Unordered XML Pattern Matching with Tree Signatures (Extended Abstract) [⋆]

Pavel Zezula[1], Federica Mandreoli[2], and Riccardo Martoglia[2]

[1] Masaryk University, Brno, Czech Republic
`zezula@fi.muni.cz`
[2] University of Modena and Reggio Emilia, Modena, Italy
{`mandreoli.federica, martoglia.riccardo`}`@unimo.it`

Abstract. We propose an efficient approach for finding relevant XML data twigs defined by unordered query tree specifications. We use the tree signatures as the index structure and find qualifying patterns through integration of structurally consistent query path qualifications. An efficient technique is proposed and its implementation tested on real-life data collections.

1 Introduction

With the rapidly increasing popularity of XML for data representation, there is a lot of interest in query processing over data that conform to the *labelled-tree* data model. The idea behind evaluating tree pattern queries, sometimes called the *twig queries*, is to find all existing ways of embedding the pattern in the data. Since XML data collections can be very large, efficient evaluation techniques for tree pattern matching are needed.

From the formal point of view, XML data objects can be seen as ordered labelled trees. Following this model, previous approaches considered also the query trees ordered, so the problem can be characterized as the *ordered tree pattern matching*. Though there are certainly situations where the ordered tree pattern matching perfectly reflects the information needs of users, there are many other that would prefer to consider query trees as unordered. For example, when searching for a twig of the element `person` with the subelements `first name` and `last name` (possibly with specific values), ordered matching would not consider the case where the order of the `first name` and the `last name` is reversed. However, this could exactly be the person we are searching for. The way to solve this problem is to consider the query twig as an unordered tree where only the *ancestor-descendant* relationships are important – the *preceding-following* relationships are unimportant.

In general, the process of unordered tree matching is difficult and time consuming. For example, the *edit distance* on unordered trees was found in [ZSS92] NP hard. To improve efficiency, an approximate searching for nearest neighbors, called ATreeGrep, was proposed in [SWS$^+$02]. However, the problem of unordered twig pattern matching in XML data collections has not been studied, to the best of our knowledge.

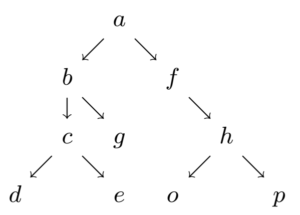
In this paper we propose an efficient evaluation of the unordered tree matching. We use the tree signature approach [ZAD03], which has originally been proposed for the ordered tree matching. In principle, we decompose the query into a collection of root to leaf paths and search for their embedding in the data trees. Then we join the *structurally consistent* path qualifications to find unordered query tree inclusions in the data.

[⋆] An extended version of this paper has been presented as invited talk at the SOFSEM 2004 Conference [ZMM04]

The rest of the paper is organized as follows. In Section 2, we summarize the concepts of tree signatures and define their properties that are relevant towards our objectives. In Section 3, we analyze the problem of unordered tree matching and of the efficient computation of the query answer set. Performance evaluations are presented in Section 4. Final conclusions are in Section 5.

2 A Brief Introduction to Tree Signatures

The idea of *tree signatures* proposed in [ZAD03] is to maintain a small but sufficient representation of the tree structures able to decide the ordered tree inclusion problem for the XML data processing. As a coding schema, the *preorder* and *postorder* ranks [Die82] are used. In this way, ordered labelled trees are linearized, and extended string processing algorithms are applied to identify the tree inclusion. We recall that an ordered tree T is a rooted tree in which the children of each node $v \in T$ are uniquely identified, left to right, as $i_1, i_2, \ldots, i_k$ (k is the number of children) and that T is a labelled tree if it associates a label (name) $t_v \in \Sigma$ (the domain of tree node labels) with each node $v \in T$. For illustration, see the preorder and postorder sequences of our sample ordered labelled tree in Fig. 1 – the node's position in the sequence is its preorder/postorder rank, respectively. For instance $pre(a) = 1$ and $post(a) = 10$.



$pre:$	a	b	c	d	e	g	f	h	o	p
$post:$	d	e	c	g	b	o	p	h	f	a
$rank:$	1	2	3	4	5	6	7	8	9	10

Fig. 1. Preorder and postorder sequences of a tree

Definition 1. *Let T be an ordered labelled tree. The signature of T is a sequence, $sig(T) = \langle t_1, post(t_1); t_2, post(t_2); \ldots t_n, post(t_n) \rangle$, of $n = |T|$ entries, where t_i is a name of the node with $pre(t_i) = i$ (i is the position of the entry in the sequence).*

For example, $\langle a, 10; b, 5; c, 3; d, 1; e, 2; g, 4; f, 9; h, 8; o, 6; p, 7 \rangle$ is the signature of the tree from Fig. 1.

By extending entries of tree signatures with two preorder numbers representing pointers to the *first following*, ff, and the *first ancestor*, fa, nodes, the *extended signatures* are also defined in [ZAD03]. The generic entry of the i-th extended signature is $\langle t_i, post(t_i), ff_i, fa_i \rangle$. Such version of the tree signatures makes possible to compute levels for any node as the level of each node $level(t_i) = ff_i - post(t_i) - 1$, because the cardinality of the descendant node set can be computed as: $size(t_i) = ff_i - i - 1$. For the tree from Fig. 1, the extended signature is: $sig(T) = \langle a, 10, 11, 0; b, 5, 7, 1; c, 3, 6, 2; d, 1, 5, 3; e, 2, 6, 3; g, 4, 7, 2; f, 9, 11, 1; h, 8, 11, 7; o, 6, 10, 8; p, 7, 11, 8 \rangle$.

A *sub-signature* $sub\_sig_S(T)$ is a specialized (restricted) view of T through signatures, which retains the original hierarchical relationships of elements in T. Specifically, $sub\_sig_S(T) = \langle t_{s_1}, post(t_{s_1}); t_{s_2}, post(t_{s_2}); \ldots t_{s_n}, post(t_{s_n}) \rangle$ is a sub-sequence of $sig(T)$, defined by the ordered set $S = \{s_1, s_2, \ldots s_k\}$ of indexes (preorder values) in $sig(T)$, such that $1 \leq s_1 < s_2 < \ldots < s_k \leq n$.

Finally, the serialization of the tree based on pre- and post-orders adopted for the tree signature allows the introduction of a direct and simple approach for the evaluation of the *ordered tree inclusion* of a query twig Q in a data D. We recall that Q is included in D, if D contains all nodes of Q and when the *sibling* and *ancestor* relationships of the nodes in D are the same as in Q. As show in [ZAD03], using the two signatures $sig(Q)$ and $sig(D)$, we can state that D is included in an ordered fashion in D if $sig(Q)$ is sequence included at the level of node names in $sig(D)$ and if such an inclusion satisfies some constrains on the pre- and post-order values of the involved nodes.

3 Unordered Tree Pattern Matching

In this section, we propose an approach to the unordered tree pattern matching by using tree signatures. The following definition specifies the notion of unordered tree inclusion.

Definition 2 (Unordered Tree Inclusion). *Given a query twig pattern Q and an XML tree D, an unordered tree inclusion of Q in D is identified by a total mapping from nodes in Q to some nodes in D, such that only the ancestor-descendent structural relationships between nodes in Q are satisfied by the corresponding nodes in D.*

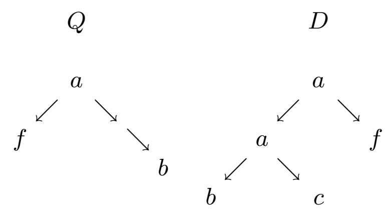


Fig. 2. Sample of unordered tree inclusion

Example 1. Consider the query Q and the data tree D of Fig. 2 where the double arrow represents an ancestor-descendant edge. Then the only unordered inclusion of Q in D is identified by the following total mapping $\{1 \mapsto 1, 2 \mapsto 5, 3 \mapsto 3\}$ where nodes are univocally identified by their pre-order value and ancestor-descendent structural relationships are satisfied (e.g. in Q 1 is the parent of 2 and in D 1 is the parent of 5).

Since signatures assume (data and query) trees always ordered, the serialization of trees based on the preorder and postorder ranks does not only capture the ancestor-descendent but also the sibling relationships. For this reason, unlike ordered tree inclusion evaluation, the unordered tree inclusion can not be evaluated by directly using extended string processing algorithms.

Lemma 1. *Suppose the data tree D and the query tree Q to be specified by signatures $sig(D) = \langle d_1, post(d_1); d_2, post(d_2); \ldots d_m, post(d_m)\rangle$, $sig(Q) = \langle q_1, post(q_1); q_2, post(q_2); \ldots q_n, post(q_n)\rangle$. The unordered query tree Q is included in the data tree D if the following two conditions are satisfied: (1) on the level of node names, an ordered set of indexes $S = \{s_1, s_2, \ldots s_n\}$ exists, $1 \leq s_i \leq n$ for $i = 1, \ldots, n$, such that $d_{s_i} = q_i$, (2) for all pairs of entries i and j, $i, j = 1, 2, \ldots |Q|-1$ and $i+j \leq |Q|$, if $post(q_{i+j}) < post(q_i)$ then $post(d_{s_{i+j}}) < post(d_{s_i})$.*

For instance consider the signatures of query Q and the signature of the data tree D of Fig. 2: $sig(Q) = \langle a, 3; f, 1; b, 2\rangle$ $sig(D) = \langle a, 5; a, 3; b, 1; c, 2; f, 4\rangle$. The only sub-signature qualifying the unordered tree inclusion of Q in D is defined by the index set $\{1, 5, 3\}$ and the corresponding sub-signature is $sub\_sig_{\{1,3,5\}}(D) = \langle a, 5; b, 1; f, 4\rangle$.

The solution we propose basically employs tree signatures to represent data trees. Then we decompose the query tree into a set of root-to-leaf paths and, by exploiting string processing algorithms, we evaluate the ordered inclusion of such multiple queries. Finally, if there are structurally consistent answers to the ordered inclusion of all the paths, the unordered tree inclusion of the query in the data tree is found.

In the following, due to the lack of space, we give a brief overview of each of the three steps which our approach is composed of. Details about the adopted algorithms and their properties can be found in [ZMM04].

3.1 Query decomposition

The query decomposition process transforms a query twig Q into a set of root-to-leaf paths so that the ordered tree inclusion can be safely applied. For efficiency reasons, we sort the paths on the basis of their selectivity, so that in the next phase, the more selective paths are evaluated before the less selective ones. By acting on the extended signature $sig(Q) = \langle t_1, post(t_1), ff_1, fa_1; \ldots; t_n, post(t_n), ff_n, fa_n\rangle$ of Q, the output of the query decomposition process is the ordered set $rew(Q)$ of the sub-signatures $sub\_sig_{P_j}(Q)$ defined by the index sets P_j, for each leaf j of Q. For instance for the query twig of Ex. 1, $rew(Q) = \{P_2, P_3\}$ where $P_2 = \{1, 2\}$ and $P_3 = \{1, 3\}$.

3.2 Path inclusion evaluation

Any path P_j represents all (and only) the ancestor-descendent relationships between the involved nodes and the ordered evaluation coincides with the unordered one. Thus, an ordered inclusion of $sig(P_j)$ in $sig(D)$ states that a mapping, keeping the ancestor-descendent relationships, exists from the nodes in P_i to some nodes in D. It can be performed by means of an extended string processing algorithm as shown by the following Lemma.

Lemma 2. *A path $P \in rew(Q)$ is included in the data tree D, in the sense of Definition 2, if the following two conditions are satisfied: (1) on the level of node names, $sub\_sig_P(Q)$ is sequence-included in $sig(D)$ determining $sub\_sig_S(D)$ through the ordered set of indexes $S = \{s_1, \ldots, s_h\}$, (2) for each $i \in [1, n-1]$: $post(d_{s_i}) < post(d_{s_{i+1}})$.*

For each path query $P \in rew(Q)$, we are thus able to efficiently compute the answer set $ans_P(D) = \{S \mid sub\_sig_S(D)$ qualifies the inclusion of P in $D\}$. For instance, $ans_{P_2}(D)$ and $ans_{P_3}(D)$ of Ex. 1 is shown in the first two Tables of Fig. 3.

3.3 Identification of the answer set

The answer set $ans_Q(D)$ of the unordered inclusion of Q in D can be determined by joining compatible answer sets $ans_P(D)$, for all $P \in rew(Q)$. The main problem is to establish how to join the answers for the paths in $rew(Q)$ to get the answers of the unordered inclusion of Q in D. Not all pairs of answers of two distinct sets are necessarily "joinable". The condition is that any pair of paths P_i and P_j share a common sub-path (at least the root) and differ in the other nodes (at least the leaves). Such *commonalities* and *differences* must meet a correspondence in any pair of index sets $S_i \in ans_{P_i}(D)$ and $S_j \in ans_{P_j}(D)$, respectively, in

$ans_{P_2}(D)$:

$\mathbf{P_2}$	**1**	**2**
	1	5

$ans_{P_3}(D)$:

$\mathbf{P_3}$	**1**	**3**
	1	3
	2	3

$sj(ans_{P_2}, ans_{P_3})$:

$\mathbf{P_2 \cup P_3}$	**1**	**2**	**3**
	1	5	3

Fig. 3. Structural join of Ex.1

order that they are joinable. In this case, we state that $S_i \in ans_{P_i}(D)$ and $S_j \in ans_{P_j}(D)$ are structurally consistent as specified by the definition. For instance $\{1,5\} \in ans_{P_2}(D)$ and $\{2,3\} \in ans_{P_3}(D)$ are not structurally consistent as the common node with pre-order value 1 in Q does not find a correspondence in the two index sets (i.e. $1 \neq 2$ as shown in the data tree D in Fig. 2, as they correspond to the two nodes with the same label a).

Definition 3 (Structural consistency). *Let Q be a query twig, D a data tree, $T_i = \{t_i^1, \ldots, t_i^n\}$ and $T_j = \{t_j^1, \ldots, t_j^m\}$ two ordered sets of indexes determining $sub\_sig_{T_i}(Q)$ and $sub\_sig_{T_j}(Q)$, respectively, $ans_{T_i}(D)$ and $ans_{T_j}(D)$ the answers of the unordered inclusion of T_i and T_j in D, respectively. $S_i = \{s_i^1, \ldots, s_i^n\} \in ans_{T_i}(D)$ and $S_j = \{s_j^1, \ldots, s_j^m\} \in ans_{T_j}(D)$ are structurally consistent if:*

- *for each pair of common indexes $t_i^h = t_j^k$, $s_i^h = s_j^k$;*
- *for each pair of different indexes $t_i^h \neq t_j^k$, $s_i^h \neq s_j^k$.*

Given two structurally consistent answers $S_i \in ans_{T_i}(D)$ and $S_j \in ans_{T_j}(D)$, where $T_i = \{t_i^1, \ldots, t_i^n\}$, $T_j = \{t_j^1, \ldots, t_j^m\}$, $S_i = \{s_i^1, \ldots, s_i^n\}$ and $S_j = \{s_j^1, \ldots, s_j^m\}$, the join of S_i and S_j, $S_i \bowtie S_j$, is defined on the ordered set $T_i \cup T_j = \{t^1, \ldots, t^k\}$ as the index set $\{s^1, \ldots, s^k\}$ where:

- for each $h = 1, \ldots, n$, $l \in \{1, \ldots, k\}$ exists such that $t_i^h = t^l$ and $s_i^h = s^l$;
- for each $h = 1, \ldots, m$, $l \in \{1, \ldots, k\}$ exists such that $t_j^h = t^l$ and $s_j^h = s^l$.

The answer set $ans_Q(D)$ can thus be computed by sequentially joining the sets of answers of the evaluation of the path queries. We denote such operation as the *structural join*.

Definition 4 (Structural join). *Let Q be a query twig, D a data tree, T_i and T_j two ordered sets of indexes determining $sub\_sig_{T_i}(Q)$ and $sub\_sig_{T_j}(Q)$, respectively.*

The structural join $sj(ans_{T_i}(D), ans_{T_j}(D))$ between the two sets $ans_{T_i}(D)$ and $ans_{T_j}(D)$ is the set $ans_T(D)$ where $T = \{t^1, \ldots, t^k\}$ is the ordered set obtained by the union $T_i \cup T_j$ of the ordered sets T_i and T_j and $ans_T(D)$ contains the join $S_i \bowtie S_j$ of each pair of structurally consistent answers $(S_i \in ans_{T_i}(D), S_j \in ans_{T_j}(D))$.

The structural join $sj(ans_{T_i}(D), ans_{T_j}(D))$ thus returns an answer set defined on the union of two sub-queries T_i and T_j as the join of the structurally consistent answers of $ans_{T_i}(D)$ and $ans_{T_j}(D)$. Starting from the set of answers $\{ans_{P_{x_1}}(D), \ldots, ans_{P_{x_k}}(D)\}$ for paths in $rew(Q)$, we get the answer set $ans_Q(D)$ identifying the unordered inclusion of Q in D by incrementally merging the answer sets by means of the structural join. Since the structural join operator is associative and symmetric, we can compute $ans_Q(D)$ as:

$$ans_Q(D) = sj(ans_{P_{x_1}}(D), \ldots, ans_{P_{x_k}}(D)) \tag{1}$$

where $rew(Q) = \{P_{x_1}, \ldots, P_{x_k}\}$.

Example 2. The answer set $ans_Q(D)$ of Example 1 is the outcome of the structural join $sj(ans_{P_2}(D), ans_{P_3}(D)) = ans_{P_2 \cup P_3}(D)$ where $P_2 \cup P_3 = \{1,2\} \cup \{1,3\}$ is the ordered set $\{1,2,3\}$. The answers to the individual paths and the final answers are shown in Fig. 3 (the first line of each table shows the query). It joins the only pair of structurally consistent answers: $\{1,5\} \in ans_{P_2}(D)$ and $\{1,3\} \in ans_{P_3}(D)$.

3.4 Efficient computation of the answer set

Till now, we have studied how tree signatures can be employed to support unordered tree pattern matching. However, XML data trees can have many nodes and the tree signatures, linearly proportional to the number of nodes, can be very large, so the performance aspects of such operation becomes a matter of concern. In the previous sections, we have specified two distinct phases for unordered tree pattern matching: the computation of the answer set for each root-to-leaf path of the query and the structural join of such sets. The main drawback of this approach is that many intermediate results may not be part of any final answer. As shown in [ZMM04], these two phases can be merged into one phase in order to avoid unnecessary computations.

In the following, we will give some hints of the adopted algorithm. The basic idea is to evaluate at each step the most selective path among the available ones and to directly combine the partial results computed up to that moment with structurally consistent answers of the paths. More precisely, assuming that pQ is the ordered index set determining the sub-signature $sub\_sig_{pQ}(Q)$ evaluated up to that moment, the algorithm incrementally joins the partial answer set $ans_{pQ}(D)$ with the answer set $ans_P(D)$ of the next path P of $rew(Q)$. As each pair of index sets must be structurally consistent in order to be joinable, we compute only such answers in $ans_P(Q)$, which are structurally consistent with some of the answers in $ans_{pQ}(D)$. As a matter of fact, only such answers may be part of the answers to Q. In order to do it, the algorithm exploits the additional information contained in the extended signature $sig(Q)$ and tries to extend each answer in $ans_{pQ}(D)$ to the answers to $pQ \cup P$ by only evaluating such sub-path of P which has not been evaluated in pQ.

In summary, the proposed solution performs a small number of additional operations on the paths of the query twig Q, but dramatically reduces the number of operations on the data trees by avoiding the computation of useless path answers. In this way, we remarkably reduce computing efforts. Indeed, while query twigs are usually very small and have a limited number of paths, XML data trees can have many nodes and tree signatures can be very large.

4 Performance evaluation

In this section we evaluate the performance of our unordered tree inclusion technique. We measure the time needed to process different query twigs using the paths decomposition approach, described in this paper, and compare the obtained results with the query processing performance of the permutation approach.

Since synthetic data sets are not significant enough to show the performance of real-life XML query scenarios, we performed our experiments on a real data set, specifically the complete DBLP Computer Science Bibliography archive as of November 2003. Notice that the file consists of over 3.8 Millions of elements, where over 3.4 Millions of them have an associated value. The size of the XML file is 156MB. All algorithms are implemented in Java JDK 1.4.2 and the experiments are executed on a Pentium 4 2.5Ghz Windows XP Professional workstation, equipped with 512MB RAM and a RAID0 cluster of 2 80GB EIDE disks with NT file system (NTFS).

We tested the performance of our approach for queries derived from six query twig templates (see Fig. 4). Such templates present different *element name selectivity*, i.e. the number of elements having a given element name, different *branching factors*, i.e. the maximum number of sibling elements, and different *tree heights*. We refer to the templates as "*xSb-h*', where S stands for element name selectivity and can be H(igh) or L(ow), b is the branching factor, and h the tree height. We used `inproceedings` for the low selectivity and `book` and `phdthesis`

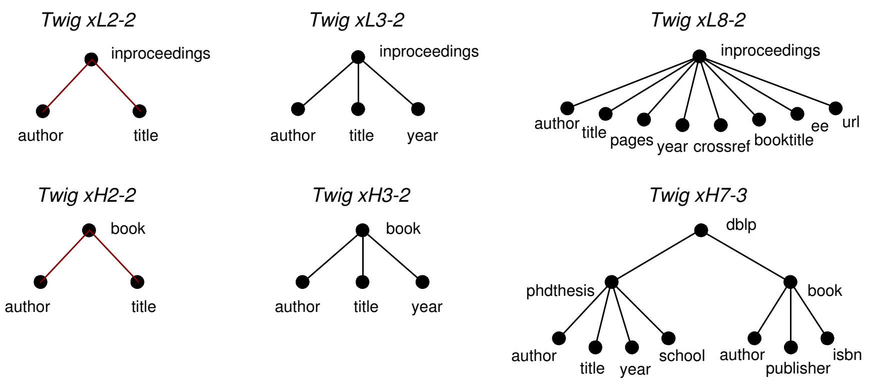


Fig. 4. The query twigs templates used in the performance tests

for the high selectivity. We conducted experiments by using not only queries defined by the plain templates (designated as "*NSb-h*"), which only contain tree structural relationships, but also queries (designated as "*VSb-h*"), where the templates are extended by predicates on the author name. Value accesses are supported by a content index. We chose the highly content-selective predicates, because we believe that this kind of queries is especially significant for highly selective fields, such as the author name. On the other hand, the performance of queries with low selectivity fields should be very close to the corresponding templates. In this way, we measured the response time of twelve queries, half of which contain predicates.

Query			**Evaluation**				
Twig	**# elements**	**# predicates**	**# solutions**	**Decomposition (sec)**	**Permutation**		
					N	**mean (sec)**	**total (sec)**
NH2-2	3	0	1343	*0.016*	2	0.014	*0.028*
NH3-2	4	0	1343	*0.016*	6	0.015	*0.105*
NH7-3	10	0	90720	*1.1*	288	0.9	*259.2*
NL2-2	3	0	559209	*2.2*	2	2.28	*4.56*
NL3-2	4	0	559209	*4.2*	6	2.49	*14.94*
NL8-2	9	0	149700	*7.7*	40320	4.8	*193536*
VH2-2	3	1	1	*0.015*	2	0.014	*0.028*
VH3-2	4	1	1	*0.016*	6	0.016	*0.096*
VH7-3	10	2	1	*0.031*	288	0.03	*8.64*
VL2-2	3	1	39	*0.65*	2	0.832	*1.664*
VL3-2	4	1	36	*0.69*	6	1.1	*6.6*
VL8-2	9	1	29	*0.718*	40320	2.3	*92736*

Table 1. Performance comparison between the two unordered tree inclusion alternatives

Table 1 summarizes the results of the unordered tree inclusion performance tests for both the approaches we considered. For each query twig, the total number of elements and predicates, the number of solutions (inclusions) found in the data set, and the processing time, expressed in seconds, are reported. For the permutation approach, the number of needed permutations and the mean per-permutation processing time are also presented. It is evident that the decomposition approach is superior and scores a lower time in every respect. In particular, with low branching factors (i.e. 2), such approach is twice as faster for both selectivity settings. With high branching factors (i.e. 3, 8) the speed increment becomes larger and larger – the number of permutations required in the alternative approach grows factorially: for queries *NL8-2* and *VL8-2* the decomposition method is more than 25000 times faster. The decomposition approach is particularly fast with the high selectivity queries. Even for greater heights (i.e. in *VH7-3*), the processing time stays in order of milliseconds. For the decomposition method, as we do not have statistics on the path selectivity at our disposal, we measured the time needed to solve each query for each of the possible order of path evaluation and reported only the lower one. As we expected, we found that starting with the most highly selective paths always increases the query evaluation efficiency. Of course, for the predicate queries the best time is obtained by starting the evaluation from the value-enabled paths.

Finally, notice that the permutation approach also requires an initial "startup" phase where all the different permutation twigs are generated; the time used to generate such permutations is not taken into account.

5 Conclusions

In this paper, we presented a summary of the study on the problem of efficient evaluation of unordered query trees in XML tree structured data collections we presented more deeply in [ZMM04]. As the underlying concept, we have used the tree signatures, which have proved to be a useful structure for an efficient tree navigation and ordered tree matching, see [ZAD03]. We have identified two evaluation strategies, where the first strategy is based on multiple evaluation of all query tree structure permutations and the second on decomposing a query tree into a collection of all root-to leaf paths.

We have studied the decomposition approach and established rules for decomposition as well as strategies for integration of partial, structurally consistent, results through structural joins. The permutation and decomposition approaches to the unordered tree matching have been tested on the DBLP data set for various types of queries. The experiments demonstrate a clear superiority of the decomposition approach, which is especially advantageous for the large query trees, and for trees with highly selective predicates.

References

[Die82] P.F. Dietz. Maintaining Order in a Linked List, In *Proceedings of STOC, 14th Annual ACM Symposium on Theory of Computing*, May 1982, San Francisco, CA, 1982, pp. 122-127.

[Gr02] T. Grust. Accelerating XPath location steps. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, Madison, Wisconsin, 2002, pp. 109-120.

[SWS+02] D. Shasha, J.T.L. Wang, H. Shan, and K. Zhang. ATreeGrep: Approximate Searching in Unordered Trees. Proceedings of the *14th International Conference on Scientific and Statistical Database Management*, July 24-26, 2002, Edinburgh, Scotland, UK, pp. 89-98.

[ZAD03] P. Zezula, G. Amato, F. Debole, and F. Rabitti. Tree Signatures for XML Querying and Navigation. In *Proceedings of the XML Database Symposium, XSym 2003*, Berlin, September 2003, LNCS 2824, Springer, pp. 149-163.

[ZSS92] K. Zhang, R. Statman, and D. Shasha. On the edit distance between unordered labeled trees. *Information Processing Letters*, 42:133-139, 1992.

[ZMM04] P. Zezula, F. Mandreoli, and R. Martoglia. Tree Signatures and Unordered XML Pattern Matching (invited talk). In *Proceedings of 30th Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2004*, Merin, January 2004.