Adaptive Search Algorithm for Patterns, in Succinctly Encoded XML

Jérémy Barbay

David R. Cheriton School of Computer Science
University of Waterloo, Canada.

Technical Report CS-2006-11
$Date: 2006/04/21 22:33:36 , Revision: 1.18 $

Abstract. We propose an adaptive algorithm for context queries (queries expressed as preorder and ancestor-descendant relations on labeled nodes), which can be used to find patterns in XML documents. Our algorithm takes advantage of the correlation between terms of the query without any preprocessed information, and it runs in time (kd(lg lg min(n,s)+lg lg(r))) in the RAM model, where k is the number of terms in the query, d is the non-deterministic complexity of the query on the multi-labeled tree (i.e. the minimum number of operations required to check the answer to the query), n is the number of nodes in the tree, s is the number of relations between nodes and labels, and r is the maximal number of nodes matching a label on any rooted path in the tree.

Keywords: XML, Multi-Labeled Tree, Context Query, Adaptive Algorithm

1 Introduction

XML is a rapidly emerging format for exchanging data on the web. It standardizes tree structures, so that general tools can be developed and used for the many distinct applications adopting this standard. Among those tools, search engines are prominent, and are strongly based on path navigation. XPath [4, 8] is a language developed for this purpose, which specifies nodes in an XML document by patterns on their rooted paths. Each pattern on paths is incrementally described by a sequence of *location steps*, composed of an *axis*, a *node test*, and optionally some *predicates*. The XPath 2.0 [4] specifications define 13 axes and many expressions for predicates, but in many applications and studies only a small subset of them is considered. In this paper we consider XPath queries using the four axes `ancestor`, `descendant`, `following` and `preceding`, as defined by the XPath 2.0 [4] specifications, with single type node tests and no predicates.

Many solutions for pattern-matching in XML documents have been studied: we cite here only a few representative ones. The native XML encoding being not very efficient (both in term of the space used and in terms of searchability), query solvers consider either an external index added to the document, or simply a different encoding of the XML document. Some of the solutions use statistical information about the two by two correlation of terms in the query to allow the algorithm solving the query to choose in which order to consider the terms to reduce the size used for intermediary results. Our algorithm does not require any statistical information, and it uses only a constant amount of space in addition to the space required to save the results: as Bruno *et al.*'s [5] algorithm, our algorithm is holistic, it considers the query as a whole, and it avoids unnecessarily large intermediate results.

All our results concerning the running time of operators and algorithms are expressed in the RAM model, where words of size $\mathcal{O}(\lg(\max\{n,\sigma\}))$ can be accessed and processed in constant time. Our work is based on a succinct encoding for multi-labeled trees from Barbay and Rao [14], which uses $t(\lg \mu\rho + o(\lg \mu\rho))$, which is $t(\lg \rho + o(\lg \rho))$ bits more than the encoding from Barbay *et al.*, where μ is a short notation for $\min(n,\sigma)$, and where ρ is the average number of nodes having the same label on a branch (it has been observed to be very small in practice by Zhang *et al.* [17]). It supports the labeled search for the first α-ancestor of node a after node b in time $\mathcal{O}(\lg\lg\mu + \lg\lg\rho_\alpha)$; and for the first α-descendant or children α-children in time $\mathcal{O}(\lg\lg\mu)$ [14, Corollary 4]

We define the concept of *context queries*, defined by a graph in which nodes are associated to labels and edges are associated to relatioons (`ancestor`, `descendant`, `following` or `preceding`) , and we propose an adaptive

algorithm to solve these queries in $\mathcal{O}(\delta k)$ operator calls and in time $\mathcal{O}(\delta k \lg\lg n)$ in the RAM model, where k is the number of terms of the query and δ is the non-deterministic complexity of the query on the multi-labeled tree (e.g. the minimum number of operations required to check the answer of the query). This algorithm can be used to solve a large sub-class XPath queries (as a particular case of context queries) and it takes advantage of the correlation in the document between the sets of nodes matching each terms of the XPath query, *without any preprocessed information about this correlation*.

The paper is organized as follows: In Section 2 we define context queries and describe our adaptive algorithm to solve those queries using the operators defined in the previous section. In Section 3 we discuss how our work compares to related studies of XML queries and adaptive algorithms. We conclude in Section 4 with a discussion of our results and a perspective on future research.

2 Adaptive Algorithm

The encoding defined in [14, Corollary 4] supports fast operators to find a single node b matching some criteria on its label and on its location relative to some node a. We now show how to use those operators to find the matchings of a whole pattern in a multi-labeled tree.

2.1 Context Queries

We consider queries slightly more general than those defined by Bruno *et al.* [5]. Mainly, we introduce the axis `following` and `preceding`, and allow cyclic condition in the definition of the query.

Definition 1. *A* **context query** *is a directed graph such that*

- *one node, called the* `target`*, is distinguished;*
- *each node is associated to a label from* $[\sigma] \cup \{*\}$*;*
- *each directed edge is associated to one of the following XPath axis [4:]* `ancestor`, `descendant`, $\texttt{following}_{\{pre\}}$, *and* $\texttt{preceding}_{\{pre\}}$.

A node **matches a context query** Q *if it matches the* `target` *node in a context matching the rest of* Q*'s graph.*

Figure 1 shows a context query searching in a medical file for all patients to whom have been administrated a test of result Y after having been prescribed a medicine X. The query specifies the list of nodes labeled `patient`, in whose subtree a node labeled `prescription` with a descendant labeled `medicine X` precedes a node labeled `test` with a descendant labeled `result Y`.

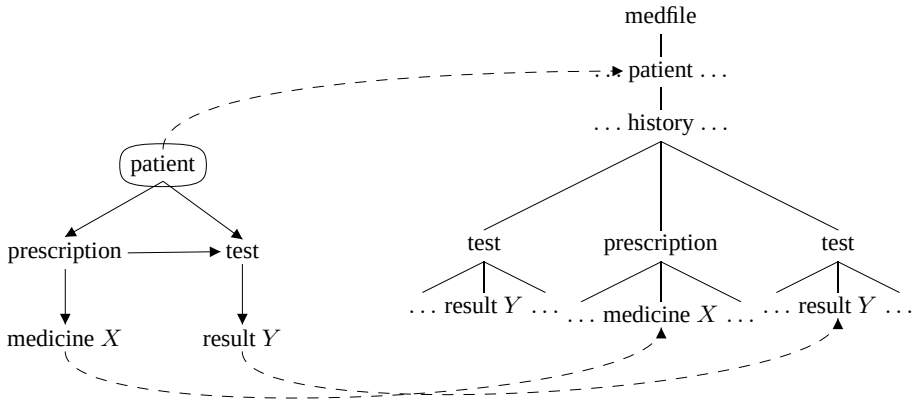

Fig. 1. A simple query, on a subset of a medical database. Vertical and diagonal edges correspond to `descendant` axes, and horizontal edges to $\texttt{following}_{pre}$ axes. The target node, here the `patient` node, is circled.

2.2 Correlation and tour of the query

The answer to a context query is never larger than the number of nodes labeled as the `target` node of the query. Algorithms inspired by joint operations in relational databases restrict this set of nodes by considering the edges of the query one by one. Such an approach is inefficient in many instances, in particular when the lists of labels associated to two nodes of the query are similar, i.e. when the two nodes are very correlated (e.g. when all nodes "`medicine` X" descend from "`prescription`" nodes).

Such instances are not unlikely in a database such as described in Figure 1, where each patient would probably be associated to many prescriptions and tests, and comparing the lists of `patient`, `prescription` and `test` nodes would not result in a significant reduction of the result set. On the other hand, even though the terms "medicine X" and "result Y" are more specific, and are more likely to help reduce more significantly the set of nodes potentially matching the query, comparing the lists of `medicine` X and `prescription` nodes, or equivalently the list of `test` and `result` Y, will not result in a significant reduction of the result set.

One way to avoid wasting time in comparing highly similar lists of nodes is to precompute the average correlation for each pair of labels. This permits us to choose in priority the pair of query nodes that will reduce the most the set of potential nodes matching the query. This approach is very common in practical implementations of relational and semi-structured databases, such as BLAS [6], but it uses $\mathcal{O}(\sigma^2)$ space, and it still does not take advantage of local correlations. For instance, suppose that the children of each history node in the database of Figure 1 are ordered chronologically, and that many prescriptions referring to medicine X occur before 2000, while most tests referring to result Y occur after this date. An algorithm considering one edge at a time might have large intermediate results independent of its choice between the `medicine` or `result` subtree of the query.

To take advantage of the local correlations between terms, any algorithm has to cycle constantly through the edges of the query, while considering the query as a while at each step, a characteristic named "holistic" by Bruno *et al.* [5]. We define here an order to cycle through the query in at most $2k$ steps.

Definition 2. *A* **tour** *of the query Q is a cyclic path of minimal length following edges without any direction constraint and visiting each edge (and as a consequence visiting each node) of the query at least once.*

The direction of the edges of the query is necessary to make sense out of the relation associated to each edge, but as the location axes are symmetric (e.g. `parent` v.s. `child`, `following` v.s. `preceding`) one can check whether two nodes match an edge in any direction. Hence the tour ignores the direction constraints. Figure 2 gives an example of such a tour on the query of Figure 1.

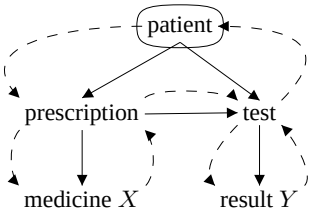

Fig. 2. A tour of the query given Figure 1.

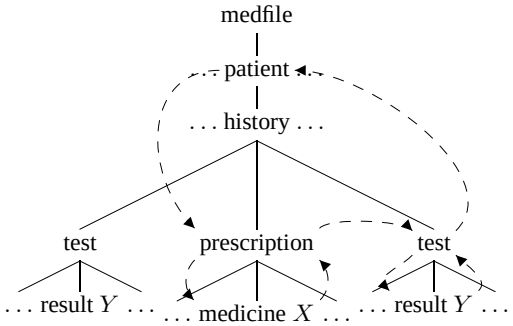

Fig. 3. An execution trace.

It is trivial to see that such a tour is of length at least k, the number of edges in the query, and at most $2k$. The complexity of following such a tour depends of the query but is independent of the size and content of the database. Any algorithm following such a tour has found a match of the query each time the $2k$ last consecutive query-edges traversed and checked matched (see Figure 3).

2.3 Algorithm

We propose an algorithm that does not require any statistical information and that avoids unnecessarily large intermediate results. As Bruno *et al.*'s [5] holistic algorithm, it considers the query as a whole rather than decomposing it in independent joints. Following a tour such as defined in Section 2.2, and searching the tree through the operators on the succinct encoding, it is possible to find all the nodes matching a context query in time proportional to the number of steps that a non-deterministic algorithm would require to *check* the correctness of the answer to the query.

Theorem 1. *Consider a context query Q of k edges on a multi-labeled tree on n nodes and σ labels in t relations. There is an algorithm solving Q in $\mathcal{O}(\delta k)$ operator calls and in time $\mathcal{O}(\delta k \lg\lg n)$, where δ is the minimum number of steps required to check the answer of the query.*

Proof (sketch). For simplicity, consider the bogus node ∞ that matches all labels and is a successor to all nodes. Our algorithm goes as follows: starting from the `target` node, it cycles through the edges of the query Q,

Algorithm 1 `Solve_Query`(Q)

Given a context query Q of target node `target`, the function outputs the list of nodes matching Q.

```
α ← target(Q); YES ← 0;
a ← the first available α-node of the document.
while a ≠ ∞ do
  (α, r, β) ← the next edge of tour(Q);
  b ← next_match(α, a, r, β, b)
  if match(a, r, b) then
    YES ← YES + 1;
    if YES = |tour(Q)| then
      Output the node a currently matching the target (of label α);
      a ← next(α, a);
    end if
  else
    YES ← 0;
  end if
  (α, a) ← (β, b);
end while
```

updating for each query node α the matching node a in the multi-labeled tree, such that any α-node preceding a in preorder has already been considered and can be ignored.

After a tour where $|\texttt{tour}(Q)|$ matches have been successfully checked, the algorithm has found a match for Q and can output the tree-node corresponding to the `target` node. To allow the search to continue, it then updates which tree-node is associated to the target, and iterates. The preorder rank of the successive nodes pointed to for each query-node is strictly increasing at each update, so that at any time, all preorder predecessors of the tree-node matching `target` have been considered. Every $|\texttt{tour}(Q)| \leq 2k$ iterations the algorithm considered at least as many nodes as a non-deterministic algorithm would have in a single operation. It takes at most $2k$ steps to eliminate as many potential result nodes as a non-deterministic algorithm which can "guess" which operation to perform to eliminate the largest number of potential result nodes.

When the preorder rank of the node associated to any query-node reaches the value ∞, all nodes matching this label have been considered (hence the correctness), and the algorithm has performed $2\delta k$ operator calls where a non-deterministic algorithm would have performed at least δ (hence the complexity result). □

Consider for instance the execution trace presented in Figure 4 and 5, for a tour of the query considering `test` nodes before `prescription` nodes. A non-deterministic algorithm can "guess" the tour described in Figure 2 to find or check the match immediately, as in Figure 3. But even when the algorithm chooses a non-optimal tour, it will find the match (or, equivalently, the key operations to prove that some nodes cannot belong to a match) in at most twice as many operations as the best non-deterministic algorithm, i.e. at most $4k$.

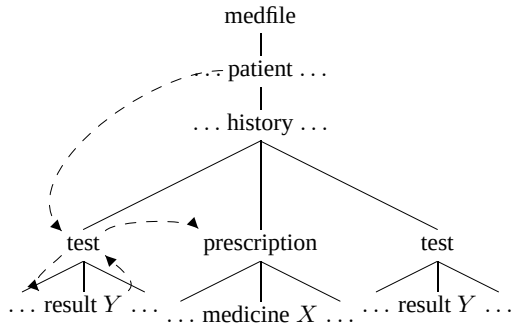

Fig. 4. The first part of the execution trace of the algorithm with another tour of the query given Figure 1, symmetric to the one given in Figure 2: the trace stops when the algorithm proves that the first `node` does not belong to a match, because no `prescription` node precedes it.

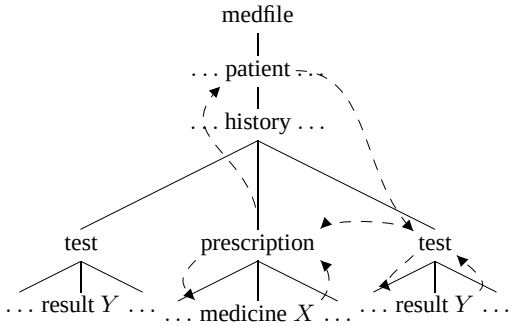

Fig. 5. The rest of the execution trace, where the algorithm follows the tour through the nodes labeled `prescription`, `medicine`, `prescription`, `patient`, `test`, `result`, `test`, and `prescription`, at which point it proved a match, having matched seven consecutive edges, the length of the tour of the query.

3 Comparison to Related Work

3.1 XML and XPath

Since data sets can be very large and can be searched very often, efficiently querying XML data is a major concern. Indexes and intelligent labeling can be used to simplify the operations in a more or less compact way [1, 2]. Grust [12] proposes a labeling scheme based on the prefix and postfix rankings of the nodes and observed that for any node v this scheme permitted to easily partition the nodes of the document in four parts, from which the set of nodes accessed from v by any axis is easily defined.

As Grust implements and queries his index using purely relational techniques, his approach is close to similar ones storing XML in relational databases. Shanmugasundaram *et al.* [16] proposed storing XML in relational databases, in order to benefit from the extensive optimizations developed for those database systems.

Bruno *et al.* [5], working on a variant of XPath called Twig Pattern Matching, suggests that "*a limitation of this approach (...) is that intermediate result sizes can get large, even when the input and output sizes are more manageable*", and proposes instead to study holistic algorithms that consider the query as a whole and avoid unnecessarily large intermediate results. This approach supposes a native implementation [13] to treat XML queries, as opposed to the storage of XML in a relational database.

We propose an holistic algorithm to solve a subset of XPath queries. As Bruno *et al.*'s algorithm for twig pattern matching, it considers the query as a whole, and it avoids unnecessarily large intermediate results as it uses only a constant amount of space in addition to the space required to save the results. The first advantage of our algorithm is that its complexity is expressed as a function of the non-deterministic complexity of the instance, and that it takes advantage of the local correlation between terms of the query. The second advantage comes from the use of the succinct encoding to perform much faster searches in the tree. One direction of research is to study how Bruno *et al.*'s [5] algorithm can be adapted to use our succinct encoding, and to determine if it can be analyzed in a way similar to our algorithm.

3.2 Adaptive Algorithms

Adaptive algorithms are algorithms that, among instances of the same size, perform better on "easier" instances, where the easiness has to be defined for each problem. Kirkpatrick and Seidel [15] were the first to point out the interest of such algorithms by giving an algorithm to compute the convex-hull of a set of points whose complexity is expressed as a function of the size of the convex-hull rather than of the size of the set. Adaptive algorithms for sorting were studied as functions of many distinct measures of difficulty, and Estivill and Castro summarized these in a survey [11].

Closer to our applications, Demaine *et al.* [9], motivated by applications in posting lists for search engines, studied adaptive algorithms for the union, intersection and difference of sets represented by sorted arrays. Their measure of difficulty is defined as the cost of encoding a certificate of the result of the instance, and they proved that their algorithms are optimal in the comparison model with respect to this measure of difficulty. Barbay and Kenyon [3] defined another measure of difficulty (denoted δ) for the intersection problem, based on the number of steps required by a non-deterministic algorithm to check the answer. They proved that their deterministic algorithm, with respect to this measure of difficulty, was optimal as compared to randomized algorithms in the comparison model.

Our algorithm and its analysis are directly inspired by the work on the Intersection problem from Barbay and Kenyon [3]. While context queries are not exactly reduced to intersection instances, the relation between both problems is very strong, especially when abstracting the algorithm in the *search* model, where the basic operation is a search (as opposed to the comparison model where the basic operation is a comparison). This suggests that other intersection algorithms such as the ones defined by Demaine *et al.* [9] could also be transcribed to this context. Of particular interest, the algorithm `Small Adaptive` has been shown to outperform other algorithms [10] on the intersection problem, and in particular to outperform an algorithm which considers the sets two by two, in a way similar to the XPath algorithms currently used in XPath query solvers such as BLAS [6]. The practical study of the adaptation of this algorithm to XPath queries sounds promising.

4 Conclusion

We consider in this paper the applications of succinct encoding techniques for data structures, and adaptive algorithm techniques for the resolution of queries. We define the concept of *context queries*, and we propose an adaptive algorithm to solve these queries in $\mathcal{O}(\delta k)$ operator calls and in time $\mathcal{O}(\delta k \lg \lg n)$ in the RAM model, where k is the number of terms of the query and δ is the non-deterministic complexity of the query on the multi-labeled tree (e.g. the minimum number of operations required to check the answer of the query). This algorithm is holistic as it takes into account the whole query rather than decomposing it in smaller chunks solved independently, and it takes advantage of the local correlation between labels in the document, *without any preprocessed information about this correlation*.

One obvious direction for future research is to extend our technique to support a broader range of XPath queries. It seems that complex node tests cannot be supported by the data structure without increasing its size by more than a lower order term. The work from Chiniforooshan *et al.* [7] is a first step towards an algorithm to support more complex node type tests.

An interesting open problem in theory is whether there is a lower bound on the complexity of any algorithm solving context queries. Using a technique similar to the one used by Barbay and Kenyon [3], it would not be difficult to prove a lower bound on the number of operator calls performed by any randomized algorithm solving context queries, but it seems much more difficult to obtain a more general lower bound, that does not suppose that the algorithm uses our encoding.

Another interesting perspective for future work concerns the implementation of our data structures and algorithms. Succinct encodings is still a young research topic, and the structures described are even more complicated to implement than to describe. As the field matures, one can hope that simpler structures will appear that will be easier to implement but with properties almost as good as those presented in this paper.

Bibliography

[1] S. Abiteboul, H. Kaplan, and T. Milo. Compact labeling schemes for ancestor queries. In *Proceedings of the Twelfth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 547–556, 2001.

[2] S. Alstrup and T. Rauhe. Improved labeling scheme for ancestor queries. In *Proceedings of the Thirteenth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 947–953, 2002.

[3] J. Barbay and C. Kenyon. Adaptive intersection and t-threshold problems. In *Proceedings of the thirteenth ACM-SIAM Symposium On Discrete Algorithms (SODA)*, pages 390–399, 2002.

[4] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernandez, M. Kay, J. Robie, and J. Simon. XML Path language (XPath) 2.0. Technical report, W3C Working Draft, November 2003. http://www.w3.org/TR/xpath20/.

[5] N. Bruno, N. Koudas, and D. Srivastava. Holistic twig joins: optimal XML pattern matching. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 310–321. ACM Press, 2002.

[6] Y. Chen, S. B. Davidson, and Y. Zheng. Blas: An efficient xpath processing system. In *Proceedings of the ACM SIGMOD*, International Conference on Management of Data. ACM, June 2004. DBLP, http://dblp.uni-trier.de.

[7] E. Chiniforooshan, A. Farzan, and M. Mirzazadeh. Worst case optimal union-intersection expression evaluation. In L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, editors, *ICALP*, volume 3580 of *Lecture Notes in Computer Science*, pages 179–190. Springer, 2005.

[8] J. Clark and S. DeRose. XML Path language (XPath). Technical report, W3C Recommendation, November 1999. http://www.w3.org/TR/xpath/.

[9] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Adaptive set intersections, unions, and differences. In *Proceedings of the* 11^{th} *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 743–752, 2000.

[10] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Experiments on adaptive set intersections for text retrieval systems. In *Proceedings of the 3rd Workshop on Algorithm Engineering and Experiments, Lecture Notes in Computer Science*, pages 5–6, Washington DC, January 2001.

[11] V. Estivill-Castro and D. Wood. A survey of adaptive sorting algorithms. *ACM Computing Surveys*, 24(4):441–476, 1992.

[12] T. Grust. Accelerating xpath location steps. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 109–120. ACM Press, 2002.

[13] H. Jagadish, S. Al-Khalifa, L. Lakshmanan, A. Nierman, S. Paparizos, J. Patel, D. Srivastava, and Y. Wu. Timber: A native XML database. *VLDB*, 11(4):274–291, 2002.

[14] S. R. Jérémy Barbay. Succinct encoding for XPath location steps. Technical Report CS-2006-10, University of Waterloo, Ontario, Canada, 2006.

[15] D. G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM J. Comput.*, 1986. 15(1):287–299.

[16] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt, and J. F. Naughton. Relational databases for querying XML documents: Limitations and opportunities. In *The VLDB Journal*, pages 302–314, 1999.

[17] N. Zhang, M. T. Özsu, A. Aboulnaga, and I. F. Ilyas. XSEED: Accurate and Fast Cardinality Estimation for XPath Queries. In *to appear Proc. 22nd Int. Conf. on Data Engineering (ICDE) 2006*, 2006.