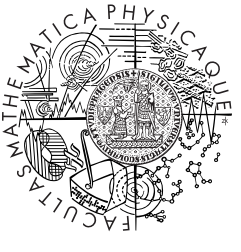
Charles University in Prague, MFF, Department of Software Engineering
Czech Technical University in Prague, FEE, Dept. of Computer Science & Eng.
VŠB–TU Ostrava, FEECS, Department of Computer Science
Czech Society for Cybernetics and Informatics

Proceedings of the Dateso 2007 Workshop

DATESO 2007

Databases, Texts, Specifications, and Objects

http://www.cs.vsb.cz/dateso/2007/
http://www.ceur-ws.org/Vol-235/



April 18 – 20, 2007
Desná – Černá Říčka

DATESO 2007

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

Page count:	147
Publication:	196$^{\text{th}}$
Impression:	150
Edition:	1st
First published:	2007

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

Published by MATFYZPRESS publishing house of Faculty of Mathematics and Physics Charles University, Ke Karlovu 3, 121 16 Praha 2, Czech Republic as its 196$^{\text{th}}$ publication.

Preface

DATESO 2007, the international workshop on current trends on Databases, Information Retrieval, Algebraic Specification and Object Oriented Programming, was held on April 18 – 20, 2007 in Desná – Černá Říčka. This was the 7^{th} annual workshop organized by VŠB-Technical University Ostrava, Department of Computer Science, FEL ČVUT Praha, Department of Computer Science and Engineering and MFF UK Praha, Department of Software Engineering. The DATESO aims for strengthening the connection between this various areas of informatics. The proceedings of DATESO 2007 are also available at DATESO Web site `http://www.cs.vsb.cz/dateso/2007/` `http://www.ceur-ws.org/Vol-235/`.

The Program Committee selected 13 papers (11 full papers and 2 posters) from 23 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. Special thanks belong to Czech Society for Cybernetics and Informatics for its support of publishing this issue.

March, 2007 J. Pokorný, V. Snášel, K. Richta (Eds.)

Program Committee

Jaroslav Pokorný (chair)	Charles University, Prague
Václav Snášel	VŠB-Technical University of Ostrava, Ostrava
Karel Richta	Czech Technical University, Prague
Vojtěch Svátek	University of Economics, Prague
Peter Vojtáš	Charles University, Prague
Tomáš Skopal	Charles University, Prague

Organizing Committee

Pavel Moravec	VŠB-Technical University of Ostrava
David Hoksza	Charles University, Prague

Table of Contents

Full Papers

Syllable-Based Burrows-Wheeler Transform 1
Jan Lánský, Katsiaryna Chernik, Zuzana Vlčková

Updating Typed XML Documents Using a Functional Data Model 11
Pavel Loupal

Genetic Algorithms in Syllable-Based Text Compression 21
Tomáš Kuthan, Jan Lánský

Using XSEM for Modeling XML Interfaces of Services in SOA 35
Martin Nečaský

A Modular XQuery Implementation 47
Jan Vraný, Jan Žák

A Content-Oriented Data Model for Semistructured Data 55
Tomáš Novotný

Index-Based Approach to Similarity Search in Protein and Nucleotide Databases 67
David Hoksza, Tomáš Skopal

Using BMH Algorithm to Solve Subset of XPath Queries 81
David Toth

Shape Extraction Framework for Similarity Search in Image Databases .. 89
Jan Klíma, Tomáš Skopal

Inductive Models of User Preferences for Semantic Web 103
Alan Eckhardt

Improvement of Text Compression Parameters Using Cluster Analysis ... 115
Jiří Dvorský, Jan Martinovič

Posters

Work with Knowledge on the Internet – Local Search 127
Antonín Pavlíček, Josef Muknšnábl

The Use of Ontologies in Wrapper Induction 132
Marek Nekvasil

Author Index 136

Syllable-Based Burrows-Wheeler Transform*

Jan Lánský, Katsiaryna Chernik, and Zuzana Vlčková

Charles University, Faculty of Mathematics and Physics
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
{zizelevak, kchernik, zuzana.vlckova}@gmail.com

Abstract. The Burrows-Wheeler Transform (BWT) is a compression method which reorders an input string into the form, which is preferable to another compression. Usually Move-To-Front transform and then Huffman coding is used to the permutated string. The original method [3] from 1994 was designed for an alphabet compression. In 2001, versions working with word and n-grams alphabet were presented. The newest version copes with the syllable alphabet [7]. The goal of this article is to compare the BWT compression working with alphabet of letters, syllables, words, 3-grams and 5-grams.

1 Introduction

The Burrows-Wheeler Transform is an algorithm that takes a block of text as input and rearranges it using a sorting algorithm. The output can be compressed with another algorithms such as bzip. To compare different ways of document parsing and their influence on BWT, we decided to deal with natural units of the text: letters, syllables and words; and compare these approaches with each other and also with the methods that divide the text into unnatural units – N-grams. In our measurements we found out that depending upon the language, words have 5–6 letters and syllables 2–3 letters in average. Therefore 3-grams were chosen to conform to the syllable length and 5-grams to correspond to average words length.

If we want to compare different BWT for various ways of text file parsing, it is necessary to use an implementation modified only in document parsing; the rest of compression method will stay unchanged.

Very important BWT parameter is a block size the document is compressed into; the larger block, the better results. For example, in bzip2 algorithm [14], the maximum block size is 900 kB. We decided to choose the block size so that any document up to 5MB could be considered as a single block. This approach is fairly-minded since e.g. word methods will not be favoured by considering the document as one block whilst letter methods would split the text into several blocks.

For our purposes, we had to alter two method's properties: We modified the XBW method [3] to be able to compress above all required entities. The XBW

* This research was partially supported by the Program "Information Society" under project 1ET100300419.

transform represents a labeled tree using two arrays, and supports navigation and search operations by means of standard query operations on arrays. Since this method was designed for syllable and word compression, it was easy to adjust it to compress also above remaining entities (letters, 3-grams and 5-grams). Since this method was intended as XML compression algorithm, we had to suit it on plain text documents - it was the second modification.

XBW method does not use syllable or word models attributes which predict a word or syllable type alternation.

In section 2 we try different strategies of document parsing. Section 3 details XBW method, section 4 describes experiments and their results. Section 5 contains the conclusion.

2 Parsing strategies

We parse an input document into words, syllables, letters, 3-grams and 5-grams. All above mentioned entity type division examples of string "consists of" are introduced in the table 1.

The word-based methods require parsing the input document into a stream of words and non-words. Words are usually defined as the longest alphanumeric strings in the text while non-words are the remaining fragments.

In [17] the syllable is defined as: "a unit of pronunciation having one vowel sound, with or without surrounding consonants, and forming all or part of a word." We do not need to follow this definition strictly. Therefore we will use simplified definition [11]: "Syllable is a sequence of speech sounds, which contains exactly one vowel subsequence." Consequently, the number of syllables in certain word is equal to the number of word's maximum sequences of vowels.

Example 1. For example, the word "wolf" contains one maximal vowel subsequence ("o"), so it is one syllable; while word "ouzel" consists of two maximal vowel sequences - "ou" and "e" – there are two syllables, "ou" and "zel".

N-grams are sequences of n letters, then 3-gram contains 3 letters, 5-gram is created by 5 letters, 1-gram is one letter.

Table 1. Examples of string parsing into words, syllables, letters, 3-grams and 5-grams

parsing type	parsed text elements
orginal	`"consists of"`
letters	`"c", "o", "n", "s", "i", "s", "t", "s", " ", "o", "f"`
3-grams	`"con", "sis", "ts ", "of"`
5-grams	`"consi", "sts o", "f"`
syllables	`"con", "sists", " ", "of"`
words	`"consists", " ", "of"`

3 XBW

We designed an XBW method based on the Burrows-Wheeler transform [3]. This XBW method was partially described in [7]; in this paper, we give a more detailed description.

The XBW method was not named very appropriately, because it can be easily mistaken by name xbw used by the authors of paper [5] for XML transformation into the format more suitable for searching. In another article these authors renamed the transformation from xbw to XBW. Moreover they used it in compression methods called XBzip and XBzipIndex.

XBW method we designed consists of these steps: 1. replacement of the tag names, 2. division into the elements (words, syllables or n-grams), 3. encoding of the dictionary of used elements [10], 4. Burrows-Wheeler transform (BWT), 5. Move to Front transform (MTF), 6. Run Length Encoding of null sequences (RLE), and 7. Canonical Huffman. The steps are all described also by examples. Our tests were performed on plain texts, therefore we will not detail this method with XML support.

3.1 Replacement of the tag names

SAX parser produces a sequence of SAX events processed by a structure coder. This coder builds up two separate dictionaries for elements and attributes of encoding.

If a corresponding dictionary contains the processed element or attribute, it is substituted by an assigned identifier. Otherwise it will be substituted by the lowest available identifier and put into the proper dictionary. Moreover the name of the element or attribute will be written to the output just after the new identifier. It ensures that the dictionaries do not need to be coded explicitly and can be reconstructed during the extraction using the already processed part.

Example 2. Suppose the input

```
<note importance="high">money</note>
<note importance="low">your money</note>
```

Then the dictionaries look like

Element dictionary		Attribute dictionary	
EA	End-of-Tag	E1	importance
A1	note	E2	*empty*

and output is *A1 E1 high EA money EA A1 E1 low EA your money EA*.

3.2 Division into words or syllables

Output of the previous step is divided into words or syllables as described in [7]. The coder then creates a dictionary of basic units (syllables or words). In this phase, the coder creates a syllable (word) dictionary. If the dictionary contains

a processed basic unit, it is substituted by its identifier. Otherwise, it is added to the dictionary and assigned a unique identifier. Then all occurrences in the code are replaced by this identifier. Resulting stream is denoted as S-stream. This part has two outputs: the S-stream and the dictionary of used basic units. The dictionary has to be also encoded because of the document reconstruction.

Example 3. Let us continue in the previous example where the syllables in the dictionary could have the following associations: 01 – high, 02 – mo, 03 – ney, 04 – low, 05 – your. The S-stream is then *A1 E1 01 EA 02 03 EA A1 E1 04 EA 05 06 02 03 EA*.

3.3 The dictionary encoding

The previous step also generates a dictionary of words or syllables which are used in the text. TD3 [10] is one of the most effective dictionary compression methods. It encodes the whole dictionary (represented as a trie) instead of the separate items stored inside.

Trie compression of a dictionary (TD) is based on the trie representing the dictionary coding. Every node of the trie has the following attributes: *represents* — a boolean value stating whether the node represents a string; *count* — the number of sons; *son* — the array of sons; *extension* — the first symbol of an extension for every son.

The latest implementation of TD3 algorithm employs a recursive procedure *EncodeNode3* (Figure 1) traversing the trie by a depth first search (DFS) method. For encoding the whole dictionary we run this procedure on the root of the trie representing the dictionary.

An example is given in Figure 2. The example dictionary contains strings `".\n"`, `"ACM"`, `"AC"`, `"to"`, and `"the"`.

In procedure *EncodeNode3* we code only the number of sons and the distances between the sons' extensions. For non-leaf nodes we must use one bit to encode whether that node represents a dictionary item (e.g. syllable or word) or not. Leafs always represent dictionary items, so it is not necessary to code them. Differences between extensions of sons are defined as distances between `ord` function values of the extending characters. For coding the number of sons and the distances between them we use gamma and delta Elias codes [4]. (We have tested other Elias codes too, but the best results were achieved for the gamma and delta codes). The number of sons and distances between them can reach the value 0 but standard versions of gamma and delta codes start from 1 — it means that these codings do not support value 0. Therefore we use slight modifications of Elias *gamma* and *delta* codes: $gamma_0(x) = gamma(x+1)$ and $delta_0(x) = delta(x+1)$.

Using the `ord` function we reorder the symbols alphabetically according to the symbols' types and their occurrence frequencies typical for a certain language. While the distances between the sons are smaller than distances coded for example by ASCII, they can be represented by shorter Elias delta codes.

```
00 EncodeNode3(node) {
     /* encode the number of sons */
01   output->WriteGamma0(count + 1);
02   if (count = 0) return;
     /* is the node a string? */
03   if (represents)
04     output->WriteBit(1);
05     else output->WriteBit(0);
06   if (IsKnownTypeOfSons)
07     previous = first(TypeOfSymbol(This->Symbol));
08     else previous = 0;
     /* iterate and encode all sons of this node */
09   for(i = 0; i < count; i++) {
       /* count and encode the distances between sons */
10     distance = ord(son[i]->extension) - previous;
11     output->WriteDelta0(distance + 1);
       /* call the procedure on the given son */
12     EncodeNode3(son[i]);
13     previous = ord(son[i]->extension);
14   }
15 }
```

Fig. 1. Procedure EncodeNode3

In our example (Figure 2) the symbols 0–27 are reserved for lower-case letters, 28–53 for upper-case letters, 54–63 for digits and 64–255 for other symbols.

Additional improvement is based on the knowledge of a syllable type that is determined by the first one or two letters of the syllable. If a string begins with a lower-case letter (lower-word or lower-syllable), the following letters must be lower-case too. In a trie every son of a node representing lower-case letter must be lower-case letter as well.

The same situation comes on for other-words and numeric-words: if a string begins with an upper-case letter, we must examine the second symbol to recognize the type of the string (mixed or upper). In our example (Figure 3.3) we know that all sons of nodes `'t'`, `'o'`, `'h'`, and `'e'` are lower-case letters.

In this ordering (described by the function `ord`), each symbol type is given an interval of potential order. Function `first` returns the lowest orders available for each given symbol type.

Each first node (son) has its imaginary left brother having default value 0. If the syllable type is defined, it is possible to set the imaginary brother's value to the corresponding value of `first`. It lowers the distance values (and shortens their codes) of the mentioned nodes.

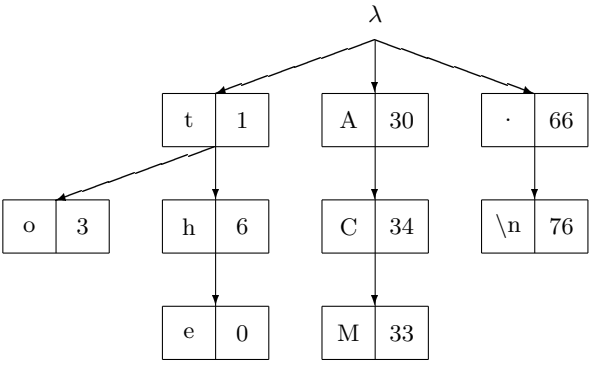


Fig. 2. Example of a dictionary for TD3

Take the node labeled 't' in Figure 2 to describe the coding procedure EncodeNode3. First we encode the number of its sons. The node has two sons therefore we write $gamma_0(2) = 011$. By writing a bit 0 we denote that the dictionary does not containt the processed word (string "t"). The value of the first son of `'t'` is encoded as a distance between its value 3 and zero by $delta_0(3-0) = 01100$. Then the first subtrie of node `'t'` is encoded by a recursive call of the encoding procedure on the first son of the actual node.

Table 2. The output: 8 / E1 06 01 02 02 E1 04 05 EA EA A1 A1 03 03

01	02	03	EA	A1	E1	04	05	06	02	03	EA	A1	**E1**
02	03	EA	A1	E1	01	02	03	EA	A1	E1	04	05	**06**
02	03	EA	A1	E1	04	05	06	02	03	EA	A1	E1	**01**
03	EA	A1	E1	01	02	03	EA	A1	E1	04	05	06	**02**
03	EA	A1	E1	04	05	06	02	03	EA	A1	E1	01	**02**
04	05	06	02	03	EA	A1	E1	01	02	03	EA	A1	**E1**
05	06	02	03	EA	A1	E1	01	02	03	EA	A1	E1	**04**
06	02	03	EA	A1	E1	01	02	03	EA	A1	E1	04	**05**
A1	**E1**	**01**	**02**	**03**	**EA**	**A1**	**E1**	**04**	**05**	**06**	**02**	**03**	**EA**
A1	E1	04	05	06	02	03	EA	A1	E1	01	02	03	**EA**
E1	01	02	03	EA	A1	E1	04	05	06	02	03	EA	**A1**
E1	04	05	06	02	03	EA	A1	E1	01	02	03	EA	**A1**
EA	A1	E1	01	02	03	EA	A1	E1	04	05	06	02	**03**
EA	A1	E1	04	05	06	02	03	EA	A1	E1	01	02	**03**

3.4 Burrows-Wheeler transform

In a BWT step we transform the S-stream into a "better" stream. The "better" stream means achieving some better final compression ratio. Obviously the

transform should be reversible, otherwise we could lose some information. Specifically we achieve a partial grouping of the same input characters. This process requires sorting of all the step input permutations. In this certain prototype we do not use an effective algorithm described in [13] but a simpler C/C++ qsort function. The use of more sophisticated algorithm would lower the compression time but would not affect the compression ratio. We realize that time and spatial complexity is markedly worse as in optimal case. Since in optimal case the order of single elements can be different, we do not mention the time complexity in the text, it would be confusing.

Suppose we have a sorted matrix of all input permutations: the transform output is then composed by its last column and by the column index of input in this matrix (Table 2).

3.5 Move to Front transform

Then the output stream of BWT is transformed by another transformation step – MTF [1]. This step translates text strings into a sequence of numbers. Suppose a numbered list of alphabet elements, then MFT reads these input elements and writes their list order. As soon as the element is processed it is moved up to the front of the list.

Example 4.

```
alphabet: 01 02 03 04 05 06 A1 E1 EA
string: E1 06 01 02 02 E1 04 05 EA EA A1 A1 03 03
output: nothing

alphabet: 03 A1 EA 05 04 E1 02 01 06
string:
output: 76230356808080
```

The output of MTF phase is “76230356808080”.

3.6 Run Length Encoding of null sequences

One MFT step may generate long sequences of zeroes (null sequences). If successful, the RLE step shrinks these null sequences and replaces them with a special symbol representing a null sequence of a given length. Finally the output is a stream of numbers and the special symbols.

Unfortunately, our example does not show a proper use of RLE. Therefore we will alter the problem and replace the string “02 01 00 00 00 03 00 07 00 00” by “02 01 N3 03 00 07 N2”.

3.7 Canonical Huffman

After the RLE step the stream is encoded by a canonical Huffman code [12]. Huffman coding is based on assigning shorter codes to more frequent characters then to characters with less frequent appearance. In our example, “76230356808080” will have assigned the following codes:

Example 5.
`0 - 00, 2 - 110, 3 - 010, 5 - 1110, 6 - 011, 7 - 1111, 8 - 10`
The output is then: 1111 011 110 010 00 010 1110 011 10 00 10 00 10 00.

4 Experiments

We compared the compression effectivity using BWT of letters, syllables, words, 3-grams and 5-grams. Testing procedures proceeded on commonly used text files of different sizes (1 kB - 5 MB) in various languages: English (EN), Czech (CZ), and German (GE).

4.1 Testing data

Each of tested languages (EN, CZ, GE) had its own plain text testing data. Testing set for Czech language contained 1000 random news articles selected from PDT [20] and 69 books from eKnihy [15]. Testing set for English contained 1000 random juridical documents from [19] and 1094 books from Gutenberg project [16]. For German, we used 184 news articles from Sueddeutche [18] and 175 books from Gutenberg project [16].

4.2 Results

The primary goal was to compare the letter-based compression, syllables and words. For files sized up to 200 kB, the letter-based compression appears to be optimal; for files 200 kB - 5 MB syllable-based compression is the most effective. Also used language affects the results. English has a simple morphology: in large documents the difference between words and syllables is insignificant. In languages with rich morphology (Czech, German) words are still about 10% worse then syllables, even on the largest documents. Language type influence on compression is detailed in [11].

As the second aim we tried to compare the syllable-based compression with 3-gram-based and word-based compression with 5-gram-based compression. Syllables as well as words are natural language units therefore we supposed using it to be more effective than using 3-grams and 5-grams. These assumptions were confirmed. Natural units were the most effective for small documents (20 - 30 %), by increasing the document size efficiency falls to 10 - 15 % for documents of size 2 - 5 MB.

5 Conclusion

In this paper, we compare experimentally the compression efficiency of Burrows-Wheeler transform by letters, syllables, words, 3-grams and 5-grams, using the XBW compression algorithm. We test common plain text files of different sizes (1kB – 5MB) in various languages (English, Czech, German). BWT block contains the whole document.

Table 3. Comparision of different input parsing strategies for Burrows-Wheeler transform. Values are in bits per character.

—	File size	100 B	1 kB	10 kB	50 kB	200 kB	500 kB	2 MB
Lang.	Method	1 kB	10 kB	50 kB	200 kB	500 kB	2MB	5 MB
CZ	Symbols	**5.715**	**4.346**	**3.512**	**3.200**	**2.998**	2.846	—
CZ	Syllables	6.712	4.996	3.765	3.280	3.003	**2.825**	—
CZ	Words	7.751	6.111	4.629	3.871	3.476	3.149	—
CZ	3-grams	8.539	6.432	4.629	3.851	3.463	3.166	—
CZ	5-grams	10.104	8.415	6.566	5.448	4.796	4.265	—
EN	Symbols	**5.042**	**3.018**	**2.552**	**2.647**	2.513	2.336	2.066
EN	Syllables	5.974	3.267	2.647	2.685	**2.486**	**2.282**	**1.996**
EN	Words	6.323	3.651	2.969	2.944	2.668	2.382	2.014
EN	3-grams	7.740	4.571	3.421	3.148	2.823	2.530	2.136
EN	5-grams	9.358	6.293	4.877	4.367	3.769	3.246	2.530
GE	Symbols	**4.545**	**3.853**	**2.914**	**2.629**	**2.491**	2.323	2.416
GE	Syllables	5.591	4.671	3.201	2.724	2.505	**2.295**	**2.354**
GE	Words	6.343	5.491	3.679	3.119	2.865	2.545	2.608
GE	3-grams	6.813	5.583	3.760	3.117	2.820	2.525	2.519
GE	5-grams	8.545	7.429	5.324	4.320	3.744	3.237	3.004

Comparing the letter-based, syllable-based and word-based compression, we find out that character-based compression is most suitable for small files (up to 200 kB) and syllable-based compression for files of size 200 kB – 5 MB (see Table 3, the compress ratio in the table is presented in bites per byte (bpc)).

The compression using natural text units like words or syllables is 10–30% better than compression with 5-grams and 3-grams.

To achieve the results introduced in this article, we use compression algorithm XBW implemented mainly for testing purposes. Our next goal is to improve this program for practical use, e.g. time complexity advance (faster sorting algorithm in BWT [13], splay trees for MTF [2]) or UTF-8 encoding support.

References

1. Arnavut, Z.: Move-to-front and inversion coding. Data Compression Conference, IEEE CS Press, Los Alamitos, CA, USA (2000) 193–202.
2. Bentley, J. L., Sleator, D. D., Tarjan, R. E., Wei, V. K.: A locally adaptive data compression scheme. Communications of the ACM, 29(4):320-330, April 1986.
3. Burrows, M., Wheeler, D. J.: A Block Sorting Loseless Data Compression Algorithm. Technical report, Digital Equipment Corporation, Palo Alto, CA, U.S.A (2003).
4. Elias, P.: Universal codeword sets and representation of the integers. IEEE Trans. on Information Theory, Vol. 21, (1975) 194–203.
5. Ferragina, P., Luccio, F., Manzini, G., Muthukrishnan, S.: Structuring labeled trees for optimal succinctness, and beyond. Proc. 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05), (2005) 184-193.
6. Ferragina, Manzini, G., Muthukrishnan, S.: Compressing and Searching XML Data Via Two Zips. Proc. WWW 2006, Edinburgh, Scotland. (2006) 751–760.

7. Galambos, L., Lansky, J., Chernik, K.: Compression of Semistructured Documents. In: International Enformatika Conference IEC 2006, Enformatika, Transactions on Engineering, Computing and Technology, Volume 14, (2006) 222–227
8. Isal, R.Y.K., Moffat, A.: Parsing Strategies for BWT Compression. Data Compression Conference, IEEE CS Press, Los Alamitos, CA, USA (2001) 429-438.
9. Isal, R.Y.K., Moffat, A.: Word-based Block-sorting Text Compression. Proc. 24th Australasian Computer Science Conference, Gold Coast, Australia, (2001) 92–99
10. Lansky, J., Zemlicka, M.: Compression of a Dictionary. In: Snasel, V., Richta, K., and Pokorny, J.: Proceedings of the Dateso 2006 Annual International Workshop on DAtabases, TExts, Specifications and Objects. CEUR-WS, Vol. 176, (2006) 11-20
11. Lansky, J., Zemlicka, M.: Text Compression: Syllables. In: Richta, K., Snasel, V., Pokorny, J.: Proceedings of the Dateso 2005 Annual International Workshop on DAtabases, TExts, Specifications and Objects. CEUR-WS, Vol. 129, (2005) 32–45
12. Moffat, A., Turpin, A.: On the implementation of minimum redundancy prefix codes. IEEE Trans.Comm. 45(1997), 1200–1207
13. Seward, J.: On the Performance of BWT Sorting Algorithms. DCC, IEEE CS Press, Los Alamitos, CA, USA (2000) 173.
14. Seward, J.: The bzip2 and libbzip2 official home page. `http://sources.redhat.com/bzip2/`
15. e-books. `http://go.to/eknihy`
16. Project Gutenberg. `http://www.promo.net/pg`
17. Compact Oxford English Dictionary. `http://www.askoxford.com`
18. Sueddeutsche. `http://www.sueddeutsche.de`
19. California Law. `http://www.leginfo.ca.gov/calaw.html`
20. The Prague Dependency Treebank. `http://ufal.mff.cuni.cz/pdt/`

Updating Typed XML Documents Using a Functional Data Model

Pavel Loupal

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

Abstract. We address a problem of updating XML documents having their XML schema described by a Document Type Definition (DTD) without breaking their validity. We present a way how to express constructs available in DTD in a functional data model and propose algorithms for performing insert, update and delete operations. After that we embed the update capability into an existing query language for XML. This paper thus outlines the whole "life cycle" of the approach from the problem analysis to its implementation.

1 Motivation and Problem Statement

During our work on a functional framework for querying XML – XML-λ – we identified a need for extending the language with support of data modification operations. Our aim is to develop an approach similar to the SQL language for relational databases, i.e. have an ability both to query and update underlying data.

With respect to our aim we set up basic requirements for our approach. First, we always consider *typed* data (this is a natural requirement because of the fact that our framework is based on a type system). At this stage we use DTD for constraining document validity. Second, we have already a query language designed. It makes sense to extend this language in a "logical" way with update operations. By the term "logical" we mean the utilization of existing constructs as sets, existing type system and the idea of functional approach in general.

The paper is structured as follows: Section 2 lists existing approaches for updating XML data and discusses their contribution. In Section 3 we briefly outline the concept of the functional framework we use, its data model and the query language that is used for implementing the proposal. We discuss the problem in Section 4 where we show our solution. Section 5 deals with enriching the syntax of our query language with update operations. In Section 6 we conclude with ideas for future work.

© J. Pokorný, V. Snášel, K. Richta (Eds.): Dateso 2007, pp. 11–20, ISBN 80-7378-002-X.
VŠB – Technical University of Ostrava, Dept. of Computer Science, 2007.

2 Languages for Updating XML

By the term *updating XML* we mean the ability of a language to perform modifications (i.e. insert, update and delete operations, etc.) over a set of XML documents.

Since the creation of the XML in 1998 there have been many efforts to develop various data models and query languages. A lot of time has also been spent on indexing and query optimization. On the other hand the problem of updating XML gains more interest in few past years. Yet there seems to be not a complete solution for this problem.

Existing papers dealing with updating XML are mostly related to XQuery [2] (and the need for having updates in XQuery is also considered as one of the most important topics in the further development of the language [4]). Lehti [7] proposes an extension to XQuery that allows all update operations but does not care about the validity of the documents. Tatarinov, et al. [11] also extends XQuery syntax with insert, update and delete operations and shows the implementation of storage in a relational database system. Benedikt, et al. [1, 10] deals in deep with the semantics of updates in XQuery.

For the sake of completeness we should not omit XUpdate [6] – a relatively old proposal that takes a different way. It uses XML-based syntax for describing update operations. This specification is probably less formal than those previous but it is often used in praxis.

Considering previous works we can deduce that there are common types of operations for performing modifications that are embedded in a language – delete, update, insert before or insert after. This seems to be a sufficient base for ongoing work. None of those proposals but deals with the problem of updating *typed data* and thus it makes sense to put some effort into studying of this problem. The evolution process around XML leads to use of types so it makes sense to work on this problem in the world of typed XML documents.

3 XML-λ Framework

XML-λ is a proposal published in 2001 by Pokorný [8, 9]. In contrast to W3C languages it uses *functional* data model instead of tree- or graph-oriented model. The primary motivation was to see XML documents as a database that conforms to some XML schema (defined, for example, by DTD).

The framework is based on type system theory – it can be informally said that first a "base" type system T_{base} is defined then a regular type system T_{reg} that extends T_{base} with regular types is induced. Upon this the T_{reg} is enriched with types corresponding to an XML schema and the T_E type system is defined. Over such type system we define a query (and update) language based on simply typed lambda calculus.

3.1 Type System Introduction

Type system is built on base $\mathcal{B}$ – a set containing finite number of base types $S_1, \ldots, S_k$ $(k \geq 1)$. Type hierarchy is then created by following inductive definition:

Definition 1. *Let $\mathcal{B}$ is a set of primitive types $S_1 \ldots S_n, k \geq 1$.* Type System *T_{base} over base $\mathcal{B}$ is the least set containing types given by 1.-4.*

1. base type: *each member of $\mathcal{B}$ is type over $\mathcal{B}$*
2. functional type: *if T_1 and T_2 are types over $\mathcal{B}$, then $(T_1 \rightarrow T_2)$ is also a type over $\mathcal{B}$*
3. n-tuple type*: if $T_1, \ldots, T_n$ $(n \geq 1)$ are types over $\mathcal{B}$, then $(T_1, \ldots, T_n)$ is type over $\mathcal{B}$*
4. union type: *if $T_1, \ldots, T_n$ $(n \geq 1)$ are types over $\mathcal{B}$, then $(T_1 + \ldots + T_n)$ is type over $\mathcal{B}$*

Subsequently we define a regular type system T_{reg} that extends type system T_{base} with regular constructs:

Definition 2. *Let $\mathcal{B}$= $\{String, Bool\}$, let $NAME$ be a set of names. Type System T_{reg} is the least set containing types given by 1.-6.*

1. *Every member of the base $\mathcal{B}$ is an (*primitive*) type over $\mathcal{B}$.*
2. named character data: *Let $tag \in NAME$. Then tag : $String$ is an (*elementary*) type over $\mathcal{B}$,*
   *tag : is an (*empty elementary*) type over $\mathcal{B}$.*
3. *Let T be a group type or named character data. Then*
   - zero or more: *$T*$ is a type over $\mathcal{B}$.*
   - one or more: *$T+$ is a type over $\mathcal{B}$.*
   - zero or one: *$T?$ is a type over $\mathcal{B}$.*
4. alternative: *Let T_1 and T_2 be types. Then $(T_1|T_2)$ is a type over $\mathcal{B}$.*
5. sequence: *Let $T_1, \ldots, T_n$ be types. Then $(T_1, \ldots, T_n)$ is a type over $\mathcal{B}$.*
6. named type: *Let T be a type given by a step from 3.-5. Let $tag \in NAME$. Then $tag : T$ is a type over $\mathcal{B}$.*

3.2 Binding Types to XML

Having the T_{reg} type system we have to extend it to be able to work with XML data. We build the type system T_E induced by T_{reg}. Key idea is to define *abstract items* that are particular XML elements or attributes with some content and also define a set containing all abstract items within an XML instance – **E**.

Definition 3. *Let T_{reg} over base $\mathcal{B}$ be a type system from definition 2 and E is the set of abstract items. Then* type system *T_E induced by T_{reg} is the least set containing type given by this rule:*

Let $tag : T \in T_{reg}$. Then $TAG : T$ is a member of T_E. (Replacement of all tags in $tag : T$ by uppercase version)

With types from T_E we can consider functional types for extracting data values from elements (via abstractions and projections) with two ways

1. for *simple element:* if $tag : String \in T_{reg}$, then $(\mathbf{E} \rightarrow tag : String) \in T_E$
2. for *compound element:* if $tag : T \in T_{reg}$, then $(\mathbf{E} \rightarrow T') \in T_E$

Note also that in T_E we can express attributes in the same way as XML elements – as functions.

3.3 Query Language Construction

Typical query has the query part – an expression to be evaluated over data – and the constructor part that wraps query result and forms the XML output. XML-λ's query language is based on λ-terms defined over the type system T_E as shown in Definition 4.

Main constructs of the language are *variables*, *constants*, *tuples*, use of *projections* and λ-calculus operations – *applications* and *abstractions*. Tagged terms might be used for declaring functions. Syntax of this language is similar to λ-calculus expression i.e. $\lambda \ldots (\lambda \ldots (expression) \ldots)$. In addition, there are also typical constructs such as logical connectives, constants or comparison predicates.

Language of terms is inductively defined as the least set containing all terms created by application of following rules:

Definition 4. *Let $T, T_1, \ldots, T_n, n \geq 1$ be members of T_{base}. Then*

1. variable: *each variable of type T is a term of type T*
2. constant: *each constant (member of $\mathcal{F}$) of type T is a term of type T*
3. application: *if M is a term of type $((T_1, \ldots, T_n) \rightarrow T)$ and $N_1, \ldots, N_n$ are (in the same order) types $T_1, \ldots, T_n$, then $M(N_1, \ldots, N_n)$ is a term of type T*
4. λ-abstraction: *if $x_1, \ldots, x_n$ are distinct variables of types $T_1, \ldots, T_n$ and M is a term of type T, then $\lambda x_1, \ldots, x_n(M)$ is a term of type $((T_1, \ldots, T_n) \rightarrow T)$*
5. n-tuple: *if $N_1, \ldots, N_n$ are terms of types $T_1, \ldots, T_n$, then $(N_1, \ldots, N_n)$ is a term of type $(T_1, \ldots, T_n)$*
6. projection: *if $(N_1, \ldots, N_n)$ is a term of type $(T_1, \ldots, T_n)$, then $N_1, \ldots, N_n$ are terms of types $T_1, \ldots, T_n$*
7. tagged term: *if N is a term of type $NAME$ and M is a term of type T then $N : T$ is a term of type $(E \rightarrow T)$.*

3.4 Query Example

For our purposes we use the notoriously known bibliography example DTD from the XML Query Use Cases [5] document. We also consider XML data provided in the same document.

A query returning all books published by "Addison-Wesley" is in XML-λ expressed as shown in Figure 1.

```
xmldata("bib.xml")
lambda b ( /book(b) and  b/publisher = "Addison-Wesley" )
```

Fig. 1. An example query written in XML-λ

3.5 Data Model Summary

Previous sections outline the definition of type system T_E that we use for modelling types in an XML schema. This means that for each DTD we can construct a particular type system of respective types. In the *Language of terms* we propose a mechanism based on lambda calculus operations (applications and abstractions) combined with projections to work with XML documents.

The most important idea in the framework is the fact that even the smallest piece of information in an XML document (e.g. an attribute of element containing just a PCDATA value) is modelled as a partial function that assigns a value for exactly one $e \in \mathbf{E}$. For example, having an XML element `<phone>+420-800123456</phone>` there is a function `phone(e)` that for exactly one $e \in \mathbf{E}$ returns value `+420-800123456`. For more complex types, e.g. `<!ELEMENT author (last, first)>` the result of the function is a Cartesian product $\mathbf{E} \times \mathbf{E}$.

In XML-λ we model each XML document by a *set of items* $\mathbf{E}$ where each $e \in \mathbf{E}$ is of type T_{ITEM}. T_{ITEM} is a type consisting of a couple $(t : TYPE, uid : INT)$; $TYPE \in T$ and *uid* is an integer value for maintaining order of items in the set. Note that some types in a particular type system can have related information attached (each item of type PCDATA has attached a value of the item – its content).

In following text we consider following semantic functions with informal meaning as summarized in following table:

Semantic Function	Behaviour
parent(e)	For an $e \in \mathbf{E}$ return its parent item
type(e)	For an $e \in \mathbf{E}$ return its type t ($t \in \mathbf{T}$)
application(e, t)	Executes an application (rule 3 in Definition 4) of t-object to the e item. In general it returns a Cartesian product of $\mathbf{E} \times \ldots \times \mathbf{E}$
projection(n-tuple, t)	Retrieves all items of type t from given n-tuple.
childTypes(t)	Retrieves an list of types (sorted by document order) that might be contained in the result of application of a t-object

For further usage we present an algorithm of traversing a fragment of XML data utilizing our functional framework. The algorithm *traverse(***E***, e, op)* takes three parameters, **E** – set of items (this represents an XML document in our

model) and e – start-up item for traversing, *op* – an operation to be performed on each node.

```
ALGORITHM traverse(E, e, op)
1: Initialize stack S;
2: Mark e as NEW; Push e to stack S;
3: while (any NEW or OPEN node in S)
4:   Pop i from S; Mark i as OPEN;
5:   Type t = type(i);
6:   n-tuple nt = application(i, t);
7:   List_of_types lt = childTypes(t);
8:   if lt is String
9:     op(i);
10:    Mark i as CLOSED;
     else
11:    For each type in lt
12:     n-tuple nt = application(i, type);
13:     Mark all items as NEW and push to S;
```

4 Updating Typed Documents

Document Type Definition (DTD) [3] is a syntactic way how to describe a *valid* XML instance. We can break all DTD features into disjoint categories:

1. Elements constraints – Specify the type of an element content. Is one of `EMPTY`, `ANY`, `MIXED` or `ELEMENT_CONTENT`,
2. Structure constraints – The occurrence of elements in a content model. Options are *exactly-one*, *zero-or-one*, *zero-or-more*, *one-or-more*
3. Attributes constraints – `#REQUIRED`, `#IMPLIED`, `#FIXED`, ID, IDREF(S)

Each update operation can or cannot affect any construct from the particular DTD. Considering a transactional behaviour we can see two violation scenarios:

1. *Fully consistent.* After each operation (insert, update or delete) the instance remains valid. This means that we have to define a complete set of operations that are strong enough to perform all possible updates.
2. *Partially consistent.* In this mode we allow partial inconsistency i.e. we consider the whole query as an atomic operation. Therefore we do not require to have atomic insert,update and delete operations but we have to ensure that at the end of the processing the instance is valid. In general it means revalidation of the document being updated.

In our approach we use the first scenario and declare all operations as *fully consistent*.

With respect to abilities of the existing XML-λ framework we have to extend this framework with features allowing us to check constraints available in DTD.

The cornerstone of the framework is its type system (it is the basis of types we can use). For modelling DTD constraints we propose four *sets of types*, where all types come from the type system, i.e. $T \in T_E$.

1. $T_{unmodifable}$ is a set of types that cannot be modified. This set contains types for attributes declared as `#FIXED` and element types with `EMPTY` content model.
2. $T_{mandatory}$ is a set of types that must not be removed from a document instance because it would break the DTD constraints. This set contains attribute types with `#REQUIRED` declaration and element types for those T iff all occurrences of T are exactly-one.
3. $T_{referencing}$ is a set of types that may reference another type, for DTDs those are attributes declared as `IDREF` or `IDREFS`.
4. $T_{referenced}$ is a set of types that may be referenced by another type, for DTDs those are attributes declared as `ID`.

These sets we use in our semantics for particular update operations. We will use access functions *isUnmodifable(e)*, *isMandatory(e)*, *isReferencing(e)* and *isReferenced(e)* that check the containment of item's type in respective sets.

In general the semantics of all operations consists of two parts: (1) Check if the operation is permitted regarding the DTD constraints and (2) Execution of given update operation. Following sections discuss the semantics of delete, insert and update operations in detail.

4.1 Delete

Deletion is a operation of removing given part of XML data (i.e. element or attribute) with its potential subelement(s). We can see the function with a signature `DELETE(`e`)` where $e : \mathbf{t} \in \mathbf{E}$, $\mathbf{t} \in T$.

With respect to validity issues there are two scenarios where this operation is denied:

1. `e` is an *attribute* and is declared as `#REQUIRED` or `#FIXED`
2. `e` is an *element* with *exactly-one* or *one-or-more* occurrence

In our framework it means checking whether the type of item being deleted is a member of $T_{mandatory}$ or $T_{unmodifiable}$ sets. After that we have to ensure that by deleting of the item we do not delete the last remaining item with *exactly-one* or *one-or-more* occurrence. Following algorithm outlines conceptually the operation.

Algorithm *delete(***E***, e)* takes two parameters, **E**– set of items (this represents an XML document in our model) and e – the item to be deleted. It returns *true* if the item has been deleted or *false* if the delete has been denied.

```
ALGORITHM delete(E, e):
1: if (isMandatory(e) or isUnmodifieble(e))
2:    return false;
```

```
   else
3:    p = parent(e);
4:    List_of_types pt = childrenTypes(p);
5:    if (type(e) in pt) is exactly-one
6:       return false;
7:    if (type(e) in pt) is one-or-more and
8:        data contains at least 1 occurrence of item with type(e)
9:        return false;
      else
10       traverse(E, e, delete);
11:      return true;
```

Informally we can imagine the operation as a subset subtraction $\mathbf{E_{result}} = \mathbf{E} \setminus e$ with ongoing renumbering of remaining items to keep document order. Maintenance of potential references in document (attributes of type ID, IDREF and IDREFS) that should take place in the `traverse` function. For now we consider it as being out scope of this paper.

4.2 Insert

By the insert operation we mean adding an XML fragment into the target document. By the fragment we understand an element, an attribute or an XML subtree. Insert is more complicated operation than delete because there are more conditions and restrictions to be checked. We write the statement as

`INSERT` e_1 (`AFTER` | `BEFORE` | `AS CHILD`) e_2`;`

where $e_1 : T_1, e_2 : T_2 \in \mathbf{E}$; T_1, T_2 are types. Note that e_1 must be a valid expression in T, i.e. it must be of a type from T_E. We can see the operations as $f_{insert}(F, \mathbf{E_1}, \mathbf{E_i}) = \mathbf{E_{result}}$ where $F \in T_E$ and $\mathbf{E_{result}} = \mathbf{E_1} \cup \mathbf{E_i}$

Algorithm *insert_after(***E***,* e_1*,* e_2*)* takes three parameters, $\mathbf{E}$ – set of items, e_1 – the item to be inserted ($e_1 \notin \mathbf{E}$) and e_2 – the context item. It returns *true* if the item has been inserted or *false* if the insert has been denied.

```
ALGORITHM insert_after (E, e1, e2):
1: p = parent(e1);
2: List_of_types pt = childrenTypes(p);
3: if (type(e1) not in pt)
4:    return false;
5: if (following type of e2) is exactly-one and
6:     item with the same type already exists in n-tuple
7:    return false;
8: Put item into E and perform uid renumbering;
9: return true;
```

4.3 Update

Update operation means replacing one item by another with the same type. We can write the signature of this operation as `UPDATE` e_1 `WITH` e_2`;` where $e_1 : T_1, e_2 : T_2 \in \mathbf{E}$; $T_1 = T_2$ are types.

Because of the fact that we require both expressions to be of the same type there cannot occur any validation conflict (both of them are valid before the operation). The algorithm for performing update is then straightforward.

5 Implanting Updates into the Language

First implementation of the XML-λ language was developed in [12]. It is basically a query language without any updating capability. We will extend this language with operations as shown above.

With respect to the concept of the language we declare all operations as *tagged terms*. This means that we consider each function as a term of functional type $(\mathbf{E} \rightarrow T)$. Consequently we define the semantics for all operations.

Following fragment of EBNF shows a concept of including all operations into the language

[1] *Query* ::= *Options* (*OpUpdateList* | *OpQuery*)

[2] *OpQuery* ::= *ConstructorPart QueryBody* Eof

[3] *OpUpdateList* ::= { *OpUpdate* }+

[4] *OpUpdate* ::= { `Delete` *SubQuery* |
`Insert` *Expr* (`after|before|as child of`) *SubQuery* |
`Update` *SubQuery* `With` *Expr* }

Note that the non-terminal *SubQuery* (rule `[18]` in [12, p.55]) presents a lambda term that may return set of items. There is also a significant advantage of using non-terminals *SubQuery* and *Expr*. In general these can be function calls (even user-defined). This is a difference with XUpdate, where only XPath expressions are used.

An example of delete operation removing all books published by "Addison-Wesley" is then written as follows

```
xmldata("bib.xml")
delete( lambda b ( /book(b) and  b/publisher = "Addison-Wesley"))
```

The insert operation that adds a new element `author` after the first author of a book specified by its name we write as

```
xmldata("bib.xml")
insert-after( lambda a (
  /book(b) and  b/title="TCP/IP Illustrated" and a=b/author[1]),
  "<author><last>Richta</last><first>Karel</first></author>"))
```

An example of the delete operation is obvious and is omitted.

6 Conclusion and Future Work

We have shown a proposal for updating XML data constrained by a Document Type Definition. We present a functional framework for querying XML that is extended by structures for expressing DTD semantics. By enriching the query language with modification operations – inserts, deletes and updates – we obtain a language suitable both for querying and updating XML documents.

There is still a lot of future work ahead. To get a complete framework we have to finalize the issue with references within documents (IDs and IDREFS). This is only a technical problem how to formalize the algorithm to be executed to keep the documents consistent and valid. Another questionable area are the dependencies of multiple update operations in one "query" statement. In this paper we do not solve any potential conflicts.

Probably the biggest challenge for future work is replacement of DTD by XML Schema. This means restructuring the type systems T_{reg} and T_E and re-developing the idea of constraint sets.

References

1. M. Benedikt, A. Bonifati, S. Flesca, and A. Vyas. Adding updates to XQuery: Semantics, optimization, and static analysis. In *XIME-P 2005*, 2005.
2. S. Boag, D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, and J. Simeon. XQuery 1.0: An XML Query Language, September 2005. http://www.w3.org/TR/xquery/.
3. T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible markup language (XML) 1.0 (third edition), February 2004. http://www.w3.org/TR/2004/REC-xml-20040204/.
4. D. Chamberlin. XQuery: Where do we go from here? In *XIME-P 2006*, 2006.
5. D. Chamberlin, P. Fankhauser, D. Florescu, M. Marchiori, and J. Robie. XML Query Use Cases, September 2005. http://www.w3.org/TR/2005/WD-xquery-use-cases-20050915/.
6. A. Laux and L. Martin. XUpdate – XML Update Language, 2000. available online at http://xmldb-org.sourceforge.net/xupdate/index.html.
7. P. Lehti. Design and implementation of a data manipulation processor for an XML query language. Master's thesis, Technische Universitaet Darmstadt, 2001.
8. J. Pokorný. XML functionally. In B. C. Desai, Y. Kioki, and M. Toyama, editors, *Proceedings of IDEAS2000*, pages 266–274. IEEE Comp. Society, 2000.
9. J. Pokorný. XML-λ: an extendible framework for manipulating XML data. In *Proceedings of BIS 2002*, pages 160–168, Poznan, 2002.
10. G. M. Sur, J. Hammer, and J. Siméon. An XQuery-Based Language for Processing Updates in XML. In *PLAN-X 2004*, 2004.
11. I. Tatarinov, Z. G. Ives, A. Y. Halevy, and D. S. Weld. Updating XML. In *ACM SIGMOD 2001*, 2001.
12. P. Šárek. Implementation of the XML lambda language. Master's thesis, Dept. of Software Engineering, Charles University, Prague, 2002.

Genetic Algorithms in Syllable-Based Text Compression*

Tomáš Kuthan and Jan Lánský

Charles University, Faculty of Mathematics and Physics
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
tkuthan@gmail.com, zizelevak@gmail.com

Abstract. Syllable based text compression is a new approach to compression by symbols. In this concept syllables are used as the compression symbols instead of the more common characters or words. This new technique has proven itself worthy especially on short to middle-length text files. The effectiveness of the compression is greatly affected by the quality of dictionaries of syllables characteristic for the certain language. These dictionaries are usually created with a straight-forward analysis of text corpora. In this paper we would like to introduce an other way of obtaining these dictionaries – using genetic algorithm. We believe, that dictionaries built this way, may help us lower the compress ratio. We will measure this effect on a set of Czech and English texts.

1 Introduction

In the early times of the computer age memory and storage capacity were highly limited and extremely expensive. This brought great pressure on storing the data as dense as possible and therefore created ideal conditions for data compression. But due to the fascinating development of computers, we have been witnessing for several last decades, storage capacity grew rapidly while it's price went down in a similar manner. Common hard disk drive of today's PC could easily carry all the code of all the computers in the early 70's. It could seem, that in this situation there is no need for effective compression. But together with the sources the demand raised as well. The amount of data companies deal with and want to or need to archive is unimaginable. Every percentage spared has it's immediate value in money. Another good example is networking. The dynamics of the growth of the network capacity does not even resemble the numbers we are used to by storage. It is very reasonable to transport data compressed to save the bandwidth.

Now that we explained the importance of data compression, we will try to specify the structure of the the coded files. Generally we recognize two types of files - binary and text files. There are many algorithms for binary compression including the whole are of lossy compression, but this concern is beyond the area

* This research was partially supported by the Program "Information Society" under project 1ET100300419.

of interest of this paper. The structure of a text file depends on it's language. We may assume, that two documents in the same language have similar structure. Two different languages may have several similar characteristics. We may ask, whether both languages have rich morphology, or whether they for example both have fixed word-order. In languages with rich morphology the words usually consist of several syllables. Here syllables play the role of natural transition between letters and words.

The size of the file is an other aspect. Experience shows us, that character-based compression is more successful with smaller files, while word-based compression works better with larger files. Syllable-based methods have shown good results when used on middle-size to small documents.

This can be quite important with the already mentioned networking. File sizes of most common network content - *html pages* - are rather smaller. This creates ideal circumstances for syllable based compression methods.

Another important aspect of syllable based compression are the dictionaries of frequent syllables. These are used to initialize the compress algorithms data structures and greatly influence their effectiveness, especially in the early phase of the compression. Later the effect of the already processed part of input dominates over the effect of the initial settings. Therefore the role of dictionaries is vital with smaller files and slightly looses it's importance on bigger files. But, as mentioned above, it is the area of small to middle-size files, where the syllable-based methods are targeted.

Building an optimal dictionary is not an easy task. Including too many rare syllables into the dictionary results in longer bit-codes of the more common ones and hence longer coded message. On the other hand not including a frequent syllable means, that it would have to be coded by symbols (which is expensive) and initialised with low frequency and accordingly longer bit-code. The number of unique syllables in one language is measured in tens of thousands and every single one of them can be either included or excluded. That brings us in front of a problem of finding optimal solution among 2^N candidates, where N is the number of unique syllables. Genetic algorithm is a search technique, which can be employed for exploring such big, highly non-linear spaces.

2 Syllable based approach to text compression

In his study from 2005 [9], Jan Lánský has introduced a new approach to compression by symbols, the syllable-based compression. This new concept led to designing two new algorithms, which had good results in practical use and under certain circumstances even outperformed such sophisticated programs as bzip2. This chapter is dedicated to presenting this technique.

2.1 Languages and syllables

Knowing and understanding the structure of coded message can be very helpful in designing new compression method. In our case the coded message is a text

in natural language. It's structure is determined by the characteristic of the particular language. One linguistic aspect is the morphology. Languages with richer morphology (Czech, German, Turkish) tend to creating new words and word-forms by concatenating the root of the word with one or several prefixes or suffixes. On the other hand in languages like English the same effect is achieved by accumulating words. In the first category of languages we may find (thanks to their agglutinative nature) many rather long words composed of higher number of syllables. Such words are not very common in English. We can expect, that syllable-based compression will give better results on the first group of languages.

What is actually a syllable? Usually it is presented as a phonetic phenomenon. American Heritage Dictionary [11] gives us the following definition: 'A unit of spoken language consisting of a single uninterrupted sound formed by a vowel, diphthong, or syllabic consonant alone, or by any of these sounds preceded, followed, or surrounded by one or more consonants.' Correct *Hyphenation* (decomposition of a word into syllables) is a highly non-trivial issue. It can depend on he etymology of the certain word and there can be several different ways, how to do it. Fortunately we do not have to decompose the word always correctly according to the linguistic rules. We have to decompose it into substrings, which appear relatively often in the language. At this purpose we can get by with the following definition: 'Syllable is a sequence of sounds containing exactly one maximal subsequence of vowels'[1].

We recognize five basic categories of syllables: *capital* (consist of upper-case letters), *small* (lower-case letters), *mixed* (first letter is upper-case, other lower-case), *numeric* (numeral characters) and *other* (characters other than letters and numbers). Capital, small and mixed syllables altogether are called *literal* syllables, while numeral and other are called *non-literal*.

2.2 Hyphenation algorithms

To perform syllable-based compression, we need a procedure for decomposition into syllables. We will call an algorithm *hyphenation algorithm* if, whenever given a word of a language, it returns it's decomposition into syllables. According to our definition of syllable every two different hyphenation of the same word always contain the same number of syllables. There can be an algorithm, that works as a hyphenation algorithm for every language. Then it is called *universal hyphenation algorithm*. Otherwise we call it specific *hyphenation algorithm*.

We will describe four universal hyphenation algorithms: universal left P_{UL}, universal right P_{UR}, universal middle-left P_{UML} and universal middle-rightP_{UMR}.

The first phase of all these algorithms is the same. Firstly, we decompose the given text into words and for each word mark it's consonants and vowels. Then we determine all the maximal subsequences of vowel. These blocks form the ground of the syllables. All the consonants before the first block belong to the first syllable and those behind the last block will belong to the last syllable.

[1] for formal definitions concerning languages and syllables see [9]

Our algorithms differ in the way they redistribute the inner groups of consonants between the two adjusting vowel blocks. P_{UL} puts all the consonants to the preceding block and P_{UR} puts them all to the subsequent block. P_{UML} and P_{UMR} try to redistribute the consonant block equally. If their number is odd P_{UML} pushes the bigger partity to the left, while P_{UMR} to the right. The only exception is, when P_{UML} deals with an one-element group of consonants. It puts the only consonant to the right to avoid creation of not so common syllables beginning with a vowel.

Example 1. Hyphenating *priesthood*

correct hyphenation	priest-hood
universal left P_{UL}	priesth-ood
universal right P_{UR}	prie-sthood
universal middle-left P_{UML}	priest-hood
universal middle-right P_{UMR}	pries-thood

We have measured the effectiveness of these algorithms. In general, P_{UL} was the worst one; it had lowest number of correct hyphenations and produced largest sets of unique syllables. The main reason for this was, that it generates a lot of vowel-started syllables, which are not very common. P_{UR} was better but the most successful were both 'middle' versions. English documents were best hyphenated by P_{UMR}, while with Czech texts P_{UML} was slightly better.

In the following few paragraphs we will describe two syllable-based compression methods

2.3 LZWL

LZWL is a syllable version of well-known LZW algorithm [14]. The algorithm uses a dictionary of phrases, which is implemented by a data structure called *trie*. Each phrase is assigned an ordinal number according to time, when it was inserted into the dictionary.

During initialization this structure is filled with small syllables from the dictionary of frequent syllables. In each step we identify the maximal syllable chain, that forms a phrase from dictionary and at the same time matches a prefix of the not yet processed part of input. The number of the phrase (or better to say it's binary representation) is printed on the output. It could happen, that this maximal chain equals empty syllable. This would mean, that there is a new syllable on the input and we would have to encode it character by character.

Before performing the next step we add a new phrase into the dictionary. This new phrase is constructed by concatenating the phrase from last step with the first syllable of the current phrase.

For more information on LZWL algorithm please consult [9].

2.4 HuffSyllable

HuffSyllable is a statistical syllable-based text compression method. This technique was inspired by the principles of HuffWord algorithm [15].

It uses *adaptive Huffman tree* [7] as it's primary data structure. For every syllable type there is one tree built. In the initialization phase the tree for small syllables is filled with frequent syllables from the database together with their frequencies. In each step of the algorithm we try to estimate the type of next syllable. If the type is different than anticipated, binary code of an *escape sequence* assigned to the correct syllable type is printed on the output. Next, the code of the syllable is printed and it's frequency value in the tree is increased by one.

When the number of incrementations reaches certain value, the actualization of frequencies takes place. All the values are halved, which enforces rebuilding the whole tree.

For more information on HuffSyllable see [9].

2.5 Dictionaries

As we see, both algorithms use dictionaries of frequent syllables during the initialization. As we already mentioned in the first section, these dictionaries have crucial effect on the effectiveness of the algorithm, especially when compressing small files. These dictionaries are obtained by analysing a specimen of texts in the given language. There are two ways of doing it described in [8] - *cumulative* and *appearance* approach.

The cumulative criterion says, that a syllable is characteristic for the language, if it's quotient of occurrence to the number of all syllables in the texts is higher than certain rate. Acronym *C65* stands for dictionary containing all the syllables having the quotient higher than 1/65000. On the other hand, building an appearance dictionary means including all the syllables, for which the number of documents, where they occurred at least once, is higher than certain percentage. Experimental results proved, that the use of appearance dictionaries gave slightly better results.

Both methods have their advantages and their draw-backs. In fourth section we will describe technique based on evolutionary algorithms, which take both aspects into account.

3 Genetic algorithms in text compression

In 1997, Üçolük and and Toroslu have published an article about use of genetic algorithm in text compression. Ideas presented in our paper are strongly influenced by the results of their research, so let us give a brief summary of their method.

Üçolük and Toroslu have studied compression based on Huffman encoding upon mixed alphabet of characters and syllables[2]. This alphabet is apparently a subset of union of all letters and syllables. The issue is, which syllables should

[2] note, that this is a slightly different approach, than the one we are using. In their concept, rare syllables are dissolved into characters every time, they occur in the coded message, raising the occurrence of it's characters. In contrast, when we come

be included to ensure the optimal length of the compressed text. Observations suggested, that including nearly all the syllables usually led to best results. To prove this theory, the whole power set of the set of all syllables had to be examined. A genetic algorithm has been designed for this task.

Nice overview of genetic algorithms can be found in [5]. The general principles are well known: Candidate solutions are encoded into individuals called *chromosomes*. Chromosomes consist of *genes*, each encoding particular attribute of the candidate solution. The values each gen can have are called *alleles*. The encoding can be done in several different ways: *binary encoding*, *permutational encoding*, *encoding by tree*, and several others. A population of individuals is initiated and then bred to provide an optimal solution. The breeding is performed by two genetic operators – *cross-over*, in which the two selected chromosomes exchange genes, and *mutation*, where the value of a random gene is switched. The quality of a candidate solution is represented by so-called *fitness*. Fitness has influence on the probability, that the chromosome will be selected for mating. The higher the value of the fitness function, the better the solution and the better chance, that genes of the individual will carry over into next generations. After certain amount of generations the algorithm should converge to the optimum.

In this particular case the candidate solution is represented by a binary string, where the value 1 of i-th position means including the i-th syllable in the alphabet and 0 excluding it. The fitness represents the length of the text, if it was coded by Huffman encoding above the candidate alphabet. But performing compression and measuring the compressed text length would be rather expensive; it would require the Huffman tree construction which is known to be of order $O(N \log N)$ with considerably large multiplicative constant. Therefore it was decided rather to estimate this value theoretically. This can be done in linear time.

The approximation is grounded on two facts. The first fact can be deduced from Shannon's contribution [6]:

Lemma 1. *If the entropy of a given text is H, then the greatest lower bound of the compression coefficient μ for all possible codes is $H/\log m$ where m is the number of different symbols of the text.*

Second, the Huffman encoding is optimal. This means the ratio of Huffman compression can be well estimated as

$$\mu = -\frac{1}{\log m} \sum_{i=1}^{m} p_i \log p_i \tag{1}$$

where p_i is the probability of the i-th symbol of alphabet to occur in the text. Having the compression ratio makes it easy to compute the final code length simply by multiplying it by the bit-length of the uncompressed text, which is

across a new syllable, we encode it character by character and add it into the set of syllables. Next time we read this syllable on input, we treat it just like any other syllable.

$n \log m$. After a little mathematical brushing up we get this formula as the desired approximation:

$$l = n \log n - \sum_{i=1}^{m} n_i \log n_i \tag{2}$$

4 Characteristic syllables and their determination by GA

We have already mentioned, how important the dictionaries of characteristic syllables were for the compression ratio. We have also made clear, that the construction of these dictionaries is a difficult issue. In this section we will finally introduce a genetic algorithm designed for this task.

The input of this algorithm is a collection of documents in given language, so-called *training set*. The algorithm returns a file containing the characteristic syllables as it's output. The encoding of candidate solutions into chromosomes is again very straightforward; provided that the training set contains a set of N unique syllables, every individual is represented by a binary string of length N, where the value 1 on i-th position means including i-th syllable in the set of characteristic syllables, while 0 means excluding it. The role of the fitness function is played by estimated compressed length of a specimen from the training set. We are breeding the population to find a solution minimizing this value.

Algorithm 1 shows, how the evaluation of characteristic syllables works.

Algorithm 1 Genetic algorithm for characteristic syllables

```
syllable space initialization
generate random initial population
while not last generation do
  select several texts for specimen
  new generation ← empty set
  while size of new generation ≤ POOLSIZE do
    A ← random individual from old generation
    B ← another random individual from old generation
    C ← cross-over(A,B)
    add C into new generation
  end while
  if best individuals of old generation are better than worst new individuals then
    replace up to KEEPRATE worst new individuals with best old individuals
    /*application of elitism*/
  end if
  switch generations
  mutate random individual
end while
```

Our fitness function tries to approximate resulting bit length of the text compressed by HuffSyll algorithm. The behaviour of this algorithm enables us

to compute this value theoretically and therefore in reasonable time. The resulting dictionary of characteristic syllables should be optimal for use with HuffSyll. It will be interesting to examine, whether this dictionary introduces some improvements of the LZWL effectiveness too.

4.1 Evaluating fitness

The most important part of a genetic algorithm is the fitness function. It has to be accurate enough to provide good ordering on the set of candidate solutions and it has to be efficient, because it is called very often. The requirements concerning speed do not allow us using sophisticated calculations with high complexity.

We have decided not to use the whole training set in the fitness evaluation, but rather it's subset. For each generation we randomly select a specimen and use it for computing the fitness of all individuals. This attitude has two advantages: first, the evaluation needs less time, and second, the appearance of the syllable in the language is taken in concern. It does not only matter, how many occurrence the syllable has in the training set, but also in how many texts it appears at least once, and therefore how big the chance is, that it will appear in the specimen. After experimenting with the specimen size, we agreed on specimen consisting of five documents.

The most accurate way of evaluating fitness would be performing the actual compression and measuring the resulting file size. Again, this would be unacceptably time-consuming. We had to do an approximation similar to the one mentioned in last section.

The contribution of the characteristic syllables to the estimated bit length may be evaluated by a formula very similar to formula 2. The only difference is, that we will not only work with syllable frequencies in the file, which compressed bit length we are trying to estimate, but also with their frequencies in the whole training set. We will refer to these global numbers as n'_i for number of occurrences of i-th syllable and n' for the number of all syllables in the training set. Our new formula will be as follows

$$l = n \log n' - \sum_{i=1}^{m} n_i \log n'_i \tag{3}$$

The situation will be slightly different with the syllables marked as rare (non-characteristic). These syllables would have to be encoded character by character in the compression. They would be initialized with lower frequency, too. We take this into account in our approximation by adding an estimate of bits necessary for encoding the syllable and by increasing it's code bit length by one.

The principals of the fitness evolution are outlined in pseudo code in algorithm 2.

4.2 Setting parameters

The behaviour and effectiveness of a genetic algorithm depends on the settings of several parameters. These parameters include size of the population, probability

Algorithm 2 Evaluation of fitness

```
R ← 0
for all file in specimen do
  N' ← 0, S ← 0, P ← 0
  for all syllable in set of syllables do
    V ← number of occurrences of syllable in file
    V' ← number of occurrences of syllable in all the files
    N' ← N' + V'
    if syllable is marked as characteristic then
      S ← S + V * lg2(V')
    else if V > 0 then
      S ← S + V * (lg2(V') − 1)
      P ← P+ estimated bit length of syllable's code
    end if
  end for
  N ← number of syllables in file
  R ← R + N * lg2(N') − S + P
end for
return R
```

of cross-over, probability of mutation, number of generations, range of elitism and degree of siding with better individuals in selection. There is no general rule for setting these parameters. The situation is even more complicated by the fact, that these parameters often act in a rather antagonistic manner.

Most authors writing about evolutionary computing agree, that among these parameters the one most important is the size of the population. Population too small does not allow the algorithm to sufficiently seek through the whole search space. Inadequately large population leads to consuming too much computational power without much significant improvement in the quality of the solution. Optimal size depends on the *nature* of the problem and on it's *size*[3]. Yong Gao insists, that the dependency with size is linear [4]. We have experienced good results with populations of several hundreds individuals.

One thing that is tight very closely to population size is the type of cross-over. In [10] the advantages of different types of cross-overs (one-point, two-point, multi-point and uniform) are discussed. We have decided for multi-point cross-over, because of it's positive effect, when used with smaller populations. It prevents the algorithm from creating unproductive clones. We have set the number of cross-over points to the value of 10.

Elitism is an instrument against loosing the best solution found so far. It means, that instead of replacing whole old population with the new one, we keep several members of the old population as long as they are better than the worst members of the new population. Too much elitism may cause *premature convergence*, which is a really unpleasant consequence. To avoid this, we restrict elitism to small number of individuals, about one percent of the population.

[3] size of problem is defined as length of candidate solution encoding

In selection, better individuals are treated with favor; better chromosome has higher chance to be chosen, than the one below standard. The probability p, that an individual is chosen, may be formalized by

$$p(x_i) = \frac{k - f(x_i)}{nk - \sum_{j=0}^{n-1} f(x_j)} \tag{4}$$

were n stands for population size, f for fitness and constant k is set equal to $max_{x \in P}(f(x)) + min_{x \in P}(f(x))$. P stands for the population.

5 Experimental Results

In this section we will present results of HuffSyll and LZWL algorithms when used with genetically determined dictionaries of common syllables. The algorithms will be compared according to resulting bpc[4] value. The test will be performed on two collections of files, one for English and one for Czech.

5.1 Training sets

We have constructed two training sets, which served us as input for the genetic algorithm. They were also used for obtaining cumulative dictionary *C65*, to which we compared the results of genetically determined dictionaries.

English set consised of 1000 documents randomly chosen from two corpora; 100 files from [2] and 900 law documents from [1].

Czech sets contained 69 middle-size (mean 215,3kB) fiction texts from [3]. The rest was formed by 931 newspaper articles obtained from Prague Dependency Treebank [12]. With mean of 1,8kB they were consider as short documents.

5.2 Test sets

Both set for testing had the size of 7000 documents. Czech set contained 69 texts from [3] and 6931 articles from [12]. English test set consisted of 300 short stories from [2] and 7000 documents randomly chosen from [1].

5.3 Results

In table 1 we can see the measured results for HuffSyll and LZWL algorithms with different hyphenation and with or without use of genetically determined characteristic syllables for Czech language. In table 2 there are the same data for English.

As we can see, with HuffSyll there is significant gain when using characteristic syllables instead of cumulative dictionary C65. This positive effect is greater for smaller files. It is not surprising, because as we have already mentioned, when compressing longer files the effect of already processed part of input overweights

[4] bits per character

Table 1. Effect of characteristic syllables in compression of Czech texts

Method	100B-1kB	1-10kB	10-50kB	50-200kB	200kB-2MB
HuffSyll + P_{UL} + C65	5.32	4.70	4.18	3.95	3.89
HuffSyll + P_{UL} + GA	4.77	4.40	4.09	3.92	3.87
HuffSyll + P_{UML} + C65	5.32	4.67	4.10	3.85	3.80
HuffSyll + P_{UML} + GA	4.70	4.31	3.99	3.81	3.78
HuffSyll + P_{UMR} + C65	5.25	4.62	4.08	3.85	3.81
HuffSyll + P_{UMR} + GA	4.72	4.33	3.99	3.82	3.79
HuffSyll + P_{UR} + C65	5.29	4.64	4.09	3.84	3.80
HuffSyll + P_{UR} + GA	4.76	4.35	4.00	3.82	3.78
LZWL + P_{UL} + C65	6.16	5.19	4.29	3.80	3.54
LZWL + P_{UL} + GA	6.08	5.19	4.31	3.81	3.54
LZWL + P_{UML} + C65	6.30	5.19	4.24	3.75	3.52
LZWL + P_{UML} + GA	6.15	5.23	4.29	3.76	3.51
LZWL + P_{UMR} + C65	6.26	5.16	4.23	3.75	3.51
LZWL + P_{UMR} + GA	5.98	5.16	4.27	3.76	3.51
LZWL + P_{UR} + C65	6.30	5.19	4.24	3.75	3.52
LZWL + P_{UR} + GA	6.20	5.26	4.31	3.77	3.52

Table 2. Effect of characteristic syllables in compression of English texts

Method	100B-1kB	1-10kB	10-50kB	50-200kB	200kB-2MB
HuffSyll + P_{UL} + C65	4.51	3.62	3.26	3.16	3.16
HuffSyll + P_{UL} + GA	3.90	3.36	3.18	3.14	3.15
HuffSyll + P_{UML} + C65	4.84	3.84	3.39	3.24	3.21
HuffSyll + P_{UML} + GA	4.04	3.46	3.26	3.20	3.20
HuffSyll + P_{UMR} + C65	4.79	3.82	3.39	3.25	3.18
HuffSyll + P_{UMR} + GA	3.98	3.45	3.25	3.21	3.17
HuffSyll + P_{UR} + C65	4.90	3.88	3.41	3.27	3.25
HuffSyll + P_{UR} + GA	4.00	3.46	3.25	3.22	3.23
LZWL + P_{UL} + C65	5.61	3.63	2.81	2.70	2.86
LZWL + P_{UL} + GA	5.43	3.69	2.86	2.73	2.87
LZWL + P_{UML} + C65	5.89	3.77	2.88	2.73	2.87
LZWL + P_{UML} + GA	5.30	3.63	2.87	2.74	2.87
LZWL + P_{UMR} + C65	5.87	3.79	2.89	2.74	2.89
LZWL + P_{UMR} + GA	5.25	3.57	2.84	2.74	2.88
LZWL + P_{UR} + C65	5.92	3.80	2.91	2.77	2.91
LZWL + P_{UR} + GA	5.25	3.59	2.87	2.76	2.91

the initial settings from dictionary. An important matter is, that although the gain gets smaller with lengthy texts, it never turns into draw-back. And the average improvement of 0.6bpc in the category of shortest Czech files and 0.8bpc in the same category of English files is noteworthy.

On the other hand, use of GA dictionaries did not bring such great effort by LZWL algorithm. There is some upturn in the categories of smaller files, but it is not as remarkable as we witnessed by HuffSyll. More important is, that in some file size categories GA dictionary gave actually slightly worse results, than ordinary C65 dictionary. To conclude, we may remark, that dictionaries genetically determined for use with HuffSyll did not astonished us when used with LZWL. There is no point in preferring them to cumulative dictionaries, which are much easier to obtain. To the contrary, if we already posses the GA dictionary for HuffSyll, we may use it for LZWL as well, without worrying that the results will be too bad.

We have also compared the effectiveness of the syllable-based methods to several commonly used compress programs, namely bzip2 1.0.3, gzip 1.3.5 and compress 4.2. For comparison we have chosen the most successful hyphenation algorithm and the most successful dictionary of characteristic syllables for the given language. Table 4 contains the results for Czech and table 3 for English. Figure 1 shows the comparison for czech documents graphically.

These numbers are slightly distorted by omission of the formats overheads. gzip uses 18B of additional data for header plus 32-bit CRC checksum, and bzip2 uses 12B of additional information. Regrettably we could not find the format specification for compress, but we assume, that the overhead is similar. There is no such additional information in files compressed by LZWL and HuffSyllable. Especially in the category of the smallest files this gives them an advantage over the others, but the effect is not so high.

Table 3. Comparison with commonly used compress programs - English texts

Method	100B-1kB	1-10kB	10-50kB	50-200kB	200kB-2MB
HuffSyll + P_{UL} + GA	3.90	3.36	3.18	3.14	3.15
LZWL + P_{UL} + GA	5.43	3.69	2.86	2.73	2.87
gzip 1.3.5	4.94	3.05	2.38	2.58	3.10
bzip2 1.0.3	5.28	3.03	2.18	2.13	2.38
compress 4.2	5.86	4.19	3.44	3.25	3.31

As we see, the results of syllable-based text compression methods were not bad. In the category of smallest files, HuffSyll has fully taken the advantage of dictionary initialization and outmatched even such sophisticated program as bzip2. As long as larger texts are concerned, bzip2 complied to it's reputation, and prevailed in a convincing manner. Relatively worst results were reached by compress; it was outperformed by both syllable-based algorithms as well as by both competition programs used in production. gzip was outmatched by HuffSyll

Table 4. Comparison with commonly used compress programs - Czech texts

Method	100B-1kB	1-10kB	10-50kB	50-200kB	200kB-2MB
HuffSyll + P_{UML} + GA	4.70	4.31	3.99	3.81	3.78
LZWL + P_{UML} + GA	6.15	5.23	4.29	3.76	3.51
gzip 1.3.5	5.68	4.56	3.87	3.61	3.53
bzip2 1.0.3	6.05	4.62	3.54	3.06	2.83
compress 4.2	6.33	5.38	4.51	4.04	3.82

in the category of short documents, but was better with longer files. With LZWL it was exactly vice-versa; for larger files LZWL gave better results than gzip.

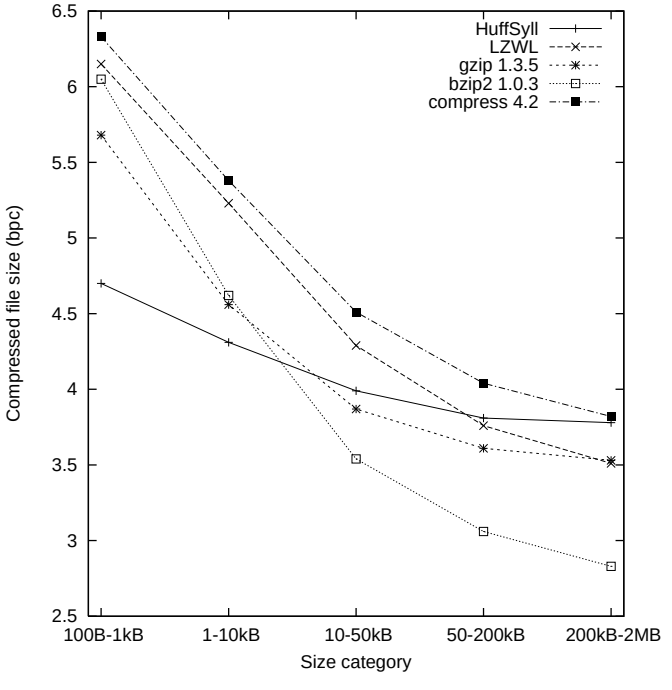


Fig. 1. Comparison with commonly used compress programs - Czech texts

6 Conclusion

We have introduced a new method for obtaining dictionaries of characteristic syllables for syllable-based text compression. On English and Czech texts, we

have documented it's advantages in comparison with cumulative dictionaries for HuffSyll. We have studied the gain it produces with respect to size of compressed files and hyphenation algorithm used. We have examined, how the use of HuffSyll optimized dictionary affects effectiveness of LZWL algorithm.

In future works, it could be interesting to design a genetic algorithm for obtaining dictionaries optimized for LZWL.

References

1. *California law* http://www.leginfo.ca.gov/calaw.html
2. *Canterbury corpus* http://corpus.canterbury.ac.nz
3. *e-knihy* http://go.to/eknihy as visited on 2nd February 2005
4. Gao, Y. Population Size and Sampling Complexity in Genetic Algorithms, *Proceedings of the Bird of a Feather Workshops (GECCO) – Learning, Adaptation and Approximation in Evolutionary Computation, 2003*
5. Goldberg, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Pub. Co. 1989, ISBN 0201157675.
6. Hamming, R. W. *Coding and Information Theory.* Prentice-Hall, Englewood Cliffs, NJ, 1986.
7. Huffman, D. A. *A method for the construction of minimum redundancy codes.* Proc. Inst. Radio Eng. 40:1098-1101, 1952
8. Lánský J., Žemlička M. *Compression of Small Text Files Using Syllables.* Technical report no. 2006/1. KSI MFF UK, Praha, January 2006.
9. Lánský J., Žemlička M. Text Compression Syllables. *Richta K., Snášel V., Pokorný J.: Proceedings of the Dateso 2005 Annual International Workshop on DAtabases, TExts, Specifications and Objects.* CEUR-WS, Vol. 129, pg. 32-45, ISBN 80-01-03204-3.
10. Spears W. M., De Jong K. A. *An Analysis of Multi-Point Crossover.* FGA, (1991) 301–315
11. *The American Heritage® Dictionary of the English Language, Fourth Edition.* Houghton Mifflin Company, 2004. http://dictionary.reference.com/browse/syllable (accessed: January 09, 2007).
12. *The Prague Dependency Treebank* http://ufal.mff.cuni.cz/pdt/
13. Üçolük G., Toroslu H.: *A Genetic Algorithm Approach for Verification of the Syllable Based Text Compression Technique.* Journal of Information Science, Vol. 23, No. 5, (1997) 365–372
14. Welsh T. A. *A technique for high performance data compression.* IEEE Computer, 17,6,8-19,1984
15. Witten I., Moffat A., Bell,T.: *Managing Gigabytes: Compressing and Indexing Documents and Images.* Van Nostrand Reinhold, 1994

Using XSEM for Modeling XML Interfaces of Services in SOA*

Martin Nečaský

Department of Software Engineering, Faculty of Mathematics and Physics, Charles University, Malostranské nám. 25, 118 00 Praha 1, Czech Republic
martin.necasky@mff.cuni.cz, http://www.necasky.net

Abstract. In this paper we briefly describe a new conceptual model for XML data called *XSEM* and how to use it for modeling XML interfaces of services in service oriented architecture (SOA). The model is a combination of several approaches in the area of conceptual modeling of XML data. It divides the process of conceptual modeling of XML data to two levels. The first level consists of designing an overall non-hierarchical conceptual schema of the domain. The second level consists of deriving different hierarchical representations of parts of the overall conceptual schema using transformation operators. Each hierarchical representation models an XML schema describing the structure of the data exchanged between a service interface and external services.

Keywords: conceptual modeling, XML, XML Schema, SOA

1 Introduction and Motivation

Recently, XML has been used for an exchange of data between heterogeneous information systems, for an internal data representation, and also as a logical database model. Therefore, modeling of XML data should become an inseparable part of the application data modeling process on the conceptual level.

For example, assume a medical application integrating data about patients from several external sources. The application works as a service. It is a black box that stores the patient data in an internal database in an internal representation and provides access to the database through predefined interfaces used by external services such as hospital systems or insurance systems. The data exchanged between an external service and the medical service is in an XML form. Each interface provides an XML schema describing the form in which the data is presented to and received from the external services through the interface. The external services do not know the structure of the internal database. They only know the XML schemes provided by the interfaces.

* This paper was supported by the National programme of research (Information society project 1ET100300419)

Assume an interface I_{exam} for exchanging results of medical examinations and an interface I_{diag} for exchanging medical diagnoses in XML. Both interfaces define XML schemes describing the required structure of exchanged XML documents. We can imagine the following scenario:

1. physician in a hospital makes a diagnosis of a patient; to decide the diagnosis, the physician needs the results of a patient's examination performed in a different hospital
2. physician requests the hospital system for the results; hospital system requests the medical service for the results through I_{exam}
3. medical service exports the results from the internal representation into an XML document with the structure defined by I_{exam} and sends it back to the hospital system
4. hospital system receives the XML document and presents the data to the physician; physician diagnoses a patient's disease and records the diagnosis to the hospital system
5. hospital system exports the diagnosis data into an XML document with the structure defined by I_{diag} and sends it to the medical service through I_{diag}
6. medical service receives the XML document with the diagnosis and stores the data into the internal database

Fig. 1 shows how the medical service similar to our example would be organized today. The figure shows the internal structure of the service. There is the internal database and a conceptual schema describing the structure of the database. For the external hospital system, the service is a black box. The hospital system communicates with the service through the interfaces I_{exam} and I_{diag}. The figure illustrates the scenario described above.

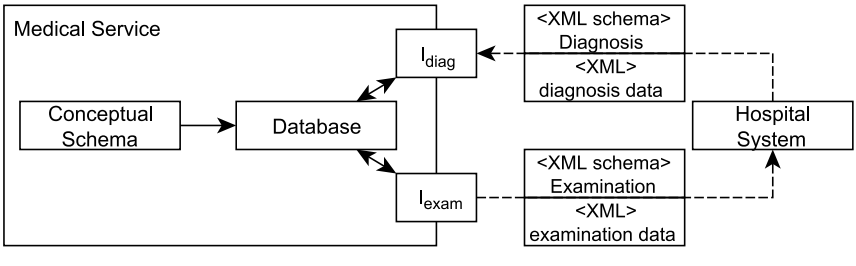


Fig. 1. Motivation

The connection of the example with the conceptual modeling is that there is a need to model the structure and semantics of XML documents exchanged between the medical service and external services/applications through the interfaces on the conceptual level. However, the following problems can occur:

1. conceptual schema and the XML schemes describing the structure of the data exchanged through the interfaces can be missing
2. if the conceptual schema and the XML schemes are present, there is no explicit binding between them (i.e. the XML schemes have to be created and maintained manually)
3. scripts for extracting data from the XML documents and transforming the data into the internal representation and vice versa must be created and maintained manually

The challenge is to eliminate these problems (1-3) by introducing a conceptual model for XML data. Such a model must allow to design an overall conceptual schema of the service domain and to derive the XML schemes describing the service interfaces (1). Even though the XML schemes organize the data into hierarchies, the overall conceptual schema need not be hierarchical. The explicit binding between the XML schemes and the overall non-hierarchical conceptual schema facilitates the maintenance of the XML schemes and the creation and maintenance of the scripts that transform data between the internal representation and the interface XML representations (2,3).

2 Related Work

If we want to model XML data on the conceptual level, we have to deal with some special features of XML data such as irregular structure, ordering, mixed content, and a hierarchical structure.

There are some approaches, for example ERX [6] or XER [8], extending the E-R model to be suitable for the conceptual modeling of XML data. Because E-R is not hierarchical (there are $M : N$ relationship types and n-ary relationship types), XML schemes must be derived in some way. The problem is that a user can not specify how the data should be organized in hierarchical XML. The hierarchical structure is derived automatically without following user requirements.

Another possibility of how to model XML data is to start from a hierarchical structure. This approach may be called *hierarchical approach*. There are conceptual models based on the hierarchical approach, for example ORA-SS [1]. Using this approach we can model hierarchical structures easily. However, a problem with modeling of non-hierarchical structures arises. Moreover, hierarchical schemes are not so transparent as non-hierarchical E-R schemes. A designer must think about the data in hierarchies which is not natural in general.

The problem of the approaches is that it is not possible to design one or more hierarchical organizations of parts of an overall non-hierarchical conceptual schema of the domain as it is required by the example medical service. Moreover, it is not possible to derive different hierarchical organizations of the same parts of the overall conceptual schema.

We propose a new conceptual model for XML called *XSEM* trying to solve the mentioned problems. In [4], we offer a survey of conceptual modeling for

XML. We propose a detailed list of requirements for conceptual models for XML, describe recent conceptual models for XML in a unified formalism, and compare the described models against the requirements. In [5], we describe our *XSEM* in a formal way.

In this paper we describe XSEM briefly and we show its possible application to modeling of XML interfaces of SOA services. There are two contributions of this paper. First, we show how XSEM can be applied to conceptual modeling of XML interfaces of SOA services and how it can facilitate important processes in the service creation and maintenance. Second, because we show a practical application of XSEM, we also concentrate on presentation features of XSEM. We show that it is necessary to extend the XSEM constructs proposed in [5] to present XSEM schemes in a transparent way.

3 Idea

We illustrate our idea with the architecture of the medical service. There is the internal logical database schema describing the structure of the data stored in the internal database. The medical service provides several interfaces used by external applications to access the internal database. Each interface provides an XML schema describing the structure of XML documents that are exchanged between the service and the external applications through the interface. We can comprehend the XML schemes as hierarchical views on parts of the internal logical schema. Each group of external applications needs different structure of XML documents. These documents can contain the same data, but in different hierarchical organizations.

Following the architecture of the medical service, we need to design an overall non-hierarchical conceptual schema of the domain. From the overall schema, we need to derive several hierarchical conceptual views. These views describe the XML schemes for the interfaces. The derivation must be driven by a designer. The hierarchical view design process consist of selecting the components of the overall schema that should be represented in the view followed by the specification of how the components should be organized in the hierarchy. At the end, the XML schemes are derived automatically from the conceptual hierarchical views.

4 XSEM Model

XSEM is a conceptual model for XML based on the previous idea. It divides the conceptual modeling process to two levels. On the first level, we design an overall non-hierarchical conceptual schema of our domain using a part of XSEM called *XSEM-ER*. On the second level, we design hierarchical conceptual schemes using a part of XSEM called *XSEM-H*. XSEM-H schemes are not designed separately from the XSEM-ER schema. We derive them from the XSEM-ER schema by so called *transformation operators*. Each XSEM-H schema is a hierarchical view on a part of the XSEM-ER schema. It describes required XML schema on

the conceptual level and there is an explicit binding between the hierarchical organization and the semantics of its components.

We can easily apply XSEM to model XML interfaces of SOA services. First, we design an overall XSEM-ER schema of the service domain and then we derive an XSEM-H schema for each service interface. The XSEM-H schema describes the XML schema for the XML data exchanged through the interface.

4.1 XSEM-ER

XSEM-ER is an extension of E-R proposed by Chen. It allows to model the special XML features like irregular structure, ordering, and mixed content. On this level, it is not important how the modeled data is organized in hierarchies. These hierarchical organizations are derived during the second part of the modeling process.

Fig. 2 shows an example XSEM-ER schema modeling a small part of the medical domain. As in the classical E-R model, there are strong and weak entity types and relationship types. *Strong entity types* represent stand alone objects and are displayed as boxes. Fore example, there is a strong entity type *Hospital* modeling hospitals or a strong entity type *Physician* modeling physicians. *Weak entity types* represent objects that depend on another objects. We display them as boxes with an inner hexagon. Each entity type the weak entity type depends on is connected with the box by a solid arrow. We call these entity types as *components* of the weak entity type. For example, there is a weak entity type *Department* with a component *Hospital* modeling departments of hospitals. *Relationship types* represent relationships between objects. Therefore, a relationship type is composed of one or more entity types called *components* of the relationship type. We display relationship types by hexagons connected by solid arrows with their components. For example, there is a binary relationship type *Employ* that represents employments of physicians at departments of hospitals or at separate clinics (an extending modeling construct described in the following text is used to unify departments and clinics).

XSEM-ER adds new modeling constructs called *data node types*, and *outgoing* and *incoming cluster types* for modeling special XML features. A *data node type* is connected with an entity type which is called *component* of the data node type. It represents data values assigned to an instance of the component. These values are not attribute values of the entity. They are data values which are mixed with the relationships and weak entities having the entity as a component value. In a graphical representation, a data node type is displayed as an ellipse with a name of the data node type.

For example, assume a patient visiting a physician (represented by the weak entity type *Visit* at Fig. 2). During the visit, the physician writes a description of the course of the visit. The description is not an attribute value of the visit, it is an unstructured text assigned to the visit. Moreover, the text is mixed with the examinations made during the visit. At Fig. 2, we use a data node type *VTxt* and an incoming cluster type (see the following text) to model this situation. Ex. 1 is a motivating example for the introduction of data node types. There is

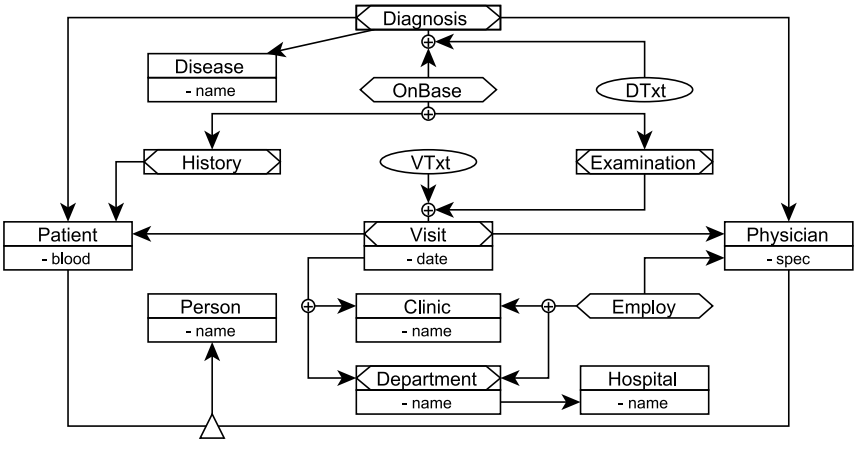


Fig. 2. XSEM-ER Schema

an element *visit* containing data about the date of the visit, the visiting patient, the visited physician, and the place of the visit. Moreover, there is a description of the visit mixed with the examinations performed during the visit.

```
<visit><date>2006-09-12</date>
  <patient><name>John Black</name></patient>
  <physician><name>Bill White</name></physician>
  <department><name>Department A</name>
    <hospital><name>Hospital B</name></hospital>
  </department>
  <description>Because of the results of a
    <examination><dsc>manual abdominal examination</dsc>...</examination>
    there is a suspicion of some liver problems. Consequently, I made a
    <examination><dsc>blood analysis</dsc>...</examination>...
  </description>
</visit>
```

Ex. 1: Mixed Content in XML

An *outgoing cluster type* represents a union of entity types and it can be used as a component of a relationship type or weak entity type. In a graphical representation, an outgoing cluster type is displayed as a circle with an inner label +. It is connected by a solid line with a relationship type or weak entity type it participates in. Each component of the cluster type is connected by an arrow going from the circle to the component.

We use outgoing cluster types for modeling irregular structure of XML. For example, patients can visit physicians at departments of hospitals and at separate

clinics. This is an example of irregular structure we can express in XML. We use an outgoing cluster type *Department*+*Clinic* to model this situation. We show the cluster type at Fig. 2. Ex. 2 is a motivating example for the introduction of outgoing cluster types. There is an element *patient* representing a patient with a name "John Black". It contains a list of *visit* elements representing patient's visits. There are two visits in the XML document. First, he visited a physician "Bill White" at a department "Department A" of a hospital "Hospital B". Then he visited a physician "Jack Brown" at a clinic "Clinic C". It is a simple example of irregular structure. There is a *department* element in the first *visit* element and a *clinic* element in the second *visit* element.

```
<patient><name>John Black</name>
  <visit><date>2006-09-12</date>
    <physician><name>Bill White</name></physician>
    <department><name>Department A</name>
      <hospital><name>Hospital B</name></hospital>
    </department></visit>
  <visit><date>2006-10-03</date>
    <physician><name>Jack Brown</name></physician>
    <clinic><name>Clinic C</name></clinic></visit>
</patient>
```

Ex. 2: Irregular structure in XML

Incoming cluster types are used for grouping different relationship types, weak entity types, and data node types having the same component. We call this component as *parent* of the incoming cluster type. The incoming cluster type specifies that instances of the components of the incoming cluster type connected with the same parent instance are mixed together. Moreover, ordering can be specified on such groups. Hence, we can use incoming cluster types for modeling mixed content in XML documents. In a graphical representation, an incoming cluster type is displayed as a circle with an inner label +. It is connected by a solid line with its parent and there is a solid arrow from each of the components to the cluster.

For example, we use an incoming cluster type $(Visit, Examination+VText)$ at Fig. 2 to model a description of a visit mixed with the examinations made during the visit. Ex. 1 described above is a motivating example. It is important to specify that the incoming cluster type is ordered because an ordering between the parts of the visit description and the examinations performed during the visit is important as shown at Ex. 1.

4.2 Hierarchical Projections

The notion of *hierarchical projections* represents the step between the non-hierarchical XSEM-ER level and the hierarchical XSEM-H level. It is a formal-

ization of binarization of relationship types and weak entity types. For example, the weak entity type $Visit$ can be represented in a hierarchy where we have a list of patients, for each patient we have the list of patient's visits, and for each patient's visit we have the visited physician and the department or clinic where the patient visited the physician. This hierarchy describes the structure of the XML document at Ex. 2.

Hierarchical projections formalize such descriptions of hierarchical organizations of non-hierarchical relationship types and weak entity types. For example, the previous hierarchy is described by the following three hierarchical projections:

$$Visit[Patient \to Visit] \quad (HP1)$$
$$Visit^{Patient}[Visit \to Physician] \quad (HP2)$$
$$Visit^{Patient}[Visit \to Department + Clinic] \quad (HP3)$$

$HP1$ represents a list of patient's visits. $HP2$ represents the visited physician and $HP3$ represent the department or clinic where the patient visited the physician. Another hierarchy is described by the following three hierarchical projections:

$$Visit[Department + Clinic \to Physician] \quad (HP4)$$
$$Visit^{Department+Clinic}[Physician \to Patient] \quad (HP5)$$
$$Visit^{Department+Clinic\,Physician}[Patient \to Visit] \quad (HP6)$$

It represents a hierarchy with a list of departments and clinics. For each department or clinic there is a list of physicians being visited by patients at the department or clinic ($HP4$). For each physician, in the context of a department or clinic, there is a list of patients who visited the physician at the department or clinic ($HP5$). Finally, for each patient in the context of a department or clinic and physician there is a list of patient's visits of the physician at the department or clinic ($HP6$).

More formally, a hierarchical projection of R is an expression $R^{context}[parent \to child]$. It specifies a hierarchy where *parent* is superior to *child*. *Context* is a sequence of components of R and specifies the context in which the projection is considered. For example, HP5 specifies a hierarchy where $Physician$ is superior to $Patient$ in the context of $Department$ or $Clinic$.

4.3 XSEM-H

XSEM-H is used for a specification of a hierarchical organization of a part of a given XSEM-ER schema using hierarchical projections. It does not add any semantics. An XSEM-H schema is an oriented graph where nodes represent entity types, relationship types, and data node types from the XSEM-ER schema and edges represent hierarchical projections of these types.

An XSEM-H schema is derived from an XSEM-ER schema by *transformation operators*. As parameters for the transformation we supply entity types, relationship types, and data node types we want to represent in the hierarchical XSEM-H schema and specify how the components should be organized in the

hierarchy. The operators are not described in a more detail here. For a more detail, see [5].

Fig. 3 shows an XSEM-H schema where the edges labeled with 1, 2, and 3 represent the hierarchical projections $HP1$, $HP2$, and $HP3$.

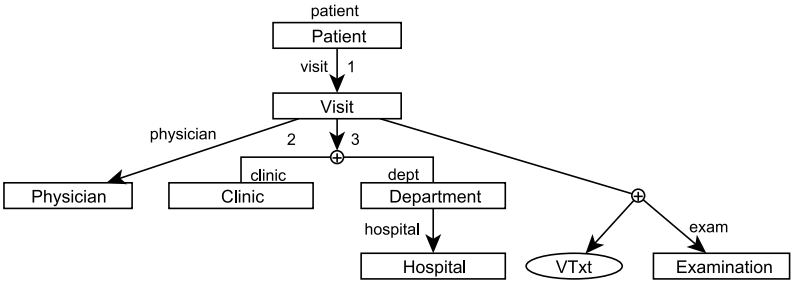


Fig. 3. XSEM-H Schema

Fig. 4 shows an XSEM-H schema where the edges labeled with 4, 5, 6, and 7 respectively, represent the hierarchical projections $HP4$, $HP4$, $HP5$, and $HP6$ respectively. At the top of the hierarchy, there is represented the outgoing cluster type $Department + Clinic$. However, we need the resulting hierarchical schema to have a tree structure. Hence, each node in the tree can have no or only one parent. For this reason we propose so called *structural representatives* in this paper as an extension to XSEM described in [5]. We represent the hierarchical projection $HP4$ by the two edges labeled with 4 and 5. Each has a child node representing *Physician*. These two child nodes have the same content which is specified by the parent of the edge labeled with 6. The child nodes of 4 and 5 are *structural representatives* of the parent of 6. The reason for this is that we specify the structure of the *Physician* representation only once and we denote the places where the *Physician* representation can be placed in the schema by one or more structural representatives.

It is important to keep a tree structure of hierarchical schemes. Otherwise, we would have problems with the schema presentation in a transparent way. Assume that the nodes representing *Department* and *Clinic* have more child nodes. Connecting the edges 4, 5 with the same child node representing *Physician* means problems with displaying the child nodes in the right order. However, this order is important when we model XML data. Moreover, such a presentation of the schema would not be so transparent in general.

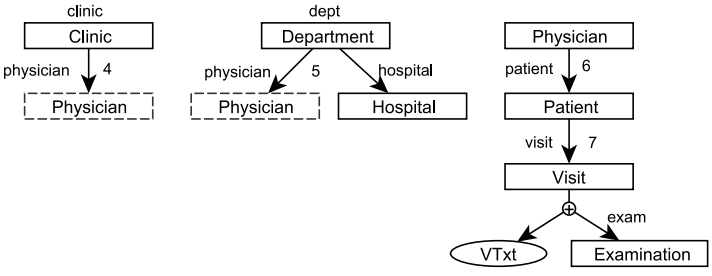


Fig. 4. XSEM-H Schema

5 Service Creation and Maintenance Support with XSEM

The goal of this paper is to show how XSEM can be applied to modeling of service XML interfaces and how it can facilitate the service creation and maintenance. We showed how we can use XSEM to model the structure and semantics of the XML data exchanged through the interfaces. We have an XSEM schema consisting of an overall non-hierarchical XSEM-ER schema describing the semantics of the data and several hierarchical XSEM-H schemes describing how the data is organized in hierarchial XML documents exchanged through the interfaces.

On the logical level we need to store the modeled data in an internal database and we need scripts for the transformation between the internal database structure and the XML structure required by the interfaces.

First, we need to derive the internal logical database schema from the XSEM schema. The process is similar to the process of derivation of relational database schemes from E-R schemes. However, this process is more complex in the case of XSEM because an XSEM schema consist of the XSEM-ER schema and one or more XSEM-H schemes. Therefore, if we want to derive an optimal database schema we have to take into account not only the non-hierarchical XSEM-ER schema but also the hierarchical XSEM-H views and optimize the logical structure of the data according to the required hierarchial organizations. There are several possibilities of how to store our data on the logical level.

(1) Native XML database systems seem as an optimal solution because we are dealing with XML data. However, if there are more XSEM-H schemes organizing the same components from the XSEM-ER schema in different hierarchies we have to choose one of them as the primary organization describing the logical schema of the native XML database and provide the scripts for the transformation between the primary organization and the other hierarchical organizations. Therefore, the performance of the system strongly depends on the ability of the native XML database system to effectively execute such transformations. However, the performance of recent native XML database systems is weaker than the performance of relational database systems.

(2) Relational database systems are very effective for strictly structured data. However, they are not too effective for semistructured data like XML. It is necessary to decompose the XML data to many tables and to join them back in different ways. It is possible to translate an XSEM-ER schema into a relational schema in a very similar way as in the case of classical E-R and to construct SQL views that build required XML documents. The advantage is that the logical database schema does not depend on required hierarchical XSEM-H views but the overall XSEM-ER schema only. Therefore, it is not a problem to add, delete, or change a particular hierarchical view. However, it can be ineffective to built frequent hierarchies repeatedly from the tables.

(3) Nor the native XML approach nor the relational approach seems to be optimal for our purposes. However, recent database systems like DB2 9 [7] offer a possibility to combine both approaches effectively. In such systems there is still a basic concept of *table* but extended with the *XML data type* supplied with an extension of SQL to build XML data from relational ones and an XQuery support. Therefore, we can combine structured and semistructured data easily in one table and combine the advantages of both approaches. Moreover, there can be many cases where data is structured but breaking the first normal form is an advantage (for attributes of entity and relationship types for example). Therefore, we can use the object-relational model instead of the relational one.

The problem to solve is what parts of the XSEM-ER schema should be represented in the object-relational model and what parts should be represented in the XML model. The basic solution is to divide the entity and relational types to two groups. The ones that are represented in more different hierarchies where it would not be effective to select one of them as the primary and transform it to the others should be represented in the object-relational model (i.e. table). The ones that are represented in only one or often repeated hierarchy where it would not be effective to construct the hierarchy frequently or the ones with a very irregular or mixed content should be represented in the XML model.

For example, the entity types *Hospital*, *Patient*, or *Physician* appear in many hierarchies with different structure. Therefore, it is better to represent them as separate tables. On the other hand, a patient history or examination have an unchanging hierarchical structure (they are documents) and their parts (not modeled in our simple XSEM-ER schema) do not appear in other hierarchies. Therefore, we can store the whole history or examination as one XML document in our database and we do not need to store their parts in separate tables.

After we have derived the logical schema from the conceptual schema we need the scripts for the transformation between the logical database structure and the XML structure of the interfaces. We have the explicit binding between the logical schema and the XSEM-ER schema, between the XSEM-ER schema and the XSEM-H schemes, and we can easily and automatically derive the XML schemes from the XSEM-H schemes. Therefore, we can automate the creation of the transformation scripts between the logical database structure and the XML interface structures.

Our approach facilitates not only the service creation but also its maintenance. Any change in the structure of the interfaces is firstly modeled in the XSEM schema. Therefore, we can propagate the change directly to the XML schemes describing the interfaces, to the logical database level, and also to the transformation scripts between them.

6 Conclusions and Future Work

In this paper, we described a new conceptual model for XML called XSEM which is based on modeling of XML schemes as views on an overall non-hierarchical schema. We described how XSEM can be used for modeling of XML interfaces of services in SOA and how it can facilitate the creation and maintenance of the services.

In our future research we will propose detailed algorithms for the translation of XSEM-H schemes to the logical XML level. Beside the grammar based XML schema languages such as XML Schema we will study the usage of pattern based XML schema languages such as Schematron [3] for the description of more complex integrity constraints. After this, we will create a prototype CASE tool for designing XSEM schemes. Moreover, we will propose algorithms for the automatic derivation of the logical database schemes from XSEM schemes and algorithms for the automatic derivation of the scripts for the transformation between the logical database structure and XML structure modeled by XSEM-H schemes as requested by this paper.

References

1. Dobbie, G., Xiaoying, W., Ling, T.W., Lee, M.L.: ORA-SS: An Object-Relationship-Attribute Model for Semi-Structured Data. TR21/00, Department of Computer Science, National University of Singapore. December 2000.
2. D. C. Fallside, P. Walmsley. *XML Schema Part 0: Primer Second Edition.* World Wide Web Consortium, Recommendation REC-xmlschema-0-20041028. October 2004.
3. International Organization for Standardization, Information Technology Document Schema Definition Languages (DSDL) Part 3: Rule-based Validation Schematron. ISO/IEC 19757-3, February 2005.
4. Necasky, M.: Conceptual Modeling for XML: A Survey. Tech. Report No. 2006-3, Dep. of Software Engineering, Faculty of Mathematics and Physics, Charles University, Prague, 2006, 54 p. `http://www.necasky.net/papers/tr2006.pdf`
5. Necasky, M.: XSEM - A Conceptual Model for XML. In Proc. Fourth Asia-Pacific Conference on Conceptual Modelling (APCCM2007), Ballarat, Australia. CRPIT, 67. Roddick, J. F. and Annika, H., Eds., ACS. 37-48., January 2007.
6. Psaila, G.: ERX: A Conceptual Model for XML Documents, *in* Proceedings of the 2000 ACM Symposium on Applied Computing, p. 898-903. Como, Italy, March 2000.
7. Saracco, C.M., Chamberlin, D., Ahuja, R.: DB2 9: pureXML Overview and Fast Start, IBM Redbooks, 134 p., June 2006.
8. Sengupta, A., Mohan, S., Doshi, R.: XER - Extensible Entity Relationship Modeling, *in* Proceedings of the XML 2003 Conference, p. 140-154. Philadelphia, USA, December 2003.

A Modular XQuery Implementation

Jan Vraný, Jan Žák

Department of Computer Science and Engineering,
FEE, Czech Technical University in Prague,
Karlovo namesti 13, 120 00, Praha, Czech Republic
{vranyj1,zakj2}@fel.cvut.cz

Abstract. *CellStore/XQuery* is modular a implementation of the XML Query Language interpreter. Primary goal of the implementation is to provide open experimental platform for implementing and testing new query optimization techniques for querying XML data. This paper describes architecture and design features of the implementation as well as possible approaches to extending the interpreter.

1 Introduction

XML is one of the most progressive technology. It has become popular as a format for data exchange between applications. However, nowadays XML is used not only for data exchange, but also for storing data. Several database management systems are specialized for storing XML documents. As more and more data were stored in the XML format, a need for querying XML data arised and thus there was a need for XML query language too.

XML query language [1] – or XQuery – is a language for querying designed by W3 Consortium. It combines features offered by two well-known and popular query languages - XPath [2] and SQL.

One of the most important and widely known constructs of SQL is *select-from-where-order by* clause. XQuery provides similar feature – so-called *FLWOR* expression. FLWOR is an acronym for *for-let-where-order by-return*.

Within an XQuery expression it is possible to use arithmetic, conditional, logical and comparison expressions, XPath expressions with predicates and function calls. It's also possible to define new functions.

XQuery is both query and transformation language, because it provides a facility for creating new XML documents using a special language construct called *constructor*.

This paper describes a XQuery implementation called *CellStore/XQuery*. The implementation was developed within the CellStore project. Primary goal of the implementation is to provide a modular implementation which is easy to understand and easy to extend. It's designed to serve as testing platform for various optimization techniques – new indexing methods, new storage models and so on.

The following text will briefly describe architecture of our XQuery implementation and possible ways how to extend the interpreter with some kind of new optimized algorithm in the future.

© J. Pokorný, V. Snášel, K. Richta (Eds.): Dateso 2007, pp. 47–54, ISBN 80-7378-002-X.
VŠB – Technical University of Ostrava, Dept. of Computer Science, 2007.

2 XQuery interpreter architecture

Because our implementation is primarily intended as an experimental framework for testing a new approaches in XML data querying, it consists of several, well-defined and well-separated components. At figure 1 there is an architecture overview. In our implementation, each component is represented by one class and all other components communicate just through calling methods of this class. In many cases, implementation of component's functionality cannot be done within one class. In such cases, there is only one class which act as a facade for the rest. These facades together with true polymorphism makes changing of the implementations very easy.

When a query is to be executed, it's sent to a parser, which converts the query from it's textual representation to internal form. Then it the query in the internal form returned to the interpreter. The interpreter will then process the query. During query processing, various data sources might (or might not) be accessed (using the `fn:doc` or `fn:collection` functions). In such case, a document provider component is asked for particular document. For data accessing and navigating through a document, a data access component is used. There are one data access component instance per document. After processing the query, result is left in interpreter's data context.

Now we will describe each component in little bit more detail.

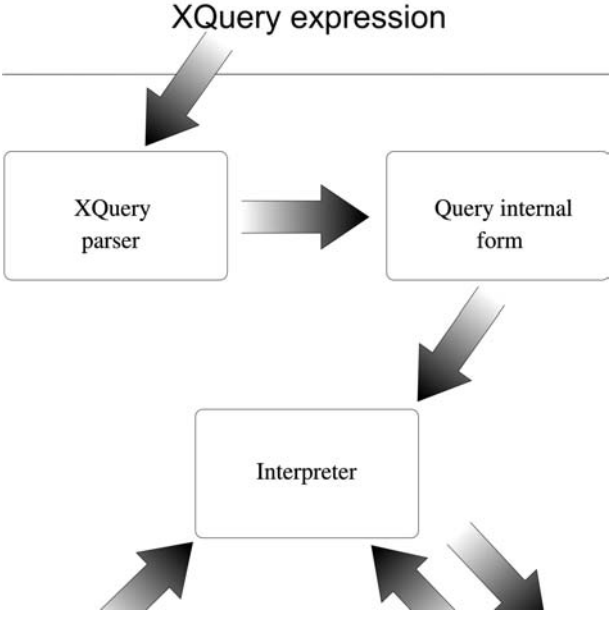


Fig. 1. XQuery architecture

2.1 Parsing XML query

The XQuery parser is implemented using SmaCC tool [5], which is able to generate LR(1) or LALR(1) parser from a grammar in the EBNF form. The grammar was taken from W3C specification. Because XQuery syntax is pretty complicated, some of its syntactic constructions cannot be easily handled by SmaCC-generated parser. This leads to some limitations of the parser which will be described in little bit more detail in the section 3.2.

2.2 Internal query representation

A parsed query is internally represented as a parse tree. For example the parse tree for the query at figure 2 is depicted at figure 3.

```
let $ a := 1
return  $ a + 1
```

Fig. 2. Example of an XQuery expression

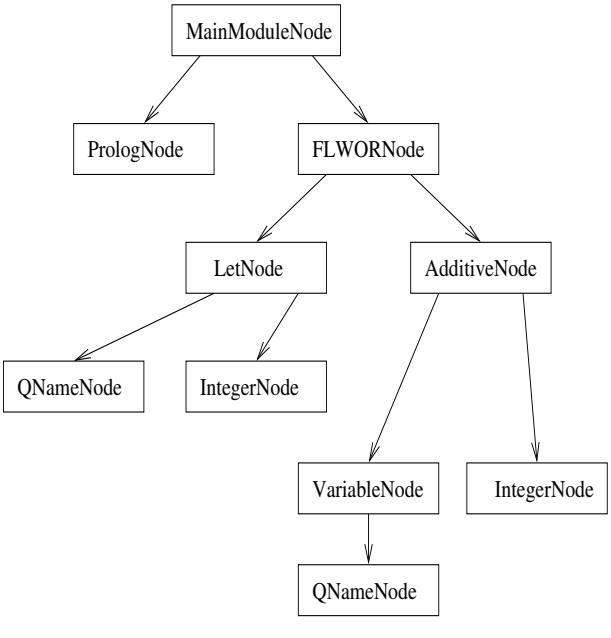


Fig. 3. Parse tree for the query at fig. 2

XQuery specifications defines so-called *XQuery Core*, which is a subset of XQuery. Every XQuery expression could be converted to XQuery Core expression, which has same semantics as the original query. Our implementation doesn't perform such conversion. A query is represented in the same form as it was written by user.

2.3 Accessing data sources

XQuery allows user to access more than one XML document per query using `fn:doc` and `fn:collection` built-in functions. These two functions loads document with given uri at current context[1], so the user can access and query data in th document. These functions could be used as many times as user wish.

Whenever a document is to be loaded into the current context, a *document provider* is asked to return the document in form in which it could be used by *interpreter* for further processing.

2.4 Accessing and navigating through XML data

Once a XML document is loaded in interpreter's context, data in the document are (not necessarily) accessed. Because we want to make our implementation as general as possible, we should have the possibility of different data source type and data models in mind. For example, documents could be stored as DOM nodes, as DOM-like nodes, as XML infoset or in data structures as they are used by various XPath accelerators.

It's important for any practical XQuery interpreter to access data stored in various data models, because user could access data in a XML database together with data stored on disk in one query.

To solve this problem, we developed a concept of *document adaptor*. It provides methods for data accessing (*xpathLocalNameOf:*, *xpathValueOf:* for example) and for navigating through the document structure in XPath axis manner (*xpathChildOf:*, *xpathParentOf:* for example). Those methods get so-called *node-id*, which identifies a node within a document, and return a set of node-ids or value represented as a string.

Full list of methods provided by the adaptor is in table 1. It's obvious that presented set of methods is not the minimal one. For example, method *xpathAncestorOf:* could be implemented using *xpathParentOf:*. It is also obvious, that such implementations might not be the most efficient ones for given storage model. Our implementation defines several so-called *primitive methods* that must be supported by the adaptor and every other method could be assembled as applications of primitive ones. However, user can override any of such methods and use the most efficient algorithm for given storage model.

Node-id could be any kind of object, it has no meaning for the rest of the system. It's used just for identifying XML nodes. For example in our implementation node-ids for DOM document tree adaptor are DOM nodes itself, whereas

[1] In fact, these function load just the root node. The rest of the document is loaded lazily.

Table 1. Document adaptor methods

method name	parameters	return value
xpathDocument		*node-id*
xpathIsAttribute:	*node-id*	`Boolean`
xpathIsDocument:	*node-id*	`Boolean`
xpathIsElement:	*node-id*	`Boolean`
xpathIsText:	*node-id*	`Boolean`
xpathLocalNameOf:	*node-id*	`String`
xpathNameOf:	*node-id*	`String`
xpathNameSpaceOf:	*node-id*	`String`
xpathValueOf:	*node-id*	`String`
xpathAncestorOf:	*node-id*	ordered set of *node-ids*
xpathAncestorOrSelfOf:	*node-id*	ordered set of *node-ids*
xpathAttributeOf:	*node-id*	ordered set of *node-ids*
xpathChildOf:	*node-id*	ordered set of *node-ids*
xpathDescendantOf:	*node-id*	ordered set of *node-ids*
xpathDescendantOrSelfOf:	*node-id*	ordered set of *node-ids*
xpathFollowingOf:	*node-id*	ordered set of *node-ids*
xpathFollowingSiblingOf:	*node-id*	ordered set of *node-ids*
xpathPrecedingOf:	*node-id*	ordered set of *node-ids*
xpathPrecedingSiblingOf:	*node-id*	ordered set of *node-ids*
xpathParentOf:	*node-id*	*node-id*

for CellStore XML database adaptor node-ids are pointers to CellStore's cell-space [6]

2.5 Interpreting the query

The XQuery interpreter is implemented as interpreter design pattern [3]. It traverses the parse tree and evaluates nodes. In order to a evaluate node, all its subnodes have to be evaluated. A node is evaluated within a *context* (which stores a data set, variable bindings and some auxiliary informations like *focus*) and as result of evaluation new context with new data is created and stored in the interpreter. There is one method per one parse tree node type in the `Interpreter` class. This has significant impact on implementation flexibility – see section 4.

An outline of method which interprets *additive node* at figure 3 is at figure 4. In fact, the real method is little bit more complicated, because XQuery defines process called *atomization*, which should be applied to every operand before addition is performed. Context handling is also a little bit tricky and should be cleaned-up.

3 Limitations

Our implementation has also some limitations, which will be described in this section.

```
visitAdditiveNode( additiveNode ) {
    var myContext, leftNodeContext, rightNodeContext;
    /* we have to save current context because it's overriden by sub-node */
    myContext = this.context;
    /* evaluate left node */
    this.evaluate( additiveNode.getLeftNode() );
    /* save its context */
    leftNodeContext = this.context;
    /* restore original context before right node's evaluation */
    this.context = myContext;
    /* evaluate right node */
    this.evaluate( additiveNode.getRightNode() );
    /* save its context */
    rightNodeContext = this.context;
    /* store result context */
    this.context = new Context( leftNodeContext.getData()
        + rightNodeContext.getData() )
}
```

Fig. 4. Method for evaluating additive node

3.1 Type system

XQuery language contains set of language constructs for typing and validating expressions against W3C XML Schema [4]. Our implementation does not support types. Unsupported constructs are not included in the grammar and thus using such constructs will result in a parse error.

Although no typing is supported in the query itself, our implementation internally uses six data types:

- `node`
- `xs:boolean`
- `xs:number`
- `xs:string`
- `xs:NCName`
- `xs:QName`

This is because XQuery contains constructs for arithmetic, conditional (*if-then-else*), logical and comparison expressions, XPath expressions with predicates and function calls.

3.2 Parser limitations

XML query language syntax contains some problematic parts, which cannot be handled by SmaCC-generated parsers.

First sort of problems is related to fact, that keywords (`if`, `div`, `return`, etc.) are not reserved words. That means, that whether some token is treated

```
element element {}
```

Fig. 5. XQuery expression with keyword as element name

as keyword token or *NCName* token depends on parsing context. Consider the query at figure 5.

Although it's perfectly legal according to the XQuery specification, our parser (generated by SmaCC) will raise a parse error because after "element" keyword (first "element") an *QName* is expected.

Another sort of problems is related to direct constructors. Direct constructors allows user to construct new nodes during the query processing using standard XML syntax. In fact, this means that every well-formed XML document is also valid XQuery expression and thus XQuery parser should parse any XML document. XML grammar itself is too complex to be handled by SmaCC. Our implementation currently supports direct constructors in a very limited way.

4 Extending the intepreter

In this section we will outline how to extend the XQueryinterpreter and add and test new optimization techniques. Basically, there are two levels at which our implementation could be extended and/or improved.

First one is the level of data access layer. To add a new type of data source, a new document adaptor has to be implemented. A complete set of tests is provided, so it's easy to find out whether new adaptor fulfils interpreters requirements. Because document adaptor implements axis-based navigation, any method that improves axis accesses can be used.

However, there are also several approaches that speeds-up not only per-axis access, but whole XPath expressions. Such techniques can be easily implemented and tested as well. Because whole query is represented as a parse tree, embedded XPath expression is represented as one parse tree node (whichh contains nodes for every part of expression) and because there is one method for each parse tree node type, one can simply override method responsible for evaluating XPath expression node and use any kind of optimization method. One can use either structural indices,value-based indices or both, if applicable. It's also possible to speed-up just some parts of XQuery expressions (for example, just parts without predicates). In fact, one can optimize any XQuery construct by this way.

5 Conclusion and future work

A prototype of the XQuery interpreter has been implemented. This implementation is currently very limited, but it supports all basic constructs: FLWOR expressions, full XPath expressions with predicates, conditionals, arithmetic, logical, comparison and quantified expressions, XML nodes construction and

function calls (both built-in and user-defined ones). It can be seen as a platform for further experiments with XML query optimization techniques using both indices (value-based and structural ones) and special techniques like structural joins.

Further development will be focused on the following topics:

- improvement of the XQuery parser
- implementing full set of built-in functions
- simplifying the context machinery in the interpreter
- improving document adaptor for CellStore/XML database by using some XML-specific optimization techniques
- creating interface to the XQuery Test Suite [7]

References

1. S. Boag, D. Chamberlin, M. Fernández, D. Florescu, J. Robie, and J. Siméon. *XQuery 1.0: An XML Query Language*. W3C, 1st edition, 2006. http://www.w3.org/TR/xquery/.
2. J. Clark and S. DeRose. *XML Path Language (XPath) Version 1.0*. W3C, 1999. `http://www.w3.org/TR/xpath`.
3. K. B. Sherman R. Alpert and B. Woolf. *The Design Patterns Smalltalk Companion*. Addison Wesley, 1st edition, 1998.
4. C. M. Sperberg-McQueen and H. Thompson. *XML Schema 1.1*. W3C, 2006. `http://www.w3.org/XML/Schema`.
5. The Refactory Inc. *SmaCC compiler-compiler*. The Refactory Inc., 1st edition, 2000. http://www.refactory.com/Software/SmaCC.
6. J. Vraný. Cellstore - the vision of pure object database. In *Processings of DATESO 2006*, pages 32–39, 2006.
7. W3C XML Query Working Group. *XML Query Test Suite*. W3C, 1st edition, 2006. http://www.w3.org/XML/Query/test-suite/.

A Content-Oriented Data Model for Semistructured Data

Tomáš Novotný

Czech Technical University, Faculty of Electrical Engineering,
Technická 2, 166 27 Prague 6, Czech Republic
novott2@fel.cvut.cz

Abstract. There are several data models that are capable of handling semistructured data. The best known are OEM, XML DOM, RDF, and the ECMAScript object model. All these models have different purpose. OEM was used by systems for integration of heterogeneous data sources. XML DOM is specified as a programming interface to manipulate XML documents used as a unified medium for data exchange. RDF provides primarily a data model for sharing metadata. The ECMAScript object model is widely used to manipulate data in web applications. However, none of these models is intended to be used directly in an interactive way. This paper presents the CO (Content-Oriented) data model, which is designed for users to browse, annotate, and relate pieces of information. It can provide change notifications hence it can be directly used in interactive applications without building an extra object model.

Keywords: data entity, data aspect, informed list, informed property

1 Introduction

Traditional mainstream databases save data according a schema that describes the context of the data. They usually need to specify the schema in advance. This requirement is for certain data very difficult or even impossible to fulfill. It is a case of irregular data or data that structure changes over time [1].

Another option is to store data together with a description of its meaning. Such data do not have to be structured according a schema and therefore they are referred to as semistructured data [2]. Data models for semistructured data can be easily used by knowledge management systems or systems for integration of heterogeneous information sources [3].

The existing well-known data models for semistructured data are not intended to be directly used in interactive applications. Programming interfaces around these models are usually based on elaborate query languages that simplify locating and extracting of particular information from these models. However, the more advanced queries can be answered, the more complex systems have to be built to detect changes. Therefore, there is usually no unified support for events that notifies user interface controls when

something has changed hence such data models cannot be directly used in interactive applications.

This paper describes the CO data model that unlike other data models for semistructured data is intended for interactive applications, especially applications that allow users to browse, annotate, and relate pieces of information. It can be also used for visual manipulation with data stored in other data models or exchange formats for semistructured data like XML or JSON. It is designed to be very simple in order to keep implementation simple.

The CO data model has a modular architecture. A core provides very limited functionality enabling basic manipulation with data. The core itself is not intended for interactive applications as the change notifications are not supported. Nevertheless, it may be used by scripts that are using the same data model. Other features are provided by external modules called extensions. This extensible architecture allows developers to choose particular configuration that will support just required features without unwanted ones. It also enables having various implementations of the same feature in order to enable direct data binding for various user interface toolkits.

Other data models for semistructured data and systems for that they were built are described in the second chapter. A design the CO data model and an implementation of a framework for the CO data model are presented in the third and fourth chapters. Conclusions and a future work are given in the last chapter.

2 Related Work

With the development and growth of the Web [4] in the last decade of the last century there arose a need to extract and integrate data that are available on the Web [5]. At the same time, there was a research project called TSIMMIS whose aim was to create a system for assisted integration of data either structured (e.g. database records), semistructured (e.g. web pages), or unstructured (e.g. plain files) from heterogeneous data sources [6].

2.1 TSIMMIS

TSIMMIS was a joint project between Stanford and the IBM Almaden Research Center [3]. It had mediator architecture [7] [8] and it basically consisted of four types of core components: translators, mediators, constraint managers, and classifier/extractors. Translators were template-based wrappers [9] that converted data from various sources into a common information model. Mediators were information routers that forward queries to particular translators and merge the results [10]. Constraint managers [11] ensured that the integrated data are consistent. Classifier/extractors automatically classified and extracted key properties of resources from the unstructured data sources. They were based on the Rufus system developed by the Almaden Research Center [12].

Information between these components was exchanged in a self-describing object model called OEM (Object Exchange Model) [13]. OEM allowed the storing of nested objects. Each object was represented by a structure with four fields: label, type,

value, and oid. The label was a string tag that describes what the object represents. The type was a data type of the object's value. The value stored the actual data. The oid was a unique identifier of the object.

TSIMMIS was not a fully automated system, but rather a tool to assist humans. It provided a graphical user interface component called MOBIE (MOsaic Based Information Explorer) [14] that allowed users to browse OEM objects and specify queries in an SQL-like language called OEM-QL [13].

2.2 LORE

LORE (Lightweight Object REpository) was a database management system designed to manage semistructured data [1]. It was built on top of the O^2 object database [15]. Semistructured data were stored in a modified OEM that was presented in the TSIMMIS project. That version of OEM could be represented as a labeled directed graph where the vertices were objects. There were two types of objects: complex objects and atomic objects. Whereas the complex objects might have outgoing edges to other objects, the atomic objects had no outgoing edges but they contained a value from one of the atomic types (e.g. integer or string).

LORE provided a query language called LOREL [16]. LOREL was defined as an extension to OQL (an SQL-like query language for the ODMG model) [17]. LOREL was also later used as a query language for TSIMMIS [8].

LORE introduced a DataGuide [18]. The DataGuide was a structure summary of the OEM model that was automatically maintained. DataGuides allowed users to browse the OEM model and formulate queries. They were also used by the system to store statistics and to optimize the queries. The DataGuide itself was an OEM object so it could be stored in and managed by an OEM DBMS.

Another feature of LORE was an external data manager [19]. The external data manager allowed users to integrate data from external data sources. The external data were represented by an external object. The external object was stored in the OEM database and it contained both a specification on how to fetch the external data and a cached version of the external data. The wrappers for the external data were reused from the TSIMMIS project.

After the emergence of XML, developers of LORE found that data models of XML and OEM were similar. So they decided to modify LORE to serve as a data management system for XML [20]. The modifications to LORE also required changes to the data model [21].

2.3 XML

XML is a markup language designed for data exchange [22]. The data are represented as documents. The XML document is a tree-like structure built from nested tagged elements. Each element can contain data stored as attributes/value pairs or as a plain text. XML also provides a mechanism to create links between elements.

There are several ways to extract information from XML documents. None of them can be considered as universal but each is convenient for a particular purpose. One of

the best known is SAX. SAX (Simple API for XML) provides a read-only and forward-only event-driven interface [23]. Another read-only interface is XmlReader (not to be confused with XMLReader, the Java interface from the SAX2 library). In contrast to SAX, XmlReader allows developers to navigate through XML on-demand in the way that is sometimes referred to as a pull model [24]. More sophisticated navigation provides XPathNavigator that enables cursor-like navigation in a XML document powered by XPath expressions [25].

There is also a standardized virtual data model for XML called DOM (Document Object Model) that is specified as a programming interface and it is developed to manipulate a memory representation of XML documents [26].

2.4 RDF

RDF (Resource Description Framework) is a set of specifications [27] that was created to provide a unified way to share metadata and it can be also used to represent data [28]. Although RDF is more complex than previous models, the data model of RDF can be represented by a collection of triples [29]. Each triple consists of a subject, a predicate, and an object. Predicates are also called properties. Subjects may be URIs or blanks. Predicates are URIs. Objects can be URIs, literals, or blanks. URIs may be used both as references to existing resources and as a global identifiers. Literals represent values. They may be plain or typed. Blanks are local identifiers.

There are various languages that extend capabilities of RDF. The best known are RDF Schema and OWL. RDF Schema allows users to describe classes and properties [30]. OWL (Web Ontology Language) enables data description using ontologies [31].

2.5 ECMAScript object model

The ECMAScript object model is widely used by web applications to manipulate data [32]. Structure of the ECMAScript is similar to LORE's version of OEM. The difference between LORE's OEM and the ECMAScript object model is that objects in the ECMAScript object model have no identifiers and each object can have only a single property with the same name. Multiple values can be represented by a special build-in type of object – an array. The textual representation of this model is sometimes referred to as JSON [33]. JSON has similar purpose as XML. There is also a standard 'ECMAScript for XML' that adds native XML support to ECMAScript [34].

2.6 iDM

iDM is an advanced data model that is designed to represent heterogeneous data [35]. iDM is now being developed as a part of a personal information management system iMeMex [36].

3 Design

As mentioned in the introductory chapter, this paper describes the CO data model that is designed for interactive applications. This implies that except of developers, the design should also take account of users. There are three elemental requirements: The first is that the data model should be simple because the simplest it would be, the easiest the implementation of a framework for the data model would be. The next is that the data model has to be extensible so the developers can use only that parts what they really need. The last requirement is that it should have similar concepts as existing systems for information handling that are widely used by users, so that users do not need to spend time for an extra training.

The most famous and widely used system that is used by users to access information is the Web. The information on the Web is usually stored in a document called a web page. There are two basic ways to access a particular web page: users can either enter the address of the web page or navigate to a particular web page from another web page. Advanced methods include searching by entering a query into a search engine or navigating through tags. Tags are usually keywords that are attached to a piece of information. In fact, navigation through tags is mostly a particular form of simple page to page navigation, as tags are usually located within the page.

The data model should offer thus similar ways of accessing information as the Web: The data model should allow users to navigate through the data and access the data directly from an address, enable developers to easily create tagging systems, and provide programming interfaces so that specialized search engines can be built or adapted to process search queries.

3.1 Data model

The architecture of the CO data model results from its requirements. The data model consists of items called entities. The entity is a wrapper for data. The data contained in the entity will be referred to as content. In addition to the content, each entity has a type and optionally has references to other entities.

The content of an entity can be anything. Entities can store numbers, text documents, multimedia files, an array of objects or other entities, or any other pieces of information. Entities do not need to contain data directly, but they can contain special objects that refer to external data stored in local files, databases, or even remote web sites.

The entities' types have a similar purpose as the labels in TSIMMIS's version of OEM or the tags in XML, with the difference that labels in OEM and tags in XML are strings but entities' types are represented by another or, in special cases, the same entity. This architecture allows saving common metadata to the entity that represents the type. Metadata can contain both information for humans, such as documentation, and information for machines, for instance specifications of constraints or processing instructions for manipulation of external data that are referred in the content. The fact that the type of entity `A` has is entity `B` can be expressed in RDF by a triple `<entity:A> <rdf:type> <entity:B>`.

Entity reference is consists of a key and a value. It is a simplified version of the properties in RDF. Both the key and value of a reference are entities. In contrast to RDF, the value of a reference cannot be a literal. As with the ECMAScript object model, entity can have at most one reference with the same key. As the reference key is an entity, the entity can store metadata that can provide information such as semantic meaning, usage constraints, or human-readable documentation. Fig. 1 shows the data model of the entity.

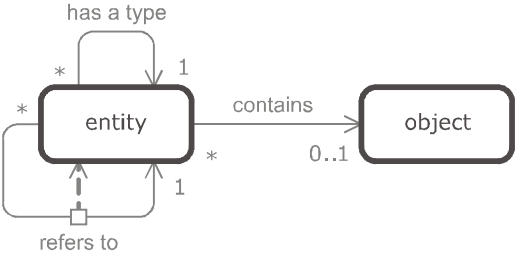


Fig. 1. The data model of the entity.

Entities can be directly accessed by URIs. However, URIs are not mandatory for entities. An entity without a URI can be accessed by searching or by local navigation.

This part described the core part the CO data model. Additional features can be provided as extensions. Following part describes representation of several basic data structures.

3.2 Data structures

Unlike RDF or the ECMAScript object model, the CO data model has no direct support for collections. As collections are the most common data structures, this part sketches how to form collections in the CO data model.

There are basically two ways to build a collection. The first way is to create a special data type that represents a collection, so the collection will be stored in the content of an entity. This method allows the storing of any type of collection. It is especially useful for vectors or matrices and it can be likened to containers in RDF or arrays in the ECMAScript object model.

The second method uses references to group items in a collection. There are two types of collection that can be formed. The first type includes collections whose entity directly refers to multiple items. Basic examples of such collections are simple or hierarchical dictionaries.

The other type of collection points directly to just one item. One of the simplest examples is a singly linked list. The singly linked list is usually formed by adding to each item a reference that points to the following item. There can be also a special object that represents a whole collection. This object has a reference to the first item in the collection. In this scenario, items of a collection do not know in what collection they are contained without an additional reference and they can be usually contained in a single collection.

The CO data model enables the creation of specialized singly linked lists where each item can be contained in multiple collections and each item knows what collections it is contained in without additional references. Such linked lists will be referred to as informed lists. The architecture of an informed list is as follows.

The entity that represents a collection remains unchanged – it has a reference to the first item. Items also contain references to the next item. However these references are not identified by a general next entity but by the entity of the collection. As each collection is represented by a different entity, items have references with different keys and hence they can be contained in multiple collections. Items also know the collections where they are included because this information is stored in the keys of the references to the next items. Fig. 2 shows an example of an informed list.

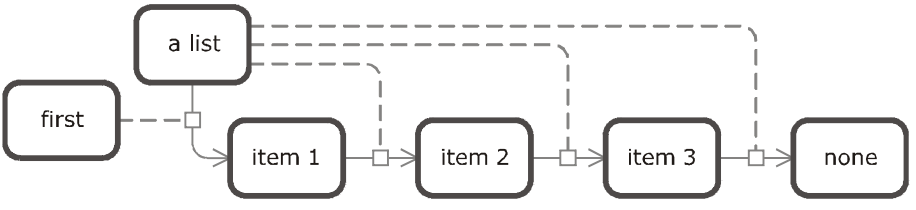


Fig. 2. Example of an informed list. The list is represented by entity “a list”. It contains three items: “item 1”, “item 2”, and “item 3”. Entity “first” represents a known entity that is used as key that refers to the first item and “none” is another known entity needed to store reference to the owning list in the last item. Entities are represented by boxes, references are rendered as solid arrows, and keys of the references are pointed by dashed lines.

One possible application for informed lists is a tagging system where each tag is represented by a collection of entities that are tagged by this tag. A tagging system can list all tags for a resource in a constant time without any additional indexes. It can also immediately retrieve an additional resource for each tag. This feature can be useful to provide a simple and fast tag-based ‘see also’.

Several informed lists can form a multilevel hierarchical structure where descendants know their ancestors. One of the possible applications is a system for hierarchical categorization.

The idea of informed lists can be also used to construct properties that can have multiple values and that know what objects refer to them. Such properties will be called informed properties. This construct can be conceived as a fusion of informed lists. Each list contains values for a particular property. And the key of the reference to the first value in each list is replaced by a key that will identify the property. An example of informed property is in Fig. 3.

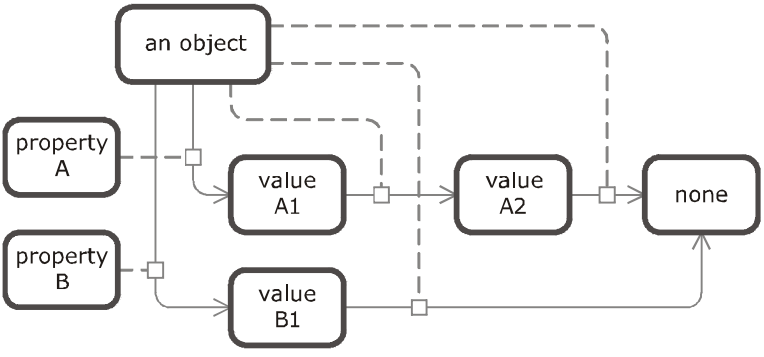


Fig. 3. Example of informed properties. Entity “an object” has two informed properties: “property A” and “property B”. Property “property A” has two values “value A1” and “value A2” and property “property B” with one value “value B1”.

4 Implementation

This chapter is divided into two parts. The first part outlines the general architecture of a framework that implements the CO data model. The second part describes a prototypical implementation to evaluate the CO data model.

4.1 Architecture

All frameworks for the CO data model should implement the core of the data model. The core was described in the Architecture part of the Design chapter. A core framework should provide programming interfaces to access entities and to get and set a type, references, and content. There are no methods to create or delete entities because each URI refers to an existing entity. The consequence is that there is an infinite number of entities. However, only a finite number of entities contain non-default information hence the framework should store only the entities with non-default information. The entities with default information are called implicit entities and the other entities are called explicit entities. The implicit entities have null content, no references, and a default type.

The minimal framework may implement only a volatile data model. Such a framework can be useful, for instance, in applications that use existing web services to store data.

Other features are optional and they are not part of the core. They are provided by extensions. There are several types of extensions. One type of the extension is a data aspect. The data aspect is a component that simplifies manipulation with a complex data structure, such as the informed list. The data aspect for the informed list can provide methods to add, remove, or search items. It can enhance the performance by cached backward links or a cached index array for immediate random access.

Another type of extension is a data store provider. The data store provider allows loading and saving data to data storages either local (e.g. files, databases) or remote

(e.g. web services). They can also serve as data adapters between the CO data model and internal object models of other applications.

Moreover, interactive applications require knowing when something has changed. This implies enhanced implementation of the core that raises an event if a change occurs. The system for event notification may be developed for a certain user interface toolkit so the application developers can directly bind controls to the data model.

4.2 Prototype

Currently, there is a prototype of a framework for the CO data model. It is written in C# [37] and runs on the .NET 3.0 Framework [38] [39]. The data model is implemented as .NET Component Model [40]. It was specially developed for the Windows Presentation Foundation [41]. It implements a data aspect for the informed list that caches backward links and an index array to provide faster random access. The data aspect also provides collection-change notifications so it can be bound to standard controls that display collections.

The framework supports simple transactions that delays commit and provides rollback of changed data. There are four data store providers: file system provider, SQL provider, Lucene.Net [42] provider, and Berkeley DB [43] provider.

4.3 Evaluation

The architecture of the data model allows simple implementation of the data model as .NET Component Model with change notifications. Therefore, application developers can directly bind data to user interface controls. Two-way data binding [44] let them develop applications where users can interactively manipulate data while the developers do not have to build an extra object model and write a glue code to transfer data between data stores and user interface controls.

5 Conclusion

The extensible architecture of the CO data model has some advantages and drawbacks. The main advantage is that complex features can be later replaced with a better implementation. However, it can result in several implementations of the same feature and developers have to choose one that best fits their needs, so they may spend extra time analyzing various implementations. This can be avoided, of course, by maintaining a list of various implementations with comparisons and use cases.

As is mentioned in the introduction, in contrast to OEM, XML, and RDF, the CO data model is intended to be directly manipulated by users through user interface controls. Therefore, before a development of other extensions is undertaken, this model must prove that it is really useful and that future work on this model will not be a waste of time. Such proof may be done by building small applications that will help users to organize information from various aspects, such as a system to store personal thoughts or to manage personal data from other applications.

If the data model will be successful, it can be later enhanced by other extensions demanded by users. Some possible extensions might be a change log to enable full undo functionality or allowing users to share their information on a peer-to-peer network and letting them specify particular access rights for particular users on a particular set of their information.

Acknowledgments. I would like to express many thanks to Christoph Quix, who has taught me to write a paper and with whom I was consulting the content of this paper, and Roger Hughes, who made a deep language correction of the draft.

References

1. J. McHugh, S. Abiteboul, R. Goldman, D. Quass, J. Widom. Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3), pages 54-66, 1997.
2. P. Buneman. Semistructured data. *In Proc. of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 117-121, 1997.
3. S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, J. Widom. The TSIMMIS Project: Integration of Heterogeneous Information Sources. *In Proc. of IPSJ Conference*, pages 7-18, 1994.
4. T. Berners-Lee, R. Cailliau, J. Groff. The World-Wide Web. *Computer Networks and ISDN Systemts*, 25, pages 454-459, 1992.
5. O. Etzioni. The World-Wide Web: quagmire or gold mine? *Communications of the ACM*. Volume 39, Issue 11, 1996.
6. J. Hammer, J. McHugh, H. Garcia-Molina, Semistructured Data: The TSIMMIS Experience, *In Proc. ADBIS'97*, St. Petersburg, Russia, 1997.
7. G. Wiederhold. Mediators in the Architecture of Future Information Systems. *Computer 25*, 1992.
8. H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajamaran, Y. Sagiv, J. Ullman, V. Vassalos and J. Widom, The TSIMMIS approach to Mediation: Data Models and Languages, *Journal of Intelligent Information Systems*, 8(2) pages 117-132, 1997.
9. J. Hammer, H. Garcia-Molina, S. Nestorov, R. Yerneni, M. Breunig, and V. Vassalos. Template-based wrappers in the TSIMMIS system. *In Proc. ACM SIGMOD International Conference on Management of Data (SIGMOD '97)*, pages 532-535, 1997.
10. Y. Papakonstantinou, S. Abiteboul, and H. GarciaMolina. Object fusion in mediator systems. *In Proc 22nd. VLDB conference*, 1996.
11. S. Chawathe, H. Garcia-Molina, J. Widom. Constraint Management in Loosely Coupled Distributed Databases. Technical report, Computer Science Department, Stanford University, 1993.
12. K. Shoens, A. Luniewski, P. Schwarz, J. Stamos, and J. Thomas. The RUFUS System: Information Organization for Semi-Structured Data. *Proceedings of the International Conference on Very Large Databases*, pages 97-107, Dublin, Ireland, 1993.
13. Y. Papakonstantinou, H. Garcia-Molina, J. Widom. Object Exchange Across Heterogeneous Information Sources. *In Proc. of 11th International Conference on Data Engineering (ICDE'95)*, pages 251-260, Taiwan, 1995.
14. J. Hammer, H. Garcia-Molina, K. Ireland, Y. Papakonstantinou, J. Ullman, J. Widom, Information translation, mediation, and mosaic-based browsing in the TSIMMIS system. *In Proc. of the ACM SIGMOD Internation Conference on Management of Data*, 1995.

15. O. Deux. The O2 system. *Commun. ACM 34*, num. 10, pages 34-48 1991.
16. S. Abiteboul, D. Quass, J. McHugh, J. Widom, J. Weiner. The Lorel Query Language for Semistructured Data. *Journal of Digital Libraries*, 1(1), pages 68-88, 1997.
17. R. Cattell, T. Atwood. *The Object database standard, ODMG-93*. Morgan Kaufmann Publishers Inc. 1994, ISBN 978-1558603028.
18. R. Goldman, J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. *In Proc. VLDB*, pages 436-445, 1997.
19. J. McHugh, J. Widom. Integrating Dynamically-Fetched External Information into a DBMS for Semistructured Data. *SIGMOD Record*, 26(4), pages 24-31, 1997.
20. J. Widom. Data management for XML: Research directions. *IEEE Data Engineering Bulletin*, 22(3), pages 44-52, 1999.
21. R. Goldman, J. McHugh, J. Widom. From semistructured data to XML: Migrating the Lore data model and query language. *In Proc. of the WebDB workshop*, 1999.
22. T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, F. Yergeau. Extensible Markup Language (XML) 1.0 (Fourth Edition), *W3C Recommendation*, 16th August 2006.
23. Simple API for XML. http://www.saxproject.org/
24. A. Skonnard. XML in .NET: .NET Framework XML Classes and C# Offer Simple, Scalable Data Manipulation. *In MSDN Magazine*, January 2001.
25. D. Esposito. Manipulate XML Data Easily with the XPath and XSLT APIs in the .NET, *In MSDN Magazine*, July 2003.
26. W3C. Document Object Model. *W3C Recommendation.*
27. F. Manola, E. Miller. RDF Primer. W3C Recommendation 10 February 2004.
28. S. Decker, S. Melnik, F. van Harmelen, D. Fensel, M. Klein, J. Broekstra, M. Erdmann, I. Horrocks. The Semantic Web: The Roles of XML and RDF. *IEEE Internet Computing*, 4 (5), pages 63-74, 2000.
29. G. Klyne, J. Carroll. RDF Concepts and Abstract Syntax. *W3C Recommendation*, 10th February 2004.
30. D. Brickley, R. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. *W3C Recommendation*, 10th February 2004.
31. D. McGuinness, F. van Harmelen. OWL Web Ontology Language. *W3C Recommendation*, 10th February 2004.
32. ECMA. ECMAScript Language Specification. *Standard ECMA-262 3rd Edition*, 1999.
33. D. Crockford. The application/json Media Type for JavaScript Object Notation (JSON). *Request for Comments: 4627*, July 2006.
34. ECMA. ECMAScript for XML (E4X) Specification. *Standard ECMA-357 2nd Edition*, 2005.
35. J. Dittrich, M. Salles. iDM: a unified and versatile data model for personal dataspace management. *In Proc. VLDB 2006*, 32, pages 367-378, 2006.
36. J. Dittrich, L. Blunschi, M. Färber, O. Girard, S. Karakashian, M. Salles. From Personal Desktops to Personal Dataspaces: A Report on Building the iMeMex Personal Dataspace Management System. *In BTW 2007*, 2007.
37. ECMA. C# Language Specification. *Standard ECMA-334 4th Edition*, 2006.
38. .NET Framework 3.0 Technologies, *MSDN* http://msdn2.microsoft.com/en-us/netframework/aa663323.aspx
39. .NET Framework 3.0 Community http://www.netfx3.com/
40. System.ComponentModel Namespace, *.NET Framework Class Library, MSDN* http://msdn2.microsoft.com/en-us/system.componentmodel(vs.80).aspx

41. Windows Presentation Foundation, *MSDN Library*
http://msdn2.microsoft.com/en-us/library/ms754130.aspx
42. Lucene.Net
http://www.dotlucene.net/
43. Oracle Berkeley DB Product Family
http://www.oracle.com/database/berkeley-db/
44. Data Binding Overview, *WPF, MSDN Library*
http://msdn2.microsoft.com/en-us/library/ms752347.aspx

Index-Based Approach to Similarity Search in Protein and Nucleotide Databases

David Hoksza and Tomáš Skopal

Department of software engineering, Faculty of Mathematics and Physics,
Charles University in Prague
Malostranské nám. 25, 118 00, Prague 1, Czech Republic
{david.hoksza, tomas.skopal}@mff.cuni.cz

Abstract. When searching databases of nucleotide or protein sequences, finding a local alignment of two sequences is one of the main tasks. Since the sizes of available databases grow constantly, the efficiency of retrieval methods becomes the critical issue. The sequence retrieval relies on finding sequences in the database which align best with the query sequence. However, an optimal alignment can be found in quadratic time (by use of dynamic programming) while this is infeasible when dealing with large databases. The existing solutions use fast heuristic methods (like BLAST, FASTA) which produce only an uncontrolled approximation of the best alignment and even do not provide any information about the alignment approximation error. In this paper we propose an approach of exact and approximate indexing using several metric access methods (MAMs) in combination with the TriGen algorithm, in order to reduce the number of alignments (distance computations) needed. The experimental results have shown that a straightforward adoption of MAMs to sequence retrieval cannot outperform the specialized heuristic algorithms (at least at the moment). On the other side, the results show MAMs could provide a basis for specialized access methods capable of precision/efficiency trade-off control.

Keywords: bioinformatics, indexing, database, MAM, (P)M-tree, TriGen, BLAST

1 Introduction

When managing biological sequence information, it is important to realize that organisms linked by common descent use similar genes to secure their biological functions. Therefore, it makes sense to store known genetic sequences in a database and search for similarities among them. Because these databases grow quickly in time and due to expensive comparison of sequences, there is a strong need for efficient methods capable of handling such amounts of data.

One way to handle the increasing number of sequences is indexing, which is unavoidable in today's situation of exponentially growing databases to be searched (Figure 1). There have appeared indexing applications in this area already, however, they usually tried to split sequences into q-grams (substrings

of length q) [14] and compute simple Hamming distance [21] on them. This is inappropriate in many cases because of neglecting the evolutionary nature of similarity in protein and nucleotide sequences. As an other disadvantage, some information is lost when cutting sequences into q-grams and computing distances just among these q-grams. Some efforts have also been undertaken in order to develop non-hamming distance which reflects better biological meaning of similarity [23], while such a distance has been used to index q-grams [11]. Anyway, today's most used methods for sequence retrieval are based on BLAST - a heuristic approach supporting local alignment (described in Section 2.2).

1.1 Protein Databases

DNA molecules consist of a linear (unbranched) string of *nucleotides* (conventionally labeled A, C, G, T) which can be transcribed to RNA and later translated to proteins. A *protein* molecule is a linear chain of *amino acids*. There are 20 amino acids, each encoded by a triplet of nucleotides (called *codon*). The assignment between codons and amino acids is determined by an evolutionarily fixed code table – the genetic code. Every protein has some biological activity which is derived from its three-dimensional structure. Similar amino acid sequences tend to have similar three-dimensional structure and therefore similar function[1]. When a protein sequence is determined (nowadays it is usually derived from an experimentally obtained sequence of the corresponding gene), its function is usually unknown. One should try to find similar proteins in the database of already known protein sequences (even in proteins of different species) to find out possible purpose of the newly sequenced protein or at least to get a clue of it. Therefore, it is appropriate to search through as large amount of data as possible to increase the probability of finding similar protein.

1.2 Nucleotide (DNA) Databases

Using nucleotide databases is not as common as using protein databases[2]. They are searched only for similarities among sequences of one species (unlike protein databases) which follows from the aims why they are analyzed, i.e.

- finding similarities in parts of nucleotide sequences which are not transcribed to mRNA (non-coding sequences)
- checking whether someone else has already sequenced given segment of DNA
- checking whether given segment was sequenced incorrectly

[1] Moreover more extensive sequence similarity can be viewed as evidence of common ancestry, and therefore as a basis for reconstructing phylogenetic history of organisms and their genes.

[2] which is the reason why we have tested our method for protein sequences only, but it can be used for nucleotide sequences without any change at all.

1.3 Existing Prominent Databases

As comes out from previous, the role of databases in bioinformatics is of major importance. Nowadays, there exist three most prominent databases of sequences – American *GenBank* [8], European *EMBL* [2] (European Molecular Biology Laboratory Data) and Japanese *DDBJ* [1] (DNA Data Bank of Japan), which are not moderated (means that anybody well-founded can add a sequence into).

Besides these databases, there exist several other (mostly protein) databases. In our experiments we used the *Swiss-Prot* [4] – a moderated database of proteins. The *Swiss-Prot* together with *TrEMBL* (TRanslated EMBL) and *PIR* (Protein Information Resource) constitute the *UniProt* database [22], which serves as a central repository of protein sequences and their functions. As we can see in Figure 1, the size of databases grows exponentially in time, so the needs for more efficient methods grow as well.

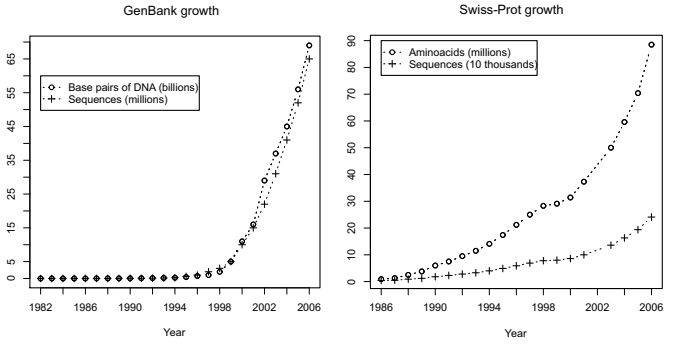


Fig. 1. Nucleotide and protein databases growth

2 Similarity Search in Sequence Databases

When searching for similarity between two sequences, there are three methods:

- analysis of the dot matrix
- dynamic programming solution
- heuristic methods using q-grams

All these methods compute a kind of biological similarity between two sequences. The similarity of two sequences is defined by an optimal *alignment* of them. An alignment of two[3] sequences is every such correspondence between letters in both (sub)sequences (one written under the other), which can be achieved by inserting gaps into either sequence. Naturally, among all possible alignments we are interested in an *optimal* one – which is the alignment with best real-value score (interpreted as sequence similarity). A particular method of scoring

[3] There can be also multiple sequences mutually aligned for some purposes, but this is out of scope of this paper.

an alignment changes with method used (see next subsections), however, it is always a sum over the scores between aligned letters/gaps.

Consequently, the similarity assessment (score of alignment) can be used in retrieval of sequences from a database similar to a query sequence.

2.1 Dot Matrix Method

A dot matrix (dotplot) analysis is a method for finding possible alignments of two sequences. In the dot matrix analysis we mark each of the axes with one amino acid sequence and then mark each position in the matrix where the two sequences match (have the same letters on given positions). Matching segments can then be seen as diagonal rows of dots going from left (see Figure 2).[4]

This algorithm can be improved by use of sliding windows of given length[5] (see Figure 2a,b). The window is successively placed to every position, while the window on every position is checked for a diagonal occurrence. If the number of matching letters in both sequences is above a fixed threshold (possibly according to a scoring matrix, see below), then a dot is placed on the position of the upper left corner of the window. Subsequently, short diagonals in the resulting plot can be filtered out for better transparency(Figure 2c).

Afterwards, the dot matrix is visually checked for diagonals long enough to be used as candidates for even longer alignment. Pairs of sequences showing sufficient similarity are then further inspected with some other techniques (e.g. tools using dynamic programming).

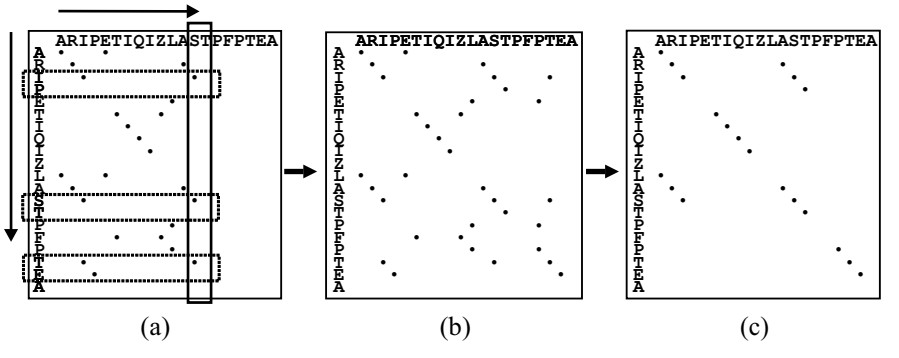


Fig. 2. Dot matrix

The dot matrix method is appropriate for visualisation and manual analysis of partial alignments of two sequences, but it does not give us the optimal alignment – especially if an alignment with gaps is our goal. To solve this problem, there exist algorithms based on dynamic programming which can compute optimal alignment with gaps in $O(mn)$ (where m, n are sequence lengths) time.

[4] This method can also be used to finding reverse matchings (diagonals go from right to left in that case).

[5] When comparing nucleotide sequences, longer windows are used.

With minor changes one can use the same algorithm for finding both, global and local alignment. A *global alignment* of sequences is such an alignment where the whole sequences must be aligned, whereas a *local alignment* methods look for best alignment for any subsequences of the input sequences.

First of all we need to define a distance to measure similarity between two sequences (possibly by use of an alignment). The simplest is the *Hamming distance* (HD) which is defined on two strings of equal lengths, and computes the number of positions for which the corresponding symbols are different. Because HD is a metric, it is often used when indexing sequences split to q-grams.

The Hamming distance is not suitable because it is sensitive to shifts in sequences. Therefore, a better measure is the *Levenshtein* (or edit) distance given by the minimum number of operations needed to transform one sequence into the other, where an operation is an insertion, deletion, or replacement of a single letter. The Levenshtein distance can be further extended with operation weights. Every replacement, insertion or deletion can be penalized by a constant value. Such an extended edit distance is called *weighted edit distance*.

However, when considering DNA or protein sequences, even weighted edit distance is not suitable. The reason is that probability of mutations from one amino acid into another (appearing in an alignment) is dependent on the two particular amino acids involved (the same holds for nucleotides but this is not so often used, since there are only four of them). To deal with this fact, a *scoring matrix* is used which is a 20×20 square matrix where on position $[i, j]$ is the so-called weight of mutation of amino acid i into amino acid j (if we use identity matrix we get an ordinary edit distance). There are many different scoring matrices used for different purposes (e.g. PAM matrices [7], BLOSUM matrices[9], etc.). The scoring matrices used in bioinformatics contain scores interpreted as similarity of two amino acids/nucleotides, so instead of distance measures we speak about similarity measures (where the greater overall scores stand for greater overall similarity). Hence, as opposite to Hamming or edit distance, an optimal scored alignment is the one with highest overall score (see Figure 3).

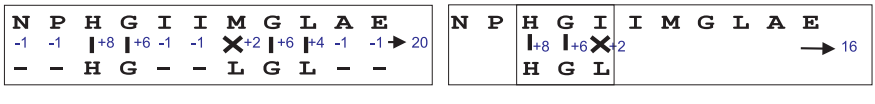


Fig. 3. Global and local alignment (and their scores) of protein sequences NPHGIIMGLAE and HGLGL according to BLOSUM62 scoring matrix

Global Alignment Measures. As mentioned earlier, there exist algorithms for both, global and local alignment (see Figure 3). The first algorithm for global alignment was published in 1970 by Needleman and Wunsch [13]. It makes use of distance matrix where on one side of the matrix is the first sequence and on the other the second one. We define a cell $s_{i,j} = [i,j]$ of the matrix as the optimal (maximal) score which belongs to prefixes of lengths i and j of the aligned sequences. The recursive formula for filling distance matrix's cell is:

$$s_{i,j} = max \begin{cases} s_{i-1,j} + \sigma \\ s_{i,j-1} + \sigma \\ s_{i,j} + \delta(a_i, b_j) \end{cases} \quad (1)$$

where a and b represent the sequences to be aligned, σ is a score for gaps and δ is scoring matrix. Since $[i, j]$ contains the score of global alignment of the i-long prefix of a and j-long prefix of b, the cell $[|a|, |b|]$ contains the alignment score for the whole sequences a, b.

Local Alignment Measures. Since we are more interested in matching subsequences, the local alignment is applied more often in computing similarity between biological sequences[6]. Local alignment of sequences a and b is finding a subsequence $\mathrm{S}(a)$ of sequence a and a subsequence $\mathrm{S}(b)$ of sequence b, such that global alignment of $\mathrm{S}(a)$ and $\mathrm{S}(b)$ provides the highest score. The algorithm for local alignment can be seen as a modification of Needleman-Wunsch and was published in 1981 by Smith and Waterman [20]. It modifies filling of the distance matrix by not allowing negative values. The optimal local alignment score is then the maximum value in the distance matrix, which means there is no way to increase the score with extending (or cutting) either of the two subsequences. The recursive formula is adjusted in the following way:

$$s_{i,j} = max \begin{cases} 0 \\ s_{i-1,j} + \sigma \\ s_{i,j-1} + \sigma \\ s_{i,j} + \delta(a_i, b_j) \end{cases} \quad (2)$$

Finally, there is usually another modification which enables differentiating between the *opening* and *extending* gap. Extending a gap is considerable less penalized than opening a gap (i.e. two single gaps in alignment are more penalized than a single two-letter gap).

2.2 Heuristic Approaches to Retrieval of Similar Sequences

The Smith-Watterman (SW) algorithm gives the optimal solution to local alignment but it is computationally expensive. Hence, there have been developed cheaper heuristics which approximate the optimal local alignment. The first wide-spread method of this type was the *FASTA* [15], which used short local alignments as *seeds* that were further extended on both sides. Nowadays, *BLAST* (Basic Local Alignment Tool) [3] algorithm is widely used, because in most cases it is faster then FASTA and gives better results (suffers from less false dismissals with respect to sequential retrieval using Smith-Watterman alignment). In short, the BLAST algorithm can be described as follows:

1. Remove low complexity regions from the query sequence (those with no meaningful alignment).

[6] Global alignment could damage alignment of perfectly conserved domain.

2. Generate all n-grams substrings of length n from query sequence.
3. Compute the similarity for every sequence of length n (on a given alphabet) and each n-gram from the previous step.
4. Filter out sequences with similarity lower than a cut-off score (called neighborhood word score threshold).
5. Remaining high-scoring sequences (organized in search tree) are then used to search all database sequences for exact match.
6. High-scoring sequences within a given distance (those on the same diagonal if we imagine sequences in dot matrix) are then connected together with gapped alignment and these are being extended as long as the score grows[7]. Such alignments are called *high scoring pairs* (HSP).
7. All HSPs with scores below a given threshold are excluded.
8. The scores of non-filtered sequences are refined by the classic Smith-Watterman algorithm.

Statistical Relevance. Because of the heuristic nature of BLAST, it is important to have a method saying how *good* the found hits are (however, there is no way to determine the retrieval error, say precision/recall in IR terminology). We are also interested in determining whether the retrieved alignments are some coincidence, or whether they really refer to a biological relation. The local alignments are statistically well understood, so we are able to say how relevant the retrieved sequences are. This statistic evidence fits primarily to ungapped local alignment, but it has been shown (mostly by computational experiments) that it applies to gapped alignments, too.

We are interested whether the resulting score (either Smith-Watterman (SW) or any other) has any statistical significance (otherwise the score alone is just a number), hence, we have to take the score distribution into account, considering the entire sequence space and a particular scoring matrix. We can derive an expected number of sequences of lengths m and n with score at least S as

$$E = Kmne^{-\lambda S} \tag{3}$$

where K and λ are characteristics of the SW score distribution. The E formula is called the *E-value* for the score S [10].

The *E-value* as defined above does not take into consideration dealing with more than two sequences, however, we should additionally take into account the fact that probability of finding a sequence with given score depends also on the number of sequences in the database or/and the query length. In order to prevent the effect that finding a short sequence with given *E-value* has the same probability as finding a longer sequence with the same *E-value*, we should multiply the *E-value* by the total number of residues (letters) in the database and obtain a new *E-value* (used by FASTA). Alternatively, instead of multiplying the *E-value* by database size, we could adjust the *E-value* by distinguishing different sequence lengths, since a query is more likely to be related to a long sequence than to a shorter one (used by BLAST).

[7] This applies to BLAST2 - previous versions of BLAST did not connect sequences on diagonals (and therefore used higher value for cut-off score in step 4).

3 Metric Sequence Indexing & Search

Since a particular method of sequence alignment can be viewed as a non-metric distance/similarity measure, we had an idea to turn it into a distance metric δ, which satisfies the metric properties (reflexivity, non-negativity, symmetry and triangle inequality). Such a metric could be then utilized by various *metric access methods* (MAMs) which have been designed to quickly search in databases modeled in metric spaces [24]. Their common characteristics reside in utilizing the triangle inequality to organize the database into metric regions, so that when a query is processed (the metric space is queried for objects falling into the query region), only the overlapping regions need to be searched. Basically, the MAMs are designed to support *range query* and *k-nearest neighbors (kNN) query*. A range query is defined by a center object and a query radius, so we ask for objects which are within a predefined distance from the query object. On the other hand, a kNN query asks for k nearest objects to the query object. Both range and kNN queries are represented by ball query regions (the radius of kNN ball is the distance to the k nearest neighbor), thus the searching by MAMs is reduced to a geometric problem (searching for database objects falling into the query regions/balls).

3.1 Creating the Metric

First, we have to decide which sequence similarity measure should be used. The simplest solution would be the Smith-Waterman (SW) score. However, the problem with SW is that it is similarity measure, not a distance (greater scores mean higher similarities). Another problem is that SW lacks statistical significance. Therefore, the use of *E-value* is much better, since it is a distance and it is also widely used in ranking the results retrieved by FASTA or BLAST. Due to this choice, we can also compare our approach to the other methods.

If we want to use E-value for metric indexing, we need an equivalent metric. As defined, the original E-value satisfies just the non-negativity property, but it can be easily modified to satisfy also reflexivity and symmetry. To enforce reflexivity, the definition of E-value can be modified in a way that identical sequences have a zero E-value. This is not a problem because two identical sequences are also most similar in mathematical and biological meaning (no other sequence can be more similar). To satisfy the symmetry, we have to accomplish a more substantial change. As mentioned in Section 2.2, the BLAST's modification of E-value takes into account the total number N of residues (letters) in database and the length n of inspected database sequence, as follows:

$$E = Kmne^{-\lambda S}N/n \tag{4}$$

If we change the role of query and database sequence, we get different values. So we changed E-value to the following form:

$$E = Kmne^{-\lambda S}N/max(m,n) \tag{5}$$

where m is the length of query sequence. This is a minor change in statistical relevance if we realize that an average length of query is very similar to the average length of database sequences. As mentioned before, e.g. FASTA does not take sequence lengths into account at all.

TriGen Algorithm. Up to now, we have turned the E-value into a semi-metric (reflexive, non-negative, symmetric distance). The last step is to enforce the triangle inequality, which is the hardest part. For this purpose we can use the TriGen algorithm [17] designed for general-purpose modifications of unknown semi-metrics into metrics (or approximations of metrics). Moreover, using TriGen we can specify (by so-called *T-error tolerance*) to what extent the resulting modified distance may violate the triangle inequality. While distances modified into full metrics guarantee 100% precision of indexing by MAMs with respect to the precision of sequential search, a modification which is only an approximation of metric (more or less violating the triangle inequality) exhibits lower *intrinsic dimensionality* [5] (i.e. better indexability) at the expense of lower precision of indexing/retrieval. In principle, the TriGen algorithm applies a class of concave modifiers such that the input semi-metric (violating triangle inequality) becomes a metric (or semi-metric where the extent of triangle violation is reduced).

3.2 LAESA

The LAESA method [12] is a typical pivot-based MAM. In principle, m database objects are selected to act as so-called *pivots*. Then each of the n database objects to be inserted is mapped using this pivots into a vector of dimension m, where into i-th coordinate the distance of the inserted object to the i-th pivot is computed and stored. This way we obtain an $n \times m$ distance matrix which serves as a metric index.

Whenever a query is to be processed, the query object is projected into the pivot space the same way as if it would be inserted. Then the distance matrix is sequentially searched and all the vectors which do not overlap the query region in the pivot space are filtered out. The remaining candidate objects (corresponding to the non-filtered vectors) are subsequently filtered in the original metric space.

The LAESA method is very powerful in its pruning effectiveness, however, due to expensive selection of pivots and due to the sequential processing of distance matrix its usage in dynamic database environments is limited.

3.3 M-tree & PM-tree

A typical tree-based MAM designed for database environments is the M-tree [6]. The concept of M-tree is a kind of generalization of R-tree into metric spaces. Instead of R-tree's MBRs, the M-tree recursively bounds the objects into balls specified by a center data object and a ball radius (unlike R-tree the M-tree cannot create a synthetic centroid, it must pick one of the indexed objects). The inner nodes of an M-tree index contain *routing entries*, consisting of a region ball

and a pointer to the subtree (all objects in a subtree must fall into the parent region ball). The leaf nodes contain *ground entries* – the DB objects themselves. For an M-tree hierarchy see an example in Figure 4a.

A range query in M-tree is processed by processing just the nodes the region ball of which overlaps the query ball. A kNN query processing is similar, however, the radius of query ball is unknown at the beginning, so it must be heuristically updated during query evaluation.

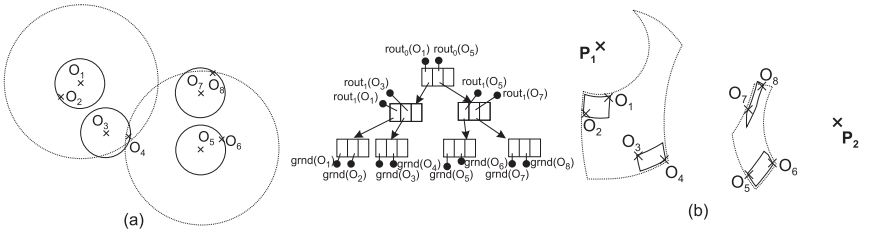


Fig. 4. (a) M-tree (b) PM-tree

As a combination of LAESA and M-tree, the *PM-tree* [16, 19] makes use of both, LAESA's pivot-based indexing and M-tree's hierarchical metric space partitioning. The main difference is that the routing entries contain also a set of $m_1 \leq m$ rings (related to m_1 of total pivots) which prune the region ball, thus, the total "volume" of a PM-tree's data region is always smaller than an equivalent M-tree region (see Figure 4b). The ground entries are extended by rings as well, but in this case the number of rings used is different ($m_2 \leq m$). By specifying $m = m_1 = m_2 = 0$ we get an ordinary M-tree.

Slim-down Algorithm. A particular hierarchy of M-tree's (or PM-tree's) nested data regions can be far from optimal, which means the volumes of regions can be very large, and they can also overlap significantly. This leads to poor search efficiency, since the larger volumes/overlaps the greater probability that a data region and the query region will overlap. To prevent such poor hierarchies, the *generalized Slim-Down algorithm* [18] has been developed to optimize an already built (P)M-tree index. Although the "slimming down" is an expensive operation, it can speed up the subsequent querying up to 10x.

4 Experimental Results

As dataset we used random subset of the Swiss-Prot database of size 3000 with total number of 1041027 amino acids. Another random 100 hundred sequences have been chosen as query sequences.All of the sequences were of maximal length 1000 which is doesn't cause any problem when we realize that there are only 9191 longer sequences out of 252616 in whole Swiss-prot which makes 3% (average sequence length is 365 in whole Swiss-prot and 335 in the reduced variant). These longer sequences could then be treated in special way since they are just small

part of the whole.
We don't show time comparation in our tests because we do not have effecitive implementation of Smith-Waterman yet, being the crucial component of the running time. But our method could be easily compared to SSEARCH (part of FASTA package) when we realize that SSEARCH is equivalent to sequence scan. To be able to compare index based methods with BLAST we distinguish number of distance computations from computational costs. We defined computational costs here as number of comparing two letters. Therefore computational cost of tree based methods are averaged as number of distance computations multiplied by the average size of distance matrix for Smith-Watterman which is $321 * 321 = 103041$. For possibility of comparing index-based methods with sequentional scan, we show number of distance computations too.
Computational costs of BLAST can be devided from description in section 2.2. Average number of neighbouring words for our dataset of query sequences is 54 and takes 81784 operations (matching letters) to get them [8] , which gives us 245352 computational operations. This means that there was built search tree with 54 items. Such a tree has aproximatly 6 levels. When searching for seeds for the high scoring pairs, there have to be done 6 comparations with every position of the database. Since every comparation of word with database costs 3 operations, the computation costs of BLAST in average are $6 * 1041000 * 3 = 1873800$ operations. Number of operations done while finding neighbouring words and finding high scoring pairs is insignificant in comparation with this number so we do not take it into account. Therefore we state that in average case, computational costs of BLAST are 18738000 operations[9].

Four indexing methods were tested - M-tree, PM-tree, slimmed PM-tree and LAESA, each of them using the same set of distance modificators [10] genereted by the TriGen algorithm. As can be seen in Figure 5a, when the zero error tolerance is used, weight of the modificator causes that number of distance computations is almost equivalent to sequence scan. When the weight is too big, it makes trinagle inequvality hold but for the price of increasing intrinsic dimension. However M-tree performes slightly better then sequence scan which means that when searching in the tree, not all of the nodes have been inspected (because of inner nodes, number of objects in the tree exceeds 3000). On the other side, PM-tree and slimmed PM-tree show worse result since there are aditional computations to pivots (mapping the query). The most similar to sequential scan is LAESA method, which uses constant number of distance computations independently on range of the query.

The situation slightly changes when we allow small error (Figure 5b). This causes that number of distance computations decrease about six percent compared to zero error tolerance. Here, PM-tree and slimmed PM tree outperform M-tree and the difference is about 1%. On the other hand LAESA showed just

[8] found out empirically
[9] We do not consider possible filtering of nonsignificant segments.
[10] fractional power modificators were used

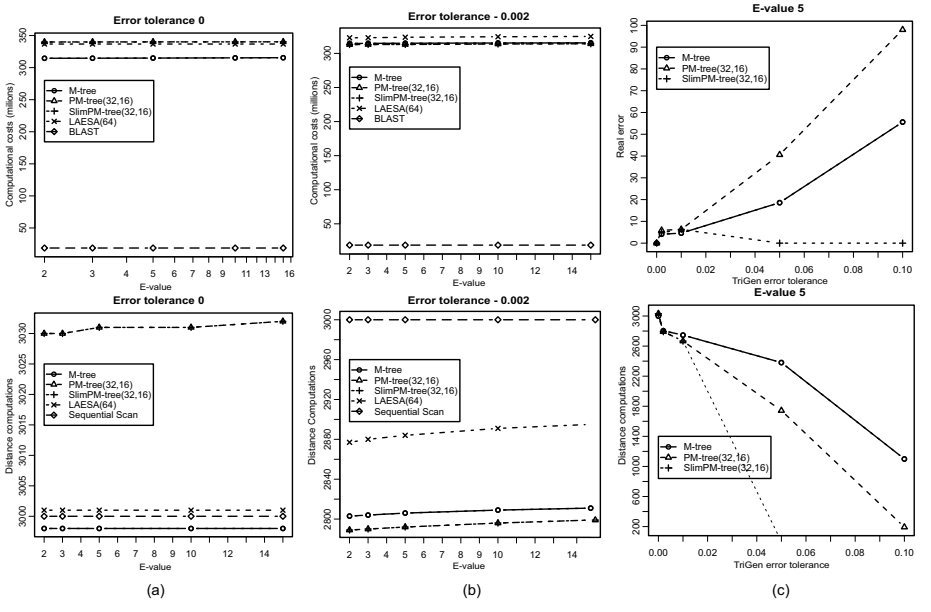


Fig. 5. Relation between E-value and number of computations - range query (a,b) and realation between real error and distance computatins (c)

slight improvement. But in both cases BLAST method is evidently more effective since effctivity of the index is almost sequence scan even if small error is allowed.

Why PM-tree and slimmed PM-tree behave better when allowing some error? Answer to this question can be seen on Figure 5c which shows on range query of E-value five the relation between declared TriGen error tolerance and real error experienced in test. Here we can see, that PM-tree real error growth more quickly than the error of M-tree and moreover, we can see that slope of those lines are almost inverse, which means that the distance computations gain of PM-tree is counterbalanced by the error. From these two graphs can also be seen that for real error 50%, there still have to be done approximately 1200 distance computations which is about 120000000 computational operations. That means that if BLAST would be such a bad hauristic that it would have just 50% successfulness, it still would be noticeable more effective.

5 Conclusions

In this paper, we have have tested suitability of metric access indexing methods for indexing protein sequences. It has been shown that these method are not applicable to sequence alignment problem without their modification. This is primarily because of quality of the data to be indexed and the distance function which is used to define similarity between them. This distance function is

highly non-metric which demands strong modifications to it, to make it metric. This modification distorts distances in a way that strongly increases intrinsic dimension of the data and therefore the efficiency of search is almost the same as efficiency of sequential scan. But against sequentional scan it has that advantage that precission can be defined and thus traded off for efficiency.
This learned facts can aim next resarch to several areas. To name a few:

- Examining TriGen modificators and finding such modificators, which would minimize real error while distributing objects (i.e. sequences) in the space in a way which will be appropriate for indexing methods (i.e. descreasing intrinsic dimension).
- Modifying the search structures. For example examining possiblities of cutting sequences to q-grams but being able to define arised error (caused by splitting and thus losing information included in the whole sequence) and (optimaly) minimalize it.
- Modifying computing of Smith-Waterman local alignment. The idea is to change computing so that it will be faster and resulting scores won't violate properites of metric so much, as they do now (for example by using borders to limit the computational space in the distance matrix).

Acknowledgments

This research has been partially supported by GAČR grant 201/05/P036 provided by the Czech Science Foundation. We would like to thank prof. Fatima Cvrčková (Department of Plant Physiology, Faculty of Science, Charles University in Prague) for helping to get in touch with biologist point of view to the problem and contributing to the paper with helpful comments.

References

1. DNA DataBank of Japan. `www.ddbj.nig.ac.jp`.
2. European Molecular Biology Laboratory Data. `www.embl.org`.
3. S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic Acids Res.*, 25:3389–3402, 1997.
4. A. Bairoch, B. Boeckmann, S. Ferro, and E. Gasteiger. Swiss-prot: Juggling between evolution and stability. *Brief. Bioinform.*, 5:39–55, 2004.
5. E. Chávez and G. Navarro. A Probabilistic Spell for the Curse of Dimensionality. In *ALENEX'01, LNCS 2153*, pages 147–160. Springer, 2001.
6. P. Ciaccia, M. Patella, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *VLDB'97*, pages 426–435, 1997.
7. M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt. A model for evolutionary change in proteins. *Atlas of Protein Sequence and Structure*, 5:345–352, 1978.
8. D. B. et al. Genbank. *Nucleic Acids Res.*, 34(Database issue):D16–D20, 2006.
9. S. Henikoff and J. Henikoff. Amino acid substitution matrices from protein blocks. *Proc. Natl Acad. Sci. USA.*, 89:10915–10919, 1992.

10. S. Karlin and S. Altschul. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. In *Proc. Natl. Acad. Sci.*, volume 87, pages 2264–2268, 1990.
11. R. Mao, W. Xu, S. Ramakrishnan, G. Nuckolls, and D. Miranker. On optimizing distance-based similarity search for biological databases. In *Proc IEEE Comput Syst Bioinform Conference*, pages 351–61, 2005.
12. M. L. Micó, J. Oncina, and E. Vidal. A new version of the nearest-neighbour approximating and eliminating search algorithm (AESA) with linear preprocessing time and memory requirements. *Pattern Recognition Letters*, 15(1):9–17, 1994.
13. S. Needleman and C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular biology*, 48(3):443–453, 1970.
14. Z. Ning, A. Cox, and J. Mullikin. Ssaha: a fast search method for large dna databases. *Genome Research*, 11(10):1725–1729, 2001.
15. W. Pearson and D. Lipman. Improved Tools for Biological Sequence Analysis. *Proc. Natl. Acad. Sci.*, 85:2444–2448, 1988.
16. T. Skopal. Pivoting M-tree: A Metric Access Method for Efficient Similarity Search. In *Proceedings of the 4th annual workshop DATESO, Desná, Czech Republic, ISBN 80-248-0457-3, also available at CEUR, Volume 98, ISSN 1613-0073,* `http://www.ceur-ws.org/Vol-98`, pages 21–31, 2004.
17. T. Skopal. On fast non-metric similarity search by metric access methods. In *Proc. 10th International Conference on Extending Database Technology (EDBT'06)*, LNCS 3896, pages 718–736. Springer, 2006.
18. T. Skopal, J. Pokorný, M. Krátký, and V. Snášel. Revisiting M-tree Building Principles. In *ADBIS, Dresden*, pages 148–162. LNCS 2798, Springer, 2003.
19. T. Skopal, J. Pokorný, and V. Snášel. Nearest Neighbours Search using the PM-tree. In *DASFAA '05, Beijing, China*, pages 803–815. LNCS 3453, Springer, 2005.
20. T. Smith and M. Waterman. Identification of common molecular subsequences. *Jurnal of molecular biology*, 147:195–197, 1981.
21. C. Weimin and K. Aberer. Efficient querying on genomic databases by using metric space indexing techniques. In *Proceedings of the 8th International Workshop on Database and Expert Systems Applications*, page 148, 1997.
22. C. Wu, R. Apweiler, A. Bairoch, D. Natale, W. Barker, B. Boeckmann, S. Ferro, E. Gasteiger, H. Huang, R. Lopez, M. Magrane, M. Martin, R. Mazumder, C. O'Donovan, N. Redaschi, and B. Suzek. The universal protein resource (uniprot): an expanding universe of protein information. *Nucleic Acids Res.*, 34(Database issue)(1):D187–D191, 2006.
23. W. Xu and D. Miranker. A metric model of amino acid substitution. *Bioinformatics*, 20(8):1214–1221, 2004.
24. P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search: The Metric Space Approach (Advances in Database Systems).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

Using BMH Algorithm to Solve Subset of XPath Queries

David Toth

Dept. of Computer Science and Engineering,
FEE, Czech Technical University,
Karlovo Náměstí 13, 121 35 Praha 2, Czech Republic
tothd1@fel.cvut.cz

Abstract. Boyer-Moore-Horspool (BMH) algorithm is commonly used to solve text searching problems. In this paper is used to solve the constraint subset of XPath queries offering effective algorithm to resolve such queries. XML can be grasp as text file contains tags and its content; that kind of view to XML is used in this work. We constraint XML document content and possible XPath queries. This work focus on key ideas and problems appertaining XPath queries execution using text search algorithm BMH.

1 Introduction

Many papers were published appertaining the XPath query effective execution issues. Many authors concern many aspects of XML and XPath. In this paper we look in detail how to solve part of XPath queries involving only path of nodes in the query but not any functions. Another extensions are possible and in future works can be exploited in more details; see into Conclusions and future works for more information.

Why bother with XML and XPath? XML is special text file format, meta-language, markup language (using tags instead of commands); XML can be represented by a tree where nodes represent tags and connections between nodes represent relationship super-element and embedded element. XML can be used for many reasons. Especially useful is using XML as data interchange format in internet environment. It is conceived as a tool assuring compatibility between different applications possibly running on different platforms and using different inherent data paradigm and formats. Today in B2B applications is XML widely used.

Of course XML has many other purposes. For example in XML documents can be stored content of any magazine or generally paper. Such kind of XML documents are known as document oriented XML; i.e. there is huge amount of text contrary to data oriented XML documents where this is not generally true. In data oriented XML documents are many tags and often as much as data itself. Finally the tag information is itself also valuable information. Document oriented XML are typically intended to use by humans in opposite to data oriented XML documents which are normally designated for computers.

Here we try to prove that whenever data oriented XML documents are supposed we can treat them as plain text files finding queried node(s) using BMH algorithm. Only when the amount of text representing structure information (for simplicity let us consider just tags) in XML file be the same order of magnitude as amount of text representing stored data itself. This is big assumption which has to be preserved to can legally speak about effective XPath query resolution.

The next section introduce basic BMH algorithm principle and show example of using this algorithm. Another section acquaint XPath queries and selected subset further elaborated. After come section named BMH in XPath where are introduced and explained all considered XPath query execution algorithms based basically on BMH from concrete queries to generic ones. Finally in Conclusions and future works section assess gained results and procedures and outline extensions and future works directions.

2 BMH Algorithm

After short introduction here we will treat BMH algorithm basics.

Here we deal with BMH (Boyer-Moore-Horspool) algorithm commonly used to solve text searching problems. The basis of this approach inhere in a little preprocessing of searching pattern and then searching the text forward but backwards from pattern. That is the trick. That is how can be raise effectivity. There can be hopped (or jumped) many irelevant letters (or generally parts of input stream). The number of reading is therefore typically much more lesser than whole length of stream..

2.1 Preprocessing issues

What must be done during preprocessing phase? For any pattern there is a need for create hopping table where must be pre-counted how many letters (or input symbols) can be jumped after input sign and the appropriate sign from pattern has been read and they do not match.

Let us show an example. Suppose input stream as: „Where are you going young man?“ Furthermore assume we trying to find pattern: „you“.

The answer obviously is that the pattern is presented 2 times 1st at position 11 and 2nd at position 21. Naive approach lead us to read whole text i.e. 31 characters and do at least the same number of comparisons.

When we try to use BMH we have to resolve how much can be hopped when some letter is achieved. The preprocessed table should look like this:

y o u x
2 1 3 3

Which means that whenever compare of input letter and pattern letter do not match we should jump ahead of 2, 1 or 3 characters depending on what letter is on input

what is talking about the 1st line in the above table. Note that x mean any other symbol than enumerated before x – in our case whatever symbol but y, o, u.

So the above example will be, using BMH algorithm, take just 16 readings.

2.2 How BMH algorithm operate?

First necessary variables:

I	position in input stream
pat	pattern

2.3 Semiformal BMH description (BMH algorithm)

1. preprocessing – the tab is calculated
2. begin
3. Go to the I where I = len(pat)
   (initial position in the input text is set to the length of pattern)
4. repeat (until in input stream the end is not reached)
5. compare letters backwards from position I with pattern until they fit each other
6. when pattern is matched then
   1. save the position of 1st matched occurence – store I into the array
   2. go to the position of possible next occurence which is: I = I + 2*len(pat)
7. (ELSE) when pattern is not matched then
   1. go ahead to the fartest position possible which is I = I + tab[K], where K is letter of input text where was expected any different symbol
8. end.

Algorithm notes. len means function returning length of its string argument and tab preprocessed table containing length of possible hops for all letters.

Useful source where formal BMH algorithm description is introduced, explained and described using an example is for example [2].

After BMH algorithm basics were explained next section deal with XPath query which we will subject of next investigation.

3 XPath queries

Now after the slim repetition of how BMH algorithm runs we will focus on XPath queries. This section deal with investigated XPath query subset which is briefly introduced.

XPath is derived from XML Path which suggests us it is about determining paths in XML documents. In fact it is more complicated. Functions and other features must be considered furthermore. But for the purpose of this paper we focus on simple queries involving just nodes and paths.

Generally there are two possibilities how to determine path using XPath. So called relative paths and absolute paths. Absolute path always syntactically begins with the sign for root of the XML document which is denoted as '/'. Absolute path is inevitably derived from the root of the document. Result of such query can be none node found, exactly one node in specified path or set of nodes of same name in specified path. Relative queries on the other side mean that somewhere in the query this sequence '//' is placed which means no matter of deep where the node will be found. Such kind of a query is practically eminently useful but it is this kind of queries which means problems with effectivity. XPath queries examples follow.

First absolute queries:

/rootnode/node1/node3 – the result are all nodes appertaining <node3> elements which are descendants of <node1> element which are descendants of <rootnode> element.

/rootnode/node6 – the result of this XPath query are all <node6> elements which are descendants of <rootnode> element.

Let us see now relative queries:

//node2 – such XPath query gives all <node2> elements in valid XML document.

//node4/node5 – the answer is: all <node5> elements whichs parents are <node4>.

And finally we will look at mixed queries:

/rootnode/node7//node8/node9 – the return all <node9> elements whichs parents are <node8> elements placed however deep inside the element <node7> which has to be direct descendant of <rootnode> element.

Many useful properties of XPath were omitted. We will treat about extensions in part Conclusions and future works. XPath query somehow specify the path in tree structure of XML document. We will not need this imagination anymore. In this paper linear conception of XML (XML viewed as a text file) will be sufficient. All about XPath can be found in [4].

After XPath query subset was delimited we will in next section focus on how to this subset can be implemented using BMH algorithm.

4 BMH in XPath

Now after we briefly examined BMH algorithm and supposed XPath queries we will look at how to use BMH when evaluating XPath queries.

First the easiest case. Let us consider the following XPath query: //nodeX. So the result of such a XPath query should be all elements named nodeX wherever inside the XML document.

Of course we suppose the input XML document is well-formed XML document as this term is commonly understand according to [3]. Furthermore we do not consider CDATA section inside the XML document but it is not hard to imagine its subsequent incorporation. But here it would break our deliberations and diverge to another

than key problems. Except CDATA we also do not consider comments in XML. Both CDATA and comments can be easily incorporated to the proposed algorithms but it is not the aim of this work to focus on every aspects. Rather we focus on key principles. See the section Conclusions and future work for summarizing involving issues.

4.1 Semiformal simple algorithm description (simpleBMH)

1. use BMH algorithm to find asked pattern on input stream
2. for all found patterns must be verified the context
    1. just a word in text inside an element but do not denote an element (if it is the case then nothing happen, go further without writing into the results; this test can be done comparing symbols before the found pattern and symbol '/' or '</')
    2. it denotes an element and then there are two options
        1. it is opening tag
           (we should write the result into the output – position when the element starts; opening tag it is if precede symbol '<')
        2. it is closing tag
           (we should write the result into the output – add the closing position to the start position; closing tag it is if precede symbols '</')

The result will create pairs of beginning and ending positions individual element found in the text. Here we can see the power of XML grasped as plain text file.

Now we can search nodeX in XML document using XPath very simple notation: '//nodeX'. How long does this take? In order of magnitude as BMH itself. There is only one or two comparisons more when pattern match and one possible writing into the results (depending on if it is beginning or closing tag). So the performance will be comparable as the performance of BMH itself which is described in detail and with experimental results for example in [1].

Let us see how will the algorithm change when we focus on XPath queries using absolute path in the beginning of the query. We will suppose such a query: /rootnode//nodeX

4.2 Semiformal algorithm description for absolute paths only (apo-algorithm)

1. the beginning of input stream must exactly match '<rootnode'
   (as in the case of previous rumination)
   Note to the following point: (in fact we will use brute force to jump over the rootnode element attributes definitions and the possible content up to the first occurrence of any element inside rootnode)
2. repeat until the end of XML is reached
3. using brute force: find name of following element (second in absolute order in first iteration) in text (in XML document)
    1. if it is the element we looking for then
        1. into the result write the beginning mark

2. apply algorithm BMH (including preprocessing) with elements' end tag as pattern content; algorithm stop when first occurrence was found
3. into the result write the ending mark of this occurrence

2. else: apply algorithm BMH (including preprocessing) with elements' end tag as pattern content; algorithm stop when first occurrence was found
(no writing to the results; only hopping is realized)

Now should be presented some mechanism for generic kind of query. We should use previously elaborated, experienced and tested work. So we will use preceding algorithms. Let us consider more generic query as: /rootnode/node1/node2/node3/nodeX.

4.3 Semiformal generic algorithm for only absolute paths (gab-algorithm)

1. do an extraction of node names into the field of names: nodext
2. let us consider L variable where will be increased the relative level of depth depending on query structure; L = 2
3. let us consider MAX as number of nodes in path in query
4. the beginning of input stream must exactly match '<rootnode'
5. repeat until the end of XML is reached
6. using brute force: find name of following element in text (XML document) or end tag nodext[L-1]
   1. if it is the element we looking for (i.e. nodext[L]) then
      1. L++ (increase L)
      2. if L = MAX then
         1. into the result write the beginning mark
         2. apply algorithm BMH (including preprocessing) with elements' end tag as pattern content; algorithm stop when first occurrence was found
         3. into the result write the ending mark of this occurrence
      3. else:
         1. go to 6.
   2. if it is some other element
      1. apply algorithm BMH (including preprocessing) with elements' end tag as pattern content; algorithm stop when first occurrence was found
      (hopping non requested element)
   3. if it is reached end tag then
      1. L-- (decrease L)

Now we'll try to extend algorithm for long relative queries as for example is this: //node1/node2/nodeX.

4.4 Semiformal generic algorithm for only relative paths (ger-algorithm)

1. make extraction of nodes from query to the field: nodext
2. variable L determine the position in nodext – relative depth; L = 1

3. let MAX denote number of nodes in a query
4. use BMH to find all opening tags: nodext[L] and for every occurrence found run own thread continuing with 5.
5. use gab-algorithm from point 5

Finally we combine absolute path at the beginning and relative path in the end of query. Suppose query like this one: /rootnode/node1/node2//node3/node4/nodeX.

4.5 Semiformal generic algorithm (gen-algorithm)

1. make extraction of nodes from a given query into two fields
   1. nodextabs – containing nodes up to sign '//' and
   2. nodextrel – containing nodes from symbol '//' till the end of a query
2. variable L1 will serve as relative depth for first part
3. variable L2 will serve as relative deptg for second query part
4. MAX1 denotes number of nodes in first part of a query
5. MAX2 denotes number of nodes in second part of a query
6. use gab-algorithm with nodextabs, L1 and MAX1
   1. for every result use ger-algorithm with, nodextrel, L2, MAX2

Author hope this summarizing of thinking can help when inventing new procedures, algorithms and other ruminating related issues. Future works should include also implementation of proposed algorithms. About future works and conclusions concern following section.

5 Conclusions and future works

Most important made assumptions are these:

1. XML document is mainly data-oriented and do not contain CDATA sections.
2. XPath query do not contain anything else but node names and sign '/' or '//'.

We stated relatively strong preconditions. None of those should have great impact on key ideas presented here. All such assumptions should let us concentrate on key and main issues of discussed problem: how can be effectively used BMH algorithm to answer constraint subset of XPath queries. These preconditions are limited but can be step by step eliminated. This should be the aim of future works; choose little pieces of what is not yet implemented and incorporate it to the solution which let grow the picture of whole.

Much work can be done here on this field. Every supposition could be subdue to rational critics and proposed algorithm should be extend to encompass every such a conditions' solution.

All proposed algorithm variants should be experimentally examined as well as all possible following extensions and modifications.

One of future works should respect document oriented XML. There are several ways how to solve this problem. One of such could be invent special object structure a priory allowing jump over parts which should be hopped – variation of state space cutting algorithm.

Consequences of proposed algorithm for execution of XPath queries. Follow most important consequences when using a variant of proposed algorithms.

1. Only data oriented XML documents will be treated truly effectively because of there will be not present long text passages to be searched.
2. When node names containing just plain text are queried and document contain predominantly nodes containing mostly numbers, which is often the case of data-oriented XML documents, then the effectivity will dramatically increase – all numbers will be as a consequence of using BMH algorithm, hopped.
3. With longer queried tag name the proposed algorithm will be more efficient.
4. It has none or just little impact when the length of queried tag name has only 1, 2 or 3 characters.

On one side stated consequences can be grasped as drawbacks but when we focus on special area of data handling here may be especially useful to integrate proposed algorithm as for example when exporting numerical data for XSLT to expose them into the web. Majority of above stated consequences appertain this special case where could be proposed algorithm implemented.

References

1. Lecroq, T.: Experimental Results on String Matching Algorithms. Software—practice and experience, vol. 25(7), John Wiley & Sons, Ltd. (1995) 727–765
2. Lovis, C., Baud, R.H.: Fast Exact String Pattern-matching Algorithms Adapted to the Characteristics of the Medical Language. J Am Med Inform Assoc. 7(4) (2000) 378–391.
3. XML 1.0, W3C Recommendation of Extensible Markup Language (XML) 1.0 (Fourth Edition), http://www.w3.org/TR/REC-xml
4. XPath, W3C Recommendation of XML Path Language (XPath) Version 1.0, http://www.w3.org/TR/xpath

Shape Extraction Framework for Similarity Search in Image Databases

Jan Klíma and Tomáš Skopal

Charles University in Prague, FMP, Department of Software Engineering
Malostranské nám. 25, 118 00 Prague, Czech Republic
irenicus@volny.cz, tomas.skopal@mff.cuni.cz

Abstract. The task of similarity search in image databases has been studied for decades, while there have been many feature extraction techniques proposed. Among the mass of low-level techniques dealing with color, texture, layout, etc., an extraction of shapes provides better semantic description of the content in raster image. However, even such specific task as shape extraction is very complex, so the mere knowledge of particular raster transformation and shape-extraction techniques does not give us an answer what methods should be preferred and how to combine them, in order to achieve the desired effect in similarity search. In this paper we propose a framework consisting of low-level interconnectable components, which allows the user to easily configure the flow of transformations leading to shape extraction. Based on experiments, we also propose typical scenarios of transformation flow, with respect to the best shape-based description of the image content.

1 Introduction

Similarity search in image databases [10] is becoming increasingly important, due to rapidly growing volumes of available image data. Simultaneously, the text-based image retrieval systems become useless, since the requirements on manual annotation exceed human possibilities and resources. The metadata-based search systems are of similar kind, we need an additional explicit information to effectively describe multimedia objects (e.g. structured semantic description, as class hierarchies or ontologies), which is not available in most cases.[1] The only practicable way how to search the vast volumes of raw image data is the content-based similarity search, i.e. we consider the real content of each particular raster image (e.g. a photography), where the images are ranked according to similarity to a query image (the example). Only such images are retrieved, which have been ranked as sufficiently similar to the query image. The similarity measure returns a real-valued similarity score for any two models of multimedia objects.

Unlike other similarity search applications, the task of highly semantic content-based search in image/video databases is extremely difficult. Because of generally unrestricted origination of a particular raster image, its visual content is

[1] The image search at `images.google.com` is a successful example of metadata-based engine, where the metadata are extracted from web pages referencing the images.

© J. Pokorný, V. Snášel, K. Richta (Eds.): Dateso 2007, pp. 89–102, ISBN 80-7378-002-X.
VŠB – Technical University of Ostrava, Dept. of Computer Science, 2007.

not structured and, on the other side, hides rich semantics (as perceived by human). The most general techniques providing extraction of distinguished features from an image are based on processing of low-level characteristics, like color histograms, texture correlograms, color moments, color layout (possibly considered under spatial segmentation). Unfortunately, the low-level features emphasize just local/global relationships between pixels (their colors, respectively), hence, they do not capture high-level (semantic) features. In turn, usage of the low-level features in similarity search tasks leads to poor retrieval results, which is often referred to as the "semantic gap" [10].

In real-world applications, a design of high-level feature extraction is restricted to the domain-specific image databases. For instance, images of human faces can be processed so that biometric features (like eyes, nose, chin, etc.) are identified and properly represented. Although domain-specific image retrieval systems reach high effectiveness in precision/recall, they cannot be used to manage heterogeneous collections, e.g. images on web.

1.1 Shape Extraction

The shapes (contours, bounded surfaces) in an image could be understood as a medium-level feature, since shape is an entity recognized also by human's perception (unlike low-level features). Moreover, shape is a practical feature for query-by-sketch support (i.e. a query-by-example where the "example" consists of user-drawn strokes), where extraction of colors or textures is meaningless.

Shape Reconstruction. The most common technique is to vectorize the contiguous lines or areas in the raster image. Prior to this, the image has to be preprocessed still on the raster basis (edge detection [17], smoothing, morphologic operations, skeletonization, etc.). The subsequent raster-to-vector transformation step follows (e.g. a binary mask is vectorized into a set of (poly)lines).

Naturally, we are not done at this moment, the hardest task is to filter and combine the "tangle" of short lines (as typically produced) into several (or even single) distinguished major shapes (polylines/polygons). This involves polyline simplification [11], removal of artifacts, line connection, etc. The most complex but invaluable part of shape reconstruction should derive the prototypical shape which is approximated by the vectorized information obtained so far.

Shape Representation & Similarity Measuring. Once we have sufficiently simplified shapes found in an image, we have to represent them in order to support measuring of similar shapes. The polygonal representation itself is not very suitable for similarity measuring, because of high sensitivity to translation, scale, rotation, orientation, noise, distortion, skew, vertex spacing/offset, etc. More likely, the raw shape is often transformed into a single vector or time series [5, 7, 9], where the shape characteristics are preserved but the transformation non-invariant characteristics are removed. The time series representations are usually measured by Euclidean distance, Dynamic time warping [4, 7], Longest common subsequence [15].

1.2 Motivation & Paper Contributions

As overviewed above, the process of shape extraction (starting from the raster image and producing a vector or time series) is very complex task, where there do not exist general recommendations about particular transformation/extraction steps. In this paper, we propose a component-based framework allowing to encapsulate and connect various image/vector-processing algorithms. Hence, a network consisting of many components can be easily created, through which a particular shape extraction scenario is configured. Following this framework, we have also implemented a catalogue of basic components which, when properly connected, can provide an effective environment for shape extraction experimentation. Finally, we present several domain scenarios for shape-extraction based on experimental results.

1.3 Related Work

Although we are aware of many existing image processing libraries, most of them lack support for end-user dataflow configuration or vector processing.

"Filters" [1] is a library of image, video and vector processing functions, based on idea of configurable filters that perform various tasks. Dataflow configuration is obtained by hardcoding or via python scripts.

"JaGrLib" [12] is a 2D/3D graphics library primarily aimed for educational purposes. JaGrLib offers both XML and GUI oriented dataflow configuration, but currently has limited shape extraction capabilities.

2 IVP Framework

The idea of Image and Vector Processing Framework [2] is to separate objects that usually figure in image processing (color bitmaps, grayscale bitmaps, binary bitmaps, gradient maps, vectors, polylines, etc.) and algorithms which work with these objects on an *input* → *output* basis. Each algorithm can be considered as a black box that expects certain input data and produces a defined kind of output data. With this point of view the whole shape extraction application reduces to a network of algorithms that send data to each other. An example of the idea is depicted in Figure 1.

This gives a view into the IVPF design: it's advantageous to code and store algorithms separately, the role of the client application is to allow user to specify which algorithms should be used, and also their *output* → *input* dependence. In the final effect, many specialized applications can be implemented (configured respectively) at high application level, just by specifying the algorithms, their settings and mutual dependencies.

2.1 Interface & Components

Each particular component class encapsulating[2] an algorithm (as mentioned above) must implement the `IIVPFComponent` interface in such a fashion that all

[2] The framework has been implemented in .NET framework 2.0.

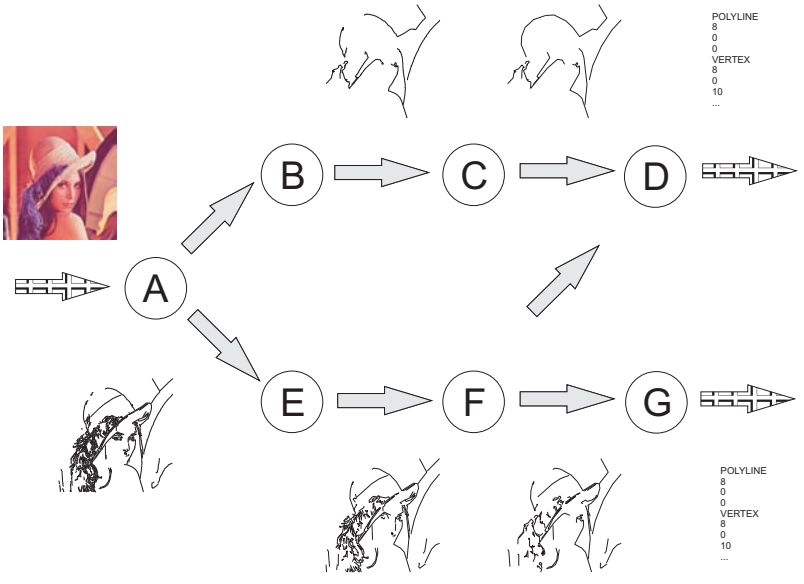


Fig. 1. Algorithms as black boxes: The example shows algorithms A-G which form an application with arrows representing their *output* → *input* dependence. A is an input algorithm that vectorizes the input image and sends the resulting vectors into the network. B-C and E-F-G are branches that transform their vector input somehow. D takes two vectorial inputs and chooses one, based on certain criteria. Then both D and G save their outputs in specified formats (to a file/stream/anything else).

public component features are accessible in a simple and transparent way. In particular, using `IIVPFComponent` interface the components are interconnected (port registration), and checked for compatibility.

Higher Level Functionality. At a higher application level, the components are just configured and their ports connected together to create a network. This can be done either via GUI or by loading configuration/connections from an XML file. During the whole network execution, all components are run starting from pure output components (no input ports) and propagating the intermediate data through the entire network to pure input components (no output ports). The whole approach enables to change the behavior of the network in two ways:

- by changing component(s) configuration
- by changing the structure of the entire component network

3 Component Catalogue

In this section we propose several components already implemented in the IVP framework. In order to ease the understanding, we use the following formal declaration of a component:

`type of input` → `component name (parameters)` → `type of output`

where the type of input/output we distinguish either `bitmap` (any 2D bitmap of color, intensity, gradient, etc.) or `vectors` (collection of polygons/polylines). The single arrow '→' means there is just a single type of connection to input/output supported, while the double arrow '⇒' supports several types of input/output[3]. The parameters declared in parentheses are component-specific, thus they have to be configured by the user.

3.1 Image Processing Components

The first class provides components serving as the basic-processing tools, which do eliminate resolution dependencies, filter the noise, and separate pixels within a specified range of colors.

Image Loader Component
`ImageFromFile (FileName)` ⇒ `bitmap` (of colors) + `bitmap` (of intensities)

To process an image, it must be loaded from a file first. During this phase grayscale image representation is computed and offered simultaneously.

Image Resample Component
`bitmap` (colors) → `ImageResample (ResamplingType, DesiredSize)` ⇒ `bitmap` (colors) + `bitmap` (intensities)

The input image might be either too small or very large for the sake of further processing. With this component it's easy to resize it suitably using Nearest Point, Bilinear and Biquadratic resampling.

Thresholding Component
`bitmap` (colors) → `ImageThresholding (RGBUpper, RGBLower)` → `bitmap` (binary)

There exist special types of images like maps or drawings where it makes little sense to do full edge detection. Instead, certain parts of interest can be extracted by simple interval thresholding.

Gaussian Smoothing Component
`bitmap`(intensity)→`GaussianIntensityFilter(WindowSize,Sigma)`→`bitmap`(intensity)

In noisy images where one would expect problematic edge detection, smoothing step is required. Gaussian smoothing is a well-established method and the implementation allows to configure both the Sigma parameter of the Gaussian function and the window size.

[3] Naturally, an output port of one component can be connected to input ports of multiple components (providing we obey port compatibility).

3.2 Edge Detection Components

For edge detection, the Canny operator [8] was chosen as a main approach, as it is acceptably stable and configurable. The edge detection is performed in multiple stages, starting on an intensity (grayscale) bitmap, which usually involve

1. Source image smoothing (typically by Gaussian convolution)
2. Gradient approximation using first derivative operator
3. Non-maximum suppression
4. Hysteresis thresholding to identify edge pixels

The Canny operator is formed by a chain of components (the latter described below): `GaussianIntensityFilter` → `GradientOperator` → `NonMaximaSuppression` → `HysteresisThresholding`. For an example of the dataflow see Figure 2.

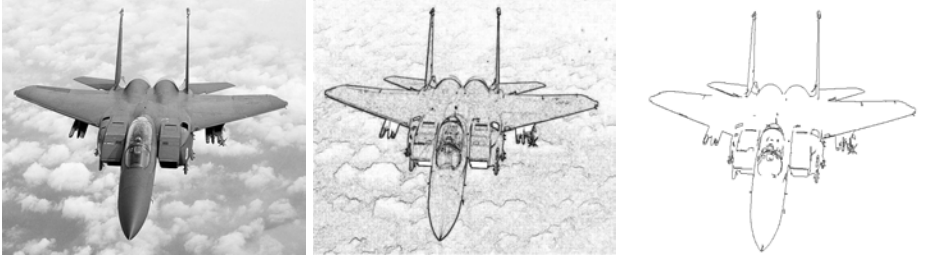


Fig. 2. An example showing input image, image's gradient map and marked edge pixels.

Gradient Operator Component
`bitmap` (intensity) → `GradientOperator (OperatorType)` → `bitmap` (gradients)

Uses simple first derivative operators (Sobel, Prewitt, Roberts-Cross) to obtain local gradient magnitudes and directions (in 45 degree steps) for each pixel.

Non Maximum Suppression Component
`bitmap` (gradients) → `NonMaximaSuppression` → `bitmap` (gradient)

Gradient map obtained by first derivative operator often contains thick regions with high gradient magnitude but to extract edges one would like areas with high gradient magnitude to be as much thin as possible. Non-maximum suppression archives this by ignoring pixels where the gradient magnitude is not maximal in the gradient direction.

Hysteresis Thresholding Component
`bitmap` (gradients) → `HysteresisThresholding (Lower, Upper)` → `bitmap` (binary)

Based on two thresholds gradient map is traced and edge pixels are extracted. Each pixel with gradient magnitude greater than the Upper threshold is marked as edge immediately. Remaining pixels are marked as edges only if they have their gradient magnitude greater than the Lower threshold and if they are connected to some existing edge pixel chain.

3.3 Binary Image Processing Components

The following components process binary bitmap inputs (e.g. obtained from the edge detection). These components could be used to simplification of contours.

Erosion And Dilatation Component
`bitmap`(binary)→`ErosionDilatation (OperationType, MaskType)`→`bitmap`(binary)

Refinement of binary images is often needed (to close gaps, round curved objects, etc.). The operators of erosion, dilatation, opening and closing (combined with a properly chosen mask) are usually a good choice to handle some input image defects.

Thinning Component
`bitmap` (binary) → `Thinning (ThinningType)` → `bitmap` (binary)

Thinning components that implement Stendiford [14] and Zhang-Suen [16] thinning algorithms are handy when it comes to polish results given by edge detection, or when there is a need to turn thick objects into one pixel thin lines. A staircase removal to refine lines contained within the binary image also fits in this category of algorithms. An example of the thinning process is given in Figure 3 (the first two images).

Fig. 3. Input binary image along with its thinned form and refined vector result containing $n = 18$ polylines.

Contouring Component
`bitmap` (binary) → `Contouring` → `bitmap` (binary)

In some cases (when the binary image contains thick objects), an information about contour is more valuable than the object's thinned form. Implemented method is based on approach found in [3] although it uses complete image matrix instead of run length encoding as the binary image representation.

Vectorization Component
`bitmap` (binary) → `Vectorization` → `vectors`

The vectorization component is responsible to turn binary image with marked edge pixels into a set of vectors (polylines). It is based on approach mentioned by [3] or [6].

First, the input binary mask is went through by shifting a 3x3 window over every pixel in order to mark critical points. Those are either endpoints (pixels with only one neighbor within the 3x3 window) or intersections (pixels with more than 2 neighbors). In second phase all polylines between critical points are traced. Another pass through

the image is needed then to identify closed polylines that were left untouched by the previous step. The resulting pixel chains are turned immediately into polylines along with junction and connectivity information (i.e. with the topology).

3.4 Vector Processing Components

Once we get a vectorized form of shape, we move to waters of geometry and graphs algorithms. Although we got rid of the raster information, we now face a tangle of (poly)lines to be meaningfully managed.

Polyline Simplification Component
`vectors → PolylineSimplification(Type, Error) → vectors`

The polylines obtained from vectorization usually carry much more information than required. It also happens that there are undesired irregularities in polylines caused by straightforward vectorization. Hence, a polyline simplification is needed. Multiple approaches are implemented, including those of Douglas-Peucker [11] and Rosin-West [13].

Artifact Removal Component
`vectors→ShortVectorRemoval(ArtifactsType,ArtifactCharacteristics)→vectors`

Aside from polyline simplification, the resulting vector image contains artifacts at higher logical level. For a list list o typical artifacts see Figure 4. Artifact removal component handles these artifacts based on configuration and produces result with reduced noise and longer (more meaningful) vectors, as shown in Figure 4.

Fig. 4. The first picture gives example of the most typical small artifacts – **a.** unconnected polylines **b.** insignificant cycles **c.** 1-connected polylines **d.** spurious loops **e.** insignificant junction connections. The second picture shows an input vectorized image. On the third picture there is output of the artifact removal component (configured to ignore short artifacts) and Douglas-Peucker polyline simplification algorithm.

Iterative Pruning
`vectors → IterativePruning (UpperBound) → vectors`

One of the goals of vector processing is to find a given number of the most significant vectors to represent the original image. With this component an experiment was made to find if a metric based on vector length is a good criterion for selection the most significant vectors.

The algorithm (our original design) works as follows: An upper bound is selected by user that represents the maximum number of vectors to obtain. First, all vectors

Fig. 5. The first picture shows line drawing containing $n = 12,617$ polylines made of $m = 38,433$ line segments. In second picture is the result of vector removal component (with *upperbound* = 20) combined with Douglas-Peucker simplification. The output contains $n = 19$ polylines with $m = 473$ line segments.

are sorted with respect to their lengths. The algorithm works in iterations and tends toward the desired number of vectors. In order to guarantee convergence, a half of the vectors are expected to be thrown away in each iteration (the number of vectors to throw away can be easily configured as well, giving slower of faster convergence).

Which vectors should be thrown away is decided upon their lengths(short vectors are considered first) and upon the knowledge similar to that used in Artifact Removal Component (most probable artifacts are thrown away first). A typical output is depicted in Figure 5.

Polyline Connection Component

`vectors→PolylineConnection(Scenarios,ScenariosCharacteristics)→vectors`

It happens (especially in edge detection) that a significant polyline gets divided into multiple parts as a result of inaccurate thresholding, noise or overlap.

Three phase algorithm is employed to join parts together into bigger entities. The first phase operates on polylines that have their endpoints very close to each other and can be joined together without additional heuristics. Second phase takes care of larger gaps between endpoints with respect to multiple criteria (vector length, distance, orientation, etc.). Third phase tries to handle disconnected corners. Figure 6 features typical scenarios of these three stages. All phases are configurable and optionally offer many important concepts like double sided check, identical angle orientation check etc. Many of these concepts are mentioned in [6].

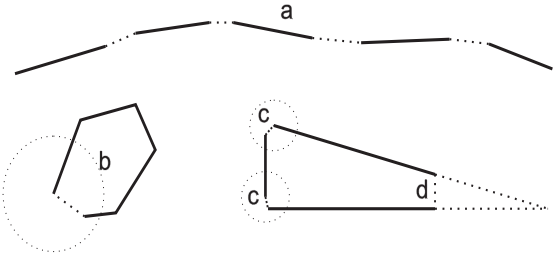


Fig. 6. An example demonstrating typical scenarios when connecting polylines: **a.** connection of lines based on their orientation, angle, ... to create one longer polyline **b.** closing disrupted cycles **c.** connecting near endpoints to form a corner **d.** connecting far endpoints to form a corner, guessing corner shape.

3.5 Vector Output Components

The last group of components provides an output to external storage (now file). Besides an one-to-one export to a well-known format (like WMF, DXF), components in this class cover also secondary feature extraction techniques needed for particular task – typically representations for subsequent similarity measuring/search.

Vectors Output Component
`vectors → VectorToFile(FileName, Format)`

This component saves its vector input into a specified format (DXF, textfile, etc.).

Angles Extraction Component
`vectors → AnglesToFile(FileName, AngleType, NumberOfAngles)`

A specialized feature component transforming polylines to (time) series. Each polyline is normalized first by splitting it to a number of parts of the same length. A set of angles is then saved to the output. The angles can be measured as the smaller angles between each pair of successive line segments, as clockwise (or counter-clockwise) angles between them, or as angles between each line segment and the X-axis.

4 Domain Scenarios

The components of the previously introduced catalogue have been designed in order to easily create scenarios suitable for extraction of simple shapes. Hence, we are primarily interested in simplified representation of shapes inside an image, rather than in a comprehensive image-to-shape transformation preserving as much information as possible.

A simplified shape could serve as a descriptor involved in measuring similarity of two images (based on shapes found inside). The requirement on small-sized descriptor is justified by handling the shape information by similarity measure. Since the similarity measure is supposed to be evaluated many times on entities of a huge image database, the similarity measuring should be as fast (yet precise) as possible. However, this goal can be achieved by a smart shape extraction providing prototypical descriptors. An image represented by a single (or very few) polylines/polygons limited to several tens or hundreds of vertices (say up to 1 kilobyte) – this is our desirable descriptor.

On the other side, we are aware of the difficulties when trying to establish such a simple descriptor. Therefore, we have performed experimentation on various images (photography, drawing, etc.) and tried to assemble several configurations of component networks (called domain scenarios) which gave us the best results for a particular image type (with respect to desirable descriptor properties). In the following sections we present three such scenarios.

4.1 Drawing

As for the drawings, the vectorization task is slightly simpler thanks to the fact that the source image contains the desired information in an easily identifiable form (monochromatic strokes describing shapes, colored areas in case of cartoons, etc.). The extracted layer or layers described by binary images can be further processed by means of thinning (in case of strokes) or contour extraction (in case of thick areas). The result often

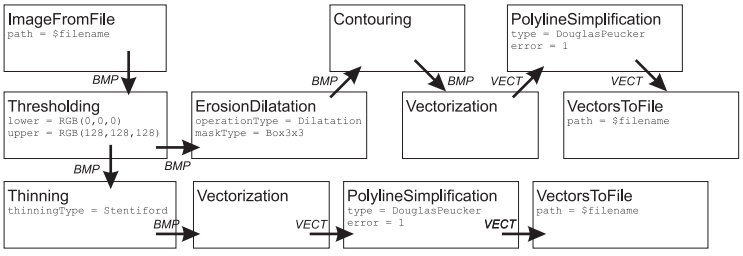


Fig. 7. Scenario 1 – Drawing.

embodies only a low level of noise and can be directly used as is or further simplified. In Figure 7 see the scheme of component configuration and interconnection under `Drawing` scenario. Note that the two branches lead to two different types of shape extraction (skeleton and contour). For an example of data flow regarding to the `Drawing` scenario, see Figure 8.

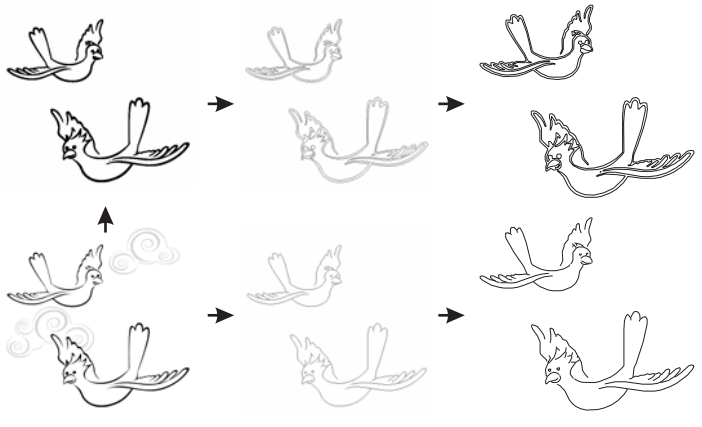


Fig. 8. Scenario 1 – Drawing – Example of data flow.

4.2 Simple Object

For high contrast images containing unsophisticated shapes, the edge detection alone is a reliable way to extract required feature information. When this is known, the artifact removal is a relatively safe operation without the risk of removing important features. A reconnection of disconnected lines (which follows then) almost completely reconstructs the full shape information. Finally, the polyline simplification should be done to straighten jagged lines and minimize the produced number of line segments. In Figure 10 see the scheme of component configuration and interconnection under the `Simple object` scenario. For an example of data flow, see Figure 10.

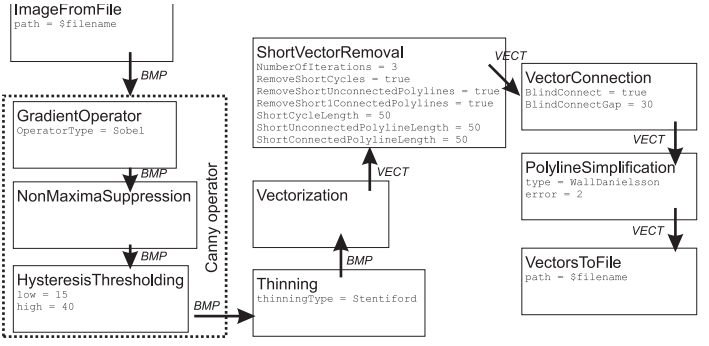


Fig. 9. Scenario 2 – Simple object.

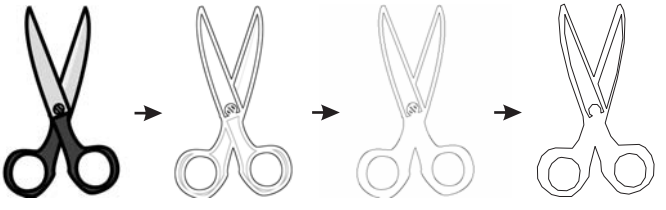


Fig. 10. Scenario 2 – Simple object – Example of data flow.

4.3 Complex Scene

In real-world images (photos), the edge detection cannot guarantee getting clean shapes, on the contrary there are usually huge amounts of false detected or unwanted edges. The iterative pruning is supposed to take care of most of the "trash" in the vector output and even then, maximum effort must be directed into connecting disrupted polylines, corner detection and polygonal approximation. In Figure 11 see the scheme of component configuration and interconnection under the `Complex scene` scenario. For an example of data flow, see Figure 12.

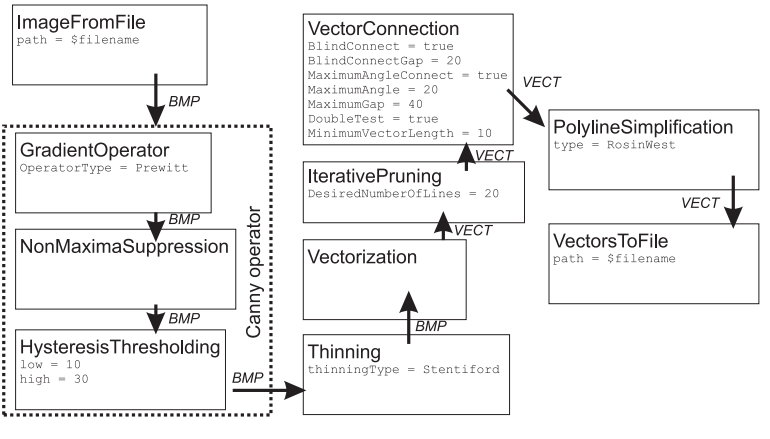


Fig. 11. Scenario 3 – Complex scene.

Fig. 12. Scenario 3 – Complex scene – Example of data flow.

5 Conclusions & Future Work

In this paper we have presented a highly configurable framework for shape extraction from raster images. Based on the framework, we have proposed a catalogue of components, which have been designed to be easily configured into a network. Based on experiments, we have recommended three domain scenarios for extraction of simple shapes, in order to create useful descriptors for similarity search applications.

In the future we would like to investigate similarity measures suitable for shape-based similarity search. An extraction of simple prototypical shapes from images (as proposed in this paper) is crucial for similarity measuring, so it is an unavoidable step when trying to "bridge the semantic gap" in image retrieval. Furthermore, we would like to automate the scenario recommendation process (where each component in scenario evaluates the goodness of what it produces), resulting in a kind of unsupervised extraction technique.

Acknowledgments.

This research has been partially supported by GAČR grant 201/05/P036 provided by the Czech Science Foundation.

References

1. Filters library (available at `http://sourceforge.net/projects/filters/`).
2. Image and vector processing framework (available at `siret.ms.mff.cuni.cz`).
3. S. V. Ablameyko. *Introduction to Interpretation of Graphic Images.* SPIE, 1997.

4. T. Adamek and N. O'Connor. Efficient contour-based shape representation and matching. In *MIR '03: Proceedings of the 5th ACM SIGMM international workshop on Multimedia information retrieval*, pages 138–143, New York, NY, USA, 2003. ACM Press.
5. T. Adamek and N. E. O'Connor. A multiscale representation method for nonrigid shapes with a single closed contour. *IEEE Trans. Circuits Syst. Video Techn.*, 14(5):742–753, 2004.
6. P. Altman. Digitalization of map. Master's thesis, Charles University, Department of Software and Computer Science Education, 2004.
7. I. Bartolini and M. Patella. Warp: Accurate retrieval of shapes using phase of fourier descriptors and time warping distance. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(1):142–147, 2005. Member-Paolo Ciaccia.
8. J. Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, 1986.
9. A. Cardone, S. K. Gupta, and M. Karnik. A survey of shape similarity assessment algorithms for product design and manufacturing applications. *Journal of Computing and Information Science in Engineering*, 3(2):109–118, 2003.
10. V. Castelli and L. D. Bergman, editors. *Image Databases : Search and Retrieval of Digital Imagery*. Wiley-Inter., 2002.
11. D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a line or its caricature. *Canadian Cartographer*, 10(2):112–122, 1973.
12. J. Pelikán and J. Kostlivý. Jagrlib: Library for computer graphics education. In *WSCG (Posters)*, pages 125–128, 2004.
13. P. L. Rosin and G. A. W. West. Segmentation of edges into lines and arcs. *Image Vision Comput.*, 7(2):109–114, 1989.
14. F. Stentiford and R. Mortimer. Some new heuristics for thinning binary hand-printed characters for ocr. *IEEE Transactions on Systems Man and Cybernetics*, 13(1):81–84, 1983.
15. M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. Keogh. Indexing multidimensional time-series with support for multiple distance measures. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 216–225, New York, NY, USA, 2003. ACM Press.
16. T. Y. Zhang and C. Y. Suen. A fast parallel algorithm for thinning digital patterns. *Commun. ACM*, 27(3):236–239, 1984.
17. D. Ziou and S. Tabbone. Edge detection techniques - an overview. *International Journal of Pattern Recognition and Image Analysis*, 8:537–559, 1998.

Inductive Models of User Preferences for Semantic Web

Alan Eckhardt

Charles University, Faculty of Mathematics and Physics
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
alan.eckhardt@mff.cuni.cz

Abstract. User preferences became recently a hot topic. The massive use of internet shops and social webs require the presence of a user modelling, which helps users to orient them selfs on a page. There are many different approaches to model user preferences. In this paper, we will overview the current state-of-the-art in the area of acquisition of user preferences and their induction. Main focus will be on the models of user preferences and on the induction of these models, but also the process of extracting preferences from the user behaviour will be studied. We will also present our contribution to the probabilistic user models.

1 Introduction

The user preference modelling plays an important role in the current web. Internet shops need to help the user to find the product he/she searches for, social webs may suggest a contact that the user wants. The process of acquisition of user's preferences starts with the acquisition of known preferences (e.g. from the user behaviour) and then using these known preferences to get the user's preferences of other objects. In this paper, an example of a user who is buying a digital camera will be used. In Section 2, several user models will be presented and in Section 3, some of current methods of induction of user preferences will be described, including our own probabilistic model.

1.1 A use case for the induction of user preferences

We will present a typical use case for the induction of user preferences. We will describe a complex system for the extraction of information from the web and for the presentation of collected information to the user. This system will be aimed to help the user to find a camera that fits best his needs. Whole system proposition is in Figure 1.

The first task of this system is to collect data from various sources from the internet. Information is stored in various forms, most often in HTML format, and it has to be transformed to a computer-readable form (Semantic data). The typical computer-readable form is RDF [2] - a language of triples of the form (subject, predicate, object). Extension of RDF is OWL [1] which is one of standard ontology languages. Ontologies can be also used to annotate raw text

© J. Pokorný, V. Snášel, K. Richta (Eds.): Dateso 2007, pp. 103–114, ISBN 80-7378-002-X.
VŠB – Technical University of Ostrava, Dept. of Computer Science, 2007.

data from the web. This transformation is called the 'semantic annotation' and there are many methods performing semantic annotation, though their accuracy and universality may be questioned. However, studying the semantic annotation is not the aim of this work, it is one of components in whole process.

With semantically annotated data or with online RDF sources, we may present the integrated information to the user. Several works (e.g. [8]) concern the graphical user interface that represents the semantic data. The task is to present the most important objects to the user in a such way that he/she will notice them before noticing the other objects.

The inductive methods enable to determine which objects are important and which are not. The process of determining the importance of an object is iteratively executed, it may be triggered e.g. by some user behaviour, for example by clicking on some object that is not considered important. Interpreting user behaviour is another part of whole process.

The interpreted behaviour is then used to generate a user preference model. This can be done by an already known inductive method. This user preference model will be then used to alter the appearance of the web page, for presenting preferred objects etc. This information can be also used by other servers, in case of a distributed computation.

In our work, we will focus on the induction process - the induction of preference model from some rated objects.

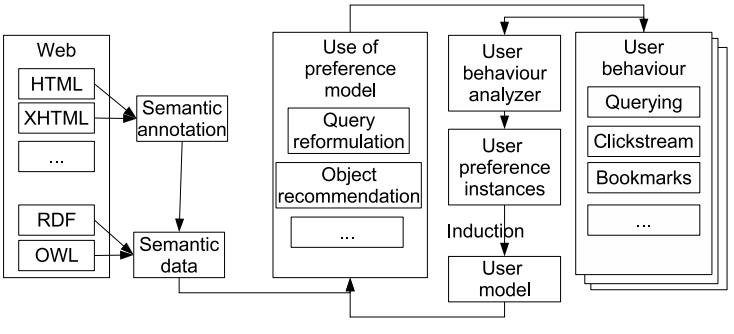


Fig. 1. Complex system for user preferences on the web

2 Models of user preferences

We will use the following notation - $u_1, ..., u_k \in U$ for the users, $o_1, ..., o_n \in O$ for the objects, $a_1^1, ..., a_1^m$ for the attribute values of the object o_1 and $A^1, ..., A^m \in A$ for the attributes of objects. When speaking only of one object, we will also use only o for object and $a^1, ..., a^m$ for its attribute values. Often, all objects are

from the same domain and they will have the same set of attributes. For that reason we will be interested mainly in the attribute values. We will denote the user's preference of an object as $P(o)$, meaning how much object o is preferred. We will use the notation $P_1(o)$, which means the preference of the user u_1 on the object o, when we want to explicitly denote the user. The range of P depends on user model used, some of them do not have direct preference, e.g. the preference relations studied in Section 2.2.

We now briefly distinguish several types of attribute domains, as was done in [5]. The first domain type is nominal. These domains have no ordering and are mainly text based. Typical example may be the color or the manufacturer. Second type are ordinal domains, on which exists some ordering, but not unit. For example set Big, Medium, Small may form an ordinal domain. When a unit of measure is added, we get interval domains - for example {1, 0.2, 0, -0.4, -1}. Finally, ratio scales have also an element 0 explicitly defined. Example of ratio scale may be the price, the number of megapixels, the weight etc. We will refer to both ratio and interval domains as numerical domains. In real data, most frequent are nominal and interval/ratio domains. Ordinal domains are created with a user influence - a user will say that every object that weights more than 600g is heavy, above 200g is medium and less than 200g is light.

2.1 Boolean preference

Boolean user preference model is used in some methods, where the user preference model is not explicitly mentioned. This model distinguish only two states of an object - either preferred or non-preferred. This is very simple approach with little semantic, but may be used when lot of computing power is required. In these cases, preference is represented by a vector of n bits, we will note this vector $v(u) : U \rightarrow [0, 1]^n$. Operations on these vectors are fast when only binary operations like and, or, xor etc. are needed. These operations may be sufficient for some inductive methods but not for others. For example, this model can be successfully used in the collaborative filtering, which is described in Section 3.2.

Comparison of two user preferences.

Computing the similarity of two users may be computed in following way

$$s(u_1, u_2) = 1 - \sum_{i=1}^{n} (v(u_1) \text{ xor } v(u_2))[i]/n.$$

The sum in equation expresses the number of ratings, where both user disagree, e.g. object is preferred for u_1 and non-preferred for u_2. If we based the similarity on common preferences of u_1 and u_2, it will be influenced by the number of rated objects, which is not desirable. If this fact is of no relevance, alternative way for computing similarity may be

$$s(u_1, u_2) = \sum_{i=1}^{n} (v(u_1) \text{ and } v(u_2))[i]/n.$$

Computing the similarity of two users is essential for methods like collaborative filtering. Surprisingly, there are not many articles concerning this problematic. Further investigation and research in this area may reveal interesting ideas.

2.2 Preference relations

Preference relations are the oldest model of user preferences models, its description may be found e.g. in [15]. Basic idea behind preference relations is to characterize the relation between objects o_1 and o_2. We can say that o_1 is more preferred than o_2, o_1 is equal to o_2, o_1 is incomparable to o_2 or that o_1 is a little better than o_2 but not much. For the strict preference, we traditionally denote this relation as P. Then $P(o_1, o_2)$ states that o_1 is more preferred than o_2. For equivalence of two objects, we use I, e.g. $I(o_1, o_2)$ means that o_1 is as preferred as o_2. Finally, relation R is created as union of P and I, $R(o_1, o_2)$ meaning o_1 is equal or preferred to o_2. For incomparability, relation J is introduced. Then $J(o_1, o_2)$ means that o_1 and o_2 are incomparable.

We left out the case when o_1 is a little better than o_2. We may create new relation Q, so that $Q(o_1, o_2)$ states that o_1 is a little better than o_2. By a simple extension, set of relation $Q_1, ..., Q_n$ will represent the fact that o_1 is a little better than o_2, with Q_1 representing the lowest difference of preference of o_1 and o_2 and Q_n the highest difference.

Properties of relations determine properties of preferences. There are several properties, such as the existence of a minimum or the completeness (linearity) of the relation. For deeper insight in these properties, we again recommend [15], which is specialized survey of preference relations.

All these structures may be extended to valued structures. One special case is many valued logic, studied more in depth in 2.3. An example of a valued structure is $\mu(P(a, b)) : O^2 \to [0, 1]$. The interval $[0, 1]$ may be replaced by any other linear numerical structure, and it represents the degree (truth value) of the relation. Valued relations may successfully replace relations $Q_1, ..., Q_n$, which are a middle step between standard relations and valued relations.

Comparison of two user preferences.

When we want to compare preferences of two users, we have to compare two preference relations. When relation is ordered linearly, we compare two ordered lists of objects. In that case, we may count the number of permutation between both lists, which is traditional measure of computing the similarity (or the distance, in this case). However, this may be not the best distance used, because it makes no difference of distance between permuted objects. Switching two neighbour objects makes less change than switching first and last object.

In [14] the distance of two interval fuzzy preference relations is described. However, it can't be simply used as a generalization of simple preference relation, because it will degrade into simple 'equal' or 'not equal' semantics.

2.3 Many valued logic

Many valued logic is an extension of the traditional two valued logic. In the two valued logic, a variable may be either true or false, in the many valued logic a whole set of possible truth values is introduced, often denoted as T. T should form a lattice, typical case is a linear structure and the most used is interval $[0, 1]$. The set T will represent the set of preference values where 1 is most preferred and 0 is least preferred. Other structures than $[0, 1]$ are possible to use, for example discrete set {Worst, Worse, Neutral, Better, Best} or {One star, Two stars, Three stars, Four stars, Five stars} may be relevant in some cases. We will interpret the truth value as a degree of preference.

When creating this extension of the two valued logic, we must define a new semantics for logical operators, predicates and quantifiers. This definition can be found for example in [13], but this is not of major interest in this work.

We will use two structures from the many valued logic - the first are fuzzy functions of an attribute domain and the second is an aggregation function. A fuzzy function represents user's ordering and normalization of a domain. For example, consider the attribute price. Most people prefer low prices over high prices. In figure 2,a) we may see an example of the fuzzy function for price.

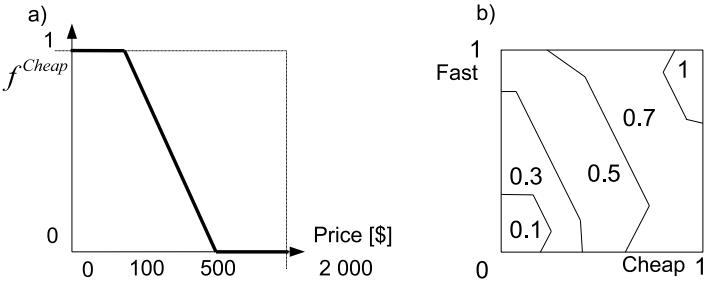


Fig. 2. Fuzzy function of price (a) and an example of a more complicated aggregation function (b)

An aggregation function is used to aggregate several truth values, or preferences, into one truth value, therefore it is a function $@ : T^m \to T$. There are few restrictions on an aggregation function - it must be monotone in all variables, and $@(0, .., 0) = 0$ and $@(1, .., 1) = 1$. An aggregation function is very suitable for the modeling of the user preferences of the complex objects. The user aggregates attributes of an object into the preference of the object itself. An example of an aggregation function may be

$$@_{U_1}(\mathrm{MPix\_}U_1(x), \mathrm{Fast\_}U_1(x), \mathrm{Cheap\_}U_1(x)) = \frac{5 * \mathrm{MPix\_}U_1(x) + 1 * \mathrm{Fast\_}U_1(x) + 3 * \mathrm{Cheap\_}U_1(x)}{9}$$

Symbols MPix, Fast and Cheap denotes fuzzy sets of particular attributes. E.g. Cheap_$U_1(D50)$ represents how camera D50 is cheap for U_1.

We consider the weighted average as a good example of a user fuzzy function. It has clear semantics, because we can see directly, which of the attributes are important for the user and which are not. Even more, from the weight we can deduce how much important an attribute is. However, many more aggregation functions that fits better to psychological aspects of human decision process may be represented. An example of a more complicated aggregation function is in Figure 2,b). These two mechanisms allow us to create very flexible model of user preferences and moreover, the aggregation function models also user decision process.

3 Inductive methods

In this section, we will examine several inductive methods that are used to create a user preference model from some input. The user preference model is often independent of the inductive method, the input may be represented in several ways but often the paradigm of object with some attributes is expected.

Most of the methods expect a training set of objects, which are supposed to belong to 'classes'. These classes of objects in the training set may have different forms, depending on the model of user preferences we are using. For example, when using many valued logic, one class may be $o : P(o) \in [0, 0.1]$. Some of the models of user preferences we have described above do not have a direct interpretation as classes. With preference relations, we have only comparison between two objects. We assume that a method will transform user preferences into several (possibly discrete) classes. For preference relations we may order objects and associate a weight corresponding to the position in the ordered set.

3.1 Inductive logic programming

Inductive logic programming is a method to obtain a logic program. This program may be very general, in our case, it will represent rules that user uses in decision process. After application of these rules to an object, the preference of that object should be obtained. We will describe only predicate logic programs, which are more expressive than sentential logic programs. An introduction to induction of logic programs may be found in [11].

Rules have a head and a body. The head of a rule is a single predicate and the body is a conjunction of predicates. When using fuzzy predicate logic, each predicate has also value that represents the truth value of the predicate. For simplicity (and because of space limitation), we will describe two-valued logic program.

For example, following rules may represent user preferences of cameras

```
GoodMPix(camera) <- MPix(camera)>5;
GoodWeight(camera) <- Weight(camera)<700 & Weight(camera)>300;
GoodCamera(camera) <- GoodMPix(camera) & GoodWeight(camera);
```

Inductive process works with this input

1. The background theory B.
2. Positive examples E^+.
3. Negative examples E^-.

Background theory is used to infer new statements H (hypothesis) about examples. Both E^+ and E^- are formulas, but E^- have empty head, e.g.

```
<- GoodCamera(D50);
<- GoodCamera(D40);
```

On the other side, positive examples have empty body, e.g.

```
GoodCamera(D200) <- ;
GoodCamera(D2x) <- ;
```

We present also an example of a background knowledge B:

```
Weight(D2x)=12 <- ;
Megapixels(D2x)=1150 <- ;
```

Four conditions must be fulfilled

1. Prior satisfiability $B \ \& \ E^- \not\models \square$
2. Posterior satisfiability $B \ \& \ E^- \ \& \ H \not\models \square$.
3. Prior necessity $B \not\models E^+$.
4. Posterior sufficiency $B \ \& \ H \models E^+$.

The symbol $\square$ represent the contradiction or false. The meaning of these condition is clear - with B and E^- we should not get a contradiction, e.g. E^- should comply to the background knowledge. With the B, E^- and H we should not get a contradiction either. On the other hand, we want that examples are not deducible from B itself, only with addition of H.

Now we will describe a general algorithm of hypothesis construction, as proposed in [11].

```
QH = Initialize();
do
   Delete H from QH;
   Choose rules r_1,...,r_k ∈ R to be applied to H;
   Apply r_1,...,r_k to H, obtaining H_1,...,H_n;
   Add H_1,...,H_n to QH;
   Prune QH;
while not Stop-criterion(QH)
```

QH is a set of candidates to hypothesis and R is a set of rules, which transform H. An example of a rule may be dropping a clause or adding a clause to the body of H. During each step, a hypothesis H and rules that will be applied to H are chosen. Result of the application of rules on H are then stored in

QH and candidate set is pruned. Pruning means that useless candidates are deleted. Implementation of each of methods Delete, Choose, Prune and Stop-criterion may be different. Also the set of available rules R may differ across implementations.

An example of application of a rule on a hypothesis `GoodCamera(D2x) <- ;` may be

```
GoodCamera(camera) <- Megapixels(camera)=12;
```

or

```
GoodCamera(camera) <- Weight(camera)=1150;
```

This is an example of a generalization rule, whose result must be verified on E^+ and E^-.

The hypothesis should be completely correct, i.e. it should have the Posterior sufficiency property. However, if we relax this property, a kind of probabilistic rules will be generated. The probabilistic approach is further studied in 3.4.

3.2 Collaborative filtering

Collaborative filtering represent widely used method for acquisition of user preferences. It is based on assumption, that the preference of user u_0 on object o will be the same as the preference of users $u_1, ..., u_k$ that are 'similar' to u_0. The similarity of users is based on similarity of ratings on objects. Many collaborative filtering methods are described in [10].

This method requires a lot of ratings of objects by a lot of users. For computing the similarity of users, we need a lot of object ratings, for accuracy of computing the rating of object o, we need a lot of users similar to u_0.

There are several different algorithms for collaborative filtering. The first and most simple is K-NN, the K nearest neighbor. This is most intuitive algorithm - for a user u_0, we find the K nearest users $u_1, ..., u_K$. The distance is computed by the similarity of users' ratings, for example

$$s(u_i, u_j) = \sqrt{\sum_{l=1}^{n} (P_i(o_l) - P_j(o_l))^2} \tag{1}$$

Having these K nearest neighbours, we may compute the rating of objects as average of users' ratings

$$P_0(o_i) = \sum_{j=1}^{K} (P_j(o_i))/K \tag{2}$$

Another method of computing new ratings is to use a naive Bayes classifier [6]. For each object o, we construct a separate Bayes classifier. Input of the network are the ratings of all objects other than o. Bayes network will answer the question "What is the value $P(o)$, when the user rated other objects this way?". Bayes network learns its parameters from the ratings of users that have rated o.

Other methods are considered as a content filtering methods - they work with the objects and their properties rather then with the preferences of other users. However, both approaches are often combined. Collaborative filtering can not be appropriately used for new objects, which have not yet been rated by any user. For this reason, some kind of the content filtering is also used and the results of both methods are combined together.

However, some of the presented methods may be used both on other users' ratings and the properties of objects, or both together.

3.3 Decision trees and rules

Decision trees are well known structure from the data mining theory, they are used to model user decision process (or any decision process). Decision trees are oriented trees with class names in leaves and a rule associated to each inner node. An example of a decision tree which models user preferences of digital cameras is in figure 3. We can see that cameras with at least 10 megapixels are good, cameras that costs more than 500$ are bad.

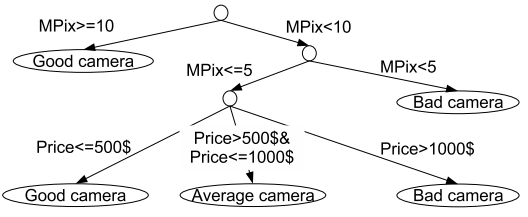


Fig. 3. An example of a decision tree

Theory of the induction of the decision trees may be found in [12]. Induction of a decision tree is discussed mainly for discrete attributes with few values. Decision trees were extended to fuzzy decision trees, one of a new contribution is in [5], older one is [3].

Basic induction procedure starts with a 'training set', which is a set of objects with known classes. From these objects, a tree is constructed and then it is verified on a test set of objects, which is also a set of objects with known attributes. The construction is typically a top-down algorithm. During each step, all possible splits are considered and the most appropriate one is chosen. The appropriateness is measured with an 'impurity measure'. Impurity measure measures how evenly the data are spread in classes. When all objects are only in one class, impurity measure is 0, when all classes contain same number of objects, impurity measure should be 1. Most used is entropy-based impurity measure ([12]) and Gini index ([4]). The structure of a tree depends mainly on the impurity measure used during its construction.

3.4 Probabilistic methods

Probabilistic methods use probability as a method of inducing user preference. There are several possible approaches to statistical interpretation of preference data, e.g. [9]. Usage of probability is reasonable when working with user preferences, because few users have consistent preferences. Often, an exception from a general rule occurs. This exception have to be handled explicitly in non-probabilistic methods, but it creates no problem in the probabilistic methods. For example in inductive logic programming, we can assign a probability to each rule, denoting how much the rule is true in general, or in decision trees the probability would be associated to each left node.

Probabilistic model for boolean preference model. The probabilistic preference model proposed in [9] is based on two measures - the first is the actual user preference and the second is the accessibility (or the frequency) of the object. The second one is important because when trying to induce user preferences from user behaviour it is apparent that user will rather examine frequent items than rare items just because they are more frequent.

The preference of an object is $Pref(x) = f(x|V)$, where V is a user profile or a user history and f returns the preference value of object x. Objects have again several attributes, in [9] called 'features'. The preference of an object is computed as

$$Pref(x) = 1/|X| \sum_{a \in X} Pref(a), \tag{3}$$

e.g. it uses the average of attributes values preferences. The problem of finding $Pref(a)$ is then analyzed, actually in the similar way as in our previous work [7]. We will compare these approaches in Section 3.4. The suggestion in [9] is to use formula

$$Pref(a) = I(X(a); V) = log \frac{P(X(a)|V)}{P(X(a))} \tag{4}$$

where $X(a)$ is the set of objects containing a. In other words, attribute value a is preferred, if the probability that the user selects an object with a is higher than the probability of the occurrence of an object with a in the whole set of objects.

Probabilistic model for many valued logic. A probabilistic model for the many valued logic is our contribution. It is aimed on nominal attributes and uses only weighted average as aggregation function. We concentrate on the case, when we know preferences of some objects and user aggregation function but not the preference of attribute values.

We are missing the preference of an attribute value a which is the value of attribute A_k. But we know the preferences of objects and the aggregation

function. We will consider the set $X(a)$ of objects which have the attribute value a. We will look into the distribution of the preference of these objects. When most of the objects in $X(a)$ have high preference, attribute value a will also have high preference. Formally,

$$P(a) = \frac{\sum_{o \in X(a)} P(o)}{|\{o \in X(a)\}|}. \tag{5}$$

The ratio between the weight of the attribute A_k in aggregation function @ and the sum of the weights of all attributes represents the probability that the preference is computed correctly. It is denoted formally in the following equation

$$P(A_k) = \frac{W(@, A_k)}{\sum_{i=1,..,m} W(@, A_i)}, \tag{6}$$

where $W(@, A_j)$ is the weight of attribute A_j in @. The preference of an object is influenced more by an attribute with a big weight than by an attribute with a small weight. Therefore this method is useful mainly for the attributes with big weight.

Computing preference of one attribute value a may be costly when the number of objects with a is big. However, higher number of objects with a means also higher precision of this method. Naturally, this method is only useful for the domains with discrete values, especially non ordered domains like color or manufacturer. This method can't be successfully used for continuous domains, because there will be very few objects with the same attribute value. However, we may divide these continuous domains to a set of discrete intervals, and use the method proposed above on these intervals.

Comparison with our model. Our model is an extension over the model proposed in [9]. There are two aspects in which our approaches differ

1. In [9] the boolean user model is used (implicitly). We use many-valued logic model, which is more general.
2. The preference of an object is computed in [9] as a simple average of preference of its attributes. In our model, weighted average is used.

However, there is a similarity in our approaches - we use the preference of objects for acquiring the preference of attribute values. This is an inverse process to deduction, where the preference of an object is computed from the preference of its attribute values.

4 Conclusion

In this paper, we have reviewed some of the main user preference models. There are other models as well, their complete listing is beyond the scope of this paper, we recommend [15] to the interested reader. The user model is used in a web

system to better present data to the user or to alter his/her query in order to the results of the query actually better fit his/her preferences.

The creation of the user model is often done by inductive methods, which were studied in this paper in Section 3. We presented methods that are used for induction of user preferences and one probabilistic model for boolean user preferences. We have developed a similar approach for many valued logic, which is more general and flexible than the method studied in section 3.4. The precision and the computational cost of our approach is still to be tested on real data. These experiments are however beyond the scope of this paper, which is an overview of methods used for the induction of the user preferences.

Acknowledgment

Supported by Czech projects MSM 0021620838, 1ET 100300517 and 1ET 100300419.

References

1. Owl, ontology web language . `http://www.w3.org/TR/owl-features/`.
2. Rdf data format. `http://www.w3.org/TR/rdf-primer/`.
3. B. Apolloni, G. Zamponi, and A. M. Zanaboni. Learning fuzzy decision trees. *Neural Networks*, 11(5):885–895, 1998.
4. L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Chapman & Hall, New York, 1993.
5. K. Cao-Van. *Supervised Ranking, from semantics to algorithms*. Ph.D. dissertation, Ghent University, 2003.
6. P. Domingos and M. J. Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130, 1997.
7. A. Eckhardt. Methods for finding best answer with different user preferences, In Czech only, Master's thesis, Charles University in Prague. 2006.
8. Lars Hult, Magnus Irestig, Jonas Lundberg *Design Perspectives. Human-Computer Interaction*. Vol. 21, No. 1, Pages 5-48, 2006.
9. S. Y. Jung, J.-H. Hong, and T.-S. Kim. A statistical model for user preference. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6):834– 843, June 2005.
10. B. Marlin. Collaborative filtering: A machine learning perspective. Master's thesis, University of Toronto, 2004.
11. S. Muggleton and L. D. Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:629–679, 1994.
12. J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, 1986.
13. P. Vojtáš. Fuzzy logic programming. Fuzzy Sets and Systems. 124,3 (2001) 361-370
14. Z. Xu. On compatibility of interval fuzzy preference relations. *Fuzzy Optimization and Decision Making*, 3(3):217–225, 2004.
15. M. Öztürké, A. Tsoukias, and P. Vincke. *Preference modelling. Multiple Criteria Decision Analysis: State of the Art Surveys*. Springer New York, 2006.

Improvement of Text Compression Parameters Using Cluster Analysis

Jiří Dvorský, Jan Martinovič

Dept. of Computer Science, VŠB – Technical University Ostrava,
17. listopadu 15, 708 33 Ostrava-Poruba, Czech Republic
{jiri.dvorsky,jan.martinovic}@vsb.cz

Abstract. Several actions are usually performed when document is appended to textual database in information retrieval system. The most frequent actions are compression of the document and cluster analysis of the textual database to improve quality of answers to users' queries. The information retrieved from the clustering can be very helpful in compression. Word-based compression using information about cluster hierarchy is presented in this paper. Some experimental results are provided at the end of the paper.

1 Introduction

The modern information society produces immense quantities of textual information. Storing text effectively and searching necessary information in stored texts are the tasks of *Information Retrieval Systems* (IRS). Information retrieval systems [1] constitute a class of program tools for processing, storing and selecting data that are texts. An IRS is accessed by a user who needs to obtain certain information (document) from this system to solve a problem. Such information is called *relevant*. Various documents are suitable to users to various extents. Therefore, we also speak of a *document relevancy ratio*. When searching information in an IRS, a system user submits his or her requirement, a *query*, and awaits a result in the form of a set of documents selected by the system as documents matching the user requirement, i.e. matching the user's query.

It is clear that the size of an IRS increases with the increasing size of available external memories. The information explosion can be avoided basically in two ways:

1. Extensively - by purchasing higher capacity memories, or
2. Intensively - by storing data in memories in a better way.

The first solution is not interesting in terms of research. The key to the second solution is *data compression*. The database of a typical IRS is a *textual database*, which stores all information that is necessary for the function of the IRS. Textual databases typically consist of the three following parts:

– full texts from documents that form a document collection

© J. Pokorný, V. Snášel, K. Richta (Eds.): Dateso 2007, pp. 115–126, ISBN 80-7378-002-X.
VŠB – Technical University of Ostrava, Dept. of Computer Science, 2007.

- data structures for searching documents
- list of document identifiers and of their attributes and other auxiliary structures

Haskin claims in [15] that the size of textual database auxiliary structures makes up 50% to 300% of the size of original documents, this implies that a textual database is a suitable material for compression. It is only necessary to use the several lossless compression methods to save space.

However, the problem of compression in IRS is not as simple as it seems at first sight. On the one hand, compression saves space for data, while, on the other hand, it may entail a certain operation overhead (i.e. adding certain amount of time to the cost of accessing the data). Also, the space saving must be significant to be useful. Therefore, the objective is not to compress the textual database as a whole. This usually does not lead to good results since individual parts of an IRS contain redundancies of different types; different data structure types are based on a different model, according to which it is possible to determine the best compression method.

Experience shows that it is useful to consider, analyze and design the best compression method when storing extensive textual databases. It also proves to be desirable to study highly specialized compression methods that are convenient only for a certain data type in an IRS. Tens of megabytes can be saved either saving one byte in data structures or by improving compression ratio of text compression in an IRS.

This paper will focus on text compression methods suitable for IRS. Factors significantly influencing compression methods that are suitable for IRS include: compression ratio, decompression speed, the possibility of decompressing the document, and connection with searching [26].

The aim of this paper is to extend of existing word-based compression methods with hierarchical agglomerative clustering to improve compression performance, especially compression ratio.

The paper is organized as follows. Sections 2 and 3 provide an outline about word-based compression methods. Section 4 briefly describes clustering methods. In section 5 characterization of our approach is provided. Experimental results are discussed in section 6. Short conclusion is provided in section 7.

2 Word-based compression

The compression algorithm transforms input data that contains a certain redundancy to output data, in which redundancy is reduced to a minimum. The input and the output of data compression algorithms are generally strings of characters over a certain alphabet. There are no requirements concerning the alphabet. The selection of the alphabet is therefore a question of choice, which is influenced by various perspectives.

The search for a suitable alphabet will proceed from the syntactical structure of natural languages: character → syllable → word → sentence. Pertaining to this structure a certain correspondence can be discovered between the language

structure and possible alphabets. Each level represents one potential alphabet. The first level is represented by a character-based alphabet. The next possible level is an alphabet of syllables. Here we face the problem of identifying syllables in the text [18]. Much greater possibilities are offered by the third level – *word-based alphabet*.

A compression method based on an alphabet of words, which will be called the *word-based compression method*, regards text as a sequence of words[1] in a certain language. The application of irregular distribution of individual word occurrence probabilities is then assumed during compression in statistical compression methods, or the clustering of words into language syntactical structures is assumed in dictionary methods. It is presupposed that the language structure controls not only characters but also words. Here are some examples:

- fixed phrases, e.g. "How do you do?"
- constructions based on grammar, e.g. constructions with an article - "the best", phrasal verb - "to be interested in"
- constructions based on the contents of the text, e.g. "data compression", "word based" are frequently repeated in this paper

It is also presupposed that these constructions are repeated and that it is possible to achieve a certain compression on the basis of this repetition. It is not presupposed that the text consist only of hapax legomena[2] – even though this assumption can be used as well.

3 Word-based compression methods

The first widely accessible description is that of Bentley et al. [2] (see also Ryabko [25]), who proposed that a dictionary of words parsed from the text should be coupled with codewords that correspond to MTF numbers. Moffat [20] also experimented with word-based models, and showed that for a range of data files the MTF transformation was less effective than a straightforward entropy code in those experiments, arithmetic coding. A similar word-based model is available as part of the arithmetic coding implementation of Moffat et al. [21].

Moffat [20] also investigated first-order and second-order word-based models, in which one or two words are used to condition the probability distribution used by the entropy coding stage (respectively). Other authors have made use of word-based models since then including Horspool and Cormack [16], Zobel and Moffat [27], and Moffat et al. [22]. de Moura et al. [5] have extended the idea of word-based compression to what is called the *spaceless words* approach, which can be considered as special case of eliminating of victim [7].

[1] Sequences of spaces, punctuation between two words is called *nonword*. HTML or XML tags are considered as the third part of word-based alphabet in the case of compression of HTML/XML document. Words, nonwords and tags are called *tokens* in general.

[2] Hapax legomenon – a word with only one occurrence in the examined text.

Huffword compression method was designed by Moffat and Zobel in 1994 [26]. HuffWord is a compression method that is specialized in texts and uses a word-based alphabet. The compression is based on the so-called Huffman canonic coding. The authors of the HuffWord claim a compression ratio of about 30%.

3.1 WLZW and WBW methods

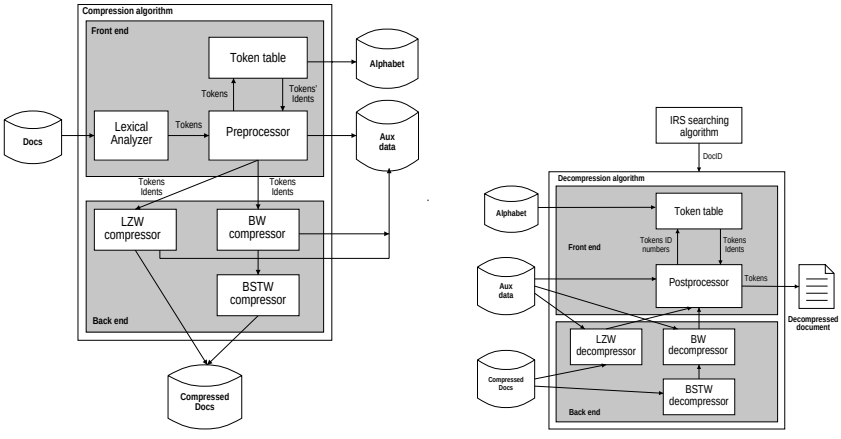
The beginning of the WLZW method dates back to 1998 when its first variant and the first results were published [6, 9]. Other modifications and results can be found in [10, 13, 11]. The WBW method is newer and its beginning dates back to 2001 [12].

Scheme of WLZW and WBW compression algorithms Figure 1(a) shows a schematic structure of compression algorithms WLZW and WBW. The figure clearly shows that both compression methods can be roughly divided into two parts that are named *front end* and *back end*. Both compression methods process document texts in two passes. The division of compression methods into two parts corresponds with those passes. Some parts are not active at all in the individual passes or their activity is different. Two algorithm phases can be distinguished according to the order of passing through document texts in both compression algorithms:

- *First phase* - corresponds to the *first pass* of the compression algorithm. A word-based alphabet is created in this phase. Individual tokens are extracted from documents through the process of lexical analysis, which is implemented by the front end part. This phase is shared with document indexing in the textual database.
- *Second phase* - corresponds to the *second pass* of the compression algorithm. A complete word-based alphabet is available upon the completion of the first phase and the actual document compression can begin. A lexical analysis is again performed and the token sequence that is being created is compressed by a chosen algorithm. Both phases of the compression algorithm, the front end and the back end, are already active in this phase.

The division of the compression algorithm into two relatively independent parts made it possible to separate two different compression algorithm phases, i.e. the creation of a word-based alphabet and the actual compression. Naturally, this separation has simplified the algorithm design, it has made the implementation more transparent, etc.

Scheme of WLZW and WBW decompression algorithms Figure 1(b) shows a structure of the decompression algorithm of WLZW and WBW methods. As the figure clearly shows, both decompression algorithms can be divided into two parts - *front end* and *back end*, like the compression algorithms. WLZW and WBW methods were constructed as asymmetric, from whence it follows that:



(a) Compressor (b) Decompressor

Fig. 1. WLZW and WBW schemes

- Decompression is simpler than compression. All operations that can be performed by the compression algorithm are transferred to this algorithm, so that the decompression algorithm performs only the necessary operations.
- Decompression involves only one phase. The decompression of a document requires only one pass through the compressed text. All objects contained in scheme 1(b) are therefore active during decompression and the decompression progress has a "through-flow" character.

The division of the decompression algorithm into two parts enabled the separation of the actual decompression and the subsequent reconstruction of the document text from a sequence of token identifiers.

Furthermore, this division made it possible to see the compression and the decompression algorithms as two dual algorithms with a similar internal structure.

4 Cluster analysis

Finding of groups of objects with the same or similar features within given set of objects is the goal of cluster analysis [3]. These groups are called *clusters*. In our case objects are equal to documents that will be stored in textual base, and clusters are equal to groups of similar documents. First of all the distance of two documents and distance matrix C for each pair of documents should be defined. Our approach of cluster analysis is based on ultrametric tree [17].

4.1 Ultrametric

The triangular inequality holds for a metric space: $d(x, z) \leq d(x, y) + d(x, z)$ for any triplet of points x, y, z. In addition the properties of symmetry and positive

definiteness are respected. The ultrametric inequality is: $d(x,z) \leq max\{(d(x,y), d(x,z)\}$ for any triplet x, y, z [23].

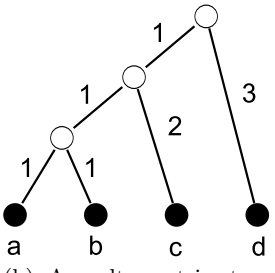
Definition 1. *A metric space (X,d) is called ultrametric if for all $x,y,z \in X$ we have $d(x,z) \leq max\{d(x,y), d(y,z)\}$ [4].*

Definition 2. *The ball on the ultrametric is $B_X(x,r) = \{z \in X | d(x,z) \leq r\}$ for a point $x \in X$ and $r \geq 0$.*

An *ultrametric tree* is a rooted tree whose edges are weighted by a non-negative number such that the lengths of all the root-to-leaf paths, measured by summing the weights of the edges, are equal. A distance matrix C is ultrametric if an ultrametric tree can be constructed from this matrix. Figure 2 shows an example of an ultrametric matrix and an ultrametric tree constructed from this matrix.

	a	b	c	d
a	0	2	4	6
b	2	0	4	6
c	4	4	0	6
d	6	6	6	0

(a) An ultrametric matrix C



(b) An ultrametric tree for C

Fig. 2. Ultrametric tree sample

As is well known, in clustering a bijection is defined between a rooted, binary, ranked, indexed tree, called a dendrogram, and a set of ultrametric distances [23, 17].

4.2 Clustering

Definition of hierarchical aglomerative clustering method outgoing from [17] which compute ultrametric tree from distance matrix C can described as:

1. Create distance matrix C.
2. 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. The most similar clusters p and q are found in distance matrix and their distance is determined as $prox_s[p,q]$.

4. Clusters p and q are joined together and the number of cluster is reduced. New formed cluster is determined as t (it replaces row and column q), and distance $prox_s[t,r]$ of new cluster t to all other clusters r are computed. For Average method $prox_s[t,r]$ is defined as:

$$prox_s[t,r] = \frac{N_p prox_s[p,r] + N_q prox_s[q,r]}{N_p + N_q}$$

   Then row and column p, corresponding to cluster p, is deleted form the distance matrix C, i.e. size of distance matrix is reduced.
5. If there are more than one cluster, go to step 3

5 Compression with clustering support

Ordering of input documents was not taken in consideration in general description of word-base compression methods. The compression method works correctly for any ordering of documents. Probably the simplest ordering of input documents is time ordering, i.e. the documents are compressed in the same order as they are added to textual database. Seeing that compression methods are based on searching of repeated parts of texts, it is easy to see, that this ordering is not necessary the best possible. Improvement of compression performance can be achieved by reordering of input documents. Better ordering of input documents moves similar documents to one another.

Similar documents are grouped together using cluster analysis 4. Of course cluster analysis is very time consuming so that it is counterproductive to perform the analysis only to enhance compression performance. But when compression method for IRS is developed, results of cluster analysis can be used in query processing [8, 19] and vice versa, cluster analysis originally devoted to query processing can be incorporated to compression.

To group similar documents together, agglomerative clustering algorithm described in section 4 was used. But the question how to convert hierarchical tree structure of clusters to linear list of documents still remains. The ultramnetric tree was created during clustering. We can used this fact and for list of documents L_X for compression used ultrametric ball query: $B_X(x,r)$, where r is maximal distance in ultrametric tree. L_X be sorted before compression aided distance $d(x,z)$ where $z \in L_X$.

Two strategies were used to reorder collection of documents entering the compression process:

Most Similar Left (MSL) – x in $B_X(x,r)$ is leftmost document in the ultrametric tree.

Most Similar Right (MSR) – x in $B_X(x,r)$ is rightmost document in the ultrametric tree.

6 Experimental Results

Some experiments were done to test impact clustering on word-based compression methods. Both compression methods were used in our tests. Two large text files were used for our tests: latimes.txt coming from TREC corpus [14], and enron.txt, which consists of emails from Enron email corpus[3]. In file latimes.txt individual documents are represented by each newspapers article and ordering is determined by date of publication. Each individual email represents document in file enron.txt, and ordering is defined as alphabetical ordering of users in Enron corpus. Results for this type of ordering without ordering is provided in Table 1.

The following notation will be used to describe results of experiments:

- CS is the size of compressed file
- CS_α, where $\alpha \in \{WLZW, WBW, GZIP, BZIP2\}$ is the size of compressed file without clustering, see Table 1
- $\Delta_{CS} = \frac{CS_\alpha - CS}{CS_\alpha} \times 100\%$
- $CR = \frac{CS}{S_0} \times 100\%$ is compression ratio
- $\Delta_{CR} = CR_\alpha - CR$, where $\alpha \in \{WLZW, WBW, GZIP, BZIP2\}$

Δ values represents difference between given value and corresponding value in compression without clustering. Positive Δ value means that given value is worse than original value, negative value means than new value is better than original one.

Table 1. Compression without clustering

		latimes.txt	**enron.txt**
Original size [bytes]	S_0	498,360,166	886,993,953
WLZW method			
Compressed size [bytes]	CS_{WLZW}	158,017,940	207,908,560
Compression ratio [%]	CR_{WLZW}	31.708	23.440
WBW method			
Compressed size [bytes]	CS_{WBW}	110,246,524	167,099,129
Compression ratio [%]	CR_{WBW}	22.122	18.839
Gzip			
Compressed size [bytes]	CS_{GZIP}	175,864,812	228,953,895
Compression ratio [%]	CR_{GZIP}	35.289	25.812
BZip2			
Compressed size [bytes]	CS_{BZIP2}	131,371,338	164,720,382
Compression ratio [%]	CR_{BZIP2}	26.361	18.571

The first experiments are focused on the file latimes.txt. This file is relatively large. The size of documents (newspapers articles) varies from two to eight kilobytes. Compression with clustering and five random permutations were tested.

[3] Duplicate emails were deleted before processing.

It is easy to see from Table 2, that clustering brings positive results in terms of compression ratio. The size of the compressed text is about 4% less than the original size in the WLZW methods, and about 5% smaller than the original one in the WBW method. The compression ratio improves to cca 1.2% with respect to original values in both cases.

Table 2. Impact of clustering on compression

	WLZW method				
Cluster strategy on file	CS [bytes]	$CS_\alpha - CS$ [bytes]	Δ_{CS} [%]	CR [%]	Δ_{CR} [%]
MSL latimes.txt	151,869,588	-6,148,352	-3.891	30.474	-1.234
MSR latimes.txt	151,973,800	-6,044,140	-3.825	30.495	-1.213
MSL enron.txt	187,951,820	-19,956,740	-9.599	21.19	-2.25
	WBW method				
Cluster strategy on file	CS [bytes]	$CS_\alpha - CS$ [bytes]	Δ_{CS} [%]	CR [%]	Δ_{CR} [%]
MSL latimes.txt	104,701,332	-5,545,192	-5.03	21.009	-1.113
MSR latimes.txt	104,706,446	-5,540,078	-5.025	21.01	-1.112
MSL enron.txt	132,707,295	-34,391,834	-20.582	14.961	-3.877
	GZip method				
Cluster strategy on file	CS [bytes]	$CS_\alpha - CS$ [bytes]	Δ_{CS} [%]	CR [%]	Δ_{CR} [%]
MSL latimes.txt	164,298,043	-11,566,769	-6.577	32.968	-2.321
MSR latimes.txt	164,322,641	-11,542,171	-6.563	32.973	-2.316
MSL enron.txt	153,765,189	-75,188,706	-32.84	17.336	-8.477
	Bzip2 method				
Cluster strategy on file	CS [bytes]	$CS_\alpha - CS$ [bytes]	Δ_{CS} [%]	CR [%]	Δ_{CR} [%]
MSL latimes.txt	120,149,683	-11,221,655	-8.542	24.109	-2.252
MSR latimes.txt	120,154,853	-11,216,485	-8.538	24.11	-2.251
MSL enron.txt	122,024,594	-42,695,788	-25.92	13.757	-4.814

Better results were achieved for file enron.txt, see Table 2. The improvement of compression ratio is cca 2 % with respect to the original compressed size in the WLZW method, and cca 4 % in the WBW method.

Random permutations deteriorate compression in all cases (see Tables 3, and 4). These negative results mean that clustering has measurable impact on compression performance, and the positive results of regarding cluster supported compression are not coincidental.

The results of standard GZip and BZip2 compression utilities provide data for comparison with our proposed word-based compression methods. As can be seen from tables, character of these results is very close to our methods; therefore clustering has serious impact on compression regardless of selected compression method.

Table 3. File latimes.txt: random permutations

	WLZW method				
Permutation	CS [bytes]	$CS_\alpha - CS$ [bytes]	Δ_{CS} [%]	CR [%]	Δ_{CR} [%]
1	160,417,812	2,399,872	1.519	32.189	0.481
2	160,456,620	2,438,680	1.543	32.197	0.489
3	160,448,056	2,430,116	1.538	32.195	0.487
4	160,456,564	2,438,624	1.543	32.197	0.489
5	160,475,324	2,457,384	1.555	32.201	0.493
Average	160,450,875	2,432,935	1.54	32.196	0.488
	WBW method				
Permutation	CS [bytes]	$CS_\alpha - CS$ [bytes]	Δ_{CS} [%]	CR [%]	Δ_{CR} [%]
1	111,686,104	1,439,580	1.306	22.411	0.289
2	111,713,942	1,467,418	1.331	22.416	0.294
3	111,718,068	1,471,544	1.335	22.417	0.295
4	111,717,879	1,471,355	1.335	22.417	0.295
5	111,712,566	1,466,042	1.330	22.416	0.294
Average	111,709,712	1,463,188	1.327	22.415	0.293
	GZip method				
Permutation	CS [bytes]	$CS_\alpha - CS$ [bytes]	Δ_{CS} [%]	CR [%]	Δ_{CR} [%]
1	182,350,555	6,485,743	3.688	36.590	1.301
2	182,612,870	6,748,058	3.837	36.643	1.354
3	182,626,115	6,761,303	3.845	36.645	1.357
4	182,616,966	6,752,154	3.839	36.644	1.355
5	182,616,986	6,752,174	3.839	36.644	1.355
Average	182,564,698	6,699,886	3.810	36.633	1.344
	BZip2 method				
Permutation	CS [bytes]	$CS_\alpha - CS$ [bytes]	Δ_{CS} [%]	CR [%]	Δ_{CR} [%]
1	133,747,217	2,375,879	1.809	26.837	0.477
2	133,859,533	2,488,195	1.894	26.860	0.499
3	133,848,650	2,477,312	1.886	26.858	0.497
4	133,864,200	2,492,862	1.898	26.861	0.500
5	133,854,622	2,483,284	1.890	26.859	0.498
Average	133,834,844	2,463,506	1.875	26.855	0.494

Table 4. File enron.txt: random permutations, WLZW method

Permutation	CS [bytes]	$CS_\alpha - CS$ [bytes]	Δ_{CS} [%]	CR [%]	Δ_{CR} [%]
1	242,459,136	34,550,576	16.618	27.335	3.895
2	249,122,668	41,214,108	19.823	28.086	4.646
3	250,203,876	42,295,316	20.343	28.208	4.768
4	250,342,664	42,434,104	20.41	28.224	4.784
5	250,511,920	42,603,360	20.491	28.243	4.803
Average	248,528,052	40,619,492	19.537	28.019	4.579

7 Conclusion and future works

Word-based compression methods combined with cluster analysis of input document have been presented in this paper. These compression methods are suitable especially for IRS. Experimental results prove that clustering has a positive impact on the compression ratio.

Acknowledgement

This work is supported by Grant of Grant Agency of Czech Republic No. 201/05/P14

References

1. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, New York, 1999.
2. J. L. Bentley, D. D. Sleator, R. E. Tarjan, and V. K. Wei. A locally adaptive data compression scheme. *Communications of the ACM*, 29(4):320–330, 1986.
3. M. W. Berry and M. Browne. *Understanding Search Engines*. SIAM Society for Industrial and Applied Mathematics, Philadelphia, 1999.
4. N. Brodskiy, J. Dydak, J. Higes, and A. Mitra. Dimension zero at all scales. *ArXiv Mathematics e-prints*, July 2006.
5. E. S. de Moura, G. Navarro, N. Ziviani, and R. Baeza-Yates. Fast searching on compressed text allowing errors. In W. B. Croft, A. Moffat, C. J. van Rijsbergen, R. Wilkinson, and J. Zobel, editors, *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 298–306. ACM Press, New York, 1998.
6. J. Dvorský. Word-based compression methods for text retrieval systems. In *WDS'98*, Praha, 1998. ISBN 80-85863-29-4.
7. J. Dvorský. *Word-based Compression Methods for Information Retrieval Systems*. Phd thesis, Charles University Prague, 2004.
8. J. Dvorský, J. Martinovic, and V. Snásel. Query expansion and evolution of topic in information retrieval systems. In V. Snásel, J. Pokorný, and K. Richta, editors, *DATESO*, volume 98 of *CEUR Workshop Proceedings*, pages 117–127. CEUR-WS.org, 2004.
9. J. Dvorský, J. Pokorný, and V. Snášel. Word-based compression methods with empty words and nonwords for text retrieval systems. In *Datasem'98*, Brno, 1998.
10. J. Dvorský, J. Pokorný, and V. Snášel. Compression methods for large multilingual text document. In *Proceedings of 1999 International Symposium on Database, Web and Cooperative Systems*, pages 158–163, Baden-Baden, 1999.
11. J. Dvorský, J. Pokorný, and V. Snášel. Word-based compression methods and indexing for text retrieval systems. In J. Eder, I. Rozman, and T. Welzer, editors, *Proceedings of ADBIS 99*, number 1691 in Lecture Notes in Computer Science, pages 75–84. Springer-Verlag, Berlin, 1999.
12. J. Dvorský and V. Snášel. Modifications in Burrows-Wheeler compression algorithm. In *Proceedings of ISM 2001*, pages 29–35, Ostrava, 2001. ISBN 80-85988-51-8.

13. J. Dvorský, V. Snášel, and J. Pokorný. Word-based compression methods for large text documents. In *Data Compression Conference - DCC '99*, page 523, Snowbird, Utah, USA, 1999.
14. D. Harman, editor. *The Forth REtrieval Conference (TREC-4)*. National Inst. of Standards and Technology, Gaithersburg, USA, 1997.
15. R. L. Haskin. Special-purpose processors for text retrieval. *Database Engineering*, 4(1):16–29, 1981.
16. R. N. Horspool and G. V. Cormack. Constructing word-based text compression algorithms. In J. A. Storer and M. Cohn, editors, *Proc. 1992 IEEE Data Compression Conference*, pages 62–71. IEEE Computer Society Press, Los Alamitos, California, Mar. 1992.
17. S. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32:241–254, 1967.
18. J. Lansky and M. Zemlicka. Text compression: Syllables. In Richta et al. [24], pages 32–45.
19. J. Martinovic and P. Gajdos. Vector model improvement by fca and topic evolution. In Richta et al. [24], pages 46–57.
20. A. Moffat. Word-based text compression. *Software-Practice and Experience*, 19(2):185–198, 1989.
21. A. Moffat, R. M. Neal, and I. Witten. Arithmethic coding revisited. *ACM Transactions on Information Systems*, 16(3):256–294, 1998.
22. A. Moffat, J. Zobel, and N. Sharman. Text compression for dynamic document databases. *Knowledge and Data Engineering*, 9(2):302–313, 1997.
23. F. Murtagh. On ultrametricity, data coding, and computation. *Journal of Classification*, Volume 21, Number 2 / September:167–184, 2004.
24. K. Richta, V. Snásel, and J. Pokorný, editors. *Proceedings of the Dateso 2005 Annual International Workshop on DAtabases, TExts, Specifications and Objects, Desna, Czech Republic, April 13-15, 2005*, volume 129 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005.
25. B. Y. Ryabko. Technical correspondence: A locally adaptive data compression scheme. *Communications of the ACM*, 30(9):792, 1987.
26. I. Witten, A. Moffat, and T. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Van Nostrand Reinhold, 1994.
27. J. Zobel and A. Moffat. Adding compression to a full-text retrieval system. *Software-Practice and Experience*, 25(8):891–903, 1995.

Work with Knowledge on the Internet – Local Search

Antonín Pavlíček, Josef Muknšnábl

Department of System Analysis, Faculty of Informatics and Statistics,
University of Economics, Prague, W. Churchill sq. 4, 130 67, Prague, Czech Republic
{antonin.pavlicek, muknsj}@vse.cz

Abstract. Authors are looking within their research grant new original web local search algorithm respecting specifics of Czech national environment. We would like to initiate further debate on topic. We are addressing three subtasks that include: identification of user geographical location, identification of web locality and final algorithm design working with these information altogether.

1 Preamble

A staggering pace of internet growth together with steadily increasing broadband penetration availability and general information literacy lead to more frequent internet usage. Such trend is not visible only in US but worldwide too - number of internet users and overall usage numbers constantly grow[1]. Although capabilities of engines and catalogs have improved significantly within last several years (especially after Google ranking algorithm arrival) they are still not perfect in terms of accuracy and relevancy. Typical areas where there is a potential for improvement are e.g. personalized and local search (in terms of geography and regions). Local search is a matter of an internal research grant that has been launched these days at University of Economics, Prague by us. A focus on that topic is not rare, especially in global scale, as several patents related to local search have already been filed[2] in US. Our main goal is to design and implement new web local search algorithm that will respect Czech national specifics and verify its function on local web page catalog Jihozapad.info. We would like to indicate possible ways of solution by the article in hope that some wider discussion bringing new ideas will be initiated.

2 Local Search and its possibilities

Web local search is a type of search when user is trying to find not only topic relevant but also locally (in terms of geographical distance) relevant web page/pages. Typically the users are searching for local/regional pages related to local businesses, local authorities or local events. Local search could be achieved by several ways. The most common one is by specification of country/state/area/district/city/village name (or other local information such as ZIP code) in query that is submitted to search site.

© J. Pokorný, V. Snášel, K. Richta (Eds.): Dateso 2007, pp. 127–131, ISBN 80-7378-002-X.
VŠB – Technical University of Ostrava, Dept. of Computer Science, 2007.

Other one is that the search site recognizes user's physical location and will offer results relevant to recognized position only. The type of way is used depends on type of search site used. All major players in search engines/web catalog branch on global/Czech level offer local search tools. Let remind at least Google Maps, Yahoo! Local, MSN Live Search, from local Czech search sites mapy.cz and centrum.cz.
As latest numbers indicate an interest in local searching (geo-searching) is not a fiction or wish but a reality everyone has to count with. For example some recent poll, provided by comScore (Global Internet Information Provider) says[6] that more than 109 million of people performed about 849 millions of local searches in July 2006 which also represents 43% year over year increase. Most of the users, about 41 % were searching for items such as car rental office or dry cleaner [6]. A split by particular search engines / portals looks like this [6]: Google sites 29,5 %, Yahoo sites 29,2 %, Microsoft 12,3 %, Time Warner Network 7,1 %, Verizon Communications 6.6 %, YellowPages.com 3.9 %, Ask Network 2.7 %, Local.com 1.9 %, InfoSpace Network 1.9 %, DexOnline.com 1.4 %, all other sites 3.2 percent.
Such trend is confirmed by other polls and studies and is generally accepted and confirmed within whole IT/marketing industry[7].

3 Our initial conditions

As already mentioned within preamble we've decided to go the way of establishing new improved local search algorithm. That algorithm should be implemented in local web catalog/directory called jihozapad.info and its results verified within set of jihozapad.info registered www links.
Web catalog / directory jihozapad.info is primarily focused on area of South-West Bohemia (part of The Czech Republic). It contains primarily www links related to local subjects such as stores, companies, authorities etc. It geographically covers an area about 17 617 km² with about 1 180 541 inhabitants (population density is 67 inhabitants / km²). The catalog was launched in August 2005 and its 12 month-average unique visitor number is 1458 visitors per month. Catalog and its interface is primarily available in Czech, other available language is German, as covered area directly neighbor with Germany. Surprisingly most of visitors is from US (62 %)[1] followed by The Czech Republic (16%). Number of German visitors is quite insignificant (about 1%)!
There are 121 registered users and 1148 registered local web links. Locality is in catalog presented by possibility to determine district within which searching should be performed (there are actually four main districts called Klatovy [KT], Domažlice [DO], Strakonice [ST] and Plzeň-jih [PJ]).[2] A catalog has no true search engine at the moment, all links are added and registered only by registered users (approved by portal administrator)

[1] Robots are excluded.
[2] Information about district is available for all registered links. It is a mandatory attribute.

4 Problem decomposition

4.1 Identification of user geographical location

is also called geo-location. Typically geo-location of users is derived from their IP addresses (or MAC address). Such service is often available on commercial basis (such as IP2Location, MaxMind etc.). However it will not be very likely our case due to from our perspective high cost of such services. We'll try to discuss that with local providers and agree on some cooperation at this point. A level of details that we can obtain from IP address will depend on quality of service/database it will be for such purpose used. The easiest way task is to obtain name of the country (IP registrars supply that information for free), the more difficult is to get some other details such as region state, province/district, city, latitude/longitude etc. Other possible and used way of determining locations of user is to use information that user provided us during portal registration (such as address, ZIP code, phone numbers, GPS coordinates etc.). The problem is that number of registered users will be very likely much smaller than number of visitors, so its capabilities will be rather limited comparing the first mentioned method. Very likely combined approach will be chosen.

4.2 Web page link and its relation to particular geographical area

There are many ways that can help us to determine web page locality. We've thought about following, so far:

Use information provided by web page owners: there is information about district for each registered link right now in Jihozapad.info. We do not consider this as fully sufficient and there has been implemented an improvement leading to make location of www links more precise these days. We still will come from information that will be entered by user during link registration but this information will be more detail and will be expressed in a standard way. As an appropriate standard we have chosen split into geographical areas based on EU legal framework for the geographical division of the territory of the European Union also know as NUTS [8]. There will be a possibility to enter for one www link more geographical locations as one www link may represent a company with different stores within region (for example www.welstam.cz). Following NUTS information will be gathered:

- NUTS1_uzemi: Česká republika (same for all registered link)
- NUTS1_kod: CZ01 (will be the same for all registered link)
- NUTS2_oblast: Jihozápad (will be the same for all registered link)
- NUTS2_kod: CZ03 (will be the same for all registered link)
- NUTS3_kraj: Jihočeský kraj / Plzeňský kraj
- NUTS3_kod: CZ031 / CZ032
- NUTS4_okres: Strakonice / Domažlice / Klatovy / Plzeň-jih
- NUTS4_kod: CZ0316/ CZ0321 / CZ0322 / CZ0324

Such information will be also enhanced by particular address in form: City/Town, Street house number, ZIP code. Also information about latitude/ longitude and altitude will be gathered include precise GPS coordination (WGS-84). We strongly hope that all gathered information will help in providing better result on local search.

Use local specialties from web page content: Such approach is applicable in the case of automated geo-spatial search engine (which is apparently not the case of such improvement because of time restrictions). The idea is to search particular web page (include all subpages) for existence of unique local words such as addresses parts (village/town/district/area names), dialect words, ZIP codes, dial codes and derive web page locality from occurrence frequency of such words (or via other algorithm). Situation in that might be complicated by fact that many addresses can be found on webpage. However as Jihozapad.info is strictly oriented on region of South-West Bohemia (districts Klatovy, Domažlice, Strakonice, Plzeň - Jih), found addresses from other regions could be ignored. Similar algorithm to "Geographic Scope[3]" developed by Kyoto researches could be applied or other algorithms coming of data-mining techniques such as association analysis, clustering methods[4] etc.
Cooperation with local webhosting providers: identification and focus on local webhosting servers where there can local content will be very likely stored. For example local Webhosting provider ŠumavaNet contains lot of regionally oriented web pages. Webhosters also could become partners in gathering locally oriented content, via some unified interface for example.
Supporting and propagating standards helping in geo-location: jihozapad.info should be prepared to extract web page locality from some HTML-GEO formats/protocols such as Microformats hCard [5] (extension of item a) or cooperate in exchange of geo-spatial data associated to GIS systems distributed in a set of predefined formats. It would significantly improve catalog accuracy however because of timing restrictions it will not be possible the case.
Although there are many ways by which we can determine web page locality, no one of them guarantees for 100% the result. The reasons for that may vary. Many regional web pages, even those locally oriented don't contain any significant information about their origin (they can be just topic oriented). Many of them are locality independent and finally quality of locality information derived by using methods mentioned above doesn't need to be sufficient for locality determination.

4.3 A final search algorithm structure

These days jihozapad.info offers its users „district" level of detail in relation to registered web pages. This granularity is of course not sufficient for being real locally oriented search site and improvements have already started to be implemented. Having all information about users accessing jihozapad.info and locality of registered web pages we can think about appropriate algorithm. At this moment we think of some kind of Google style ranking algorithm with different weights for particular levels of granularity (region/district/town/street) and specific metrics for deriving web page importance in given area (pages with links from other pages same district/region/town etc. would be considered as more relevant).

5 Conclusion

To find a good algorithm for local searching is a complex task that combines methods from many areas such as data mining, web pages constructions, search engine principles etc. We are tat the beginning right now, all methods mentioned in our article would help us, finding and optimal balance that will provide the most relevant and accurate result will be matter of real algorithm tuning on real data.

References

1. Market Research, *Internet World Stats – Usage and Population Statistics* [online]. [cit. 2006-12-20]. URL: <http://www.internetworldstats.com/stats.htm>.
2. SLAWSKI, William, *Assigning Geographic Locations to Web Pages* [online]. [cit. 2006-12-28]. URL: <http://www.seobythesea.com/?p=386>
3. YAMADA, Naoharu – LEE, Ryong – KAMBAYASHI, Yahiko. *Classification of Web Pages with Geographic Scope and Level of Details for Mobile Cache Management, Proceedings of the Third International Conference on Web Information Systems Engineering (Workshops)* 0-7695-1754-3/02, 2002 IEEE, [online]. [cit. 2006-12-20]. URL: <http://csdl.computer.org/dl/proceedings/wisew/2002/1813/00/18130022.pdf>
4. HAN, Jiawei – KAMBER, Micheline. Data Mining: Concepts and Techniques. San Diego,(CA), USA: Academic Press, 2001, 550 s., ISBN 1-55860-489-8
5. *hCard Description* [online]. [cit. 2006-12-20]. URL: <http://microformats.org/wiki/hcard>.
6. *comScore: Local Web Searching Soars* [online]. [cit. 2006-10-02]. URL: <http://www.mediaweek.com/mw/search/article_display.jsp?vnu_content_id=1003188359&schema=>.
7. *New Developments in Local Search, Part 4* [online]. [cit. 2003-11-19]. URL: <http://www.clickz.com/showPage.html?page=clickz_print&id=3110641>.
8. *Common classification of territorial units for statistical purposes* [online]. [cit. 2006-02-06]. URL: < http://europa.eu/scadplus/leg/en/lvb/g24218.htm>.

The Use of Ontologies in Wrapper Induction

Marek Nekvasil

Department of Information and Knowledge Engineering, University of Economics,
Winston Churchill Sq. 4, 130 67, Prague 3, Czech Republic
nekvasim@vse.cz

Abstract. The purpose of this entry is to bring in an extension of ontologies so that they can be utilized in the process of automated information extraction from the web documents. Major part of it is dedicated to a proposition and derivation of an inference model for evaluation of the pattern matches and their combination. Further is proposed a simple naïve method of wrapper induction which is able to use the results of the first part.

Keywords: ontology, automatic annotation, information extraction, wrapper

1 Introduction

One of the simplest alternatives to the manual handling of web documents is the use of *wrappers*, sets of rules to identify the desired values in the document. These rules can be created either by hand or automatically, in which case we are talking about *wrapper induction* [3]. For automatic wrapper induction in principle some examples of the real data to be extracted are needed along with their respective context, which form basically an annotated document. The purpose of this entry is to propose a concept by which it should be possible to annotate the documents automatically with the use of *ontologies*. We will call any ontology that is designed to this use in information extraction an *extraction ontology*.

2 Extending an OWL ontology

An ontology written in a bare OWL language has very limited capabilities of describing the possible values of datatype properties (properties which's value is a literal) and therefore does not contain enough information to identify these inside a document. To remove this insufficiency we introduce an extension which will enable us to append a *pattern* of typical values to each datatype property.
Any general rule for which it is possible on any continuous part of a document (in terms of a string of characters or words) to determine to what extent it is satisfied we call a *pattern*. An example of a pattern can be a rule that evaluates whether a given part of a document is a number from a given range or a string from a given list.
From now on we will distinguish between two kinds of patterns. Foremost there will be simple patterns, or rather *atomic patterns* that will be formed by a simple rule, which can be directly evaluated on a part of a document. An example of atomic patterns can be the aforementioned patterns that match a number or a string.
Moreover there will be *composite patterns*, which we will define as such a combination of rules that can be evaluated on a given continuous part of a document as a whole. As such, the composite pattern will be always a combination of other

patterns, both atomic and composite. The composite patterns can be hierarchically combined which can be of significant concern in some specific situations.
As it has sense to assign only one pattern per datatype property, more patterns will have to be joined via including them in some composite pattern. Every part of a document that matches a given pattern, i.e. the pattern rule evaluates positively on that part, will be considered a suspected *partial candidate* for the occurrence of the value of datatype property the pattern is assigned to. If that given pattern is the one that is assigned directly to the datatype property, every matching part of the document will be considered to be a suspected *candidate* for the occurrence of the value.

2.1 Atomic Patterns

While evaluating the match of atomic patterns we encounter the problem of deriving the certainty degree of marking the candidate. We take a pattern as an algorithm that can tell for every place in the document to what extent the rule is satisfied depending on its parameters. Here we have two distinct terms, the *degree of pattern match* which represents the certainty with which the pattern's algorithm marked the given place in the document, and the *certainty degree of marking the partial candidate for the value of a certain datatype property* which represents the certainty that the given place in the document really is the occurrence of the value, given sole by the observation of the single pattern and independently of any other patterns. We will denote marking the partial candidate as the *pattern evidence* and therefore the second term will be equivalent to a *degree of pattern evidence*.
The degree of pattern match and the degree of pattern evidence should intuitively correspond. If we denote the pattern match as $\boldsymbol{A}$ and the pattern evidence as $\boldsymbol{E}$ we can write down this inference rule:

$$\boldsymbol{A \rightarrow E} \quad \textbf{(1)}$$

We have chosen for our purposes a fuzzy logic inference model [2], but it should be possible to use any other. In fuzzy logic we can define $\boldsymbol{A}$ and $\boldsymbol{E}$ as propositions

- $\boldsymbol{A}$ – "*The pattern has marked the given place in the document*."
- $\boldsymbol{E}$ – "*The marked place is really a pattern evidence*"

and corresponding degrees as their truth values (i.e. degree of pattern match $a=val(\boldsymbol{A})$ and the degree of pattern evidence $e=val(\boldsymbol{E})$). We introduce also two universal parameters for every pattern, namely *precision* and *cover* and we define them:

$$p = val(\boldsymbol{A \rightarrow E}) - \textit{pattern precision} \quad \textbf{(2)}$$

$$c = val(\boldsymbol{E \rightarrow A}) - \textit{pattern cover} \quad \textbf{(3)}$$

While using these parameters we can derive on Łukasiewicz fuzzy logic this form of the inference rule (the detailed derivation is available in [6]):

$$\boldsymbol{((A \,\&\, (A \rightarrow E)) \vee \neg(E \rightarrow A)) \Rightarrow E} \quad \textbf{(4)}$$

If we take into account the prescriptions of p and c and do not overestimate the degree of pattern evidence e, we get:

$$e = max(a + p - 1,\ 1 - c) \quad \textbf{(5)}$$

With the use of parameter p we can set a top limit to the degree of a given pattern evidence. The pattern precision denotes in this context a certainty with what the high degree of pattern match leads to a high degree of pattern evidence. The c parameter sets the lowest possible value of the degree of pattern match.

2.2 Composite patterns

Would we like to combine the evidences of multiple patterns it will be a task in form of a set operation. For this purpose just two set operations come on force, namely union and intersection, which in combination with the complement operation can form any other set operation possible. The determination of the degree of composite pattern evidence itself is then a trivial matter. The degree of composite pattern match will be determined by simply assembling the degrees of evidences of the partial patterns with the appropriate logical operation, thus there will be conjoint patterns and disjoint patters (and possibly negating patterns). From the pattern match defined in this way we get the composite pattern evidence by the same way as we did in case of atomic patterns.
It is interesting to discuss the meaning of precision and cover parameters of the partial patterns in contrast with the use of the different kinds of composite patterns. In case of a disjoint composite pattern the high value of p implies that the partial pattern is a sufficient condition and in case of conjoint composite patterns the high value of c means that the partial pattern forms a necessary condition.

2.3 Designing the patterns

We will denote the patterns in the extraction ontology as XML elements from a special namespace nested in the elements of the datatype properties. The extent of the patterns can vary from distinguishing time values, named entities to patterns that evaluate the context or format of the document. A few basic patterns are proposed in [6], however many others are possible. While designing a new pattern it is needed to keep in mind the way it evaluates and think carefully the possibilities of its combination of other existing patterns.

3 A simple wrapper induction method

By applying the rules of patterns on the content of a document we get a set of evidences along with their certainty degrees for every datatype property in the extraction ontology with a pattern assigned. If we rely on tabular structure of data we can try to separate the evidences in a few segments according to the resemblance of their XPath. We can purge the sets of evidences if we realize that the precision attribute specifies the mean ratio of evidences that are marked correctly by the pattern. Therefore up to $1-p$ of evidences supplied by this pattern can be false and hence we can remove that much of the worst segments. If the data are stored in the tabular structure the relevant parts of text are generally contained in the same structure of elements that is not changing throughout the segment. On the level of XPath expression this will show up as a single changing index in the absolute path by omitting which we get a set of elements that would ideally all contain the value of the respective property.
The cover parameter is the mean rate of the evidences that the pattern identifies to the total real number of occurrences of the respective property. While the generalized XPath expression should identify all possible occurrences of the property we can

calculate the proportion of evidences of the pattern to this "complete" set and difference from the parameter *c* represent the error caused by generalizing the paths. Based on the number of evidences in segments and the respective absolute XPath we can assign the corresponding segments of different properties and form the instances of extracted class.
To sum up this approach is just a simple method and has many limitations. Besides that this method can extract only properties with cardinality 1 (the tabular structure) it is also limited in its tolerance to the irregularities in the structure of the document, on the other hand to the irregularities in the extracted values it is rather resistant.

4 Conclusion and future work

The proposed method of pattern notation allows hierarchical combining of partial patterns and is open to the possibility of designing additional patterns according to one's need. Similar approach is taken by [4] and [5], however unlike them we do not design proprietary formats of ontologies but try to start from OWL standard.
The limitation of the proposed wrapper induction method is the fact that it relies on the tabular structure of extracted data but the extraction is completely automatic and with proper setting of the attributes allows the estimation of extraction error.
To propose a way of automatic learning of the patterns or at least of their parameters could be an interesting subject of future work

Acknowledgement
The research leading to this paper was supported by the European Commission under contract FP6-027026, Knowledge Space of semantic inference for automatic annotation and retrieval of multimedia content, K-Space.

Reference

1. Anton T.: XPath-Wrapper Induction by generalizing tree traversal patterns, in: Antoniou, G., van Harmelen, F.: *A Semantic Web Primer*, Cambridge MA.: MIT Press, 2004, ISBN 0-262-01210-3
2. Hájek P.: *Metamathematics of fuzzy logic*, Dordrecht: Kluwer, 1998, ISBN: 0-792-35238-6
3. Kushmerick, N.: *Wrapper induction for information extraction*, PhD thesis, University of Washington, 1997
4. Labský M., Svátek V.: *On the Design and Exploitation of Presentation Ontologies for Information Extraction*, ESWC'06 Workshop on Mastering the Gap: From Information Extraction to Semantic Representation, Budva, Montenegro, 2006
5. Muslea, I., Minton, S., Knoblock, C.: *A Hierarchical Approach to Wrapper Induction*, *3rd Conference on Autonomous Agents*, 1999, http://www.isi.edu/~muslea/papers.html
6. Nekvasil M., *Využití ontologií při indukci wrapperů*, diplomová práce, VŠE, Praha 2006

Author Index

Dvorský, Jiří, 115

Eckhardt, Alan, 103

Hoksza, David, 67

Chernik, Katsiaryna, 1

Klíma, Jan, 89
Kuthan, Tomáš, 21

Lánský, Jan, 1, 21
Loupal, Pavel, 11

Martinovič, Jan, 115
Muknšnábl, Josef, 127

Nečaský, Martin, 35
Nekvasil, Marek, 132
Novotný, Tomáš , 55

Pavlíček, Antonín, 127

Skopal, Tomáš, 67, 89

Toth, David, 81

Vlčková, Zuzana, 1
Vraný, Jan, 47

Žák, Jan, 47