagram must be correspondinly transformed as well, i.e. the attribute *name* of *Customer* must be replaced with new attributes *first-name* and *family-name*. A downward propagation must follow. It means to propagate the transformation to the XML documents with sales reports and, moreover, to the other XML formats, i.e. to the XML format for purchase requests. Since there can be queries for each XML format querying names of customers, they can be influenced as well.

7 Conclusion

The aim of this paper was to show that the problem of XML schema evolution has been highly marginalized so far. In particular, all the existing works only deal with subparts of the problem. Firstly, we have showed that the schema evolution problem must be viewed from the perspective of multiple applications and we have defined five levels of XML schema evolution that cover all the existing works. Secondly, we have discussed how the respective transformations at particular levels influence the neighboring ones. And, finally, we have described use cases related to the problem that cannot occur unless we consider all the evolution levels.

Currently, we are dealing with a throughout implementation of the proposed system. For this purpose we want to extend system *XCase* [1] which enables to design conceptual diagrams and map them to respective XML schemas.

Apparently, there exist several open issues: Similarly to the existing works we need to make the adaptation at all levels (and especially the newly proposed ones) efficient and, consequently, we need to be able to find the least expensive sequence of transformations. Secondly, there remains the question of performing the adaptations. Instead of using a brute-force, we want to output a set of XSLT scripts that can be applied on the respective XML data. Naturally, this approach requires the existence of an XML representation of the non-XML levels. For the sake of full generality, we intend to involve all the relevant XSD constructs as well as other high-level transformations that we have omitted for simplicity. And, last but not least, we want to extend the approach with concurrency control. Having such a robust system, it is quite natural that there are multiple users to work with it, i.e. multi-user access and transactions need to be incorporated.

References

1. *XCase – A Tool for XML Data Modeling.* 2008. `http://kocour.ms.mff.cuni.cz/~necasky/xcase/`.

2. L. Al-Jadir and F. El-Moukaddem. Once Upon a Time a DTD Evolved into Another DTD... In *Object-Oriented Information Systems*, pages 3–17, Berlin, Heidelberg, 2003. Springer.
3. P. V. Biron and A. Malhotra. *XML Schema Part 2: Datatypes (Second Edition).* W3C, October 2004. `http://www.w3.org/TR/xmlschema-2/`.
4. S. Boag, D. Chamberlin, M. F. Fernndez, D. Florescu, J. Robie, and J. Simeon. *XQuery 1.0: An XML Query Language.* W3C, January 2007. `http://www.w3.org/TR/xquery/`.
5. B. Bouchou, D. Duarte, M. Halfeld Ferrari Alves, D. Laurent, and M. A. Musicante. Schema Evolution for XML: A Consistency-Preserving Approach. In *Mathematical Foundations of Computer Science*, pages 876–888, Prague, Czech Republic, 2004. Springer-Verlag.
6. T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. *Extensible Markup Language (XML) 1.0 (Fourth Edition).* W3C, September 2006. `http://www.w3.org/TR/REC-xml/`.
7. J. Clark. *XSL Transformations (XSLT) Version 1.0.* W3C, November 1999. `http://www.w3.org/TR/xslt`.
8. J. Clark and S. DeRose. *XML Path Language (XPath) Version 1.0.* W3C, November 1999. `http://www.w3.org/TR/xpath`.
9. S. V. Coox. Axiomatization of the evolution of xml database schema. *Program. Comput. Softw.*, 29(3):140–146, 2003.
10. E. Dominguez, J. Lloret, A. L. Rubio, and M. A. Zapata. Evolving XML Schemas and Documents Using UML Class Diagrams. In *DEXA '05: Proceedings of the 16th International Conference on Database and Expert Systems Applications*, pages 343–352, Berlin, Heidelberg, 2005. Springer.
11. M. Klettke. Conceptual XML Schema Evolution - the CoDEX Approach for Design and Redesign. In *BTW Workshops*, pages 53–63. Verlagshaus Mainz, Aachen, 2007.
12. M. Mesiti, R. Celle, M. A. Sorrenti, and G. Guerrini. X-Evolution: A System for XML Schema Evolution and Document Adaptation. In *EDBT '06: Proceedings of the 10th International Conference on Extending Database Technology*, pages 1143–1146, Berlin, Heidelberg, 2006. Springer.
13. I. Mlynkova and M. Necasky. Five-Level Multi-Application Schema Evolution. Technical Report 2008/7, Department of Software Enginneedring, Charles University, Prague, Czech Republic, 2008.
14. M. M. Moro, S. Malaika, and L. Lim. Preserving XML Queries During Schema Evolution. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 1341–1342, New York, NY, USA, 2007. ACM.
15. M. Necasky. *Conceptual Modeling for XML.* PhD thesis, Charles University, 2008. `http://kocour.ms.mff.cuni.cz/~necasky/dw/thesis.pdf`.
16. H. Su, D. K. Kramer, and E. A. Rundensteiner. XEM: XML Evolution Management. Technical Report WPI-CS-TR-02-09, Computer Science Department, Worcester Polytechnnic Institute, Worcester, Massachusetts, 2002.
17. M. Tan and A. Goh. Keeping Pace with Evolving XML-Based Specifications. In *EDBT '04 Workshops*, pages 280–288, Berlin, Heidelberg, 2005. Springer.
18. 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/`.

Tequila - a query language for the Semantic Web

Jakub Galgonek

Department of software engineering, Faculty of Mathematics and Physics,
Charles University in Prague,
Malostranské nám. 25, 118 00, Prague 1, Czech Republic
galgonek@ksi.mff.cuni.cz

Abstract. In order to realize the idea of the Semantic Web, many tools and technologies need to be developed, including a query language. One of the most important technologies that creates a base for the Semantic Web is the RDF (Resource Description Framework). There are many query languages for the RDF, but almost all of them are very weak and they can not be used for a general purpose. In this paper, we introduce the Tequila query language, developed for the recently introduced Trisolda infrastructure for the Semantic Web. In particular, we describe the syntax and semantics of the Tequila language and we show usage of this language and its power on several examples. The examples show how to convert a RDF Collection into a RDF Bag or how to evaluate an aggregation function (e.g., maximum of values) without the need of a built-in support. Unlike almost all other query languages, Tequila is powerful enough to express both of the tasks.

Keywords: Semantic Web, RDF, query language, Trisolda

1 Introduction

The Semantic Web is an idea of storing data together with their meaning. In order to its realisation, many tools and technologies need to be developed. It includes a language for storing data, vocabularies for describing an ontology and others. In order to have the ability to query on specific data, it also includes a query language as one of the important parts.

The Resource Description Framework (RDF) [3] is widely used as a language for storing data and it creates a base of the Semantic Web. The simple RDFS vocabulary or the more complicate OWL vocabulary are used for describing an ontology. There are also many query languages for RDF [6][8], such as the SPARQL language [4] or the SeRQL [2] language. However, almost all of them are very weak and they are not well applicable for a general purpose.

1.1 Troubles of the current query languages

Troubles of the current query languages for RDF can be divided into practical troubles and philosophical troubles. Most of them are related to each other.

The practical troubles include the poor ability to resolve a general task. The languages are either too specialised or they have just weak constructs. For example, many of the query languages are not able to select a RDF Collection from a data source, although the structure of a RDF Collection is directly defined in the RDF specification. Generally, the current query languages have troubles with a selection of a recursively defined structure.

Yes, some languages introduce powerful constructs such as regular paths (the ARQ language [1]) or deductive rules (the TRIPLE languages [9]). Although these constructs give very good power for selecting data, they are still too weak for creating data. The ability to create data is useful not only for storing new data to a data source but also for composing queries, where a result of one query is used as an input source for other query.

The weak ability to create data also includes the bad support for blank nodes. For example, the SPARQL language cannot refer to the same blank node from different CONSTRUCT patterns, so it is impossible to connect more sub-solutions to one blank node. On the other hand, the SeRQL language introduces global identification of blank nodes, although it roughly breaks the RDF blank node semantics.

The philosophical troubles include that the current languages are not closed. So a solution of a query cannot be used as an input for other query. In addition, they do not use RDF to represent solutions of their queries. It is not a good property of the current RDF query languages.

1.2 A new query language

Due to these troubles, we have decided to design a new query language [7]. Several requirements have been taken on the proposed query language. It must be applicable for a general purpose. It must be closed, so it must use the RDF data model for solutions of its query. The language must also be strong as for selecting data so as for creating data.

Patterns have been used as a base of the language. They are widely used in other languages and they are very intuitive for a user. It is possible to say that everything in the new language is a pattern. Not only selecting data but whole evaluation is controlled by the patterns. However, many languages use patterns and they are weak, so there is the need to do the patterns stronger. The new language uses a simple way to do this. It makes it possible to name a pattern and to use it later by its name. So it is possible to make a recursive pattern. For example, a selection of a RDF Collection is simple. A pattern selects one element of the RDF Collection and then it uses itself on the rest of the RDF Collection recursively.

The troubles with blank nodes are resolved by introduction of local identification for a blank node, which make it possible to refer to a blank node. However, it still keeps the fact that a concrete name of a blank node is not significant and that it is not global.

The new language has been implemented for the semantic web infrastructure Trisolda [5] and it has got the name Tequila, which comes from Trisolda Query

Language. The next section describes the language in detail. Section 4 presents some examples of usage of the language.

2 The Tequila language

The main feature of the Tequila language is its ability to name a pattern (Section 2.3), which makes it possible to express a recursive pattern. Other powerful features include query composition (Section 2.5), usage of multiple data sources (Section 2.6), a good support for blank nodes (Section 2.10), and the ability to construct new triplets (Section 2.4).

2.1 Base of the language

The Tequila language is based on the SPARQL language, thus it is a pattern-based query language. It fully adopts the syntax and semantics of URIs (including qnames), literals, blank nodes, variables and comments. It also adopts the syntax and semantics of definitions of qname prefixes, which can be used in the prologue of a query.

The syntax of Tequila patterns is also similar to the syntax of SPARQL patterns, but its semantics is totally different. The main difference is that a solution of a Tequila query (or generally a pattern) is not a solution mapping, but a RDF graph is. Variable bindings, which are called a solution mapping in the SPARQL language, have effect only during evaluation of a pattern.

A pattern can have more than one solution. A solution can evoke some variable bindings. If a variable is bound to some value, then the next use of this variable is equivalent to use this value, so the binding influences solutions of other patterns. Due to this fact, it is reasonable speak about a solution (or solutions) of a pattern with respect to the actual variable bindings. A binding of a variable is active, until the solution that evokes this binding is refused.

In the next text, the word "solution" always means "a solution with respect to the actual variable bindings", but sometimes it is used the full form due to emphasis.

2.2 Base patterns

This subsection introduces base patterns, which are more or less adopted from the SPARQL language. Tequila language specific patterns are introduced in the next sections.

Query patterns. A Tequila query, which is fully called a query pattern, consists of the keyword **get** followed by a compound subpattern. A solution of a query pattern is the union of all solutions of its compound subpattern. In the case of a main query, a solution of the query may be represented as a sequence of solutions of the subpattern.

Compound patterns. A compound pattern consists of a sequence of subpatterns enclosed in braces. It is possible use any type of a pattern as its subpattern. The semantics of the evaluation of a compound pattern is a little bit complicated, but it is very important for the Tequila language. However, we hope that this semantics will be familiar for people who know the semantics of the Prolog language.

For the evaluation of a compound pattern, its subpatterns are subsequently evaluated. If some subpattern (with respect to the actual variable bindings) has no other solution, the evaluation goes back on the previous subpattern, it refuses the solution of this subpattern and it searches for the next solution of this subpattern. If it also has no other solution, the evaluation goes back again. If a solution is found, the evaluation continues in search for solutions of the next subpatterns. The evaluation of the next subpatterns starts from scratch, so solutions that have been found previously have no effect on the next solutions.

If a solution of the last subpattern is found, then a solution of the compound pattern is found and it is equal to the union of solutions of the subpatterns. In order to search for the next solution of the compound pattern (i.e., after a refusal of the previous solution of the compound pattern), the solution of the last subpattern is refused and it is searched for the next one. If the first subpattern has no other solution, then also the compound pattern has no other solution.

Triplet patterns. A triplet pattern is a basic construct for selecting data from a data source. The base form of a triplet pattern is similar to a triplet pattern from the SPARQL language. A solution of a triplet pattern is one triplet from a data source that matches the pattern with respect to the actual variable bindings. The next solutions return other triplets that match the pattern from the data source. When a solution (i.e. a triplet) is found, the unbounded variables of a triplet pattern are bound to values according to the values from the found triplet. Note that a blank node term do not have a variable character as in the SPARQL language, it is just identification of a blank node. See section 2.10.

Filter patterns. A FILTER pattern is the only direct way how to test values of variables. A FILTER pattern consists of the keyword **filter** followed by an expression and enclosed by the point. If a value of a FILTER expression is true (with respect to the actual variable bindings), then the FILTER pattern has just one solution, namely an empty RDF graph. If its value is false, then the FILTER pattern has no solution.

The semantics of a FILTER pattern might looks strange. However, in combination with the semantics of a compound pattern, it has a very good point. If a value of the expression is false, then the FILTER pattern has no solution and thus the evaluation of the compound pattern goes back, so that a bad solution is filtered. If a value of the expression is true, then a solution of the FILTER pattern is just an empty graph and thus the evaluation of the compound pattern goes through.

```
get
{
    ?author ex:name "Milan Rufus".
    optional ?author ?ex:born ?place.
    get { ?book ex:author ?author. }
}
```

Listing 1.1. A simple example for a library system

Union patterns. A UNION pattern is useful for describing variants. It consists from two subpatterns connected by the keyword **union**. A solution of a UNION pattern is initially searched in the first subpattern. After what the first subpattern has no other solution, then the solution of the UNION pattern is searched in the second subpattern.

Optional patterns. An OPTIONAL pattern consists of the keyword **optional** followed by a subpattern. If the subpattern has any solution, then the evaluation of an OPTIONAL pattern is equivalent to the evaluation of its subpattern. If the subpattern has no solution, then a solution of the OPTIONAL pattern is an empty RDF graph. Hence, an OPTIONAL pattern has always at least one solution.

Example. Listing 1.1 shows an example of a simple query, which finds information about Milan Rufus. It finds a triplet with the object "Milan Rufus" and it binds his URI to the variable `?author`. Then it selects his birth place, if this information is included in a data source. Finally, it selects all his books.

2.3 Data selection

As mentioned above, the ability to select data is improved by the introduction of the named patterns.

Named patterns. A named pattern is identified by a URI. The URI is used only for identification and it need not meet any special conditions. A definition of a named pattern must precede a main query. It consists of a named pattern URI followed by a list of formal parameters (i.e. list of variables) enclosed in parentheses. A compound pattern that represents the body of the named pattern follows.

A use of a named pattern (i.e. the named pattern itself) consists of the keyword **use** followed by the URI of the named pattern and a list of actual parameters enclosed in parentheses. The evaluation of a named pattern is equivalent to the evaluation of the named pattern body that has been specified in the definition of the named pattern. However, there are some differences. Variables that are used in the named pattern body are local for the evaluation of the named pattern. Therefore, before the evaluation of the named pattern starts, a copy of its body is made and the variables of its formal parameters are unified with the corresponding actual parameters. So if an actual parameter is a value, then a corresponding formal parameter variable is bound to this value. If an actual parameter is a variable, then a corresponding formal parameter variable is considered to be the same as the actual parameter variable.

A named pattern can also use a variable instead of a URI. If this variable is bound to a URI value, then evaluation of the named pattern is equivalent to the case in which it is directly used the URI value. If the variable is not bound to a URI value, then the named pattern has no solution.

If the URI that is used in a named pattern does not appear in any definition of a named pattern, then the named pattern has no solution.

Imports of named patterns. Some definitions of named patterns can be useful for many different queries. So it is useful to have a mechanism that allows reuse of them.

In the Tequila language, it is possible to write down definitions of named patterns to a file. This file then can be import to a query. An IMPORT directive consists of the keyword **import** followed by a file name enclosed by quotation marks. IMPORT directives follow PREFIX directives. PREFIX or IMPORT directives can be also used in an imported file, then an effect of the PREFIX directive is local for the file.

Example of a named pattern. Listing 1.2 shows the definition of the simple recursive named pattern `rdf:list`, which selects a RDF Collection. It is just an example of an import file, so it does not contain a main query.

The formal parameter `?N` is a resource that represents a RDF Collection. If it is different from `rdf:nil`, then the first case of the UNION pattern (line 5) matches the first element of the RDF Collection. In detail, the first triplet pattern (line 6) matches the `rdf:first` property triplet and the second triplet pattern (line 7) matches the `rdf:rest` property triplet of the element. Then the named pattern use itself (line 8) on the rest of the RDF Collection recursively. If the resource that represents the RDF Collection is `rdf:nil`, then the second case of the union matches it (line 11).

It is good to say that if somebody creates a cyclic RDF Collection, then this named pattern never ends its evaluation. It is possible to adapt this example to be cyclic safe, but then this example becomes more complicated and less effective.

2.4 Data construction

Now it is time to show how it is possible to construct a new triplet in the Tequila language. It is possible in just one way by using of a CONSTRUCT pattern.

Construct patterns. A CONSTRUCT pattern consists of the keyword **construct** followed by a triplet of variables, URIs, literals or blank nodes enclosed

```
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

rdf:list(?N)
{
    {
        ?N rdf:first ?F.
        ?N rdf:rest  ?R.
        use rdf:list(?R)
    }
    union
    {
        filter ?N = rdf:nil.
    }
}
```

Listing 1.2. The RDF Collection pattern

by the point. A CONSTRUCT pattern has just one solution, which is equal to its triplet with respect to the actual variable bindings. If some variable of the CONSTRUCT pattern is not bound to a value, then the variable is bound to a new (unique) blank node.

More CONSTRUCT patterns can be enclosed by braces following the keyword **construct**. The individual CONSTRUCT patterns then have not the keyword **construct**.

2.5 Query composition

The ability to create a query composition is very important for a query language. The Tequila language has a more general construct.

From pattern. A FROM pattern consists of two subpatterns connected by the keyword **from**. The second subpattern is used as a source pattern for the first subpattern. Initially, a solution of the source pattern is found and it is used as a data source for the first subpattern. Solutions of the FROM pattern are then searched in the first subpattern. If the first subpattern has no other solution, then the next solution of the source pattern is searched for and it is used as data source for the first pattern again. The evaluation of the first pattern starts from scratch. If the source pattern has no other solution, then the FROM pattern also has no other solution.

2.6 Multiple sources

Other useful property of a query language is the ability to combine multiple data sources. Also this property is supported by a pattern.

Source pattern. A SOURCE pattern consists of the keyword **source** followed by a URI of a data source. Usually, it is used in combination with a FROM pattern.

A SOURCE pattern has just one solution, which includes all triplets of the data source of the pattern. It is possible to use a variable instead of a URI of a data source. If the variable is bound to a URI, semantics is same. If the variable is not bound to a URI, solution is an empty RDF graph.

2.7 Other patterns

Now we introduce other patterns, which are also very important for the ability to use the language for a general purpose.

Match patterns. It is sometimes useful to bind variables to values according to the content of a data source. It is especially useful when we want to perform some conversion of a data source but we do not want to include the original content of the data source to a solution. Although some pattern can make this binding, it also returns some triplets in its solution. To avoid this, a MATCH pattern has been introduced.

A MATCH pattern consists of the keyword **match** followed by a subpattern. The evaluation of a MATCH pattern is equivalent to the evaluation of its subpattern, but with one difference. If the subpattern has a solution, then a solution of the MATCH pattern is an empty RDF graph. If the subpattern has no other solution, then the MATCH pattern also has no other solution. Thus only variable bindings are performed and the content of the solution is ignored.

Else patterns. An ELSE pattern is a way how to express a graph condition. An ELSE pattern consists of two subpattern connected by the keyword **else**. If the first subpattern has any solution, then the evaluation of the ELSE pattern is equivalent to the evaluation of the first subpattern. If the first subpattern has no solution, then the evaluation of the ELSE pattern is equivalent to the evaluation of the second subpattern.

Any patterns. For a recursive walk through a graph, it is useful to have the ability to select just one solution from all possible solutions. It is a goal of an ANY pattern. An ANY pattern consists of the keyword **any** followed by a subpattern. If its subpattern has a solution, then a solution of the the ANY pattern is an arbitrary solution of solutions of the subpattern and the pattern has no other solution. If its subpattern has no solution, then the ANY pattern has also no solution.

2.8 Syntactic sugar

There are some other constructs for more comfort. They just form a syntactic sugar and they do not increase a power of the language.

Where patterns. A WHERE pattern is a syntactic sugar for people familiar with the SQL-like syntax. A WHERE pattern consists of two subpatterns connected by the keyword **where**. The pattern `subpattern1 where subpattern2` is equivalent to the following construct:

```
match {subpattern2} {subpattern1}
```

Uniplet pattern. Sometimes, it is useful to create (or to select) only a single value, not a whole triplet. On the other hand, it is not good to change the RDF data model. So the Tequila language introduces the syntactic sugar for a uniplet.

The uniplet `node`, which can be used instead of a triplet in a triplet pattern or a CONSTRUCT pattern, is equivalent to the triplet `tql:shadowSubject tql:shadowPredicate node`[1].

Get-where-from query. If either a main query (i.e. query pattern which is not used as a subpattern) or a expression query has a following structure[2]:

```
get {get {getpattern} where {wherepattern} from {frompattern}}
```

then the outer keyword **get** with its braces can be omitted.

2.9 Pattern operator priority

By now, we only say that a pattern consists of some subpattern in the definitions and we do not say, which types of the subpatterns are possible to use. A restriction on types of subpatterns is needed to have an unambiguous grammar for the language.

It is possible to look at the keywords that are used for specifying patterns as on pattern operators and on the braces of compound patterns as on parentheses. The restriction on types of subpatterns is then done by an operator priority. The operators have following descendent priorities:

1. `any`, `match`, `optional`, `get`
2. `where`
3. `from`
4. `else`
5. `union`

2.10 Blank nodes.

According to the RDF specification, blank nodes have no names. However, for many reasons, it is necessary to introduce some kind of identification of blank nodes. One of the reasons is the need to identify blank nodes in a data source and to distinguish them from each other. Other reason is the necessity to have the ability to refer to a concrete blank node in a triplet CONSTRUCT pattern.

The syntax of the Tequila language as well as of the SPARQL language include a blank node term[3], but the semantics are totally different. In contrast to the SPARQL language, where the semantics of a blank node term is more similar to the semantics of a variable, a blank node term is just a local name of a blank node in the Tequila language. If a blank node term had the semantics

[1] The `tql:` prefix is bound to the URI http://ulita.ms.mff.cuni.cz/tequila/term#.
[2] The WHERE part or the FROM part can be missing.
[3] A qname with the prefix _:

similar to a variable, then a use of the blank node term would not be equivalent to a use of a variable bound to a blank node. It is other reason for the used semantics.

Although a blank node term means an identification of a blank node in the Tequila language, the semantics of the Tequila language keep the fact that blank nodes from two different data sources can never be equal. Also it keeps the fact that a concrete form of a blank node name that is used in a non-query data source is hidden for a user. These are the main ideas of blank nodes.

How does it do that? A name of a blank node has two components. The first component determines a data source name; the second component determines a local name of the blank node in the data source. Two blank nodes are equal, if their source names and their local names are equal. Therefore, two blank nodes from two different sources can never be equal, because their source names are different.

A blank node term written directly in a pattern determines the blank node that has the source name `[query]` and the local name that is equal to the local name of the qname of the blank node term. Therefore, a blank node term can never match a blank node from a non-query data source, so that the fact that a concrete form of a name of a non-query blank node is hidden for a user is kept. Finally, note that a blank node created by a CONSTRUCT pattern during binding unbound variables has the source name `[construct]`.

2.11 Expressions

A base of the syntax and semantics of the Tequila expressions is adopted from the SPARQL language. In addition, it is extended by various concatenations and a query expression. The LIKE operator from the SeRQL language is also adopted.

A Tequila expression is used in a FILTER pattern in the same way as in the SPARQL language. In addition, it is also possible to use it in a triplet pattern, a CONSTRUCT pattern or a named pattern instead of a RDF term. In this case, an expression must be enclosed by parentheses.

If a pattern contains an expression, then (before the evaluation of the pattern begin) the expression is evaluated first and its result value is used during the evaluation of the pattern instead of the expression as a RDF term. If the evaluation of the expression returns the error value, then the pattern has no solution.

It is important to point out that an element `?V` used in a pattern is a variable, but an element `(?V)` is an expression. If the variable is bound to a value, the effect is same. But if the variable is not bound to a value, then the evaluation of the expression returns the error value and the pattern has no solution.

Literal concatenations. A literal can be concatenated with other literal or a URI. The simple string concatenation is used; a datatype and a language tag are ignored, so a result of the literal concatenation is always a simple literal.

URI concatenations. A URI can be concatenated with other URI or a literal. The simple string concatenation is used again. The URI concatenation is important, for example, for creating a RDF Container, where it is necessary to create URIs that have the form `rdf:_`*n*.

Blank node concatenations. A result of a concatenation of two blank nodes is a blank node, which source name is obtained by concatenation of their source names and its local name is obtained by concatenation of their local names. An exception is a concatenation of two query blank nodes. In this case, the source name of a result is `[query]` again, because local names of query blank nodes are not considered as hidden for a user.

If the second operand is not a blank node, then the source name of the second operand is rated as `[query]` and the local name of the second operand is rated as a result of the `STR` function applied on the second operand. Therefore, it is not possible to get any other blank node from knowledge of some non-query blank node, so the fact that a name of a blank node is hidden for a user is still kept.

The blank node concatenation is a flexible way, how to create a new blank node and be able to refer to them. For example, if there is the need to create a blank node for each work group, then we simply create one by a concatenation of a blank node base with a work group URI. If there is need to have a blank node for each person, we concatenate other blank node base with a person URI. If a person is added to different work groups, then this procedure creates always the same blank node for this person. In addition, if there is the need to have a blank node for each combination of a person and a work group, a solution is also simple. We concatenate some blank node base with a work group URI and a person URI.

Determining a type of the + operation. The numeric addition and all types of the concatenations use the same symbol +. Determining a type of the operation depends on the type of its first operand.

Query expressions. It is possible to use a query pattern as a unary expression. A query expression is evaluated as a normal query pattern. If its solution has just one triplet, then the object of the triplet is a result value of the expression. If the solution has more than one triplet, then one triplet is selected from the solution randomly and its object is used. If the solution has no triplet, then a result value of the expression is `"false"^^xsd:boolean`.

3 Possible improvements

Apparently, the Tequila language is not finished yet and there is still space for improvements. Specially, it is in the following areas:

```
 1 prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
 2 prefix ex:   <http://www.example.org/term#>
 3
 4 ex:convert(?bag, ?list)
 5 {
 6     {
 7          match any
 8          {
 9              ?bag ?pred ?item.
10              filter ?pred like ("" + rdf: + "_[0-9]+").
11          }
12
13          use ex:convert(?bag, ?sublist) from get
14          {
15              ?bag ?Y ?Z.
16              filter ?Y != ?pred && ?Z != ?item.
17          }
18
19          construct ?list rdf:first ?item.
20          construct ?list rdf:rest  ?sublist.
21     }
22     else
23     {
24         match ?list. from construct {rdf:nil.}
25     }
26 }
```

Listing 1.3. Converting a RDF Collection into a RDF Bag

1. The expressions should be more powerful. A good thing is a general support for different datatypes, not only for built-in datatypes.
2. A support of a RDF dataset will bring the ability to have multiple sets of triplets in one pattern solution. This may be useful for some type of problems.
3. A support for a RDF representation of a pattern will bring the ability to store the pattern in a data source. Therefore, it will be possible to store data together with the patterns, which can manipulate with these data.

4 Examples

This section presents two interesting examples. The first one shows how to convert a RDF Collection into a RDF Bag. The second one shows how to evaluate the aggregation function `max` without the need to have a built-in support for it.

4.1 Conversion of a RDF Collection into a RDF Bag.

This example is shown in Listing 1.3. The conversion is performed by the named pattern `ex:convert`, which has two formal parameters. The parameter `?bag` represents an input bag and the parameter `?list` represents an output collection. If the input bag is not empty, then the first case of the ELSE pattern (line 6) is a success. The ANY pattern (line 7) selects one item from the bag. Due to the MATCH pattern (line 7), only variable bindings are considered. Then the `ex:convert` pattern uses itself on the rest of the bag (line 13). The rest of the bag is generated by the query pattern (line 13). Finally, a new collection is made (line 19) from the selected item (variable `?item`) and from the created sub-collection (variable `?sublist`). The variable `?list` (line 19) is not bound

```
 1 prefix tql: <http://ulita.ms.mff.cuni.cz/tequila/term#>
 2
 3 tql:max()
 4 {
 5     {
 6         match any ?subj ?pred ?obj.
 7
 8         match ?max. from use tql:max() from get
 9         {
10             ?X ?Y ?Z.
11             filter !(?X = ?subj && ?Y = ?pred && ?Z = ?obj).
12         }
13
14         {
15             filter ?obj > ?max || ?max = "none".
16             construct ?obj.
17         }
18         else
19         {
20             construct ?max.
21         }
22     }
23     else
24     {
25         construct "none".
26     }
27 }
```

Listing 1.4. Evaluation of the aggregation function max.

to a value, so the CONSTRUCT pattern bind it to a new blank node, which will represent the new collection.

If the bag is empty, then the second case of the ELSE pattern (line 23) is a success and the parameter `?list` is bound to the empty collection `rdf:nil` (line 24) by the MATCH pattern (line 24).

4.2 Evaluation of the aggregation function max.

This example, which is shown in Listing 1.4, is partially similar to the previous one. The aggregation function `max` is implemented by the named pattern `tql:max`. The maximum value is calculated from the objects of the triplets of an input data source.

If the data source is not empty, then the first case of the ELSE pattern (line 5) is a success. The ANY pattern and the MATCH pattern (line 6) select just one triplet and they bind the object of the triple to the variable `?object`. Then the `tql:max` pattern uses itself (line 8) on the rest of the input to get the maximum value of the rest. The rest of the input is generated by the query pattern (line 8). The maximum value of the rest is bound to the variable `?max` by the MATCH pattern (line 6). Then the variable `?max` and the variable `?object` are compared by the ELSE pattern (lines 18 and 15) and the biggest value is constructed (line 16 or 20). If the variable `?max` is bound to the value `"none"`, then a value bound to the variable `?object` is constructed.

If the data source is empty, then the second case of the ELSE pattern is a success (line 23) and the special value `"none"` is constructed.

5 Conclusion

In this paper, we introduce the new pattern-based query language Tequila. We demonstrate its power on several examples, which are difficult (or impossible) to express in other languages. Although the Tequila language is good suitable to solve these examples, it does not use any single-purpose constructs to express them. The language is designed to involve only general-purpose constructs.

The main construct of the language is a named pattern, which can be used to express a recursive query. This construct makes the Tequila language different from other query languages and makes the language so strong. On the other hand, it is possible to specify a query, for which the evaluation will never finish. It is one of disadvantages of the Tequila language, but it is the tax for the power.

Acknowledgements

This research was supported in part by Czech Science Foundation Project 201/09/06

References

1. ARQ - Property Paths, as accessible in February 2009.
   URL `http://jena.sourceforge.net/ARQ/property_paths.html`.
2. User Guide for Sesame,Chapter 6. The SeRQL query language (revision 1.2).
   URL `http://www.openrdf.org/doc/sesame/users/ch06.html`.
3. RDF Primer, W3C Recommendation, February 2004.
   URL `http://www.w3.org/TR/2004/REC-rdf-primer-20040210/`.
4. SPARQL Query Language for RDF, W3C Recommendation, January 2008.
   URL `http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/`.
5. Jiří Dokulil, Jakub Yaghob, and Filip Zavoral. Trisolda: The environment for semantic data processing. In *International Journal On Advances in Software 2008*, volume 1. IARIA, 2009.
6. Tim Furche, Benedikt Linse, François Bry, Dimitris Plexousakis, and Georg Gottlob. Rdf querying: Language constructs and evaluation methods compared. In *Reasoning Web, Second International Summer School 2006*, volume 4126 of *LNCS*. 2006.
   URL `http://www.pms.ifi.lmu.de/publikationen/#PMS-FB-2006-33`.
7. Jakub Galgonek. Query languages for the Semantic web. Master thesis, Charles University in Prague, Czech republic, 2008.
   URL `http://siret.ms.mff.cuni.cz/galgonek/thesis/thesis.pdf`.
8. Peter Haase, Jeen Broekstra, Andreas Eberhart, and Raphael Volz. A comparison of rdf query languages. In *Proceedings of the Third International Semantic Web Conference, Hiroshima, Japan, 2004.*, November.
   URL `http://www.aifb.uni-karlsruhe.de/WBS/pha/rdf-query/rdfquery.pdf`.
9. Michael Sintek and Stefan Decker. Triple - a query, inference, and transformation language for the semantic web. In *ISWC '02: Proceedings of the First International Semantic Web Conference on The Semantic Web*, pages 364–378, London, UK, 2002. Springer-Verlag.
   URL `http://portal.acm.org/citation.cfm?id=646996.711416`.

The BPM to UML activity diagram transformation using XSLT*

Ondřej Macek[1] and Karel Richta[1,2]

[1] Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University,
Karlovo náměstí 13, 121 35, Praha 2, Czech Republic
{macekо1, richta}@fel.cvut.cz

[2] Department of Software Engineering, Faculty of Mathematics and Physics,Charles University,
Malostranské náměstí 25, 118 00, Praha 1, Czech Republic
karel.richta@mff.cuni.cz

Abstract. The Business Process Model represented as a diagram in Business Process Modeling Notation (BPMN) is a commonly used way how to describe business processes of an organization. Problems connected with a complexity of notation and missing support in tools for the software development can be solved by a transformation to a Unified Modeling Language activity diagram. Another reason for creating such a kind of transformation is that it can solve problems of time, cost and quality associated with software creation in the scope of Model Driven Development.
This article describes common problems with the transformation of a BPMN diagram to a Unified Modeling Language activity diagram. One of the key features of the described transformation is that it is tool independent. This feature was achieved by using an XML metadata interchange representation of both models as an input and output and by using XSLT transformation for the model transformation itself.

Keywords: BPM, BPMN, UML, model transformation, XSLT

1 Introduction

The Business Process Model (BPM) is a model which describes business processes of an organization. It is an important tool for understanding the activities and information which are typically used to achieve business goals. So far it is a popular way of describing and improving business processes. The BPM can be described in various notations: in Business Process Modeling Notation (BPMN) [1], in Eriksson-Penker's notation [2], or sometimes as the Unified Modeling Language (UML) activity diagram (UML-AD) [3].

* This paper was partially supported by the MŠMT grant No. MSM 6840770014. This research has also been partially supported by the grant GACR No. GA201/09/0990

The business process modeling is a recommendation for software development according to the Unified Process [4]. The business process model is one of utilizable analytical models. The aim of the business process modeling in the phase of analysis is to understand processes in a domain.

2 The Need of the Transformation

The BPMN is commonly used by business process managers; therefore this part of software analysis is often done before the process of software development even starts. The BPM created by a business process manager is often represented as a diagram in Business Process Modeling Notation because BPMN has a variety of symbols allowing the description of the process effectively and in detail. Although this representation is correct, it can have several disadvantages - the BPMN is not fully supported by modeling tools used in the software development and in the scope of support of Model Driven Software Development (MDD) there exist only a few methods how to transform BPMN to other models.

Another problem connected with the BPMN is its complexity. The BPMN is designed as a tool for efficient modeling of a business process; therefore it contains symbols which represent non-atomic action and thus speeding the process of modeling. This is an advantage for a creator of a model, but not for a reader unfamiliar with the notation. And in phase of software analysis the analysts (a creator of the model) often consult its models with such a kind of reader. Therefore it will be useful to have the possibility to present the BPMN diagram also in different notation. Different notation can carry new point of view at the BPM or it can be easier to read and to understand by a customer (a reader of the model).

There were published articles [5], [6] handling with the transformation between the business process model and the UML models, especially the transformation into the use case model (diagram) and to the class diagram. If we look closer at these transformations, we will see that these transformations are information-loss, because both final diagrams contain only information about the names of actors and actions and about the connection between them, but information about the action flow is lost. With regard to this fact it will be very useful, if a transformation is created between the BPMN and some of UML behavioral diagrams (activity or sequence), because this transformation will preserve both information - about the action flow, actors and action names.

From this point of view, the representation of the BPM according to the notation of UML-AD can be more useful. At present, the support and the usage of UML standard is matter of fact. There will be no problem in supporting UML-AD in modeling tools. UML-AD can also help with the problem associated to the complexity of BPMN. If we compare both notations, we realize that a lot of symbols of BPMN cannot be easily represented as the only one symbol in UML-AD. Often one symbol of the BPMN is represented as a complex structure of elements in UML-AD. Therefore UML-AD can be considered as more naive and easier to understand, because the reader has to understand the meaning of fewer symbols.

Despite of these reasons, it seems that the transformation between BPMN and UML activity diagram is needed, although it will be a transformation between two models which are used to describe business processes. The transformation will allow us to use already created BPMN diagrams in the software development, and the transformation can also assist in the communication with a customer

3 Transformation Method

The transformation between models can be realized in various ways. It should serve as a bridge among two different models and between tools which support BPMN and tools supporting UML-AD. It will be useful if it is created in such a way that it will be easy to implement as a plug-in to existing tools and can be also used independently of any actual tool.

The first idea how to create the transformation was to create it by an Object Management Group (OMG) standard for model transformations - Query View Transformation language (QVT) [7], but there are not many modeling tools with a QVT processor and the support for both BPMN and UML-AD.

The problem with a missing language processor in modeling tools is the problem associated with the most of all existing modeling and transforming languages or with a language that we can create. Another problem connected with the usage of the modeling language will be how to guarantee the interoperability between the modeling tools, because each tool uses a different internal representation of a model. Therefore, an alternative way of transformation is needed.

We decided to use model representation based on extensible markup language (XML) [8]. This XML representation is standardized as the XML Metadata Interchange (XMI) standard [9] by the OMG as an instrument for the transport from one modeling tool to another. Thus, the XMI standard is based on the XML and common tools working with XML can also be used to work with XMI. The most interesting one is XSLT [10], because it allows transformation from one XML document to another one and enables any change in its structure. The combination of XMI and XSLT satisfies conditions on the tool independence and solves the problem of a model representation. Similar approach was used at the transformation between UML models in [11].

4 The Input and Output

As was written earlier, the input and output of the transformation will be the XMI representation of appropriate models. The input will be an XMI representation of a diagram in BPMN - see chapter 4.1 - and the output will be an XMI representation of a UML-AD - see chapter 4.2. The XMI standard describes the nodes in the diagram and the way how the nodes are connected by edges. Unfortunately, the description of the graphical representation of a model is not defined in the XMI standard. The description of the graphical part is typically hidden in an XMI element extension which is defined as a container for the tool-specific data. Therefore, the graphical layout is not universally transportable and in the transformation we will handle the diagram transfer only.

4.1 Description of the BPMN in XMI

The BPMN has no standardized XMI representation. Therefore, we decide to use the representation which is used in Altova UModel [12], so BPMN diagrams will be easy to visualize and XMI easy to generate. The BPMN representation is based on similarities between BPMN and UML-AD. The BPMN model is described by extending UML-AD XMI representation. This extension is made by adding new elements and attributes to the UML-AD representation. In this way a new profile is defined, which is a part of the BPMN XMI document and serves as a declarative reference.

Every symbol in the BPMN has a corresponding element in XMI which defines its type and attributes. Attribute xmi:type refers to the UML-AD symbol which is extended and xmi:extension element and its subelements are used to define the features typical for the BPMN by referring to the profile stored in the document. Following XML snippet represents basic start event node:

```
<node xmi:type="uml:AcceptEventAction"
      xmi:id="Uacfd64a5-9c7e-4eaf-995d-ab3849b7f8c9" name="start node">
 <xmi:Extension extender="UModel">
  <appliedStereotype xmi:type="uml:StereotypeApplication"
                     xmi:id="Ufbe96d2e-05fd-4c35-a6e9-49ab669549e2"
 classifier="U00200106-7510-11d9-86f2-000476a22f44">
    <slot xmi:type="uml:Slot" xmi:id="U1e97ab77-7cb5-47c9-bb56-28b4eb3a77fd"
  definingFeature="U00080106-7510-11d9-86f2-000476a22f44">
     <value xmi:type="uml:InstanceValue"
            xmi:id="Ucec9e7bc-8716-4662-a2a5-de44a55930b7"
            instance="U00100106-7510-11d9-86f2-000476a22f44"/>
    </slot>
    <slot xmi:type="uml:Slot" xmi:id="Ub80e3e24-3d83-437d-bf2f-46a84c93a341"
          definingFeature="U00220106-7510-11d9-86f2-000476a22f44">
     <value xmi:type="uml:InstanceValue"
            xmi:id="Uc01150e2-3256-4052-9506-450485581e7c"
            instance="U00140106-7510-11d9-86f2-000476a22f44"/>
    </slot>
   </appliedStereotype>
  </xmi:Extension>
 <clientDependency xmi:idref="U54b4a7eb-4e3f-4964-9323-268e4ffb5164"/>
</node>.
```

This representation has the advantage that the connection between the symbols is mostly obvious. This fact helps to create transformation rules. On the other hand, the BPMN XMI file is illegible for humans and, moreover, the slot elements sometimes contain redundant information (the information was defined earlier or it is not necessary to define). Therefore, it is good reason why to improve this XMI notation in the future.

4.2 UML-AD Description in XMI

The XMI representation of the UML-AD is defined by the OMG as a part of the UML standard (current definition can be found in [13]), therefore the XMI file dedicated to transport the UML diagram uses the namespace:

```
http://schema.omg.org/spec/UML/version,
```

where the version represents the number of a current UML version (e.g. 2.1.2 for used UML version)used to describe symbols contained in the diagram. The XMI file of a UML activity diagram contains only the description of the nodes in the diagram and the way they are connected. This is based on their definition in the namespace. No extra profile is required. Since the description is based on the usage of the UML namespace, the model description is very straightforward and easy to read. This is the example of the XMI representation of an UML-AD initial node (describes the same as the example of start event node in chapter 4.1):

```
<node xmi:type="uml:InitialNode" xmi:id="U002" name="start node">
 <outgoing xmi:idref="U004"/>
</node>.
```

If both ways of the diagram representation are compared, it is obvious that creating an XMI profile for the BPMN similar as the XMI profile for the UML-AD, will be very useful.

5 Transformation problems

Although both models are very similar, the transformation from the BPMN to the UML-AD is not as straightforward as it seems. Although both models describe the same thing and use similar symbols, the models differ fundamentally. It is caused by the complexity of the BPMN symbols. In the UML activity diagram, every symbol represents one concrete and atomic information. On the other hand, the symbols in the BPMN compress the information. For example the symbol for the loop (see Fig. 1) in the BPM has no appropriate equivalent in the group of UML-AD symbols, because the loop symbol contains non-atomic information (information about the action and information about the loop condition). The compression of the information is very useful for business process managers, because it makes the diagram more synoptical, and therefore easy to create and read. On the other hand,this complexity complicates the transformation from the BPMN to the UML-AD.

If we compare the BPMN and UML-AD, we will see that some symbols can be transformed directly by using one-to-one transformation (e.g. the task node in the BPMN is transformed to the UML activity node behavioral action), but in most cases the compression of the information in BPMN symbols causes

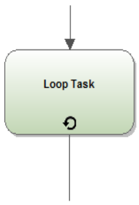


Fig. 1. The BPMN symbol for loop.

the transformation is not one-to-one (one BPMN symbol to one UML activity symbol) but typically one-to-many.

To demonstrate this fact there can be used the example of the BPMN loop symbol (see Fig. 1), whose complexity was mentioned in the previous text. This single symbol in the BPMN could not be transformed to one symbol of the UML activity diagram. The loop symbol has an appropriate representation in a construction consisting of four UML activity symbols - TASK, DECISION and two edges, where one edge leads from the TASK to DECISION and the other one leads backwards. The TASK represents the action and the decision node is used to resolve the loop condition. Based on the result of the condition the activity flow leads to the TASK or continues to the symbol following after the loop. The construction of the loop in the UML activity can have two different orders of the nodes. It depends on the fact if the loop condition is tested before or after the action (see Fig. 2).

Further complication lies in the fact that some loops are limited by the number of loops and not by the condition - in the BPMN this information is hidden in loop symbol attributes, but it does not hold for UML-AD. In this case, the UML-AD construction matching a BPMN loop symbol should contain an action initializing the counter of loops. The type of a loop determines the existence of two possible constructions. The loops differ in time of checking a loop condition - before and after the task. We assume that the counter is incremented in the loop action. The loop with the counter can have also two possible matches in UML-AD according the time when the condition is being checked. The incoming edges, which were connected to the original loop node, is also necessary to redirect according to the order of the task and decision symbols. Thus, one symbol in the BPMN is replaced by four nodes in the UML activity which can be in two different orders.

Besides the symbols which can be transformed as one-to-one or one-to-many, there exist also symbols which can be sometimes transformed using one-to-one

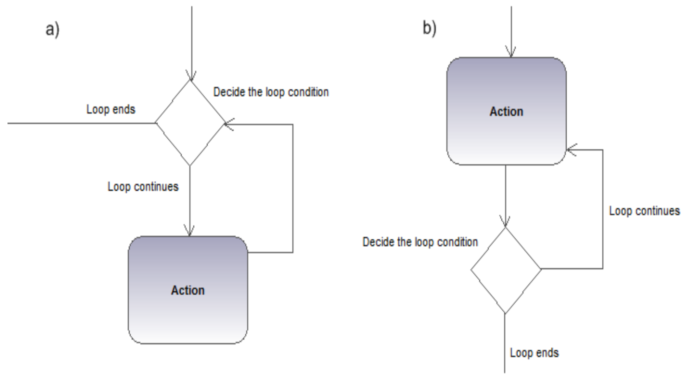


Fig. 2. Two possible representation of the loop in the UML activity diagram - a) shows loop where the condition is checked before the action is taken and b) shows the loop where the loop condition is checked after taking action.

and sometimes using one-to-many. Typical example is the symbol representing the start event which is transformed according to its trigger in both ways. If the type of a start event trigger is NONE, MESSAGE or TIMER, the node could be transformed as one-to-one, since there are appropriate symbols in the UML activity diagram. Other trigger types (RULE, LINK, and MULTIPLE), however, have not appropriate representation in the UML activity. That is why the start event must be, in this case, transformed to more nodes of the UML activity diagram.

Start event symbol can be transformed in two ways. First, as a construction consisting of INITIAL NODE and DECISION (to resolve the event type) and EDGE connecting the INITIAL NODE and the DECISION. In this case the trigger became a part of the process itself, therefore the UML-AD will have a little bit different meaning than the input BPMN diagram. Second, these nodes can be transformed by creating an INITIAL NODE with a note, where the trigger will be described. In this case, the trigger is not the part of the process.

Another problem is that some symbols in the BPMN can have two different meanings according to the position in the diagram, concretely in dependency on the number of edges leading to the node. Typical examples are decisions nodes. The decision nodes can be used in two different ways. The decision nodes enable either the branching or merging of the activity flow. As an example we can use a parallel decision node - see Fig. 3.

This problem can be solved either by a naive method or by a method of finding the pair. The naive method assumes that the decision node with two or more entering edges is a MERGE node. The method of finding the pairs will search the diagram through and it will find the decision nodes pairs. If the method finds a pair of decision nodes, the first of them (in the action flow) will

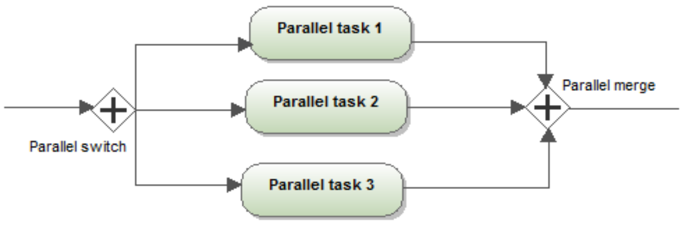


Fig. 3. The BPMN symbol for parallel gateway (rhombus with a cross inside) can be used for branching or merging the activity flow.

be transformed as a decision node and the other one as a merge node. In case that no pair is found the naive method can be used or all decision nodes in the BPMN are replaced by the decision nodes in UML-AD. In our transformation we use the naive method, so all decision nodes with two or more incoming edges will be transformed as merge nodes.

From the previous text it is obvious that the transformation of the BPMN to the UML activity is possible, because the BPMN and the UML-AD has a very similar representation of nodes. The structure of diagrams will differ strongly, because the BPMN symbols cannot be transformed to the UML-AD matching symbols one-to-one.

6 The Transformation

There was created an XSLT stylesheet to solve a proper transformation in between two discussed models. The transformation was tested on several BPMN diagrams. See Fig. 4 where is a model describing a process of receiving the order in a company. This diagram was exported to the XMI file and then the transformation stylesheet was applied. The result was imported to the modeling tool as is shown in Fig. 5. Both diagrams describe the same process. The only disadvantage was that the graphical layout of the activity diagram had to be created manually.

The realized transformation satisfies all conditions which were set in analytic parts of this paper and can be used in practice. Another positive feature is that the transformation is fully automatic and no user intervention is needed.

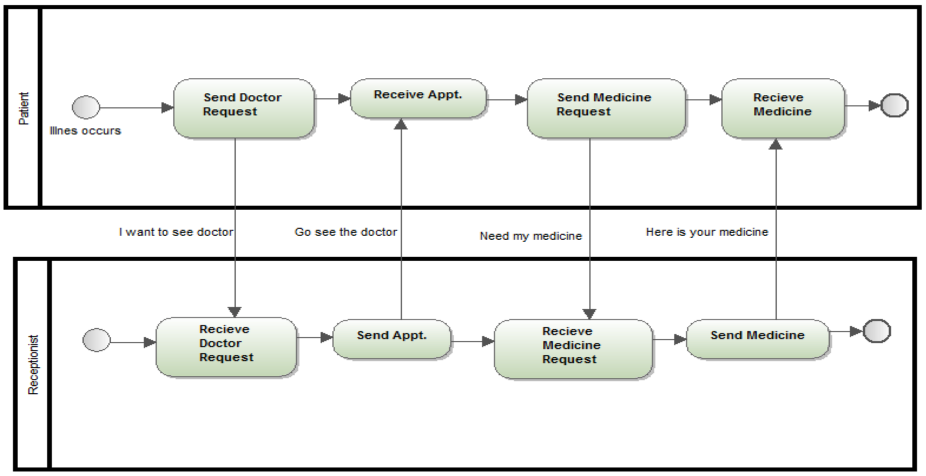


Fig. 4. The diagram in BPMN. Figure from [1]

The disadvantage of the transformation using XMI and XSLT is that it cannot be used so easily for keeping consistency between two models. To solve the problem, it is necessary to carry out a backward transformation (from UML-AD to the BPMN) so that the changes in a UML-AD diagram could be reflected back to a corresponding BPMN diagram.

7 The Backward Transformation

In the previous paragraphs it was stated that the backward transformation from the UML-AD to BPMN is needed. Basically, such a transformation can be realized by rewriting the UML-AD by BPMN symbols. This way of transformation will use the fact that UML-AD symbols match the BPMN symbols. If we like to have the output model more sophisticated, we can design the transformation by reversal rewriting the rules from the original transformation. In this case, we get a business process model that will contain also complex BPMN symbols, but not all BPMN symbols can be reached by an automatic transformation, since there is not enough information in the UML-AD. This lack of information can be solved by adding this information manually. The aim was to create an automatic transformation, and therefore we cannot use this solution.

Although the backward transformation is needed, it will be difficult to create it to keep consistency of both models. The backward transformation can create the BPMN diagram with the same information, which has the input UML-AD. Thus, we are able to transform one model to another and vice versa. In case of

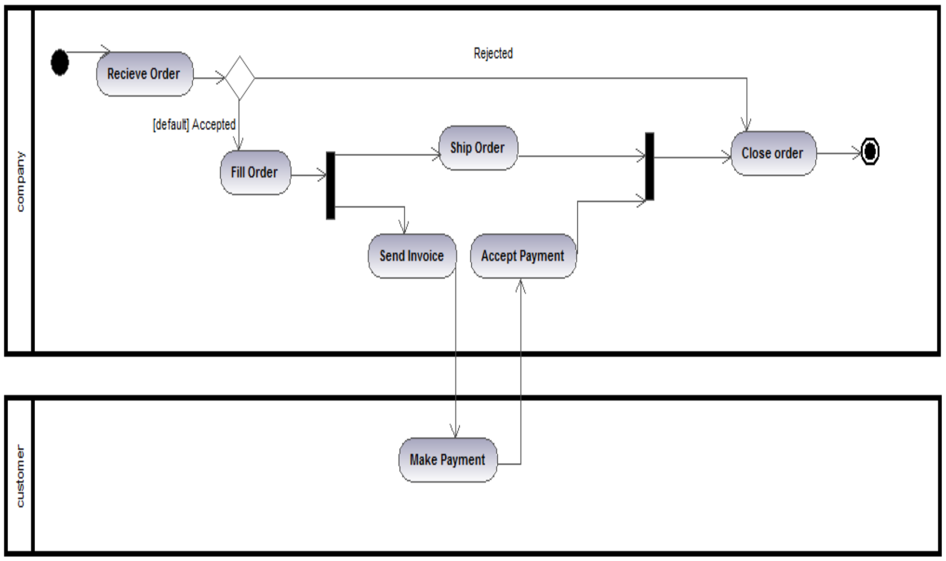


Fig. 5. The diagram in UML activity diagram which was created by using described transformation from the diagram in Fig. 4. The graphical layout was modified manually.

keeping the consistency, there appears a problem how to match the UML-AD symbols and construction with the BPMN symbols and how to solve the lack of information in UML-AD (some information which was part of the BPMN symbols can be not reachable in the UML-AD).

The problem connected with the symbol matching can be solved easily by using an unique identifier for matching symbols. Similar solution can be used for matching the UML-AD constructions and BPMN symbols. The UML-AD construction should have an identifier which will carry information that the symbols are parts of one construction and together are matched to concrete BPMN symbol.

The problem of missing information can be solved by adding parameters manually.

8 Conclusions

The transformation between diagrams in the BPMN and the UML activity diagrams is needed, because it will help to improve the development of software. The UML-AD has better support in modeling tools and is easier for the customer (a reader of the diagram unfamiliar with the BPMN). The transformation among these two notations can serve as a bridge between the tools supporting the business process management and the tools for the software development.

The transformation is realized by using representation of both models in XMI as the input and output. Transformation rules described in the form of XSL transformation satisfy the requirements on the tool independence and integration possibility. Disadvantage of this process is that the information about graphical layout of the model is lost.

The designed transformation should be completed by creating the backward transformation (from the UML-AD to the BPMN). Then the transformations can be used to keep consistency between diagrams in these notations. The task of backward transformation is complicated by the fact, that UML-AD does not contain all needed information. This information can be added manually, but in that case the backward transformation will not be automatic.

References

1. Object Management Group. *Business Process Modeling Notation (BPMN).* `http://www.omg.org/technology/documents/br_pm_spec_catalog.htm`, version 1.2, 3 January 2009 .
2. Eriksson, H., Penker, M.. *Business Modeling with UML: Business Patterns at Work.* John Wiley & Sons. ISBN 978-0-471-29551-8, 2000.
3. Object Management Group: *Unified Modeling Language (UML).* `http://www.omg.org/technology/documents/modeling_spec_catalog.htm`, version 2.1.2, 4 November 2007.
4. Jacobson, I., Booch, G., Rumbaugh, J.: *The Unified Software Development Process.* Addison-Wesley Professional. ISBN 020-1-571-692, 1999
5. Rodriguez, A., Fernandez-Medina, E., Piattini, M.: *Analysis-Level Classes from Secure Business Processes Through Model Transformations.* In Trust, Privacy and Security in Digital Business 2007. Springer Berlin / Heidelberg. pp. 104-114. ISBN 978-3-540-744, 2007
6. Rodriguez, A., Fernandez-Medina, E., Piattini, M.: *CIM to PIM Transformation: A Reality.* In Research and Practical Issues of Enterprise Information Systems II. Springer Boston. pp. 1239-1249. ISBN 978-0-387-763, 2008
7. Object Management Group: *MOF Query / Views / Transformations.* version 1.0, April 2008 . `http://www.omg.org/technology/documents/modeling_spec_catalog.htm`,
8. World Wide Web Consortium: *Extensible Markup Language (XML).* version 1.0(fifth edition), 26 November 2008. `http://www.w3.org/XML/`
9. Object Management Group: *XML Metadata Interchange (XMI).* version 2.1.1, 1 December 2007. `http://www.omg.org/technology/documents/modeling_spec_catalog.htm#XMI`,
10. World Wide Web Consortium: *XSL Transformations (XSLT).* version 1.0, 16 November 1999. `http://www.w3.org/TR/xslt`
11. Kovse, J., Härder, T.: *Generic XMI-Based UML Model Transformations.* In Object-Oriented Information Systems 2002. Springer Berlin / Heidelberg. pp. 183-190. ISBN 978-3-540-44087-1, 2002
12. Altova: *UModel 2009.* cite 10.1.2009. `https://shop.altova.com/category.asp?catalog_name=V2008R2C3_shop&category_name=UModel&Page=1`,
13. Object Management Group: *Documents associated with UML Version 2.1.2*, 2006 . `http://www.omg.org/spec/UML/20061001/Superstructure.cmof` .

XML-λ Type System and Data Model Revealed

Pavel Loupal

Dept. of Computer Science and Engineering
Faculty of Electrical Engineering, Czech Technical University
Karlovo nám. 13, 121 35 Praha 2
Czech Republic
loupalp@fel.cvut.cz

Abstract. Within this paper we provide formal description of a functional type system for modeling XML formatted data along with an annotated example of an XML document modeled using such approach. We discuss its advantages and drawbacks in comparison with existing solutions. This submission is a part of our long-term endeavor to propose, examine, and implement an environment for XML data management, especially utilized for XPath/XQuery semantics description.

1 Introduction

The Extensible Markup Language (XML) 1.0 [1] specifies both physical and logical structure of XML documents. For purpose of related specifications (e.g., XPath or DOM) and other XML applications is such low-level description too punctual and hence a number of more abstract *data models* has been proposed. The best known data models nowadays are the XML Information Set (XML Infoset) [4] and the XQuery 1.0 and XPath 2.0 Data Model [7].

This paper describes an alternative data model for XML that forms a part of the XML-λ Framework – a functional framework for modeling, querying and updating XML. The main goal of this work is to provide the reader very intimate knowledge of the Framework and cover all its key concepts and properties. We suppose that the accompanying example presents these facts in reasonable complexity.

Contributions. The main outcome of this paper for the reader should be deep insight into the type system concept and familiarization with the XML-λ data model. We have aimed our endeavor to provide following contributions:

- Formal description of the functional type system for XML.
- Annotated example of an XML document modeled using the XML-λ.
- Informal discussion of Framework's properties and brief comparison with existing data models.
- Outline of the proof-of-the-concept prototype implementation.

Structure. The paper is organized as follows: Section 2 contains formal description of the XML-λ Framework and shows a simple XML document modeled using this approach. General properties, features, and drawbacks of the Framework along with a brief comparison with other data models are discussed in Section 3. Then, in Section 4, we outline our prototype implementation and finally conclude our contribution in Section 5 together with outlook for future work.

Related Work. The main area of interest that is discussed within this paper is the domain of data models for XML. There are two principal data models for XML in use nowadays; the first is the XML Infoset [4] and the second is the XQuery 1.0 and XPath 2.0 Data Model [7]. Both specifications are proposed by the W3C [15] and are employed in interrelated XML standards. Roots of the XML-λ Framework might be found in Pokorný's work [12, 13].

2 The XML-λ Framework

Under the term "framework" we understand both the data model and the query language based on this model. Here, we define the fundamental cornerstone of the framework, the functional type system for XML that is suitable for modeling XML — $\mathcal{T}_E$. Then, in Section 2.3, we show a detailed example of an XML document instance realized using the XML-λ Framework.

2.1 Overview

Our research is significantly influenced by the idea of a functional approach for XML published by Pokorný in [12] and [13]. We found his work very appealing and useful in wide range of applications related to XML. Our main motivation is the belief that it is possible use this framework for expressing semantics of various XML query languages in formally clear and understandable way by using a simply-typed lambda calculus and a general type system. Thus, we set out a goal to express the semantics of XPath and XQuery languages using this approach called XML-λ. We do not aim to propose a new query language for XML but we rather offer an alternative solution for evaluation existing query languages.

Informal Introduction. Figure 1 illustrates the relationship between W3C and XML-λ models. We can observe that the document schema expressed by DTD is in the Framework modeled with two sets; the first one is a set of *element types* available in XML schema and the second one is a set of *element objects* that stores information about document structure. Every document instance then comprises of a set of abstract elements (each is of an element type) and a set of element object instances. Note that these two sets are changing in time, so as the document changes.

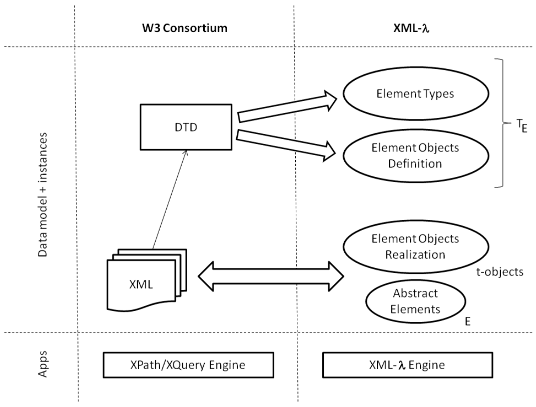


Fig. 1. The relationship between W3C and XML-λ models

2.2 Type System Definition

The main purpose of a *type system* is to prevent the occurrence of execution errors in a program[1] [2]. In this work we understand this definition as checking for correct nesting of XML document structure, using the type system for semantic query validation, and its further evaluation (including optimization).

But there is a unifying cornerstone for all type systems — each *type system* has a finite set of rules for type construction and set of respective types defined within this system. Following sections describe step-by-step process of definition of a functional type system for XML. We begin with a general functional type system and extend it with support for regular types. Finally, we propose the type system $\mathcal{T}_E$ that forms the core formalism in the XML-λ Framework.

Functional Type System First, we define a general functional type system $\mathcal{T}$. This definition basically follows the common approach presented for example in [16, p.143].

Definition 1 (Type system $\mathcal{T}$). *Let $\mathcal{B}$ is a set of (atomic) types $S_1 \dots S_n$, $n \geq 1$. Type System $\mathcal{T}$ over $\mathcal{B}$ is obtained by the grammar*

$$\begin{aligned} T := {} & S && \textit{primitive type} \\ & \mid (T_1 \rightarrow T_2) && \textit{functional type} \\ & \mid (T_1, \dots, T_k) && \textit{tuple type} \\ & \mid (T_1 + \dots + T_k) && \textit{union type} \end{aligned}$$

where $S \in \mathcal{B}$.

Presuming the members of $\mathcal{B}$ being non-empty disjoint sets, then the type $(T_1 \rightarrow T_2)$ means a set of all (both total and partial) functions from T_1 into T_2.

[1] In the context of this work, we understand by the term "program" an XML query we aim to evaluate.

$(T_1, \ldots, T_n)$ denotes the Cartesian product $(T_1 \times \ldots \times T_n)$ and $(T_1 + \ldots + T_n)$ denotes the union $T_1 \cup \ldots \cup T_n$. We denote $o : T$ an object o of type T (also called "T-object").

Regular Type System Subsequently, we define a regular type system $\mathcal{T}_{reg}$ that extends the type system $\mathcal{T}$ with regular constructs. For this definition, we employ the set $Bool \equiv \{TRUE, FALSE\}$ with its common Boolean semantics. Next, let us suppose a set $Name$ that contains all possible element names allowed by the XML grammar [1].

Definition 2 (Type System $\mathcal{T}_{reg}$). *Let $\mathcal{B} = \{String, Bool, Name\}$, $\boldsymbol{tag} \in Name$. The type system $\mathcal{T}_{reg}$ over $\mathcal{B}$ is defined as follows.*

$$\begin{array}{lll} T := & \boldsymbol{tag} : String \mid \boldsymbol{tag} : & \textit{elementary regular expression} \\ & \mid T* & \textit{zero or more (Kleene closure)} \\ & \mid T+ & \textit{one or more (positive closure)} \\ & \mid T? & \textit{zero or one} \\ & & \textit{where } T \textit{ is an alternative or elementary regular expression.} \\ & \mid (T_1 \mid T_2) & \textit{alternative} \end{array}$$

Upon this type system we can define a type system for XML denoted $\mathcal{T}_E$ as follows.

$\mathcal{T}_E$ — The Type System for XML The last step for obtaining a type system suitable for modeling XML data is a slight alternation of $\mathcal{T}_{reg}$ aiming to support data types available from DTD. For this purpose, we have to take a closer look at DTD properties. Due to the fact that we consider only typed XML documents (i.e. always with an attached DTD) we can distinguish the type of each particular XML element in a document. It is also obvious that in a document there can exist two different elements with the same tag and content, therefore we have to be able to treat them as distinct instances. Therefore, terms *abstract element* and *element object* (or *T-object*) were introduced.

Definition 3 (Abstract Element). *For each XML element there exists exactly one* abstract element*. The (infinite) set of abstract elements is denoted as E.*

Definition 4 (Element Object). *An element object of type T, denoted as T-object, is a partial function of type $E \rightarrow T_{rng}$ where T_{rng} is an* element type *or String.*

An arbitrary XML element is then modeled as a mapping from one particular *abstract element* to its respective codomain (whose type is determined by corresponding *element type*). Hence, by application of an T-object to this *abstract element* we may obtain either a character string (typed as PCDATA in DTD) or a more complex structure regarding to the type system.

Definition 5 (Type System $\mathcal{T}_E$). *Let $\mathcal{T}_{reg}$ over $\mathcal{B}$ be the type system from Definition 2 and E be the set of abstract elements. Then the type system $\mathcal{T}_E$ induced by $\mathcal{T}_{reg}$ is defined as follows.*

$$\begin{array}{lll}
E := \boldsymbol{TAG} : T \mid \boldsymbol{TAG} : & & \textit{elementary element types} \\
\quad\quad \textit{where } \boldsymbol{tag} : T \textit{ and } \boldsymbol{tag} : & & \textit{are elementary regular expressions over } \mathcal{B}. \\
\quad \mid E* & & \textit{zero or more (Kleene closure)} \\
\quad \mid E+ & & \textit{one or more (positive closure)} \\
\quad \mid E? & & \textit{zero or one} \\
\quad \mid (E_1 \mid E_2) & & \textit{alternative} \\
\quad \mid \boldsymbol{TAG} : (E_1, \ldots, E_k) & & \\
\quad\quad \textit{where } \boldsymbol{tag} \in Name & & \textit{complex element types}
\end{array}$$

Elementary element types and complex element types are hereafter denoted as *element types*. For example, with respect to the DTD shown in Figure 2, we write respective *element types* as:

```
BIB : BOOK*
BOOK : (TITLE, (AUTHOR+ | EDITOR+), PUBLISHER, PRICE)
AUTHOR : (LAST, FIRST)
EDITOR : (LAST, FIRST, AFFILIATION)
TITLE : String
LAST : String
FIRST : String
AFFILIATION : String
PUBLISHER : String
PRICE : String
```

Due to constraints given by DTD, there exists exactly one content model for each XML element (see the "local tree languages" in [11]). Therefore, we can omit the part beyond the colon because it cannot lead to any misunderstanding; i.e., we may use a shorter notation and write `AUTHOR` instead of `AUTHOR : (LAST, FIRST)`.

For convenience, we also define a *nullary function*. It is a useful construct that allows us to access abstract elements and filter them by corresponding element type. Later on, we will utilize it in the query language.

Definition 6 (Nullary function). *A T-nullary function returns the domain of a T-object, where T denotes an element type.*

Note that the set E of all abstract elements is thus split by these functions into a number of disjoint subsets for all respective *element types*.

Attributes. Until now we have discussed only elements, their structural properties, and their content. Other important data in XML are elements' attributes. In the XML-λ Framework we use the same way for accessing them as for elements; we model attributes as functions too. For example, we model the `year` attribute of the book element as a partial function $YEAR : E \rightarrow String$ whereas its domain is $E_{BOOK} \subseteq E$ such that $BOOK$-nullary function is there defined (thus, this function is defined only for elements that have associated attributes with this name).

2.3 Data Model Definition

A *data model* is an abstract model that describes how data is represented and accessed. In our work we employ simple and functional types defined by the $\mathcal{T}_E$ type system (see Section 2.2). Apparently, in the XML-λ Framework the main entity to be described is the XML document. We can see the instance of an XML document (which is basically a valuation of a type from $\mathcal{T}_E$) as a structure containing

– set of abstract elements, denoted E_{doc} (subset of the infinite set E),
– set of T-objects,
– set of all text content of document's elements and attributes.

Note that for simplification we consider character strings typed as *String* for all textual content instead of more accurate definitions of PCDATA and CDATA proposed in [1].

A Data Model Example This section provides an example of a type system definition and realization of an XML document instance in the XML-λ Framework. Let us consider a sample DTD[2] shown in Figure 2 and a respective XML document in Figure 3.

```
<!ELEMENT bib  (book*)>
<!ELEMENT book  (title,  (author+ | editor+), publisher, price)>
<!ATTLIST book  year CDATA  #REQUIRED >
<!ELEMENT author  (last, first)>
<!ELEMENT editor  (last, first, affiliation)>
<!ELEMENT title  (#PCDATA)>
<!ELEMENT last  (#PCDATA)>
<!ELEMENT first  (#PCDATA)>
<!ELEMENT affiliation  (#PCDATA)>
<!ELEMENT publisher  (#PCDATA)>
<!ELEMENT price  (#PCDATA)>
```

Fig. 2. An example XML schema — biblio.dtd

[2] This DTD is taken from the "XMP" Use Case published in the XML Query Use Cases [3] specification

```
<?xml version="1.0"?>
<!DOCTYPE bib SYSTEM "biblio.dtd">
<bib>
  <book year="2008">
    <title> XML technologie </title>
    <author>
      <last> Pokorny </last>
      <first> Jaroslav </first>
    </author>
    <author>
      <last> Richta </last>
      <first> Karel </first>
    </author>
    <publisher> Grada </publisher>
    <price> 286.00 </price>
  </book>
</bib>
```

Fig. 3. A well-formed and valid XML document

For given schema we obtain (as illustrated in Section 5) following set of both elementary and complex element types: $\{BIB$, $BOOK$, $AUTHOR$, $EDITOR$, $TITLE$, $LAST$, $FIRST$, $AFFILIATION$, $PUBLISHER$, $PRICE\}$. The set of abstract elements, E_{doc}, contains for the XML document in Figure 3 eleven items: $E_{doc} = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}\}$. The set of element objects with corresponding mappings is illustrated in Table 1.

In this scenario, for instance, the *title*-element object (a function of type $E \rightarrow String$) is defined exactly for one abstract element (the one that serves for modeling the `title` element) and for this abstract element it returns value "XML technologie".

In a more complex case, function *book* of type $BOOK : E \rightarrow E \times 2^E \times E \times E$, applied to an arbitrary abstract element from its domain, returns a quadruple — subset of this Cartesian product. Within the tuple, we can then access its particular components by performing name-based projections (more precisely, "element type name"-based projections).

3 Framework Features Summary

The previous section describes formally the XML-λ Framework along with a simple example. In this part of the contribution, we discuss various aspects of the Framework, its attributes, features, weaknesses, and drawbacks. We have selected nine topics that are, as for us, important for comprehension, usage, and comparison of existing data models for XML. We present such brief comparison with the XML Infoset [4] and the XQuery 1.0 and XPath 2.0 Data Model [7].

Function	Function domain/range	Mappings
bib	$BIB : E \rightarrow 2^E$	$bib(e_1) = \{e_2\}$
book	$BOOK : E \rightarrow E \times 2^E \times E \times E$	$book(e_2) = (e_3, \{e_4, e_7\}, e_{10}, e_{11})$
book_year	$YEAR : E \rightarrow String$	$book\_year(e_2) =$ "2008"
title	$TITLE : E \rightarrow String$	$title(e_3) =$ "*XML technologie*"
author	$AUTHOR : E \rightarrow E \times E$	$author(e_4) = (e_5, e_6)$, $author(e_7) = (e_8, e_9)$
editor	$EDITOR : E \rightarrow E \times E \times E$	none
last	$LAST : E \rightarrow String$	$last(e_5) =$ "*Pokorny*", $last(e_8) =$ "*Richta*"
first	$FIRST : E \rightarrow String$	$first(e_6) =$ "*Jaroslav*", $first(e_9) =$ "*Karel*"
affiliation	$AFFILIATION : E \rightarrow String$	none
publisher	$PUBLISHER : E \rightarrow String$	$publisher(e_{10}) =$ "*Grada*"
price	$PRICE : E \rightarrow String$	$price(e_{11}) =$ "286.00"

Table 1. Set of T-objects induced by the running example

Following paragraphs discuss each topic one-by-one and this text is later summarized[3] in Table 2.

1.Functional Approach. One of the main forces at the rise of XML-λ was the aim to provide an alternative to existing W3C proposal. Therefore the Framework is strictly tied to the concept of functions; it uses both a functional type system and a query language based on simply typed lambda calculus.

2.Multiple Data Models. The XML-λ Framework is presented here as a tool for modeling and querying (and updating) XML formatted data typed by the DTD. Considering the properties of the relational model we can also provide a relational type system within the Framework and work with relational data, or even pursue heterogeneous data transformation and integration between XML and relational data sources.

In addition, although the DTD is still a relatively sufficient solution for most real XML applications we claim that we can provide a type system that is based on W3C's XML Schema [5]. This extension of the Framework will be available in our consecutive work. Yet there are (at least for now) some formal difficulties and therefore we expect that we will be able to support only a part of the specification.

3.Full XML Coverage. Within the Framework we can access and modify only elements and attributes. With regard to the XML Infoset Data Model [4] there may occur other information items in an XML document — e.g., comments, processing instructions, etc. At the present time we do not plan to extend this model but it is certainly possible to enrich it with other sorts of such items.

[3] Y/N/? stands for Yes/No/Unknown; feature **is** supported, **is not** supported, or is **not examined** yet.

4.Uniform Data Model Access. The XML-λ data model does not strongly distinguish between elements and attributes. We model them with the same approach — by functions; and access their content by evaluating these functions independently on their "origin".

5.Element Ordering. Formal foundations of the Framework stand on the set theory. For its utilization in the world of XML we have to employ a kind of ordering among elements. The concept known as "document order" [6, Section 2.4] is realized in the Framework as a partial function $f : E \rightarrow Integer$ that assigns to each abstract element an unique number according to this specification.

6.Mixed Content Model. This is probably the weakest spot of the Framework. With respect to proposed type system it cannot handle data with mixed content model [1]. This is still an issue that must be resolved as soon as possible since it disables the usage of the Framework for document-centric XML data.

7.Mandatory Typing. Another questionable feature is the necessity of an XML schema existence for all XML data processed by the XML-λ Framework. It is a natural requirement with respect to the fact that the type system is built upon this schema. Moreover, for vast majority of XML data available on the web such schema does exist and is used [10].

8.Support for Recursive Types. XML data may also contain some recursive content, even though in small amount only [10]. However, all three data models we mention here can settle this situation.

9.Performance. This aspect is one of the most important data model characteristics for its production deployment. We have carried out some preliminary XPath 1.0 benchmarks with promising results in [14] but we consider this early evaluation as superficial and do not publish and discuss details here. With no doubt, it is one of important issues that have to be furthermore examined.

Feature	**XML-λ**	**Infoset**	**XDM**
1. Functional Approach	Y	N	N
2. Multiple Data Models	Y	N	N
3. Full XML Coverage	N	Y	Y
4. Uniform Data Model Access	Y	N	N
5. Element Ordering	Y	N	Y
6. Mixed Content Model	N	Y	Y
7. Mandatory Typing	Y	N	N
8. Support for Recursive Types	Y	Y	Y
9. Performance	?	?	?

Table 2. The comparison of selected data models

4 Prototype Implementation Overview

We already have a Java prototype implementation available. The main part is a core library that realizes the type system and data model. Existing applications, an XPath 1.0 Processor and the ExDB [9] database management system, are built upon this library. The internal structure of this component is illustrated in the UML class diagram in Figure 4.

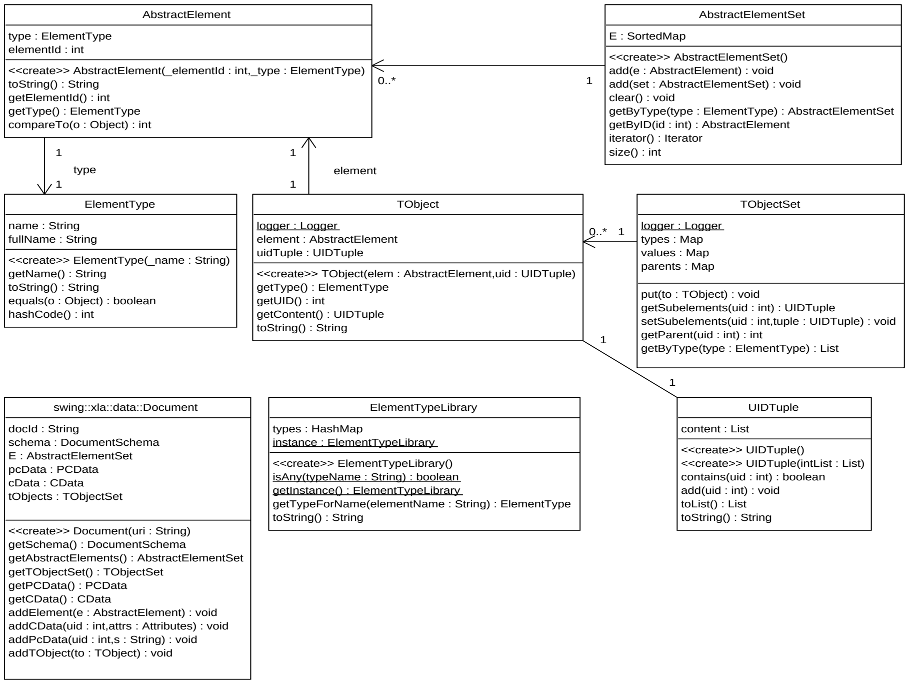

AbstractElement
type : ElementType
elementId : int
<<create>> AbstractElement(_elementId : int,_type : ElementType)
toString() : String
getElementId() : int
getType() : ElementType
compareTo(o : Object) : int
AbstractElementSet
E : SortedMap
<<create>> AbstractElementSet()
add(e : AbstractElement) : void
add(set : AbstractElementSet) : void
clear() : void
getByType(type : ElementType) : AbstractElementSet
getByID(id : int) : AbstractElement
iterator() : Iterator
size() : int
0..*
1
type
element
ElementType
name : String
fullName : String
<<create>> ElementType(_name : String)
getName() : String
toString() : String
equals(o : Object) : boolean
hashCode() : int
TObject
logger : Logger
element : AbstractElement
uidTuple : UIDTuple
<<create>> TObject(elem : AbstractElement,uid : UIDTuple)
getType() : ElementType
getUID() : int
getContent() : UIDTuple
toString() : String
TObjectSet
logger : Logger
types : Map
values : Map
parents : Map
put(to : TObject) : void
getSubelements(uid : int) : UIDTuple
setSubelements(uid : int,tuple : UIDTuple) : void
getParent(uid : int) : int
getByType(type : ElementType) : List
swing::xla::data::Document
docId : String
schema : DocumentSchema
E : AbstractElementSet
pcData : PCData
cData : CData
tObjects : TObjectSet
<<create>> Document(uri : String)
getSchema() : DocumentSchema
getAbstractElements() : AbstractElementSet
getTObjectSet() : TObjectSet
getPCData() : PCData
getCData() : CData
addElement(e : AbstractElement) : void
addCData(uid : int,attrs : Attributes) : void
addPcData(uid : int,s : String) : void
addTObject(to : TObject) : void
ElementTypeLibrary
types : HashMap
instance : ElementTypeLibrary
<<create>> ElementTypeLibrary()
isAny(typeName : String) : boolean
getInstance() : ElementTypeLibrary
getTypeForName(elementName : String) : ElementType
toString() : String
UIDTuple
content : List
<<create>> UIDTuple()
<<create>> UIDTuple(intList : List)
contains(uid : int) : boolean
add(uid : int) : void
toList() : List
toString() : String

Fig. 4. The main classes in the `swing.xla.core` package

The core package comprises of approximately 20 classes that provide basic functionality as a SAX handler, `XMLOutputFormatter` and other classes that cover the essential functionality.

Our preliminary benchmark of the XPath 1.0 processor based upon this library is ready to be published in [14]. This work contains a comparison with state-of-the-art XPath processors inspired by the XPathMark test [8]. We consider the results as promising but the test did detect a serious performance leak in XML document parsing; in contrast to the others, the process of memory allocation and internal data model structures building behaves exponentially instead of linearly as expected. It is an implementation issue that must be fixed in order to catch up with the rivals.

5 Conclusion

We have published a detailed description of the XML-λ's type system based on the Document Type Definition along with a data model suitable for description of XML formatted data. This model is subsequently demonstrated on a simple example where we describe the concept and utilization of the Framework. Further, we have discussed various issues and features of our approach. We can observe that some of the features are interesting for more detailed examination, e.g. the elegance of the functional approach combined with a reasonable performance of our prototype implementation. Despite of few remaining issues, we still believe that the idea of a functional approach we have presented is interesting, interim results are promising, and our work on this topic will bring meaningful accomplishments.

Future Work. So far, we have proposed a theoretical base for modeling and querying XML. As shown in this paper there are still issues to be solved, in particular the ability to treat mixed element content and developments with the prototype implementation. We find these topics apparently as the most important for the moment.

Acknowledgments. This research has been partially supported by the grant of The Czech Science Foundation (GAČR) No. GA201/09/0990.

References

1. 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.
2. L. Cardelli. *Handbook of Computer Science and Engineering*, chapter 103: Type Systems. Digital Equipment Corporation, 1997.
3. D. Chamberlin, P. Fankhauser, D. Florescu, M. Marchiori, and J. Robie. XML Query Use Cases, March 2007. http://www.w3.org/TR/2007/NOTE-xquery-use-cases-20070323/.
4. J. Cowan and R. Tobin. XML information set (second edition), April 2004. http://www.w3.org/TR/2004/REC-xml-infoset-20040204/.
5. D. C. Fallside, P. Walmsley, H. S. Thompson, D. Beech, M. Maloney, N. Mendelsohn, P. V. Biron, and A. Malhotra. XML Schema 1.0, October 2004. http://www.w3.org/XML/Schema.
6. M. Fernández, A. Malhotra, J. Marsh, M. Nagy, and N. Walsh. XQuery 1.0 and XPath 2.0 Data Model, September 2005. http://www.w3.org/TR/xpath-datamodel/.
7. M. Fernández, A. Malhotra, J. Marsh, M. Nagy, and N. Walsh. XQuery 1.0 and XPath 2.0 Data Model, January 2007. http://www.w3.org/TR/2007/REC-xpath-datamodel-20070123/.
8. M. Franceschet. XPathMark: An XPath Benchmark for the XMark Generated Data. In S. Bressan, S. Ceri, E. Hunt, Z. G. Ives, Z. Bellahsene, M. Rys, and R. Unland, editors, *XSym*, volume 3671 of *Lecture Notes in Computer Science*, pages 129–143. Springer, 2005.

9. P. Loupal. Experimental DataBase (ExDB) Project Homepage. http://swing.felk.cvut.cz/~loupalp.
10. I. Mlýnková, K. Toman, and J. Pokorný. Statistical Analysis of Real XML Data Collections. In *COMAD '06: Proceedings of the 13th International Conference on Management of Data*, pages 20 – 31. Tata McGraw-Hill Publishing Co. Ltd., December 2006.
11. M. Murata, D. Lee, M. Mani, and K. Kawaguchi. Taxonomy of XML schema languages using formal language theory. *ACM Trans. Internet Techn.*, 5(4):660–704, 2005.
12. J. Pokorný. XML functionally. In B. C. Desai, Y. Kioki, and M. Toyama, editors, *Proceedings of IDEAS2000*, pages 266–274. IEEE Comp. Society, 2000.
13. J. Pokorný. XML-λ: an extendible framework for manipulating XML data. In *Proceedings of BIS 2002*, pages 160–168, Poznan, 2002.
14. J. Stoklasa and P. Loupal. Benchmarking a lambda calculus based XPath processor. Yet unpublished.
15. The World Wide Web Consortium. W3C Homepage. http://www.w3.org.
16. J. Zlatuška. *Lambda-kalkul.* Masarykova univerzita, Brno, Česká republika, 1993.

Combination of TA- and MD-algorithm for Efficient Solving of Top-K Problem according to User's Preferences

Matúš Ondreička and Jaroslav Pokorný

Department of Software Engineering, Faculty of Mathematics and Physics, Charles University, Malostranské nám. 25, 118 00 Praha 1, Czech Republic
`matus.ondreicka@mff.cuni.cz,jaroslav.pokorny@mff.cuni.cz`

Abstract. In this article we focus on efficient solving of searching the best K objects in more attributes according to user's preferences. Local preferences are modelled with one of four types of fuzzy function. Global preferences are modelled concurrently with an aggregation function. We focused on searching the best K objects according to various user's preferences without accessing all objects. Therefore we deal with the use of TA-algorithm and MD-algorithm. Because of local preferences we used B^+-trees during computing of Fagin's TA-algorithm. For searching the best K objects MD-algorithm uses multidimensional B-tree, which is also composed of B^+-trees. We developed an MXT-algorithm and a new data structure, in which MXT-algorithm can effectively find the best K objects by user's preferences without accessing all the objects. We show that MXT-algorithm in some cases achieves better results in the number of accessed objects than TA-algorithm and MD-algorithm.

1 Introduction and Motivation

Nowdays, huge amounts of data in various web systems grow exponentially. Users try to find various objects in this data, such as laptops, cars, apartments, holidays, etc. These objects have various attributes. According to the values of attributes, users search for the best objects for them. However, each user prefers objects with other values of attributes [7][6].

Most users look for only a few objects that are best suited to their preferences. Therefore, it is advantageous according to the user's preferences directly to find the best K objects. Making it possible to find the best K objects from the set of objects X, it must be possible to judge which objects are better or worse for a user U. Every user prefers objects with own preferences, which are modelled locally by means of a fuzzy function and globally by means of an aggregation function [9]. In this paper we assume that the set of objects of the same type is stored in one data structure. The problem of searching the best K objects according to values of different attributes in the same time is indicated as a *top-K problem* [4][7].

In the next, text we describle using B^+-tree [2] for sorting objects acording to fuzzy function for supporting the local preferences [1][3][10].

We deal also with methods and data structures for effective solution of top-K problem [1]. We discuss the problem of finding the best K objects without accessing all the objects via Fagin's TA-algorithm [4] and MD-algorithm [1]. For application of local preferences TA- and MD-algorithm are using data structures based on B$^+$-trees [1]. These structures are independent from user's preferences. Moreover, it is possible to update these structures easily and quickly.

We developed a MXT-algorithm and a new data structure based on B$^+$-trees, in which MXT-algorithm can effectively find the best K objects by user's preferences without accessing all the objects. We will show advantages of the new MXT-algorithm and a comparison with the results of other algorithms. We show that MXT-algorithm in some cases achieves better results in the number of accessed objects than TA- and MD-algorithm.

A background concerning modelling user preferences is contained in Section 2. Section 3 presents use of B$^+$-tree for approaching the top-K problem. Sections 4 and 5 are devoted to explaining principles of Fagin's TA-algorithm and MD-algorithm. In Section 6 we describe our new MXT-algorithm and new data structure, which is used during computation of MXT-algorithm. Section 7 presents the results of the tests and compares MD-algorithm with Fagin's algorithms. Finally, Section 8 provides some suggestions for a future work.

2 User's preferences

In this article, we suppose a set of objects X with m attributes $A_1, ..., A_m$. Every object $x \in X$ has m values $a_i^x, ..., a_m^x$ of corresponding attributes. When searching for the best K objects, a user U chooses user's preferences, which determine suitability of the object $x \in X$ in dependence with its m values of attributes $a_1^x, ..., a_m^x$ for user U. In accordance to user U preferences, it is possible to find the best K object for user U. We are using a model of user preferences, which is based on concatenation of local preferences and global preferences. At first, user U decides how he prefers objects according to each of attributes A_i, $i = 1, ..., m$, and then chooses a relationship between these attributes.

Local preferences reflect how the object is preferred according to only one attribute. In this work, we used a mapping f_i: $dom(A_i) \rightarrow [0, 1]$. In general, we differentiate two possible attribute types.

Nominal attribute has a finite range of possible values, usually strings. For a nominal attribute A_i user U has to rate each value of the attribute. For example, it is mark of some products. Local preference of user U for the nominal attribute A_i is represented as mapping $f_i^U(x)$: $a_i^x \rightarrow [0, 1]$, where $i = 1, ..., m$.

Ordinal attribute has some natural value ordering, other than lexical ordering. Typical examples are integer numbers. Domain of ordinal attribute is subset of continuous interval. The local preference of user U for the ordinal attribute A_i is represented by *fuzzy function* [9]. We denoted it also as $f_i^U(x)$ the user's fuzzy function for i-th attribute. In this case, fuzzy functions have a scheme $f_i^U(x)$: $a_i^x \rightarrow [0, 1]$, where $i = 1, ..., m$.

For the worst object $x \in X$, $f_i^U(x) = 0$ holds and for the best one, $f_i^U(x) = 1$. Comparing $f_i^U(x_1)$ and $f_i^U(x_2)$ of two objects x_1 and x_2, it is possible to decide which of them is suitable for the user U according to the value of i-th attribute. When $f_i^U(x_1) > f_i^U(x_2)$, then x_1 is more suitable. When $f_i^U(x_1) = f_i^U(x_2)$, then x_1 and x_2 are equally suitable.

Article [10] describes four types of user's fuzzy functions, which are sufficient for entering user's preference of ordinal attributes. It is showed in Figure 1. There are nondecreasing function, nonincreasing function, function in the form of hill, and function in the form of valley. In next text we suppose only these four types of user's fuzzy functions.

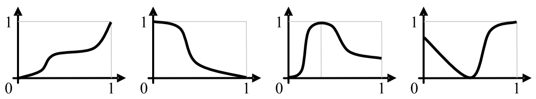


Fig. 1. Four used types of fuzzy functions: nondecreasing, nonincreasing, hill, valley.

Global preferences express how the user U prefers objects from X according to all attributes $A_1, ..., A_m$. On the set of objects X, we consider aggregation function @ with m variables $p_1, ..., p_m$ specified as $@(p_1, ..., p_m) : [0,1]^m \rightarrow [0,1]$. For the user U with his user fuzzy functions $f_1^U, ..., f_m^U$, a *user rating function* $@^U$ originates by means of substitution of $p_i = f_i^U(x)$, $i = 1, ..., m$. Then, for every $x \in X$ $@^U(x) = @(f_1^U(x), ..., f_m^U(x))$. With $@^U(x)$ it is possible to evaluate global rating of each object $x \in X$.

With aggregation function, a user U can define the mutual relations of the attributes. For example, we can use *arithmetic average*. For implementation of user's influence to the aggregate function, it is possible to use *weighted average*, where weights $u_1, ..., u_m$ of single attributes $A_1, ..., A_m$ determine how the user prefers single attributes, $@(x) = \frac{u_1 \cdot p_1 + ... + u_m \cdot p_m}{u_1 + ... + u_m}$. When the user does not care about i-th attribute A_i when rating the objects, he can then put 0 into the aggregate function, i.e. $u_i = 0$.

In this work, every user U express his preference as functions $f_1^U, ..., f_m^U$ and $@^U$. Every object $x \in X$ has its own special global rating $@^U(x)$, a measure of pertinence for the user U. In this way, it is possible to find the best K objects for the user U. When there are more objects with the same rating as rating of the best K-th object, a random object is chosen.

3 Usage of B$^+$-tree

For application of local user's preferences we need a data structure, from which it is possible to obtain objects according to user's fuzzy function f^U in descending order according to f^U. Let the objects of the set X be indexed in the B$^+$-tree [2] according to the values of attribute A [1][3][10].

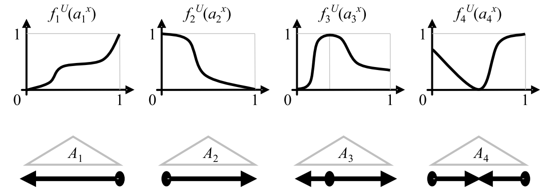


Fig. 2. Ordering interpretation of four used types of fuzzy functions.

Every object $x \in X$ is indexed by the key $k \in dom(A)$, a value of which is equal to a value of attribute a^x of object x, i.e. $k = a^x$. For a key k, in general, more objects with the same A attribute value can exist in X. Therefore, we use the *object array*, in which are stored objects with the same value of the key k. For every key k there is a pointer to corresponding object array.

Since the leaf nodes of the B$^+$-tree are linked in two directions [2], it is possible to cross the B$^+$-tree through the leaf level and to get all the keys. According to course of user's fuzzy function f^U, which is one of four types [10], we can obtain objects in the following different ways (Figure 2).

1. *Nondecreasing function.* In this case, we have to cross the leaf level of the B$^+$-tree from the right to the left. It is possible to get the pairs $(x, f^U(x))$ in the descending order according to the user's preference f^U, because $a^x \leq a^y \Rightarrow f^U(x) \leq f^U(y)$ holds.
2. *Nonincreasing function.* In this case, we have to cross the leaf level of the B$^+$-tree from the left to the right. It is possible to get the pairs $(x, f^U(x))$ in the descending order according to the user's preference f^U, because $a^x \geq a^y \Rightarrow f^U(x) \leq f^U(y)$ holds.
3. *Function in the form of hill.* In this case, we must find key k in leaf level of B$^+$-tree. Key k is the nearest point from maximum of fuzzy function f^U. From the key k we cross the leaf level in two ways concurrently from k to both borders of the B$^+$-tree as in previous two cases.
4. *Function in the form of valley.* In this case, we have to cross the leaf level of the B$^+$-tree in two ways concurrently from both borders to the key k in to inside of leaf level. Key k is the nearest point from minimum of f^U.

In general, f^U cannot be monotone in its domain. In this case, the domain can be divided into continuous intervals (also leaf level of B$^+$-tree), where f^U is monotone on each of these intervals. From the intervals, objects are obtained concurrently according to f^U as well as for monotone fuzzy function [1].

4 Fagin's TA-algorithm

Fagin et al. describe in [4] top-K algorithms, which solve top-K problem without searching of all objects in set X. Algoritms assume that the objects from the set X are, together with values of their m attributes $A_1, ..., A_m$, stored in m lists

$L_1, ..., L_m$. Each i-th list L_i contains pairs (x, a_i^x). Lists $L_1, ..., L_m$ are sorted in descending order according to the values of attributes $a_1^x, ..., a_m^x$ of objects $x \in X$. The aggregate function @ must be monotone according to the ordering in lists [4][1], e.g. weighted average. Then Fagin's algorithms can find the best K objects with regard to existing aggregate function @ without making $|X| \cdot m$ accesses into the lists $L_1, ..., L_m$ [4].

Fagin's algorithm TA (threshold algorithm) needs to access the lists L_1, ..., L_m *sequentially* and also *directly*. With the sequential access, TA searches the lists step by step from the top to the bottom and obtains pairs (x, a_i^x). For every object x, which is detected for the first time in obtained pair (x, a_i^x), TA obtains the missing attribute values of the object x by a direct access to the other lists.

TA uses the list T_K, in which it keeps the best actual K objects ordered according to their value of $@(x)$. During the run of TA a threshold T_h is counted, which is calculated by means of inserting the last seen values of attributes $a_1^{last}, ..., a_m^{last}$ in the lists $L_1, ..., L_m$, respectively, at the sequential access to the aggregate function $T_h = @(a_1^{last}, ..., a_m^{last})$. Rating of the K-th best object in T_K we denote M_K. When $T_h \leq M_K$, TA is able to stop and returns T_K as the list of the best K objects. TA can finish before it comes to the end of all the lists [4]. TA-algorithm is described by the following pseudo-code.

```
Input: Lists $L_1, ..., L_m$, Aggregation @, int $K$;
Output: List TK;
var List TK;                {temporary list of objects}
begin
   while(|TK| < $K$ or $T_h$ > $M_K$)do
      $(x, a_i^x)$ = getNextPair($L_1, ..., L_m$);   {it obtains next pair from
      $a_i^{last}$ = $a_i^x$;                          one of the lists $L_1, ..., L_m$} [1][5]
      $T_h$ = $@(a_1^{last}, ..., a_m^{last})$;
      if($x \notin$ TK)then
         get the missing attribute values of the object $x$;
         if(|TK| < $K$)then
            insert object $x$ to the list TK on the right place according to $@(x)$;
         else
            if($@(x)$ > $M_K$)then
            begin
                delete $K$-th object from the list TK;
                insert object $x$ to the list TK on the right place by $@(x)$;
            end;
   endwhile;
end.
```

4.1 Application of local preferences

Original Fagin's algorithms offer the possibility of rating objects only globally with an aggregate function @ and to find the best K object for the user U only according to his global preference. For the support of the local preferences, it is necessary that every i-th list L_i contains pairs $(x, f_i^U(x))$ in descending order according to user's fuzzy function $f_i^U(x)$.

For application of local user's preferences, we use B$^+$-tree (see Section 3). We use as the lists $L_1, ..., L_m$ a list of m B$^+$-trees $B_1, ..., B_m$, where in B$^+$-tree B_i all objects are indexed by values of i-th attribute A_i. Moreover, this structure is common for all users. A particular user U only specifies his local preferences with fuzzy functions $f_1^U, ..., f_m^U$ and global preferences with function $@^U$. Then the algorithm sequentially obtains pairs $(x, f_i^U(x))$ from B$^+$-trees $B_1, ..., B_m$ according to preferences of user U [1].

Algorithm TA also uses the direct access to the lists $L_1, ..., L_m$, where for object x it is needed to obtain its unknown value a_i^x from L_i. B$^+$-tree is not able to make this operation, because it is not possible to obtain the value directly. For a realization of direct access we can use, for example, an associative array.

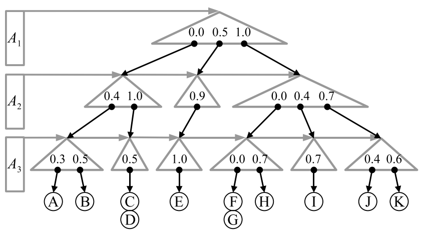
5 Multidimensional B-tree and MD-algorithm

In [1] we describe a top-K algorithm, which solves top-K problem with using the multidimensional B-tree [8] (*MDB-tree*) without searching all the objects. MDB-tree with m levels allows to index set of objects X by attributes A_1, ..., A_m. Each level of MDB-tree is composed from B$^+$-trees containing key values from domain of one attribute. It is showed in Figure 3. Ordering of attributes in levels of MDB-tree is very important [1] (see Section 6).

We use the mark $\rho(k_i)$ for the pointer of the key k in B$^+$-tree in i-th level of MDB-tree. If $i < m$, then $\rho(k_i)$ refers to B$^+$-tree in $(i+1)$-th level of MDB-tree. If $i = m$, i.e. B$^+$-tree is in the last level of MDB-tree, then $\rho(k_i)$ refers to *object array*, where objects with the same values of all the m attributes are stored.

For explicit identification of B$^+$-tree in MDB-tree, we use the sequence of keys, which is called *tree identifier* [1]. An identifier of B$^+$-tree in i-th level is $(k_1, ..., k_{i-1})$. Tree identifier $(\emptyset)$ identifies B$^+$-tree in the first level of MDB-tree.

Furthermore, in MDB-tree we use the *best rating* $B(S)$ of B$^+$-tree S [1]. By the best rating $B(S)$ of B$^+$-tree S with identifier $(k_1, ..., k_{i-1})$ in the i-th level of MDB-tree, we understand a rating of not yet known object x, calculated with $@(a_1^x, ..., a_m^x)$, where the first $i-1$ attribute's values of object x are $k_1, ..., k_{i-1}$ and values of other attributes are 1, i. e. $B(S) = @(k_1, ..., k_{i-1}, 1, ..., 1)$.



	A_1	A_2	A_3
A	0.0	0.4	0.3
B	0.0	0.4	0.5
C	0.0	1.0	0.5
D	0.0	1.0	0.5
E	0.5	0.9	1.0
F	1.0	0.0	0.0
G	1.0	0.0	0.7
H	1.0	0.0	0.7
I	1.0	0.4	0.7
J	1.0	0.7	0.4
K	1.0	0.7	0.6

Fig. 3. Set of eleven objects with values of three attributes stored in MDB-tree.

5.1 MD-algorithm

Obtaining all the objects from MDB-tree is possible with a recursive procedure, which searches MDB-tree in depth-first search. The work [1] includes a sequence of statements with proofs, which shows that it is possible to find best K objects in MDB-tree with the recursive procedure `findTopK` according to a monotone aggregate function @ and without getting all the objects.

Let the keys from the B$^+$-tree S with the identifier $(k_1, ..., k_{i-1})$ be the keys obtained one by one by a run of procedure `findTopK`. The pointer $\rho(p)$ refers to B$^+$-tree P in the next level or to the object array P. Let M_K be the rating of the K-th best object in temporary list T_K of best K objects. If $B(P) \leq M_K$ holds, then the procedure `findTopK(`*MDB-tree*`,` $(k_1, ..., k_{i-1})$`)` can stop in S.

The following pseudo-code describes MD-algorithm.

```
Input: MDBtree MDB-tree, Aggregation @, int $K$;
Output: List TK;
var List TK;            {temporary list of objects}
begin
   findTopK(MDB-tree, $(\emptyset)$, @, $K$);  return TK;
end.

procedure findTopK(MDBtree MDB-tree, Identifier $(k_1, ..., k_{i-1})$,
                   Aggregation @, int $K$);
   while(there is next key in $B^+$-tree)do   {with identifier $(k_1, ..., k_{i-1})$}
      $k_i$ = getNextKey(MDB-tree, $(k_1, ..., k_{i-1})$);
            {$\rho(k_i)$ refers to B$^+$-tree $P$ or it refers to the object array $P$}
      if(|TK| = $K$ and $B(P) \leq M_K$)then return;
      if($P$ is B$^+$-tree)then
         findTopK(MDB-tree, $(k_1, ..., k_{i-1}, k_i)$, @, $K$);
      if( $P$ is object array)then
         while(there is the next object $x$ in $P$)do
            if(|TK| < $K$ )then
               insert object $x$ to TK to the right place according to @$(x)$;
            else
               if(@$(x)$ > $M_K$)then begin
                   delete $K$-th object from the list TK;
                   insert object $x$ to the list TK on the right place by @$(x)$;
               end;
         endwhile;
   endwhile;
end.

procedure getNextKey(MDBtree MDB-tree, Identifier $(k_1, ..., k_{i-1})$);
   choose the next key $k_i$ with next highest value of $A_i$
   in B$^+$-tree of MDB-tree with identifier $(k_1, ..., k_{i-1})$;
   return $k_i$ ;
end.
```

5.2 Application of local preferences

From B$^+$-tree we can obtain keys in descending order according to user's fuzzy function $f_i^U(x)$. Because MDB-tree is composed from B$^+$-trees, it is possible to use application of the local user preferences directly in MD-algorithm by finding the K best objects in MDB-tree. The following procedure `getNextKey` changes the original MD-algorithm.

```
procedure getNextKey(MDBtree MDB-tree, Identifier $(k_1, ..., k_{i-1})$,
                     FuzzyFunction $f_i^U$);
    choose the next key $k_i$ with next highest value of $i$-th fuzzy function
    in B$^+$-tree of MDB-tree with identifier $(k_1, ..., k_{i-1})$;
    return $k_i$;
end.
```

6 Combination of TA and MD-algorithm

Here we describe a new top-K algorithm, which is based on combination of MD-algorithm and Fagin's TA-algorithm.

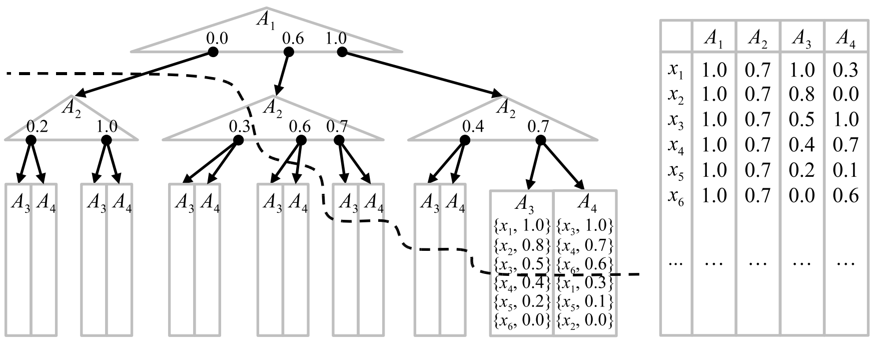
MD-algorithm has the best results, when the objects stored in MDB-tree are distributed regularly (uniform distribution) [1]. When attributes of objects have different size of their actual domains, order of attributes in levels of MDB-tree is very important for efficiency of MD-algorithm. It is better for MD-algorithm to build MDB-tree with smaller actual domains in its higher levels and attributes with bigger actual domains in its lower levels. When most of the object's attributes have their actual domains of big sizes, the usage of MD-algorithm is not suitable solution of top-K problem. In this case, the usage of TA-algorithm is more suitable. In general, we can suppose according to size of attribute's domains of a known object set, which of the algorithms will achieve better results.

Example 1. We have set of 8 822 flats for rent in Prague. Flats have four important attributes for users, *District*, *Type*, *Area*, and *Price*. These attributes have sizes of domains, $\|dom(District)\| = 10$, $\|dom(Type)\| = 10$, $\|dom(Area)\| = 229$, $\|dom(Price)\| = 411$, respectively. When a user prefers attributes *District* and *Type*, then it is better to store flats in MDB-tree and to use MD-algorithm. On the other hand, when a user prefers attributes *Area* and *Price*, then it is better to use TA-algorithm and to store flats in Fagin's lists.

In general, the attribute with a small domain size is *nominal attribute* and attribute with big domain size is *ordinal attribute* (see Section 2). It is valid also in Example 1, attributes *District* and *Type* are nominal attributes, *Area* and *Price* are ordinal attributes.

6.1 MXT-algorithm

Therefore we developed a new top-K algorithm, *MXT-algorithm*, which is based on combination of MD-algorithm and Fagin's TA-algorithm. MXT-algorithm



A1
0.0 0.6 1.0
A2
0.2 1.0
A2
0.3 0.6 0.7
A2
0.4 0.7
A3 A4
A3
{x1, 1.0} {x2, 0.8} {x3, 0.5} {x4, 0.4} {x5, 0.2} {x6, 0.0}
A4
{x3, 1.0} {x4, 0.7} {x6, 0.6} {x1, 0.3} {x5, 0.1} {x2, 0.0}

	A_1	A_2	A_3	A_4
x_1	1.0	0.7	1.0	0.3
x_2	1.0	0.7	0.8	0.0
x_3	1.0	0.7	0.5	1.0
x_4	1.0	0.7	0.4	0.7
x_5	1.0	0.7	0.2	0.1
x_6	1.0	0.7	0.0	0.6
...	...	...	...	...

Fig. 4. Two nominal attributes are stored as MDB-tree and two ordinal attributes are stored as Fagin's lists. Under dotted line there is part of the data structure, in which the MXT-algorithm does not access during its computation.

uses a new data structure, which is composed of MDB-tree and Fagin's sorted lists. This data structure is shown in Figure 4.

We suppose a set of objects X with m attributes $A_1, ..., A_m$. Attributes $A_1, ..., A_n$ are nominal attributes and $A_{n+1}, ..., A_m$ are ordinal attributes. Attributes $A_1, ..., A_n$ are stored in MDB-tree with n levels. Instead of the following $m-n$ levels of MDB-tree, groups of $m-n$ Fagin's sorted lists are used. These lists contain pairs (x, a_i^x) with values of attributes $A_{n+1}, ..., A_m$.

MXT-algorithm is developed on the base of MD-algorithm. First n attributes $A_1, ..., A_n$ are searched in the same way as during the computation of MD-algorithm. In every B$^+$-tree there are references into the groups of $m-n$ Fagin's sorted lists. In each of these groups a new instance of TA-algorithm is run.

The efficiency of MXT-algorithm is based on idea that we do not need to obtain the best K objects from each the group of $m-n$ Fagin's sorted lists. We need to obtain only objects with better rating as M_K (rating of the K-th best object in global list T_K). There, it is sufficient that local threshold T_h^{local} (in each the group of $m-n$ Fagin's sorted lists) with M_K is compared. Local TA-algorithm is able to stop earlier in group of Fagin's sorted lists, $T_h^{local} \leq M_K$ holds earlier than in original TA-algorithm. (see Figure 4, dotted line).

The following pseudo-code describes the MXT-algorithm, where procedure `getNextKey` is the same as in the MD-algorithm (see Section 5).

```
Input: MDBtree $MDB\text{-}tree$, Aggregation @, int $K$;
Output: List TK;
var List TK;                {temporary list of objects}
begin
   findTopK($MDB\text{-}tree$, ($\emptyset$), @, $K$);
   return TK;
end.
```

```
procedure findTopK(MDBtree $MDB\text{-}tree$, Identifier $(k_1,...,k_{i-1})$,
                   Aggregation @, int $K$);
  while(there is next key in $B^+\text{-}tree$)do        {$with\ identifier\ (k_1,...,k_{i-1})$}
    $k_i$ = getNextKey($MDB\text{-}tree$, $(k_1,...,k_{i-1})$);
        {$\rho(k_i)$ refers to B$^+$-tree $P$ or it to group of Fagin's lists $P$}
    if(|TK| = $K$ and $B(P) \leq M_K$)then return;
    if($P$ is B$^+$-tree)then
      findTopK($MDB\text{-}tree$, $(k_1,...,k_{i-1},k_i)$, @, $K$);
    if( $P$ is group of Fagin's lists)then
      while(|TK| < $K$ or $T_h^{local}$ > $M_K$)do
        $(x, a_i^x)$ = getNextPair($L_1,...,L_m$);  {$it\ obtains\ next\ pair\ from$
        $a_i^{last}$ = $a_i^x$;                       $one\ of\ the\ lists\ L_1,...,L_m$}
        $T_h^{local}$ = $@(a_1^{last},...,a_m^{last})$;
        if($x \notin$ TK)then
          get the missing attribute values of the object $x$;
          if(|TK| < $K$)then
            insert object $x$ to the list TK on the right place by $@(x)$;
          else
            if($@(x)$ > $M_K$)then
              begin
                delete $K$-th object from the list TK;
                insert object $x$ to the list TK oaccording to $@(x)$;
              end;
      endwhile;
  endwhile;
end.
```

6.2 Application of local preferences

Analogously to MD-algorithm in attributes $A_1,...,A_n$ it is possible to use application of the local user preferences. Procedure getNextKey changes MXT-algorithm in the same way as in the MD-algorithm (see Section 5.1).

In attributes $A_{n+1},...,A_m$ it is also possible to use application of the local user preferences. Analogously to TA-algorithm, we use as the group of $m-n$ Fagin's lists $L_{n+1},...,L_m$, a list of $m-n$ B$^+$-trees $B_{n+1},...,B_m$, and for a realization of direct access we can use, for example, an associative array.

7 Implementation and Experiments

We implemented top-K TA-algorithm, MD-algorithm and MXT-algorithm using lists based on B$^+$-trees. The implementation has been developed in Java and it uses data structures created in memory. Important for us was the number of accesses into used data structures during calculation of top-K algorithms.

We tested TA-, MD- and MXT-algorithm. During the tests, we used uniform preferences. We used user's fuzzy functions with course "$f(x)$=x" as user's local preferences, and we used the arithmetic average as user's global preference.

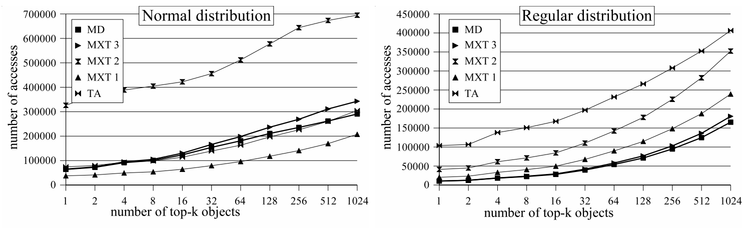


Fig. 5. Normal and regular distribution of the attributes values.

At first, we tested two sets of 100 000 objects with 5 attributes with normal and regular (uniform) distribution of attribute values. We used TA-algorithm, MD-algorithm and three variants of MXT-algorithm, i.e. MXT 3, MXT 2, MXT 1, because attribute types were not important there. For example, variant MXT 3 uses first 3 attributes as nominal attributes, which are stored as MDB-tree with 3 levels and other 2 attributes are stored as groups of 2 Fagin's sorted lists. Figure 5 shows results of this test.

The best results have been achieved with MXT 3 and MD-algorithm for the set of objects with the regular distribution of the attributes values. Test for sets of objects with normal distribution of attribute values has shown that the new MXT-algorithm can in some cases also achieve worst results.

Afterwards, we tested the sets of 8 822 flats for rent in Prague (see Section 6, Example 1). There were two nominal attributes with a small domain size and two ordinal attributes with a big domain size. We used TA-algorithm, MD-algorithm and the most suitable variant of MXT-algorithm (as MXT 2 in previous test). Figure 6 shows results of this test for uniform preferences, and for real user's preferences, where user prefers flats of some types in specific districts, with smaller prices and bigger areas. These two ordinal attributes were preferred according to non-monotone fuzzy functions.

The best results has been achieved with MXT-algorithm for both of used preferences. The test has shown that MXT-algorithm is most efficient solution of top-K problem for some cases. Especially it is efficient for a set of objects, which has nominal attributes with a small domain size and several ordinal attributes with a big domain size.

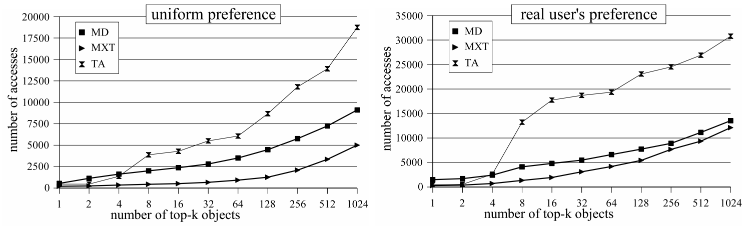


Fig. 6. Number of accesses objects during finding the best flats in Prague.

8 Conclusion

We developed a new MXT-algorithm, which can effectively find the best K objects by user's preferences without accessing all the objects. We implemented top-K algorithms TA-, MD- and MXT-algorithm with support of user's preferences. Results of MXT-algorithm have shown to be comparable with those obtained by others used top-K algorithms.

MXT-algorithm is based on combination TA- and MD-algorithm. According to type of object attributes it is possible to store a set of objects in MDB-tree, in Fagin's lists or in data structure, which MXT-algorithm uses. Moreover, MXT-algorithm can find the best K objects in each of these data structures. In this meaning, TA- and MD-algorithm are extreme cases of MXT-algorithm.

Motivation of future research can be to find application of developed algorithms in various data structures. The article [11] deals with solving of the top-K problem in XML. Application of our algorithms in XML might be also interesting.

References

1. Ondreička, M., Pokorný J.: Extending Fagin's algorithm for more users based on multidimensional B-tree. In: Proc. of ADBIS 2008, P. Atzeni, A. Caplinskas, and H. Jaakkola (Eds.), LNCS 5207, Springer-Verlag Berlin Heidelberg, 2008, pp. 199214.
2. Bayer, R., McCreight, E.: Organization and Maintenance of Large Ordered Indices, Acta Informatica, Vol. 1, Fasc. 3, 1972 pp. 173-189.
3. Eckhardt, A., Pokorný, J., Vojtáš, P.: A system recommending top-k objects for multiple users preference. In Proc. of 2007 IEEE International Conference on Fuzzy Systems, July 24-26, 2007, London, England, pp. 1101-1106.
4. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. Journal of Computer and System Sciences 66 (2003), pp. 614-656.
5. Gurský, P., Lencses, R., Vojtáš, P.: Algorithms for user dependent integration of ranked distributed information. Proceedings of TED Conference on e-Government (TCGOV 2005), Pages 123-130, 2005.
6. Bruno, N., L. Gravano, L., Marian, A.: Evaluating top-k queries over web-accessible databases. in: Proc. of International Conference on Data Engineering (ICDE), 2002, pp. 369 - 380.
7. Chaudhuri, S., Gravano, L., Marian, M.: Optimizing Top-k Selection Queries over Multimedia Repositories. IEEE Trans. On Knowledge and Data Engineering, August 2004 (Vol. 16, No. 8) pp. 992-1009.
8. Scheuerman, P., Ouksel, M.: Multidimensional B-trees for associative searching in database systems. Information systems, Vol. 34, No. 2, 1982.
9. Vojtáš, P.: Fuzzy logic aggregation for semantic web search for the best (top-k) answer, Capturing Intelligence, Volume 1, Chapter 17, 2006, Pages 341-359.
10. Gurský P., Vaneková V., Pribolová J.: Fuzzy User Preference Model for Top-k Search. Proceedings of IEEE World Congress on Computational Intelligence (WCCI), Hong Kong, FS0377, 2008.
11. Marian, A., Amer-Yahia, S., Koudas, N.: Adaptive Processing of Top-k Queries in XML. Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on 05-08 April 2005 Page(s):162 - 173.

Efficiency Improvement of Narrow Range Query Processing in R-tree*

Peter Chovanec and Michal Krátký

Department of Computer Science
Technical University of Ostrava, Czech Republic
{peter.chovanec,michal.kratky}@vsb.cz

Abstract. Indexing methods for efficient processing of multidimensional data are very requested in many fields, like geographical information systems, drawing documentations etc. Well-known R-tree is one of the multidimensional data structures. The R-tree is based on bounding of spatial near points by multidimensional rectangles. This data structure supports various types of queries, e.g. point and range queries. The range query retrieves all tuples of a multidimensional space in the defined query box. Narrow range query is an important type of the range query including at least one narrow dimension. Despite many variants of R-trees, narrow range query processing is inefficient. In this paper, we depict a modification of Signature R-tree: data structure for the narrow range query processing. This data structure applies signatures for a description of tuples stored in a tree's page. We present an improvement of this technique.

Key words: multidimensional data structure, narrow range query, R-tree, signature

1 Introduction

Multimedia databases have become increasingly important in many application areas such as medicine, CAD, geography, and molecular biology. Processing of multi-dimensional data is requested in almost all fields. There are a lot of applications of *multi-dimensional data structures* [15], e.g., data mining [10], term indexing [5, 12], XML documents [8, 11], text documents and images [4]. Query processing in high-dimensional spaces has therefore been a very prominent research area over the last few years. A number of new index structures and algorithms have been proposed.

There are two major approaches to multi-dimensional indexing [16]: data structures for indexing metric spaces and data structures for indexing vector spaces. The first approach includes, for example, n-dimensional B-tree [7], R-tree [9], R*-tree [2], Signature R-tree [13], X-tree [3], UB-tree [1] and BUB-tree [6]. The second one includes M-tree [4], for example.

* Work is partially supported by Grant of GACR No. 201/09/0990

A multi-dimensional data structure often supports the range query. This query may be written as the following pseudo SQL statement:
`SELECT * FROM T WHERE` $ql_1 \leq t_1 \leq qh_1$ `AND` ... `AND` $ql_n \leq t_n \leq qh_n$.

The narrow range query is special type of the range query, where at least one dimension is narrow. In Figure 1, we see examples of query boxes for the narrow range queries in spaces with the dimensions $n = 2$ and $n = 3$, respectively. Another example of this query is the following SQL statement: `SELECT * FROM <table_name> WHERE` $1 < a_0 < 10000$ `AND` $a_1 = 2$ `AND` $a_2 = 3$

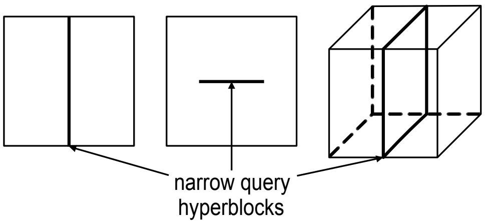


Fig. 1. Examples of the narrow range queries in spaces with the dimensions $n = 2$ and $n = 3$, respectively.

In paper [13], we depicted that the narrow range query processing is rather inefficient in current multidimensional data structures. Signature R-tree, handling point data, is introduced in this work. In our paper, we describe an improvement of the Signature R-tree called ESR-tree. In Section 2, we review existing approaches for the narrow range query processing. Section 3 presents our improvement of the Signature R-tree. Section 4 reviews possibilities of signature building for this data structure. In Section 4.3, we outline some implementation details of the improved data structure. In Section 5, we put forward experimental results. Finally, we conclude with a summary of contributions and discussion about future work.

2 Existing Approaches

In this section, we describe two data structures which have been often applied for the narrow range query processing. These data structures, B-tree and Signature R-tree, are compared with the novel ESR-tree in Section 5.

2.1 B-trees

The B-tree is an m-ary balanced tree introduced by Bayer in 1972. This structure guarantees the logarithmic complexity for the item searching. Due to the ordering, B-tree enables searching only by one attribute. In many cases, it is necessary to search by more than one attribute. Therefore, there are some improvements of B-tree for searching of multidimensional data, e.g. B-tree with compound keys. In this case, we create one compound index with keys related to each narrow dimension. Obviously, this technique is not as general as multidimensional data structures. For example, we create the compound index for dimensions 1, 3, and 10 for a 10-dimensional tuple collection. It means, narrow range query with these narrow dimensions is as efficient as possible. Moreover, no intermediate results are created. If want to process a query with the narrow dimensions: 1, 3, 10, and 5, we retrieve an intermediate result again. If we want to solve a general query without the intermediate result, we should create n! compound indices. Therefore, we do not suppose this technique in our article.

If the range query is concerned in B-tree, we must index each attribute (or dimension) in a separate B-tree. The range query is processed by a sequence of searching in B-trees, and individual intermediate results are joined. Obviously, there are two issues. If the size of an intermediate result >> the overall result, this processing is rather inefficient. The second issue is the size of the index file. In Section 5, we use this implementation for a comparison with our method. We propose that the index file built in this way is 3× larger in average than the index file of a multidimensional data storage.

2.2 Signature R-trees

Since 1984 when Guttman proposed his method [9], R-trees have become the most cited and most used as reference data structure in this field. The R-trees can be thought of as an extension of B-trees in a multi-dimensional space. It corresponds to a hierarchy of nested n-dimensional *minimum bounding boxes* (MBB). There are many approaches based on an improvement of original R-trees. In [13], Signature R-tree was introduced for more efficient processing of narrow range queries.

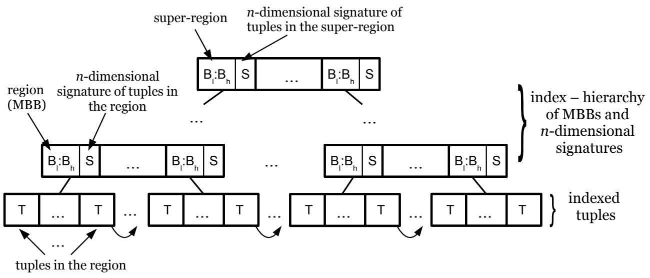

super-region
n-dimensional signature of tuples in the super-region
B_l:B_h
S
...
region (MBB)
n-dimensional signature of tuples in the region
index – hierarchy of MBBs and n-dimensional signatures
T
indexed tuples
tuples in the region

Fig. 2. Structure of the Signature R-Tree

Signature R-tree is a variant of the R-tree including multidimensional signatures for a more efficient filtration of irrelevant tree nodes. In this case, irrelevant node does not contain any tuple of the query box. The multidimensional signature contains a signature of tuples in the node for each dimension (see Section 4.1). A general structure of the Signature R-tree is presented in Figure 2. Leaf nodes include tuples clustered into MBBs (MBB is defined by two multidimensional points). These MBBs are clustered into super-MBBs as well. This hierarchy is finished by MBBs in the root node. Consequently, MBBs are stored in inner nodes. In the case of Signature R-tree, the multidimensional signature is assigned to each MBB. It means that each inner node item includes the MBB definition together with multidimensional signature, where this signature is superimposed with signatures of the node's children. Consequently, such a tree contains two hierarchies, the hierarchy of MBBs and hierarchy of signatures.

In [13], we show that many irrelevant nodes are skipped during the narrow range query processing if we compare Signature R-tree and R-tree. However, we distinguish some negative consequences of this data structure. If signatures are long, the inner node capacity is low. This issue results in a degeneration of the tree: inner node can contain only a trivial number of items and, therefore, the height of the tree is very high. Similar issue appears if we use more hash functions for each signature. Therefore, in this paper, we introduce an improvement of Signature R-tree supporting these refinements.

3 ESR-Tree – An Improvement of Signature R-tree

3.1 Introduction

Signature R-tree significantly decreases the number of processed irrelevant nodes, however the number > 0 (see Section 6). In [13], we introduced the following quality measurement of the range query processing (so called *relevance*): $c_Q = N_r/N_p$, where N_r is the number of relevant nodes, N_p is the number of all processed node. Obviously, if only relevant nodes are processed during a range query then $c_Q = 1$.

If we want to reach the most efficient c_Q value, we must use longer signatures, a higher number of hash functions (it means more signatures is related to one dimension), signatures with different lengths for different levels of a tree, and various hashing functions building the signatures. However, in the case of Signature R-tree, each this idea extends the size of inner node item and decreases the inner node capacity. Therefore, our improvement of Signature R-tree, called ESR-tree, removes signatures from inner nodes: the signatures are stored in a persistent array. A structure of the novel ESR-tree is presented in Figure 3.

The isolation of signatures from R-tree enables to enlarge signatures although the node capacity is not decreased. Moreover, we can use signatures with different lengths for different levels of the tree. In our paper, we use the inverted level number for the signature labeling. Consequently, we use S^0 as the label of an MBB's signature, S^1 as the label of a super-MBB signature, and so on.

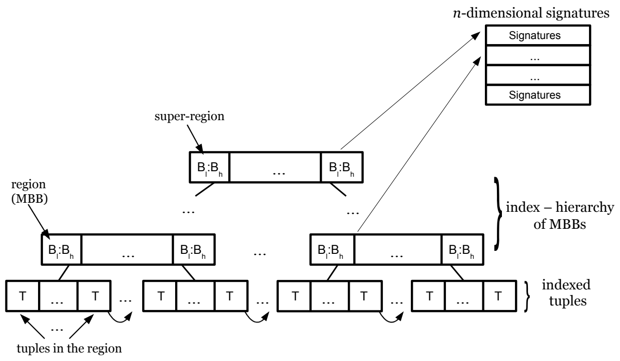


Fig. 3. A structure of the ESR-tree

3.2 ESR-tree Operations

Operations `Insert`, `Delete` and `Find` (or point query) are handled by algorithms of the selected R-tree variant. In the case of the Insert operation (see Algorithm 1), we must update the signature of the MBB which was changed. In this way, we must update signatures for each level of the tree from the current node to the root node. In this algorithm, we use the following variables: *tuple* is an inserted tuple, Z represents a stack containing the current path of the tree, and N is the current node.

Signature index may be created after all tuples are inserted into the tree. This bulkload algorithm must preordered process all tree nodes and create the signature for each MBB. Obviously, signatures may be created for an arbitrary level of the tree.

A common issue of ESR-tree is that the query processing efficiency of a common range query is not influenced by signatures. Signature R-tree includes signatures in tree's nodes, therefore, signatures are read from the secondary storage although these signatures are not used for the node filtering. On the other hand, in the case of the narrow range query, we apply signatures for the more efficient filtration of irrelevant tree nodes.

Let us suppose the range query algorithm. `Intersection` operation computes whether an MBB is intersected by the query box in the linear time, on the other hand, `AND` operation is used as a test of the signature matching. If both operations are matched, the child node is processed.

Delete operation is based on the algorithm of the R-tree variant used. It is necessary to mention that after the item is deleted, unperfect signature may be related to the leaf node containing the deleted item. It means that the signature may contain bits describing a tuple that is not included in the R-tree anymore. Signatures for higher levels of the tree may be unperfect as well. Consequently, we may correct the signatures related the changed node. The complete description of this operation is out of scope of this paper.

Algorithm 1: Insert algorithm

```
N = root ;
Z.Push (N);
splitted ← true ;
while ¬Z.IsEmpty() do
    if splitted then
        N.InsertItem();
        if N.IsOverfull() then
            N.GenerateSignature();
            NewNode.GenerateSignature();
        end
        else
            N.AddSignature(tuple.GetSignature());
            splitted ← true ;
        end
        if N.IsLeaf() then
            N = Z.Pop();
        end
        if ¬N.IsLeaf() then
            if Z.LastMBB() then
                Z.Push(N);
            end
            else
                N = Z.Pop();
            end
        end
    end
end
```

4 Signature Generating

4.1 Signature Methods

The signature is a bit string formed from the terms which are used to index records in a data file [14]. Each term is converted to a bit string by the hashing function. The number of 1's in the signature S is called *weight* $\gamma(S)$. In the case of a query, we build the query signature in the same way as record signatures have been created. If the query signature has 1's in the same positions as the record signature, the record can be considered as a potential match. There can be a case where a record signature matches a query signature, however the record itself does not satisfy the query. This is called the *false drop*.

4.2 Signature Generating for the Irrelevant Node Filtering

The Hamming distance is applied for measuring of the signatures similarity. Signature data structures like S-tree [14] are based on clustering of signatures with the minimal Hamming distance. However, R-tree clusters tuples into MBBs. Nodes do not contain tuples with the minimal Hamming distance. If signatures of tuples in an MBB include many true bits, then the MBB's signature contains almost only true bits. In this case, irrelevant node filtering is not successful. Consequently, one true bit is set for one tuple coordinate and the query signature includes only one true bit for each dimension. In other word, weight of the query signature is rather low.

Our improvement is based on the following assumptions. Query signature weight should be closed to 0.5 as it is known in signature methods [14]. However, signature weight for all tuples of an MBB should be closed to 0.5 as well. These two assumptions are in a contradiction. In this paper, we set more bits as well as we use more hashing functions for one tuple coordinate. In Section 5, we show the efficiency improvement of these novel features.

4.3 Implementation

In the case of ESR-tree, signatures are stored out of the R-tree. In this way, tree height is not influenced by the signature length. The relation between an MBB and its signatures is provided by a conversion table. The conversion table contains couples ⟨node index in the R-tree, signature index in the persistent data structure⟩.

5 Experimental Results

In our tests[1], we compare ESR-tree with Signature R*-tree, R*-tree, and the proposed B-tree-based implementation. We have implemented B$^+$-tree, R*-tree, and ESR-tree in C++. Three collections have been chosen for these tests. We created two random collections with million tuples of dimensions 2 and 10. The third collection represents a set of paths in an XML document [11]. These paths are modeled as 10-dimensional tuples. All collections have been inserted into the multidimensional data structure (see Table 1 for index characteristics). In our experiments, we do not use the Signature R-tree, we use signatures with one true bit generated for one tuple coordinate as the Signature R-tree uses this. We call this signature as the simple signature. This fact has a significant impact on DAC as well as query processing time. However, result relevances are credible for a comparison of Signature R-tree and ESR-tree.

Efficiency of the narrow range query processing was measured by DAC, c_Q, and query processing time. Tested range queries have various number of narrow dimensions $|N_{rq}|$. The 2 kB page size and random accesses are applied in the case of all data structure, therefore, we can measure DAC by the number of MBs read in the secondary storage.

[1] The experiments were executed on an AMD Opteron 865 1.8Ghz, 2.0 MB L2 cache; 2GB of DDR333; Windows 2008 Server.

Table 1. Characteristics of the R-tree indices

	1st Random Collection	2nd Random Collection	Real Collection
Dimension	2	10	10
Coordinate value range	$< 0, 10^9 >$	$< 0, 5 \cdot 10^4 >$	$< 0, 10^9 >$
Result size	$< 31, 40 >$	$< 1, 1 >$	$< 0, 12000 >$
#Nodes	120	499	852
Inner item capacity	102	48	48
#Items	8,513	15,793	21,612
Node utilization	69.6%	65.9%	52.8%
#Leaf nodes	8,394	15,295	20,761
Leaf item capacity	170	92	92
#Leaf items	999,904	1,000,000	1,031,080
Leaf node utilization	70.1%	71.1%	54.0%
#Leaf signatures – S^0	8,394	15,295	20,761
#Overleaf signatures – S^1	117	483	821

5.1 1st Random Collection

In Table 2, DAC results are presented for the random collection of dimension 2. We use 2 hashing functions and 3 true bits in the signatures. Results are average values of 10 various queries. We measure DAC for R-tree (RT) and signature data structure (SA).

Table 2. 1st Random Collection: DAC

Signature Length	**DAC [MB]**											
	Simple			ESR-tree – S^0			ESR-tree – S^1			ESR-tree – S^0 AND S^1		
S^0/S^1	RT	SA	RT + SA	RT	SA	RT + SA	RT	SA	RT + SA	RT	SA	RT + SA
128/1,024	0.55	0.01	0.56	0.57	0.02	0.59	0.83	0.01	0.84	0.57	0.04	0.61
256/3,072	0.36	0.02	0.38	0.23	0.05	0.28	0.80	0.03	0.83	0.22	0.08	0.30
384/7,168	0.28	0.04	0.32	0.14	0.07	0.21	0.65	0.07	0.72	0.12	0.13	**0.25**
512/10,240	0.25	0.05	0.30	0.11	0.10	**0.21**	0.59	0.10	0.69	0.09	0.17	0.26
640/12,288	0.20	0.06	**0.26**	0.10	0.12	0.22	0.57	0.12	0.69	0.09	0.21	0.30
768/14,336	0.20	0.07	0.27	0.10	0.15	0.25	0.54	0.14	**0.69**	0.08	0.24	0.32
R-tree without signature filtering: **0.83**												

Obviously, DAC is 4× lower if we compare the R-tree and ESR-tree with the leaf signature of the length 512. DAC of signature reading is an essential part of overall DAC, therefore, we can not use longer signatures. Table 3 including the query processing time supports this conclusion. This time is 5.25× lower than in the case of R*-tree. Obviously, we see that ESR-tree saves 20% of the query processing time of the simple signature.

Another view of the trend is the higher values of relevance. We use the various count of hashing functions, more true bits of the signature as well as various signature lengths. In Table 3, relevances for leaf and overleaf nodes are presented.

Relevance rapidly increases with the increasing signature length. Summary Table 4 presents results for the relevance > 0.9. We suppose that the signature weight should be closed to 0.5. In the case of this experiment, we get the signature weight in the range 0.38 – 0.51.

Table 3. 1st Random Collection: results for 2 hashing functions, 3 true bits

Signature Length	**Time [s]**				**c_Q of S^0**				**c_Q of S^1**
	Simple	S^0	S^1	S^0 AND S^1	Simple	S^0	S^1	S^0 AND S^1	
128/1024	0.045	0.064	0.069	0.042	0.19	0.18	0.14	0.18	0.67
256/3072	0.037	0.020	0.061	0.020	0.26	0.36	0.14	0.37	0.70
384/7168	0.022	0.020	0.050	0.020	0.31	0.61	0.16	0.66	0.87
512/10240	0.022	0.016	0.044	0.011	0.34	0.85	0.16	0.89	0.94
640/12288	0.020	0.016	0.045	0.014	0.41	0.93	0.17	0.95	0.94
768/14336	0.020	0.019	0.044	0.016	0.44	0.96	0.17	0.98	0.98
	R-tree: **0.084**				R-tree: **0.14**				R-tree: **0.67**

Table 4. 1st Random Collection: summary table

Signature Type	**Results for $c_Q > 0.9$**			
	Signatures Length	DAC – RT [MB]	DAC – SA [MB]	Time [s]
2 hash/ 3 bits	640/12288	0.087 (0.054+ 0.033)	0.207	0.0140
2 hash/ 1 bit	-	-	-	-
3 hash/ 1 bit	512/4096	0.093 (0.057 + 0.036)	0.178	0.0140
2 hash/ 6 bits	768/10240	0.088 (0.052+ 0.036)	0.217	0.0078

5.2 2nd Random Collection

The second collection includes 10-dimensional randomly generated tuples. 3 narrow dimensions of range queries are used. In this case, we can use shorter signatures than in the case of the first test. It seems that this 10-dimensional space is sparser than the 2-dimensional space, therefore, shorter signatures can describe the tuple distribution as well. In Table 5, we can see that DAC of ESR-tree is much more lower than DAC of R*-tree. The query processing time is improved $9\times$. Obviously, we can see the importance of overleaf signatures in this case. Summary Table 6 presents results for the relevances > 0.9.

5.3 Real Collection

Third collection contains a set of paths in an XML document. We test 15 range queries Q_1–Q_{15} with various narrow dimensions. We use the same signature

Table 5. 2nd Random Collection: DAC

Signature Length S^0/S^1	**DAC [MB]**											
	Simple			ESR-tree – S^0			ESR-tree – S^1			ESR-tree – S^0 AND S^1		
	RT	SA	RT + SA	RT	SA	RT + SA	RT	SA	RT + SA	RT	SA	RT + SA
96/1024	4.62	0.69	5.31	4.35	1.39	5.74	23.25	0.95	24.20	3.99	2.23	6.22
128/1536	3.20	0.93	4.13	2.21	1.85	4.06	16.84	1.43	18.27	1.44	2.67	4.11
160/2048	2.58	1.16	3.74	1.71	2.31	**4.02**	10.92	1.91	12.83	0.71	2.91	3.62
192/2560	2.20	1.39	**3.59**	1.62	2.78	4.40	5.69	2.38	8.07	0.36	3.01	**3.37**
224/3072	1.99	1.62	3.61	1.60	3.24	4.84	2.79	2.86	5.65	0.20	3.22	3.42
256/3584	1.89	1.85	3.74	1.59	3.70	5.29	1.58	3.34	**4.92**	0.14	3.56	3.70
R-tree without signature filtering: **25.28**												

Table 6. 2nd Random Collection: summary table

Signature Type	**Results for** $c_Q > 0.9$				
	Signatures length	Relevance	DAC – RT [MB]	DAC – SA [MB]	Time [s]
2 hash/ 3 bits	256/3584	0.95	0.140 (0.004 + 0.136)	3.564	0.0780
2 hash/ 1 bit	352/2048	0.95	0.214 (0.004 + 0.210)	2.499	0.0936
3 hash/ 1 bit	160/1376	0.90	0.261 (0.004 + 0.257)	2.444	0.1092
2 hash/ 6 bits	352/4096	1.00	0.342 (0.004 + 0.338)	4.924	0.1420

lengths as in the case of the second collection. In Table 7, DAC and c_Q are put forward. We use signature lengths 256/3584 with 2 hashing functions and 3 true bits. We see that the c_Q of Signature R*-tree significantly increases in the comparison with the common R*-tree. Obviously, this relevance is not sufficient in many cases. ESR-tree overcomes the Signature R*-tree and R*-tree.

5.4 The Comparison with B-tree

In this test, we compare the efficiency of ESR-tree and B-tree (one B-tree was created for each dimension). We use the queries from the previous test. The join operation takes the most processing time (see Table 8). Obviously, B-tree is much more efficient in the case of small intermediate results. With the increasing size of intermediate results, join processing time increases. Table 8 includes information about all queries. We measure DAC for B-trees (BT) and tuple array (TA) including whole tuples. Obviously, ESR-tree clearly overcomes this B-tree based implementation, DAC is 7.5× lower in the case of ESR-tree. Another important issue is the index size. In Table 9, we see that the index size of the ESR-tree is 3× lower in all tested cases. The experiments show an significant improvement, e.g. applications of two hashing functions causes double increasing of the relevance.

6 Conclusion

In this article, we present an improvement of Signature R-tree, data structure for the efficient processing of narrow range queries. Moreover, we introduce an enhanced signature creation that provides more efficient filtration characteristics. Since signatures are relevant only in higher levels of a tree, it is not appropriate to handle them to each MBB of an inner node. We show some advantages

Table 7. Real Collection Test: comparison of R*-tree, Signature R*-tree, and ESR-tree

Query	**Result Sizes**	**R*-tree**		**Simple**		**ESR-tree** – S^0 **AND** S^1	
		c_Q	**DAC [MB]**	c_Q	**DAC [MB]**	c_Q	**DAC [MB]**
Q1	2000	0.76	1.04	1.00	0.82+0.07	1.00	0.78+0.34
Q2	1201	0.26	13.77	0.80	4.88+1.03	0.98	3.91+2.71
Q3	12000	0.64	15.82	1.00	10.43+1.19	1.00	10.22+2.98
Q4	3	0.27	0.09	1.00	0.06+0.003	1.00	0.04+0.08
Q5	10	0.36	0.09	1.00	0.07+0.003	1.00	0.05+0.08
Q6	6	0.03	0.91	0.13	0.29+0.06	1.00	0.06+0.24
Q7	10	0.13	0.30	0.88	0.13+0.02	1.00	0.07+0.17
Q8	18	0.48	0.18	0.63	0.16+0.01	1.00	0.09+0.13
Q9	1	0.0006	6.02	0.003	1.43+0.45	0.08	0.13+0.74
Q10	27	0.03	3.99	0.23	0.68+0.29	0.79	0.23+0.70
Q11	43	0.03	5.19	0.21	1.13+0.38	0.87	0.35+1.11
Q12	13	0.18	0.36	0.58	0.19+0.02	1.00	0.11+0.24
Q13	1	0.06	0.13	1.00	0.07+0.005	1.00	0.03+0.11
Q14	24	0.50	0.14	1.00	0.11+0.004	1.00	0.07+0.14
Q15	3	0.14	0.15	1.00	0.07+0.01	1.00	0.05+0.11
Average		**0.26**	**3.21**	**0.69**	**1.87 + 0.24**	**0.91**	**1.08 + 0.66**

of longer signatures, however longer signatures mean the lower node's capacity. Consequently, we put signatures of each MBB in a special data structure: a persistent array. In our experiment, we test range queries and compare the efficiency of our approach with R-tree, Signature R-tree, and B-tree based implementation. From DAC point of view, ESR-tree is up to 3× more efficient than R-tree and 6× than B-tree. Obviously, DAC of the signature retrieval is often rather high. Therefore, we want to develop a more efficient data structure for the storage of signatures.

References

1. R. Bayer. The Universal B-Tree for multidimensional indexing: General Concepts. In *Proceedings of WWCA'97, Tsukuba, Japan*, 1997.
2. N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD*, pages 322–331.
3. S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An index structure for high-dimensional data. In *Proceedings of the 22nd International Conference on VLDB*, pages 28–39, San Francisco, U.S.A., 1996. Morgan Kaufmann Publishers.
4. P. Ciaccia, M. Pattela, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *Proceedings of 23rd International Conference on VLDB*, pages 426–435, 1997.
5. V. Dohnal, C. Gennaro, and P. Zezula. A Metric Index for Approximate Text Management. In *Proceedings of IASTED International Conference Information Systems and Database – ISDB 2002*, 2002.

Table 8. Real Collection Test: results of the B-tree-based implementation

Query	$\lvert N_{rq}\rvert$	**Intermediate Result Sizes**	**Result Size**	**DAC (BT + TA) [MB]**	**Time join [s]**	**Time [s]**
Q1	2	2000; 1031080;	2000	19.80 (15.90 + 3.90)	9.91	14.22
Q2	2	12000; 2539;	1201	2.58 (0.24 + 2.35)	0.13	0.27
Q3	1	12000;	12000	23.63 (0.19 + 23.44)	0	1.91
Q4	4	1031080; 105161; 10; 3;	3	17.51 (17.50 + 0.01)	4.06	4.07
Q5	3	1031080; 105161; 10;	10	17.51 (17.49 + 0.02)	4.06	4.07
Q6	2	7512; 126;	6	0.14 (0.13 + 0.01)	0.031	0.032
Q7	4	1031080; 279205; 13; 10;	10	20.19 (20.17 + 0.02)	4.73	4.73
Q8	3	1031080; 279205; 18;	18	20.20 (20.17 + 0.04)	4.79	4.79
Q9	2	25500; 1;	1	0.41 (0.41 + 0.002)	0.093	0.094
Q10	2	37843; 104;	27	0.65 (0.60 + 0.05)	0.14	0.14
Q11	2	12715; 43;	43	0.30 (0.21 + 0.08)	0.046	0.047
Q12	4	1031080; 251465; 69340; 13;	13	20.84 (20.81 + 0.03)	4.86	4.86
Q13	4	1031080; 105161; 9; 1;	1	17.502 (17.50 + 0.002)	4.06	4.06
Q14	4	1031080; 4324; 25; 24;	24	16.00 (15.95 + 0.05)	3.77	3.77
Q15	4	1031080; 105161; 10; 3;	3	17.51 (17.50 + 0.01)	4.06	4.06
Average			**1024**	**12.98 (10.98 + 1.99)**	**2.98**	**3.41**

Table 9. Sizes of indexes

Dimension	**Data Collection**	**R-tree + SA [MB]**	**B-trees + Tuples Array [MB]**
2	1st Random	16.89 (16.6 + 0.29)	40.95 (2 × 16.6 + 7.63)
10	2st Random	65.1 (61.6 + 3.49)	204.2 (10 × 16.6 + 38.2)
10	Real	89.38 (84.4 + 4.98)	204.4 (10 × 16.6 + 38.4)

6. R. Fenk. The BUB-Tree. In *Proceedings of 28rd VLDB International Conference on VLDB, Hongkong, China*, 2002.
7. M. Freeston. A General Solution of the n-dimensional B-tree Problem. In *Proceedings of SIGMOD International Conference, San Jose, USA*, 1995.
8. T. Grust. Accelerating XPath Location Steps. In *Proceedings of ACM SIGMOD 2002, Madison, USA*, June 4-6, 2002.
9. A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of ACM SIGMOD 1984, Annual Meeting, Boston, USA*, pages 47–57. ACM Press, June 1984.
10. N. Karayannidis, A. Tsois, T. Sellis, R. Pieringer, V. Markl, F. Ramsak, R. Fenk, K. Elhardt, and R. Bayer. Processing Star Queries on Hierarchically-Clustered Fact Tables. In *Proceedings of VLDB Conf. 2002, Hongkong, China*, 2002.
11. M. Krátký, J. Pokorný, and V. Snášel. Implementation of XPath Axes in the Multi-dimensional Approach to Indexing XML Data. In *Current Trends in Database Technology, Int'l Conference on EDBT 2004*, volume 3268. Springer–Verlag, 2004.
12. M. Krátký, T. Skopal, and V. Snášel. Multidimensional Term Indexing for Efficient Processing of Complex Queries. *Kybernetika, Journal*, 40(3):381–396, 2004.
13. M. Krátký, V. Snášel, P. Zezula, and J. Pokorný. Efficient Processing of Narrow Range Queries in the R-Tree. In *Proceedings of IDEAS 2006*. IEEE CS Press, 2006.
14. Y. Manolopoulos, A. Nanopoulos, and E. Tousidou. *Advanced Signature Indexing for Multimedia and Web Applications.* Kluwer, 2003.
15. Y. Manolopoulos, Y. Theodoridis, and V. Tsotras. *Advanced Database Indexing.* Kluwer Academic Publisher, 2001.
16. C. Yu. *High-Dimensional Indexing.* Springer–Verlag, LNCS 2341, 2002.

Author Index