Efficient Algorithms for Context Query Evaluation over a Tagged Corpus

Jérémy Barbay
Departamento de Ciencias de la Computación (DCC),
Universidad de Chile, Santiago, Chile.
Email: jbarbay@dcc.uchile.cl

Alejandro López-Ortiz
Cheriton School of Computer Science (CSCS),
University of Waterloo, Canada.
Email: alopez-o@uwaterloo.ca

***Abstract*—We present an optimal adaptive algorithm for context queries in tagged content. The queries consist of locating instances of a tag within a context specified by the query using patterns with preorder, ancestor-descendant and proximity operators in the document tree implied by the tagged content. The time taken to resolve a query Q on a document tree T is logarithmic in the size of T, proportional to the size of Q, and to the difficulty of the combination of Q with T, as measured by the minimal size of a certificate of the answer. The performance of the algorithm is no worse than the classical worst-case optimal, while provably better on simpler queries and corpora. More formally, the algorithm runs in time $\mathcal{O}(\delta k \lg(n/\delta k))$ in the standard RAM model and in time $\mathcal{O}(\delta k \lg\lg \min(n,\sigma))$ in the $\Theta(\lg(n))$-word RAM model, where k is the number of edges in the query, δ is the minimum number of operations required to certify the answer to the query, n is the number of nodes in the tree, and σ is the number of labels indexed.**

I. Introduction

We consider efficient algorithms for resolving queries in a tagged text corpus. The tags delimit regions which are either properly contained within one another or mutually disjoint. As such, the tags can be interpreted as a fully parenthesized expression or document tree, with the ancestor relationship denoting containment and the sibling relationship denoting sequential ordering in the linear presentation of the text. Among well known examples of tagged content are SGML, XML, HTML, RTF (rich text format) and PostScript. We are interested in a particular type of query consisting of a sequence of tags with ancestor-descendant or sequential relationship between them, which are similar (though not identical) to *twig queries* in XML content.

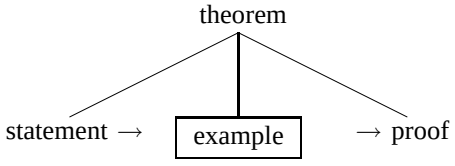
A query in a tagged corpus consists of an interleaving of keywords and structural terms that best denote the information that is being seeked. We illustrate with two examples how structural information can be used to create richer queries. First, consider a corpus consisting of bibliographic information. To retrieve a a list of all authors we query for the pattern (AUTHOR:…). In this case the parenthesis denote the fact that we are only interested in the regions of the text tagged as author-name together with the text contained within, denoted in this case by the ellipsis. A more complicated example would be to search for all books written by Shakespeare. In this case we first restrict the search space to book entries, and within those the ones listing Shakespeare as author. Since we are searching for the titles only we request the text within the title field as follows: (BOOK: (AUTHOR: (Shakespeare)) (TITLE:…)). Observe that the keyword Shakespeare is itself within parenthesis to indicate that Shakespeare should be *contained* within the author field. The same query without parenthesis around Shakespeare would not match `<author> William Shakespeare </author>` nor `<author> <lastname> Shakespeare </lastname> <firstname> William </firstname> </author>`

In general, query languages have been enlarged in various different ways to support the tag structure of the text. Examples of such are the Westlaw legal search service (http://www.westlaw.com), started in 1975 and more recently the XPath query language [4], [7]. In this work we consider an extension which is similar (but more powerful) to a type of twig query which evaluates to all instances in the corpus of a target. The target is a tag within a context specified by additional preorder, ancestor-descendant and proximity patterns in the query. We term these queries *context queries*.

A context query gives a sequence of tags and the relationship between them in the tag tree (XML tree). Recall that a linearly, properly nested tagged text such as HTML and XML can be interpreted as a tree (see Figure 1).

For the case of text it has been observed that proximity is also a useful primitive and hence it is supported in many commercial query languages. For one, it is often used to find passages where a given subject is being discussed. For example, to find all places where President Clinton discussed the AIDS crisis in Africa, a query such as Clinton within.3.of Africa within.3.of AIDS would find all text portions where the words Africa, Clinton and AIDS appear in a span of at most nine words.

The examples above show that nesting and proximity are relevant components of a query. In this last example we illustrate the use of context. Consider a query to find all the examples that follow the statement of a theorem but precede its proof in the text. This query would be denoted as follows in the document tree implied by the tagged content:



```
<library><book> <title> Combinatorial
Optimization </title> <author> <first>
Alexander </first> <last> Schrijver
</last> </author> <publisher> Springer
</publisher> </book> <book> <title>
Introduction to Information Retrieval
</title> <author> <first> Christopher
</first> <last> Manning </last> </author>
<author> <first> Prabhakar </first> <last>
Raghavan </last> </author> <author>
<first> Hinrich </first> <last> Schutze
</last> </author> <publisher> Cambridge
University Press </publisher>
</book></library>
```

```
<library>
   <book>
      <title>Combinatorial Optimization</title>
      <author>
         <first>Alexander</first>
         <last>Schrijver</last>
      </author>
      <publisher>Springer</publisher>
   </book>
   <book>
      <title>Introduction to Information Retrieval</title>
      <author>
         <first>Christopher</first>
         <last>Manning</last>
      </author>
      <author>
         <first>Prabhakar</first>
         <last>Raghavan</last>
      </author>
      <author>
         <first>Hinrich</first>
         <last>Schutze<last>
      </author>
      <publisher>Cambridge University Press</publisher>
   </book>
</library>
```

Fig. 1. Tagged text with its corresponding tree structure representation

To sum up, context queries support ancestor-descendant, proximity and precedence operators which are useful in narrowing the set of potential answers.

A. Previous Work

Bruno *et al.* [5] observed that it is possible to compute the answer to a Twig query in time linear on the input and output size in the worst case. However, for certain problems the worst case time can be a large overestimate of the actual required time for a given input. In particular, Demaine *et al.* [8] showed that while the worst case complexity for the intersection of sorted arrays is $\Theta(n)$, there are instances in which set intersection can be computed in $O(1)$ time such as, for example $\{1, 2, \ldots, \lceil n/2 \rceil\} \cap \{\lceil n/2 \rceil + 1, \ldots, n\}$). In this case a single comparison between the last element of the first sorted array and the first element of the second array would suffice to determine that the intersection is empty namely $\lceil n/2 \rceil < \lceil n/2 \rceil + 1$. Following up on this Demaine *et al.* gave an adaptive set intersection algorithm whose running time is proportional to the difficulty on the instances as measured by a difficulty metric D and input size n. For simple instances the performance is far superior to the worst case, while it is never worse than the worst-case $\Theta(n)$ algorithm.

The same observation applies for context queries, in which the worst case occurs only in certain contrived examples over a corpus with a complex tag structure. In practice such instances are usually rare, and hence there would be practical benefits from an adaptive algorithm which is optimal over every instance. Moreover, it has been observed that careless evaluation of the query may result in large intermediate results, even if the final answer is small [5].

The main techniques used so far to speed up query time are indexing and careful order of evaluation of the components of the query. Grust [10] proposed a labeling scheme based on the prefix and postfix rankings of the nodes. The resulting algorithm has query time linear in the worst case and sublinear in practice. Shanmugasundaram *et al.* [15] proposed storing XML in relational databases, in order to benefit from the extensive optimizations developed for those database systems, but this tends to yield algorithms with very large intermediate results as described above. Bruno *et al.* [5], proposed an algorithm that considers the query as a whole and avoids unnecessarily large intermediate results using only a constant amount of space in addition to the space required to save the results.

In this paper we give an optimal adaptive algorithm for context queries in tagged text. The algorithm runs in $\mathcal{O}(\delta k)$ searches, where k is the number of terms in the query, and δ is the minimum number of operations required to certify the answer to the query. The time required by each search depends of the implementation chosen for the index. In the standard RAM model the index can be implemented via postings lists, supporting the searches in amortized logarithmic time so that the algorithm runs in time $\mathcal{O}(\delta k \lg(n/\delta k))$, where n is the number of nodes in the tree whose label appear in the query. In the $\Theta(\lg(n))$-word RAM model, the index can be encoded in a *succinct data structure* [1], supporting searches in time $\mathcal{O}(\lg\lg\min(n, \sigma))$ so that the algorithms runs in time $\mathcal{O}(\delta k \lg\lg\min(n, \sigma))$, where σ is the number of labels indexed.

The paper is organized as follows: In Section II we define formally the queries that we consider, and the notion of certificate of the answer to a query on a given corpus. In Section III we describe our adaptive algorithm to solve those queries using the operators defined in the previous section. We conclude in Section IV with a discussion of our results and a perspective on future research.

II. Definitions

We consider a tagged text corpus with the tags defining regions that are either properly nested or disjoint. As such, the text can be represented as a tree with edges denoting containment relationships between tags. This graph forms a tree which is known as the document tree corresponding to the corpus. Nodes in the tree are labeled by the tag region they represent.

We consider context queries, partially inspired by the twig queries originally defined by Bruno *et al.* [5], with the key differences that context queries distinguish a particular node

among the pattern and support successor-predecessor operators between nodes. The distinguished node model allows the algorithm to avoid materializing a number of different ways in which the same target node satisfies the answer, which leads, in the worst case, to a combinatorial explosion.

More formally,

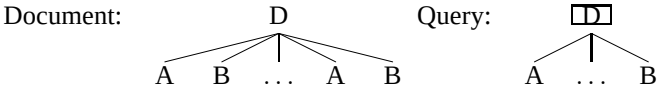
Definition 1 *A* **context query** *is a directed graph such that*

- *it has a distinguished node called the* `target`*;*
- *each node has an associated tag or label*
- *each directed edge is labeled as one of* `ancestor`*,* `descendant`*,* `following`*, and* `preceding`*;*
- *each directed edge is associated to a maximal preorder distance* `distance` $\in \{1, \ldots, n, \infty\}$*, where* ∞ *is coding for the absence of restriction.*

Definition 2 *A node* x *in the document tree* T *is a* **match** *to a context query* Q *if its label is the same as the target node in* Q *and if each of the other nodes of* Q *can be associated to a node of the document so that the relations described by the edges are all respected.*

A context query searches for occurrences of a given node within a given context. Thus the expressivity of twig queries is enriched by supporting precedence operators and a distinguished node. Observe that while it is true that context queries (without `following` and `preceding`) could be evaluated by performing the corresponding twig query without a distinguished node and then filtering the result set for duplicates, the twig query result set can be exponentially larger.

For example consider the following document tree and context query:



In this example the query has a single match, namely the node D whereas the corresponding twig query would produce $m(m-1)/2$ matches, where m is the number of consecutive pairs $A()B()$ present in the query.

Hence the concept of context queries allows both for richer expressivity and more efficient execution of certain queries which might otherwise be too burdensome.

Figure 2 shows a context query for a medical file for all patients to whom have been administrated a test with result Y after having been prescribed 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`.

III. An Adaptive Algorithm for Context Queries

In this section we first give a difficulty measure for a given instance which gives a lower bound on the minimum amount of work that an algorithm must perform over an instance. We then give an algorithm whose running time is proportional to the difficulty of the given instance.

A. Difficulty of an Instance

Any correct algorithm must be able to prove that the answer it produces is correct. We formalize this notion through the concept of certificate of an instance, based on basic operations on the document tree, all supported by basic encodings of the related ordinal tree and binary relation:

Definition 3 *Consider a context query* Q *of* k *edges on a multi-labeled tree* T *and a set of nodes* S *answering* Q *on* T*. A* **certificate** *of* S *is a sequence of assertions of types "*x *is a descendant of* y*", "*x *and* y *are within distance* `distance` *in preorder" and "*x *is the first node labeled* A *after* y *in preorder", such that* T *satisfies those assertions and furthermore for any other tree* T' *satisfying these assertions, the answer to* Q *in* T' *is the same set* S*.*

While the notion of minimal certificate is more common in computational complexity, we use it here to measure the difficulty of an instance, as a lower bound on the smallest number of operations performed by a non-deterministic algorithm to answer the query on the document.

Definition 4 *Consider a context query* Q *of* k *edges on a document tree* T *on* n *nodes and* σ *labels, and a set of nodes* S *supposedly answering* Q *on* T*. The* **non-deterministic complexity** *of* Q *on* T *is the length of the shortest certificate of the answer of* Q *on* T*.*

The non-deterministic complexity provides a lower bound which is not tight, but more importantly it provides a measure of the difficulty of the instance, as shown by the results of our algorithm.

B. Strategy of the Algorithm

Observe that the answer to a context query is a subset of the set S of nodes with the same label as the target node. A straightforward algorithm would be to iteratively test each member of the set S. However, this can be rather inefficient if the result set is comparatively much smaller. In general fully resolving one aspect of a query expression without paying attention to others can result in large inefficiencies. For example, in the evaluation of Boolean set operations in sets and in database queries, it is not hard to construct an SQL or keyword based query in which one of a set of complex views being joined is actually empty, and determining this first would greatly reduce the amount of work. Similarly, for context queries, consider the impact of the order of evaluation in a query tree with two subtrees, one of which has a large evaluation cost and while the second readily evaluating to the empty set. An algorithm that commits to fully evaluating the more complex of the subtrees would be arbitrarily slower than one which hedge its bets between each of the subtrees. Hence proper care must be taken in the order of evaluation of the overall query (this observation is termed "holistic query evaluation" in the field of database research [5]). This means that if we wish to match the most efficient evaluation possible

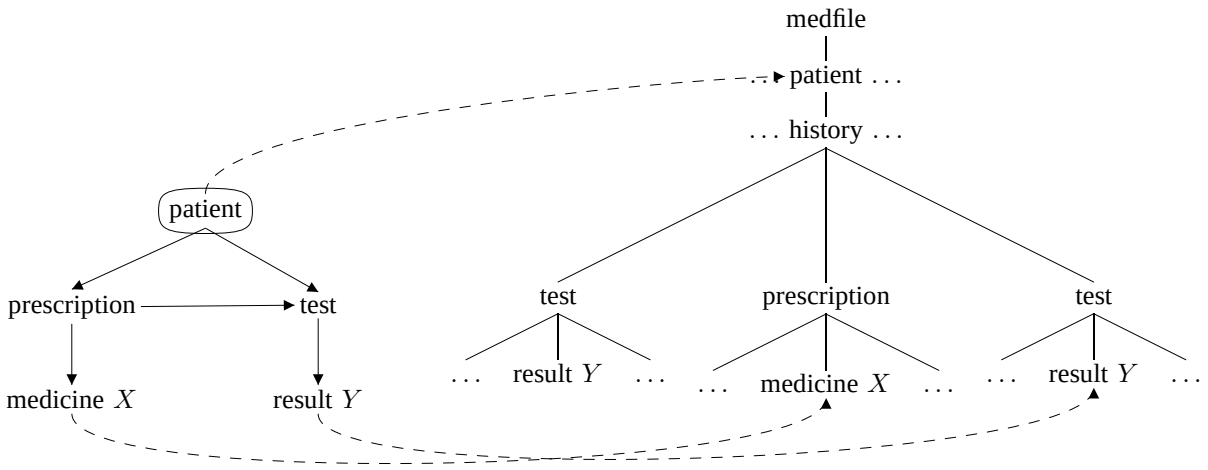


Fig. 2. A context query on the left, on a subset of a medical database depicted on the right. Downward edges correspond to `descendant` axes, and horizontal edges to $\texttt{following}_{pre}$ axes. The target node (here the `patient` node) is circled.

the algorithm must actively examine all relations in the query searching for the best possible way to select the next candidate.

First, we denote each edge in the query by (A, r, B) which represents the requirement in the query that there be a node labeled A in relation r with a node labeled B in the document tree of the text, where $r \in \{\text{ancestor}, \text{predecessor}, \text{within}\}$.

The algorithm cycles repeatedly through the edges of the query in an arbitrary tour of minimal length. Such a tour visits at least once and at most twice every edge (because of the minimality), so that its length is at most $2k$ if there are k edges in Q. For each edge (A, r, B), consider A the previous query-node visited, and a the first A-node in preorder of T which is a candidate for a match of Q (the definition is symmetrical if B is the previous query-node visited). The algorithm searches for the first B-node in preorder of T which could be in relation r with a. If there is such a node b, it is added to the potential match and the algorithm continues its tour. If there is no such node, then obviously a is not part of a match of Q, hence the algorithm fixes b to the first B-node which could be in relation r with a preorder *successor* of a, and continues its tour with a potential match constituted only of b.

For the evaluation of the query as a whole (i.e. satisfying all relations simultaneously) we observe that the set of nodes with the same given tag forms an ordered list of their positions in the corpus in left to right order of the text. The position can be given as a byte offset, token count, or any other such suitable mechanism. Hence an ancestor/descendant relationship can be encoded by a predecessor/successor relation in the ordered list. In particular node b is contained within node a if the opening tag of b appears after the opening tag of a but before the closing tag of a and node b is a successor of node a if the closing tag of a appears before the opening tag of b.

Assume that we have identified a set of nodes which satisfy all relations tested thus far. Let (A, r, B) be the next edge in the traversal of the query. Since we traverse the graph in connected order there is already a candidate node a associated to the requirement A of the edge. We then request the successor of position a in the ordered set of labels B. Let b be this node. The algorithm verifies if (a, r, b) is indeed a valid relation in the document tree of the text. If so it increments the count of number of relations satisfied thus far. Otherwise the counter is reset to zero and we advance a by searching for the first closing tag of type A that appears after the opening tag of b. We repeat this process in the order of tour of the query graph. Whenever the counter of satisfied relations is the same as the number of edges in the query graph Q, this means that all relations are satisfied the algorithm outputs the current target node t and it advances this node.

C. Data Structures

To simplify the understanding of the algorithm, it might help to envision the testing of each of the successor/descendant operations as simple offset comparisons in the sorted list as described above. However, we note that any one of a variety of data structures implementing ordinal trees and binary relations can be used (e.g. [9], [13], [11]). Each of them has a different trade-off between the space used and the time required to search in it, hence for generality we express the complexity of our algorithms in terms of the number of "search" operations performed on these data structures, so that the complexity of the algorithm can be inferred for each data structure.

Two classic examples are (i) a binary search on a sorted array which finds the rankof a particular element in a sorted array of n_A elements in time $\mathcal{O}(\lg n_A)$, and a straightforward variant can be used to search the positions of a set of m increasing values in time $\mathcal{O}(m \lg(n_A/m))$ [2] and (ii) a compressed bit-vector [14], [12], supporting the search of the rank of a particular element in constant time, at the expense of space; or using less space at the expense of time [1].

D. The Algorithm

We now give a formal description of the procedure and its proof of correctness.

Theorem 1 *Consider a context query Q of k edges on a multi-labeled tree on n nodes and σ labels. There is an algorithm solving Q in $\mathcal{O}(\delta k)$ operator calls and in time $\mathcal{O}(\delta k \lg(n/\delta k))$ on a document tree T, where δ is the non-deterministic complexity of Q on T. This is optimal in the worst case over all instances of same difficulty, document and query size, for any algorithm computing a certificate as described in Definition 3.*

Proof: For simplicity, we introduce a sentinel node positioned at ∞ that matches all labels and is a successor to all nodes. The algorithm goes as follows: starting from the

Algorithm 1 `Solve_Query`(Q)

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

```
Let A initially be the label of the target node in Q;
NumSatisfiedRelations ← 0;
a ← the first available A-node of the document.
while a ≠ ∞ do
  (A, r, B) ← the next edge of Q;
  b ← the next node matching B and potentially in relation
  r with a;
  if a is in relation r with b then
    NumSatisfiedRelations ←
    NumSatisfiedRelations + 1;
    if NumSatisfiedRelations = 2k then
      Output the current node matching the target node;
      a ← the next target node in the document tree;
    end if
  else
    NumSatisfiedRelations ← 0;
  end if
end while
```

`target` node, it cycles through the edges of the query Q, updating for each query node A the matching node x in the multi-labeled tree, such that any A-node preceding x in preorder has already been considered and can be ignored.

After a tour where $2k$ consecutive matches have been successfully checked, the algorithm has found a match for Q and can output the tree-node corresponding to the `target` node. Observe that for every $2k$ such checks that the algorithm considered, a correct non-deterministic algorithm needs to either confirm all of them explicitly if the target node was found, or refute at least one relationship to prove that the current node set is not a valid answer as it can be shown by a straightforward adversarial argument[1]. Note that in our model there are two ways in which an algorithm (deterministic or otherwise) can learn that two given nodes a and b are not in the desired relation r. One is by a direct testing if b is a successor of a by testing for the condition $a < b$. The second is indirectly, by testing the condition $a < b'$ and then deriving from the left to right ordering of the text that $b' < b$. Analogous conditions hold for descendant and within queries.

From this it follows that every $2k$ checks the algorithm above discovers at least one refutation which is also known to the non-deterministic algorithm. If the refutation was discovered explicitly by the non-deterministic algorithm then we have a ratio of 1:$2k$ in the amount of extra work performed by our algorithm. Alternatively, if the refutation was discovered implicitly by the non-deterministic algorithm, this is due to a refutation for an element b' appearing before b. Observe however, that our algorithm must have considered b' as a potential candidate and discarded it, hence our algorithm also knows that b has been refuted implicitly and it wouldn't be under current consideration.

In sum, 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.

To show correctness, first we observe that if a node t matching the label of the target node is output then it satisfies all k relations in the context query (some of them having been tested twice) and hence by construction t is a correct answer. Conversely we need to show that if there is a node t which is a valid answer then the algorithm will report it.

Since t is a valid answer there is a set of nodes S^* in the text instantiating to the query nodes such that all the relations are satisfied. We will show that the algorithm will either discover this set of nodes or an alternative set of satisfying nodes involving the same target node t will be discovered. We prove this by showing that the candidate node set held by the algorithm for a given target node t is never to the right of the valid answer set corresponding to that target node t.

We prove this by induction on the number of steps performed. Initially this is trivially true, as the candidate set of nodes is empty. Now by induction suppose the algorithm has a candidate set of nodes each of which appears no later than the corresponding node in S^*. When trying to satisfy a relation corresponding to an edge (A, r, B) the algorithm searches for the next node b that is in relation r with a. If this node b is to the right of the corresponding node b^* in S^*, we claim that b^* also satisfies the relation r. Observe that the relation operators successor, descendant and within.n appear in left to right order in the text, hence we have $a^* < b^*$ and $a < b$. By hypothesis we also have $a < b^* < b$, hence b^* is also a descendant/successor/within.n of a and thus the algorithm must have selected $b \equiv b^*$ if not an earlier instance. Hence the condition holds.

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).

[1]If none of the relationships have been refuted by the non-deterministic algorithm and it rejects the node, then an adversary can change the instance to have the relationships hold and the non-deterministic algorithm is incorrect.

The optimality results from a simple reduction to the alternation analysis of the conjunctive queries in the intersection problem [3]. A binary relation composed of t relations between n objects and σ labels can be encoded as a labeled tree of $t+1$ nodes and $\sigma+1$ labels, where the root has a label of his own and each branch corresponds to one object x, the nodes of this branch being labeled with the labels associated to x, in order. The conjunctive query composed of k labels then corresponds to a twig pattern of k edges, starting from the root and listing k nodes labeled with the k labels in order. The matches of this twig pattern trivially yield the answer to the conjunctive query.

Since Barbay and Kenyon proved a computational lower bound of $\Omega(\delta k \lg(n\delta k))$ for solving the conjunctive query in the comparison model, where δ is the non deterministic complexity of the instance, k the number of terms in the query, and n the number of nodes which match at least one of the label of the query, this trivially yield the desired optimality in our model. ■

Consider for instance the execution trace presented in Figure 3 and 4, for a tour of the query considering `test` nodes before `prescription` nodes. 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) using at most $2k$ times as many operation as the best non-deterministic algorithm.

IV. Conclusions

We propose in this paper an adaptive algorithm for the evaluation of context queries in tagged text. The algorithm evaluates these queries in $\mathcal{O}(\delta k)$ search operations, 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 corresponds to an execution in $\mathcal{O}(\delta k \lg(n/\delta k))$ in the standard RAM model, where n is the number of nodes in the tree; and to an execution time in $\mathcal{O}(\delta k \lg\lg\min(n,\sigma))$ in the $\Theta(\lg n)$-word RAM model, where σ is the number of labels indexed.

A natural direction for future research is to extend our technique to support a broader range of 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.* [6] is a first step towards an algorithm to support more complex node type tests.

An interesting open problem 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 [2], 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 assume that the algorithm uses a specific encoding of the document tree.

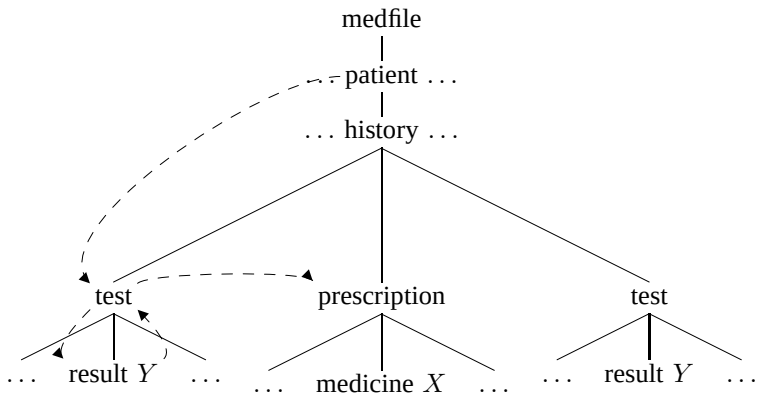


Fig. 3. The first part of the execution trace of the algorithm with another tour of the query given in Figure 2 : the trace stops when the algorithm proves that the first `node` does not belong to a match, because no `prescription` node precedes it.

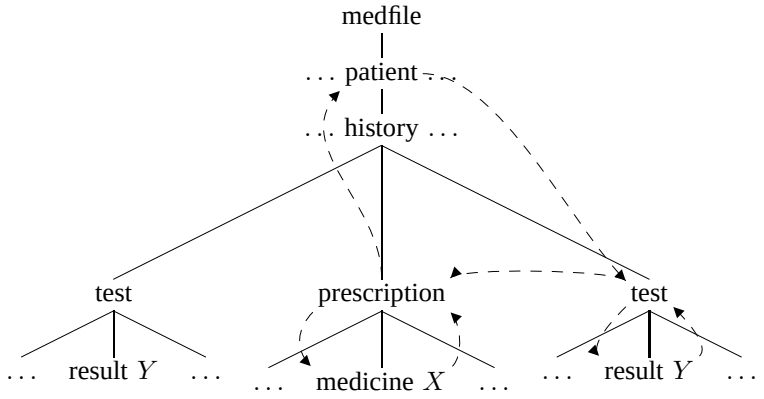


Fig. 4. The rest of the execution trace, where the algorithm follows the tour through the nodes labeled `prescription`, `medicine`, `prescription`, `patient`, `test`, `result`, `test`, and `prescription`, at which point it proved a match, having matched seven consecutive edges, the length of the tour of the query.

References

[1] J. Barbay, M. He, J. I. Munro, and S. S. Rao. Succinct indexes for strings, binary relations and multi-labeled trees. In *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 680–689. ACM, 2007.

[2] J. Barbay and C. Kenyon. Adaptive intersection and t-threshold problems. In *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 390–399. Society for Industrial and Applied Mathematics (SIAM), January 2002.

[3] J. Barbay and C. Kenyon. Alternation and redundancy analysis of the intersection problem. *ACM Trans. Algorithms*, 4(1):1–18, 2008.

[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] 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 (LNCS)*, pages 179–190. Springer, 2005.

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

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

[9] R. F. Geary, R. Raman, and V. Raman. Succinct ordinal trees with level-ancestor queries. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1–10, 2004.

[10] 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.

[11] G. Jacobson. Space-efficient static trees and graphs. In *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 549–554, 1989.

[12] D. Okanohara and K. Sadakane. Practical entropy-compressed rank/select dictionary. In *Proceedings of the 14th International Workshop on Algorithms and Data Structures (WADS)*, volume 1671 of *Lecture Notes in Computer Science (LNCS)*. Springer-Verlag, 2007.

[13] R. Raman, V. Raman, and S. S. Rao. Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete algorithms*, pages 233–242, 2002.

[14] K. Sadakane and R. Grossi. Squeezing succinct data structures into entropy bounds. In *Proceedings of the 17th annual ACM-SIAM symposium on Discrete algorithm*, pages 1230–1239, 2006.

[15] 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.