Department of Computers, Czech Technical University, Prague
Department of Software Engineering, Charles University, Prague
Department of Computer Science, VŠB – Technical University of Ostrava
ČSKI, OS Computer Science and Society

Proceedings of the Dateso 2004 Workshop

DATESO 2004

Databases, Texts, Specifications, and Objects

http://www.cs.vsb.cz/dateso/2004/

AMPHORA RESEARCH GROUP

April 14 – 16, 2004
Desná – Černá Říčka

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

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

Page count:	171
Impression:	150
Edition:	1st
First published:	2004

This proceedings was typeset by PDFLaTeX.
Cover design by Tomáš Skopal, tomas.skopal@vsb.cz.
Printed and bound in Olomouc, Czech Republic by Vydavatelství Univerzity Palackého.

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

Preface

DATESO 2004, the international workshop on current trends on Databases, Information Retrieval, Algebraic Specification and Object Oriented Programming, was held on April 14th – 16th, 2004 in Desná – Černá Říčka. This was the fourth annual workshop organized by FEL ČVUT Praha, Department of Computer Science and Engineering, MFF UK Praha, Department of Software Engineering, and VŠB-Technical University Ostrava, Department of Computer Science. The DATESO aims for strengthening the connection between this various areas of informatics. The proceedings of DATESO 2004 are also available at DATESO Web site `http://www.cs.vsb.cz/dateso/2004/`.

The Program Committee selected 15 papers from 17 submissions, based on two independent reviews.

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

March, 2004 V. Snášel, J. Pokorný, K. Richta (Eds.)

Program Committee

Organizing Committee

Table of Contents

Designing Indexing Structure for Discovering Relationships in RDF Graphs

Stanislav Bartoň

Faculty of Informatics, Masaryk University, Brno, Czech Republic
xbarton@fi.muni.cz

Abstract. Discovering the complex relationships between entities is one way of benefitting from the Semantic Web. This paper discusses new approaches to implementing ρ-operators into RDF querying engines which will enable discovering such relationships viable. The cornerstone of such implementation is creating an index which describes the original RDF graph. The index is created in two steps. Firstly, it transforms the RDF graph into forest of trees and then to each tree creates its extended signature. The signatures are accompanied by the additional information about transformed problematic nodes breaking the tree structure.

1 Introduction

One form of retrieving information from the Semantic Web is to search for relations among entities. The simple relations such are the *is-a* or *is-part-of* relations can be found easily. For example using RQL [3] one can find direct relationship among entities. This means that we are able to retrieve all the descending classes of one class, even on a different level. For example the user can ask for all instances of a class 'artist' as it is shown in Figure 1. The answer to such query would be all instances of both its subclasses in the knowledge base, all painters and sculptors. But in the Semantic Web there can be observed more complex relationships among entities [7] than those simple ones.

Such complex relationship can be represented by a path between two entities consisting of other entities and their properties. To discover such complex relationships ρ-operators [1] have been developed. In this paper, the complex relationships are discussed and are referred to as Semantic Associations [7]. The ρ-operators are precisely the tools for discovering such Semantic Associations. This class contains ρ *path*, ρ *connect* and ρ *iso* operators.

ρ **path** - This operator returns all paths between two entities in the graph. An example of such relation can be seen in Figure 1 between resources `&r6` and `&r8`. Such association represents an information that a painter called Pablo Picasso had painted a painting which is exhibited in Reina Sofia Museum.

ρ **connect** - This one returns all intersecting paths, on which the two entities lie. An example of those two intersecting paths is the one between resources `&r6` and `&r8` and between resources `&r9` and `&r8`. This association represents a fact that two artist had their artifacts (in one case it was a painting and in the other a sculpture) exhibited in the same museum.

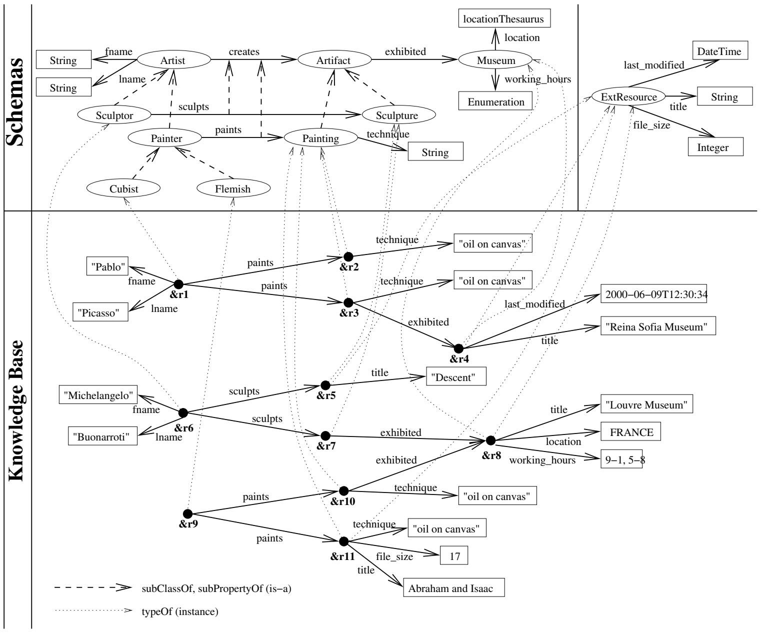


Fig. 1. An example of RDF graph

ρ **iso** - This operator implies a similarity of nodes and edges along the path, and returns such similar paths between entities. An example of such association is the relation between resources `&r1` and `&r9`. This represents a fact that both subjects are classified as painters.

The ρ iso operator should also return an information that the subjects are artists besides that they are both painters. Ranking of such answers is very important since the fact that two subjects are painters is obviously more relevant than they are artists, due to its greater specialization. The relevance of relations found of course significantly depends on the context in which are the queries asked.

The possible usage of searching such complex associations can found in the field of national security. For example the system could find its usage on airports helping to identify suspicious passengers by looking for available connections between them.

In this paper we mainly focus on the former two operators which are the ρ path and ρ connect. We introduce a design of a indexing structure for the RDF graph that will make the discovery of the relationships described by these ρ operators effective.

Section 3 discusses the related work to the topic of indexing RDF graphs. Section 2 contains a brief introduction into the RDF and the RDF Schema. In Section 4 we present out contribution to the issue by introducing the transformation of the RDF graph into forest of trees and after-wards the application of tree signatures to those trees. Section 5 outlines possible improvements to the indexing structure that is designed in this paper. Finally Section 6 concludes the whole paper.

2 Preliminaries

The RDF graph depicted in Figure 1 is visualization of an RDF and RDF Schema notation. These two languages are used to state the meta information about resources. The following subsections briefly describe this technology. In the scope of this paper the RDF is used to create the knowledge base and the RDF schema to build the schema parts of the RDF graph.

2.1 RDF

The abbreviation RDF stands for resource description framework and according to [4] is supposed to be a foundation for processing metadata. It basically provides a data model for describing machine-processable semantics of data. The RDF statement is a triple (O, P, V) which parts stand for object, property and value. Object is usually identified by URI. It is basically a resource. The value can be either an explicit value or a resource also. Since this triple itself can be considered as a resource it can appear in an RDF statement as well. This means that the data model can be envisioned as a labeled hypergraph (each node can be an entire graph) where an edge between two nodes represents the property between the object and value.

2.2 RDF Schema

Because the modeling primitives of RDF are so basic, there is no way to define the class-subclass relation. Therefore an externally specified semantics to some resources was provided. Such enriched RDF is called RDF Schema [2]. Those specific resources are for example rdfs:class and rdfs:subclass.

In such enriched environment we are able to define a simple model of classes and their relations. This can be used to define simple ontologies in the web space. The RDF Schema statements are expressed using XML together with its specific namespace. Even RDF statements can be expressed using XML with its specific namespace.

3 Related work

To make the best of these operators, they should be implemented into an RDF querying system. One of such implementation is presented in [5]. The effort described there demonstrates an implementation of ρ path operator above the RDF Suite [3]. The implementation cornerstones are two indices, the Path index

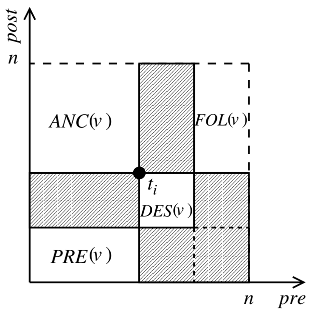


Fig. 2. Properties of the preorder and postorder ranks.

and a Schema index. The former one is two-dimensional array of paths - it carries the information about all paths between Class i and Class j in the schema part of the RDF graph. The latter one is used to search for a path between classes in different schemas. The Path index is very memory intensive when the data grows to large amounts. Therefore this paper discusses a different approach to index the data for the purpose of discovering Semantic Associations.

4 Indexing RDF graphs

The idea of indexing RDF graph demonstrated in this paper is to transform it into tree or forest of trees in which the searching for relationship between particular nodes will be much easier than in general directed graph. If we consider ρ-path and ρ-connect operators, the problem is to find certain paths among particular nodes. Therefore we deploy convenient indexing structure to each tree to optimize such searching. Thus the signature [8] to each tree will be created. This approach solves the problem of getting the relationship between each pair of nodes in a tree by an atomic operation. Such relationship between two nodes in a tree is represented by their mutual position in such tree (i.e. ancestor, descendant, preceding or following). Tree signatures are described in the following subsection.

4.1 Tree signatures

The idea of the tree signature is to maintain a small but sufficient representation of the tree structures. The preorder and postorder ranks[1] are used as suggested in [6] to linearize the tree structure.

The basic tree signature is a list of pairs. Each pair contains a tree node name along with the corresponding postorder rank. The list is ordered according to the preorder rank of nodes.

[1] How the preorder and postorder ranks are obtained please refer to [8].

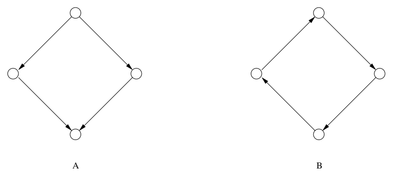


Fig. 3. Directed graphs that are not trees.

Given a node v with particular preorder and postorder ranks, their properties can be summarized in a two-dimensional diagram, as illustrated in Figure 2, where *ancestors ANC(v)*, *descendants DES(v)*, *preceding PRE(v)*, and *following FOL(v)* nodes of v in the particular tree are clearly separated in their proper regions. Due to these properties the mutual position of two nodes within one signature is clear immediately after reading a record of either of them in the particular signature.

According to the signature structure the basic tree signature can be further extended. To each entry a pair of preorder numbers is added. Those numbers represent pointers to the *first following*, and the *first ancestor* nodes of a given node. If no terminal node exists, the value of the first ancestor is zero and the value of the first following node is $n+1$, where n is the number of nodes in a tree. Such enriched signature is called *extended signature*. Later on when we refer to signature we will mean the extended one.

4.2 Transforming the graph into forest of trees

The structure of the RDF Schema and the knowledge base can be envisioned as a directed graph with arcs provided with labels, example is shown in Figure 1. The inconvenience of this structure lies in the problem of searching path between nodes. Such searching algorithms work with great time computational complexity.

Because the structure depicted above is not really a general directed graph, we can get the benefit of the schema part of the structure since it carries useful information about the knowledge base. The schema part has the same function as a schema in the relational database. Then if we could reduce the problem of searching in the whole graph to the problem of searching in the schema, which is considerably smaller, we could use the same algorithms with better time complexity results. But since the graph can contain several schema definitions and the resources can be derived from more than one schema, the desired paths can only be found using the real data, because they would not be included in the schema definition.

Knowledge base transformation A tree can be defined as a directed graph in which is true that (1) each node has zero or one incoming edge and (2) it does not contain a cycle. Directed graphs marked as A and B depicted in Figure 3 break those rules respectively. The transformation of the directed graph into forest of trees lies in the removal of such problematic cases.

If we consider the problem marked as (A) in Figure 3, part (1) in Figure 4 shows a transformation to achieve structure conforming to the rule marked as (1). The black node in the phase 1 in Figure 4, means that the node will be 'divided' into two nodes in the following phase. The next phase has two alternatives, phase 2a demonstrates the division of a node with a duplication of all descendants to all divided nodes. Phase 2b shows the division without duplication. The right way to handle such situation is to use the former method since it prevents the uncontrollable growth of the structure. This assures that the structure will grow in linear space instead of possible exponential growth. The descending nodes should be cut off into stand alone component to avoid 'short cuts' within one component. This becomes important in the moment of finding paths between nodes.

Thus the whole graph is traversed and all the nodes that have more than one incoming edge are divided into exact amount of nodes that is the number of that node's incoming edges. This transformation can lead to breaking the graph[2] into several components. These components are either trees or directed graphs containing a cycle. To identify which components are trees a rule that a graph is a tree only if it has exactly *n+1* edges, where *n* represents the number of nodes in a particular component. The non-tree components are then transformed as follows.

The transformation of the directed graph containing a cycle is depicted in the part marked as (2) in the Figure 4. The spanning tree of such component is found and the nodes, which edges are not contained in the spanning tree are divided. The transformation works in the way that it divides the particular node into two, that the first one contains all the edges that have the original node as the terminal one, and the extra node has all the edges that had the original as a initial one.

[2] We consider that at the beginning the graph consists from only one component.

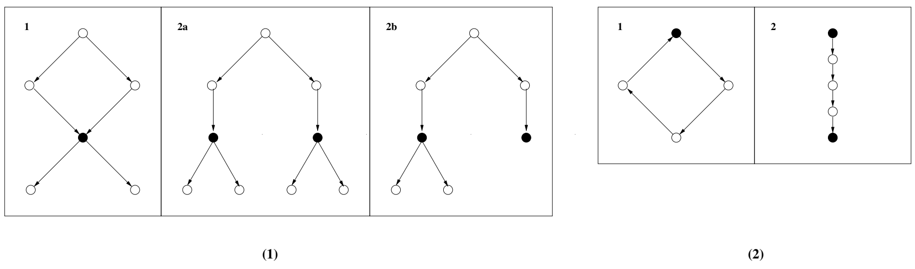


Fig. 4. Transformation of a graph to conform with rules (1) and (2) respectively.

Obviously, after transforming all the non-tree components, we get a forest of trees representing the original graph. Of course we have to store the information about the divided nodes to assure that no information contained in the original graph will not be lost in the new structure. Such information is stored in two tables where the first one is used to get all the multiple nodes[3] in the particular signature, and the second table stores to each multiple node all signatures it appears in. Those two tables connect the components back into the original graph.

The time computation complexity of the transformation of a general directed graph into forest of trees is estimated to $\mathcal{O}(2n)$ in the worst case. It mostly depends on the number of components which after the first transformation contain cycle. The spanning tree to such component has to be built but that means that all the nodes of such components has to be traversed again. If all the newly built components have cycles we have to traverse the whole graph again.

4.3 ρ path and ρ connect implementation

Once we obtain the desired forest of trees we create a signature to each component (tree) of the transformed graph which together with the additional information about multiple nodes will represent the index to the original RDF graph. The time computational complexity of such operation is equal to $\mathcal{O}(n)$ since the algorithm used traverses each node in each component once. The additional information connecting signatures together is built along and deploys only atomic operations. Such information about the multiple nodes is represented by two tables. One has in each row a name of a multiple node together with a particular signature or signatures it appears in. And the other one has a row for each signature with a list of multiple nodes contained in it.

Above such index an algorithms implementing the ρ path and ρ connect operators have been designed. The outline of those algorithms are demonstrated in Algorithm 1 and Algorithm 2.

Algorithm for discovering paths The first algorithm returns an answer whether there exists a path between two nodes. The algorithm traverses the forest of trees only in one direction, so to tell whether there is really a path between two nodes we have to switch the start and end node and deploy the algorithm again if the search has not been successful for the first time. As a by-product it also creates a list of multiple nodes that lie on the path between the two nodes. The exact path is not computed at this point. Another function, to which this list is passed, takes care of the exact path computation. To make the most from the tree structure of this index, the path is computed from the bottom to the top, the first ancestor pointer from the signature is used to traverse the path.

[3] A node which was represented as a one in the original graph, but is represented by several nodes in the new structure.

Algorithm 1 Name: findPathUp

Input: startSignature, startNode, checkedMultiples, endNode, wholePath
Output: true if the path between startNode and endNode exists else false

```
returnValue = false;
if startSignature.isNodeInSig(endNode) then
  return startNode.isDescendantOrSelfOf(endNode);
else
  multiplesInSignature = getMultiplesInSignature(startSignature);
  if multiplesInSignature.isEmpty() then
    return false
  end if
  Set usableMultiples = $\emptyset$;
  for all multipleNode in multiplesInSignature do
    if multipleNode.isAncestorOrSelfOf(startNode) AND !checkedMultiples.contains(multipleNode) then
      usableMultiples.add(multipleNode);
    end if
  end for
  if usableMultiples.isEmpty() then
    return false;
  end if
  for all usableMultiple in usableMultiples do
    usableSigantures = getSignaturesToMultiple(usableMultiple);
    checkedMultiples.add(multipleNode);
    for all usableSignature in usableSignatures do
      uniqueNodes = getSetOfMultipleNodeNamesInSignature();
      for all uniqueNode in uniqueNodes do
        if !checkedMultiples.contains(uniqueNode) then
          if findPathUp(usableSignature, uniqueNode, checkedMultiples, endNode, wholePath) then
            wholePath.add(uniqueNode);
            returnValue=true;
          end if
          checkedMultiples.add(uniqueNode);
        end if
      end for
    end for
  end for
end if
return returnValue;
```

Therefore the algorithm traverses the index structure in only one direction, from bottom to top, it has to be deployed twice unless the path has not been found in the first deployment. Thus to check whether there is not a path between two nodes we have to execute the algorithm twice with both nodes used as a starting point respectively. This implies that the time computational complexity of finding a path between two nodes mainly depends on existence of such path. The problem of dual execution could be solved if we could tell the mutual position of the two nodes in the indexing structure. Then we could deploy the algorithm exactly once to tell whether there exist a path between the two nodes or it does not.

Algorithm for discovering connections As for the ρ connect operator, the nature of the designed index structure implies that the connection, the intersecting node, can only be a multiple node. Therefore the problem of finding two paths that intersect is reduced to finding a multiple node, to which exists a path

Algorithm 2 Name: findConnection

```
Input: node1, node2
Output: node where the two paths intersect or null
 1: Set checkedMultiples1, checkedMultiples2, testedIntersections = ∅;
 2: List multiples1, multiples2, toDoMultiples1, toDoMultiples2, wholePath1, wholePath2 =
    ∅;
 3: done=false; found = false;
 4: while !done do
 5:   Set usableMultiples1, usableMultiples2;
 6:   checkOneUsableMultiple(toDoMultiples1, usableMultiples1, checkedMultiples1, node1,
      node1Signature);
 7:   checkOneUsableMultiple(toDoMultiples2, usableMultiples2, checkedMultiples2, node2,
      node2Signature);
 8:   toDoMultiples1.addAll(usableMultiples1); multiples1.addAll(usableMultiples1);
 9:   toDoMultiples2.addAll(usableMultiples2); multiples2.addAll(usableMultiples2);
10:   testMultiples = (multiples1 ∩ multiples2) - testedIntersections;
11:   if !testMultiples.isEmpty() then
12:     intersection = testMultiples.get(0);
13:     intSignature = getSignature2Node(intersection);
14:     testedItersections.add(intersection);
15:     if findPathUp(intSignature, intersection, node1, wholePath1) AND find-
        PathUp(intSignature, intersection, node2, wholePath2) then
16:       return intersection;
17:     end if
18:     if toDoMultiples1.isEmpty() AND toDoMultiples2.isEmpty() then
19:       done=true;
20:     end if
21:   end if
22: end while
23: return null;
```

from either node. The outlined Algorithm 2 searches the index structure in a direction that the edges have. Its starting nodes are the two nodes to which it is looking for connection.

Throughout the algorithm a set of multiple nodes, nodes which lie below the particular starting node and are possible intersection, a set of checked nodes, nodes through which the algorithm already switched to different signatures and got all usable multiples in it, and a set of to do multiple nodes, nodes that have to be still checked, are built to each starting node. In each cycle iteration those sets are updated for each starting node separately, each starting node gets one turn to check one multiple node. At the end of each iteration, the algorithm checks whether there is a non-empty intersection of possible intersecting nodes and if such intersection exists, it checks whether there exist paths from this node to both starting nodes.

The above outlined algorithm for finding path intersection also very intensively depends on the existence of such intersection. So far we can not stop the algorithm without searching the entire index that is reachable from the two starting points. It obviously also suffers from the impossibility of telling the mutual position of two nodes in the indexing structure. Therefore the time computational complexity is unacceptably high when looking for a connection that apparently does not exist in a very large graph.

Summarization The outlined algorithms in this section are implemented with an emphasis to represent the idea of searching paths using the designed index to RDF graphs. Thus they provide a vast space for further optimization, for example to speed the first algorithm up a cycle could be used instead of the approach with recursion. Also the access structures to the indexing structure could be improved to save significant amount of time to speed the algorithms up. Another way to improve the indexing structure is to create a second level that would ease the problem of mutual position of nodes in the graph. This approach is further discussed in the next section.

5 Future work

As it was proposed in the previous section, the future work will focus on two directions of improving this project. Firstly, we will work on optimization of the implemented algorithms to get better results on large scale data.

Secondly, as it was recognized in the previous section the main drawback of both algorithms is that it is not able to tell the mutual position of two nodes in the graph. So the future work in this direction is to solve this problem. Under heavy research is a second level indexing structure on the designed index. The idea is that the signatures, representing transformed stand-alone components of the original RDF graph will be assumed as nodes and the edges will be the connections through the multiple nodes to other signatures. Then the same transformation as it is presented in this paper will be applied again.

6 Concluding remarks

In comparison to the index structure designed in [5], the path between two nodes, if it exists, does not have to be precomputed to create the index. Instead, the index is used to compute the path or the ρ *connection* between two nodes. One of pros of the former indexing structure is that once all paths among all nodes are computed, the searching of the ρ path is very fast and effective. The cons are that such index is very memory intensive and the time to create such index is great. The index structure discussed in this paper can be created in linear time and the storage complexity is also linear to the size of the original RDF graph.

As was mentioned in Section 1 we recognize three ρ operators but the demonstrated indexing structure together with its algorithms can handle only two of them. The future work thus will also focus on extending the index to provide search for all Semantic Associations defined by the ρ operators.

The aim of this project is to create a scalable indexing structure for RDF graphs accompanied with algorithms providing the ρ operators functionality with acceptable time and space computational complexity. In present time the designed indexing structure provides solid base for such work but new approaches mentioned in previous sections has to be taken into account to make it all possible.

References

1. Kemafor Anyanwu and Amit Sheth. The rho operator: Discovering and ranking on the semantic web. 2003.
2. D. Brickley and R. V. Guha. Resource description framework schema specification. 2000.
3. G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl. RQL: A declarative query language for RDF. In *The 11th Intl. World Wide Web Conference (WWW2002)*, 2002.
4. O. Lassila and R. R. Swick. Resource description framework: Model and syntax specification. 1999.
5. Agarwal Minal, Gomadam Karthik, Krishnan Rupa, and Yeluri Durga. Rho: Semantic operator for extracting meaningful relationships from semantic content.
6. T.Grust. Accelerating xpath location steps. In *The 11th Intl. World Wide Web Conference (WWW2002)*, pages 109–120, 2002.
7. Sanjeev Thacker, Amit Sheth, and Shuchi Patel. Complex relationships for the semantic web. 2001.
8. Pavel Zezula, Giuseppe Amato, Franca Debole, and Fausto Rabitti. Tree signatures for XML querying and navigation. *Lecture Notes in Computer Science*, 2824:149–163, 2003.

LSI vs. Wordnet Ontology in Dimension Reduction for Information Retrieval*

Pavel Moravec, Michal Kolovrat, and Václav Snášel

Department of Computer Science, FEI, VŠB - Technical University of Ostrava,
17. listopadu 15, 708 33, Ostrava-Poruba, Czech Republic
{pavel.moravec, michal.kolovrat}@vsb.cz

Abstract. In the area of information retrieval, the dimension of document vectors plays an important role. Firstly, with higher dimensions index structures suffer the "curse of dimensionality" and their efficiency rapidly decreases. Secondly, we may not use exact words when looking for a document, thus we miss some relevant documents. LSI (Latent Semantic Indexing) is a numerical method, which discovers latent semantic in documents by creating concepts from existing terms. However, it is hard to compute LSI. In this article, we offer a replacement of LSI with a projection matrix created from WordNet hierarchy and compare it with LSI.

Keywords: vector model, latent semantic indexing, LSI, information retrieval, dimension reduction, WordNet

1 Introduction

The *information retrieval* [13,3] deals among other things with storage and retrieval of multimedia data, which can be usually represented as vectors in multidimensional space. This is especially suitable for *text retrieval*, where we store a *collection* (or *corpus*) of texts. There are several models used in text retrieval, from which we will use the *vector model* [10,12] providing qualitatively better results than the *boolean model* [13], which combines word matching with boolean operators.

In vector model, we have to solve several problems. The ones addressed in this article are the size of resulting index, search efficiency and precision and search for documents similar to the query.

To measure the improvement of a new indexing method, we can use several measures, both quantitative and qualitative. The quantitative measures show us the performance of an indexing structure, they can be for example number of disc accesses – *disc access cost* (*DAC*) – or total time of performed indexing and search – *wall clock time*. The qualitative measures tell us how good does this new indexing structure reflect the reality when obtaining an *answer set* A for a

* This paper was partially supported by GAČR 201/03/1318 grant.

given *query* Q. The most commonly used qualitative measures are *precision* (P) and *recall* (R) [3], where precision is a fraction of relevant documents in answer set and recall is a fraction of retrieved relevant documents in all relevant ones.

In second chapter, we will describe classic vector model and above mentioned problems. In third, we will describe *latent semantic indexing* (*LSI*). In fourth chapter a basic description of English *WordNet* ontology will be given. In fifth chapter we will offer a way how to use WordNet for concept creation instead of LSI and in sixth we will present comparison of our approach with LSI and random projection on real documents from TREC collection.

2 Vector Model

In vector model, a document D_j is represented as a vector d_j of term weights, which record the extent of importance of the term for the document.

To portrait the vector model, we usually use an $n \times m$ *term-by-document matrix* A, having n rows – term vectors $t_1 \ldots t_n$ – where n is the total number of terms in collection and m columns – document vectors $d_1, \ldots d_m$, where m is the size of collection C.

Term weights can be calculated in many different ways – $t_i \in \{0, 1\}$, as a membership grade to a fuzzy set, or as a product of functions of term frequency both in a document and in the whole collection [11] (usually $tf * idf$ – count of term occurrences in the document multiplied by a logarithm of the inverse portion of documents containing the term). Sometimes is the normalisation of document vectors applied during index generation phase to make the calculation in retrieval phase faster.

A query Q is represented as an n-dimensional vector q in the same vector space as the document vectors. There are several ways how to search for relevant documents. Generally, we can compute some L_n metrics to represent similarity of query and document vectors. However, in text retrieval can be better results obtained by computing *cosine measure*:

$$sim_{cos}(d_j, q) = \frac{d_j q}{||d_j|| \times ||q||} = \frac{\sum_{i=1}^{n} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{n} w_{i,j}^2} \times \sqrt{\sum_{i=1}^{n} w_{i,q}^2}}$$

As one can see, we do not only obtain documents which are considered relevant, but according to their distance (or similarity) to the query vector, we can order them and obtain rank for every document in answer set. We can define a *threshold* t, too, meaning that all documents closer than this threshold will be considered relevant, whilst the rest will be irrelevant. However, the choice of the threshold is not exact and its value is usually determined experimentally.

The main problem of vector model is, that the document vectors have a big dimension (e.g. 150,000) and are quite sparse (i.e. most coordinates are zero). If we store them as classical vectors, the storage volume is huge – consider size of a term-by-document matrix consisting of 100,000 terms and 200,000 documents.

We can use existing compression schemes for the term-by-document matrix representation like the *compressed column storage* (*CCS*) to conserve memory, but the co-ordinate access time is much longer and we are limited by the fact, that we cannot access the term vectors quickly. Or we can use combined storage with both CCS and *compressed row storage* (*CRS*). Anyway, updating would still be a problem.

The second problem is the so-called *"curse of dimensionality"*, which causes classical indexing structures like M-trees, R-trees [6], A-trees, iDistance, etc. to perform same or even worse than sequential scan in higher dimension. Moreover, the vectors are placed almost equidistantly from each other, which makes clustering ineffective.

Third, even there is a better chance that we can find relevant documents when using some terms which are not contained in them, the synonyms and other semantically related words are not taken in account.

The first two problems can be addressed for queries containing only a few words by *inverted list*, which is in fact compressed storage of term vectors. Only term vectors for terms contained in a query Q are loaded and processed, computing rank for all documents containing at least one of the terms at once. However, the inverted list is not efficient when searching for similar documents, because significant part of index must be processed.

3 Latent Semantic Indexing

Latent semantic indexing (*LSI*) [3,4] is an algebraic extension of classical vector model. First, we decompose the term-by-document matrix A by either *principal component analysis* (*PCA*), which computes eigenvalues and eigenvectors of covariance matrix or *singular value decomposition* (*SVD*), calculating singular values and singular vectors of A. SVD is especially suitable in its variant for sparse matrices (Lanczos [7]).

Theorem 1. (Singular value decomposition): *Let A is an $n \times m$ rank-r matrix. Be $\sigma_1 \geq \cdots \geq \sigma_r$ eigenvalues of a matrix $\sqrt{AA^T}$. Then there exist orthogonal matrices $U = (u_1, \ldots, u_r)$ and $V = (v_1, \ldots, v_r)$, whose column vectors are orthonormal, and a diagonal matrix $\Sigma = diag(\sigma_1, \ldots, \sigma_r)$. The decomposition $A = U\Sigma V^T$ is called* singular decomposition *of matrix A and 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 original term-by-document matrix A. Needless to say, the left and right singular vectors are not sparse. We have at most r nonzero singular numbers, where rank r is smaller of the two matrix dimensions. However, we would not conserve much memory by storing the term-by-document matrix this way. Luckily, because the singular values usually fall quickly, we can take only k greatest singular values and corresponding singular vector coordinates and create a *k-reduced singular decomposition* of A.

Definition 1.: *Let us have $k, 0 < k < r$ and singular value decomposition of A*

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

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

We would not conserve any space with the matrix A_k. So instead of the A_k matrix, a concept-by-document matrix $D_k = V_k \Sigma_k$ with k concepts is used. To convert a query Q to the concept-space, we create $q_k = U_k^T q$ [1]. The similarity of terms in concept space can be calculated from $U_k \Sigma_k$ [2].

If every document contains only one topic (for more details see [9]), we obtain a *latent semantics* – semantically related terms will be close in concept space and will result in similar answer set when querying. This addresses the third of above mentioned problems. And since the first co-ordinates of D_k have the greatest influence on similarity, the clustering results are better.

Experimentally was k determined to several tens or hundreds (e.g. 50–250), exact value of k is however a mystery; it is dependent on the number of topics in collection. For a illustration of rank-k SVD see Figure 1.

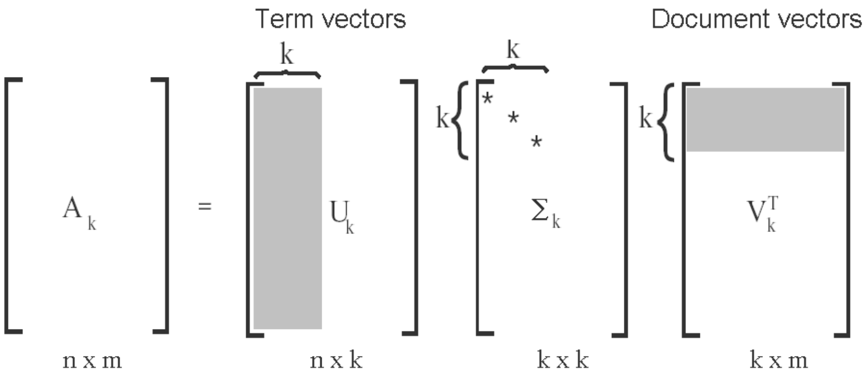


Fig. 1. k-reduced singular value decomposition

Rank-k SVD is the best rank-k approximation of original matrix A. This means, that any other decomposition will increase the sum of squares of matrix $A - A_k$. However, this does not mean, that we could not obtain better precision and recall values with a different approximation.

The LSI is hard to compute and once computed, it reflects only the decomposition of original term-by-document matrix. If several hundreds of documents

[1] The second approach is to use a matrix $D'_k = V_k$ instead of D_k and $q'_k = U_k^T \Sigma_k^{-1}$

[2] or U_k in second approach

or terms have to be added to existing decomposition (*folding-in*), the decomposition may become inaccurate. The recalculation of LSI is expensive, so it is impossible to recalculate LSI every time documents and terms are inserted. The *SVD-Updating* [8] is a partial solution, but since the error slightly increases with inserted documents and terms, the recalculation of SVD may be needed soon or later.

4 WordNet Ontology

WordNet is an online lexical reference system whose design is inspired by current psycholinguistic theories of human lexical memory. English nouns, verbs, adjectives and adverbs are organised into synonym sets, each representing one underlying lexical concept.

The goal of WordNet project is the creation of dictionary and thesaurus, which could be used intuitively. The next purpose of WordNet is the support for automatic text analysis and artificial intelligence. WordNet is also useful for determining semantic connections between sets of synonyms, for tracing morphological connections between words.

The ontology is organised not only by the "is-the-synonym-of" relation; the verbs and nouns are hierarchically organised via the *hypernym/hyponym* relation (superior/inferior concepts), too. An example of hypernyms for "ontology" is given in figure 2.

```
psychological feature
  cognition, knowledge, noesis
    content, cognitive content, mental object
      knowledge domain, knowledge base
        discipline, subject, field, study, bailiwick, ...
          humanistic discipline, humanities, liberal arts, arts
            philosophy
              metaphysics
                ontology
```

Fig. 2. Example of hypernyms of the term "ontology"

EuroWordNet is a multilingual database with WordNets for several European languages (Dutch, Italian, Spanish, German, French, Czech and Estonian). The WordNets are structured in the same way as the American WordNet for English (Princeton WordNet) in terms of synsets (sets of synonymous words) with basic semantic relations between them. Each WordNet represents a unique language-internal system of lexicalizations.

In addition, the WordNets are linked to an *Inter-Lingual-Index*, based on the Princeton WordNet. Via this index, the languages are interconnected so that it is possible to go from the words in one language to similar words in any other language.

This index also gives access to a shared top-ontology of 63 semantic distinctions which provides a common semantic framework for all the languages, while language specific properties are maintained in the individual WordNets. The database can be used, among others, for monolingual and cross-lingual information retrieval, which was demonstrated by the users in the project.

5 Using WordNet Hypernyms instead of LSI Concepts

As mentioned above, the calculation of SVD is quite difficult and since the resulting matrices U and V are dense, memory can be exhausted quite quickly. So we face a question, how to create concepts for given document collection.

One possibility is the usage of Papadimitriou's two-step algorithm [9] combining *random projection* (see e.g. [1]) with LSI. Simply said, we first create a pseudoconcept-by-document matrix A' with a suitable number of pseudoconcepts by multiplication of a zero-mean unit-variance projection matrix and the term-by document matrix A. In second step we calculate rank-$2k$ LSI of A', which gives us a very good approximation of rank-k LSI of original matrix A.

We experimentally verified this method recently and showed that the approximation error against LSI is low, however we do not obtain same concepts as with original LSI. Because there is usually not the same number of singular values as for the original matrix A (e.g. 60'000), but only the number for the reduced dimension (e.g. 1000), the concepts are created differently and are almost equal (having similar singular values). This results in poor clustering and worse selection of k.

However, we know the hierarchy of concepts defined by the WordNet synonym/hypernym organisation. We can use all hypernyms of given term from l top levels, applying a fraction of term weight dependent on its level in WordNet to hypernym weight in concept-by-document matrix. This would give us a different linear combination of term vectors than classical LSI, with non-negative concept weights.

This way we can create a term-to-concept projection matrix, applying adequate parts of each term to its hypernyms. The top l levels of hypernyms will give us new concepts. The results may be worse than in case of LSI, but would give us a better starting point than random projection which chooses the pseudoconcepts as a random linear combination of terms.

Is the resulting dimension too high for convenient direct use (but not too high to make the LSI calculation problematic as in case of original matrix A), we can replace random projection in the Papadimitriou's two-step approximate LSI calculation method by the term-to-concept projection matrix and calculate LSI on generated term-by-concept matrix, which will improve the response time.

The problem is, that hypernym hierarchy was created only for nouns and verbs. Adjectives and adverbs can't be handled this way, which brings some complications. We can either use all concepts from this area, or silently ignore them, which will cause either an increase of reduced dimension or worse recall. The same problem is with numbers and names. While a number can be easily identified, we can create a category "Number" to place all numbers in, we cannot do this with names, so we either ignore them, too, or we can create a predefined number of random concepts which would contain terms not found in WordNet with a given weight.

6 Experimental Results

For the comparability with LSI, 5000 Los Angeles Times articles from the TREC 5 document collection were indexed. LSI into a dimension of 100 and 200 was calculated using both classical and two-step algorithm. For comparison, the random projection was calculated, too.

Tests were run with English Wordnet version 2.0; the WordNet concepts were used both directly and as the first step in the two-step algorithm instead of random projection. In first case, two and three top levels were used. In second case, four top levels of WordNet hierarchy were used. The term weight in a concept was inversely proportional to a logarithm of concept level.

Tests were executed on AMD Athlon 2200+ with VIA KT-400 chipset and 1GB DDR-333 RAM. The LSI and random projection routines were written in C/C++.

The average precision and recall for 50 TREC queries was calculated for all mentioned methods and classical vector model. The results are summarised in table 1.

Table 1. Precision and recall of 50 executed TREC queries

Method	**Precision**	**Recall**
Original sparse term-by-document matrix	79%	74%
rank-100 LSI	74%	100%
rank-200 LSI	74%	100%
rank-200 LSI after RP to dim. 1000	75%	94%
Random projection to dimension 1000	77%	82%
rank-200 LSI of 5961 WordNet concepts	74%	96%
2747 WordNet concepts calculation	73%	100%
502 WordNet concepts calculation	73%	100%

Unfortunately, there are some problems which reduce usability of these results. First, the TREC queries consist of a small number of words, thus they are not usable as document similarity queries. Second, because of the collection size, there were between 1 and 5 relevant documents for each query and 0 to

10 documents which are surly irrelevant. The rest is supposed to be irrelevant but was not checked manually when the queries were created. When we treat these documents as non-relevant, the precision for both LSI and WordNet-based reduction and cosine measure is poor (around 1%).

Because the response times for TREC queries were too short and they did not represent the similarity queries we are mainly focused on, we created a set of 1000 document similarity queries for following tests. The query times for these query set are shown in table 2 together with index size and dimension reduction time. The query times represent an average over 5 test runs.

Table 2. Dimension reduction and query times in seconds; document matrix size

Method	**Reduction time**	**Query time**	**Index size**
Original sparse term-by-document matrix	N/A	33	8,3 MB
rank-100 LSI	2730	6	1,9 MB
rank-200 LSI	3784	11	3,8 MB
rank-200 LSI after RP to dim. 1000	2026	10	3,8 MB
Random projection to dimension 1000	20	44	19,0 MB
rank-200 LSI of 5961 WordNet concepts	3262	10	3,8 MB
2747 WordNet concepts calculation	267	50	16 MB
502 WordNet concepts calculation	227	23	5,15 MB

7 Conclusion

We have shown, that using WordNet ontology instead of random projections offers better results and is even comparable with classical LSI. This could make the dimension reduction feasible even for huge document collections.

The selection of concepts was very rough, we can employ a filtration step and use concepts whose document frequency lies within given borders, which will probably lead to improved precision. We can also filter out several inappropriate meanings of given terms and use other lexical categories like antonyms. Both weighting approach and similarity function may be further modified to provide better precision.

With the use of EuroWordnet's Inter-Lingual-Index for mapping of given concepts, we may be even able to index texts written in different languages.

We are currently studying an interesting application offered by a reversed approach – we try to verify an existing ontology with LSI. If a suitable mapping could be found and formally described, it may become possible to create an ontology on a collection of multimedia documents, e.g. images by calculation of LSI of the collection or its randomly-selected sample [5].

References

1. D. Achlioptas. Database-friendly random projections. In *Symposium on Principles of Database Systems*, 2001.
2. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison Wesley, New York, 1999.
3. M. Berry and M. Browne. *Understanding Search Engines, Mathematical Modeling and Text Retrieval.* Siam, 1999.
4. M. Berry, S. Dumais, and T. Letsche. Computation Methods for Intelligent Information Access. In *Proceedings of the 1995 ACM/IEEE Supercomputing Conference*, 1995.
5. A. Frieze, R. Kannan, and S. Vempala. Fast Monte-Carlo Algorithms for Finding Low Rank Approximations. In *Proceedings of 1998 FOCS*, pages 370–378, 1998.
6. 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.
7. R. M. Larsen. Lanczos bidiagonalization with partial reorthogonalization. Technical report, University of Aarhus, 1998.
8. G. W. O'Brien. Information Management Tools for Updating an SVD-Encoded Indexing Scheme. Technical Report ut-cs-94-258, The University of Tennessee, Knoxville, USA, December, 1994.
9. C. H. Papadimitriou, H. Tamaki, P. Raghavan, and S. Vempala. Latent semantic indexing: A probabilistic analysis. In *Proocedings of the ACM Conference on Principles of Database Systems (PODS)*, pages 159–168, 1998.
10. G. Salton. *The SMART Retrieval System – Experiments in Automatic Document Processing.* Prentice Hall Inc., Englewood Clifs, 1971.
11. G. Salton and C. Buckley. Term weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.
12. G. Salton and M. Lesk. Computer evaluation of indexing and text processing. *Journal of the ACM*, 15(1):8–39, January 1968.
13. G. Salton and G. McGill. *Introduction to Modern Information Retrieval.* McGraw-ill, 1983.

Pivoting M-tree: A Metric Access Method for Efficient Similarity Search

Tomáš Skopal

Department of Computer Science, VŠB–Technical University of Ostrava,
tř. 17. listopadu 15, Ostrava, Czech Republic
tomas.skopal@vsb.cz

Abstract. In this paper pivoting M-tree (PM-tree) is introduced, a metric access method combining M-tree with the pivot-based approach. While in M-tree a metric region is represented by a hyper-sphere, in PM-tree the shape of a metric region is determined as an intersection of the hyper-sphere and a set of hyper-rings. The set of hyper-rings for each metric region is related to a fixed set of pivot objects. As a consequence, the shape of a metric region bounds the indexed objects more tightly which, in turn, improves the overall efficiency of the similarity search. Preliminary experimental results on a synthetic dataset are included.

Keywords: PM-tree, M-tree, pivot-based methods, efficient similarity search

1 Introduction

Together with the increasing volume of various multimedia collections, the need for an efficient similarity search in large multimedia databases becomes stronger. A multimedia document (its main features respectively) is modelled by an object (usually a vector) in a feature space $\mathcal{U}$ thus the whole collection can be represented as a dataset $S \subset \mathcal{U}$. Similarity search is then provided using a *spatial access method* [1] which should efficiently retrieve those objects from the dataset that are relevant to a given similarity query.

In context of similarity search, a similarity function (dissimilarity function actually) can be modeled using a metric, i.e. a distance function d satisfying the following metric axioms for all $O_i, O_j, O_k \in \mathcal{U}$:

$$\begin{array}{rlr} d(O_i,O_i) = 0 & & \text{reflexivity} \\ d(O_i,O_j) > 0 & (O_i \neq O_j) & \text{positivity} \\ d(O_i,O_j) = d(O_j,O_i) & & \text{symmetry} \\ d(O_i,O_j) + d(O_j,O_k) \geq d(O_i,O_k) & & \text{triangular inequality} \end{array}$$

Given a metric space $\mathcal{M} = (\mathcal{U}, d)$, the *metric access methods* [2] organize (or index) objects of a dataset $S \subset \mathcal{U}$ just using the metric d. Most of the metric access methods exploit a structure of metric regions within the space $\mathcal{M}$. Common to all these methods is that during a search process the triangular inequality of d allows to discard some irrelevant subparts of the metric structure.

2 M-tree

Among many of metric access methods developed so far, the M-tree [3,5] (and its modifications) remains still the only indexing technique suitable for an efficient similarity search in large multimedia databases.

The M-tree is based on a hierarchical organization of feature objects $O_i \in S$ according to a given metric d. Like other dynamic, paged trees, the M-tree structure is a balanced hierarchy of nodes. The nodes have a fixed capacity and a utilization threshold. Within the M-tree hierarchy, the objects are clustered into metric regions. The leaf nodes contain *ground entries* of indexed objects themselves while *routing entries* (stored in the inner nodes) represent the metric regions. A ground entry has a format:

$$grnd(O_i) = [O_i, oid(O_i), d(O_i, P(O_i))]$$

where $O_i \in S$ is an appropriate feature object, $oid(O_i)$ is an identifier of the original DB object (stored externally), and $d(O_i, P(O_i))$ is a precomputed distance between O_i and its parent routing entry. A routing entry has a format:

$$rout(O_j) = [O_j, ptr(T(O_j)), r(O_j), d(O_j, P(O_j))]$$

where $O_j \in S$ is a feature object, $ptr(T(O_j))$ is pointer to a covering subtree, $r(O_j)$ is a covering radius, and $d(O_j, P(O_j))$ is a precomputed distance between O_j and its parent routing entry (this value is zero for the routing entries stored in the root). The routing entry determines a hyper-spherical metric region in space $\mathcal{M}$ where the object O_j is a center of that region and $r(O_j)$ is a radius bounding the region. The precomputed value $d(O_j, P(O_j))$ is redundant and serves for optimizing the M-tree algorithms.

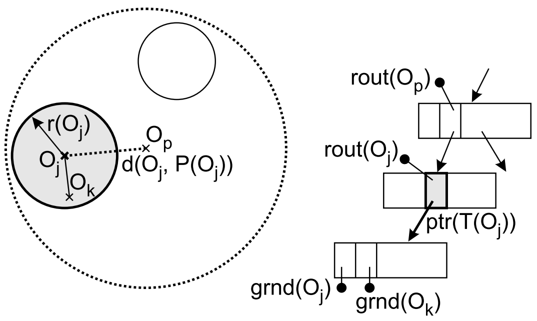


Fig. 1. A metric region and its routing entry in the M-tree structure.

In Figure 1, a metric region and its appropriate routing entry $rout(O_j)$ in an M-tree are presented. For a hierarchy of metric regions (routing entries $rout(O_j)$ respectively) the following condition must be satisfied:

All feature objects stored in leafs of covering subtree of $rout(O_j)$ must be spatially located inside the region defined by $rout(O_j)$.

Formally, having a $rout(O_j)$ then $\forall O_i \in T(O_j), d(O_i, O_j) \leq r(O_j)$. If we realize, such a condition is very weak since there can be constructed many M-trees of the same object content but of different structure. The most important consequence is that many regions on the same M-tree level may overlap.

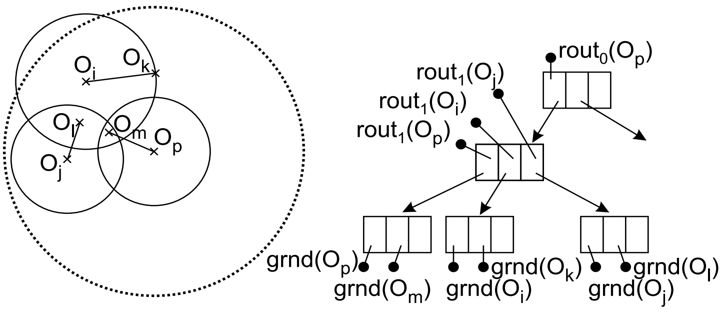


Fig. 2. Hierarchy of metric regions and the appropriate M-tree.

An example in Figure 2 shows several objects partitioned into metric regions and the appropriate M-tree. We can see that the regions defined by $rout_1(O_p)$, $rout_1(O_i)$, $rout_1(O_j)$ overlap. Moreover, object O_l is located inside the regions of $rout_1(O_i)$ and $rout_1(O_j)$ but it is stored just in the subtree of $rout_1(O_j)$. Similarly, the object O_m is located even in three regions but it is stored just in the subtree of $rout_1(O_p)$.

2.1 Similarity Queries

The structure of M-tree was designed to natively support similarity queries. A similarity measure is here represented by the metric function d. Given a query object O_q, a similarity query returns (in general) objects $O_i \in S$ close to O_q.

In the context of similarity search we distinguish two kinds of queries. A *range query* is specified as a hyper-spherical *query region* defined by a query object O_q and a query radius $r(O_q)$. The purpose of a range query is to return all the objects $O_i \in S$ satisfying $d(O_q, O_i) \leq r(O_q)$. A query with $r(O_q) = 0$ is called a *point query*. A *k-nearest neighbours query* (k-NN query) is specified by a query object O_q and a number k. A k-NN query returns the first k nearest objects to O_q. Technically, a k-NN query can be implemented using a range query with a dynamic query radius.

During a similarity query processing the M-tree hierarchy is being traversed down. Only if a routing object $rout(O_j)$ (its metric region respectively) intersects the query region, the covering subtree of $rout(O_j)$ is relevant to the query and thus further processed.

2.2 Retrieval Efficiency

The retrieval efficiency of an M-tree (i.e. the costs of a query evaluation) is highly dependent on the amount of overall volume[1] of the metric regions described by routing entries. The larger metric region volumes (and also volumes of region overlaps) the higher probability of intersection with a query region.

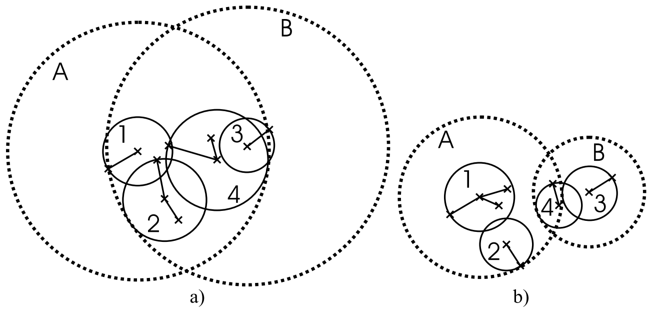


Fig. 3. a) An M-tree with large volume of regions. b) An M-tree with small volume of regions.

In Figure 3 two different yet correct M-tree hierarchies for the same dataset are presented. Although both M-trees organize the same dataset, a query processing realized on the second M-tree will be more efficient (in average) due to the smaller region volumes.

Recently, we have introduced two algorithms [6] leading to a reduction of the overall volume of metric regions. The first method, the multi-way dynamic insertion, finds the most appropriate leaf for each object being inserted. The second (post-processing) method, the generalized slim-down algorithm, "horizontally" (i.e. separately for each tree level) tries to redistribute all entries among more appropriate nodes.

3 Pivoting M-tree

A metric region (as a part of routing entry) of M-tree is described by a bounding hyper-sphere (given by a center object and a radius). However, the shape of hyper-spherical region is far from optimal since it does not "wrap" the objects tightly together and the region volume is too large. In other words, relatively to the hyper-sphere volume there is only a "few" objects spread inside the hyper-sphere thus a huge proportion of an empty space[2] is covered. Consequently,

[1] We consider only an imaginary volume since there exists no universal notion of volume in general metric spaces.

[2] The uselessly indexed empty space is sometimes refered as a "dead space".

for hyper-spherical regions of large volumes the query processing becomes less efficient.

In this section we introduce an extension of M-tree, called *pivoting M-tree* (PM-tree), exploiting the pivod-based idea for metric region volumes reduction.

3.1 Pivot-based Methods

Similarity search realized by pivot-based methods [2,4] is based on a single general idea. A set of p (random) objects $\{p_1, ..., p_l, ..., p_k\} \subset S$ is selected, called *pivots*. The dataset S (of size n) is preprocessed so as to build a table of $n * p$ entries, where all the distances $d(O_i, p_l)$ are stored for every $O_i \in S$ and every pivot p_l. When a range query $(O_q, r(O_q))$ is processed, we compute $d(O_q, p_l)$ for every pivot p_j and then try to discard such O_i that $|d(O_i, p_l) - d(O_q, p_l)| > r(O_q)$. The objects O_i which cannot be eliminated with this rule have to be directly compared against O_q.

The simple pivot-based approach is suitable especially for applications where the distance d is considered expensive to compute. However, it is obvious that the whole table of $n * p$ entries must be sequentially loaded during a query processing which significantly increases the disk access costs.

3.2 Structure of PM-tree

Since PM-tree is an extension of M-tree we just describe the new facts instead of a comprehensive definition. To exploit advantages of both, the M-tree and the pivot-based approach, we have enhanced the routing and ground entries by a pivot-based information.

First of all, a set of p pivots $p_l \in S$ must be selected. This set is fixed for all the lifetime of a particular PM-tree index. Furthermore, we define a routing entry of a PM-tree inner node as:

$$rout_{PM}(O_j) = [O_j, ptr(T(O_j)), r(O_j), d(O_j, P(O_j)), HR]$$

The additional HR attribute stands for an array of p_{hr} *hyper-rings* ($p_{hr} \le p$) where the l-th hyper-ring $HR[l]$ is an interval (possibly the smallest) covering distances between the pivot p_l and each of the objects stored in leafs of $T(O_j)$, i.e. $HR[l].min = min(\{d(O_i, p_l)\})$ and $HR[l].max = max(\{d(O_i, p_l)\})$ for $\forall O_i \in T(O_j)$. Similarly, for a PM-tree leaf we define a ground entry as:

$$grnd_{PM}(O_i) = [O_i, oid(O_i), d(O_i, P(O_i)), PD]$$

The additional PD stands for an array of p_{pd} *pivot distances* ($p_{pd} \le p$) where the l-th distance $PD[l] = d(O_i, p_l)$.

Since each hyper-ring stored in HR defines a metric region containing *all* the objects indexed by $T(O_j)$, an intersection of hyper-rings and the hyper-sphere forms a metric region bounding all the objects in $T(O_j)$. Furthermore, due to the intersection with hyper-sphere, the PM-tree metric region is always smaller

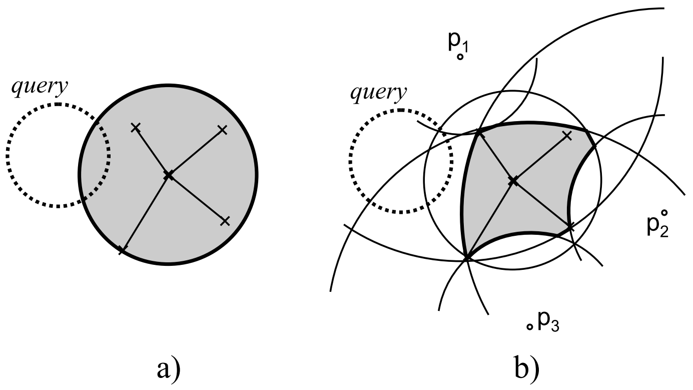


Fig. 4. a) Region of M-tree. b) Reduced region of PM-tree (with three pivots).

than the original M-tree region defined just by a hyper-sphere. For a comparison of an M-tree region and an equivalent PM-tree region see Figure 4.

The PM-tree, as a combination of M-tree and the idea of pivoting, represents a metric access method based on *hierarchical pivoting*. The numbers p_{hr} and p_{pd} (both fixed during a PM-tree index lifetime) allow us to specify the "amount of pivoting". For $p_{hr} > 0$ and $p_{pd} = 0$ only the hierarchical pivoting will take place while for $p_{hr} = 0$ and $p_{pd} > 0$ a query will be processed like in the ordinary M-tree with subsequent pivot-based filtering in leafs. Obviously, using a suitable $p_{hr} > 0$ and $p_{pd} > 0$ the PM-tree can be tuned to achieve an optimal storage/retrieval efficiency.

3.3 Building the PM-tree

In order to keep *HR* and *PD* arrays up-to-date, the original M-tree construction algorithms [5,6] must be adjusted. The adjusted algorithms still preserve the logarithmic time complexity.

Object Insertion.
During an object O_i insertion, the *HR* array of each routing entry in the insertion path must be updated by values $d(O_i, p_l), \forall l \leq p_{hr}$.

For the leaf node in the insertion path a new ground entry must be created together with filling its *PD* array by values $d(O_i, p_l), \forall l \leq p_{pd}$.

Node Splitting.
When a node is split, a new *HR* array of the left new routing entry is created by union of all appropriate intervals $HR[l]$ ($PD[l]$ in case of leaf splitting) stored in routing entries (ground entries respectively) of the left new node. A new *HR* array of the right new routing entry is created similarly.

3.4 Query Processing

Before processing any similarity query the distances $d(O_q, p_l), \forall l \leq max(p_{hr}, p_{pd})$ have to be computed. During a query processing the PM-tree hierarchy is being traversed down. Only if the metric region of a routing entry $rout(O_j)$ intersects the query region $(O_q, r(O_q))$, the covering subtree $T(O_j)$ may be relevant to the query and thus it is further processed. In case of a relevant PM-tree routing entry the query region must intersect all the hyper-rings stored in *HR*. Prior to the standard hyper-sphere intersection check (used by M-tree), the intersection of hyper-rings $HR[l]$ with the query region is checked as follows (note that no additional d computation is needed):

$$\bigwedge_{l=1}^{p_{hr}} (d(O_q, p_l) - r(O_q) \leq HR[l].max \ \wedge \ d(O_q, p_l) + r(O_q) \geq HR[l].min)$$

If the above hyper-ring intersection condition is false, the subtree $T(O_j)$ is irrelevant to the query and thus discarded from further processing. On the leaf level a relevant ground entry is determined such that the following condition must be satisfied:

$$\bigwedge_{l=1}^{p_{pd}} |d(O_q, p_l) - PD[l]| \leq r(O_q)$$

In Figure 4 an example of query processing is presented. Although the M-tree metric region cannot be discarded (see Figure 4a), the PM-tree region can be discarded since the hyper-ring $HR[2]$ is not intersected (see Figure 4b).

The hyper-ring intersection condition can be incorporated into the original M-tree range query as well as k-NN query algorithms. In case of range query the adjustment is straightforward – the hyper-ring intersection condition is combined with the original hyper-sphere intersection condition. However, the k-NN query algorithm (based on priority queue heuristics) must be redesigned. In the experiments we have considered range queries only – the design of a k-NN query algorithm for PM-tree is a subject of our future research.

3.5 Hyper-Ring Storage

In order to minimize storage volume of the *HR* and *PD* arrays in PM-tree nodes, a short representation of object-to-pivot distance is necessary.

We can represent a hyper-ring $HR[l]$ by two 4-byte reals and a pivot distance $PD[l]$ by one 4-byte real. When (a part of) the dataset is known in advance we can approximate the 4-byte distance representation by a 1-byte code. For this reason a distance distribution histogram is created by random sampling of objects from the dataset along with comparing them against all the pivots. Then a distance interval $\langle d_{min}, d_{max} \rangle$ is computed so that most of the histogram distances fall into the interval. See an example in Figure 5, where such an interval covers 90% of sampled distances (the d^+ value is an (estimated) maximum distance of a bounded metric space $\mathcal{M}$).

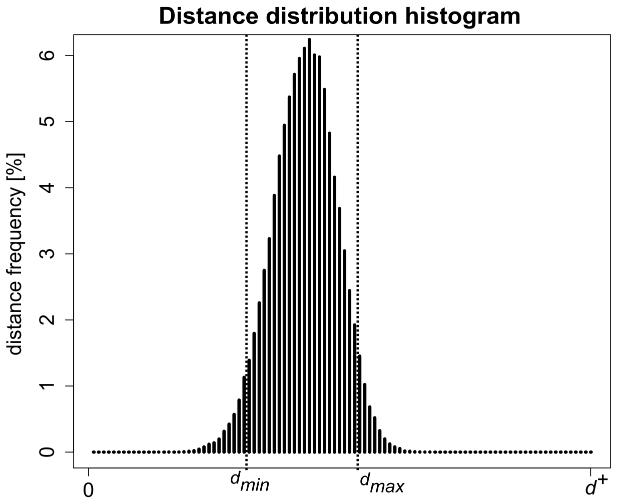


Fig. 5. Distance distribution histogram, 90% distances in interval $\langle d_{min}, d_{max} \rangle$

Values $HR[l]$ and $PD[l]$ are scaled into the $\langle d_{min}, d_{max} \rangle$ interval using a 1-byte code. Experimental results have shown that a 1-byte distance approximation is almost as effective as a 4-byte real while by using 1-byte approximation the PM-tree storage savings are considerable. As an example, for $p_{hr} = 50$ together with using 4-byte distances, the hyper-rings stored in an inner node having capacity 30 entries will consume $30 * 50 * 2 * 4 = 12000$ bytes while by using 1-byte distance codes the hyper-rings will take only $30 * 50 * 2 * 1 = 3000$ bytes.

4 Experimantal Results

We have made several preliminary experiments on a synthetic dataset of 250,000 10-dimensional vectors. The vectors were distributed within 2500 spherical clusters of a fixed radius (over the whole extent of the vector space domain). As a distance function the Euclidean metric (L_2) was used. Each label `PM-tree(x,y)` in the figures below stands for a PM-tree index where p_{hr} = x and p_{pd} = y. The sizes of PM-tree indices varied from 19MB (in case of PM-tree (0,0), i.e. M-tree) to 64MB (in case of PM-tree (100,100)). The PM-tree node size (disk page size respectively) was set to 4KB. For each index construction the `SingleWay + MinMax` techniques were used (we refer to [6]).

In the experiments a retrieval efficiency of range query processing was evaluated. The query objects were randomly selected from the dataset and each particular query test consisted of 200 range queries of the same query selectivity (the number of objects in query result). The results were averaged. Disk access costs (DAC) and computation costs of the query evaluation were examined, according to the number of pivots used (p_{hr} and/or p_{pd}) as well as according to query selectivity. The query selectivity was ranged from 5 to 50 objects. The experiments were intended to compare PM-tree with M-tree hence the PM-tree query costs are related to the costs spent by processing the same query by the M-tree index.

4.1 Disk Access Costs

In Figure 6a DAC according to query selectivity are presented. We can see that for querying `PM-tree(60,0)` index there is needed from 80% to 90% (increasing with selectivity) of DAC needed by the M-tree. The `PM-tree(200,0)` index is even more efficient since only 65% to 85% of DAC is needed. On the other side, `PM-tree(200,50)` index consumes up to 150% DAC since the long *PD* arrays (storing 50 pivot distances for each ground entry) cause the 250,000 ground entries must be stored in 9177 leafs (the M-tree needs only 4623 leafs).

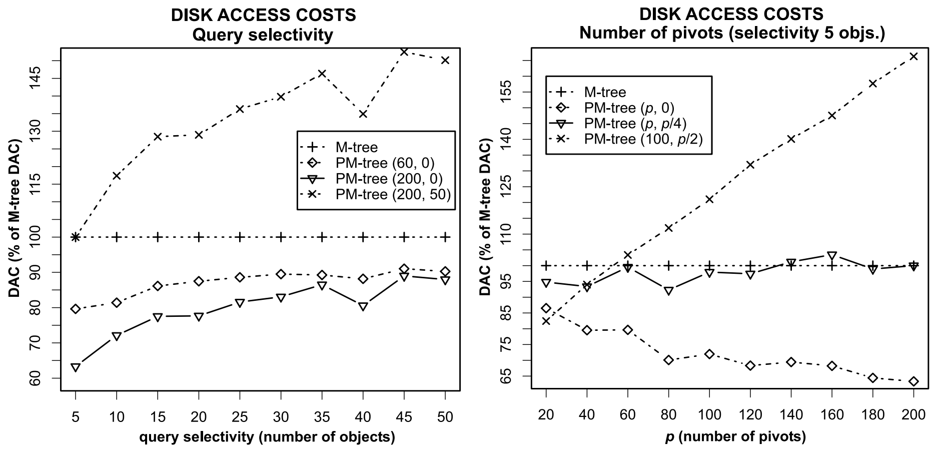


Fig. 6. Disk access costs: a) Query selectivity b) Number of pivots

Disk access costs according to the number of pivots are presented in Figure 6b. With the increasing p the disk access costs for `PM-tree(`p`,0)` indices decrease from 85% to 65% since more hyper-rings help to discard more irrelevant subtrees while the index sizes grow slowly (e.g. size of `PM-tree(200,0)` index is 24MB). The `PM-tree(100,`$p/2$`)` indices are more efficient than the M-tree for $p < 60$ only.

Interesting results are presented for `PM-tree(`p`,`$p/4$`)` indices where DAC remain about 100%. These results are better than for `PM-tree(100,`$p/2$`)` indices but worse than for `PM-tree(`p`,0)` indices. The reason for such behaviour is that with increasing p more hyper-rings help to discard more irrelevant subtrees but, on the other hand, due to the even longer PD arrays the index sizes grow quickly.

4.2 Computation Costs

Unlike for disk access costs, the increasing number of pivot distances in PD arrays positively affects the computation costs. In Figure 7a we can observe `PM-tree(200,50)` index to be more than 10 times as efficient as the M-tree index. However, for $p > 80$ the indices `PM-tree(`p`,`$p/4$`)` and `PM-tree(100,`$p/2$`)` consume the same computation costs (see Figure 7b). This happens particularly due to the increasing number of leafs which must be payed by a higher number of routing entries in inner nodes.

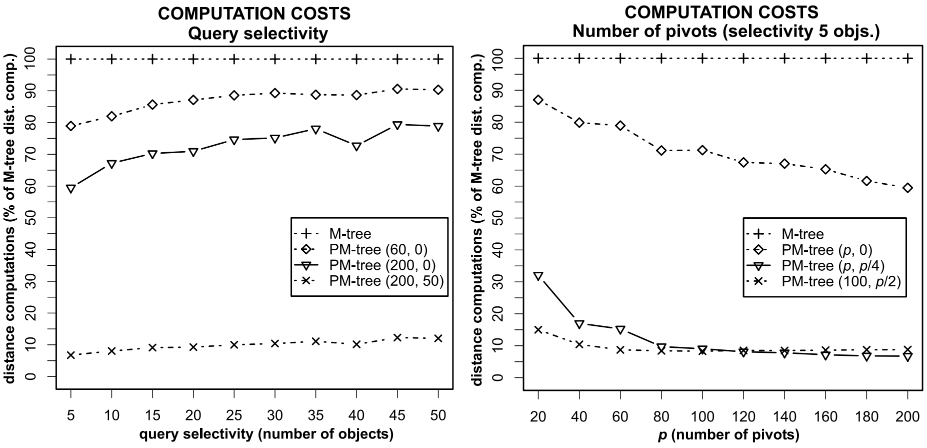


Fig. 7. Computation costs: a) Query selectivity b) Number of pivots

4.3 Summary

Based on the experimental results we are able to claim several facts (relative to the M-tree efficiency):

– For increasing p where $p_{hr} = p$ and $p_{pd} = 0$ the disk access costs as well as the computation costs steadily decrease.
– For increasing p where $p_{hr} \gg p_{pd}$ (say $p_{hr} = p, p_{pd} = \frac{p}{4}$) the disk access costs are similar to the M-tree DAC but the computation costs can be considerably lower. Such a behaviour can be useful when a distance computation is more expensive than a single disk access.
– In cases where $p_{hr} \leq p_{pd}$ the PM-tree behaviour acts similarly like the simple pivot-based filtering does since the disk access costs are high.

5 Conclusions and Outlook

In this paper the pivoting M-tree (PM-tree) was introduced. The PM-tree combines M-tree hierarchy of metric regions together with the idea of pivot-based methods. The result is a flexible metric access method providing even more efficient similarity search than the M-tree. The preliminary experimental results on a synthetic dataset indicate various efficiency trends for various PM-tree configurations.

In the future we plan to develop new PM-tree building algorithms exploiting the pivot-based information. Second, the original M-tree k-NN query algorithm has to be redesigned. Our next goal is formulation of a cost model making possible to tune PM-tree parameters for an estimated efficiency. Last but not least, extensive experiments on huge multimedia datasets have to be performed.

References

1. C. Böhm, S. Berchtold, and D. Keim. Searching in High-Dimensional Spaces – Index Structures for Improving the Performance of Multimedia Databases. *ACM Computing Surveys*, 33(3):322–373, 2001.
2. E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín. Searching in Metric Spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.
3. P. Ciaccia, M. Patella, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *Proceedings of the 23rd Athens Intern. Conf. on VLDB*, pages 426–435. Morgan Kaufmann, 1997.
4. L. Mico, J. Oncina, and E. Vidal. A new version of the nearest-neighbor approximating and eliminating search (aesa) with linear preprocessing-time and memory requirements. *Pattern Recognition Letters*, 15:9–17, 1994.
5. M. Patella. *Similarity Search in Multimedia Databases.* Dipartmento di Elettronica Informatica e Sistemistica, Bologna, 1999.
6. T. Skopal, J. Pokorný, M. Krátký, and V. Snášel. Revisiting M-tree Building Principles. In *ADBIS 2003, LNCS 2798, Springer-Verlag, Dresden, Germany*, 2003.

Storage and Retrieval of First Order Logic Terms in a Database

Peter Gurský

Department of Computer Science, Faculty of Science
P.J.Šafárik University Košice
Jesenná 9, 040 01, Košice
gursky@vk.science.upjs.sk

Abstract. In this paper we present a storage method for sets of first order logic terms in a relational database using function symbols based indexing method of Discrimination trees. This is an alternative method to a published one, based on attribute indexing. This storage enables effective implementation of several retrieval operations: unification, generalization, instantation and variation of a given query term in the language of first order predicate calculus. In our solution each term has unique occurrence in the database. This is very useful when we need to store a large set of terms that have identical many subterms.

Key words: first order logic terms, relational database storage and retrieval, first order logic term indexing

1 Introduction

A term in the alphabet of first order logic theory is defined inductively as follows: A variable or a constant is a term and if f is an n-ary function symbol and $t_1, ..., t_n$ are terms, then $f(t_1, ..., t_n)$ is a term. A constant can be sometimes consider as the 0-ary function symbol. In this paper the notion term has always this meaning. Please do no confuse it with other usage of the expression 'term' in information retrieval, digital libraries or other parts of computer science. Terms constitute the basic representational unit of information in several disciplines of computer science such as automated deduction, term rewriting, symbolic computing, logic and functional programming and inductive logic programming. Computation is done by operations such as, e.g., unification, generalization, instantation or variation. Often these operations are performed on large collections of terms. For instance, in logic programming, deductive databases, and theorem-proving for model elimination we need to select all candidate clause-heads in the program that unify with a given goal. In the absence of techniques for speeding up the retrieval of candidate terms, the time spent in identifying candidates may be overshadow the time spent in performing other useful computation. See [1].

In almost all programs that work with terms or sets of terms, there is a question how to store and retrieve[1] them effective. Majority of them take, at first, all the program and transform text representations of the terms, occurred in the program, to their internal structures in the main memory. If this program contains a large amount of terms, then this initialization can be expensive. On the other side, it is possible, that we do not have enough main memory. This leads to the problem of storing terms in persistent form on the disk. One solution is to use the relational database as the standard application to store a large amount of data that are related together. It is also the standard device for sharing data between the systems. Implementation described in this paper is based on theoretical results of [1] and [2] and provides an alternative solution for all of mentioned retrieval operations. We can store the sets of terms and quickly find the terms, that fulfills given requirement with the use of indexing technique called Discrimination trees. (See [1])

The paper is organized as follows: section 2 describes the problem of first order logic term indexing and explains some useful expressions. Section 3 provides deeper look on Discrimination trees indexing technique. Section 4 shows how to represent the structure of a term with the use of directed acyclic graph. In section 5 previously described structures are implemented in a relational database. Finally, section 6 mentions conclusions.

2 First order logic term indexing

The problem of first order logic term indexing by [1] can be formulated abstractly as follows. Given a set $\mathcal{L}$ (called the *set of indexed terms* or the *indexed set*), a binary relation R over terms (called the *retrieval condition*) and a term t (called the *query term*), identify the subset $\mathcal{M}$ of $\mathcal{L}$ consisting of all of the terms l such that $R(l,t)$ fulfills ($\mathcal{M} = \{l : R(l,t)\}$).

In some applications it is enough to search for a superset $\mathcal{M}'$ of $\mathcal{M}$ ($\mathcal{M}' \supseteq \mathcal{M}$), i.e., to also retrieve terms l for which $R(l,t)$ does not hold, but we would naturally like to minimize the number of such terms in order to increase the effectiveness of indexing. In information retrieval terminology we aim for a complete query answering with possibly lower precision which can be improved by additional search. When this happen, we say that indexing performs *imperfect filtering* in terminology of [1]. Retrieved terms, in this case, we will call *candidate terms.*

A *substitution* σ in a first order logic is a finite set of the form $\{v_1 \to t_1, ..., v_n \to t_n\}$, where each v_i is a variable, each t_i is a term distinct from v_i and the variables $v_1, ..., v_n$ are distinct.

Let $\sigma = \{v_1 \to t_1, ..., v_n \to t_n\}$ be a substitution and s be a term. Then $s\sigma$ is the term obtained from s by simultaneously replacing each occurrence of the variable v_i in s by the term t_i. We emphasize that it is important that

[1] In this paper the notion *retrieve* means, that we search for terms, that fulfill some kind of condition

replacements are done simultaneously. For example let $s = p(x, y, f(a))$ and $\sigma = \{x \rightarrow b, y \rightarrow x\}$, then $s\sigma = p(b, x, f(a))$.

In the context of term indexing, it is usually the case that the relation R of interest is such that $R(s, t)$ fulfills if there exists substitution σ and β such that $s\sigma = t\beta$, and furthermore, these substitutions satisfy certain additional constraints. In addition in order to identifying the terms that fulfills the retrieval condition, we sometimes need to compute the substitutions σ and β as well.

Under retrieval condition we will understand unification, generalization, instantation or variation. Given a query term t and indexed set $\mathcal{L}$, the retrieval operation is concerned with the identification of subset $\mathcal{M}$ of those terms in $\mathcal{L}$ that have specified relation R to t. The retrieval relation R identifies those terms $l \in \mathcal{L}$ that need to be selected. We will be interest in these retrieval conditions:

unif(l,t) $\Leftrightarrow \exists$ substitution σ: $l\sigma = t\sigma$;
inst(l,t) $\Leftrightarrow \exists$ substitution σ: $l = t\sigma$;
gen(l,t) $\Leftrightarrow \exists$ substitution σ: $l\sigma = t$;
var(l,t) $\Leftrightarrow \exists$ substitution σ: ($l\sigma = t$ and σ is a renaming substitution).

To understand these retrieval conditions we can make following example:

Let $l = f(x, g(a))$, $t = f(g(y), x)$, $s = f(g(b), g(a))$ and $u = f(g(x), y)$ where x and y are variables and a and b are constants.

Then unif(l,t) holds with substitution $\sigma = \{x \rightarrow g(a), y \rightarrow a\}$,
inst(s,l) holds with substitution $\sigma = \{x \rightarrow g(b)\}$,
gen(t,s) holds with substitution $\sigma = \{y \rightarrow b, x \rightarrow g(a)\}$
and var(t,u) holds with substitution $\sigma = \{x \rightarrow y, y \rightarrow x\}$.

On the other side e.g. unif(l,u), gen(s,l), inst(t,s) and var(t,s) do not hold, because there are no substitutions to fulfill the corresponding equality.

Thus, the retrieval condition is based on identification of a substitution between the query term and indexed terms, with various constraints placed on the substitution. The question of whether the retrieval condition holds between the query term and an indexed term is determined by the function symbols in both these terms. Thus, in every positions where both the query term and candidate term contain a function symbol, these symbols must be identical, because substitution does not change function symbols, it changes only variables. So we can make use of the function symbols in the indexed terms in determining the candidate terms. Most known term indexing techniques are based on this observation, and we refer to such techniques broadly as *function symbols based indexing*, or simply as *symbol-based indexing*. Representative of this techniques is also an indexing technique called Discrimination trees described in [1,2], that we will use below.

An alternative to symbol-based indexing is *attribute-based indexing*. In attribute-based indexing, we map som features of a term t into a simple-valued (say, integer-valued) attribute a_t. This solution is based on the assumption that a relation involving simple-valued attributes is much easier to compute than performing term matching or unification. However, according to [1] it has several disadvantages. Firstly, the precision of attribute-based indexing is typically low.

Second, if the index set is large, the coarse filter may still be inefficient as it may involve checking the retrieval relation from each term in a set. For details see [1,5].

In [5] authors use the attribute-based indexing for storing the terms in a relational database. Our solution is an alternative method with a use of more powerful filtering of symbol-based indexing also stored in a relational database.

3 Discrimination trees

Now, if we know that function symbols of the language of predicate calculus of first order logic in every position in the query term and the candidate term must be identical, we need some method to compare them. One solution is to construct a string of symbols from the query term, and identify a candidate term if this string matches the string constructed from the indexed terms. We can make these strings by writing out the symbols occurring in a term in some sequence. However, such an approach may lose some of the information captured by the term structure.

In discrimination trees we generate a single string (called the *path string* or the *p-string*) from each of indexed terms. These p-strings are obtained via a preorder traversal (left-to-right, depth-first direction) of the terms. To construct index structure we mount these p-strings in the index trie. The trie structure is described in [1]. We can see that there is a unique correspondence between the string obtained by preorder traversal (i.e. by writing out the symbols occurring on this traversal) and the text representation of a term. If we say that we will go through the terms only by preorder traversal, we can generate p-string very fast. If we have the only traversal, we don't need any added information about positions captured to function symbols, as it is in many other indexing techniques described in [1], e.g. Path indexing.

Except function symbols, and constants (that can be seen as function symbol with null arity), there are variables in terms too. This method speeds up finding of candidate terms by replacing variables by symbol *. This leads, of course, to imperfect filtering (lowers precision but preserves completeness), because it is impossible to search for substitutions or check out, if retrieval condition holds between the query and the indexed terms. We have to compute resultant substitutions, that is very expensive, after retrieval of candidate terms.

We can illustrate discrimination tree indexing using the example set of indexed terms and the p-strings generated from these terms.

{1}	f(g(a,*),c)	f.g.a.*.c
{2}	f(g(*,b),*)	f.g.*.b.*
{3}	f(g(a,b),a)	f.g.a.b.a
{4}	f(g(*,c),b)	f.g.*.c.b
{5}	f(*,*)	f.*.*

The retrieval of generalization of query term $f(g(a,c),b)$ from the indexed trie obtained from these p-strings is shown in Figure 1. To understand the process

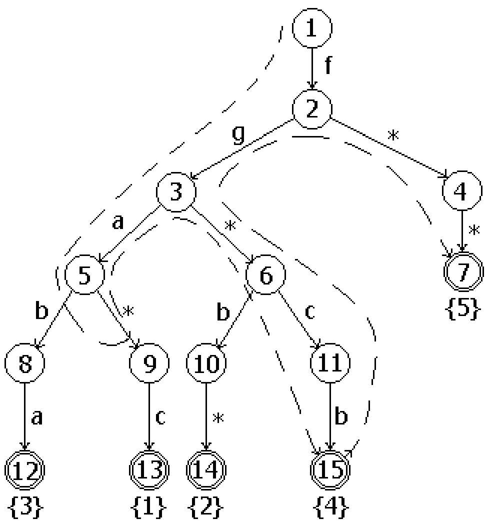


Fig. 1. Index and retrieval of generalizations of the query term *f(g(a,c),b)*

of indexing, note that the string corresponding to the query term is $f.g.a.c.b$. We compare the symbols in this p-string successively with the symbols on the edges in the path from the root to state 5. At this point, we cannot take the left path, as the symbol b on this edge conflicts with the symbol c in the query term. However, the symbol $*$ on the edge leading to state 9 is entirely plausible, since taking this edge corresponds finding a generalization (namely, a variable) of the subterm c of the query term. However, we cannot proceed further from state 9 because of constant c, so we have to backtrack to state 3. At this point, we can follow down the $*$ branch all the way down the final state 15, identifying candidate term 4. If we are interested in all generalizations, we have to backtrack further to state 2, and then finally follow to state 7, identifying candidate term 5.

In order to perform retrieval of unifiable terms and instances, we must efficiently deal with situations where the query term has a variable at a point where the indexed terms contain a function symbol. In such a case, we need a mechanism to efficiently skip the corresponding subterms in the indexed terms. It is also not trivial, witch part of p-string generated from query term, we need to skip, if we find $*$ on some edge of index.

To perform traversal of a term t we will need two operations on term positions explained in [1]: $next_t$ and $after_t$, witch can be informally explained as follows. Represent the term t as a tree and imagine a term traversal in the left-to-right, depth-first direction (i.e. preorder traversal). Suppose that s is a descendant of t and its (unique) position in the tree is p. Then $next_t(p)$ is the position of subterm of t visited immediately after s, and $after_t(p)$ is the position of subterm visited immediately after traversal of all subterms of s.

Figure 2 illustrates the behavior of next and after on the positions in the term $f(g(a,b),c)$, when we mark the position of the symbol t by Λ, the position of the symbol g by 1, a by 1.1, b by 1.2 and the position of the symbol c by 2. We also need a special object ε, that is representative of the "end position" in the term.

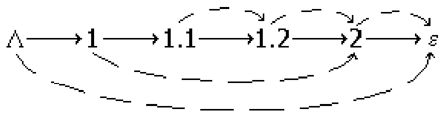


Fig. 2. $next_t$ and $after_t$ on the positions in term t = f(g(a,b),c). Solid straight lines represents $next_t$ and dashed lines represents $after_t$

When we have these two functions, it is easy to perform traversal through the query term during the retrieval operation. Thus, if we make traversal in the index trie and find the function symbol on the edge, we have to compare it with relevant function symbol in the query term, and if they match, we call the function $next_t$ to determine the next comparing position. If we find symbol * on the edge, we can call the function $after_t$ for the position of next comparing. Thus, we said that we can substitute the symbol * with all the subterm on this position, and next comparing will be with his next sibling in the query term tree or next symbol in preorder traversal after all this subterm.

Function $after_t$ in the case of query term we cannot use, of course, when the retrieval condition is instance, but we need mechanism similar to this function in the index trie. This requirement stands out also when we need to find unifiable terms. For this purpose we will use the structure named *Jump lists* from [1]. It can be seen that there must be an analogy with the function $after_t$.

We can make following example. We can add new term t = f(g(b,c),*) with its p-string f.g.b.c.* into the index trie in figure 1. Now imagine its traversal functions $next_t$ and $after_t$. Those structure is identical to that in Figure 2. It can be seen, that there is reciprocal corresponding between the function $next_t$ and the respective branch in the indexed trie. We can see added branch on Figure 3. In every state on this branch (except the last, that represents the end position ε) we will add a link to the states corresponding those positions in term t that determine the function $after_t$. In this case, we will add in the edge 1 the link to state 18, similar in state 2 to state 17, in state 3 to state 16, in state 16 to state 17 and in state 17 to state 18.

In Figure 3 we can see retrieval of terms unifiable with $f(g(b,*),a)$. Dashed lines are Jump lists only for those nodes where jump links go to a state different from the immediate child of a node, and except links from root to the leafes.

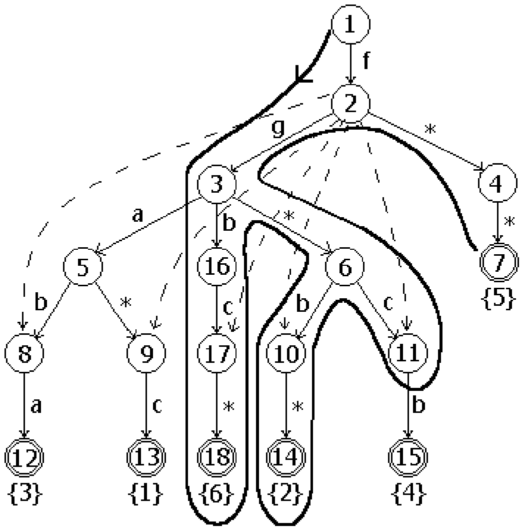


Fig. 3. Retrieval of terms unifiable with $f(g(b,*),a)$

All the structure of discrimination trees including index trie with jump lists we implement in a relational database in section 5.

4 Storing of terms

When we add a new first order logic term to the database, the standard input is the text-representation of this term and the name or the id of the set, where this term have to be a member. Storing only the pure text-representation of terms is not suitable. It is known that, term can be represented as a tree or *directed acyclic graph* (*DAG*) [1,2]. The root node of the tree representation of a term t contains the root symbol of t and pointers to nodes that correspond to immediate subterms of t. As compared to a tree representation, a DAG representation presents an opportunity to share subterms. We are trying to ensure that exists only a single copy of a term, regardless of number of contexts in which it occurs. This solution is called *perfect sharing* or *aggressive sharing* [2]. Such sharing can contribute to as much as an exponential decrease in the size of the term. The example in Figure 4 shows DAG representation of term $f(a, g(1, h(b), a), h(4, h(b)))$ with aggressive sharing.

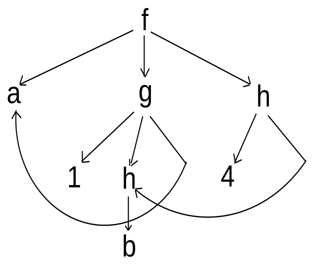


Fig. 4. DAG representation of $f(a, g(1, h(b), a), h(4, h(b)))$ with aggresive sharing

The benefits of sharing are significant in practice. It is very useful when we want, for example, to unificate two terms. There is known some unification algorithms like the *unification on term dags* or *an almost-linear algorithm* described in [2], that need on the input the structure contains two DAGs of the terms with shared subterms that we want to unificate. These algorithms are more powerful than any other unification algorithms based on any different structures. Those main advantage is that each substitution is computed only once, because there is always only one instance of each variable and after computing of any particular substitution this can be implement on the each of positions immediately at once and the substituted variable will never occur again.

When we do not share subterms, unification algorithm can be very inefficient. In the worst case, its running time can be an exponential function of the length of the input, for example when we want to unificate terms $s = p(x_1, ..., x_n)$ and $t = p(f(x_0, x_0), ..., f(x_{n-1}, x_{n-1}))$. For details see [2,5].

5 Implementation in relational database

In chapter 2 we have followed the informal description of the algorithms for retrieving of candidate terms and in previous chapter we have said how we can represent first order theory terms according to [1], [2] and [3]. In this chapter we present our method of storing terms in a relational database. The decomposition of data to the relational schema is shown in Table 1. For better notion of relationships between the tables, there is the database diagram on Figure 5.

For better understand of how to store and retrieve terms with use of this decomposition, we will show this on following example.

Imagine that we want to insert term t=$f(a, g(1, h(b), a), h(4, h(b)))$ where a and b are variables, whose DAG representation is shown in Figure 4. We also need to know the name or id of the set, of which this term have to be a member. For simplicity suppose that its name is $Set1$ with id 1 already stored in table SET, with root_state 1. The root_state is a number, that tell us, which state in index trie belong to this set is a root one. It is better to store for each set its

SET(id, name, root_state)
INSERTED(id, id_term, id_set)
TERM(id, id_symbol)
ATTRIBUTE(id_father, id_son, position)
SYMBOL(id, name, arity)
STATE(id, id_symbol, next)
JUMP(state, jump)

Table 1. Relational schema

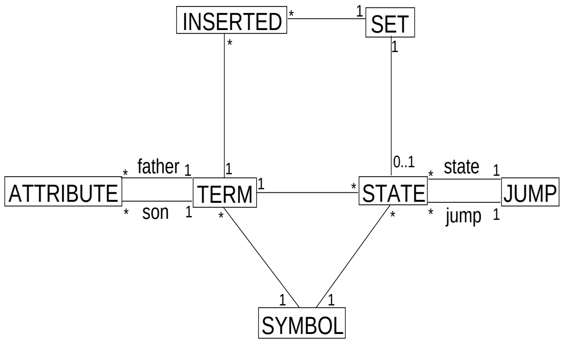


Fig. 5. Database diagram

own index trie. This solution is not suitable only if we want to retrieve always from all stored terms. In this case is the information about sets needless and, of course, we do not need table SET. In other cases our implementation allow to have a smaller index tries and faster retrieval from the individual sets of indexed terms. At first, we will point a look on storing of DAG representation of terms. An example can be seen in Table 2. For better understand of relations we do not start to generate id-s in table TERM from 1 but 11.

The symbols occurred in terms with their arities are stored in table SYMBOL. The column *arity* in the table SYMBOL is captured with the name, because it can really speed up retrieval from the index. Arity is also useful, when we can read all the structure of any term for the database. In this case we know, if we have to seek the table ATTRIBUTE for the subterms. Note that the arity -1 denotes a variable and the arity 0 denotes a constant.

The structure of the term is covered in table ATTRIBUTE. For example, as we can see in Table 2, the term with id 11 (whose text representation is symbol f) has as children the terms with *id*-s 12, 13 and 17 respectively. The order of these subterms is determined by the column *position*.

A expert fluent in databases would say that information in table TERM is redundant. It became useful when there is an identical function symbol with the same arity but with different subterms. In this case, if we have not the table

SYMBOL		
id	**name**	**arity**
1	f	3
2	a	-1
3	g	3
4	1	0
5	h	1
6	b	-1
7	h	2
8	4	0

TERM	
id	**id_symbol**
11	1
12	2
13	3
14	4
15	5
16	6
17	7
18	8

ATTRIBUTE		
id_father	**id_son**	**position**
11	12	1.
11	13	2.
11	17	3.
13	14	1.
13	15	2.
13	12	3.
15	16	1.
17	18	1.
17	15	2.

Table 2. Example of representation of inserted term $f(a, g(1, h(b), a), h(4, h(b)))$

TERM, we should have to add a new symbol as a row to the table SYMBOL that would have the same name and arity as already stored symbol, just with different *id*. For example if we want to add the term h(a), we can use fifth symbol, e.g. we add the couple (19,5) to the table TERM and triple (19,12,1) to the table ATTRIBUTE.

We can see, that we store only one instance of each subterm. We suppose that aggressive sharing is not concern only on individual terms, and there is only one instance of each subterm or term in all the database. Thus, if we add another term, that has any subterm identical to some subterm, that was stored before, we simply refer to the stored one. This happen also when this term is stored in another set.

There is one another table INSERTED, that was not mentioned. It stores the information, which term was original inserted (more precise it stores its root *id*) and to which set.

Now, when we have stored the DAG representation, we need to know, how we represent the index trie of Discrimination trees. As we have described in section 3, at first we make the p-string and add it as a new branch into the index trie. Structure of the branch from our example can be seen in Figure 6.

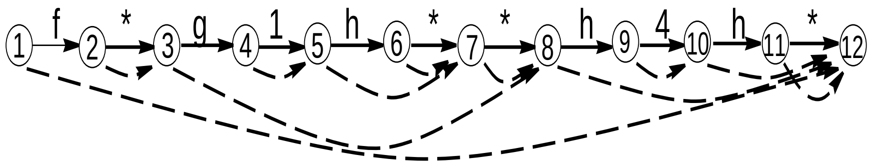


Fig. 6. Index representation of term $f(a, g(1, h(b), a), h(4, h(b)))$. Solid lines represents *next* and dashed *jump*

We can store this structure to the tables NEXT and JUMP as we can see in Table 3.

SYMBOL		
id	**name**	**arity**
1	f	3
2	a	-1
3	g	3
4	1	0
5	h	1
6	b	-1
7	h	2
8	4	0

STATE		
id	**id_symbol**	**next**
1	1	2
2	NULL	3
3	3	4
4	4	5
5	5	6
6	NULL	7
7	NULL	8
8	7	9
9	8	10
10	5	11
11	NULL	12
12	1	0

JUMP	
id_state	**id_jump**
1	12
2	3
3	8
4	5
5	7
6	7
7	8
8	12
9	10
10	12
11	12

Table 3. Representation of indexing structure for term $f(a, g(1, h(b), a), h(4, h(b)))$

The structure of these tables is, almost all, obvious from the Figure 6, when we say, that we are replacing a symbol * by NULL value for simpler differentiate between function symbols and variables with preserving of integer value of column *id_symbol*. The only difference between Figure 6 and Table 2 is on the last raw of table STATE. On this place we are making a trick, and saying, that when in the column *next* is value 0, than we are on leaf of index trie, and in the column *id_symbol* is the id of one of the terms, that are attached to this final state and stored under this id in table TERM or table INSERTED (vote is on the man, who want to implement it).

Now, when we are familiar in database structure, we can demonstrate, how to retrieve terms, that fulfill a retrieval condition. Let us have the text or DAG representation of a query term, retrieval condition and the name of set, that we want to seek for retrieval. At first, we need a list of symbols with arities equal to arities of the query term and its subterms. Then we must have a look to the table SYMBOL to get *id*-s of these symbols. If we do not find relevant rows for function symbols (with arity greater or equal to 0), than if the retrieval condition is instance or variation, we can say, that there is no relevant candidates in database. In other cases we can assign to such a symbols or variables unused *id*-s, e.g. negative numbers. Now we can seek indexed trie of given set of terms with root symbol registered in table SET. Traversal on this trie was described in section 3 with the difference that we do not match symbols but *id*-s and symbol * was substituted with NULL value. The function $next_t$ we easy can simulate with the use of table STATE and the function $after_t$ with the use of table JUMP.

On this part of enumeration we have two possibilities. We can wait for all the set of candidate terms or use the aspect of Discrimination trees, that we can receive candidates in sequence one by one. This allows to do next enumerations with the use of threading.

Further, it remains us to compute substitution for each couple of the query and the candidate term. We have to select all the structure of a candidate term from the database. One advantage is, that we obtain the structure, that answers the DAG representation with aggressive sharing. So we can use very fast algorithms as it was written in section 4. Finally, we have to delete those candidates, for which the substitution cannot be computed.

This solution of implementation of sets of terms in database provides a structures for storing and retrieval. This system is suitable primarily for applications, where retrieval performance is important and efficiency of maintenance operations is not a concern. If we want to insert or delete a branch from index trie we need to take care of shared states and jump lists. Similar, when we want to insert or delete a term from set (or more complicated from several sets) there is need to see if any subterm is a part of some other term or as a term is a member of a different set. In spite of that, there is a lot of programs, in which the sets of terms are almost static and the primary requirement is retrieval time.

6 Conclusions

In this paper we have presented a storage method for sets of first order logic terms in a relational database using Discrimination trees. Our solution is an alternative to a [5], based on attribute indexing. This storage enables effective implementation of retrieval operations unification, generalization, instantation and variation of a given query term. In our solution each term has unique occurrence in the database and can be easy converted to the DAG representation of terms. This provides very fast verifying of candidates returned from the index.

Acknowledgement. I would like to express my thanks to my master thesis supervisor RNDr. Peter Eliaš PhD.

Supported by project VEGA 1/0385/03.

References

1. R.Sekar, I.V.Ramakrishnan, Andrei Voronkov *Term indexing* in Alan Robinson, Andrei Voronkov *Handbook of automated reasoning.* Elsevier Science Publishers B.V. 2001
2. Franf Baader, Wayne Snyder *Unification theory* in Alan Robinson, Andrei Voronkov *Handbook of automated reasoning.* Elsevier Science Publishers B.V. 2001
3. Peter Gurský *Implementation of formal structures in database systems.* Master thesis under supervision of Peter Eliaš (in Slovak). Košice 2003.

4. J.W.Lloyd *Foundations of logic programming.* Springer-Verlag New York Berlin Heidelberg 1987. ISBN 3-540-18199-7, 0-387-18199-7
5. Paul Singleton, O.Pearl Brereton *Storage and retrieval of first-order terms using a relational database*, 1993. ISSN 1353-7776

Storing XML Data In a Native Repository

Kamil Toman

Dept. of Software Engineering
Charles University, Faculty of Mathematics and Physics
Malostranské náměstí 25
118 00 Praha 1
E-mail: `ktoman@ksi.mff.cuni.cz`

Abstract. This paper is concerned with storing XML data in a native repository suitable for querying with modern languages such as XPath or XQuery. It contains a description of the experimental database, SXQ-DB, its basic principles and system internals. Some of query evaluation techniques and problems related with those methods in relation to amount of stored information are mentioned.

1 Introduction

The XML language [1] was first published in 1998 but it has already become very popular. In the first place it is used as a standard for electronic interchange of application data and also as a flexible format allowing to store various information in a human readable form. With the expansion of Internet the data are gathered from various locations thus we cannot rely on their homogenity. On the contrary, we need to adapt applications to be able to handle them.

XML documents are logically formated documents which lessen the difference between pure text without any explicit formatting and rigidly structured data stored traditionally in relational databases.

Contents of XML documents are split up to smaller parts—elements—which are specifically named and which form one logical unit. From this point of view we can look on XML data as a database however it does not have a given hard-set structure and the structure of XML document itself provides a portion of complete information.

XML documents are often bound to their respective DTDs (*Document Type Definitions*). The purpose of DTD is to define the legal building blocks of an XML document. It defines possible structures together with a list of legal elements and attributes which might appear in the document. XML documents with a common DTD are called *document collections.*

To retrieve XML data from XML databases several new XML query languages have been proposed but only the minority survived. The most studied XML query languages with the most recent experimental implementations are XPath [6] and XQuery [5]. Both of them use *path expressions* as one of their basic constructs. Path expressions allow users to navigate through arbitrary paths of the XML tree and to address some portions of documents.

Despite many attempts to store XML data in relational, object-relational or object-oriented databases all existing approaches fail to supply sufficient functionalities to effectively manage and query XML data. Often, to process a simple path expression a sort of XML tree traversal is necessary. This might result in complex highly nested SQL queries which are hard to be effectively executed. Similarly in object-oriented databases, OQL is also not very suitable for expressing XML queries because it does not cover all basic constructs of XPath or XQuery.

In order to efficiently answer database queries the traditional DBMS leverage the usage of various indices. However, these index structures are tightly bound to a rigid database schema. That is something what, to some extent, prevents us from using them for XML indexing.

The database systems specially designed to store XML data are called *native XML repositories* or *native XML databases*. The storage is maximally adapted for tree shaped data contained in XML documents and as such it can be implemented in a practically arbitrary way. In comparison with traditional systems the index structures in native repositories are even more important. Often there is no way how to evaluate XML queries without a particular type of index. It does not, however, mean that a native XML database has to be implemented all over from the beginning. Some parts of such systems like the transaction manager, access control etc. can be often adopted from existing object or relational database systems with minimum changes.

In this text the new native XML database SXQ-DB (*Simple XQuery DataBase*) is described. In Section 2 the overall architecture and some implementation details of its XML storage module are discussed. Later on this section the query processing module is described. In Section 3 the brief overview how other systems process XML queries is presented. Conclusions summarize the contribution of this paper and give outlook to future work.

2 Native XML Database

SXQ-DB [3] is an experimental database suitable to store and manage collections of XML documents. Its current implementation consists of a native XML storage and a simple implementation of a non-trivial XML query language.

The goal of the work was to design a general and extensible architecture usable for testing various implementations of database operations and for verification basic qualities of the generic XML framework XMLCollection [2]. Unlike other projects the most important aspect was not the overall performance but the high-level design and the evaluation of the used data model with respect to more complex constructions of XML query languages. The accent was also on the fact that all accessed XML data were stored in the external memory.

As the XML query language has been implemented the language SXQ (*Simple XQuery*) which has been designed to cover the most important aspects of XQuery.

2.1 Overall Architecture

Due to given qualities and requirements of the initial XML framework the application operates on collections of XML documents characterized by a common DTD. These documents are stored in the external memory in a special binary format which is appropriate for more flexible access to data and hastens the effective successive processing.

The important decision on the implementation of the system was the choice of the modular architecture which allows an easy addition or a replacement of any system component.

Unlike programs managing XML files mostly in internal memory the module providing data querying is strictly separated from the actual XML repository. The Query Processing Module ensures syntactic analysis, processing and evaluating a query, the XML repository serves just for manipulation with persistent XML data. This design also allows the usage of several different XML storage modules without imposing other changes to the system. The overall architecture is depicted on Figure 1.

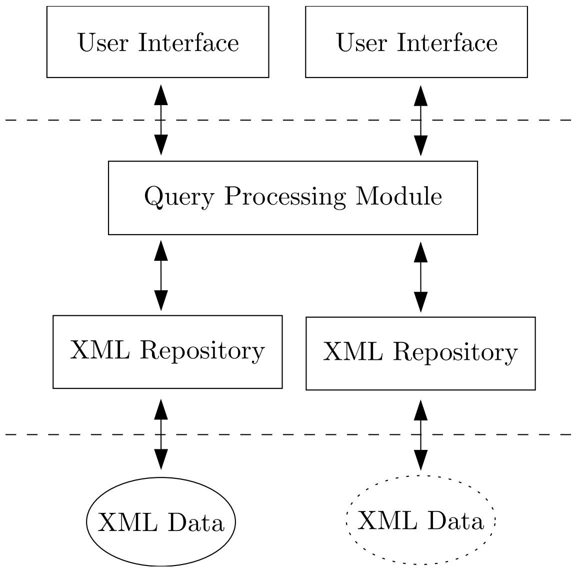


Fig. 1. Overall Architecture of SXQ-DB

2.2 XML Repository

The essential part of the SXQ-DB is the native XML repository implemented by utilizing the application framework XMLCollection. This module intermediates via an application interface information about data and structure of stored XML documents to other modules. It also provides access to individual elements and attributes stored in external memory.

The data repository also allows to limit the system resources—for example the size and number of system buffers or the maximal number of simultaneously accessed objects. Unlike other modules we put the accent on effectivity of data management.

Representation of XML Document. The used data model generally adheres the model as defined in XML Information Set [4] augmented by constructions defined by XQuery Data Model [5]. In our representation we use only those qualities which are required for evaluation of the SXQ language. In brief, we look on XML documents as oriented trees where to each vertex is associated a type of the node and a label. The vertices relevant to a common parent are ordered left-to-right. This ordering is imposed by the *global document order* requirement.

Text values are not assigned to all elements and attributes but only to special (artificial) nodes. This approach is primarily advantageous because all elements and attributes can be accessed the same way including elements with *mixed contents*.

Node Identification. In order to reasonably access individual nodes of XML tree we need to select a system of node identification. By a *numbering scheme* of the logical document model we understand a function which assigns a unique binary identifier (binary string) to each node of an XML tree. This identifier then can be used as a reference to the given node in an index or while a query evaluation.

The most common and also the simpliest numbering scheme in systems managing XML data is the *sequential* numbering scheme. The individual identifiers are assigned to particular nodes starting from one immediately after the nodes have been inserted to the system. The primary advantage of this scheme is its inherent simplicity and the maximal size of identifier which is at most $1 + \lfloor \log_2(n) \rfloor$ bits where n is number of nodes inserted to the system (including already deleted nodes).

The major disadvantage of the sequential numbering scheme is that it does not provide structural information – relations between nodes must be stored in separate.

The best way seems to be to leverage one of existing structural number schemes [7], [11], [14] which allow very effectively to determine the relation of arbitrary two nodes of the XML tree just from the information contained in their respective identifiers and thus they allow effective query evaluation using for example *structural joins* [11]. The maximum size of an identifier is still at most $2\lfloor 1 + \log_2(n) \rfloor$ bits.

On the other hand we have to take into account that we might need not only to change contents of individual XML nodes but also the document structure. Unfortunately all currently known structural numbering schemes are basically static – the numbering of nodes is based on the fact that we know or at least we assume the potential shape of the XML tree. If there are substantial differences from the anticipated XML data it is necessary to renumber the whole XML tree.

At the same time, neither contemporary techniques used for XML data management nor languages specially developed for XML document actualization like XUpdate [13] give us enough information about possible shapes of inserted subtrees or about scale of modifications to be done on the stored documents.

In general, it is possible to create a structural numbering scheme which is usable even in case when we know nothing about shapes of inserted trees. However, as proven in [14], the worst-case maximum size of resulting identifiers assigned to individual nodes is $O(n)$ bits.

Occasional renumbering of nodes in some XML subtrees does not imply an insurmountable problem because it can be partially prevented by a sensible utilization of structural statistics. Much bigger problem is that the renumbering of XML nodes imposes changes in practically all indexes that might be built up upon stored data. As XML indices are often relatively complex the total overhead related to their actualization might be enormous.

For this reason it is sensible to choose a compromise. For the direct numbering of nodes (in the core of XML repository) we use the basic sequential numbering scheme, plus we define a secondary numbering scheme (secondary identifiers) which will hold the structural information useful for quering.

In all indices we will use only references to primary node identifiers thus we avoid forced updates of indices if the structure changes. The disadvantage of this approach is slower evaluation of structural joins because of another level of logical mapping and also increased requirements for the disk space to manage primary and secondary identifiers.

Document Collections. We can also take advantage of the above approach to store the whole collection of XML documents at once. Because we use a simple sequential scheme as a primary identification mechanism the shape and the size of the tree does not pose a problem. Thus we can look on the whole collection as one XML document. It suffices to create two new special types of elements – the artificial root and document root elements. The abstraction of XML collection illustrates Figure 2.

The usage of the artificial collection root has one small additional advantage – it will always exist even if the collection is empty and can serve as a stable entry point with a fixed identifier.

Elements `DOCUMENT` allow mutual differentiation between individual XML documents and their attributes may also be used to keep user's information about stored documents like title, author, date etc. The information then can be later accessed even by the standard constructions of the query language.

Besides API simplification, the storage of the whole XML collection in one tree is often advantageous when the indices are built up. For example, if we create a word index it will be certainly more space efficient to build it up upon the whole collection than to build individual indices for every document in the

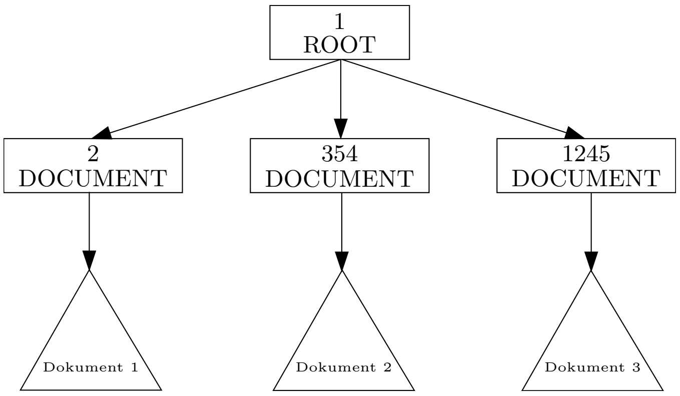


Fig. 2. Representation of XML Collection

collection. In case we want to query the whole collection at once it is also much more effective.

XML Node Types. Because we manage only XML documents with a common DTD we can also use numbers instead of labels for all types of elements and attributes (within different namespaces). The translation tables then can be stored together with DTD of the collection. This has two advantages: this approach to type identification is much more economical than storing text labels directly and it is also much easier and faster to use them during evaluation.

Architecture of XML Repository. The implemented XML repository can be logically divided into several modules. Each of them operates independently and ensures a different type of functionality:

– the DTD Storage module holds the DTD of the collection, name mapping tables, types and logical mapping of identifiers,
– the Element Storage module maintains the relations between XML nodes,
– the Value Storage module manages text values associated with elements and attributes,
– the rest of modules are mostly repository indices.

Strict module separation allows to hide unimportant details from the rest of the system. The communication between individual modules proceeds only via general application interface.

Naturally, this does not prevent us from using a common infrastructure. It is implemented as a separate module as well. The advantage is that potential changes in basic infrastructure can be done just once. The overview of the architecture of the XML repository module is demonstrated on Figure 3.

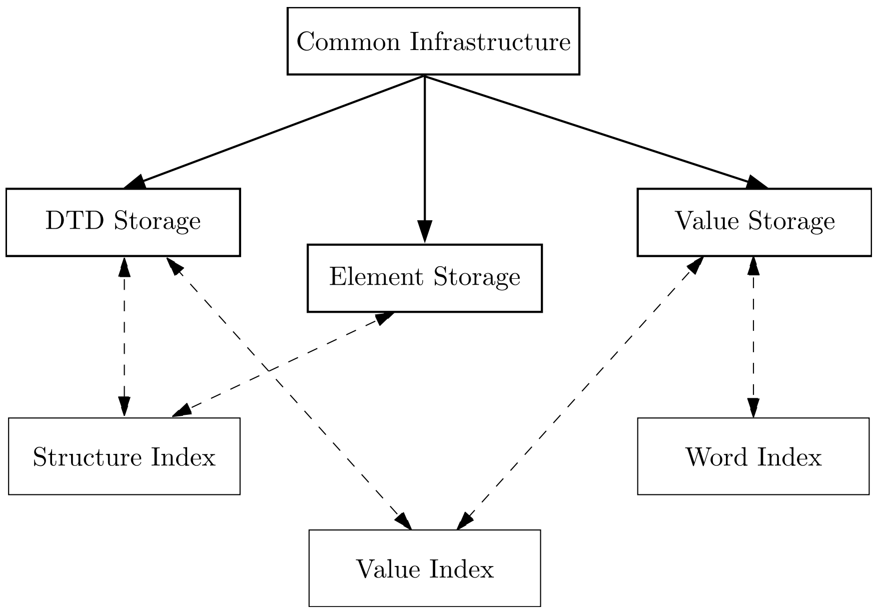


Fig. 3. Representation of XML Collection

The core modules cover only the basic functionality of the repository. For real utilization we have to take into consideration various indices as well. However, the indices might differ substantially from each other not only in implementation details but also by usage. It is hard to design a common interface which would allow a direct integration of an arbitrary index into the system.

To resolve this problem we expose only descriptions of events that may occur instead of detailed application interface between the core modules and indices – for instance a creation or a deletion of an XML node, its value actualization etc.

Each index is registered at particular modules so it can react to arisen events. That means we neither need to anticipate what data the index requires nor we need to know anything about its functionality. On the other hand we must ensure that the implementation of each index will be able to work with data structures of modules where it is registered.

Each index can propagate all events towards the rest of the modules which are built up on other than core modules of the system. Thanks to this mechanism we are able to sustain the whole system up-to-date.

Physical Access To External Memory. The important feature of the original implementation was to limit the system resources, the memory consumption in the first place. This property is ensured by the general *paging mechanism.*

Storage of Document Structure. The storage of a XML document structure is based on modeling local relations between XML nodes where every vertex "knows" only its direct neighbours. The representation of more complex relations is left to structural indices or to the query processing module.

The XML nodes are assigned indentifiers by the simple sequential numbering scheme and together with their types and identifiers of adjacent nodes are stored into fixed-length records in a binary file.

In order to access the nodes effectively we need to be able to quickly localize the information about individual nodes of the XML tree. To achieve this we index all records in a B$^+$-tree.

Secondary Object Cache. If there are some nodes which are accessed much more frequently during a query evaluation the above described method of locating nodes has still great overhead. The position of every located node must be at first looked up in the external index (possibly unbuffered) and then the respective position must be computed. The located page has to be loaded into main memory and requested data obtained from the computed offset.

For that reason a secondary object cache is implemented. Queries for information about an XML node are directed at first to this cache and only if it does not already contain the requested information the mechanism described above is used.

Notice that information about XML nodes is mostly short-lived. We often need to reach the record of the node just to find its neighbours or to check its value. If we implemented objects holding such information in a standard way frequent allocations and repeatedly released memory would kill the application performance. All cache objects are therefore kept in the main memory at all times and only if needed they are reinitialized with new data. The number of cache objects is given by the configuration of the XML repository module.

2.3 Query Processing Module

The XQuery language was created only recently and despite its basic features have their origin in previous proposals of XML query languages there is still no general technique how to evaluate all its queries. There is still nothing like relation algebra for classic relational database systems.

This is one of the reasons why classic navigational methods are still used for the evaluation of more general XML queries. More effective techniques like structural joins can be used only for special cases—mostly for path expressions and indispensable minimum of conditional expressions. Furthermore, expressions which can be evaluated by structural joins are very hard to distinguish from those which cannot be evaluated this way. Many of practical implementations avoid this problem simply by supporting only a limited set of XML queries [11], [20], [8]. The rest is modestly ignored.

Unlike other implementations our goal was to support all basic constructs of XPath and XQuery languages, not only path expressions. The implementation of SXQ language thus reminds a simple compiler of a general programming language. The architecture of the module and the individual phases of query processing is demonstrated on Figure 4.

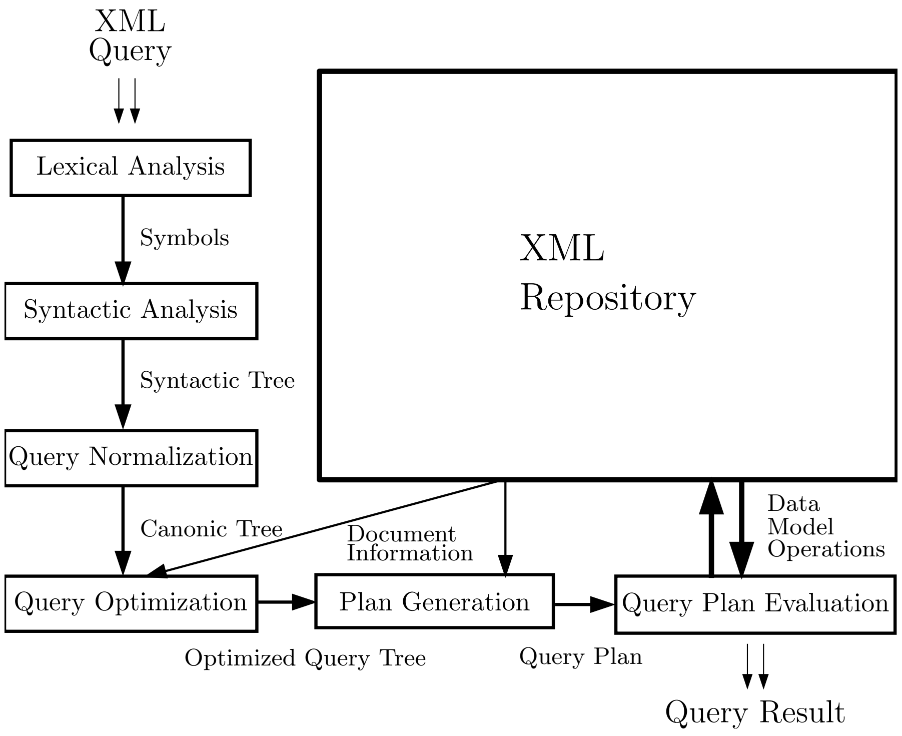


Fig. 4. SXQ Query Processing

At first the module disassembles the query to individual lexical elements which are subsequently used for syntactic analysis. In this phase context dependent keywords are resolved. In current implementation this is done via a finite automaton. The output of this stage is the syntactic tree which represents the independent form of a given query. However this tree is yet too complex for further processing. For that reason it is at first normalized into a *canonic tree.*

The canonic tree can be distilled from the syntactic tree by applying successive sequence of formal rewriting rules eliminating compound operations and “syntactic sugar”, i.e. operations which are not indispensable and can be equivalently described using basic operations. Part of a query normalization is also a unification of expressions—for instance, we can reorder some constructs to adhere a fixed form though formally the expressions do not depend on the order of terms. The important thing is that the canonic tree is semantically equivalent to syntactic tree but substantially simplified and also with rather more rigid structure. The canonic tree is more suitable for logical optimization, apart from other things, because by tree normalization we radically decrease the number of different shapes an XML query tree might have.

Similarly, the logical optimization usually constitutes of a set of rewriting rules. Unlike the previous ones the goal is not to simplify the structure of the canonic tree but to reduce the time needed for the query evaluation. The rules are often heuristic but the processing time generally should not be much longer if the conditions were mispredicted. As an example of such a logical optimization we can mention e.g. invariant motion (a separation and a movement of some parts of the query away from a repeatedly evaluated expressions) or constraint motion (evaluation of constraints and conditions as soon as possible). The overview of such rewriting rules can be found in [12]. This phase might also include the elimination of common subexpressions of the query.

The logically optimized query tree is then passed to a generator of query plans. This module constructs possible procedures of query evaluation and accordingly to information supplied by XML repository it chooses the optimal plan.

This plan of query evaluation is consecutively executed by the computation engine which makes up the result of the query.

3 Query Processing In Other XML Database Systems

Tree pattern queries or correlated path expressions are the most accented constructs of XPath and XQuery querying languages. A pattern trees representing parent-child, ancestor-descendant relations between XML nodes bound with some additional constraints are to be matched against a source XML tree or a XML document collection.

The currently used evaluation techniques use extensive indices built mostly as combinations of structural path summaries [15], value indexing and tree traversal (Lore [16]) or identifier schemes (XISS [11]). However the storage efficiency is often not considered in these approaches.

Earlier systems relied on tree traversal techniques and structural indices like DataGuides or T-indices which are very inefficient when they are stored in the external memory. These methods have been surpassed with more modern structural joins (XISS, eXist [20]) which compose the tree patterns by pairwise matching parent-child and ancestor-descendant relations between candidate XML nodes. However the most commonly used indices used for structural joins can generally exceed the size of the whole source XML tree not giving any additional information besides the transitive ancestor-descendant relationship [17].

A few other indexing schemes like SphinX [18] or APEX [19] reduce the size of resulting indices by deliberately not covering all necessary information at the expense of generality or guaranteed performance. Though in practice they may perform quite well.

A novel approach of processing XML queries is being developed for project Timber [9] which is based on a complete and closed algebra named TAX which is a generalization of the current relational algebra for tree structures. The project still uses the old object manager Shore to manage the XML persistence but a transition to a native XML repository Natix [10] is planned.

4 Conclusions and Future Work

In this text, we described concepts and the implementation of SXQ-DB, the experimental native XML database. We demonstrated some advantages of its modular architecture and showed the basic data flow in the system. We also outlined some problems concerned with XML node insertions, numbering schemes and XML query evaluations, the tree pattern matching queries in the first place.

Future work in this area should probably be focused on two things: to find a more general way how to express and evaluate the most common XML queries and also to reduce space needed for structural and term indices used by the database application. Some recent more advanced proposals of XML indexing like multidimensional trees and UB-trees [21] are also subjects to be studied.

References

1. XML CoreWorking Group: Extensible Markup Language (XML). (2000) `http://www.w3.org/XML/`
2. M. Kopečný: Implementan prosted pro kolekce XML dat. Thesis (In Czech), MFF UK (2002)
3. K. Toman: XML data na disku jako databáze. Thesis (In Czech), MFF UK (2003)
4. J. Cowan, R. Tobin: XML Information Set. (2001) `http://www.w3.org/TR/xml-infoset`
5. M. Marchiori: XML Query Specifications. (2003) `http://www.w3.org/XML/Query#specs`
6. J. Clark, S. DeRose: XML Path Language (XPath) Version 1.0. (1999) `http://www.w3.org/TR/xpath`
7. P. F.'Dietz: Maintaining order in a linked list. Proc. of the Fourteenth Annual ACM Symposium on Theory of Computing: 122-127. (1982)
8. A. Sahuguet: Kweelt, the Making-of Mistakes Made and Lessons Learned. Technical report, Department of Computer and Science, University of Pensylvania. (2000)
9. H. V. Jagadish, S. Al-Khalifa, A. Chapman, L. V. S. Lakshmanan, A. Nierman, S. Paparizos, J. M. Patel, D. Srivastava, N. Wiwatwattana, Y. Wu, C. Yu: TIMBER: A Native XML Database. (2002) `http://www.eecs.umich.edu/db/timber`
10. C. Ch. Kanne, G. Moerkotte: Efficient Storage of XML Data. Poster abstract in Proc. ICDE: 198. (2000)
11. Q. Li, B. Moon: Indexing and Querying XML Data for Regular Path Expressions. VLDB Conference: 361-370 (2001)
12. M. Grinev, S. Kuznetsov: Towards an Exhaustive Set of Rewriting Rules for XQuery Optimization: BizQuery Experience Advances in Databases and Information Systems (2002)
13. XUpdate Working Group: XUpdate – XML Update Language. (2003) `http://www.xmldb.org/xupdate/`

14. E. Cohen, H. Kaplan, T. Milo: Labeling Dynamic XML Trees. Symposium on Principles of Database System (PODS): 271-281 (2002)
15. R. Goldman, J. Widom: DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. VLDB Conference (1997)
16. J. McHugh, J. Widom, S. Abiteboul, Q. Luo, A. Rajaraman: Indexing Semistructured Data. Technical report, Stanford University (1999)
17. C. Zhang, G. He, D. J. DeWitt, J. F. Naughton: On supporting containment queries in relational database management systems. In SIGMOD International Conference on Management of Data: 425-436 (2001)
18. L. K. Poola, J. R. Haritsa: Schema-consious XML Indexing. Indian Institute of Science, Dept. of Computer Science & Automation (2001)
19. Ch.-W. Chung, J.-K. Min, K. Shim: APEX: An Adaptive Path Index for XML data. ACM SIGMOD (2002)
20. W. Meier: eXist: An Open Source Native XML Database. (2002) `http://exist-db.org`
21. M. Krátký, J. Pokorný, V. Snášel. Indexing XML Data with UB-Trees. ADBIS (2002)

Concept Lattices Constrained by Attribute Dependencies

Radim Bělohlávek, Vladimír Sklenář, Jiří Zacpal

Dept. Computer Science, Palacký University, Tomkova 40, CZ-779 00, Olomouc, Czech Republic, radim.belohlavek@upol.cz

Abstract. The input data to formal concept analysis consist of a collection of objects, a collection of attributes, and a table describing a relationship between objects and attributes (so-called formal context). Very often, there is an additional information about the objects and/or attributes available. In the analysis of the data, the additional information should be taken into account.
We consider a particular form of the additional information. The information is in the form of particular attribute dependencies. The primary interpretation of the dependencies is to express a kind of relative importance of attributes. We introduce the notion of a formal concept compatible with the attribute dependencies. The main gain of considering only compatible formal concepts and disregarding formal concepts which are not compatible is the reduction of the number of resulting formal concepts. This leads to a more comprehensible structure of formal concepts (clusters) extracted from the input data. We illustrate our approach by examples.

Keywords: formal context, formal concept, concept lattice, clustering, constraint, attribute dependency

1 Introduction and problem setting

Finding interesting patterns in data has traditionally been a challenging problem. Particular attention has been paid to discovering interesting clusters in data. Recently, there has been a growing interest in so-called formal concept analysis (FCA) [4] which provides methods for finding patterns and dependencies in data which can be run automatically. The patterns looked for are called formal concepts. Both foundations and applications (classification, software (re)engineering, document and text organization, etc.) of formal concept analysis are documented (see [4] and [1], and the references therein).

The central notion of all clustering methods is that of a cluster. Clusters are supposed to be meaningful pieces of data which are cohesive in some way. To have a good notion of a cluster, one should exploit all the information about the data available which can contribute to identification of meaningful clusters.

Formal concept analysis deals with input data in the form of a table with rows corresponding to objects and columns corresponding to attributes which describes a relationship between the objects and attributes. The data table is formally represented by a so-called formal context which is a triplet $\langle X, Y, I\rangle$ where I is a binary relation between X and Y, $\langle x, y\rangle \in I$ meaning that the object x has the attribute y. For each $A \subseteq X$ denote by $A^{\uparrow}$ a subset of Y defined by

$$A^{\uparrow} = \{y \mid \text{for each } x \in X : \ \langle x, y\rangle \in I\}.$$

Similarly, for $B \subseteq Y$ denote by $B^{\downarrow}$ a subset of X defined by

$$B^{\downarrow} = \{x \mid \text{for each } y \in Y : \ \langle x, y\rangle \in I\}.$$

That is, $A^{\uparrow}$ is the set of all attributes from Y shared by all objects from A (and similarly for $B^{\downarrow}$). A formal concept in $\langle X, Y, I\rangle$ is a pair $\langle A, B\rangle$ of $A \subseteq X$ and $B \subseteq Y$ satisfying $A^{\uparrow} = B$ and $B^{\downarrow} = A$. That is, a formal concept consists of a set A of objects which fall under the concept and a set B of attributes which fall under the concept such that A is the set of all objects sharing all attributes from B and, conversely, B is the collection of all attributes from Y shared by all objects from A. This definition formalizes the traditional approach to concepts which is due to Port-Royal logic [2]. The sets A and B are called the extent and the intent of the concept $\langle A, B\rangle$, respectively. The set $\mathcal{B}(X, Y, I) = \{\langle A, B\rangle \mid A^{\uparrow} = B, B^{\downarrow} = A\}$ of all formal concepts in $\langle X, Y, I\rangle$ can be naturally equipped with a partial order $\leq$ defined by

$$\langle A_1, B_1\rangle \leq \langle A_2, B_2\rangle \text{ iff } A_1 \subseteq A_2 \text{ (or, equivalently, } B_2 \subseteq B_1).$$

That is, $\langle A_1, B_1\rangle \leq \langle A_2, B_2\rangle$ means that each object from A_1 belongs to A_2 (or, equivalently, each attribute from B_2 belongs to B_1). Therefore, $\leq$ models the natural subconcept-superconcept hierarchy under which dog is a subconcept of mammal.

The structure of $\mathcal{B}(X, Y, I)$ is described by the so-called main theorem of concept lattices [4,6].

Theorem 1. *(1) The set $\mathcal{B}(X, Y, I)$ is under $\leq$ a complete lattice where the infima and suprema are given by*

$$\bigwedge_{j\in J} \langle A_j, B_j\rangle = \langle \bigcap_{j\in J} A_j, (\bigcup_{j\in J} B_j)^{\downarrow\uparrow}\rangle \ , \bigvee_{j\in J} \langle A_j, B_j\rangle = \langle (\bigcup_{j\in J} A_j)^{\uparrow\downarrow}, \bigcap_{j\in J} B_j\rangle \ . \quad (1)$$

(2) Moreover, an arbitrary complete lattice $\mathbf{V} = \langle V, \leq\rangle$ is isomorphic to $\mathcal{B}(X, Y, I)$ iff there are mappings $\gamma : X \to V$, $\mu : Y \to V$ such that

(i) $\gamma(X)$ is $\bigvee$-dense in V, $\mu(Y)$ is $\bigwedge$-dense in V;
(ii) $\gamma(x) \leq \mu(y)$ iff $\langle x, y\rangle \in I$.

In the basic setting of formal concept analysis, no further information except for $\langle X, Y, I\rangle$ is taken into account. However, more often than not, both the set of objects and the set of attributes are supplied with an additional information. Further processing of the input data (formal context) should therefore take the additional information into account. For example, some attributes may be relevant (or relevant to some degree) with respect to a particular kind of decisions while some may be not. When processing a respective formal context in order to get some support for the decisions in question, the attributes which are not relevant to the decision may be disregarded. In the end, this may result in a simplification of the overall processing.

In this paper, we consider additional information which has the form of formulas (so called AD-formulas) describing particular dependencies between attributes expressing their relative importance. We introduce the notion of a formal concept compatible with an AD-formula. This enables us to eliminate formal concepts which are not compatible with the information about the relative importance of attributes. An important effect of the elimination is a natural reduction of the size of the resulting conceptual structure making the structure more comprehensible. This paper extends in a natural way our previous approach [3] in that the constraints expressible by AD-formulas are more general and thus more expressive than those of [3]. Such an extension is needed, as we discuss in the text and show by examples.

2 Constraints by attribute dependencies

Basic motivation When people categorize objects by means of the object attributes, they naturally take into account the importance of attributes. Usually, attributes which are less important are not used to form large categories (clusters, concepts). Rather, less important attributes are used to make a finer categorization within a larger category. For instance, consider a collection of certain products offered on a market, e.g. home appliances. When categorizing home appliances, one may consider several attributes like price, the purpose of the appliance, the intended placement of the appliance (kitchen appliance, bathroom appliance, office appliance, etc.), power consumption, color, etc. Intuitively, when forming appliance categories, one picks the most important attributes and forms the general categories like "kitchen appliances", "office appliances", etc. Then, one may use the less important attributes (like "price $\leq$ \$10", "price between \$15–\$40", "price $>$ \$100", etc.) and form categories like "kitchen appliance with price between \$15–\$40". Within this category, one may further form finer categories distinguished by color. This pattern of forming categories follows the rule that when an attribute y is to belong to a category, the category must contain an attribute which determines a more important characteristic of the attribute (like "kitchen appliance" determines the intended placement of the appliance). This must be true for all the characteristics that are more important than y. In this sense, the category "red appliance" is not well-formed since color is considered less important than price and the category "red appliance" does not contain any information about the price. Which attributes and characteristics are considered more important depends on the particular purpose of categorization. In the above example, it may well be the case that price be considered more important that the intended placement. Therefore, the information about the relative importance of the attributes is to be supplied by an expert (the person who determines the purpose of the categorization). Once the information has been supplied, it serves as a constraint for the formation of categories. In what follows, we propose a formal approach to the treatment of the above-described constraints to formation of categories.

Constraints by attribute-dependency formulas Consider a formal context $\langle X, Y, I\rangle$. We consider constraints expressed by formulas of the form

$$y \sqsubseteq y_1 \sqcup \cdots \sqcup y_n. \tag{2}$$

Formulas of this form will be called AD-formulas (attribute-dependency formulas). The set of all AD-formulas will be denoted by ADF. Let now $\mathcal{C} \subseteq ADF$ be a set of AD-formulas.

Definition 1. A formal concept $\langle A, B\rangle$ satisfies an AD-formula (2) if we have that

$$\text{if } y \in B \text{ then } y_1 \in B \text{ or } \cdots \text{ or } y_n \in B.$$

The fact that $\langle A, B\rangle \in \mathcal{B}(X, Y, I)$ satisfies an AD-formula φ is denoted by $\langle A, B\rangle \models \varphi$. Therefore, $\models$ is the basic satisfaction relation (being a model) between the set $\mathcal{B}(X, Y, I)$ of all formal concepts (models, structures) and the set ADF of all AD-formulas (formulas).

As usual, $\models$ induces two mappings, $\mathrm{Mod} : 2^{ADF} \to 2^{\mathcal{B}(X,Y,I)}$ assigning a subset

$$\mathrm{Mod}(\mathcal{C}) = \{\langle A, B\rangle \in \mathcal{B}(X, Y, I) \mid \langle A, B\rangle \models \varphi \text{ for each } \varphi \in \mathcal{C}\}$$

to a set $\mathcal{C} \subseteq ADF$ of AD-formulas, and $\mathrm{Fml} : 2^{\mathcal{B}(X,Y,I)} \to 2^{ADF}$ assigning a subset

$$\mathrm{Fml}(U) = \{\varphi \in ADF \mid \langle A, B\rangle \models \varphi \text{ for each } \langle A, B\rangle \in U\}$$

to a subset $U \subseteq \mathcal{B}(X, Y, I)$.

The following result is immediate [5].

Theorem 2. *The mappings* Mod *and* Fml *form a Galois connection between* ADF *and* $\mathcal{B}(X, Y, I)$*. That is, we have*

$$\mathcal{C}_1 \subseteq \mathcal{C}_2 \textit{implies} \mathrm{Mod}(\mathcal{C}_2) \subseteq \mathrm{Mod}(\mathcal{C}_1), \tag{3}$$

$$\mathcal{C} \subseteq \mathrm{Fml}(\mathrm{Mod}(\mathcal{C})), \tag{4}$$

$$U_1 \subseteq U_2 \textit{implies} \mathrm{Fml}(U_2) \subseteq \mathrm{Fml}(U_1), \tag{5}$$

$$U \subseteq \mathrm{Mod}(\mathrm{Fml}(U)). \tag{6}$$

for any $\mathcal{C}, \mathcal{C}_1, \mathcal{C}_2 \subseteq ADF$*, and* $U, U_1, U_2 \subseteq \mathcal{B}(X, Y, I)$*.*

Thus, more generally, for $U \subseteq \mathcal{B}(X, Y, I)$ and $\mathcal{C} \subseteq ADF$ we write $U \models \mathcal{C}$ if $U \subseteq \mathrm{Mod}(\mathcal{C})$ which is equivalent to $\mathcal{C} \subseteq \mathrm{Fml}(U)$ (the meaning: each $\langle A, B\rangle \in U$ satisfies each $\varphi \in \mathcal{C}$).

Definition 2. For $\mathcal{C} \subseteq ADF$ we put

$$\mathcal{B}_{\mathcal{C}}(X, Y, I) = \mathrm{Mod}(\mathcal{C})$$

and call it the *constrained (by $\mathcal{C}$) concept lattice* induced by $\langle X, Y, I\rangle$ and $\mathcal{C}$.

For simplicity, we also denote $\mathcal{B}_{\mathcal{C}}(X, Y, I)$ simply by $\mathcal{B}_{\mathcal{C}}$. That is, $\mathcal{B}_{\mathcal{C}}(X, Y, I)$ is the collection of all formal concepts from $\mathcal{B}(X, Y, I)$ which satisfy each AD-formula from $\mathcal{C}$ (satisfy all constraints from $\mathcal{C}$).

Note that (3)–(6) have a natural interpretation. For instance, (3) says that the more formulas we put to $\mathcal{C}$ (the more constraints), the fewer formal concepts are in $\mathcal{B}_{\mathcal{C}}$.

Remark 1. (1) In [3], we introduced constraints by a hierarchy on Y which is represented by a partial order $\trianglelefteq$ on Y. A formal concept $\langle A, B\rangle \in \mathcal{B}(X, Y, I)$ is called compatible with $\trianglelefteq$ if for each $y \in B$ and $y \trianglelefteq y'$ we have $y' \in B$. Denote $\mathcal{B}(X, \langle Y, \trianglelefteq\rangle, I)$ the set of all formal concepts from $\mathcal{B}(X, Y, I)$ which are compatible with $\trianglelefteq$. It is clear that putting $\mathcal{C}_{\trianglelefteq} = \{y_1 \sqsubseteq y_2 \mid \langle y_1, y_2\rangle \in \trianglelefteq\}$, we have $\mathcal{B}(X, \langle Y, \trianglelefteq\rangle, I) = \mathcal{B}_{\mathcal{C}_{\trianglelefteq}}(X, Y, I)$. This way our current approach generalizes that one of [3].

(2) Note that our present approach is needed. For instance, if y, y_1, and y_2 stand for "price $>$ \$100", "kitchen appliance", and "office appliance", respectively, then $y \sqsubseteq y_1 \sqcup y_2$ represents a natural constraint which cannot be directly expresses by a hierarchy $\trianglelefteq$ in the sense of [3].

In the rest of this section we briefly discuss selected topics related to constraints by AD-formulas. Due to the limited scope, we omit details.

Structure of $\mathcal{B}_{\mathcal{C}}(X, Y, I)$ Contrary to [3], we lose some nice properties under the present approach. For example, although $\mathcal{B}_{\mathcal{C}}(X, Y, I)$ is a partially ordered subset of $\mathcal{B}(X, Y, I)$, it does no need not be a sup-sublattice of $\mathcal{B}(X, Y, I)$ as is the case of $\trianglelefteq$.

Example 1. Let $X = \{x_1, x_2\}$, $Y = \{y_1, y_2, y_3\}$, $I = \{\langle x_1, y_2\rangle, \langle x_1, y_3\rangle, \langle x_2, y_1\rangle, \langle x_2, y_3\rangle\}$, $\mathcal{C} = \{y_3 \sqsubseteq y_1 \sqcup y_2\}$. Then $\mathcal{B}_{\mathcal{C}}$ is not a sup-sublattice of $\mathcal{B}(X, Y, I)$.

Entailment of AD-formulas Another interesting issue, in fact, a very important one, is that of entailment of AD-formulas (i.e. the notion of entailment of an AD-formula by a set of AD-formulas). Namely, AD-formulas which follow from $\mathcal{C}$ may be ignored because do not represent any additional constraint. Conversely, it might be interesting to look for a base of a set $\mathcal{C}$ of AD-formulas, i.e. a subset $\mathcal{C}' \subseteq \mathcal{C}$ such that each $\varphi \in \mathcal{C}$ follows from $\mathcal{C}'$ and $\mathcal{C}'$ is a minimal one with this property. Due to the limited scope, we omit any further details.

Expressive power of AD-formulas A given $y \in Y$ may occur on left hand-side of several AD-formulas. For example, we may have $y \sqsubseteq y_1 \sqcup y_2$ and $y \sqsubseteq y_3 \sqcup y_4$. Then, for a formal concept $\langle A, B\rangle$ to be compatible, it has to satisfy the following: whenever $y \in B$ then it must be the case that $y_1 \in B$ or $y_2 \in B$, and $y_3 \in B$ or $y_4 \in B$. Therefore, it is tempting to allow for expressions of the form

$$y \sqsubseteq (y_1 \sqcup y_2) \sqcap (y_3 \sqcup y_4)$$

with the intuitively clear meaning of compatibility of a formal concept and a formula of this generalized form. Note that a particular form is also e.g. $y \sqsubseteq y_2 \sqcap y_3$. One may also want to extend this form to formulas containing disjunctions of conjunctions, e.g.

$$y \sqsubseteq (y_1 \sqcap y_2) \sqcup (y_3 \sqcap y_4).$$

It is not difficult, however, somewhat tedious, to show that the expressive power of such generalized formulas remains the same. More precisely, to each set $\mathcal{C}$ of generalized formulas there exists a set $\mathcal{C}'$ of ordinary AD-formulas such that for each formal concept $\langle A, B\rangle$ we have that $\langle A, B\rangle \models \mathcal{C}$ iff $\langle A, B\rangle \models \mathcal{C}'$.

3 Examples

We now present illustrative examples. We assume that the reader is familiar with Hasse diagrams which will be used for visualization of concept lattices and attribute hierarchies. We label the nodes corresponding to formal concepts by boxes containing concept descriptions. For example, $(\{1, 3, 7\}, \{3, 4\})$ is a description of a concept the extent of which consists of objects 1, 3, and 7, and the intent of which consists of attributes 3 and 4.

Example 2. **Using attribute dependencies for generation of views on databases.** Suppose we have a relational database with particular car models as objects and selected car properties as attributes. We have attributes like "hatchback", "sedan", "diesel engine", "gasoline engine", "air-conditioning", "ABS", etc. This data can be understood as a (bivalent) formal context. This context induces a corresponding concept lattice containing all formal concepts hidden in the database. In general, this concept lattice contains a large number of formal concepts. This fact makes the concept lattice not comprehensible by humans. With respect to a particular aim (e.g. a decision making), the concept lattice will contain both important and natural concepts as well as concepts which are considered not important.

To get a more precise idea, suppose a customer wants to buy a car and wants to look at the concept lattice to help him select one. He has a certain idea of what the car should fulfill. Some car properties can be more important for him then others.

Consider the formal context $\langle X, Y, I\rangle$ in Tab. 1 . The context contains cars as the objects (labeled 1–8) and some of their properties as the attributes (labeled

	1	2	3	4	5	6	7	8
car 1	1	0	1	0	0	1	0	1
car 2	1	0	1	0	1	1	0	1
car 3	0	1	1	0	0	0	0	1
car 4	0	1	0	1	1	0	0	0
car 5	0	1	1	0	1	1	0	0
car 6	0	1	0	1	0	1	1	0
car 7	0	1	0	1	1	1	1	1
car 8	0	1	0	1	0	0	0	1

attributes: 1 - diesel engine, 2 - gasoline engine, 3 - sedan, 4 - hatchback, 5 - air-conditioning, 6 - airbag, 7 - power stearing, 8 - ABS

Table 1. Formal context given by cars and their properties.

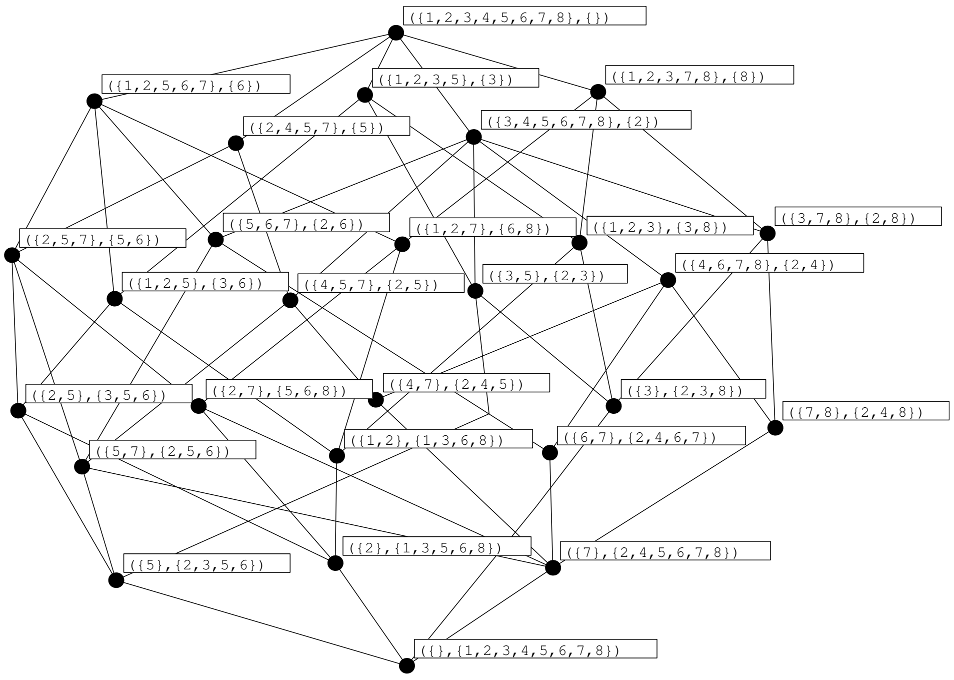


Fig. 1. Concept lattice corresponding to the context from Tab. 1

1–8). The concept lattice $\mathcal{B}(X,Y,I)$ corresponding to formal concept $\langle X,Y,I\rangle$ contains 27 formal concepts and is depicted in Fig. 1.

The formal concepts of $\mathcal{B}(X,Y,I)$ represent all concept-clusters that are present in the data. No attention is paid to importance or relative importance of attributes.

Let us now consider some attribute dependencies and the corresponding constrained concept lattices $\mathcal{B}(X,Y,I)$.

First, consider a set of AD-formulas (7)–(12). They represent the fact that most important car propeties (for a particular user) are the kind of engine, etc.

$$air - conditioning \sqsubseteq hatchback \sqcup sedan \tag{7}$$
$$powerstearing \sqsubseteq hatchback \sqcup sedan \tag{8}$$
$$airbag \sqsubseteq hatchback \sqcup sedan \tag{9}$$
$$ABS \sqsubseteq hatchback \sqcup sedan \tag{10}$$
$$hatchback \sqsubseteq gasoline\ engine \sqcup diesel\ engine \tag{11}$$
$$sedan \sqsubseteq gasoline\ engine \sqcup diesel\ engine \tag{12}$$

The concept lattice $\mathcal{B}(X,Y,I)$ constrained by AD-formulas (7)–(12) contains 13 formal concepts and is depicted in Fig. 2.

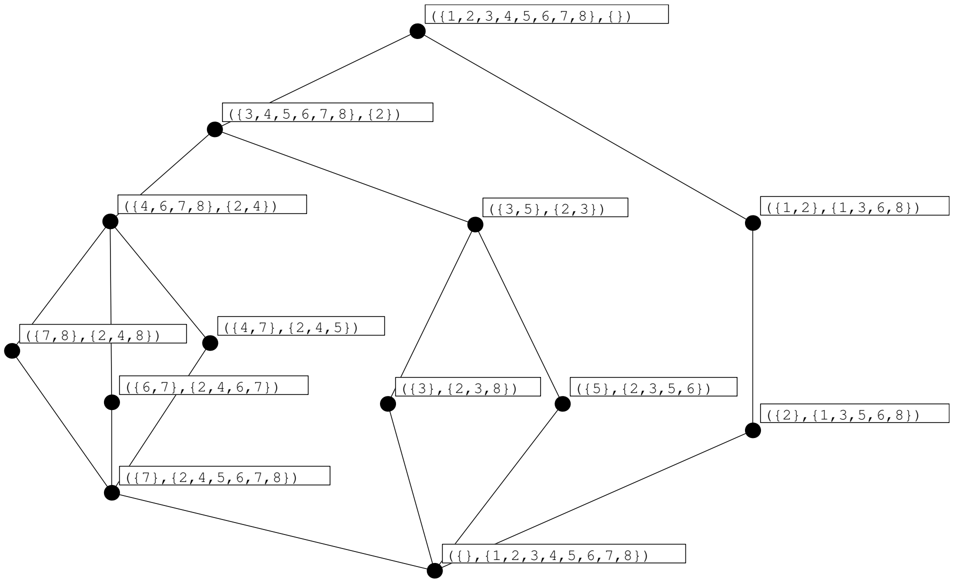


Fig. 2. Concept lattice constrained by AD-formulas (7)–(12)

Second, consider a set of AD-formulas (13)–(18). Contrary to the previous example, the importance of the type of a car and the kind of the engine are reversed.

$$air - conditioning \sqsubseteq diesel\ engine \sqcup gasoline\ engine \tag{13}$$
$$powerstearing \sqsubseteq diesel\ engine \sqcup gasoline\ engine \tag{14}$$
$$airbag \sqsubseteq diesel\ engine \sqcup gasoline\ engine \tag{15}$$
$$ABS \sqsubseteq diesel\ engine \sqcup gasoline\ engine \tag{16}$$
$$gasoline\ engine \sqsubseteq hatchback \sqcup sedan \tag{17}$$
$$diesel\ engine \sqsubseteq hatchback \sqcup sedan \tag{18}$$

The concept lattice $\mathcal{B}(X, Y, I)$ constrained by AD-formulas (13)–(18) contains 14 formal concepts and is depicted in Fig. 3.

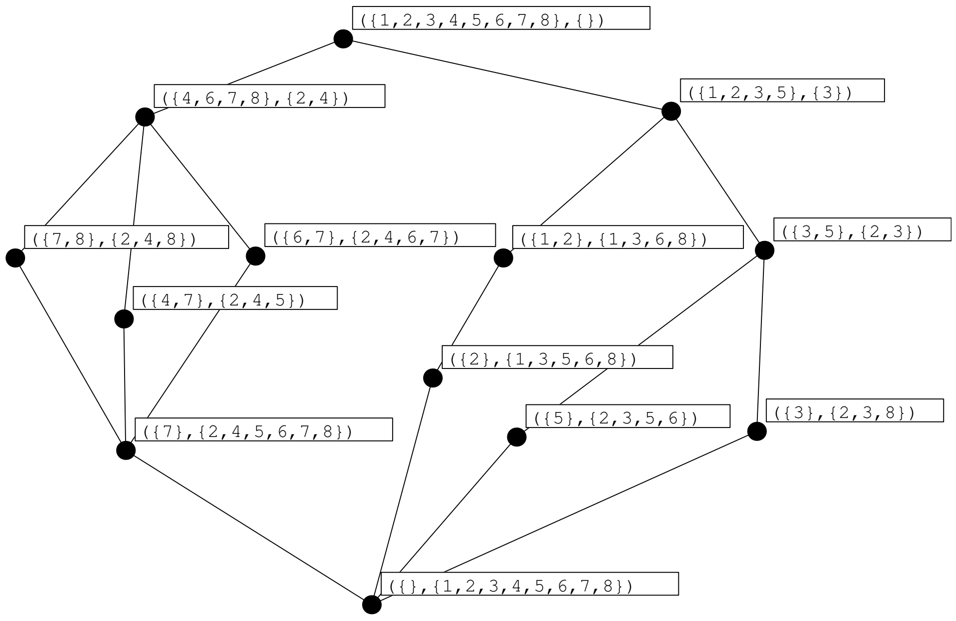


Fig. 3. Concept lattice constrained by AD-formulas (13)–(18)

Third, suppose the user finds the most important car property to be safety. The situation is described by AD-formulas (19)–(26)

$$air - conditioning \sqsubseteq diesel\ engine \sqcup gasoline\ engine \tag{19}$$
$$powerstearing \sqsubseteq diesel\ engine \sqcup gasoline\ engine \tag{20}$$
$$gasoline\ engine \sqsubseteq hatchback \sqcup sedan \tag{21}$$
$$diesel\ engine \sqsubseteq hatchback \sqcup sedan \tag{22}$$
$$sedan \sqsubseteq ABS \tag{23}$$
$$hatcback \sqsubseteq ABS \tag{24}$$
$$ABS \sqsubseteq airbag \tag{25}$$
$$airbag \sqsubseteq ABS \tag{26}$$

The concept lattice $\mathcal{B}(X, Y, I)$ constrained by AD-formulas (19)–(26) contains 6 formal concepts and is depicted in Fig. 4.

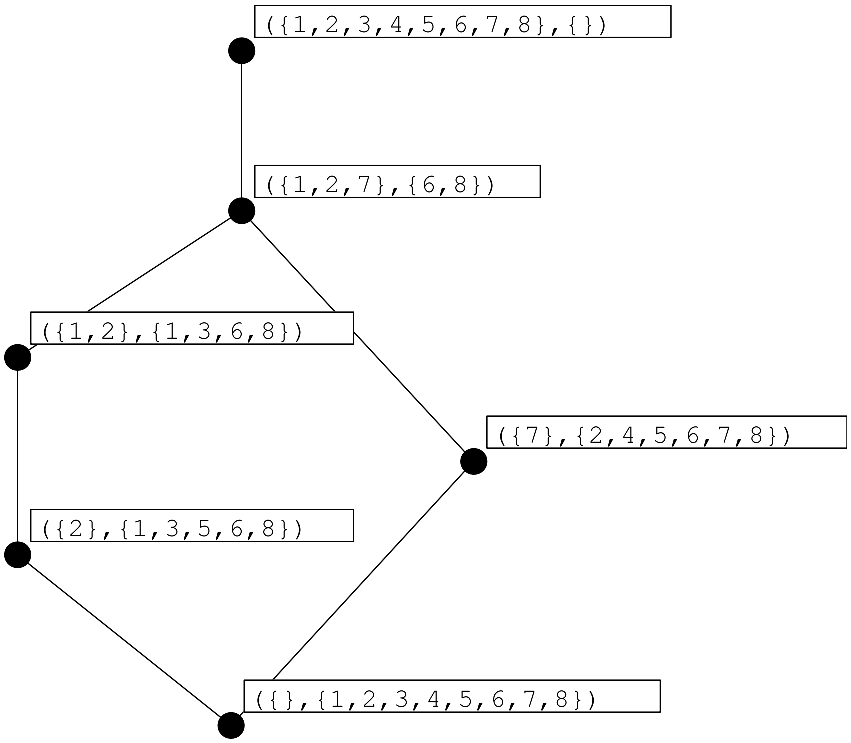


Fig. 4. Concept lattice constrained by AD-formulas (19)–(26)

Acknowledgement The research of the first author was supported by grant No. 201/02/P076 of the GAČR.

References

1. http://www.mathematik.tu-darmstadt.de/ags/ag1/Literatur/literatur_de.html
2. Arnauld A., Nicole P.: *La logique ou l'art de penser.* 1662. Also in German: *Die Logik oder die Kunst des Denkens.* Darmstadt, 1972.
3. Bělohlávek R., Sklenář V., Zacpal J.: Formal concept analysis with hierarchically ordered attributes. *Int. J. General Sytems* (to appear).
4. Ganter B., Wille R.: *Formal concept analysis. Mathematical Foundations.* Springer-Verlag, Berlin, 1999.
5. Ore O.: Galois connections. *Trans. Amer. Math. Soc.* **55**(1944), 493–513.
6. Wille R.: Restructuring lattice theory: an approach based on hierarchies of concepts. In: Rival I.: *Ordered Sets.* Reidel, Dordrecht, Boston, 1982, 445—470.

Concepts Valuation by Conjugate Möebius Inverse

Petr Gajdoš, Václav Snášel

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

Abstract. There is several known algorithm to construct concept lattices. The question is, how could we simplify this lattice into concepts, that are important and have selected features. According to "A Theory of Diversity", we can compute the diversity of a set of objects recursively from the pairwise dissimilarities between its elements. Using Conjugate Möebius Inverse, we can compute weights of each concept from these diversities. Determining attribute weights is a complex task, however, since there are as many potential attributes as there are non-empty subset of object. The document shows the implementation of Möebius function on concept lattices and then determinig concepts weights by pairwise between objects. We suppose, this is the way to simplify concept lattices.

Keywords: Möebius, concept lattice, diversity, dissimilarity, weight

1 Introduction

This article addresses the problem of the weighting of concepts. In "A Theory of Diversity" (Nehring and Puppe, 2002, henceforth TD), we proposed a multi-attribute approach according to which the diversity of a set of objects is determined by the number and weight of the different features (attributes) possessed by them. In some cases, the diversity of a set can be computed recursively from the pairwise dissimilarities between its elements (plus their value as singletons). Two basic models for which this is possible are the hierarchical model studied by Weitzman (1992, 1998) in the context of biodiversity and the more general line model introduced in TD. As already observed by Weitzman (1992), then hierarchical model implies that the two greatest dissimilarities between three points are always equal if singletons are equally valued. The purpose of the present paper is to show, how could we compute the weights of concepts using knowledge of this models.

Section 2 procides the necessary background from formal concepts analysis and TD. Section 3 shows the implementation of Conjugate Möebius Function to valuating concept lattice and another way to get same values by pairwise dissimilarities between objects. Last section is devoted to the poser with weighting concept lattices.

2 Backgound

This section shows some definitions and tools, that are important for our later valuating of concepts. First we define a context and concept lattice. Next, we summarize the basic features of the multi-attribute model developed in TD.

2.1 Context and concept lattice

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

Definition 2. *for a set* $A \subset G$ *of object we define*

$$A^{'} = \{m \in M \mid gIm \ for \ all \ g \in A\}$$

(the set of attributes common to the objects in A*). Correspondingly, for a set* B *of attributes we define*

$$B^{'} = \{g \in G \mid gIm \ for \ all \ m \in B\}$$

(the set of objects which have all attributes in B*).*

Definition 3. *A formal concept of the context* (G, M, I) *is a pair* (A, B) *with* $A \subseteq G$*,* $B \subseteq M$*,* $A^{'} = B$ *and* $B^{'} = A$*. We call* A *the extent and* B *the intent of the concept* (A, B)*.* $\mathfrak{B}(G, M, I)$ *denotes the set of all concepts of context* (GMI)

Definition 4. *The concept lattice* $\underline{\mathfrak{B}}(G, M, I)$ *is a complete lattice in which infimim and supremum are given by:*

$$\bigwedge_{t \in T} (A_t, B_t) = \left(\bigcap_{t \in T} A_t, \left(\bigcup_{t \in T} B_t \right)^{''} \right)$$

$$\bigvee_{t \in T} (A_t, B_t) = \left(\left(\bigcup_{t \in T} A_t \right)^{''}, \bigcap_{t \in T} B_t \right) .$$

We refer to [1].

[1] The attribute has different meaning in the Conjugate Möebius Inverse. It's a set of objects.

2.2 Diversity function

Definition 5. *Let F be the totality of all features deemed relevant in the specific context, and denote by $R \subseteq X \times F$ the "incidence" relation that describes the features possessed by each object, i.e. $(x, f) \in R$ whenever object $x \in X$ possesses feature $f \in F$. For each relevant feature $f \in F$, let $\lambda_f \geq 0$ quantify the value of realization of f. Upon normalization, λ_f can thus be thought of as the relevant importance, or weight of feature f. The diversity value of a set S is defined as*

$$v(S) = \sum_{f \in F : (x,f) \in R \ for \ some \ x \in S} \lambda_f \tag{1}$$

The diversity value of a set is given by the total weight of all different features possessed by some objects in S. Note expecially that each feature occurs at most once at sum. In particular, each single object contributes to diversity the value af all those features that are not possessed by any already existing objects.

For any subset $A \subseteq X$ of objects denote by F_A the set of features that are possessed exactly the objects in A. Each feature in F_A is possessed by all elements of A and not possessed by any element of $X \setminus A$. Then we can write

$$v(S) = \sum_{A \cap S \neq \emptyset} \sum_{f \in F_A} \lambda_f \tag{2}$$

Then, for each subset $A \subseteq X$ denote by $\lambda_A := \sum_{f \in F_A} \lambda_f$ the total weight of all features with extension A, with the convention that $\lambda_A = 0$ whenever $F_A = \emptyset$. With this notation we write

$$v(S) = \sum_{A \cap S \neq \emptyset} \lambda_A \tag{3}$$

2.3 Conjugate Möebius Inverse

Theorem 1. *For any function $v : 2^X \rightarrow R$ with $v(\emptyset) = 0$ there exists unique function $\lambda : 2^X \rightarrow R$, the Conjugate Möebius Inverse, such that $\lambda_\emptyset = 0$ and, for all S,*

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

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

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

where S^c denotes the complement of S in X.

We refer to [4].

3 Concept lattice and Möebius function

This part shows, how can we compute weights and diversity of concepts of particular concept lattice. Then we use dissimilarity and similarity function to get the same result by easier way.

Description of objects and features in incidence matrix.

C	=	cat	q	=	quadrupped (four feet)
M	=	monkey (chimpanzee)	p	=	pilli
D	=	dog	i	=	intelligence
F	=	fish (delphinus)	w	=	live in water
H	=	human	h	=	hand
W	=	whale			

Table 1. Incidence relation matrix.

	$\lambda_q = 2$	$\lambda_p = 3$	$\lambda_i = 4$	$\lambda_w = 2$	$\lambda_h = 2$
	q	**p**	**i**	**w**	**h**
C	x	x			
M		x	x		x
D	x	x			
F			x	x	
H			x		x
W			x	x	

There are all subsets $A \subseteq X$ in the table 2. F_A presents a set of relevant features $f \in F$, which are possessed by all elements of set A, but not possessed by any element of a set $X \setminus A$. By $\lambda_A := \sum_{f \in F_A} \lambda_f$, we get the values in the table.

By $v(S) = \sum_{A: A \cap S \neq \emptyset} \lambda_A$, we compute the diversity of each subset of objects of universum X, $S \subseteq X$ in the table 3. In this time, we include all attributes and their weights to compute diversities of subsets S. The large the incidence matrix the large count of conceivable attribues and subsets of objects, so it's more difficult to compute diversity function.

Next, we consider only attributes corresponding to concepts. Sets of attributes are not sets of features of concept but they are identical to sets of objects.

We use Conjugate Möebius Inverse (5) to compute weights of attributes (concepts) from diversities in the table 4.

Table 2. All conceivable attributes.

A	F_A	λ_A	A	F_A	λ_A	A	F_A	λ_A	A	F_A	λ_A	A	F_A	λ_A
$\emptyset$			FW	w	2	CMD	p	3	MFHW	i	4	CMDFH		
C			MH	h	2	CMF			CMDF			CMDFW		
M			CD	q	2	CMH			CMDH			CMDHW		
D			CM			CMW			CMDW			CMFHW		
F			CF			CDF			CMFH			CDFHW		
H			CH			CDH			CMFW			MDFHW		
W			CW			CDW			CMHW			CMDFHW		
			MD			CFH			CDFH					
			MF			CFW			CDFW					
			MW			CHW			CDHW					
			DF			MDF			CFHW					
			DH			MDH			MDFH					
			DW			MDW			MDFW					
			FH			MFH			MDHW					
			HW			MFW			DFHW					
						MHW								
						DFH								
						DFW								
						DHW								
						FHW								

Table 3. Diversities of subsets S of objects.

S	$v(S)$	S	$v(S)$	S	$v(S)$	S	$v(S)$	S	$v(S)$	S	$v(S)$
$\emptyset$	0	FW	6	CMD	11	MFHW	11	CMDFH	13	CMDFHW	13
C	5	MH	9	CMF	13	CMDF	13	CMDFW	13		
M	9	CD	5	CMH	11	CMDH	11	CMDHW	13		
D	5	CM	11	CMW	13	CMDW	13	CMFHW	13		
F	6	CF	11	CDF	11	CMFH	13	CDFHW	13		
H	6	CH	11	CDH	11	CMFW	13	MDFHW	13		
W	6	CW	11	CDW	11	CMHW	13				
		MD	11	CFH	13	CDFH	13				
		MF	11	CFW	11	CDFW	11				
		MW	11	CHW	13	CDHW	13				
		DF	11	MDF	13	CFHW	13				
		DH	11	MDH	11	MDFH	13				
		DW	11	MDW	13	MDFW	13				
		FH	8	MFH	11	MDHW	13				
		HW	8	MFW	11	DFHW	13				
				MHW	11						
				DFH	13						
				DFW	11						
				DHW	13						
				FHW	8						

Table 4. Weighting by CMI

A	$S : S \subseteq A$	S^c	$(-1)^{\|A\|-\|S\|+1}$	$v(S)$	λ_A
FW	F	CMDHW	+1	13	2
	W	CMDFH	+1	13	
	FW	CMDH	−1	11	
	∅	CMDFHW	−1	13	
MH	M	CDFHW	+1	13	2
	H	CMDFW	+1	13	
	MH	CMDW	−1	11	
	∅	CMDFHW	−1	13	
CD	C	MDFHW	+1	13	2
	D	CMFHW	+1	13	
	CD	MFHW	−1	11	
	∅	CMDFHW	−1	13	
cmd	C	MDFHW	−1	13	3
	M	CDFHW	−1	13	
	D	CMFHW	−1	13	
	CM	DFHW	+1	13	
	CD	MFHW	+1	11	
	MD	CFHW	+1	13	
	CMD	FHW	−1	8	
	∅	CMDFHW	+1	13	
MFHW	M	CDFHW	+1	13	4
	F	CMDHW	+1	13	
	H	CMDFW	+1	13	
	W	CMDFH	+1	13	
	MF	CDHW	−1	13	
	MH	CDFW	−1	11	
	MW	CDFH	−1	13	
	FH	CMDW	−1	13	
	FW	CMDH	−1	11	
	HW	CMDF	−1	13	
	MFH	CDW	+1	11	
	MFW	CDH	+1	11	
	MHW	CDF	+1	11	
	FHW	CMD	+1	11	
	MFHW	CD	−1	5	
	∅	CMDFHW	−1	13	
M	M	CDFHW	−1	13	0
	∅	CMDFHW	+1	13	
∅					0
CMDFHW	all subsets	all subsets			0

Any diversity function satisfies this formula:

$$v(S \cup \{x\}) - v(S) = \sum_{A \ni x, A \cap S = \emptyset} \lambda_A \qquad (6)$$

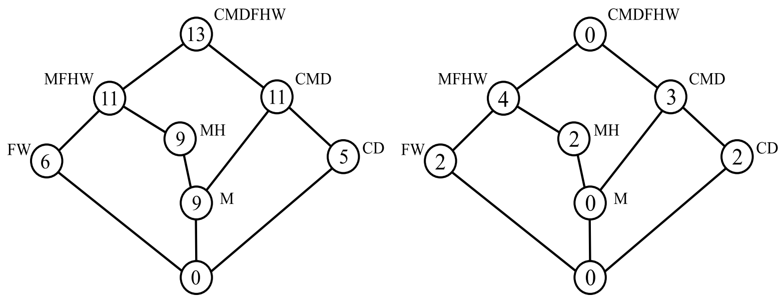


Fig. 1. a) Diversities of concepts b) Weights of concepts

Table 5. Dissimilarities

	C	**M**	**D**	**F**	**H**	**W**
C	0	2	0	5	5	5
M	6	0	6	5	3	5
D	0	2	0	5	5	5
F	6	2	6	0	2	0
H	6	0	6	2	0	2
W	6	2	6	0	2	0

By (6), the marginal diversity of an object x at a set S is given by the total weight of all attributes possessed by x but by no element of S. Accordingly, we will refer to marginal diversity also as the distinctiveness of x from S, which we denote by

$$d(x, S) := v(S \cup \{x\}) - v(S). \tag{7}$$

A diversity function naturally induces a notion of pairwise dissimilarity between objects as follows.

Definition 6. *For all x, y*

$$d(x, y) := d(x, \{y\}) = v(\{x, y\}) - v(\{y\}). \tag{8}$$

By (6), $d(x, y)$ is the weight of all attributes possessed by x but not by y. Note that, in general, d need not be symmetric. We can read it from table (5).

As we said at the beginning, there are two models in TD. The hierarchical and the more general line model. All concept lattices are hierarchical ordered. But, weighting of concepts is a difficult task. We can assign values to concepts only in small and simly lattice because of next condition.

Definition 7. *A model $H \subseteq 2^X$ is called a (taxonomic) hierarchy if the elements of H are nested in the sence that, for all $A, B \in H$,*

$$A \cap B \neq \emptyset \Rightarrow [A \subseteq B \ \vee \ B \subseteq A]. \tag{9}$$

Accordingly, we will refer to a diversity function v, as well as to the associated attribute weighting function λ, as hierarchical if the support Λ of relevant attributes forms a hierachy. Diversity function is hierarchical if and only if, for all x and S,

$$v(S \cup \{x\}) - v(S) = \min_{y \in S}[v(\{x,y\}) - v(\{y\})] \tag{10}$$

or, equivalently,

$$d(x,S) = \min_{y \in S} d(x,y) \tag{11}$$

Theorem 2. Conjugate Möebius Inverse on a hierarchy. *Let v be a diversity function with attribute weighting function λ. If v is hierarchical, then for all $A \in \Lambda$ and all $x \in A$,*

$$\lambda_A = \min_{y \in A^c} d(x,y) - \max_{y \in A} d(x,y) \tag{12}$$

Conversely, suppose that,
for all $A \in \Lambda$ and all $x \in A$, $\lambda_A = d(x,A^c) - \max_{y \in A} d(x,y)$, then λ is hierarchical.

According to (12) we compute weights of concepts in the table 6. We can see, that most of values are correct (compare with table (4)). But, some of them are not right although we have used the same formula (12). We can find hierarchical ordering in the concept lattice but it is different to hierarchy defined in (9).

Table 6. Values of concepts according to (12)

Concept	A	A^c	*sel.* x	$\min_{y \in A^c} d(x,y)$	$\max_{y \in A} d(x,y)$	λ_A
C1	CD	MFHW	C	2	0	2
			D	2	0	2
C2	M	CDFHW	M	0	0	0
C3	FW	CMDH	F	2	0	2
			W	2	0	2
C4	MH	CDFW	M	5	3	2
			H	2	0	2
C5	CMD	FHW	M	3	6	-3
			C	5	2	3
			D	5	2	3
C6	MFHW	CD	M	6	5	1
			F	6	2	4
			H	6	2	4
			W	6	2	4
C7	CMDFHW	∅	C	0	0	0
C8	∅	CMDFHW	∅	0	0	0

Proof. Suppose that Λ is a hierarchy. Let $x \in A \in \Lambda$, and define $z^* := argmax_{z \in A} d(x,z)$. Since $\Lambda_x = \{B \in \Lambda : x \in B\}$ is a chain, one has $B \subset A \Leftrightarrow z^* \notin B$ for all $B \in \Lambda_x$, where "$\subset$" denotes the proper subsethood relation.
Hence,

$$\begin{aligned}
&\min_{z \in A^c} d(x,z) - \max_{z \in A} d(x,z)\\
&\quad = v(\{x\} \cup A^c) - v(A^c) - d(x,z^*)\\
&\quad = \lambda(\{B : x \in B \subseteq A\}) - \lambda(\{B : x \in B, z^* \notin B\})\\
&\quad = \lambda_A + \lambda(\{B : x \in B \subset A\}) - \lambda(\{B : x \in B, z^* \notin B\})\\
&\quad = \lambda_A.
\end{aligned}$$

Conversely, suppose that Λ is not a hierarchy, i.e. suppose there exists $A, C \in \Lambda$ such that $A \cap C$, $A \setminus C$, and $C \setminus A$ are all non-empty. Let $x \in A \cap C$. Without loss of generality we may assume that A is an minimal element of Λ satisfying $x \in A$ and $A \setminus C \neq \emptyset$, i.e. for no proper subset A' of A, $x \in A' \in \Lambda$ and $A' \setminus C \neq \emptyset$. Let $y \in A \setminus C$. By construction one has $\{B \in A : x \in B, B \subset A\} \subset \{B \in \Lambda : x \in B, y \notin B\}$ since C belong to the latter but not to the former set. Since assumption, $\lambda_C > 0$, this implies $\lambda(\{B : x \in B \subset A\}) - \lambda(\{B : x \in B, y \notin B\}) < 0$.
Therefore,

$$\begin{aligned}
&d(x, A^c) - \max_{z \in A} d(x,z)\\
&\quad = v(\{x\} \cup A^c) - v(A^c) - \max_{z \in A} d(x,z)\\
&\quad \leq v(\{x\} \cup A^c) - v(A^c) - d(x,y)\\
&\quad = \lambda(\{B : x \in B \subseteq A\}) - \lambda(\{B : x \in B, y \notin B\})\\
&\quad = \lambda_A + \lambda(\{B : x \in B \subset A\}) - \lambda(\{B : x \in B, y \notin B\})\\
&\quad < \lambda_A.
\end{aligned}$$

According to next definition, we can divide the concept lattice into hierarchies to compute weights of concepts by CMI.

Definition 8. *A* **lattice hierarchy** *H in lattice L is join-sublattice where $a \cap b \neq \emptyset \Rightarrow [a \leq b \vee b \leq a]$, $a, b \in H$. $\mathfrak{H}(L, \subseteq)$ denotes the poset of all lattice hierarchies of lattice L.*

Theorem 3. *Let H be the lattice hierarchy. Then Hasse diagram $H \setminus 0$ is rooted tree.*

Proof. Because H is finite join-sublattice then exists join r for all element of H. It is easy to show that r is root. Hence Hasse diagram $H \setminus 0$ is connected. Let $H \setminus 0$ contain a cycle i.e. suppose there exists $a, b \in H \setminus 0$ such that $a \| b$ and $a \wedge b = c$, $c \neq 0$. We obtain a contradiction with presumption.

4 Conclusion

Concept lattice is ordered but we can not use a simple method to compute weights of concepts by Conjugate Möebius Inverse. We try to delegate this problem to pairwise of elements of the hierarchical model. We get values of weights or diversities but this method ensures right results if and only if, concept lattice or a part of lattice satisfy the condition of hierarchical structure (9).

We see another way to solve this problem. In future, we want to prove, that we can "supply" any concepts lattice by finite set of trees, that are ordered and they satisfy our condition of hierarchical structure. We want to find minimal count of hierarchies, that can cover concept lattice.

References

1. B. Ganter and R. Wille. *Formal Concept Analysis.* Springer-Verlag Berlin Heidelberg, 1999.
2. K. Nehring. A theory of diversity. *Ecometrica 70*, pages 1155–1198, 2002.
3. K. Nehring and C. Puppe. Modelling phylogenetic diversity. *Resource and Energy Economics*, 2002.
4. K. Nehring and C. Puppe. Diversity and dissimilarity in lines and hierarchies. *Mathematical Social Sciences 45*, pages 167–183, 2003.
5. I. Vondrak and V. Snasel. Using concept lattices for organizational structure analysis. *In Proceedings of European Concurrent Engineering Conference ECEC '02 (Modena, Italy)*, 2002.

Querying the RDF: Small Case Study in the Bicycle Sale Domain

Ondřej Šváb, Vojtěch Svátek, Martin Kavalec, and Martin Labský

Department of Information and Knowledge Engineering,
University of Economics, Prague, W. Churchill Sq. 4, 130 67 Praha 3, Czech Republic
{xsvao06,svatek,kavalec,labsky}@vse.cz

Abstract. We examine the suitability of RDF, RDF Schema (as simple ontology language), and RDF repository *Sesame*, for providing the back-end to a prospective domain-specific web search tool, targeted at the offer of bicycles and their components. Actual data for the RDF repository are to be extracted by analysis modules of a distributed knowledge-based system named *Rainbow*. Attention is paid to the comparison of different query languages and to the design of application-specific templates.

1 Introduction

The goal of *semantic web* initiative is to endow web data with formal syntax and semantics and thus make them available for automated reasoning. Such reasoning could improve information retrieval (moving from keyword-based to content-based retrieval), but also increase the degree of automation in data/application integration or website design. Among the 'semantic web' representation languages, RDF[1] has prominent position. It is used to express interconnected logical facts, which can be *semantically* viewed as simple kind of structured *knowledge*. On the other hand, semantic web facts may potentially arise in large quantities, and thus, at the *syntactical level*, require treatment similar to traditional, tabular *data*. Several tools have been developed for RDF storage and retrieval, such as *Jena*[2], *RDF Suite* [1], and finally *Sesame*, which is the focus of this paper. There are also multiple query languages implemented in different tools.

Although semantic annotations written by hand have the best quality, it is unrealistic to obtain the critical mass of semantic web purely manually. Automated annotation of legacy pages (or service descriptions) by means of lingustic or statistical techniques became a hot topic in semantic web research[3]. In this paper, we analyse the applicability of RDF query languages and of the *Sesame* repository for storage and retrieval of facts potentially discovered by *Rainbow*—a distributed knowledge-based system for analysis of web content and structure.

[1] http://www.w3.org/RDF

[2] http://www.hpl.hp.com/semweb/jena2.htm

[3] Cf. the workshop on 'Human Language Technology for Semantic Web and Web Services' at the last International Semantic Web Conference, http://gate.ac.uk/conferences/iswc2003.

Section 2 of this paper discusses how facts (on bicycle sales) extracted by *Rainbow* can be represented in RDF and shows the underlying RDF Schema. Section 3 describes the architecture of *Sesame*. Section 4 compares the major RDF query languages with respect to the given application. Finally, section 5 deals with application-specific templates that can shield a user of the (prospective) semantic search tool from the syntax of query languages.

2 Representing the *Rainbow* Results in RDF

2.1 Architecture of *Rainbow*

The loosely-coupled architecture of *Rainbow* consists of a *web spider*, a *full-text database tool*, and several *analytical tools*, interfacing with each other by means of *web service* protocols. Analytical tools specialise in different forms of web data, such as free text, text structured with HTML tags, website topologies or images. The semantics of services is modelled by an *application ontology*. In the current state of the system, the services can only be invoked procedurally, in a fixed order; the ontology is hence merely used indirectly, at design time of the composed application. A more flexible composition solution (based on skeletal planning) is envisaged for the future. More details can be found in [8] and at the project homepage[4]. The current application of *Rainbow* aims at development of a semantic search tool for the domain of *bicycle products*. Websites of bicycle-selling companies[5] are systematically analysed, with emphasis on catalogue information, but also including a general profile of the company.

2.2 Resource Description Framework and Vocabulary Language

Resource description framework[6] (RDF) is a language developed for representing information about resources in the World Wide Web; it can however be used as general language for encoding facts. RDF statements, also called *triples*, consist of three parts: *subject*, *predicate* (property), and *object*. An statement may e.g. say that a particular web page was created by a particular human: the page then is the subject, the human is the object, and the relation 'created by' is the predicate. Any real-world entity (and even property) can be understood as resource, whether accessible via the web or not. Object is the only part of statement where not only a resource but also a *literal* (i.e. simple string or numerical value) may appear. RDF literals can also be *typed*, via reference to *XML Schema datatypes*. Even whole statements can be declared as resources: this technique is called *reification*[7], and enables to assert facts about statements themselves. To identify a particular information resources, RDF uses *URI*, the

[4] `http://rainbow.vse.cz`

[5] As initial data source, we use the sites referenced by the *Google Directory*, namely its node `Sports/Cycling/BikeShops/Europe/UK/England`.

[6] `http://www.w3.org/RDF`

[7] From Latin: *res*= 'thing', since the statement thus becomes an object of discourse.

Uniform Resource Identifier. RDF statements may be encoded using varying *serialization syntax*[8] but always conform to the same, *graph-oriented data model*, where subjects and objects are represented by nodes, predicates by directed arcs, and each node-arc-node triple represents one RDF statement.

RDF is endowed with more expressive power through RDFS: the *RDF Schema* [2], which plays the role of vocabulary language. Individual resources can be assigned types, i.e. *classes*; RDFS then allows to build a *hierarchical structure* over classes and properties, and to declare the *domain* and *range* of properties.

2.3 RDF Facts and RDFS Ontology on Bicycle Sites

When applied on bicycle-selling sites, analytical modules of Rainbow are typically able to extract the *name* of a bike, its *price*, details on its *components* (such as fork, frame, rear derailer etc.), its *picture*, and possibly some information about the *company* that offers it. Bikes, as well as separately-sold bike components are associated with *retail offers*. Examples of information 'triples' (in free-text form, to avoid syntax issues) are "Company X offers bike Y". "Bike Y has name Rockmachine Tsunami", "Bike Y has fork Z". "Fork Z has name Marzocchi Air", "Price of bike Y is 2500." Furthermore, we need to represent *metadata* associated with the extracted facts, such as "Statement XY has certainty 0.75" or "Statement XY was produced by URL analysis module".

The *RDF schema* (i.e. simple ontology) of our domain uses four namespaces: `bike` dealing with bikes themselves, `comp` dealing with (not necessarily 'bike') companies, `pict` dealing with pictures on web pages, and `meta` dealing with metadata on statements extracted by *Rainbow*. Its graph is shown on Fig. 1 and 2 (decomposed for easier readability). The central point of the schema is the concept of RetailOffer. It corresponds to an offer of BikeProduct (whole bike or component) by a Company; it is also associated with the Name under which and Price for which it is offered, and URL of associated Picture. URI of particular RetailOffer corresponds to the URL of catalogue item containing the offer[9]. BikeProduct is superclass of all bike products. Note that BikeProduct and its subclasses only have 'types' of products as their instances, not individual physical entities. Such 'type' of product can be offered for different prices and even under slightly different names (associated with the given instance of RetailOffer) and accompanied with different pictures, while BikeProduct itself has a 'canonical' name, specified e.g. by its manufacturer. Finally, let us explain the nature of *metadata*. Our solution to representing them is based on reification and inspired by the SWAP project[10]. In order to store metadata about e.g. origin, confidence,

[8] The standard format is *RDF/XML* (see `http://www.w3.org/RDF`); there is also a line-based encoding format, *N-triples* (see `http://www.w3.org/TR/2002/WD-rdf-testcases-20020429`) and a format easily readable for humans, *Notation 3* (N3, see `http://www.w3.org/2000/10/swap/Primer`).

[9] Typically the place from where the core information was extracted.

[10] Ongoing IST project on Semantic Web and Peer-to-Peer (knowledge nodes), see `http://swap.semanticweb.org/`

security and caching of each piece of knowledge, they set up a complex metadata schema [3]. In contrast, we are only need a few metadata items, such as, information on which *analysis module* the statement was obtained from, or its *certainty factor*. Metadata are grouped under an abstract class called Meta.

3 Architecture of *Sesame*

Sesame allows persistent storage and querying of RDF data and schema[11]; it consists of three functional modules:

– *Query Module* parses a query, builds the query tree to optimise it, and finally evaluates it in a streaming fashion.
– *Admin Module* enables to insert and delete RDF data and schema, checks for consistency of newly added statements with statements in the repository, and infers entailed information. Inferencing is done according to rules and axioms defined in [6] as well as to custom (application-specific) rules/axioms[12].
– *Export Module* exports the content of repository (data or schema or both).

Additionally, *Sesame* uses a stack of SAILs (Storage and Inference Layers), which transparently ensure access to specific implementations of repository. The underlying repositories can be based on a (relational or object-oriented) DBMS, RDF files cached in memory, RDF network services or existing RDF stores.

Our choice of *Sesame* was to some extent motivated by our close contacts with its developers. There are at least two comparable tools for RDF storage and retrieval. *RDF Suite*[13] [1], developed by ICS FORTH, Greece, is potentially faster for large queries thanks to flexible adaptation of database schema to the given RDF schema. It supports the full RQL query language (see section 4.1), and enables dynamic loading of multiple RDF schemata. *Jena*[14], developed by HP Labs, Bristol, UK, offers a user-friendly interface for writing RDF schemata, and an API for ontology languages (OWL, DAML+OIL, RDFS). It only supports the RDQL query language (see section 4.3). Arguments in favour of *Sesame* might be close adherence to the most recent 'inference-centric' updates of RDF, and some features of its original query language, SeRQL (see section 4.2). Given the experimental nature of our project, response time, reliability (typically decreasing with increasing role of inference), and quality of editing interface do not play so crucial a role, and since we deal with a single RDF Schema fully under our control, there is no need for dynamic schema integration. In our setting for *Sesame*, we further opted for RDBMS back-end.To enable easy search in the bicycle data repository, an *HTML interface* is being developed, with pre-fabricated query templates (cf. section 5).

[11] The first stable version, *Sesame 1.0RC1*, can be downloaded from `http://sourceforge.net/projects/sesame`. *Sesame* is developed by the Dutch company *Aduna* (earlier *Aidministrator*), see `http://sesame.aidministrator.nl`.

[12] In our case, it is e.g. possible to define a rule stating that property partOfModel is inverse to property hasBikePart.

[13] `http://139.91.183.30:9090/RDF`

[14] `http://www.hpl.hp.com/semweb/jena2.htm`

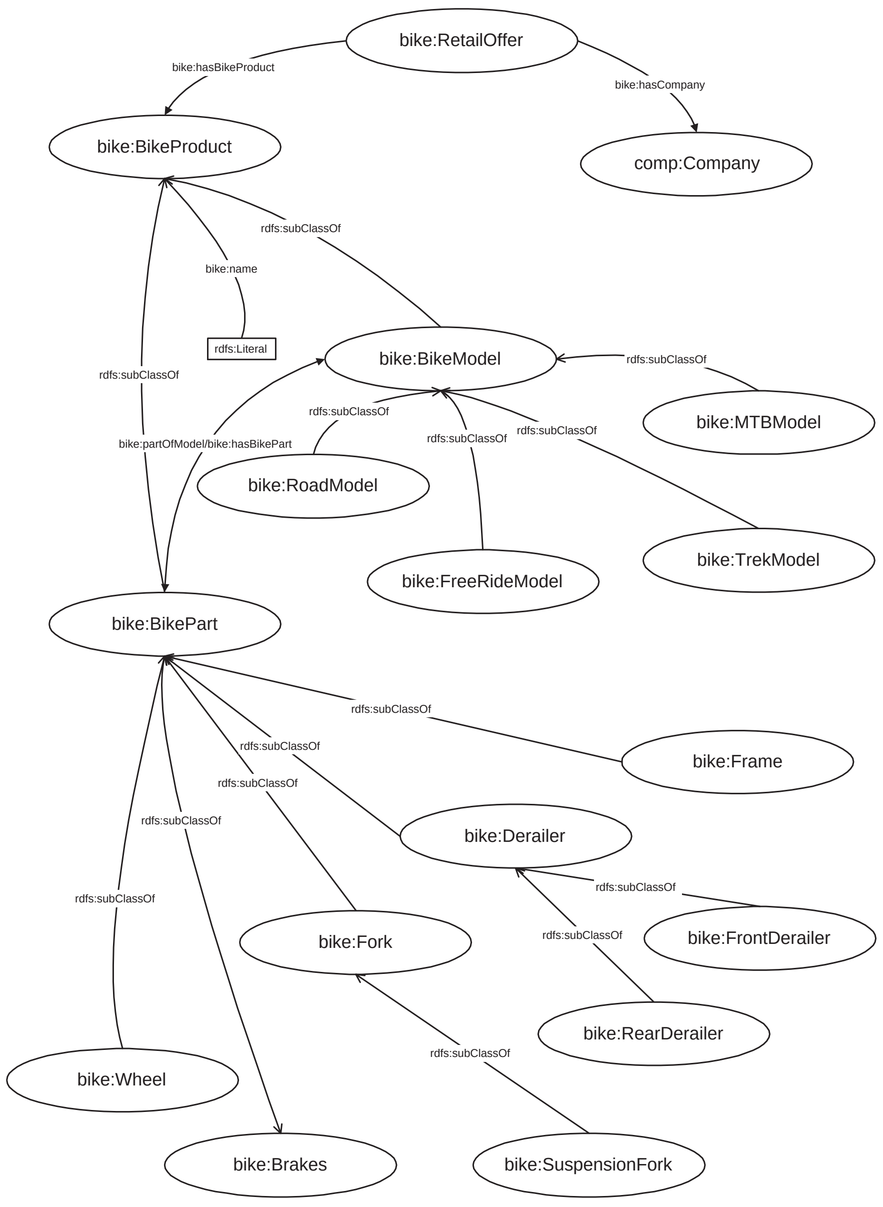


Fig. 1. RDF schema of bicycle domain 1/2

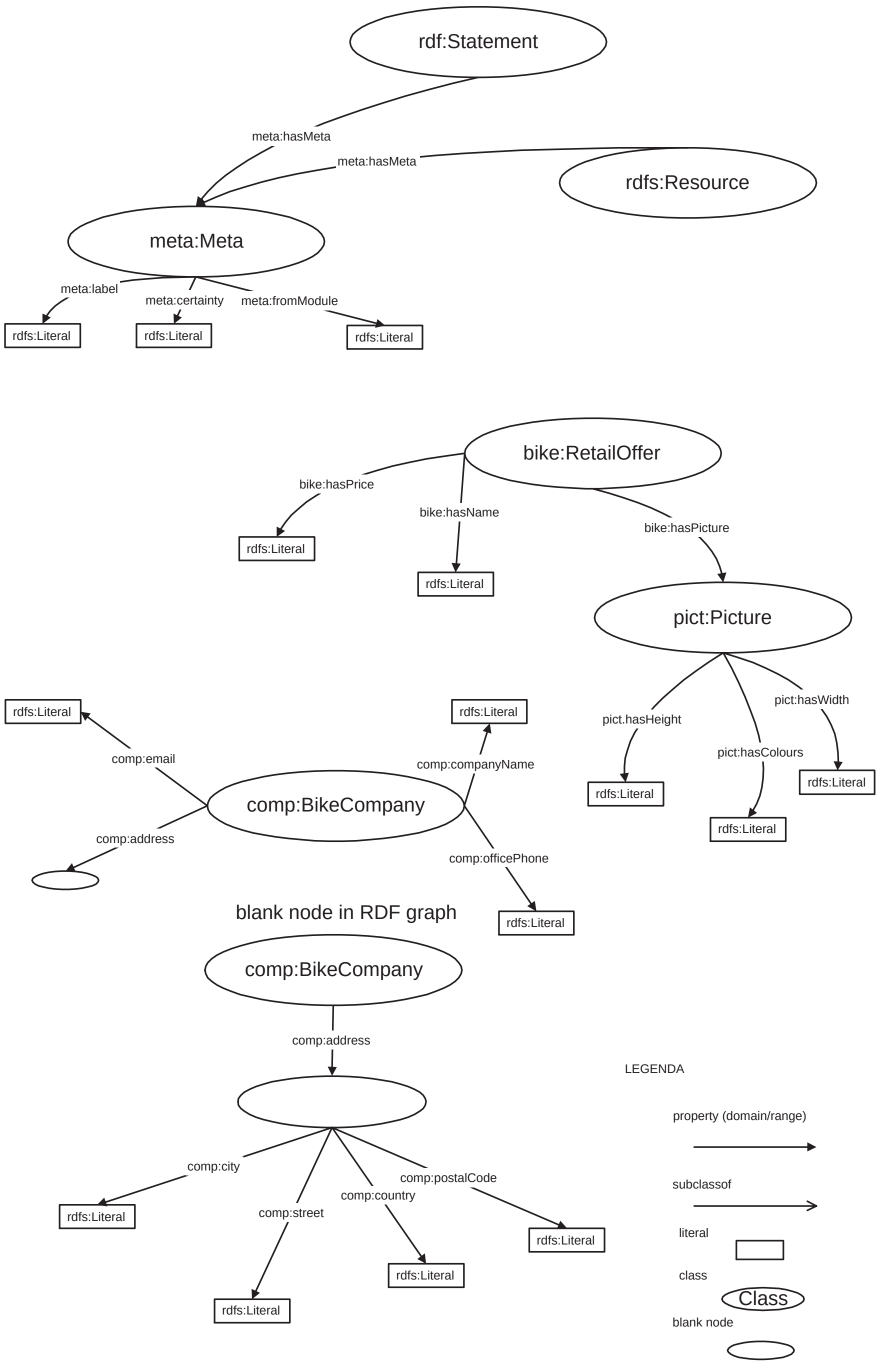


Fig. 2. RDF schema of bicycle domain 2/2

4 Querying the RDF in *Sesame*

There are three RDF query languages implemented in *Sesame*: RQL, RDQL and SeRQL. Unlike standard SQL, these languages only serve for querying and not for data manipulation. The main accent is laid on RQL and SeRQL, which enable querying both the RDF data and associated RDF schemata. We will demonstrate RQL and SeRQL on examples, and attempt to expose their weak and strong aspects. We will also briefly mention RDQL and the language implemented in the PerlRDF tool produced by Ginger Alliance (i.e. not supported by *Sesame*).

4.1 RQL

RQL is a declarative query language over RDF and RDFS. It was originally proposed in the context of *RDF Suite* [1]; its implementation in *Sesame* is only partial and adheres more closely to the recent RDF updates by the W3C. The building blocks of RQL are *functions* and *path expressions*. Functions enable to directly query the RDF Schema. Examples of functions are `Class` (returning the set of all classes), `Property` (returning the set of all properties), `domain` or `range` (returning the domain/range of a property). They can be be combined in many ways, even with path expressions.

RQL offers the SELECT-FROM-WHERE construct known from SQL, with some differences. It has two obligatory and two optional parts. In the (obligatory) SELECT clause, we list the variables (selected from the subsequent FROM clause), the values of which we want in the result. We could also use an asterisk, representing all variables. In the (obligatory) FROM part, we specify the RDF subgraph over which we query, in the form of path expression representing a filter on the graph. Further conditions are expressed is the (optional) WHERE clause. A WHERE expression is typically a *comparison* of variables from FROM clause and concrete values; we can also compare a variable with the result of an embedded query (see example 2Q), and use boolean *connectives*. RQL offers the comparison operators `<,>,=,>=,<=` and `like` (string matching, with possible left or right expansion). The RQL query engine tries to convert the operands to be compared to the same type. It is doing in this sequence: classes, properties (both compared hierarchically), real numbers, integers, literals and finally, resources. The last clause is (optional) USING NAMESPACE; it enables to write elements from certain namespace in short form (prefix:local name = Qname)[15].

Now we demostrate the use of RQL on examples related to our bicycle application; they conform to the RDF Schemata in Fig. 1 and 2. Path expressions of the queries are shown in Fig. 3.

Let us first demonstrate the use of RQL functions.

1Q: *Find restrictions (domain and range) of property hasWidth.*

```
select domain(@predicate), @predicate, range(@predicate)
from  {} @predicate {}
where @predicate = pict:hasWidth
```

[15] In all examples in this paper we omit namespace definitions, for brevity.

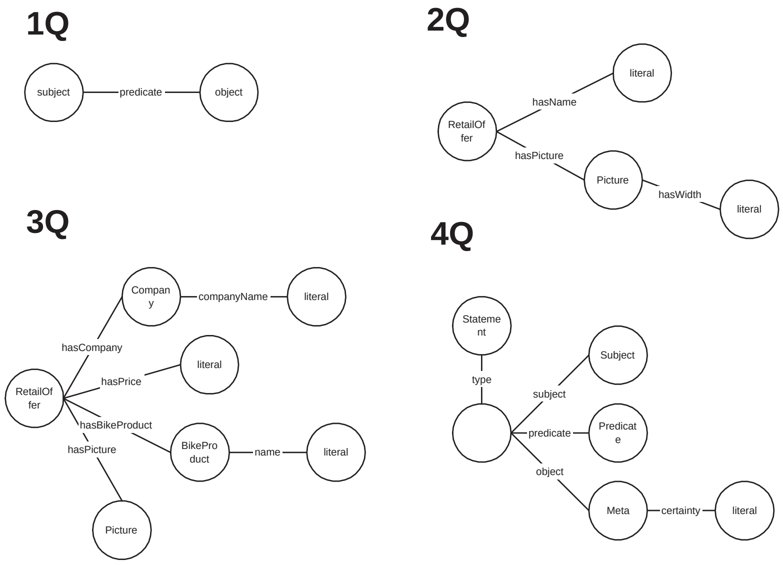


Fig. 3. Path expressions for sample queries

The next example already demonstrates the use of path expressions.
2Q: *Find all retail offers with name starting with letter "l" and having a picture with width lower than 70.*

```
select *
from {X : bike:RetailOffer } bike:hasName {name},
     {X} bike:hasPicture {Y}. pict:hasWidth {width}
where name like "l*" and width < 70
```

We see that in RQL, a variable (denoting a resource) can be assigned class (here, RetailOffer) using *shortcut notation* (with colon). This query also features lexicographic and numerical *comparisons* and a slightly more complicated *path expression*. It contains two paths starting from the same node specified by variable X. In the node specified by variable Y, the path is extended across the arc (property) hasWidth (see also Fig. 3).

RQL is however not very suitable for expressing *optional* path expressions, as manifested on the following example (note the last sentence).
3Q: *Find all retail offers of bicycles that have a concrete bike component. Output the name of company that offers the bike, the picture of retail offer, the price of bike (offer). Retrieve the retail offer even if the URL of picture is not known.*

In *RQL* must be this type of query expressed by applying a Boolean union on the results of two partial queries; also notice the use of operator `in` (reference to embedded query) in the second subquery, in order to eliminate duplicate results (obviously, with increasing number of optional parts of the query graph we would face combinatorial explosion):

```
(select web, company, price, picture, name
 from {X : bike:RetailOffer } bike:hasCompany
      {web : comp:Company }.  comp:companyName {company},
      {X} bike:hasPrice {price},
      {X} bike:hasPicture {picture},
      {X} bike:hasBikeProduct {idtyp}. bike:name {name}
 where  idtyp=data:part1)
union
(select web, company, price, null, name
 from {X : bike:RetailOffer } bike:hasCompany
      {web : comp:Company }.  comp:companyName {company},
      {X} bike:hasPrice {price},
      {X} bike:hasBikeProduct {idtyp}. bike:name {name}
 where  idtyp=data:part1
        and not (X in select X
                      from {X} bike:hasPicture {picture}))
```

4.2 SeRQL

SeRQL [4] ("*Sesame* RDF Query Language", pronounced as 'circle') is a declarative query language over RDF and RDF Schema; in contrast to RQL, there is explicit support for *optional path expressions*. There are two alternative types of queries, SELECT and CONSTRUCT. While SELECT returns a table of results, CONSTRUCT returns again an RDF graph, which is part of the graph being queried or derived from it via introducing new properties or classes. The parts of SELECT have the same meaning as in RQL; the only difference of CONSTRUCT is in the CONSTRUCT clause itself, where a structure of RDF triples appears in the place of variable list. An analogy of RQL functions are SeRQL *built-in predicates*, they however only cover some of the queries that could not be made using the RDF representation of the schemata themselves. This is the case of `<serql:directSubClassOf>`, `<serql:directSubPropertyOf>` and `<serql:directType>`, since the information on e.g. direct vs. inferred subclass relationship is not preserved in the repository in RDF form. In contrast to RQL, *XML datatypes* can be used in queries. SeRQL does not have an implicite sequence of type comparisons: unless the RDF literals themselves are typed (contain information on their datatypes), we must explicitly say how the query engine should compare two expressions in the WHERE clause, e.g. `WHERE width < "150"^^<xsd:positiveInteger>`. In SeRQL, we can, again, use the Boolean connectives AND, NOT and OR (in WHERE clause). In the following, we will translate the examples (1Q, 2Q and 3Q) to SeRQL.

1Q: *Find restrictions (domain and range) of property hasWidth.*
There are no built-in construct for accessing the domain/range. They can however be retrieved in the *RDF representation of the schema*, which is a part of the repository. We also benefit from *shortcut notation*: multiple edges from the same node are separated with semicolon, and the node is not repeated any more:

```
select domain, range
from {<pict:hasWidth>} <rdfs:domain> {domain}; <rdfs:range> {range}
```

2Q: *Find all retail offers that have a name starting with letter "l" and their picture has width lower than 70.*

```
select *
from {X} <bike:hasName> {name},
     {X} <bike:hasPicture> {Y} <pict:hasWidth> {width}
where name like "l*" and width < "70"^^<xsd:integer>
```

3Q: *Find all retail offers of bicycles that have a concrete bike component. Output the name of company that offers the bike, the picture of retail offer, the price of bike (offer). Retrieve the retail offer even if the URL of picture is not known.*

```
select prv, web, company, price, picture, name
from {prv} <serql:directType> {<bike:RetailOffer>};
           <bike:hasPrice> {price};
          [<bike:hasPicture> {picture}];
           <bike:hasBikeProduct> {idtyp},
     {idtyp} <bike:name> {name},
     {prv} <bike:hasCompany> {web} <rdf:type> {<comp:Company>};
                                  <comp:companyName> {company}
where idtyp = <data:part1>
```

The query shows a strong aspect of SeRQL: *optional path expressions* (in brackets). Also notice the combination of shortcut and not-shortcut notations.

The last, new example (we omitted its RQL form for brevity) deals with *reified* statements (with the abstract 'meta' resource, see section 2).
4Q: *Find all statements that have certainty higher than 0.9.*
Queries to reified statements may use their own *shortcut form* in SeRQL:

```
  select *
  from { {reifSubj} reifPred {reifObj} }
           <meta:hasMeta> {obj} <meta:certainty> {certainty}
  where certainty > "0.9"^^<xsd:double>
```

We choose SeRQL for our application, mainly because of the need for *optional path expressions* (since we deal with often incomplete data extracted from HTML pages) and shortcut querying of *reified statements*. The strong aspect of RQL—functions for direct querying of RDF Schema—was found idle for our purpose, since we deal with relatively small and stable schemata.

4.3 Comparison with Other Languages

RDQL [7] was originally developed for the *Jena* tool. Its version in *Sesame* takes into account RDF data with schema but without inferential capability. It allows to specify a path expression but without support for optional parts. The SELECT clause has different syntax but usual meaning. There is no FROM clause, and the graph pattern is specified in the WHERE clause, as list of triples;

partial paths are bound together with variables. Finally, the AND clause specifies filters on variable values and the USING clause maps to namespaces.

Now we demonstrate RDQL on one example: *find all retail offers which have not the name* `liberta` *and their picture has width lower than 70.*

```
SELECT ?retailoffer, ?name, ?picture, ?width
WHERE (?retailoffer, <rdf:type>, <bike:RetailOffer>) ,
      (?retailoffer, <bike:hasName>, ?name) ,
      (?retailoffer, <bike:hasPicture>, ?picture) ,
      (?picture, <pict:hasWidth>, ?width)
AND ( ?width < 70 && ?name ne "liberta")
```

The last query language we mention is part of *PerlRDF*, a collection of tools developed by Ginger Alliance (`http://www.gingerall.com`). It offers path expressions, comparisons, functions and namespaces; there is no support for inferencing nor optional path expressions. We illustrate this RDF query language on the query: *find all retail offers that have a picture with width more than 10.*

```
Select ?retailoffer, ?name,
       ?picture->[http://rainbow.vse.cz/schema/picture.rdfs#hasHeight],
       ?picture->[http://rainbow.vse.cz/schema/picture.rdfs#hasWidth]
From bike:RetailOffer::?retailoffer->bike:hasName{?name},
           ?retailoffer->bike:hasPicture{?picture}
Where
  ?picture->[http://rainbow.vse.cz/schema/picture.rdfs#hasWidth] > '10'
```

5 Query Templates for the Bicycle Application

In order to make our prospective RDF repository available for a casual user, we decided to prepare a domain-specific *HTML interface* with several (SeRQL) *query templates*. The templates should *shield* the user from the syntax of the query language, and even offer a very simple form of *navigational retrieval*. Our idea is based on two-stage querying. The template for *initial query* (specifying its FROM part) is quite complicated, rich in optional path expressions:

```
from {idretail} <rdf:type> {<bike:RetailOffer>};
               [<bike:hasCompany> {idweb}
                     <rdf:type> {<comp:Company>};
                     <comp:companyName> {company};
                     <comp:address> {} <comp:city> {city}];
               [<bike:hasPrice> {price}];
               [<bike:hasPicture> {picture}];
                <bike:hasBikeProduct> {idbike}
                     <rdf:type> {<bike:BikeModel>};
                     <bike:name> {name}
                    [<bike:hasBikePart> {idFork}
                          <rdf:type> {<bike:Fork>};
                          <bike:name> {Fork}];
```

The final shape of the query will be tuned by the user, who may refine the SELECT clause (variables), FROM clause (optional or not), and WHERE clause (comparisons). The results of the initial query are the starting point for *follow-up querying*. For example, the initial query might be: *Find all bikes that are sold by this company*. The results contain various information about each retail offer of the company. There might be e.g. the fact that some offered product has a certain type of frame. Now the user can choose (i.e. click on) the option 'offer', which means: *Query on all retail offers of this product* (i.e., this type of frame). Follow-up queries will be mediated by simpler templates such as for pictures, compaines, retail offers of a particular type of bike or component and so on.

6 Conclusions and Future Work

We discussed the way a concrete RDF storage-and-retrieval tool, *Sesame*, can be used for our specific application within the *Rainbow* project, presented the *RDF schema* for this application, and analysed the *RDF query languages* implemented in *Sesame* Eventually, SeRQL was found suitable for our purposes. We plan to experimentally evaluate our hypotheses on a repository filled with a solid amount of *real-world data*. The data will be accessible through a *domain-specific HTML interface* with pre-fabricated *query templates* as well as through queries manually compiled by the user. In conjuction with this first live experiment, we will also have to *integrate* the results of multiple analytical modules of *Rainbow*.

The project is partially supported by grant no. 201/03/1318 of the Grant Agency of the Czech Republic.

References

1. Alexaki S., Christophides V., Karvounarakis G., Plexousakis D., Tolle K.:The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases, 2^{nd}International Workshop on the Semantic Web, in conjunction with WWW10, Hongkong, 2001.
2. Brickley D., Guha R.V.: RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, World-Wide Web Consortium, Feb. 2004
3. Broekstra J., Ehrig M., Haase P., van Harmelen F., Kampman A., Sabou M., Siebes R., Staab S., Stuckenschmidt H., Tempich C.: A Metadata Model for Semantics-Based Peer-to-Peer Systems. In: Proceedings of the WWW'03 Workshop on Semantics in Peer-to-Peer and Grid Computing, Budapest, 2003.
4. Broekstra J., Kampman A.: User Guide for Sesame. `http://sesame.aidministrator.nl/publications/users/`
5. Broekstra J., Kampman A.: Sesame: A generic Architecture for Storing and Querying RDF and RDF Schema, On-To-Knowledge project deliverable 10, 2001.
6. Hayes P., McBride B.: RDF Semantics. W3C Recommendation, World-Wide Web Consortium, Feb. 2004, `http://www.w3.org/TR/2004/REC-rdf-mt-20040210/`.
7. Seaborne A.:A Programmer's Introduction to RDQL, `http://jena.sourceforge.net/tutorial/RDQL`, April 2002
8. Svátek V., Kosek J., Labský M., Bráza J., Kavalec M., Vacura M., Vávra V., Snášel V.: Rainbow - Multiway Semantic Semantic Analysis of Websites. In: 2^{nd} DEXA Int'l Workshop on Web Semantics, Prague, IEEE Computer Society Press 2003.

On Efficient Part-match Querying of XML Data*

Michal Krátký, Marek Andrt

Department of Computer Science, VŠB – Technical University of Ostrava
17. listopadu 15, 708 33 Ostrava–Poruba
{michal.kratky,marek.andrt}@vsb.cz
Czech Republic

Abstract. The XML language have been becoming de-facto a standard for representation of heterogeneous data in the Internet. From database point of view, XML is a new approach to data modelling. Implementation of a system enabling us to store and query XML documents efficiently (so called native XML databases) require a development of new techniques. The most of XML query languages are based on the language XPath and use a form of path expressions for composing more general queries. These languages make it possible a part-match querying string values of elements and attributes. Particularly, such queries are common for document-centric XML documents. The document-centric documents are often widely unstructured, contain the mixed content and so on. In particular, such documents are (after a transformation to well-formed XML documents) entire information of broad Web. Previously published multi-dimensional approaches to indexing XML data use paged and balanced multi-dimensional data structures. In the paper we extend the approach for the part-match querying XML data.

Key words: XML, indexing XML data, multi-dimensional data structures, part-match querying, XPath, XQuery

1 Introduction

The mark-up language *XML (eXtensible Markup Language)* [22] is recently understood as a language for data representation. Important properties of the language are heterogeneity, extensibility, and flexibility. From database point of view, the XML is a new approach to data modelling [18]. A *well-formed* XML document or a set of documents is an XML database and the associated DTD or schema specified in the language *XML Schema* [23] is its database schema. Implementation of a system enabling us to store and query XML documents efficiently (so called *native XML databases*) requires a development of new techniques [18]. A number of languages have been developed for querying over XML data e.g., *XML-QL* [8], *XPath* [21], and *XQuery* [20]. The common feature of such languages is the usage of regular path expressions for formulation of the

* Work is partially supported by Grant of GACR No. 201/03/0912.

path in the graph modelling an XML document. Such a path is a sequence of element or attribute names from the root element to a leaf.

The XML data are instance of *semistructured* data. An unstructured data may occur into the structured elements. Further, an XML document can be classified on the basis of contained data, as a *data-centric* or *document-centric*. The data-centric XML documents have got well defined regular structure and capture a structured data, e.g. forms. Such data is often possible to map in a set of relations [17]. On the other hand, the *document-centric* documents are often much unstructured, contain fewer elements with amount of unstructured data (e.g. an XML database of articles). However the most of XML documents are combined from both types (so-called *hybrid* documents).

Languages as the XQuery and XPath contain many constructs for querying both data-centric and document-centric XML documents. Such languages provide structures for part-match querying values of elements and attributes. For example, the XPath language allows a filtration unmeant elements in a result set using the *predicate filter*. One from the predicate filters is filter category applicable to string values of elements. For example, the function *contains()* provides a selection of the elements, which contain a substring specified as a parameter. Since a structure of the document-centric documents is not regular, languages were necessary to extend to operators which provide a querying data with *mixed content*. Consequently, such systems extend classical *information retrieval* (*IR*) models for querying XML data. For example, a query retrieving the elements containing an ordered term or phrase are required in the case of querying document-centric XML documents. In Chapter 2 some existing query languages and index approaches for part-match querying XML data are described.

This paper addresses the indexing and querying the document-centric XML documents using multi-dimensional data structures. Chapter 3 presents previously published multi-dimensional approach to indexing data-centric XML documents, particularly how to map paths to points in a multi-dimensional space. Our approach enables an efficient accomplishment of querying text content of an element or an attribute value as well as of queries based on regular path expressions and XPath axes. Chapter 4 extends this approach for part-match querying XML data. Chapter 5 describes multi-dimensional data structures *UB-tree* [2] and *R^*-tree* [3], which are used for indexing XML document. In conclusion we summarize the paper content and outline possibilities of a future work.

2 State of the art

Recently there are many languages and algorithms deal with matching phrases in XML documents with a mixed content. Now, some of them are described. The XQuery-IR [7] is an extension of the XQuery language which supports phrase matching in document fragments and ranks them according to their relevance by using $TF \times IDF$ weights. A tenet of this weight consists in preference terms that occur frequently with one fragment and infrequently in the rest of document. The XXL [19] is a system with similar syntax like SQL language and using a part-match operator allowed querying conformable terms contained in the element or attribute name. The XIRQL [9] language exploits weights and vague predicates, using appropriate DTD, and creates disjoint index contexts for $TF \times IDF$ weights. The weights are applied to rank of relevant document parts regarding to a specified query.

The XKeyword [12] system applies the rank based on a graph distance between the matched words and allows matching words anywhere in a document. Algorithm *PIX (Phrase matching In XML)* [2] for phrase and similar phrase matching in XML documents does not need a exact path specification like XPath. This algorithm provides a phrase matching overlapping separate elements. The *TIX (Text In XML)* algebra [1] uses the scored pattern tree which contains formulas of boolean combination of predicates (applicable to nodes), a set of scoring function (calculate of the score for each node) and also edges labelled in the sense of XPath axis. Operators as the selection, projection, join, and so on are defined under TIX algebra and enable a ranking relevant elements which contain the phrase in dependence on a document structure.

3 Multi-dimensional Approach to Indexing XML Data

In [13] a multi-dimensional approach to indexing XML data was introduced. A revision of this approach was described in [14]. This approach applies multi-dimensional data structures (see Section 5) to indexing XML data.

3.1 Model of XML documents

An XML document may be modelled by a tree, whose nodes correspond to elements and attributes. String values of elements or attributes or empty values occur in leafs. An attribute is modelled as a child of the related element. Consequently, an XML document may be modelled as a set of paths from the root node to all leaf nodes. Note, unique number $id_U(u_i)$ of a node u_i (element or attribute) is obtained by counter increments according to the document order [11]. Unique numbers may be obtained using an arbitrary numbering schema. Of course, the document order must be preserved.

Let $\mathcal{P}$ be a set of all paths in a XML tree. The path $p \in \mathcal{P}$ in an XML tree is sequence $id_U(u_0), id_U(u_1), \ldots, id_U(u_{\tau_P(p)-1}), s$, where $\tau_P(p)$ is the length of the path p, s is `PCDATA` or `CDATA` string, $id_U(u_i) \in D = \{0, 1, \ldots, 2^{\tau_D} - 1\}$, τ_D is the chosen length of binary representation of a number from domain D. Node u_0 is always the root node of the XML tree. Since each attribute is modelled as a super-leaf node with `CDATA` value, nodes $u_0, u_1, \ldots, u_{\tau_P(p)-2}$ represent elements always.

A labelled path lp for a path p is a sequence $s_0, s_1, \ldots, s_{\tau_{LP}(lp)}$ of names of elements or attributes, where $\tau_{LP}(lp)$ is the length of the labelled path lp, and s_i

```
<!DOCTYPE books [
  <!ELEMENT books(book)>
  <!ELEMENT book(title,author)>
  <!ATTLIST book id CDATA #REQUIRED>
  <!ELEMENT title(#PCDATA)>
  <!ELEMENT author(#PCDATA)>
]>
```

```
<?xml version="1.0" ?>
<books>
 <book id="003-04312">
  <title>The Two Towers</title>
  <author>J.R.R. Tolkien</author>
 </book>
 <book id="001-00863">
  <title>The Return of the King</title>
  <author>J.R.R. Tolkien</author>
 </book>
 <book id="045-00012">
  <title>Catch 22</title>
  <author>Joseph Heller</author>
 </book>
</books>
```

Fig. 1. (a) DTD of documents which contain information about books and authors. (b) Well-formed XML document valid w.r.t. DTD.

is the name of the element or attribute belonging to the node u_i. Let us denote the set of all labelled paths by $\mathcal{LP}$. A single labelled path belongs to a path, one or more paths belong to a single labelled path. If the element or attribute is empty, then $\tau_P(p) = \tau_{LP}(lp)$, else $\tau_P(p) = \tau_{LP}(lp) + 1$.

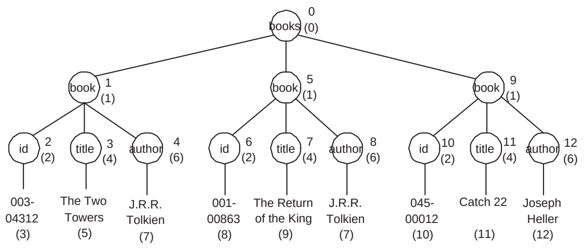


Fig. 2. Example of XML tree with unique numbers $id_U(u_i)$ of elements and attributes u_i and unique numbers $id_T(s_i)$ of names of elements and attributes and their values s_i (values in parenthesis).

Example 1 (Decomposition of XML tree to paths and labelled paths).
In Figure 1 we see an example of an XML document. In Figure 2 we see an XML tree modelling the XML document. We see that this XML document contains paths:

– `0,1,2,'003-04312'`; `0,5,6,'001-00863'` ; and `0,9,10,'045-00012'` belong to the labelled path `books,book,id`,

– `0,1,3,'The Two Towers'`; `0,5,7,'The Return of the King'`; and `0,9,11, 'Catch 22'` belong to the labelled path `books,book,title`,

– `0,1,4,'J.R.R. Tolkien'`; `0,5,8,'J.R.R. Tolkien'`; and `0,9,12,'Joseph Heller'` belong to the labelled path `books,book,author`.

The *term index* which contains all strings s_i of an XML document and their unique numbers $id_T(s_i)$ is used in this approach.

Definition 1 (point of n-dimensional space representing a labelled path).
*Let $\Omega_{LP} = D^n$ be an n-dimensional space of labelled paths, $|D| = 2^{\tau_D}$, and $lp \in \mathcal{LP}$ be a labelled path $s_0, s_1, \ldots, s_{\tau_{LP}(lp)}$, where $n = max(\tau_{LP}(lp), lp \in \mathcal{LP}) + 1$. **Point of n-dimensional space representing a labelled path** is defined $t_{lp} = (id_T(s_0), id_T(s_1), \ldots,\ id_T(s_{\tau_{LP}(lp)})) \in \Omega_{LP}$, where $id_T(s_i)$ is a unique number of term s_i, $id_T(s_i) \in D$. A unique number $id_{LP}(lp_i)$ is assigned to lp_i.* ■

Definition 2 (point of n-dimensional space representing a path).
*Let $\Omega_P = D^n$ be an n-dimensional space of paths, $|D| = 2^{\tau_D}$, $p \in \mathcal{P}$ be a path $id_U(u_0), id_U(u_1), \ldots, id_U(u_{\tau_{LP}(lp)}), s$ and lp a relevant labelled path with the unique number $id_{LP}(lp)$, where $n = max(\tau_P(p), p \in \mathcal{P}) + 2$. **Point of n-dimensional space representing path** is defined $t_p = (id_{LP}(lp), id_U(u_0), \ldots,$ $id_U(u_{\tau_{LP}(lp)}),\ id_T(s)) \in \Omega_P$.* ■

We define three indexes:

1. **Term index**. This index contains a unique number $id_T(s_i)$ for each term s_i (names and text values of elements and attributes). The unique numbers can be generated by counter increments according to the document order. We want to get a unique number for a term and a term for a unique number as well. This index can be implemented by the B-tree.
   In Figure 2 we see the XML tree with unique numbers of terms in parenthesis.
2. **Labelled path index**. Points representing labelled paths together with labelled paths' unique numbers (also generated by counter increments) are stored in the labelled path index.
   In Figure 2 we see that the document contains three unique labelled paths `books,book,id`; `books,book,title`; and `books,book,author`. We create points `(0,1,2)`; `(0,1,4)`; and `(0,1,6)` using id_T of element's and attribute's names. These points are inserted into a multi-dimensional data structure with id_{LP} `0`, `1`, and `2`.
3. **Path index**. Points representing paths are stored in the path index.
   In Figure 2 we see unique numbers of elements. Let us take the path to the value `The Two Towers`. Relevant labelled path `books,book,title` has got id_{LP} 1 (see labelled path index). We get point `(1,0,1,3,5)` after inserting unique numbers of labelled path id_{LP}, unique numbers of elements id_U and term `The Two Towers`. This point is stored in a multi-dimensional data structure.

An XML document is transformed to points of vector spaces and XML queries are implemented using a multi-dimensional data structure queries. The multi-dimensional data structures provide a nature processing of *point* or *range queries* [2]. The point query probes if the vector is or is not present in the data structure. The range query searches all points in a query box $T_1 : T_2$ defined by two points T_1, T_2.

3.2 Queries for values of elements and attributes

Now, implementation of a query for values of elements and attributes and query defined by a simple path based on an ancestor-descendent relation will be described. Query processing is performed in three phases which are connected:

1. **Finding unique numbers id_T of query's term in the term index**.
2. **Finding labelled paths' id_{LP} of query in the labelled path index**. We search the unique numbers in a multi-dimensional data structure using point or range queries.
3. **Finding points in the path index**. We find points representing paths in this index using range queries. Now, we often want to retrieve (using labelled paths and term index) names or values of elements and attributes.

4 Efficient part-match querying of XML data

Now, an extension of the multi-dimensional approach for part-match querying XML data is described. We aim to querying individual terms of the element and attribute string values mainly. Operator $\sim=$ is defined for such query. The individual terms must be indexed, but we need preserve an information about pertinence of the term to the path and labelled path. The *Path-Labelled path-Term* (*PLT*) index satisfies such requirements. This storage contains points of an 3-dimensional space $\Omega_{PLT} = D_{id_P} \times D_{id_{LP}} \times D_{id_T}$. Consequently, items of the space are points $(id_P(p_i), id_{LP}(lp_i),\ id_T(t_i))$. In order to the index can be used for a part-match querying, a unique number $id_P(p_i)$ of path p_i is stored in the first coordinate of the point representing the path p_i: $t_{p_i} = (id_P(p_i), id_{LP}(lp_i), id_U(u_0)$, $id_U(u_1), \ldots,\ id_U(u_{\tau_P(p_i)}))$. During a parsing string values of elements and attributes we could use the stop-list known in IR systems [3]. For example, frequent terms (e.g. conjunctions) are eliminated by the stop-list. Such terms are not important for a querying. Since whole values of elements and attributes are important and $it_T(t_i)$ of the whole string t_i is removed in the point representing the path, whole short values are inserted in the term index and PLT index.

Example 2 (Creation of the PLT index).

Let us take a document-centric XML document. For example, Shakespeare's Hamlet in XML [6]. $id_{LP}('PLAY, SCENE, ACT, SPEECH, SPEAKER') =$ 100 and id_P of belonging path is 110 (see Figure 3).

```
<PLAY>...
  <SCENE>...
    <ACT>...
      <SPEECH>
        <SPEAKER>MARCELLUS</SPEAKER>
        <LINE>It faded on the crowing of the cock.</LINE> ...
      </SPEECH>
      ...
```

Fig. 3. A part of Shakespeare's Hamlet in XML.

$id_{LP}('PLAY, SCENE, ACT, SPEECH, LINE') = 101$ and id_P of belonging path is 111. Unique numbers of terms: $id_T('MARCELLUS') = 120$, $id_T('crowing') = 121$, $id_T('cock') = 122$, and so on. After the insertion of points representing the path, labelled path, $id_T(t_i)$ and term t_i into the term index, the points are created and inserted into the PLT index: $(110, 100, 120)$, $(111, 101, 121)$, $(111, 101, 122)$, and so on. ■

Now, processing the query `/books/book[keywords∼='XML']/title` over an XML database of books is described. Note, query box $(qb_1, min(D), \dots, min(D)) : (qb_1, max(D), \dots, max(D))$ may be written as $(qb_1, *, \dots, *)$.

1. Finding $id^1_{LP} = id_{LP}('books, book, keywords')$ and $id^1_T = id_T('XML')$.
2. Processing the narrow range query $(*, id^1_{LP}, id^1_T)$ in the PLT index. The result is k unique numbers $id_P(p_1), \dots, id_P(p_k)$ of relevant paths $p_1, \dots, p_k$.
3. Processing the complex range query $(id_P(p_1), id^1_{LP}, *, \dots, *)$,...,$(id_P(p_k), id^1_{LP}, *, \dots, *)$. The result is the points representing the relevant paths.
4. Finding $id^2_{LP} = id_{LP}('books, book, title')$.
5. Performing the child XPath axis with id^2_{LP} in the second coordinate. The child XPath axis is implemented by a sequence of range queries (see [14]). The result is m paths $p^f_1, \dots, p^f_m$.
6. Performing the complex range query $(id_P(p^f_1), id^2_{LP}, *), \dots, (id_P(p^f_m), id^2_{LP}, *)$. An output is collection of $id_T(t_i)$. Strings of titles t_i are retrieved from the term index and the strings are returned as a result.

Query processing of a general part-match query is a generalization of above described procedure. The XML query languages make it possible to place a complex query condition using boolean operators e.g., `AND` and `OR`. In described approach a query defined by the `OR` operator is possible to process effectively. For example, the query `/books/book[keywords∼='XML' OR keywords∼='SGML']/title` is performed according to above techniques, but the first two steps are distinguish.

1. Finding $id^1_{LP} = id_{LP}('books, book, keywords')$, $id^1_T = id_T('XML')$, and $id^2_T = id_T('SGML')$.

2. Processing the narrow range queries $(*, id^1_{LP}, id^1_T)$ and $(*, id^1_{LP}, id^2_T)$ in the PLT index. The result is k unique numbers $id_P(p_1), \dots, id_P(p_k)$ of relevant paths $p_1, \dots, p_k$.

Next steps of this query processing are the same.

5 Index Data Structures

Due to the fact that an XML document is represented as a set of points representing paths and labelled paths in the multi-dimensional approach, we use multi-dimensional data structures for their indexing, e.g., paged and balanced multi-dimensional data structures like UB-tree [2], and R*-tree [3].

(B)UB-tree data structure applies *Z-addresses* (*Z-ordering*) [2] for mapping a multi-dimensional space into single-dimensional. Intervals on *Z-curve* (which is defined by this ordering) are called *Z-regions.* (B)UB-tree stores points of each Z-regions on one disk page (tree leaf) and a hierarchy of Z-regions forms an index (inner nodes of tree). In the case of indexing point data, an R-tree and its variants cluster points into *minimal bounding boxes* (*MBB*s). Leafs contain indexed points, super-leaf nodes include definition of MBBs and the other inner nodes contain hierarchy of MBBs. (B)UB-tree and R-tree support *point* and *range queries* [11], which are used in the multi-dimensional approach to indexing XML data. The range query is processed by iterating through the tree and filtering of irrelevant tree nodes, i.e. (super)Z-regions in the case of (B)UB-tree and MBBs in the case of R-tree, which do not intersect a query box.

One more important problem of the multi-dimensional approach is the unclear dimension of spaces of paths and labelled paths. A naive approach is to align the dimension of space to the maximal length of path. For example, points of dimension 5 will be aligned to dimension 36. This technique increases the size of index and the overhead of data structure as well. In [15] BUB-forest data structure was published. This data structure solves the problem of indexing points with different dimensions. The range query used in the multi-dimensional approach is called *narrow range query.* Points defining a query box have got some coordinates the same, whereas the size of interval defined by other coordinates near to the size of space's domain. Many irrelevant regions are searched during processing the narrow range query in multi-dimensional data structures. In [9] Signature R-tree data structure was introduced. This data structure enables efficient processing the narrow range query.

6 Conclusion

In our future work we would like to test this approach over a current test XML document collections. Test queries are often defined for such collections. Therefore a comparison of our approach with another XML indexing approaches is possible. INEX [10] collection seems to be hopeful. INEX contains 12,000 IEEE articles since 1995. The size of the collection is 500MB.

References

1. S. Al-Khalifa, C. Yu, and H. Jagadish. Querying Structured Text in an XML Database. In *Proceedings of International Conference on Management of Data (SIGMOD), San Diego, CA*, June 2003.
2. S. Amer-Yahia, M. Fenández, D. Srivastava, and Y. Xu. Phrase Matching in XML. In *Proceedings of the 29th VLDB Conference, Berlin, Germany*, 2003.
3. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison Wesley, New York, 1999.
4. R. Bayer. The Universal B-Tree for multidimensional indexing: General Concepts. In *Proceedings of WWCA'97, Tsukuba, Japan*, 1997.
5. 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 International Conference on Management of Data*, pages 322–331.
6. J. Bosak. Shakespeare in XML, 1999, `http://www.ibiblio.org/xml/examples/shakespeare/`.
7. J.-M. Bremer and M. Gertz. XQuery/IR: Integrating XML document and data retrieval. In *Proceedings of the 5th International Workshop on the Web and Databases (WebDB)*, June 2002.
8. A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. XML-QL: A Query Language for XML. Technical report, WWW Consortium, August, 1998.
9. N. Fuhr and K. Grossjohann. XIRQL: An extension of XQL for information retrieval. In *Proceedings of SIGIR*, 2001.
10. N. Fuhr, N. Gvert, S. Malik, M. Lalmas, and G. Kazai. INEX – Initiative for the Evaluation of XML Retrieval, 2003, `http://www.is.informatik.uni-duisburg.de/projects/inex/index.html.en`.
11. T. Grust. Accelerating XPath Location Steps. In *Proceedings of ACM SIGMOD 2002, Madison, USA*, June 4-6, 2002.
12. V. Hristidis, Y. Papakonstantinou, and A. Balmin. Keyword proximity search on XML graphs. In *Proceedings of the ICDE*, 2003.
13. M. Krátký, J. Pokorný, T. Skopal, and V. Snášel. The Geometric Framework for Exact and Similarity Querying XML Data. In *Proceedings of First EurAsian Conferences, EurAsia-ICT 2002, Shiraz, Iran.* Springer–Verlag, LNCS 2510, 2002.
14. M. Krátký, J. Pokorný, and V. Snášel. Implementation of XPath Axes in the Multi-dimensional Approach to Indexing XML Data. In *Accepted at International Workshop DataX, Int'l Conference on EDBT, Heraklion - Crete, Greece*, 2004.
15. M. Krátký, T. Skopal, and V. Snášel. Multidimensional Term Indexing for Efficient Processing of Complex Queries. *Kybernetika, Journal of the Academy of Sciences of the Czech Republic, accepted*, 2003.
16. M. Krátký, V. Snášel, J. Pokorný, P. Zezula, and T. Skopal. Efficient Processing of Narrow Range Queries in the R-Tree. In *Submitten at VLDB 2004*, 2004.
17. Q. Li and B. Moon. Indexing and Querying XML Data for Regular Path Expressions. In *Proceedings of 27th VLDB International Conference*, 2001.
18. J. Pokorný. *XML: a challenge for databases?*, pages 147–164. Kluwer Academic Publishers, Boston, 2001.

19. A. Theobald and G. Weikum. The index-based XXL search engine for querying XML data with relevance ranking. In *Proceedings of EDBT*, 2002.
20. W3 Consortium. XQuery 1.0: An XML Query Language, W3C Working Draft, 15 November 2002, `http://www.w3.org/TR/xpath/`.
21. W3 Consortium. XML Path Language (XPath) Version 2.0, W3C Working Draft, 15 November 2002, `http://www.w3.org/TR/xpath20/`.
22. W3 Consortium. Extensible Markup Language (XML) 1.0, 1998, `http://www.w3.org /TR/REC-xml`.
23. W3 Consortium. XML Schema Part 1: Structure, 2001, `http://www.w3.org/TR/xmlschema-1/`.
24. C. Yu. *High-Dimensional Indexing.* Springer–Verlag, LNCS 2341, 2002.

INEX – a Broadly Accepted Data Set for XML Database Processing?*

Pavel Loupal and Michal Valenta

Dept. of Computer Science and Engineering
FEE, Czech Technical University
Karlovo náměstí 13, 121 35 Praha 2
Czech Republic
P.Loupal@sh.cvut.cz, valenta@fel.cvut.cz

Abstract. The aim of the article is to inform about the INEX initiative, its testing data set, actual results, and future plans. We discuss and demonstrate possible utilization of the INEX data set for our own research and testing purposes. Our example – adaptation of approximate tree embedding algorithm - provides a basis for discussion about INEX data set suitability and about eventual consecutive experiments.

1 Introduction

Until now, there is no broadly accepted database nor data set for testing new search or index algorithms or query language specifics in branch of XML processing research. Such referential data set would be useful mainly for more accurately comparing of individual algorithms and approaches.

INEX data set can be discussed as a hot candidate for such purposes, although the INEX initiative focuses itself rather to information retrieval research than to XML query languages aspects. But its data set seems suitable, because it is large enough and its structure is also appropriately complex.

Hence the aim of the article is to inform about INEX initiative, its background, participants, plans, and results in order to initiate relevant discussion of accepting or rejecting INEX data set as a referential database for comparing research results.

We have developed a simple web based application which provides access to INEX data set and enables easy algorithm testing and results evaluation. We have prepared an example – adaptation of approximate tree embedding algorithm – in order to provide couple of concrete arguments for discussion about INEX data set relevance.

The paper is organized as follows: The second section brings basic information about INEX initiative. Subsection 2.1 informs about background, plans, founders, and participants of INEX initiative. Subsection 2.2 discusses INEX

* The research was partially supported by the grant GAČR 201/03/0912

data set structure, organization of consecutive tasks, and several results. The third section is dedicated to our utilization of INEX data set. Subsection 3.1 introduces our approach for accessing the INEX data set. Subsection 3.2 demonstrates the application on concrete example – adaptation of approximate tree embedding algorithm.

2 INEX initiative

2.1 History, participants, purposes

INitiative for the Evaluation of XML retrieval (INEX) was founded three years ago. The motivation and the main aim of the project is to provide a referential database for purposes of data retrieval research community.

Actually 69 participants mainly from universities have taken their active part in INEX project. Concrete list of participant's organizations and responsible persons could be found in INEX home page [5]. In the head of initiative stand Norbert Fuhr, Saadia Malik (Duisburg-Essen University) and Maunia Lalmas (Queen Mary University London).

Project is organized as a set of consecutive steps. Each step consists of the set of tasks which are spred among all participants. When all individual tasks are solved by their responsible participants, the result is considered together and evaluated by participants forum discussion. Then step is closed and project moves to the next stage.

The first step of the project consisted only from acquiring appropriate data set. It was supplied by IEEE – several volumes of IEEE journals. In the second step set of data retrieval queries were developed and evaluated by participants. The third step consisted of hand made relevance assessment process of individual queries. The next step is actually object of participant's discussion. It should be focused to the efectivity of relevance assessment process and to the study of searchers behaviors and also to the topics of distributed data sources. Follow open discussion of the third INEX workshop in [5].

2.2 Data structure, queries, relevance assessments

INEX data set (actual version 1.4) has 536MB of XML data. It is exactly 12,107 articles from 6 IEEE transactions and 12 journals from years 1995 to 2002. Pictures are not included – data set consists only of XML formated text.

Data set is organized in file structure. Root directory consists of two subdirectories – *dtd* (holds structure information - DTD specification article element) and *xml*. Each journal/transaction has its own two-letter named subdirectory inside xml directory. Journal/transaction is further divided into the directories by the year of publication. Finally each article is stored in individual xml file, which name consists of a letter following by four-digit number and xml suffix. Structure is schematically shown in figure 1.

In average each article contains 1,532 XML nodes, where the average depth of node is 6.9. See [3] for detail characteristics of data set.

```
/inex-1.4
        /dtd
             ...
             xmlarticle.dtd
        /xml
             /an
                    /1995
                             ...
                             a1019.xml
                             a1032.xml
                             a1034.xml
                             ...
                    /...
                    /2002
             /...
             /ts
```

Fig. 1. INEX data set file structure

DTD specification or article element is too complex to be clearly presented here. Instead of this more illustrative fragment of typical article is shown in figure 2. Picture is taken from [3].

The second stage of INEX project was focused to construction of suitable data retrieval queries (*topics*). Topics were constructed by individual participants and then were accepted or rejected by discussion forum of all participants. Each participant had to design 6 queries.

Topics were divided into two groups – Content Only (*CO*) and Content And Structure (*CAS*) topics. CAS topics have a structure condition inside their specification, for example they interests only in abstracts etc. CAS topics are then classified either Strict (*SCAS*) or Vague (*VCAS*). The final set of INEX'03 topics consists of 36 CO and 30 CAS queries.

Then each participant did 3 runs of each topic and select the first 1000 most relevant documents. Individual runs were averaged so there was a set of 1000 most relevant documents for each topic. Statistics of individual runs were computed and they are available for participants purposes.

Evaluation stage covers hand made relevance assessments of 1000 documents selected in the previous stage with respect to the given topic. The evaluation was done through web based assessment application, see figure 4. Relevance assessment was expressed by two independent scales – *specificity* and *exhaustivity*.

```
|<article>
|  <fm>
|    ...
|    <ti>IEEE Transactions on ...</ti>
|    <atl>Construction of ...</atl>
|    <au>
|      <fnm>John</fnm>
|      <snm>Smith</snm>
|      <aff>University of ...</aff>
|      </au>
|    </au>...</au>
|    ...
|  </fm>
|  <bdy>
|     <sec>
|    <st>Introduction</st>
|    <p>...</p>
|    ...
|    </sec>
|  <sec>
|    <st>...</st>
|    ...
|    <ss1>...</ss1>
|    <ss1>...</ss1>
|    ...
|    </sec>
|   ...
|  </bdy>
|  <bm>
|    <bib>
|     <bb>
|       <au>...</au><ti>...</ti>
|     ...
|     </bb>
|    ...
|    </bib>
|  </bm>
|</article>
```

Fig. 2. INEX typical article structure

Each scale has three values – *marginal, fairly, high* specific/exhaustive, so element can be maked by one of nine assessment values. The tenth assessment value is *not relevant*. This is the way to express more relevant parts inside the document. Icons were used to express given relevance mark for chosen XML node in INEX assessment interface.

Moreover there are several parent-child dependencies in assigning relevance mark to the element. For example exhaustivity level of a parent element is always equal to or greater than the exhaustivity level of its children elements. These dependencies are fixed and they are automatically checked by assessment interface.

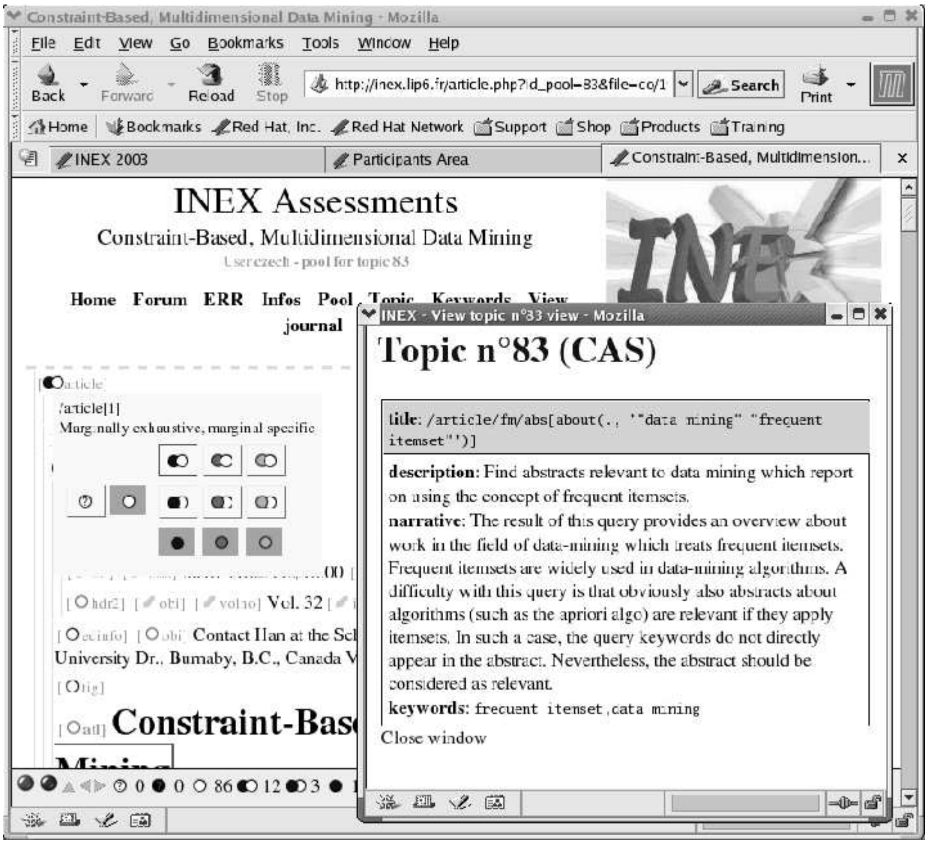
The result of assessment process is published as XML document according to DTD specification is shown in figure 3.

Nowadays, all INEX topics have been processed and there is XML file with assessment results for each topic. This stage of project has been discussed in the second INEX workshop, see [4] for details.

The next stage of INEX project will focus on detail study of searchers behaviors (this research starts from analysis of handy made assessments from previous stage) and also to data retrieval from heterogeneous sources and distributed systems. Actually, details of the further stage of INEX project are discussed in forum of project participants. See discussion in [5] for details.

```
<!ELEMENT assessments (file+)>
<!ATTLIST assessments
          topic          CDATA              #REQUIRED
>
<!ELEMENT file           (path*)>
<!ATTLIST file
          file           CDATA              #REQUIRED
>
<!ELEMENT path           EMPTY>
<!ATTLIST path
          path           CDATA              #REQUIRED
          exhaustiveness ( 0 | 1 | 2 | 3 ) #REQUIRED
          specificity    ( 0 | 1 | 2 | 3 ) #REQUIRED
>
```

Fig. 3. INEX DTD topic assessments


Constraint-Based, Multidimensional Data Mining - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://inex.lip6.fr/article.php?id_pool=83&file=co/1
Search Print
Home Bookmarks Red Hat, Inc. Red Hat Network Support Shop Products Training
INEX 2003 Participants Area Constraint-Based, Multidimension...
INEX Assessments
Constraint-Based, Multidimensional Data Mining
User czech - pool for topic 83
Home Forum ERR Infos Pool Topic Keywords View journal
article
/article[1]
Marginally exhaustive, marginal specific
Vol. 32
Contact Han at the Sch University Dr., Burnaby, B.C., Canada V
Constraint-Base
Mining
0 0 86 12 3
INEX - View topic n°83 view - Mozilla

Topic n°83 (CAS)

title: `/article/fm/abs[about(., '"data mining" "frequent itemset"')]`

description: Find abstracts relevant to data mining which report on using the concept of frequent itemsets.
narrative: The result of this query provides an overview about work in the field of data-mining which treats frequent itemsets. Frequent itemsets are widely used in data-mining algorithms. A difficulty with this query is that obviously also abstracts about algorithms (such as the apriori algo) are relevant if they apply itemsets. In such a case, the query keywords do not directly appear in the abstract. Nevertheless, the abstract should be considered as relevant.
keywords: `frequent itemset, data mining`

Close window

Fig. 4. INEX XML assessment interface

3 Data set utilization

The first step for utilization of the INEX data set for our research purposes is preparation of common interface for data access. As shown in figure 1 data is stored in directories relevant to journals and volumes. Our approach should keep this structure to avoid any disorders. Also for that reason we decided to use a solution based on a native XML database.

3.1 Native XML storage

Native XML databases have some advantages in comparison with ordinary method of storing XML files in a file system. The concept of native XML databases is based on filesystem structure i.e. data can be stored in collections which mean the same as directories in filesystems but additionally this kind of databases has many enhanced features useful for processing XML data.

For our purposes and with respect to INEX structure we decided to build our approach on the Apache Xindice native XML database [6]. This product is an open source project developed by the Apache Software Foundation with several contributors involved into its advancement.

Starting from version 1.1, Xindice is not a standalone server anymore. The server functions are now based on any Servlet 2.2 (or higher) compliant application server - in our case we decided to use Apache Tomcat 5.0.16 application server. Bundling database core functionality into a servlet container allows users not to create only standalone console applications but easily develop also web based applications using e.g. Java Server Pages (we use this technology in our tree-embedding example).

Regarding to user documentation Xindice was not designed for handling huge documents, rather, it was designed for collections of small to medium sized documents. The INEX structure complains optimally this requirement (see section 2.2).

Following built-in key features are important for data processing:

- **Standard API for accessing data -** In this case it is the XML:DB interface [7]. This simple interface is developer-friendly and supports common data processing methods.
- **XPath expresions -** In many applications XPath is only applied at the document level but in Xindice XPath queries are executed at the collection level. This means that a query can be run against multiple documents and the result set will contain all matching nodes from all documents in the collection.
- **Usage of metadata -** Xindice allows an user to store metadata that is associated with any collection or document. Metadata is data that is associated with a document or collection but is not part of that document or collection. This metadata could be used for storing temporar information

e.g. algorithm's subresults or more permanent data such as a list of indexed keywords contained by that collection or document.

Actually there are two ways for developers how to access Xindice database:

- **XML:DB XML Database API** - is used for developing applications in Java language. An example of usage of this Application Programming Interface (API) is shown in figure 5. This example shows an basic approach for retrieving a document from specified collection.
- **Xindice XML-RPC API** - is used when accessing Xindice from language other than Java.

One example of usage of the XML:DB API is shown in figure 5.

```
Collection col = null;
try {
  String driver = "org.apache.xindice.client.xmldb.DatabaseImpl";
  Class c = Class.forName(driver);

  Database database = (Database) c.newInstance();
  DatabaseManager.registerDatabase(database);
  col =
    DatabaseManager.getCollection(
          "xmldb:xindice://nonstop.sh.cvut.cz:8080/db/inex/mu/2001");

  XMLResource document = (XMLResource) col.getResource("a1019.xml");
  if (document != null) {
    // Print out document's content
    System.out.println(document.getContent());
  }
  else {
    System.out.println("Document not found");
  }
}
```

Fig. 5. Example Java code for retrieving an XML document from database

3.2 Example – Approximate tree embedding algorithm

We have addopted an approximate tree embedding algorithm using the INEX data set. Our implementation uses core tree embedding algorithm written by Jan Váňa (see [2]) and is wrapped into a simple graphical user interface. This interface allows user to select a collection (with or without all subcollections) where to search and a query to search for. Details of this algorithm are described in following section.

The algorithm. Approximate tree embedding algorithms were studied in early 90's and Kilpelainen showed that the decision problem whether a tree can be embedd into another is NP-complete. Schlieder [1] created an algorithm which behaviour is polynomial in practical examples. Váňa [2] modified that algorithm and added few improvements in some special boundary cases and exceptional situations.

The core part of this algorithm could be described as *embeddTree*(T_q, T_d), where T_q is a query tree and T_d is data tree. One possible matching mapping between query and data tree is shown in figure 6. Algorithm searches for all embeddings of query tree in data tree and also returns a rating for each match - this "cost" basically means the number of elements which had to be skipped when embedding tree.

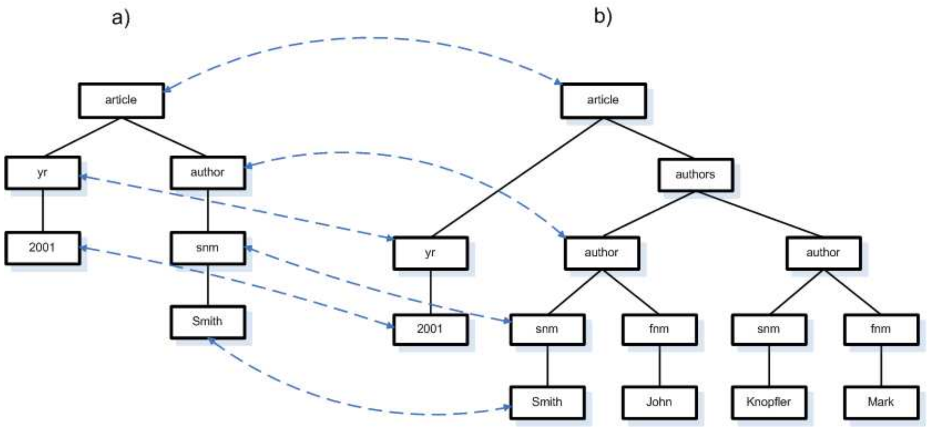
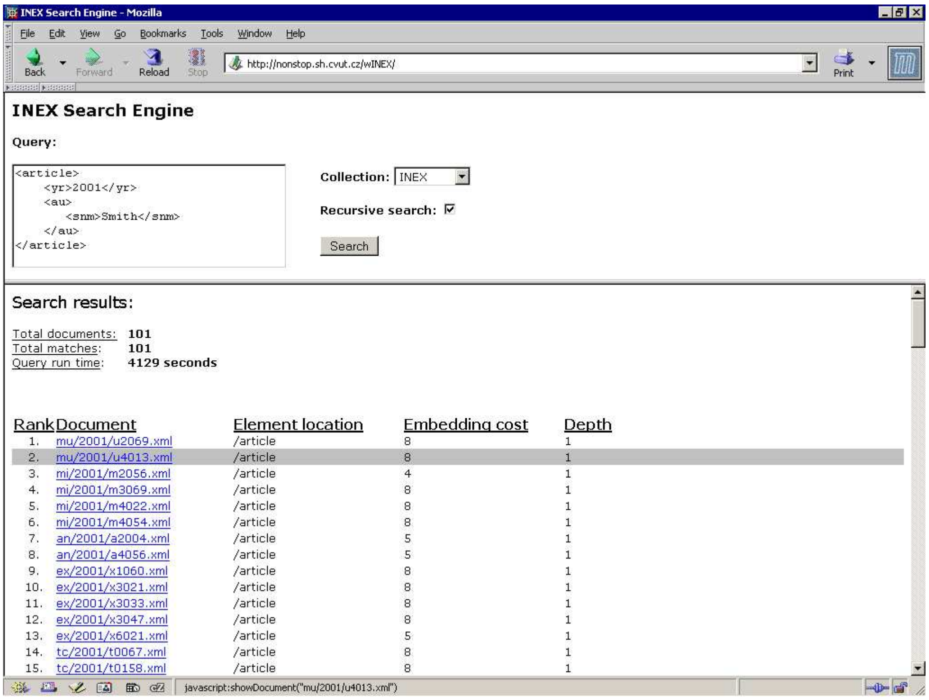


Fig. 6. Example matching between two trees. a) query tree T_q, b) data tree T_d

This paper is not primarily addressed to discuss this problem. This example was chosen for showing adaptation of such algorithm over the INEX data set. Detailed description of this algorithm can be found in [1], [2].

Implementation. We have created an application which integrates approximate tree embedding algorithm with access to documents stored in XML database. Web-based user interface allows user submit his query over specified collection and get results with their ranking. Our algorithm traverses (recursively) specified collection, fetches list of XML resources stored in and tries to embedd query tree into all documents.

Actually algorithm uses two metrics for rank matched result. The first one is based on number node which need to be skipped when embedding tree and the second one is the level of root node of the query tree T_q in data tree T_d.


INEX Search Engine - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://nonstop.sh.cvut.cz/wINEX/
Print
Search
javascript:showDocument("mu/2001/u4013.xml")

INEX Search Engine

Query:

```
<article>
    <yr>2001</yr>
    <au>
       <snm>Smith</snm>
    </au>
</article>
```

Collection: INEX

Recursive search: ✓

Search results:

Total documents:	101
Total matches:	101
Query run time:	4129 seconds

Rank	Document	Element location	Embedding cost	Depth
1.	mu/2001/u2069.xml	/article	8	1
2.	mu/2001/u4013.xml	/article	8	1
3.	mi/2001/m2056.xml	/article	4	1
4.	mi/2001/m3069.xml	/article	8	1
5.	mi/2001/m4022.xml	/article	8	1
6.	mi/2001/m4054.xml	/article	8	1
7.	an/2001/a2004.xml	/article	5	1
8.	an/2001/a4056.xml	/article	5	1
9.	ex/2001/x1060.xml	/article	8	1
10.	ex/2001/x3021.xml	/article	8	1
11.	ex/2001/x3033.xml	/article	8	1
12.	ex/2001/x3047.xml	/article	8	1
13.	ex/2001/x6021.xml	/article	5	1
14.	tc/2001/t0067.xml	/article	8	1
15.	tc/2001/t0158.xml	/article	8	1

Fig. 7. Simple web interface for querying database

Results. Our adaptation of the approximate tree embedding algorithm allows user to get results as supposed by its description, i.e. it finds all embeddings in documents contained in specified collection(s). The correctness of the implementation was not proved exactly but our queries and their respective results were manually checked for match.

Performace. Our implementation uses platform independent Java XML:DB interface outlined above. This approach is probably slower than a similar native solution written in language like C++ but in this case speed is not the crucial requested property.

To get an estimation of time consumption when performing our algorithm we ran a simple query (see figure 8) on a computer with the Intel Pentium III processor (500MHz) with 256 MB of memory. This run over the complete INEX data set took about 68 minutes and returned 101 matches.

Usability. In spite of correctness of the algorithm, practical benefit is a moot question. The main drawback is strict comparison of data items used - the content of an element must exactly match data in given query. For practical purposes users would usually ask queries not about structual match but more on approx-

```
<article>
    <yr>2001</yr>
    <au>
        <snm>Smith</snm>
    </au>
</article>
```

Fig. 8. Query example - all articles published in the year 2001 written by author with surname "Smith"

imate content. Therefore, operators such as *contains()* or *starts-with()* known from XPath could extend practical algorithm applicability.

4 Conclusions

The INEX data set is a huge collection of "real" and meaningful documents. Storing such data in native XML database (in our case Apache Xindice) allows researchers to implement and test wide range of algorithms over these data. Technical environment supports developing both console and web-based Java applications using common standards for manipulating XML data.

Our example, approximate tree embedding algorithm, shows one example of possible utilization of the INEX data set. It is just one of possible implementations but the common access interface allows researches to "plug in" another solution of the embedding algorithm or even to use another algorithm which has a different goal.

Further work on this algorithm should be focused on experiments with order evaluating function. Some hypothesis about local and global rate of the order function had been stated in an unformal discussion in the last DATESO workshop. These hypothesis should be formalized, implemented, and proved on a "real" data set.

In addition the INEX data set has been adopted into our frame and can be used for arbitral experiments in the branch of XML database processing research. Although it is only a side-effect of the INEX project, we have shown in this article, it can be used also for our own research with a good benefit.

References

1. Schlieder, T., Naumann, F.: Approximate tree embedding for querying XML data. In ACM SIGIR Workshop On XML and Information Retrieval, Athens, Greece, 2000.
2. Váňa, J.: Integrity of XML data (*in Czech*). Master Thesis, Dept. of Software Engineering, Charles University, Prague. 2001.

3. Fuhr, N., Gvert, N., Kazai, G., Lalmas, M.: INitiative for the Evaluation of XML retrieval (INEX). Proceedings of the First INEX Workshop.ERCIM Workshop Proceedings. ERCIM, Sophia Antipolis, France, 2003.
4. Fuhr, N., Malik, S., Kazai, G., Lalmas M. (*Editors*): Proceedings of the 2nd Initiative on the Evaluation of XML Retrieval (INEX 2003). ERCIM Workshop Proceedings. 2003
5. INEX 2003 - home page. http://inex.is.informatik.uni-duisburg.de:2003/index.html.
6. Apache Xindice - Native XML database. http://xml.apache.org/xindice.
7. XML:DB initiative - Application Programming Interface for accessing native XML databases. http://www.xmldb.org.

Query Expansion and Evolution of Topic in Information Retrieval Systems⋆

Jiří Dvorský, Jan Martinovič, and Václav Snášel

Department of Computer Science, VŠB-Technical University of Ostrava,
17. listopadu 15, Ostrava - Poruba, Czech Republic
{jiri.dvorsky,jan.martinovic,vaclav.snasel}@vsb.cz

Abstract. Approach based on clustering will be described in our paper. Basic version of our system was given in [5] allows us to expand query through special index. Hierarchical agglomerative clustering of the whole document collection generates the index. Retrieving of topic development is specific problem. Standard methods of IR does not allow us such kind of queries for appropriate solution of information problem. The goal of presented method is to find list of documents that are bearing on topic, represented by user-selected document, sorted with respect to historical development of the topic.

1 Introduction

There are plenty of large text collections in the world. In connection with expansion of Internet these collections get bigger and bigger. Amount of processed data reach in present time dimensions that statistical properties of texts in collection become evident. This fact leads to new approaches, which involve methods from statistics, linear algebra, neural networks, and other (see [1, 9]).

Another important feature of these collections is their dynamic character. Modern surveys of information retrieval (IR) supposes that text collections have static character. Prior knowledge of topics' distribution is other presumption of current IR methods. Omission of these presumptions is more adequate to today's demands [2].

Retrieving of topic development is specific problem. Let's imagine that we want to perform query about war in Iraq from open source text collection. Set of terms contained in documents describing initial part of the war will be different from set of terms in document that characterize current state of war. Standard methods of IR [3] does not allow us such kind of queries for appropriate solution of information problem.

There are many IR systems based on Boolean, vector, and probabilistic models. All of them use their model to describe documents, queries, and algorithms to compute relevance between user's query and documents. Each model contains some constraints. Constrains cause disproportion between expected (relevant)

⋆ This work was done under grant from the Grant Agency of Czech Republic, Prague No.: 201/03/1318 documents and documents returned by IR system. One of the possibilities how to solve the disproportion are systems for automatic query expansion, and topic development observing systems.

Approach based on clustering (see [7]) will be described in this paper. Basic version of our system was given in [5]. This version allows us to expand query through special index. Hierarchical agglomerative clustering of the whole document collection generates the index [6, 5].

Basic definitions of vector model will be briefly repeated in section 2. Section 3 contains introduction to cluster analysis, and description of agglomerative clustering. Section 4 is dedicated to query expansion algorithms that are based on work [5], including conclusions from tests. Description of topic development observing system, and its relationship to clustering algorithms are given in section 5. Section 6 gives us some conclusions and presents possibilities of future works.

2 Vector model

Vector model is dated back to 70th of the 20th century. The main goal of vector model is to enhance IR system based on Boolean model. Let's suppose vector IR system containing information about n *documents.* The documents are indexed by set of m *terms.* m dimensional vector represents each document in document collection, where every part of the vector corresponds to weight of particular term in given document. Formally:

$$d_i = (w_{i,1}, w_{i,2}, \ldots, w_{i,m}) \in \langle 0, 1 \rangle^m$$

Index of vector IR systems is the represents by matrix

$$D = \begin{pmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,m} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n,1} & w_{n,2} & \cdots & w_{n,m} \end{pmatrix} \in \langle 0, 1 \rangle^{n \times m}$$

Query in vector model is again m dimensional vector:

$$Q = (q_1, q_2, \ldots, q_m) \in \langle 0, 1 \rangle^m$$

Similarity between query Q and each document d_i can be computed as

$$Sim(Q, d_i) = \frac{\sum_{k=1}^{m} q_k w_{i,k}}{\sqrt{\sum_{k=1}^{m} (q_k)^2} \sqrt{\sum_{k=1}^{m} (w_{i,k})^2}}$$

There are many formulæ how to compute the similarity, we use one of the most frequent - cosine measure. The similarity can be understood as "distance" between query vector and vectors of documents in some vector subspace defined by matrix M.

Detail information about vector model can be found for example in [3].

3 Cluster analysis

The weight matrix M described above represents vast amount of numbers, that can be interpreted in some way. Among others, there is possibility to put documents together, which have approximately same coefficient of similarity to the potential query.

Finding of group of objects with the same or similar features within given set of objects i the goal of cluster analysis. These groups are called *clusters*. In other words, the group of similar objects forms the cluster.

Hierarchical clustering methods are important tool of cluster analysis. Hierarchical methods create hierarchy of clusters, grouped in cluster levels. The cluster levels arise during the computation and represents structure of hierarchy.

Hierarchical clustering methods can be divided into two groups:

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

divisive - The method works in reverse manner. At the beginning there is one cluster containing all objects. The clusters are sequentially divided until each cluster contains only one object.

Agglomerative clustering algorithm

1. **Create matrix of objects' distances**
   Matrix of objects' distances will be equal to term-document weight matrix D.
2. **Define each object as cluster**
   At the beginning each object is considered as one cluster i.e. there are as many clusters as objects. Sequentially, clusters are joined together and number of clusters drops down, when finally there is one cluster.
3. **Join pair of clusters with the least mutual distance**
   There many strategies, how to compute the distance. Among the most frequently used strategies belong:
   - *strategy of forthcoming neighbour* - Distances among all objects in two clusters are computed. The distance of clusters is then defined as minimal distance between any objects in these two clusters.
   - *strategy of farthermost neighbour* - Same as strategy of forthcoming neighbour, but maximum distance is chosen.
   - *average distance strategy* - Distances among all objects in two clusters are computed. The distance of clusters is then defined as average distance among any objects in these two clusters.
   - *median strategy* - The distance of clusters is then defined as median of distances among objects in these two clusters.
   - *Ward's method* - two cluster are joined together, if increase of sum of distances from centroid of clusters is minimal.

4. **recalculation of objects distance matrix**
   There are several strategies how to recalculate distance between new cluster, and other clusters:
   – Recalculation of all possible distances among new cluster and other clusters.
   – Already known distances between clusters, forming new cluster, can be exploited. Let c_3 be a cluster that is union of clusters c_1 a c_2. The distance of new cluster c_3 in respect of other clusters c_i can be determined as:

   $$d(c_3, c_i) = min(d(c_1, c_i), c(c_2, c_i))$$

   This recalculation strategy is faster than previous one, because there is no need of calculation of all distances for new cluster again.
5. **if there are more than one cluster, go to step 3**

4 Query expansion

Query expansion algorithms at first evaluate given query on collection of documents, and then select from relevant documents appropriate terms[1]. The original query is expanded with such selected terms. The expanded query is used to retrieve new set of relevant documents. The method is called feedback.

The feedback method has one important drawback. User query must be performed at first. And after searching, query is expanded with terms selected from retrieved documents. In our approach the relevant documents are replaced with the documents from cluster that is the most similar to the user query.

Two algorithms for query expansion were proposed (for details see [5]).

4.1 Description UP-DOWN-1 method

1. Algorithm begins at root of cluster hierarchy (cluster tree).
2. Similarity coefficient between the user query and the current cluster is calculated. If the similarity is greater than given threshold value, stop the algorithm and return current cluster.
3. If current cluster is a leaf in cluster hierarchy (i.e. the cluster contains only one object - document), stop the algorithm and return current cluster.
4. The similarity coefficients are calculated for both clusters that form current cluster.
5. The number of documents is determined in both clusters that form current cluster.

[1] What is appropriate term is another question. Good selection algorithm should prefer terms, that are specific for relevant documents to general terms in collection of documents. The selection algorithms therefore evaluate all terms in documents, considered in query expansion, and then they select terms with the highest value. For evaluation are often used Rocchio's weights, Robertson Selection Value, or Kullback-Leibler distance [4].

6. If the number is less than documents' number threshold, stops the algorithm, and return cluster with greater similarity coefficient.
7. In other case current cluster become cluster with greater similarity coefficient, and goes to step 2.

4.2 Description UP-DOWN-2 method

1. Algorithm begins at root of cluster hierarchy (cluster tree).
2. Similarity coefficient between the user query and the current cluster is calculated. If the similarity is greater than given threshold value, stop the algorithm and return current cluster.
3. If current cluster is a leaf in cluster hierarchy (i.e. the cluster contains only one object - document), stop the algorithm and return current cluster.
4. The number of documents is determined in both clusters that form current cluster.
5. *The algorithm goes to step 2 for all clusters with nonzero similarity coefficient, and with the number of documents greater than the threshold value.*

4.3 Experimental results

The UP-DOWN-1 method was tested at first. Original vector query, and vector query expanded with UP-DOWN-1 method was tested. Average improvement (rather impairment) of number of relevant documents can be seen at Fig 1.

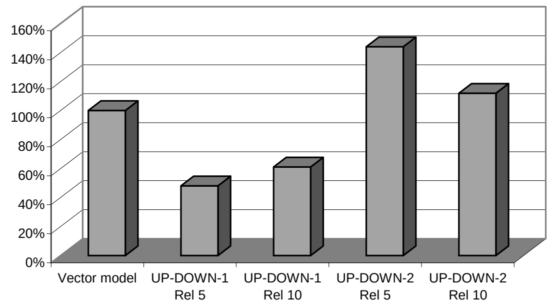


Fig. 1. Comparison of UP-DOWN-1 and UP-DOWN-2 methods

Graph 1 clearly shows that in case of using UP-DOWN-1 method, there is only 48% of relevant documents among the first five retrieved documents, with respect to the original vector query. Situation become better in case of the first ten documents, but only half of retrieved documents are relevant. The UP-DOWN-1 method does not ensure finding of the most similar cluster to the query, but it finds only one of similar clusters. After extensive exploration we found out that the similarity coefficient partially depends on size of cluster. In other words document with less similarity to the query in one document cluster can have greater similarity coefficient than more similar documents in bigger cluster. In that case the query is expanded according to less similar cluster.

The UP-DOWN-2 method was proposed to eliminate this disadvantage. The method was tested on the same documents, and queries as UP-DOWN-1 method. The results can be seen at Fig 1. There can be seen that UP-DOWN-2 method gives much better results than UP-DOWN-1 method. There are three times more relevant documents in the first five documents with respect to the query expanded with the UP-DOWN-1 method. Moreover this method can find more relevant documents than original vector query.

5 Monitoring evolution of topic

Our research concern with he topics undergo an evolution. Let's assume document from collection of documents, that describes some topic. It is clear, that there will be some other documents in the collection that describes the same topic, but use different words to characterize the topic. The difference can be caused by many reasons. Among reasons belong evolution of topic in time. The first document about the topic use some set of words, that can change after some time period due to for example exploration of new circumstances, new fact, new political situation etc. Our experimental method should search an evolution for a given document and sort the result of the query.

Example 1. We want to search documents about an operation system. We have a document about the very latest operation system. Name of the operation system is changed during evolution. We want to search documents about all versions operation system.

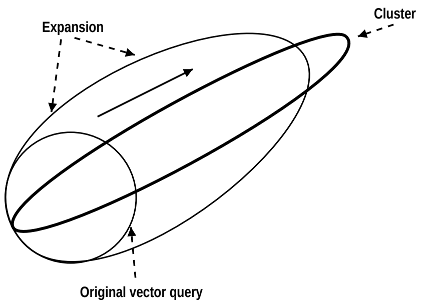


Fig. 2. Query extension

In the first way, vector query was expanded by terms from as close as possible cluster (see Figure 2). Experiments show, that our assumption was not correct.

Expanded vector query was not moved on clusters but only growing up (see Figure 3). This query expansion decreased coefficient of relevance, because non relevant documents are contained in result of expanded vector query.

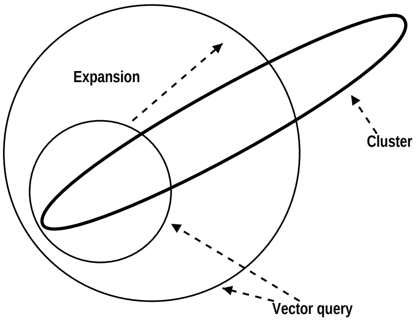


Fig. 3. Increase neighborhood

These experiments show that query expansion do not satisfy expectation. This results lead to drop this method whereas it leads to other method. The method finds cluster with similar documents and evolution of topic is examined in this cluster only. In the first instance we verify whether clusters include similar documents or not. And consequently when we obtain better result than vector query or not.

5.1 Experimental results

Collection of documents for testing purposes contain 1065 randomly selected documents from Parliament library from 1996 and 1998. The test consists of three steps:

1. Executing of vector query. Document 96-T0419 represents the query. Results of the query is given in table 1. The evolution of topic starts from document 96-T0419 i.e. query documents and ends in document 98-T0506 (see Figure 4).
2. Summarization[2] of documents in evolution of topic (see table 2).
3. Selection of the most important terms in documents from evolution (see table 3).

6 Conclusion

There are plenty of large text collections in the world. In connection with expansion of Internet these collections get bigger and bigger. Amount of processed data

[2] The summarization was done in MS Word. It is intended for checking the results.

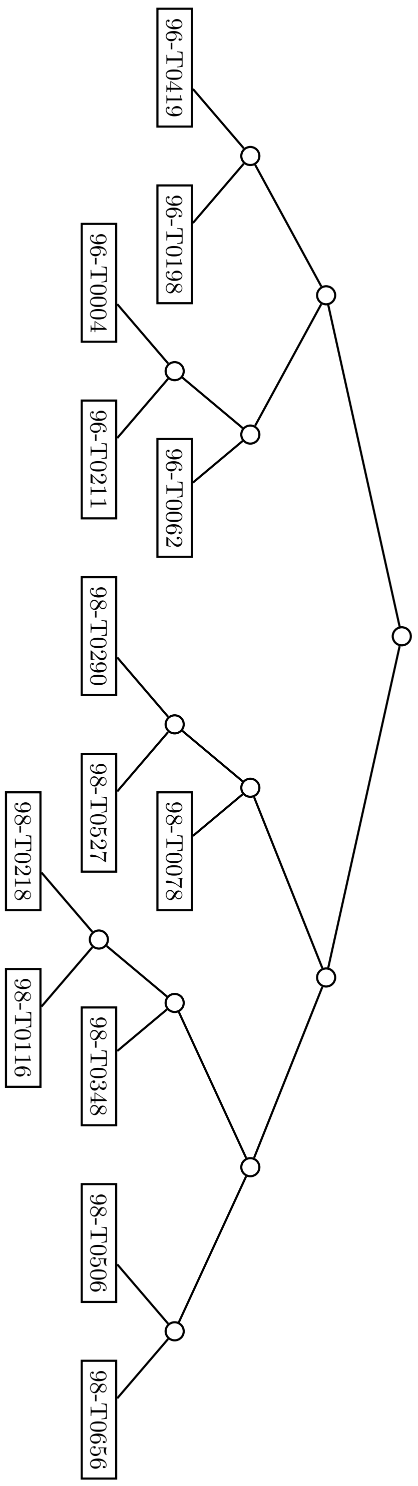


Fig. 4. Cluster tree corresponding to evolution of topic. Query document is the leftmost, and the last document in evolution is the rightmost.

Document	Similarity	Contained in topic evolution?
96-T0419	1.0000	yes
96-T0198	0.7550	yes
96-T0179	0.2385	
96-T0182	0.2766	
96-T0226	0.1514	
98-T0656	0.1290	yes
98-T0506	0.1098	yes

Table 1. Vector query results

96-T0419	na období 1998 - 2000PROTIDROGOVÉ POLITIKY VLÁDY NA OBDOBÍ 1998 - 2000Oblast snižování nabídky drog 93.3Oblast koordinace protidrogové politiky 213.5PROGRAM PROTIDROGOVÉ POLITIKY NA OBDOBÍ 1998 - 2000 305.1Oblast snižování nabídky drog 315.3- Počet předčasných úmrtí v důsledku užívání drog se na začátku 90. 3.2 OBLAST SNIŽOVÁNÍ NABÍDKY DROG
96-T0198	1. Rozsah užívání drog v ČR 2. Oblast snižování nabídky drog 1. Rozsah užívání drog v ČR Přibližně 10% dlouhodobých uživatelů drog je bez stálého bydliště. ČR je významnou tranzitní zemí kokainu. Opatření proti šíření a zneužívání drog v ČRNa tzv. 2. Oblast snižování nabídky drog drog
96-T0004	Zpráva o bezpečnostní situaci na území ČRMateriál se předkládá na základě usnesení vlády ČR č. 280Zpráva o bezpečnostní situaci na území ČR7. Bezpečnostní rizika na rok 1996 - shrnutí Zpráva o bezpečnostní situaci na území ČR/viz Příloha č. 1//viz Příloha č. 2
96-T0211	Zpráva o bezpečnostní situaci na území ČR v roce1996Materiál se předkládá na základě usnesení vlády ČR č. 308 2.5 Oběti trestné činnosti 5.2 Prevence kriminality Na objasněné trestné činnosti páchané na železnici se podíleli 40,23% (-13,08%).Na území hl. Počet trestných činů policistů vzrostl na 37,4 (+16,5%, +53 tr. č.).
96-T0062	května 1996 metodiku programů sociální prevence a prevence kriminality na místní úrovni, - přehled systému sociální prevence a prevence kriminality na ústředních orgánech státní správy.3. Plnění programu sociální prevence a prevence kriminality v předcházejícím obdobíPříloha č. 1 Schéma prevence kriminality f) projekty prevence kriminality na místní úrovni

Table 2. Summarization of documents in evolution of topic

96-T0419	drog drogách drogami drogové drogy oblast politiky prevence protidrogové snižování
96-T0198	drog drogách drogami drogové heroin pervitin prevence protidrogové především resocializace
96-T0004	bezpečnostní cizinců činů kriminalita kriminality migrace počet rizika trestné trestných
96-T0211	bezpečnostní činů kriminalita kriminality objasněnosti počtu policie trestné trestných zjištěných
96-T0062	kriminality prevence prevenci preventivních republikového sociální úrovni vnitra výboru východiska
98-T0290	malého malých podnikání podniků podporu podpory podpořeno projektů středního středních
98-T0527	čmzrb podnikatelského podnikatelský podporu podpory poskytnutí program programu projektu úvěru
98-T0078	doporučení malých měli míst podnicích podniků podniky pracovních pracovníků středních
98-T0218	bilance czechtrade deficitu dovozu egap exportu proexportní proexportních růstu vývozu
96-T0116	bilance deficitu dovozu firem obchodu především růst růstu vývozu zahraničního
96-T0348	aktivity exportní exportu obchodu podporu politiky proexportní vláda vývozu zahraničního
98-T0506	acquis evropské harmonizace legislativy oblast phare politiky programu přípravy vstupu
98-T0656	evropské nato politika politiky programovým průmyslu schválila systému vláda vládě

Table 3. The most important terms in documents contained in evolution of topic

reach in present time dimensions that statistical properties of texts in collection become evident. This fact leads to new approaches, which involve methods from statistics, linear algebra, neural networks, and other (see [1, 9]).

We can characterize a distribution of topics in a document with using clusters. These clusters are possible to use for the vector model and analysis how the topics undergo an evolution.

We developed the UP-DOWN-2 method which increase amount of founded relevant document.

In the future work we want to use large collection as a WebTrec for check whether this method. Next step we want to use Latent Semantic Indexing for computing document similarity [8].

References

1. Berry, M. W.; Browne, M.: Understanding Search Engines: Mathematical Modeling and Text Retrieval. SIAM Book Series: Software, Environments, and Tools, 1999.
2. Berry, M. W. (Ed.): Survey of Text Mining: Clustering Classification, and Retrieval. Springer Verlag 2003.
3. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. Addison Wesley, New York, 1999.
4. Carpineto C., de Mori R., Romano G., Bigi B.: An information-theoretic approach to automatic query expansion., in ACM Trans. Inf. Syst. 19(1), pp. 1-27, 2001
5. Dvorský J., Martinovič J., Pokorný J., Snášel V.: A Search topics in Collection of Documents. (in Czech), Znalosti 2004, in print.
6. Downs G.: Clustering in chemistry (an overview w.r.t. chemoinformatics), MathFIT Workshop, Belfast, 27th April 2001 - Barnard Chemical Information Ltd.
7. Jain A., Dubes R.: Algorithms for Clustering Data. Prentice-Hall, Englewood Cliffs, NJ, 1988.
8. Praks, P., Snášel, V., Dvorský, J.: Latent Semantic Indexing for Image Retrieval Systems, SIAM LA, International Linear Algebra Society (ILAS), 2003.
9. Shumsky S., Yarovoy A.: Assocative searching of textual information, Neuroinformatics 1-99. MIFI, Moscow 1999.

Using Blind Search and Formal Concepts for Binary Factor Analysis

Aleš Keprt

Dept. of Computer Science, FEI, VŠB Technical University Ostrava, Czech Republic
ales.keprt@vsb.cz

Abstract. Binary Factor Analysis (BFA, also known as Boolean Factor Analysis) may help with understanding collections of binary data. Since we can take collections of text documents as binary data too, the BFA can be used to analyse such collections. Unfortunately, exact solving of BFA is not easy. This article shows two BFA methods based on exact computing, boolean algebra and the theory of formal concepts.

Keywords: Binary factor analysis, boolean algebra, formal concepts

1 Binary factor analysis

1.1 Problem definition

To describe the problem of Binary Factor Analysis (BFA) we can paraphrase BMDP's documentation (Bio-Medical Data Processing, see [1]).

BFA is a factor analysis of dichotomous (binary) data. This kind of analysis differs from the classical factor analysis (see [16]) of binary valued data, even though the goal and the model are symbolically similar. In other words, both classical and binary analysis use symbolically the same notation, but their senses are different.

The goal of BFA is to express p variables $X = (x_1, x_2, \ldots, x_p)$ by m factors $(F = f_1, f_2, \ldots, f_m)$, where $m \ll p$ (m is considerably smaller than p). The model can be written as

$$X = F \odot A$$

where $\odot$ is matrix multiplication. For n cases, data matrix X, factor scores F, and factor loadings A can be written as

$$\begin{bmatrix} x_{1,1} \ldots x_{1,p} \\ \vdots \ \ddots \ \vdots \\ x_{n,1} \ldots x_{n,p} \end{bmatrix} = \begin{bmatrix} f_{1,1} \cdots f_{1,m} \\ \vdots \ \ddots \ \vdots \\ f_{n,1} \cdots f_{n,m} \end{bmatrix} \odot \begin{bmatrix} a_{1,1} \ldots a_{1,p} \\ \vdots \ \ddots \ \vdots \\ a_{m,1} \ldots a_{m,p} \end{bmatrix}$$

where elements of all matrices are valued 0 or 1 (i.e. binary).

1.2 Difference to classical factor analysis

Binary factor analysis uses boolean algebra, so matrices of factor scores and loadings are both binary. See the following example: The result is 2 in classical algebra

$$[1\ 1\ 0\ 1] \cdot \begin{bmatrix}1\\1\\0\\0\end{bmatrix} = 1 \cdot 1 + 1 \cdot 1 + 0 \cdot 0 + 1 \cdot 0 = 2$$

but it's 1 when using boolean algebra.

$$[1\ 1\ 0\ 1] \cdot \begin{bmatrix}1\\1\\0\\0\end{bmatrix} = 1 \cdot 1 \oplus 1 \cdot 1 \oplus 0 \cdot 0 \oplus 1 \cdot 0 = 1$$

Sign $\oplus$ marks disjunction (logical sum), and sign $\cdot$ mars conjunction (logical conjunction). Note that since we focus to binary values, the logical conjunction is actually identical to the classic product.

In classical factor analysis, the score for each case, for a particular factor, is a linear combination of all variables: variables with large loadings all contribute to the score. In boolean factor analysis, a case has a score of one if it has a positive response for any of the variables dominant in the factor (i.e. those not having zero loadings) and zero otherwise.

1.3 Success and discrepancy

It is obvious, that not every X can be expressed as $F \odot A$. The success of BFA is measured by comparing the observed binary responses (X) with those estimated by multiplying the loadings and the scores ($\hat{X} = F \odot A$). We count both positive and negative discrepancies. Positive discrepancy is when the observed value (in X) is one and the analysis (in $\hat{X}$) estimates it to be zero, and reversely negative discrepancy is when the observed value is zero and the analysis estimates it to be one. Total count of discrepancies d is a suitable measure of difference between observed values $x_{i,j}$ and calculated values $\hat{x}_{i,j}$.

$$d = \sum_{i=1}^{n} \sum_{j=1}^{p} |\hat{x}_{i,j} - x_{i,j}|$$

1.4 Terminology notes

Let's summarize the terminology we use. Data to be analyzed are in matrix X. Its columns x_j represent **variables**, whereas its rows x_i represent **cases**. The factor analysis comes out from the generic thesis saying that variables, we can observe, are just the effect of the factors, which are the real origin. (You can find more details in [16].) So we focus on factors. We also try to keep number of factors as low as possible, so we can say ”reducing variables to factors”.

The result is the pair of matrices. Matrix of **factor scores** F expresses the input data by factors instead of variables. Matrix of **factor loadings** A defines the relation between variables and factors, i.e. each row a_i defines one particular factor.

1.5 An example

As a basic example (see [1]) we consider a serological problem[1], where p tests are performed on the blood of each of n subjects (by adding p reagents). The outcome is described as positive (a value of one is assigned for the test in data matrix), or negative (zero is assigned). In medical terms, the scores can be interpreted as antigens (for each subject), and the loading as antibodies (for each test reagent). See [14] for more on these terms.

1.6 Application to text documents

BFA can be also used to analyse a collection of text documents. In that case the data matrix X is built up of a collection of text documents D represented as p-dimensional binary vectors d_i, $i \in 1, 2, \ldots, n$. Columns of X represent particular words. Particular cell $x_{i,j}$ equals to *one* when document i contains word j, and *zero* otherwise. In other words, data matrix X is built in a very intuitive way.

It should be noted that some kind of smart (i.e. semantic) preprocessing could be made in order to let the analysis make more sense. For example we usually want to take *world* and *worlds* as the same word. Although the binary factor analysis has no problems with finding this kind similarities itself, it is computationally very expensive, so any kind of preprocessing which can decrease the size of input data matrix X is very useful. We can also use WordNet, or thesaurus to combine synonyms. For additional details see [5].

2 The goal of exact binary factor analysis

In classic factor analysis, we don’t even try to find 100% perfect solution, because it’s simply impossible. Fortunately, there are many techniques that give a good suboptimal solution (see [16]). Unfortunately, these classic factor analysis techniques are not directly applicable to our special binary conditions. While classic techniques are based on the system of correlations and approximations,

[1] Serologic test is a blood test to detect the presence of antibodies against microorganism. See serology entry in [14].

these terms can be hardly used in binary world. Although it is possible to apply classic (i.e. non-boolean non-binary) factor analysis to binary data, if we really focus to BFA with restriction to boolean arithmetic, we must advance another way.

You can find the basic BFA solver in BMDP – Bio-Medical Data Processing software package (see [1]). Unfortunately, BMDP became a commercial product, so the source code of this software package isn't available to the public, and even the BFA solver itself isn't available anymore. Yet worse, there are suspicions saying that BMDP's utility is useless, as it actually just guesses the F and A matrices, and then only explores the similar matrices, so it only finds local minimum of the vector error function.

One interesting suboptimal BFA method comes from Húsek, Frolov et al. (see [15], [7], [6], [2], [8]). It is based on a Hopfield-like neural network, so it finds a suboptimal solution. The main advantage of this method is that it can analyse very large data sets, which can't be simply processed by exact BFA methods.

Although the mentioned neural network based solver is promising, we actually didn't have any one really exact method, which could be used to proof the other (suboptimal) BFA solvers. So we started to work on it.

3 Blind search based solver

The very basic algorithm blindly searches among all possible combinations of F and A. This is obviously 100% exact, but also extremely computational expensive, which makes this kind of solver in its basic implementation simply unusable.

To be more exact, we can express the limits of blind search solver in units of n, p and m. Since we need to express matrix X as the product of matrices $F \odot A$, which are $n \times m$ and $m \times p$ in size, we need to try on all combinations of $m \cdot (n + p)$ bits. And this is very limiting, even when trying to find only 3 factors from 10×10 data set ($m = 3$, $n = 10$, $p = 10$), we end up with computational complexity of $2^{m \cdot (n+p)} = 2^{60}$, which is quite behind the scope of current computers.

4 Revised blind search based solver

In order to make the blind search based solver more usable, we did several changes to it.

4.1 Preprocessing

We must start with optimizing data matrix. The optimization consist of these steps:

Empty rows or columns All empty rows and empty columns are removed, because they has no effect on the analysis. Similarly, the rows and columns full of one's can be removed too. Although removing rows and columns full of one's can lead to higher discrepancy (see sec. 1.3), it doesn't actually have any negative impact on the analysis.

Moreover, we can ignore both cases (rows) and variables (columns) with too low or too high number of one's, because they are usually not very important for BFA. Doing this kind of optimization can significantly reduce the size of data matrix (directly or indirectly, see below), but we must be very careful, because it can lead to wrong results. Removing too many rows and/or columns may completely degrade the benefit of exact BFA, because it leads to exact computing with inexact data. In regard to this danger, we actually implemented only support for removing columns with too low number of one's.

Duplicate rows and columns With duplicate rows (and columns resp.) are the ones which are the same to each other. Although this situation can hardly appear in classic factor analysis (meaning that two measured cases are 100% identical), it can happen in binary world much easier, and it really does. As for duplicate rows, the main reason of their existence is usually in the preprocessing. If we do some kind of semantic preprocessing, or even forcibly remove some columns with low number of one's, the same (i.e. duplicate) rows occur. We can remove them without negative impact to the analysis, if we remember the repeat-count of each row. We call it *multiplicity.*

Then we can update the discrepancy formulae (see sec. 1.3) to this form:

$$d = \sum_{i=1}^{n} \sum_{j=1}^{p} (m_i^R \cdot m_j^C |\hat{x}_{i,j} - x_{i,j}|)$$

where m_i^R and m_j^C are multiplicity values for row i and column j respectively.

We can also compute the *multiplicity matrix* M:

$$M = \begin{bmatrix} m_{1,1} \dots m_{1,p} \\ \vdots \quad \ddots \quad \vdots \\ m_{n,1} \dots m_{n,p} \end{bmatrix}$$

where $m_{i,j} = m_i^R \cdot m_j^C$. Although this leads to simpler and better readable formulae

$$d = \sum_{i=1}^{n} \sum_{j=1}^{p} (m_{i,j} |\hat{x}_{i,j} - x_{i,j}|)$$

it isn't a good idea, since the implementation is actually inefficient, since it needs a lot of additional memory ($n \cdot p$ numbers compared to $n + p$ ones).

The most important note is, that the merging of duplicate rows and columns lead to a significant reduction in computation time, and still doesn't bring any errors to the computation.

4.2 Bit-coded matrices

Using standard matrices is simple, because it is based on classic two-dimensional arrays and makes the source code well readable. In contrast, we also implemented the whole algorithm using bit-coded matrices and bitwise (truly boolean) operations (like and, or, xor). That resulted in not so nice source code, and also required some tricks, but also saved a lot of computation speed. We actually sped up the code by 20% by using bit-coded matrices and bitwise (boolean) operations.

4.3 The strategy

Although all the optimizations presented above lead to lower computation time, it is still not enough. To save yet more computation time, we need a good strategy.

The main problem is that we need to try too many bits in matrices F and A. Fortunately there exist a way of computing one of these matrices from the other one, thanks to knowing X. Since we are more concerned in A, we check out all bits in that one, and then find the right F. In summary:

1. Build up one particular candidate for matrix A.
2. Find the best F for this particular A.
3. Multiply these matrices and compare the result with X. If the discrepancy is smaller to the so far best one, remember this F, A pair.
4. Back to step 1.

After we go through all possible candidates for A, we're done.

4.4 Computing F from A and X

Symbolically, we can express this problem as follows. We are trying to find F and A, so $X = F \odot A$. Let we know X and A, so we only need to compute F. If we take a parallel from numbers, we can write something like $F = X/A$. Unfortunately, this operation isn't possible with common binary matrices.

If we bit-code matrices X and A on row-by-row basis, so $X = [x_1, \dots, x_n]^T$ and $A = [a_1, \dots a_p]^T$, then

$$x_i = \sum_{k=1}^{m} f_{i,k} \cdot a_j$$

From this formulae we can compute F on row-by-row basis, which significantly speeds up whole algorithm. The basic idea still relies on checking out all bit combinations for each row of F, which is $2^m \cdot m$ in total, but we can possibly find a better algorithms in future. In our implementation we compute discrepancy together with finding F, so we can abort the search whenever the partial discrepancy is higher than the so far best solution. This way we get some speedup which could be made yet higher by pre-sorting rows of A by the discrepancies caused by particular rows, etc. Exploration of these areas isn't very important, because the possible speedup is quite scanty.

Note that in this place we can also focus to positive or negative discrepancy exclusively. It can be done using boolean algebra without any significant speed penalties.

5 Parallel implementation

The bind-search algorithm (including the optimized version presented above) can exploit the power of parallel computers (see [9]). We used PVM interface (Parallel Virtual Machine, see [4]) which is based on sending messages. The BFA blind search algorithm is very suitable for this kind of parallelism, because we just need to find a smart way of splitting the space of possible solutions to be checked out to a set of sub-spaces, and distribute them among available processor in our parallel virtual machine.

We tested this method using 2 to 11 PC desktop computers on a LAN (local area network). We managed to gain the absolute efficiency around 92%[2], which is very high compared to usual parallel programs. (The number 92% says that it takes 92% of time to run 11 consecutive runs on the same data, compared to a single run of the parallel version on the network of 11 computers).

6 Concept lattices

Another method of solving BFA problem is based on concept lattices. This section gives minimum necessary introduction to concept lattices, and especially *concepts*, which are the key part of the algorithm.

Definition 1 (Formal context, objects, attributes).
Triple (X, Y, R)*, where* X *and* Y *are sets, and* R *is a binary relation* $R \subseteq X \times Y$*, is called* **formal context**. *Elements of* X *are called* **objects**, *and elements of* Y *are called* **attributes**. *We say "object* A *has attribute* B*", just when* $A \subseteq X$*,* $B \subseteq Y$ *and* $(A, B) \in R$*.* □

Definition 2 (Derivation operators).
For subsets $A \subseteq X$ *and* $B \subseteq Y$*, we define*

$$A^{\uparrow} = \{b \in B \mid \forall a \in A : (a, b) \in R\}$$
$$B^{\downarrow} = \{a \in A \mid \forall b \in B : (a, b) \in R\}$$

□

[2] It was measured in Linux, while Windows 2000 performed a bit worse and its performance surprisingly fluctuated.

In other words, $A^{\uparrow}$ is the set of attributes common to all objects of A, and similarly $B^{\downarrow}$ is the set of all objects, which have all attributes of B.

Note: We just defined two operators $^{\uparrow}$ and $^{\downarrow}$:

$$^{\uparrow} : P(X) \to P(Y)$$
$$^{\downarrow} : P(Y) \to P(X)$$

where $P(X)$ and $P(Y)$ are sets of all subsets of X and Y respectively.

Definition 3 (Formal concept).
Let (X, Y, R) be a formal context. Then pair (A, B), where $A \subseteq X$, $B \subseteq Y$, $A^{\uparrow} = B$ and $B^{\downarrow} = A$, is called **formal concept** *of (X, Y, R).*

Set A is called **extent** *of (A, B), and set B is called* **intent** *of (A, B).* □

Definition 4 (Concept ordering).
Let (A_1, B_1) and (A_2, B_2) be formal concepts. Then (A_1, B_1) is called subconcept of (A_2, B_2), just when $A_1 \subseteq A_2$ (which is equivalent to $B_1 \supseteq B_1$). We write $(A_1, B_1) \leq (A_2, B_2)$. Reversely we say, that (A_2, B_2) is superconcept of (A_1, B_1). □

In this article we just need to know the basics of concepts and their meaning. For more detailed, descriptive, and well understandable introduction to Formal Concept Analysis and Concept Lattices, see [3], [11] or [13].

7 BFA using formal concepts

If we want to speed up the simple blind-search algorithm, we can try to find some factor *candidates*, instead of checking out all possible bit-combinations. The technique which can help us significantly is Formal Concept Analysis (FCA, see [11]). FCA is based on concept lattices, but we actually work with formal concepts only, so the theory we need is quite simple.

7.1 The strategy

We can still use some good parts of the blind-search program (matrix optimizations, optimized bitwise operations using boolean algebra, etc.), but instead of checking out all possible bit combinations, we work with concepts as the factor candidates. In addition, we can adopt some strategy optimizations (as discussed above) to concepts, so the final algorithm is quite fast; its strength actually relies on the concept-building algorithm we use.

So the BFA algorithm is then as follows:

1. Compute all concepts of X. (We use a standalone program to do this.)
2. Import the list of concepts, and *optimize* it, so it correspond to our optimized data matrix X. (This is simple. We just merge objects and attributes the same way, as we merged duplicate rows and columns of X respectively.)
3. Remove all concepts with too many one's. (The number of one's per factor is one of our starting constraints.)
4. Use the remaining concepts as the factor candidates, and find the best m-element subset (according to discrepancy formulae).

This way we can find the BFA solution quite fast, compared to the blind search algorithm. Although the algorithm described here looks quite simple[3], there is a couple of things, we must be aware of.

7.2 More details

The most important FCA consequence is that 100% correct BFA solution can always be found among all subsets of concepts. This is very important, because it is the main guarantee of the correctness of the concept based BFA solver.

Other important feature of FCA based concepts is that they never directly generate any negative discrepancy. It is a direct consequence of FCA qualities, and affects the semantic sense of the result. As we discussed above (and see also [1]), negative discrepancy is a case when $F \odot A$ gives 1 when it should be 0. From semantic point of view, this (the negative discrepancy) is commonly unwanted phenomenon. In consequence, the fact that there's no negative discrepancy in the concepts, may have negative impact on the result, but the reality is usually right opposite. (Compare this to the quick sort phenomenon.)

The absence of negative discrepancies coming from concepts applies to A matrix only. It doesn't apply to F matrix, we still can use any suitable values for it. In consequence, we always start with concepts not generating negative discrepancy, which are semantically better, and end up with best suitable factor scores F, which give the lowest discrepancy. So it seems to be quite good feature.

7.3 Implementation issues

It's clear that the data matrix X is usually quite large, and makes the finding of the formal concepts the main issue. Currently we use the standalone CL (concept lattice) builder. It is optimized for finding concept lattices, but that's not right what we need. In the future, we should consider adopting some kind of CL building algorithm directly into BFA solver. This will save a lot of time

[3] *Everything's simple, when you know it.*

when working with large data sets, because we don't need to know the concept hierarchy.

We don't even need to know all the formal concepts, because the starting constraints limit the maximum number of one's in a factor, which is directly applicable to CL building.

8 Comparing the results and the computation times

The two algorithms presented in this article were tested on the test data suite taken from the neural algorithm mentioned above (see [15], [7], [6], [2], [8]). We focused to test data sets `p2` and `p3`, which are both 100×100 values in size, and differs in the ones' density. All three algorithms gave the same results, so they all appear to be correct (from this point of view).

Table 1. Computation times

data set	factors	one's	time (m:s)	discrepancy	notes
p3.txt	5	2–4	61:36	0	375 combinations
p3.txt	5	3	0:12	0	120 combinations
p3.txt	5	1–10	0:00	0	8/10 concepts
p2.txt	2	6	11:44	743	54264 combinations
p2.txt	5	1–10	0:07	0	80/111 concepts
p2.txt	5	6–8	0:00	0	30/111 concepts

The results are shown in table 8. Data set `p3` is rather simple, its factor loadings (particular rows of A) all have 3 one's. The first row in the table shows that it takes over 61 minutes to find these factors, when we search all combinations with 2, 3 or 4 one's per factor. If we knew that there are just 3 one's per factor, we can specify it as a constraint, and we get the result in just 12 seconds (see table 1, row 2). Indeed we usually don't know it in real situations.

Third row shows that when using formal concepts, we can find all factors in just 0 seconds, even when we search all possible combinations with 1 to 10 one's per factor. You can see the concept lattice in picture 1, with factors expressively circled.

Data set `p2` is much more complex, because it is created from factors containing 6 one's each. In this case the blind-search algorithm was able to find just 2 factors. It took almost 12 minutes, and discrepancy was 743. In addition, the two found factors are wrong, which is not a surprise according to the fact that there are actually 5 factors, and they can't be searched individually. It was not possible to find more factors using blind-search algorithm. Estimated times for computing 3 to 5 factors with the same constraints (limiting number of one's per factor to 6) are shown in table 8. It shows that it would take up to 3.5×10^9 years to find all factors. Unfortunately, we can't afford to wait so long...

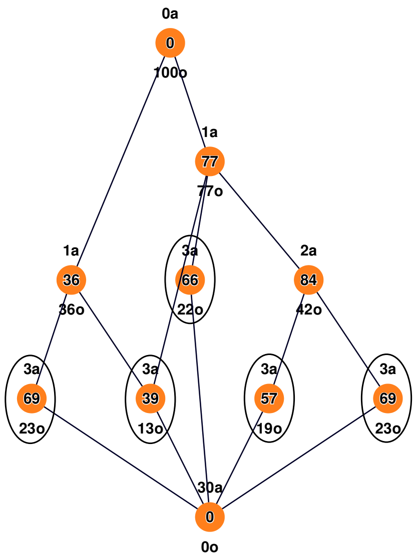


Fig. 1. Concept lattice of p3 data set.

Table 2. Estimated computation times

data set	factors	one's	estimated time
p2.txt	3	6	440 days
p2.txt	4	6	65700 years
p2.txt	5	6	3.5×10^9 years

As you can see at the bottom of table 1, we can find all 5 factors of p2 easily in just 7 seconds, searching among candidates containing 1 to 10 one's. The time can be reduced to 0 seconds once again, if we reduce searching to the range of 6 to 8 one's per factor. You can see the concept lattice in picture 1, with factors marked as well. As you can see, the factors are non-overlapping, i.e. they are not connected to each other. Note that this is not a generic nature. Generally, factors can arbitrarily overlap.

9 Conclusion

This article presented two possible algorithms for exact solving of Binary Factor Analysis. The work on them originally started as a simple blind search algorithm in order to check out the results of P8M of BMDP (see [1]), and the promising neural network solver (see [15], [7], [6], [2], [8]). As the work progressed, the theory of Concept Lattices and Concept Analysis was partially adopted into it, and it was with an inexpectably good results. For sure, the future work will more

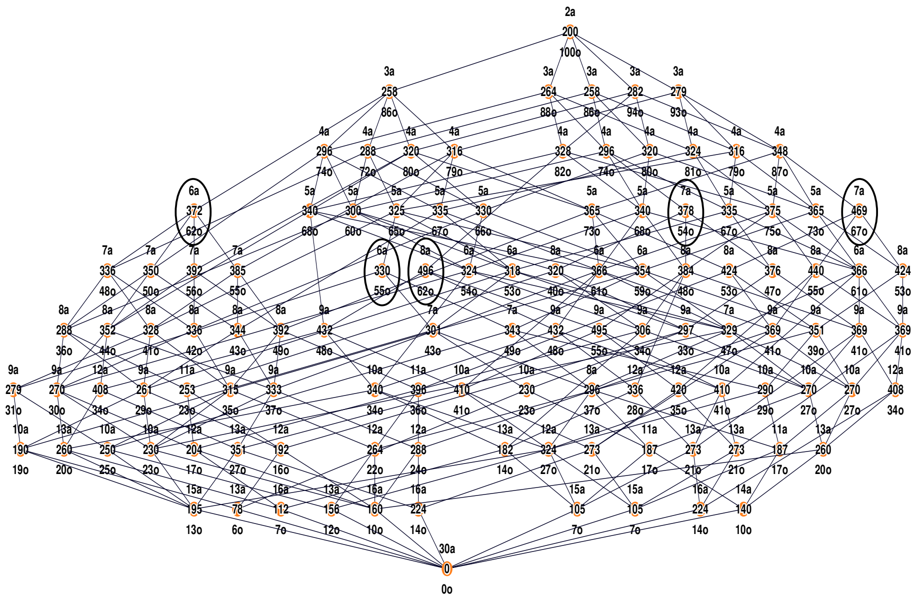


Fig. 2. Concept lattice of p2 data set.

focus on the possibilities of exploiting formal concepts and concept lattices for BFA.

References

1. *BMDP (Bio-Medical Data Processing).* A statistical software package. SPSS. `http://www.spss.com/`
2. A.A.Frolov, A.M.Sirota, D.Húsek, I.P.Muraviev, P.A.Polyakov: *Binary factorization in Hopfield-like neural networks: Single-step approximation and computer simulations.* 2003.
3. Bernhard Ganter, Rudolf Wille: *Formal Concept Analysis: Mathematical Foundations.* Springer–Verlag, Berlin–Heidelberg–New York, 1999.
4. Al Geist et al.: *PVM: Parallel Virtual Machine, A User's Guide and Tutorial for Networked Parallel Computing.* MIT Press, Cambridge, Massachusetts, USA, 1994.
5. Andreas Hotho, Gerd Stumme: *Conceptual Clustering of Text Clusters.* In Proceedings of FGML Workshop, pp. 37–45. Special Interest Group of German Informatics Society, 2002.
6. D.Húsek, A.A.Frolov, I.Muraviev, H.Řezanková, V.Snášel, P.Polyakov: *Binary Factorization by Neural Autoassociator.* AIA Artifical Intelligence and Applications - IASTED International Conference, Benalmádena, Málaga, Spain, 2003.
7. D.Húsek, A.A.Frolov, H.Řezanková, V.Snášel: *Application of Hopfield-like Neural Networks to Nonlinear Factorization.* Proceedings in Computational Statistics Compstat 2002, Humboldt-Universitt, Berlin, Germany, 2002.

8. D.Húsek, A.A.Frolov, H.Řezanková, V.Snášel, A.Keprt: *O jednom neuronovém přístupu k redukci dimenze.* In proceedings of Znalosti 2004, Brno, CZ, 2004. ISBN 80-248-0456-5.
9. Aleš Keprt: *Paralelní řešení nelineární booleovské faktorizace.* VŠB Technical University, Ostrava (unpublished paper), 2003.
10. Aleš Keprt: *Binary Factor Analysis and Image Compression Using Neural Networks.* In proceedings of WOFEX 2003, Ostrava, 2003. ISBN 80-248-0106-X.
11. Christian Lindig: *Introduction to Concept Analysis.* Hardvard University, Cambridge, Massachusetts, USA.
12. Christian Lindig: *Fast Concept Analysis.* Harvard University, Cambridge, Massachusetts, USA.
    `http://www.st.cs.uni-sb.de/~lindig/papers/fast-ca/iccs-lindig.pdf`
13. Christoph Schwarzweller: *Introduction to Concept Lattices.* Journal Of Formalized Mathematics, volume 10, 1998. Inst. of Computer Science, University of Bialystok.
14. *Medical encyclopedia Medline Plus.* A service of the U.S. National Library of Medicine and the National Institutes of Health.
    `http://www.nlm.nih.gov/medlineplus/`
15. A.M.Sirota, A.A.Frolov, D.Húsek: *Nonlinear Factorization in Sparsely Encoded Hopfield-like Neural Networks.* ESANN European Symposium on Artifical Neural Networks, Bruges, Belgium, 1999.
16. Karl Ueberla: *Faktorenanalyse* (2^{nd} edition). Springer–Verlag, Berlin–Heidelberg–New York, 1971. ISBN 3-540-04368-3, 0-387-04368-3.
    (slovenský překlad: Alfa, Bratislava, 1974)

Finite State Automata and Image Recognition

Marian Mindek

Department of Computer Science, FEI, VŠB - Technical University of Ostrava,
17. listopadu 15, 708 33, Ostrava-Poruba, Czech Republic
marian.mindek@vsb.cz

Abstract. In this paper we introduce finite automata as a tool for specification and compression of gray-scale image. We describe, what are interests points in pictures and idea if they can hang together with resultant finite automata.

Keywords: finite automata, interest points, image recognition

1 Introduction

Karel Culik II and Vladimir Valenta have proposed fractal-coding technique which is based on automata theory. In their paper [1] describe a inference algorithm for generalized finite automata and a lossy compression system for bi-level images based on this algorithm and vector quantization. In another paper [2] describe a similar algorithm for gray-scale image, which use a weighted finite automata (WFA). We're issue these ideas and describe algorithm for gray-scale pictures based on simple solution for bi-level pictures.

2 Finite automata

A digitized image of the finite resolution *m x n* consists of *m x n* pixels each of which takes a Boolean value (1 for black, 0 for white) for bi-level image, or real value (practically digitized to an integer value 0 and 256) for a gray-scale image.

Here we will consider square images of resolution 2^n x 2^n (typically $6 \leq n \leq 11$). In order to facilitate the application of finite automata to image description we will assign each pixel at 2^n x 2^n resolution a word of length *n* over the alphabet $\Sigma=\{0,1,2,3\}$ as its address. A pixel at 2^n x 2^n resolution corresponds to a sub square of size 2^{-n} of the unit square. We choose ε as the address of the whole unit square. Its quadrants are addressed by single digits as shown in Fig. 1 on the left. The four sub square of the square with address *w* are addressed *w0, w1, w2* and *w3,* recursively. Address of all the sub square (pixels) of resolution 4 x 4 are shown in Fig. 1, middle. The sub square (pixel) with address *3203* is shown on the right of Fig. 1.

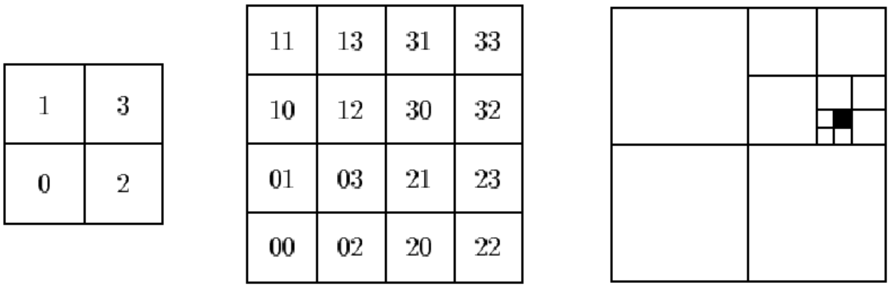


Figure 1. The addresses of the quadrants, of the sub square of resolution 4 x 4, and the sub square specified by the string 3203.

In order to specify a black and white image of resolution 2^m x 2^m, we need to specify a Boolean function $\Sigma^m \rightarrow \{0,1\}$, or alternately we can specify just the set of pixels which are black, i.e. a language $L \subseteq \Sigma^m$. Frequently, it is useful to consider multi-resolution images, that is images which are simultaneously specified for all possible resolution, usually in some compatible way. (We denote Σ^m the set of all words over Σ of the length *m*, by Σ^* the set of all words over Σ)

In our notation a bi-level multi-resolution image is specified by a language $L \subseteq \Sigma^*$, $\Sigma=\{0,1,2,3\}$, i.e. the set of addresses of all the black squares, at any resolution. Now, we are ready to give some examples. We assume that the reader is familiar with the elementary facts about finite automata and regular sets, see e.g. [4].

A finite automaton is displayed by its diagram which is directed graph whose nodes are states, with the initial node indicated by an incoming arrow and the final nodes by double circles. An edge labeled *a* from state *i* to state *j* indicates that input *a* causes the transition from state *i* to state *j*. A word in the input alphabet is accepted by the automaton if it labels a path from the initial state to the final state. The se (language accepted by automaton A) is denoted *L(A)*.

Example. The 2 x 2 chess-board in Fig. 2 looks the same for all resolution 2^m x 2^m, m ≥ 1. For depth *m*, the specification is the finite set $\{1,2\}\Sigma^{m-1}$, the multi-resolution specification is the regular set $\{1,2\}\Sigma^*$. The 8 x 8 chess-board in Fig. 2 as a multi-resolution image is described by the regular se $\Sigma^2\{1,2\}\Sigma^*$ or by automaton *A* of Fig. 3.

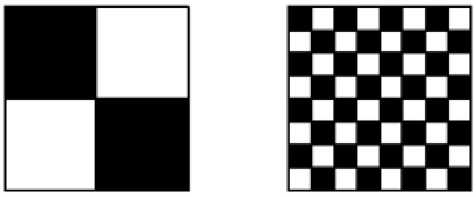


Figure 2. 2 x 2 and 8 x 8 chess-boards.

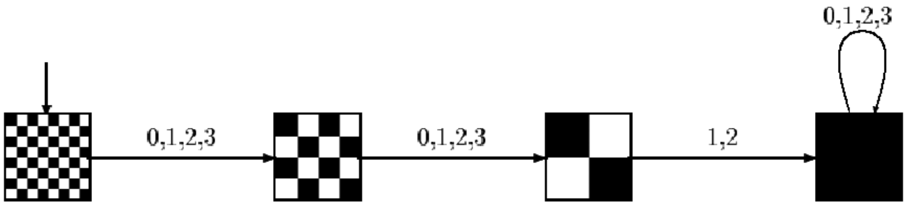


Figure 3. Finite automaton *A* defining the 8 x 8 chess-board.

Notice that here we used the fact that the regular expression $\Sigma^2\{1,2\}\Sigma^*$ is the concatenation of two regular expression Σ^2 and $\{1,2\}\Sigma^*$. It is easy to show that in general if the image is described by the concatenation of two languages $L=L_1L_2$,, then the image L is always obtained by placing copes of the image L_2 into all the squares addressed by the 4 x 4 chess-board $\Sigma\{1,2\}\Sigma^*$ into the squares addressed 0,1,2 and 3, that is as concatenation of Σ and $\Sigma\{1,2\}\Sigma^*$.

Our concatenation decomposition $L=L_1L_2$,, works even when language L_1, is infinite as shown by the following example.

Example. Clearly, $L_1=\{1,2\}^*0$ are addresses of the infinitely many squares illustrated at the left of Fig. 4. If we place the completely black square defined by $L_2=\Sigma^*$ into all these squares we get the image specified by the concatenation $L_1L_2=\{1,2\}^*0\Sigma^*$ which is the triangle shown in the middle of Fig. 4.

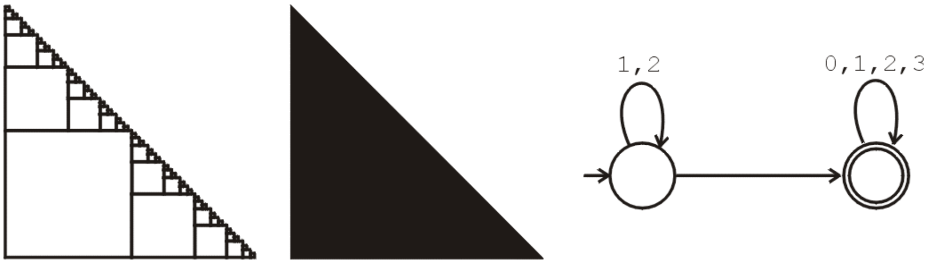


Figure 4. The squares specified by $\{1,2\}^*0$, a triangle defined by $\{1,2\}^*0\Sigma^*$, and the corresponding automaton.

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

Zooming is easily implemented for images represented by regular sets. Let an image be represented by language L. Zooming to sub square with address w, i.e. expanding the image in square w to the whole unit square, is done as follows. We take the left quotient of L with respect to w, that is $L_w=\{x\in\Sigma \mid wx\in L\}$. This is especially easy when L is specified by a deterministic finite automaton (DFA) A. The DFA A_w

accepting L_w is obtained by simply replacing the initial state of *A* by the state reached by input string *w*.

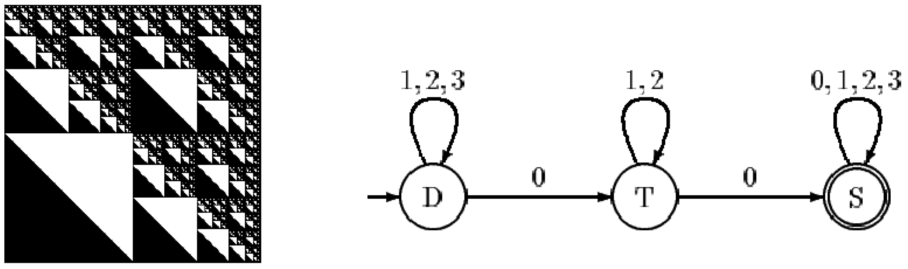


Figure 5. The diminishing triangles defined by {1,2}*0Σ*, and the corresponding automaton.

We have just shown that a necessary condition for black and white multi-resolution image to be represented by a regular set, is that is has only a finite number of different sub images in all the sub squares with addresses from Σ*. We will show that this condition is also sufficient. Therefore, images that can be perfectly (i.e. with infinite precision) described by regular expressions (finite automata) are images of regular or fractal character. Self-similarity is a typical property of fractals. Any image can by approximated by a regular expression (finite automaton), however, an approximation with a smaller error might require a larger automaton.

Now, we will give a theoretical procedure which, given a multi-resolution image, finds a finite automaton perfectly specifying it, if such an automaton exists.

Procedure *Construct Automaton*
For given image *I*, we denote I_w the zoomed part of *I* in the square addressed *w*. The image represented by state number *x* is denoted by u_x.

1. $i=j=0$.
2. Create state 0 and assign $u_0=I$.
3. Assume $u_i=I_w$. Process state *i*, that is for k=0,1,2,3 do:
   If $I_{wk}=u_q$ for some atate *q*, then create an edge labeled *k* from state *i* to state *q*; otherwise assign $j=j+1$, $u_j=I_{wk}$,and create an edge labeled *k* from state *i* to the new state *j*,
4. if $i=j$, that is all states have been processed, stop;
   otherwise $i=i+1$, go to 3.

The procedure *Construct Automaton* terminates if there exists an automaton that perfectly specifies the given image and produces a deterministic automaton with the minimal number of states. Our algorithm for gray-scale image is based on this procedure, but it will use valuated finite automata (as like WFA) introduced in the section 4 and only replacing black and white color to 256 color (or grayness) image and no creating loop.

For the image *diminishing triangles* in Fig. 5, the procedure constructs the automaton shown at the right-hand side of Fig. 5. First the initial state *D* is created an processed. For 0 a new state *T* is created, for 1,2 and 3 a loop to itself. Then state *T* is processed for 0 a new state *S* is created, for 1 and 2 a loop to *T*. There is no edge labeled 3 coming out of *T* since the quadrant 3 for *T* (triangle) is empty. Finally the state *S* (square) is processed by creating loops back to *S* for all 4 inputs.

3 Interest points

Follows section is based on [4]. The system for image capturing produce mostly the images that are represented discretely by matrices of value. Each element in the matrix expresses either the brightness or the intensities of the color components at the corresponding image point (pixel). In order to capture the image precisely, many pixels are usually used, which yields high volumes of data. If an image is to be analyzed, it is often difficult or even impossible to use all this information directly. Many systems work in such a way that they divide the process of analyzing the image into two steps. In the first step, the important features in the images are found in a rather „mechanical“ way. The features are then used for analyzing the image in the second step. (Let us recall the well-known fact that this scheme need not be accepted without exceptions. The desired result of the first step may depend on the image content which, however, is not known at the time when the first step is carried out.)

The first step, in which the important features are found (usually without deeper understanding the content of the image), has grown into a large and important field in digital image processing that includes finding areas, edges, and corners. Great attention was paid to solve the mentioned problems in the past. Despite this effort, the research in the are does not seem to be closed. If more effective and especially more reliable solutions were available, it could improve the overall performance of the whole systems.

This part of work focuses on the problem of detecting the corners (points of interest, feature points, junction points, dominant points). By the term *corner*, we mean the point at which the direction of the noundary of object changes abruptly. The object is a continuos image area with a constant (or nearly constant) brightness or color. Alternatively, we could say that the corner is an intersection point between two or more edge segments. Let us use Figs. 6-10 to illustrate the term more clearly. Fig. 6 depicts a very simple image containing several objects with the corners indicated in the figure. Fig. 7 shows an example of a typical shape of function of brightness in the neighborhood of a corner. A more complicated brightness function is depicted in Fig. 8. Fig. 9 and 10 show an artificial and real image, respectively, with the corners indicated in them. The simplest possible type of corner depicted in Fig. 7 is called the L-cornel. The image may also contain more complicated corners referee to as T, Y, and X-corners. The corner depicted in Fig. 8, for example, is T-corner. Various types of corners are depicted in Fig. 11. We remark that some authors use the term corner only for the points at which two edges intersect. They then use the term j*unction* or *vertex* for more complicated situations. To be concise, we use the term corner in all that cases.

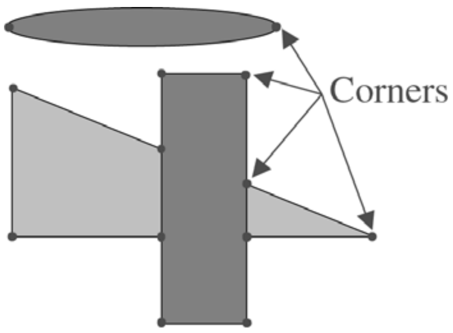


Figure 6. A simple image counting objects (gray areas). The boundaries (solid lines) and corners (small circles) are indicated in the image.

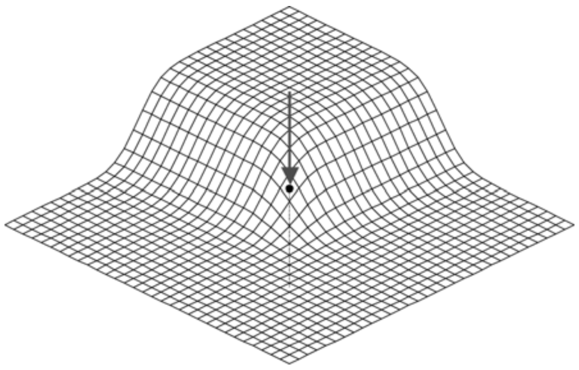


Figure 7. A typical shape of the surface that is defined by the function of brightness in the neighborhood of corner (L-corner). The arrow indicates the theoretical corner point.

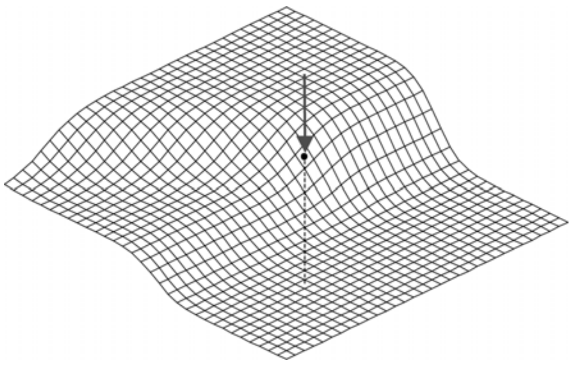


Figure 8. A more complicated surface of brightness (T-corner).

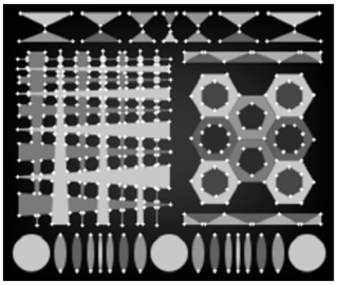


Figure 9. Detected corners (white crosses) in an artificial image.

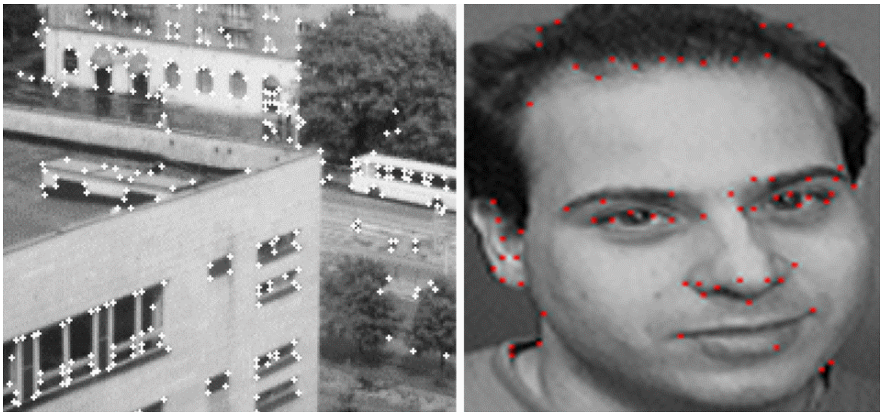


Figure 10. Detected corners in an image obtained from a CCD camera.

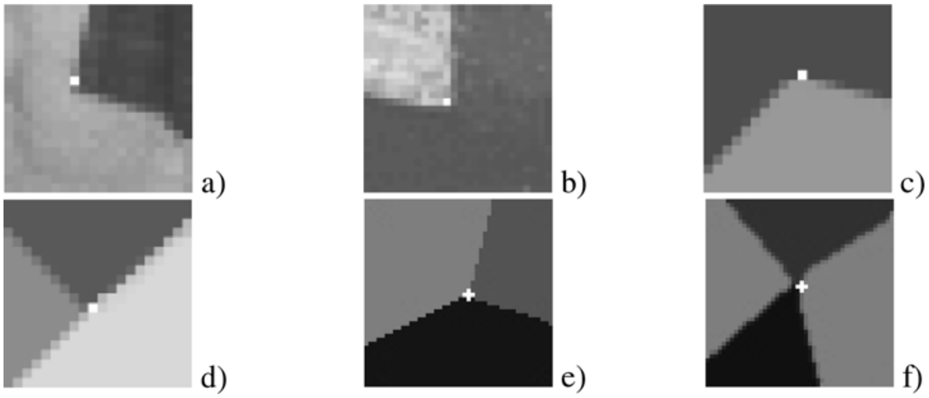


Figure 11. How the corners manifest themselves in real images. L-corners(a,b,c), T-corner (d), Y-corner (e), X-corner (f).

Since the corners convey rich information for many applications in digital image processing and computer vision, the problem of detecting the corners is recognized and well known. Corner detection is often an important step in various image-understanding and scene-reconstructing systems, in which objects are to be detected, tracked, recognized and reconstructed. Some authors claim that the corners play the important role in human perception, which seems probable. It also explain why the use of corner detection may be a logical step in artificial systems too.

If a theoretical analysis is to be carried out, a certain mathematical model of corner is usually required. We will show a widely used model of the L-corner. Let $\psi(\xi)$ be the unit step function defined as follows

$$\psi(\xi) = \left(1 \text{ if } \xi \geq 0, 0 \text{ otherwise} \right) \qquad \textbf{(3.1)}$$

Consider an L-corner that is created as an intersection of two non-collinear straight edges. Let φ_1, $\varphi_2 \in <0,2\pi)$ be the directions perpendicular to the edges oriented to the side with higher brightness. We set $n_i=(\cos\varphi_i, \sin\varphi_i)$, i=1,2. Consider the image containing a single convex corner at a point *C*, The brightness function, denoted by *b(X)*, in such an image can be described by the following equations (the term $n_i(X-C)$ expresses the signed distance of *X* from the *i*-th edge)

$$\begin{gathered} b_0(X) = \psi(n_1 . (X - C))\, \psi(n_2 . (X - C)) , \\ b(X) = G(X)*b_0\,(X) \end{gathered} \qquad \textbf{(3.2)}$$

where * means the convolution, * denotes the dot product, and *G*(*X*) stands for the two dimensional Gaussian filter.

The corner depicted in Fig. 7 was generated by making use of Eq- (3.1) too. For more complex corners (T, Y, X-corners) the corresponding models can also be introduced [4]. These, however, will not be necessary in this work.

Detecting the corners reliably and effectively in real images that are processed in practice is a difficult problem (Fig. 10). Although many detectors usually work well on simple test images, all the existing algorithm have problem in practical applications if more complicated images are to be processed.

For more information about corners detection, and much more see [5].

4 Image recognition

Our algorithm is based on algorithm shown in section 2. From every node graph lead maximum 4 edges, which they are evaluation numbers of represented image part. At every node is storage information of average grayness in sub square represented thereby state.

Procedure *Construct Automaton for Recognition*
For given image I, we denote I_w the zoomed part of I in the square addressed w. The image represented by state number x is denoted by u_x.

1. $i=j=0$.
2. Create state 0 and assign $u_0=I$. (Image represented by empty word) and define average grayness of image I.
3. Assume $u_i=I_w$. Process state i, that is for k=0,1,2,3 do:
   If $I_{wk}=u_q$ (with small error) or if the image I_{wk} can be expressed as a part or expanded part of the image u_q for some state q, then create an edge labeled k from state i to state q;
   otherwise assign $j=j+1$, $u_j=I_{wk}$,and create an edge labeled k from state i to the new state j,
4. if $i=j$, that is all states have been processed, stop;
   otherwise $i=i+1$, go to 3.

The procedure *Construct Automaton for Recognition* terminates if there exists an automaton that perfectly (or with small defined error) specifies the given image and produces a deterministic automaton with the minimal number of states. The number of state can be small reduced, or extended by changing error or do tolerance for average grayness of image part. For reconstruct image from automata and compute a interesting point for image recognition we propose follow recursively algorithm.

Procedure *Reconstruct Image for Recognition*
For given automata A, we make image I_w the zoomed part of I in the square addressed w. The image represented by state number x is denoted by u_x.

1. Assign the initial state q_o to the image represented by the empty word, that is, to the whole image I, and define $i(q_o)=1$, $t(q_o)=\varnothing(\varepsilon)$, the average grayness of the image I, which we change to computed color, if we wont that.
2. Recursively, for a state q assign to square specified by a string u, consider four sub square specified by a string $u0, u1, u2, u3$. Denote the image in square by I_{ua}. If the image is everywhere $t(q_o)$ and word has shorter then requested, assign a new input state q that representative image specified by a input word uX where X is denoted part of image. Otherwise, assign a new input state $q(uY)$ where Y is a next part of image.
3. Repeat step 3 for each state, and stop if no founded new input state, or input word is a equal to requested.

The procedure *Reconstruct Image for Recognition* was stop for every automata computed by a procedure *Construct Automaton for Recognition,* or other similar algorithm.

With previous procedure can mark the interest point for recognition. There is many method for reflecting point. For example on Fig. 12 (for this and another example on left is original image and on right-hand computed image) is on the left image where lighter color is for state, that construct later (part has longest word) on the right is lighter color for the state with less sub square (white color is for state, has not sub square).

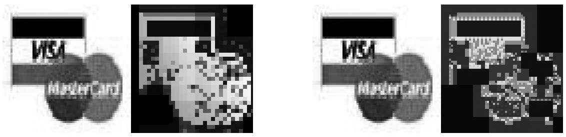


Figure 12. Reconstructed image on the left-hand with marked later state, on the right-hand with state, has not sub square.

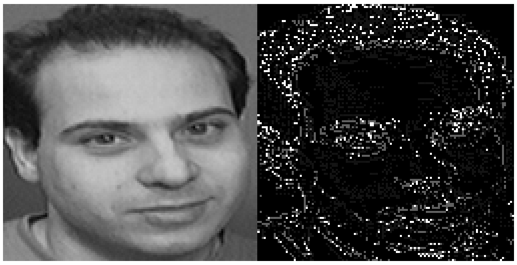


Figure 13. Reconstructed image with marked state.

On Fig. 13 is reconstructed image from automata which have deep 5 and compression is not loss. Number of state is 3317 and 43 latest has not a sub square. Lighter point on right-hand part is marked state without neighbor with same over squere. Some of this point is correspond with interest point (corners) on right part Fig. 10. Finally on Fig. 14 is decompressed image with error 16%, only 16 level of gray and computed automata have 175 state (latest 8 have not sub square). White dot on middle part is state without sub square. On right-hand is marked corners (red dot) computed by method described in [5], many of this point correspond with automata state. For compare on Fig. 15 is all state marked, where darkness color is newer state and white color have latest state.

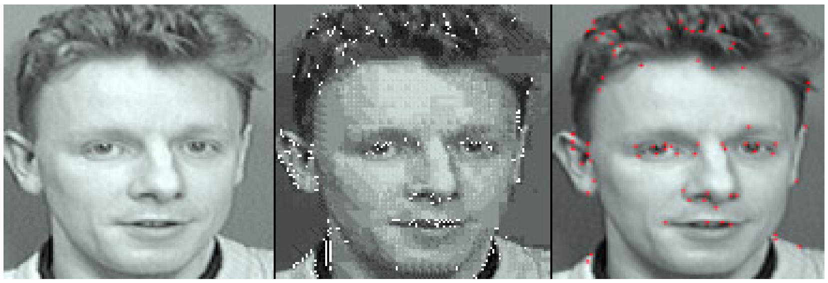


Figure 14. Reconstructed image with marked state.

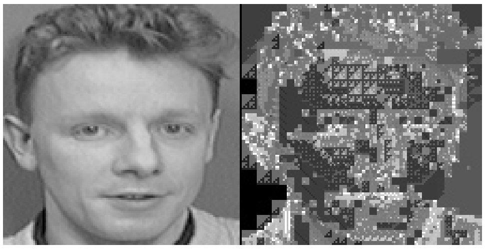


Figure 15. States of computed automata represented by color (on right-hand).

5 Conclusions

In this paper we have proposed an alternative solution for image recognition and finding a interesting points as a corners. This method is based on finite automata compression. The interesting property of this approach is an ability of similarity recognition.

References

1. K. Culik II and V. Valenta. Finite automata based compression of bi-level and Simple Color Images.
2. K. Culik II and J. Kari. Image compression Using Weighted Finite Automata, in *Fractal Image Compression: Theory a Techniques,* Ed. Yuval Fisher, Springer Verlag, pp 243-258 (1994)
3. R. Deriche and G.Giraudon, A computational approach for corner and vertex detection, *International Jurnal of Computer Vision*, 10(2), 101-124 (1993)
4. J.E.Hopcroft and J.D.Ullman. *Introduction to automata theory, languages and computation*. Addison-Wesley (1979).
5. E. Sojka, *A New Algorithm for Direct Corner Detection in Digital Images*, VŠB-Technical University of Ostrava, Faculty of Electrical Engineering and Computer Science, (2002)

Multi-dimensional Sparse Matrix Storage*

Jiří Dvorský, Michal Krátký

Department of Computer Science, VŠB – Technical University of Ostrava
17. listopadu 15, 708 33 Ostrava–Poruba
{jiri.dvorsky,michal.kratky}@vsb.cz

Abstract. Large sparse matrices play important role in many modern information retrieval methods. These methods, such as clustering, latent semantic indexing, performs huge number of computations with such matrices, thus their implementation should be very carefully designed. In this paper we discuss three implementations of sparse matrices. The first one is classical, based on lists. The second is previously published approach based on quadrant trees. The multi-dimensional approach is extended and usage of general multi-dimensional structure for sparse matrix storage is introduced in this paper.

Key words: sparse matrix, multi-dimensional data structure, quadrant tree, BUB-tree, R-tree

1 Introduction

Numerical computations represent serious problem for generations of mathematicians. There were not suitable device to make the computations, only human being. Development of computers gives to people power to perform computations, which were impossible in past. Many of these computations have matrix character. Thus one of the first task for computers was matrix and vector computations i.e. liner algebra. Although amount of memory in computers grows very rapidly, there are still matrices that are bigger than available memory. But many of these matrices are sparse, so that storage only non-zero values can solve the problem. Large sparse matrices play important role in industrial computations (e.g. FEM - Finite Elements Method), in computer science (indexing of class hierarchy [5]), and in many modern information retrieval methods. These methods, such as clustering, latent semantic indexing, performs huge number of computations with such matrices, thus their implementation should be very carefully designed.

This paper is organized as follows. Section 2 describe state-of-art in sparse matrix implementation. A previously published approach for sparse matrix storage [10] based on finite automata is given in Section 3. The multi-dimensional approach is extended and usage of general multi-dimensional structure for sparse

* This work was done under grant from the Grant Agency of Czech Republic, Prague No.: 201/03/1318

matrix storage is introduced in this paper. This storage method is described in Section 4. In Section 6 preliminary experimental results are shown. Finally, we conclude with a summary of contributions and discussion on future work.

2 Short survey of sparse matrix storage

Let $\mathcal{A}$ be a sparse matrix of order $n \times m$. The matrix $\mathcal{A}$ can be be efficiently processed, if the zero elements of $\mathcal{A}$ are not stored. There are many methods for storing the data (see for instance [1]). Here we will discuss Compressed Row and Column Storage.

2.1 Compressed Row Storage (CRS)

The Compressed Row Storage (CRS) format puts the subsequent nonzeros of the matrix rows in contiguous memory locations. Assuming we have a nonsymmetric sparse matrix $\mathcal{A}$, we create 3 vectors: one for floatingpoint numbers (val), and the other two for integers (col_{ind}, row_{ptr}). The val vector stores the values of the nonzero elements of the matrix $\mathcal{A}$, as they are traversed in a rowwise fashion. The col_{ind} vector stores the column indexes of the elements in the val vector. That is, if $val(k) = a_{i,j}$ then $col_{ind(k)} = j$. The row_{ptr} vector stores the locations in the val vector that start a row, that is, if $val(k) = a_{i,j}$ then $row_{ptr(i)} \leq k < row_{ptr(i+1)}$. By convention, we define $row_{ptr(n+1)} = n_{nz} + 1$, where n_{nz} is the number of nonzeros in the matrix $\mathcal{A}$. The storage savings for this approach is significant. Instead of storing n^2 elements, we need only $2n_{nz}+n+1$ storage locations.

The CRS format for this matrix is then specified by the arrays val, col_{ind}, row_{ptr} given in Table 1. If the matrix $\mathcal{A}$ is symmetric, we need only store the upper (or lower) triangular portion of the matrix. The tradeoff is a more complicated algorithm with a somewhat different pattern of data access.

2.2 Compressed Column Storage (CCS)

Analogous to Compressed Row Storage there is Compressed Column Storage (CCS), which is also called the *Harwell-Boeing sparse matrix format* [6]. The CCS format is identical to the CRS format except that the columns of $\mathcal{A}$ are stored (traversed) instead of the rows. In other words, the CCS format is the CRS format for $\mathcal{A}^T$.

The CCS format is specified by the 3 arrays val, row_{ind}, col_{ptr}, where row_{ind} stores the row indices of each nonzero, and col_{ptr} stores the index of the elements in val which start a column of $\mathcal{A}$. The CCS format for the matrix $\mathcal{A}$ in equation (1) is given in Table 2.

Example 1. As an example, consider the nonsymmetric matrix $\mathcal{A}$ defined by

$$\mathcal{A}=\begin{pmatrix} 10000 & -2 & 0 \\ 3900 & 0 & 3 \\ 0787 & 0 & 0 \\ 3087 & 5 & 0 \\ 0809 & 9 & 13 \\ 0400 & 2 & -1 \end{pmatrix} \tag{1}$$

Table 1. The CRS format for the matrix $\mathcal{A}$ in equation (1)

val	10	-2	3	9	3	7	8	7	3	...	9	13	4	2	-1
col_{ind}	1	5	1	2	6	2	3	4	1	...	5	6	2	5	6

row_{ptr}	1	3	6	9	13	17	20

Table 2. The CCS format for the matrix $\mathcal{A}$ in equation (1)

val	10	3	3	9	7	8	4	8	8	...	9	2	3	13	-1
row_{ind}	1	2	4	2	3	5	6	3	4	...	5	6	2	5	6

col_{ptr}	1	4	8	10	13	17	20

2.3 Properties of CRS and CCS formats

The Compressed Row and Compressed Column Storage formats are general formats: they make absolutely no assumptions about the sparsity structure of the matrix, and they does not store any unnecessary elements.

On the other hand, these methods effectively support only part of matrix operations. While CRS can access any row vector in time $O(1)$, column vector can be selected in $O(m \times \log_2 \Delta)$, where $\Delta = row_{ptr(i)} - row_{ptr(i+1)}$ ie. number of nonzero elements in row i. Time complexity of these operations in CCS format is reverse. For example CRS format can effectively perform matrix - colum vector and CCS row vector - matrix multiplication. Any other matrix operation (eg. selection of submatrix) can be done with these formats, but time complexity is very high. Moreover the formats can be used only in the main memory of computer.

Aim of our work is to develop storage format for large sparse matrices. The format should support:

– random access to the matrix,
– effective selection of any submatrix
– persistence of the matrix (usage of secondary memory).

3 Sparse matrices and finite automata

Culik and Valenta [4] introduced finite automata for compression of bi-level and simple color images. A digitized image of the finite resolution $m \times n$ consists of $m \times n$ pixels each of which takes a Boolean value (1 for black, 0 for white) for bilevel image, or a real value (practically digitized to an integer between 0 and 256) for a grayscale image.

Sparse matrix can be viewed, in some manner, as simple color image too. Zero element of matrix corresponds to white pixel in bi-level image and nonzero element to black or gray-scale pixel.

Here we will consider square matrix $\mathcal{A}$ of order $2^n \times 2^n$ (typically $13 \leq n \leq 24$). In order to facilitate the application of finite automata to matrix description we will assign each element at $2^n \times 2^n$ resolution a word of length n over the alphabet $\Sigma = \{0, 1, 2, 3\}$ as its address. A element of the matrix corresponds to a subsquare of size 2^{-n} of the unit square. We choose ε as the address of the whole square matrix.

Its submatrices (quadrants) are addressed by single digits as shown in Figure 1(a). The four submatries of the matrix with address ω are addressed $\omega 0$, $\omega 1$, $\omega 2$ and $\omega 3$, recursively. Addresses of all the submatrices of dimension 4×4 are shown in Figure 1(b). The submatrix (element) with address 3203 is shown on the right of Figure 1(c).

In order to specify a values of matrix of dimension $2^n \times 2^n$, we need to specify a function $\Sigma^n \to R$, or alternately we can specify just the set of non-zero values, i.e. a language $L \subseteq \Sigma^n$ and function $f_{\mathcal{A}} : L \to R$.

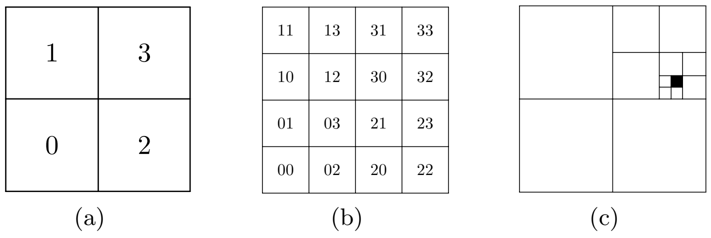


Fig. 1. The addresses of the submatrices (quadrants), of the subsmatrices of dimension 4×4, and the submatrix specified by the string 3203

This kind of storage system allows direct access to stored matrix. Each of elements can be accessed independently to previous accesses and access to each element has same, constant time complexity. Let $\mathcal{A}$ be a matrix of order $2^n \times 2^n$. Then time complexity of access is bounded by $O(\log_2 n)$. For detail information see [10].

Example 2. Let $\mathcal{A}$ be a matrix of order 8×8.

$$\mathcal{A} = \begin{pmatrix} 20000000 \\ 04001000 \\ 00300609 \\ 00010000 \\ 00001000 \\ 00000500 \\ 00000090 \\ 00000007 \end{pmatrix}$$

The language $L \subseteq \Sigma^3$ is now

$$L = \{111, 112, 121, 122, 211, 212, 221, 222, 303, 310, 323\}.$$

Then function $f_{\mathcal{A}}$ will have following values (see Table 3).

Table 3. Positions in matrix $\mathcal{A}$ and corresponding values – function $f_{\mathcal{A}}$

$x \in L$	$f_{\mathcal{A}}(x)$	$x \in L$	$f_{\mathcal{A}}(x)$
111	2	221	9
112	4	222	7
121	3	303	6
122	1	310	1
211	1	323	9
212	5		

Now automaton that computes function $f_{\mathcal{A}}$ can be constructed (see Figure 2). The automaton is 4-ary tree, where values are stored only at leaves. This knowledge leads to multi-dimensional sparse matrix storage and usage of the quadrant tree.

4 Multi-dimensional sparse matrix storage

In order to a general multi-dimensional data structure can be used for the sparse matrix storage, the following definitions must be introduced.

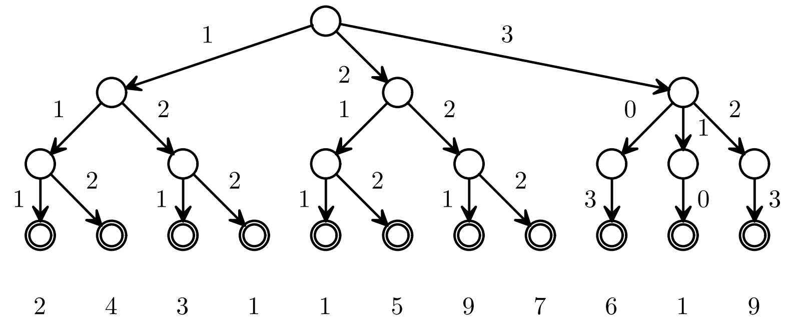


Fig. 2. Automaton for matrix $\mathcal{A}$

Definition 1 (A matrix as tuples of 2-dimensional space).

Let $\mathcal{A}$ be a matrix of order $n \times m$ and $\Omega_{MT} = D_N \times D_M$ be an 2-dimensional discrete space (called matrix space), where $D_N = \{0, 1, \ldots, 2^{l_N} - 1\}$, $D_M = \{0, 1, \ldots, 2^{l_M} - 1\}$. It holds $n \leq 2^{l_N} - 1$, $m \leq 2^{l_M} - 1$. For all $a_{i,j} \in \mathcal{A}$ there is mapping $\alpha : \mathcal{A} \rightarrow \Omega_{MT}$ such that $\alpha(a_{i,j}) = (i, j)$.

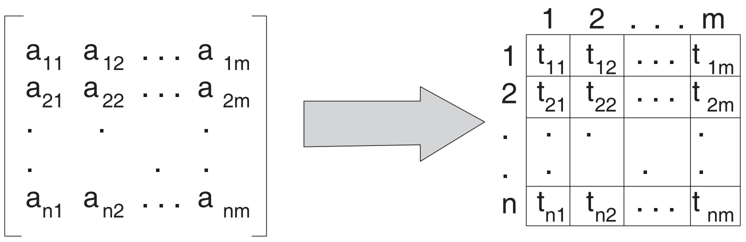


Fig. 3. A matrix as tuples of 2-dimensional space.

The mapping α transforms elements of matrix $\mathcal{A}$ to 2-dimensional space Ω_{MT}. The matrix space can be seen in Figure 3. The matrix space seems to be 3-dimensional. However, indices of matrix elements have to be indexed. The value of element is stored as non-index data. Consequently, only two coordinates must be indexed, so that the matrix space is only 2-dimensional.

4.1 Retrieving of a sub-matrix

A sub-matrix is retrieved using the range query.

Definition 2 (Range query).

Let Ω be an n-dimensional discrete space, $\Omega = D^n$, $D = \{0, 1, \ldots, 2^{l_D} - 1\}$, and points (tuples) $T^1, T^2, \ldots, T^m \in \Omega$. $T^i = (t_1, t_2, \ldots, t_n)$, l_D is the chosen length of a binary representation of a number t_i from domain D. The range query *RQ is defined by a* query hyper box *(*query window*) QB which is determined by two points $QL = (ql_1, \ldots, ql_n)$ and $QH = (qh_1, \ldots, qh_n)$, QL and $QH \in \Omega$, ql_i and $qh_i \in D$, where $\forall i \in \{1, \ldots, n\} : ql_i \leq qh_i$. This range query retrieves all points $T^j = (t_1, t_2, \ldots, t_n)$ in the set $T^1, T^2, \ldots, T^m$ such as $\forall i : ql_i \leq t_i \leq qh_i$.* ■

Let be $\mathcal{A}_{i_1 j_1 i_2 j_2}$ a sub-matrix of matrix $\mathcal{A}$. Sub-matrix is retrieved from the matrix space using the range query $(i_1, j_1) : (i_2, j_2)$. A column vector and row vector are special kind of the sub-matrix. Consequently, the column vector $c_i^{\mathcal{A}}$, $1 \leq i \leq m$, is retrieved using the range query $(1, i) : (n, i)$, the row vector $r_j^{\mathcal{A}}$, $1 \leq j \leq n$, is retrieved using the range query $(j, 1) : (j, m)$. Such range query is called the *narrow range query*. Next Section describes some multi-dimensional data structures, especially a multi-dimensional data structure for efficient processing of the narrow-range query.

5 Multi-dimensional data structures

Due to the fact that a matrix is represented as a set of points in 2-dimensional space in the multi-dimensional approach, we use multi-dimensional data structures for their indexing, e.g., paged and balanced multi-dimensional data structures like UB-tree [2], BUB-tree [7], R-tree [8], and R*-tree [3].

(B)UB-tree data structure applies *Z-addresses* (*Z-ordering*) [2] for mapping a multi-dimensional space into single-dimensional. Intervals on *Z-curve* (which is defined by this ordering) are called *Z-regions*. (B)UB-tree stores points of each Z-regions on one disk page (tree leaf) and a hierarchy of Z-regions forms an index (inner nodes of tree). In Figure 4(a) we see two-dimensional space with 8 points (tuples) and Z-regions dividing the space. Figure 4(b) denotes schematically a BUB-tree indexing this space.

In the case of indexing point data, an R-tree and its variants cluster points into *minimal bounding boxes* (*MBB*s). Leafs contain indexed points, super-leaf nodes include definition of MBBs and the other inner nodes contain hierarchy of MBBs. (B)UB-tree and R-tree support *point* and *range queries* [11], which are used in the multi-dimensional approach to sparse matrix storage. The range query is processed by iterating through the tree and filtering of irrelevant tree nodes, i.e. (super)Z-regions in the case of (B)UB-tree and MBBs in the case of R-tree, which do not intersect a query box.

The range query often used in the multi-dimensional approach is called *narrow range query*. Points defining a query box have got some coordinates the same, whereas the size of interval defined by other coordinates near to the size of space's domain. Notice, regions intersecting a query box during processing of a range query are called *intersect regions* and regions containing at least one point of the query box are called *relevant regions*. We denote their number by

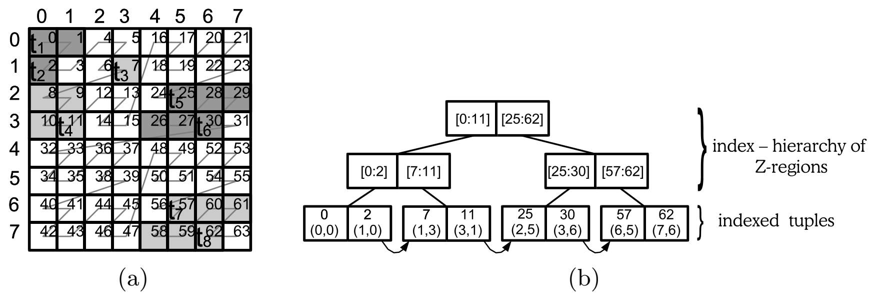


Fig. 4. (a) 2-dimensional space 8 × 8 with points $t_1 - t_8$. These points define partitioning of the space to Z-regions [0:2],[7:11],[25:30],[57:62] by capacity of BUB-tree's nodes 2. (b) BUB-tree indexing this space.

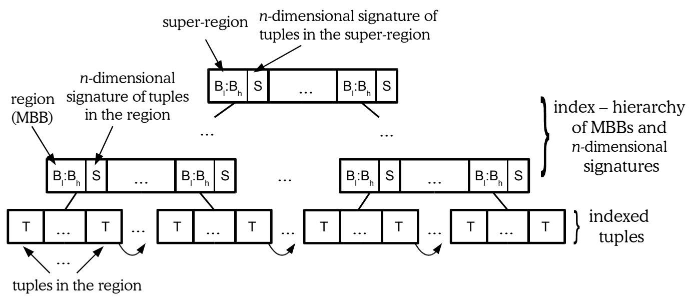

super-region
n-dimensional signature of tuples in the super-region
region (MBB)
n-dimensional signature of tuples in the region
index – hierarchy of MBBs and n-dimensional signatures
indexed tuples
tuples in the region

Fig. 5. A structure of the Signature R-Tree.

N_I and N_R, respectively. Many irrelevant regions are searched during processing of the narrow range query in multi-dimensional data structures. Consequently, a ratio of relevant and intersect regions, so called *relevance ratio* $c_R \ll 1$ with an increasing dimension of indexed space. In [9] Signature R-tree data structure was introduced. This data structure enables efficient processing of the narow range query. Items of inner nodes contain a definition of (super)region and n-dimensional signature of tuples included in the (super)region (see Figure 5). A superposition of tuples of coordinates by operation OR creates the signature. Operation AND is used for better filtration of irrelevant regions during processing of the narrow range query. Other multi-dimensional data structures (e.g. (B)UB-tree) are possible to extend in the same way.

6 Experimental results

In our experiments[1], we used a randomly generated sparse matrix $10^7 \times 10^6$. The matrix contains 5×10^6 of non-zero values. The BUB-tree was used for our test. The index size is 80 MB (compare to 38MB of CRS matrix storage). In Table 4 a characterization of the BUB-tree for storage of the matrix is shown.

Table 4. A characterization of BUB-tree used for sparse matrix storage

Dimension	2	Utilisation	68.1%
l_N, l_M	24	D_N, D_M	$2^{24} - 1$
Number of tuples	5,244,771		
Number of inner nodes	20,351	Number of leaf nodes	249,297
Inner node capacity	19	Leaf node capacity	30
Item size	12 B	Node size	308 B

In the test, randomly generated column and row vectors were retrieved from the BUB-tree. The average number of result tuples (items of a sub-matrix), searched leaf nodes (Z-regions), disk access cost (DAC), and time were measured. A ratio of the searched leaf nodes and all leaf nodes is shown in square brackets. Table 4 shows the result of our tests.

Table 5. Experimental results of the multi-dimensional sparse matrix storage

Number of result tuples	**Number of searched leaf nodes**	**DAC**	**Time** [s]
199	101 [0.041%]	285	0.04

We see that very small part of the index was searched and time of searching was low as well. Experiments prove the approach can serve as efficient sparse matrix storage. The index size is lager than in the case of classical CRS or CCR sparse-matrix storage, but a arbitrary sub-matrix may be retrieved in our approach.

7 Conclusion

In this contribution the multi-dimensional approach to indexing sparse matrix was described. Previously published approach [10] using the quad tree was de-

[1] The experiments were executed on an Intel Pentium ®4 2.4Ghz, 512MB DDR333, under Windows XP.

scribed and a general multi-dimensional approach was introduced. Our experiments prove the approach can serve as efficient sparse matrix storage. In our future work, we would like further to test our approach over a real matrix and to compare the approach with other sparse matrix storage approaches.

References

1. R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. V. der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition.* SIAM, Philadelphia, PA, 1994.
2. R. Bayer. The Universal B-Tree for multidimensional indexing: General Concepts. In *Proceedings of WWCA'97, Tsukuba, Japan*, 1997.
3. 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 International Conference on Management of Data*, pages 322–331.
4. K. Culik and V.Valenta. Finite automata based compression of bi-level and simple color images. In *Computer and Graphics*, volume 21, pages 61–68, 1997.
5. P. Dencker, K. Drre, and J. Heuft. Optimization of parser tables for portable compilers. *ACM Transactions on Programming Languages and Systems*, 6(6):546–572, 1984.
6. I. S. Duff, R. G. Grimes, and J. G. Lewis. Sparse matrix test problems. *ACM Trans. Math. Softw.*, 15(1):1–14, 1989.
7. R. Fenk. The BUB-Tree. In *Proceedings of 28rd VLDB International Conference on VLDB*, 2002.
8. 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.
9. M. Krátký, V. Snášel, J. Pokorný, P. Zezula, and T. Skopal. Efficient Processing of Narrow Range Queries in the R-Tree. In *Submitten at VLDB 2004*, 2003.
10. V. Snášel, J. Dvorský, and V. Vondrák. Random access storage system for sparse matrices. In G. Andrejková and R. Lencses, editors, *Proccedings of ITAT 2002*, Brdo, High Fatra, Slovakia, 2002.
11. C. Yu. *High-Dimensional Indexing.* Springer–Verlag, LNCS 2341, 2002.

Design of Structure and Realisation of Game Rules Database of Robot-Soccer Game

Bohumil Horák[1] and Václav Snášel[2]

[1]Department of Measurement and Control, FEI, VŠB - Technical University of Ostrava, 17.listopadu 15, 708 33 Ostrava-Poruba, Czech Republic
[2]Department of Computer Science, FEI, VŠB - Technical University of Ostrava, 17. listopadu 15, 708 33, Ostrava-Poruba, Czech Republic
{bohumil.horak, vaclav.snasel}@vsb.cz

Abstract. In this paper we developed system for coordinatization the robot-soccer game. This coordinatization we want to use for strategy extraction. The robot soccer is bimilar ant-like systems, which take advantage of agents' situatedness to reduce or eliminate the need for centralized control or global knowledge. This reduces the need for complexity of individuals and leads to robust, scalable systems. Such insect-inspired situated approaches have proven effective both for task performance and task allocation. The desire for general, principled techniques for situated interaction has led us to study the exploitation of abstract situatedness – situatedness in non-physical environments. The port-arbitrated behavior-based control approach provides a well-structured abstract behavior space in which agents can participate in situated interaction. We focus on the problem of role assumption, distributed task allocation in which each agent selects its own task-performing role. This paper details our discretization the robot-soccer game.

Keywords: mobile robotics, multi-robot coordination, behavior-based control, group behavior

1 Introduction

The typical example of distributed control system with embedded systems is the proposal of control system of mobile robots for the task robot-soccer game. The selection of this game for laboratory task was the motivation both for students and for the teachers as well because this was a question of proposal and realization of complicated multidisciplinary task which can be divided into a whole number of partial tasks (the evaluation of visual information and processing of image, the hardware and software implementation of distributed control system, wireless data transmission and processing of informations and the control of robots). For sophistication of the own (and opponent) game and strategy is necessary her description. With it is related design of structure of game-rules-database of robot-soccer game.

Ant-like systems take advantage of individual agents' situatedness to reduce or eliminate the need for centralized control or global knowledge. This reduces the need for complexity (of sensing, computation, and communication) of individuals and leads to robust, scalable systems. Such insect-inspired situated approaches have proven effective both for task performance see [4,12,5].

The game system is represented by up to 11 own and 11 opponent autonomous mobile robots at game site up to 200x100cm. The core of mobile robot is digital signal processor Motorola DSP56F805. PWM outputs of signal processor are connected to pair of power H-bridge circuits, which supply pair of DC drives with integrated pulse encoders. For communication with the higher level of control system is used the communication module with the control IC Nordic nRF2401. The higher level of control system is represented by personal computer. In the PC entered the signal, which represent the picture of scene with robots scanned with above the playground placed CCD camera. At the output is connected radio line which transmits commands for all own mobile robots. The software part of distributed control system is realized by decision making and executive agents. The algorithm of the agents cooperation was proposed with the control agent on higher level. The algorithms for agents realized in robots are the same. The control agent determines the required behaviour of the whole control system as the response on dynamic behaviour of robots and on the own global strategy of the task and knowledge about last situations which are saved in the database of the scene. The agent on higher level controls the other agents [11]. The separate task is the transformation which converts the digital picture into the object coordinates (robots and ball in the task of robots soccer) which are saved in the database of the scene [1]. This database is common for all agents in the control system. Each agent sees actual the whole scene and is capable to control its behaviour in a qualified way. The basic characteristic of control algorithm of subordinate agent is the independence on number of decision making agents for robots on the game site. Both agent teams (own and opponent) have common goal, to score the goal and not to have any. For successful assertion of own game strategy is very important the extraction and knowledge of opponent game strategy. From object coordinates of samples of picture scene and game-rules-database create strategy extraction algorithms the own (and opponent) game strategy database.

2 Game site

Own and opponent robots created by own movements very dynamic changed environment. This environment is scanned by CCD camera with sample frequency (in present time) up to 75 fps. Picture sample before processing is demonstrated at Fig.1.

In our approach, a game is coded as a game matrix. We can extract vector V_t in time t from game:

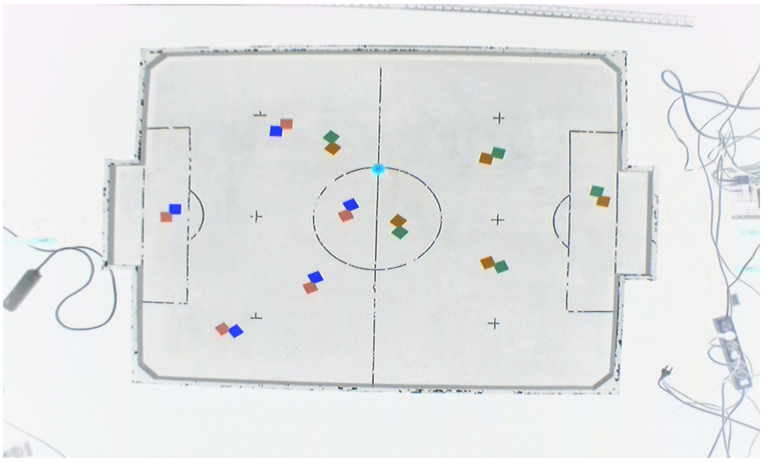


Fig. 1. Picture sample from CCD camera signal before processing.

$V_t = \{t, X_1; Y_1; \alpha_1; X_2; Y_2; \alpha_2; X_3; Y_3; \alpha_3; X_{100}; Y_{100}; \alpha_{100}; X_{200}; Y_{200}; \alpha_{200}; X_{300}; Y_{300}; \alpha_{300}; X_{30}; Y_{30}; \alpha_{30}\}$

Where are:

X_i	is x coordinate of own robot i
Y_i	is y coordinate of own robot i
α_i	is angle of orientation of own robot i
X_{i00}	is x coordinate of opponent robot i
Y_{i00}	is y coordinate of opponent robot i
α_{i00}	is angle of orientation of opponent robot i
X_{30}	is x coordinate of ball
Y_{30}	is y coordinate of ball
α_{30}	is angle of motion α ball

A game matrix GM we can define by following way:

$GM = (V_0^T, V_1^T, \ldots, V_{n-1}^T, V_n^T)$

The game matrix GM is very wide matrix (up to 420000 vectors - processed samples in time of game). This matrix inputs in the proper extraction of game strategy process by using the latent semantic analysis (LSA).

LSA is a statistical model of word usage that permits comparisons of the semantic similarity between pieces of textual information. Was originally designed to improve the effectiveness of information retrieval methods by performing retrieval based on the derived "semantic" content of words in a query as opposed to performing direct word matching. LSA was used for extraction semantic in many other situations see [2,3,4].

In this paper, LSA is used as a tool for strategy extraction problem. In our approach, a game is coded as a game matrix.

The results in [2,3,8,9,6] indicate that LSA can perform matching based on semantic content. The game matrix we analyze by LSA and obtain semantic information about game. This semantic information we can interpret as strategy. This strategy is use for for agent management see [10,7].

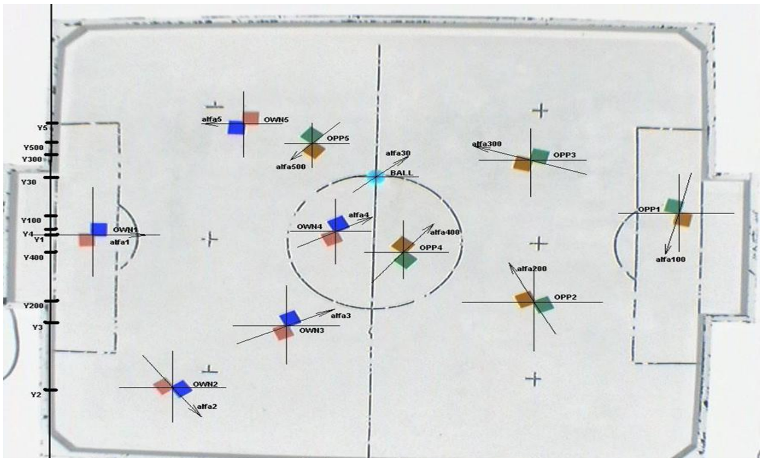


Fig. 2. Picture sample with transposed marked positions.

Extent of real position-game-matrix is wide (1024x768 pts, this is up to 787 kB). Very important is sequentially data reduction without information loss. The presented idea of game site description is similar to board game - chess. Thus be created virtual grid covered the same game site. Dimensions of grid are calculated from technical parameters of CCD camera and velocity of mobile robots. This step allows position data reduction without information loss up to 200-times (in dependence on grid dimensions). Reduction of data volume allows increase computation speed and advantageous motion description.

3 Virtual grid

The virtual grid allows reduction of data volume for game-strategy purposes and easy motion description. Description system with virtual grid works parallel

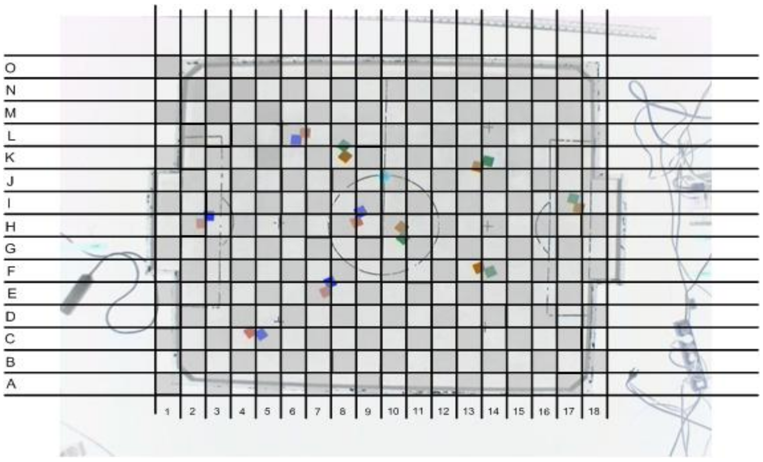


Fig. 3. Picture sample with (similar to chess) transposed grid.

with real coordinate system for exact sensing of subject position. Data volume of description with primary and secondary virtual grid is dependent on frequency of samples of used CCD camera (25-75 fps) and velocity of movement of mobile robot (robot-soccer player) at game site (up to 2,5m/s). In constituent discrete frame samples is possible study of movements of own (and opponent) robots. Distance between two points, which drive the robot at two in sequence frames, determine dimensions of primary virtual grid. To calculate with velocity of movements of robot, was primary virtual grid divided to more $(2, 4, 8, \ldots)$ parts. So was created secondary virtual grid (in next SVG).

Note: Primary virtual grid has, with maximal robot velocity 2,5m/s and frame samples frequency of CCD camera 25Hz, a dimension 10 x 10cm. By other velocity of robot and other frame samples frequency will be dimensions of primary virtual grid others.

Description of robot position and game movements If is for description of movements and velocity of robot in one frame sample used the secondary virtual grid, is afterwards possible alphanumeric description of robot and his direction and velocity of movement. Description is illustrated at Fig.4.

Note: Alphanumeric description of position of robot in given picture sample, his velocity of movement and direction of movement considering to previous picture sample by calculation or for prediction for next progression of game situation is possible describe by symbol (alphabetic) of player e.g. attacker (A), goalkeeper (G) and defender (D), by symbol (numeric) sequence of player function in team $(1, 2, \ldots)$, by symbol (alphanumeric) his current position in secondary virtual grid e.g. (HA24) and by symbol (alphanumeric) in case of strategic planning of his next direction and velocity of movement (HC24). Situation illustrates Fig.5. Robot goalkeeper No. 1 state at position CHA24 in secondary virtual

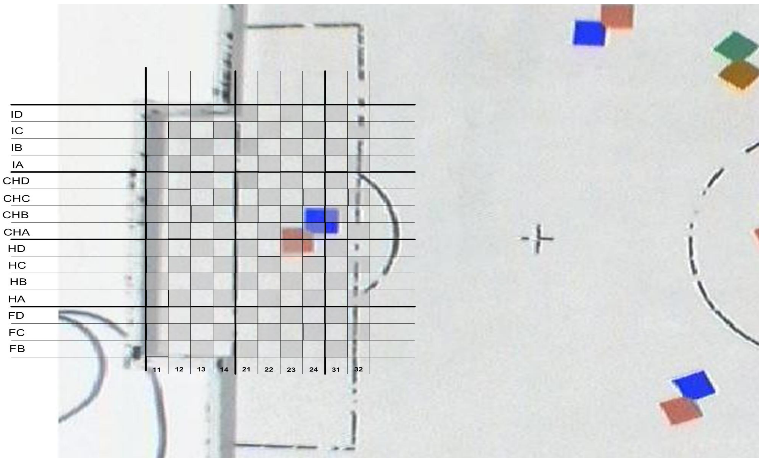


Fig. 4. Illustration of robot position description in SVG $\frac{1}{4}$ (Goalkeeper at position [CHA,24]).

grid (with division 4SVG=1PVG). Is planned strategic expedient movement (at HC24) with direction y-axis with velocity vT (vT = 3/4 . vmax). Alphanumeric description of such movement is [G1HA24HC24].

4 Strategic game movements

Strategic game movements each of robotic players can be dynamic changed and unreeled from their function during game progress and their momentary position at game site, position of opponents and ball. Game progress can be divided to three ground playing classes (in next GPC) and two ground playing situations (in next GPS):

- GPC of game opening (GPCO)
- GPC of movements in game site (GPCS)
- GPC of game end (GPCE)
- GPS attack (GPSA)
- GPS defence (GPSD)

Offensive: Interaction of simple behaviors causes the robots to fall into a V-formation when the ball is in motion roughly towards the opponent's goal. Perceptual properties limit the formation to three robots.

Defensive: When the ball is not moving roughly towards the opponent's goal, the robots cluster around it to form an effective barrier and be in good

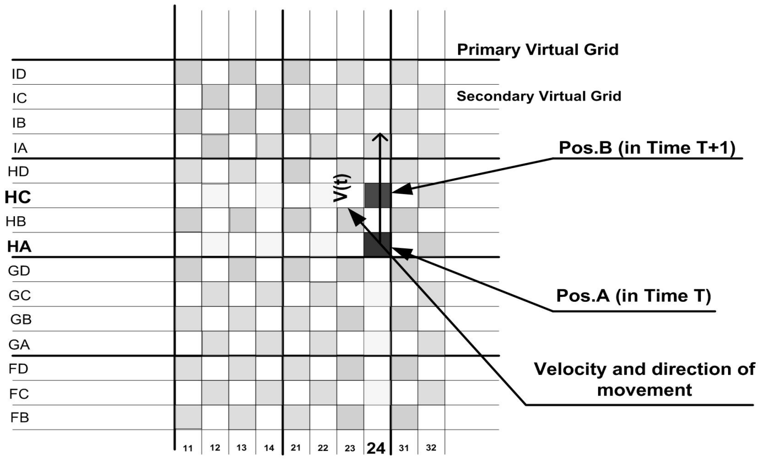


Fig. 5. Illustration of robot game movements with description [G1HA24HC24].

positions for recovery. to use similar means of assuming effcient roles. Here we discuss a system we have implemented for robotic soccer - which is also able to use local interactions to determine globally efficient roles.

Each GPC have own different movement rules. Classes GPCO and GPCE consists of finite number of movement variants. These come out from defined positions of players and ball at game site and defined direction of ball movement. Class GPCS have infinite number of movement variants, limited in current game situation (GPS) by ground game rules and game situation supported by own global game strategy (in next GGS).

Example of limitation of robot movements under the influence of his function during the game process. Example is presented via robotic player in goalkeeper function. Charge of goalkeeper is preventing the opponent to score a goal. His movements are, with only for minor exceptions, limited at own goalmouth near of goal line. Preferred movements are in goal line direction. Preference of these movements comes from idea of GGS, when goalkeeper prevent to score of goal so, that build own new position (near goal line) at line between the central goal point and ball (their point - centre of gravity and/or ball last movement vector).

Preference of other movement directions be created with GPSA, when movements of goalkeeper must secured kick away the ball from own defence zone. Situation illustrates Fig.5.

5 Conclusion

The algorithm of the control system should be proposed in a such way so that it would ensure the requirements for immediate response of control, so that the system with robots would be controlled in real-time. That is why, it is very important so that the algorithm for critical speed would be optimized. The system response should be shorter than time between two frames from camera. In the event that this limit is exceeded, the frame is cut out and the control quality is decreased. The main possibilities of the algorithm adjustment are as follows:

– Dynamic control in control and decision module of control agent.
– The control and decision modules and communication protocol of the decision agents.
– Strategy of planning in control model of the action agent.
– Extraction of opponent game strategy and using of extraction results for decision rules generation as a part of rules decision database of decision agent

It is necessary to know, that the system response should take a shorter time than the time between the frames from the PAL movie camera, e.g. 20 ms. If this limit is exceeded, the frame is dropped and a control quality decreases. Parallel proceed extraction algorithms of own (and opponent) game strategy collaborated with game-rules-database allows add and refine informations of own (and opponent) game strategy. This way is expand the rule database of decision agent for decision making within the bounds of own game strategy given by control agent.

References

1. Bernatík,R., Horák,B., Kovář,P. (2001): Quick image recognize algorithms. In: Proceeding International workshop Robot-Multi-Agent-Systems R-MAS 2001. VSB Ostrava 2001, Czech Republic, ISBN 80-7078-901-8 p.53-58.
2. Berry, M. W., Browne, M. (1999): Understanding Search Engines: Mathematical Modeling and Text Retrieval. SIAM Book Series: Software, Environments, and Tools, (June 1999), ISBN: 0-89871-437-0.
3. Berry, M. W. (Ed.) (2003): Survey of Text Mining: Clustering Classification, and Retrieval. Springer Verlag 2003.
4. J. L. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain, and L. Cretien. The dynamics of collective sorting: Robot-like ants and ant-like robots. In Proceedings of the First International Conference on Simulation of Adaptive Behavior: From Animals to Animats, pages 356–363. MIT Press, 1991
5. O. Holland and C. Melhuish. Stigmergy, self-organisation, and sorting in collective robotics. Artiffcial Life, 5:2:173–202, 2000.
6. Húsek D., Frolov A. A., Řezanková H., Snášel V. (2002): Application of Hopfield-like Neural Networks to Nonlinear Factorization. COMPSTAT 2002, Proceedings in Computational Statistics (Eds.: Hrdle W., Rnz B.), 177-182. Physica-Verlag, Heidelberg 2002. ISBN 3-7908-1517-9.

7. M.Obitko, Snášel V. (2004):Ontology Repository in Multi-Agent System. IASTED, International Conference on ARTIFICIAL INTELLIGENCE AND APPLICATIONS (AIA 2004), Innsbruck, Austria.
8. Praks P., Dvorský J., Snášel V.(2003): Latent Semantic Indexing for Image Retrieval Systems. SIAM Conference on Applied Linear Algebra (LA03) The College of William and Mary, Williamsburg, U.S.A. 2003.
9. Praks P., Dvorský J., Snášel V., Černohorský J. (2003): On SVD-free Latent Semantic Indexing for Image Retrieval for application in a hard industrial environment. IEEE International Conference on Industrial Technology - ICIT'03, Maribor 2003.
10. J.Smid, M.Obitko, Snášel V. (2004): Communicating Agents and Property-Based Types versus Objects. Sofsem MatfyzPress 2004.
11. Srovnal V., Pavliska,A. (2002): Robot Control Using UML and Multi-agent System. In: Proceeding 6th World Multiconference SCI 2002. Orlando, Florida, USA, ISBN 980-07-8150-1, p.306-311.
12. B.B.Werger and M.J.Matarič. From Insect to Internet: Situated Control for Networked Robot Teams. Annals of Mathematics and Artiffcial Intelligence (2000).

Author Index