

Twig Patterns: From XML Trees to Graphs*

[Extended Abstract]

Benny Kimelfeld and Yehoshua Sagiv
The Selim and Rachel Benin School of Engineering and Computer Science
The Hebrew University of Jerusalem, Edmond J. Safra Campus
Jerusalem 91904, Israel

{bennyk,sagiv}@cs.huji.ac.il

ABSTRACT

Existing approaches for querying XML (e.g., XPath and twig patterns) assume that the data form a tree. Often, however, XML documents have a graph structure, due to ID references. The common way of adapting known techniques to XML graphs is straightforward, but may result in a huge number of results, where only a small portion of them has valuable information. We propose two mechanisms. Filtering is used for eliminating semantically weak answers. Ranking is used for presenting the remaining answers in the order of decreasing semantic significance. We show how to integrate these features in a language for querying XML graphs. Query evaluation is tractable in the following sense. For a wide range of ranking functions, it is possible to generate answers in ranked order with polynomial delay, under query-and-data complexity. This result holds even if projection is used. Furthermore, it holds for any tractable ranking function for which the top-ranked answer can be found efficiently (assuming that equalities and inequalities involving IDs of XML nodes are permitted in queries).

1. INTRODUCTION

Twig patterns are simple tree-structured queries for XML that include three basic language elements, namely, arbitrary *node conditions*, *parent-child edges* and *ancestor-descendant edges*. These features make it possible to pose queries with only a limited knowledge of the XML hierarchy, the names of elements and the precise data stored under each element. Furthermore, fuzzy conditions (e.g., “about(*axis,value*)” [23]) can be used and, so, twig patterns are applicable to information-retrieval (IR) as well as database settings. In summary, twig patterns provide an appealing tradeoff of expressiveness vs. simplicity and flexibility.

Twig patterns, however, suffer from some severe drawbacks. For one, XML documents (e.g., DBLP¹ and Mondial²) are often graphs and not trees, due to ID references. Consequently, there could be many different paths between any given pair of nodes, leading to a potentially huge number of matches for descendant edges when applying the conventional approach [24] of generalizing twig patterns to XML graphs. Our experience shows that some simple and natural twig queries over DBLP have an unexpected huge number (tens of thousands) of answers, if ID references are taken into

*This research was supported by The Israel Science Foundation (Grants 96/01 and 893/05).

¹<http://dblp.uni-trier.de/xml>

²<http://www.dbis.informatik.uni-goettingen.de/Mondial>

account. Most of these answers are derived from rather weak semantic relationships among XML elements. For example, nodes with high out-degree usually represent weak semantic connections and, yet, they are included in many paths. As another example, long paths are commonly viewed (e.g., [4, 13, 11, 1, 15, 19]) as an indication of less meaningful relationships, but XML graphs have many such paths. Thus, querying XML graphs using twig patterns is often ineffective.

In this paper, we investigate essential properties for facilitating effective querying of XML graphs. In particular, we present a language that incorporates filtering and ranking mechanisms while retaining the simplicity and efficiency of twigs. In our framework, nodes and edges of XML graphs have weights, which is not new. But our treatment of weights is novel in two aspects. First, when a user formulates a query, she can override and fine-tune some of these weights. This is an essential feature, since different users have diverging views regarding the strength of some semantic connections. Second, weights are used not just for ranking, but also for filtering; that is, the user can decide from the outset that she is not interested in paths above a certain length.

The most important feature of our language is the ability to generate the top-*k* answers quickly, according to a wide range of ranking functions. In principle, the following two properties are necessary for efficiently finding the top-*k* answers. First, the ranking function should be efficiently computable. Second, it should be possible to generate the first (i.e., top-ranked) answer efficiently. We show that in our language, these two properties are also sufficient for generating answers in ranked order with polynomial delay (in the size of query and the XML graph) between consecutive answers. We identify a large family of ranking functions that have the above properties. These functions satisfy a monotonicity condition that makes it possible to compute the top answer bottom-up.

It is shown that our complexity results hold even if projections are used and duplicates are eliminated. Note that simply applying projection in the last step is not enough for deriving this result, since intermediate results could be exponentially larger than the final one. Moreover, this result holds even if the ranking function depends on data that is projected out and, hence, not included in the final result.

Earlier work on ranked evaluation over XML [22, 14, 6, 3, 2, 20] considered only trees and, as a result, did not address the main issues of this paper. The approaches of [22, 14] are based on the *threshold algorithm* [10], which is designed for joining objects that have shared keys and, hence, cannot guarantee polynomial delay when evaluating tree-structured queries. The work of [3, 2, 20] considered ranking functions that measure the amount of *structural relaxations* of patterns and, so, are different from ours. Evaluating twig pattern over general graphs was discussed in [24], but they did

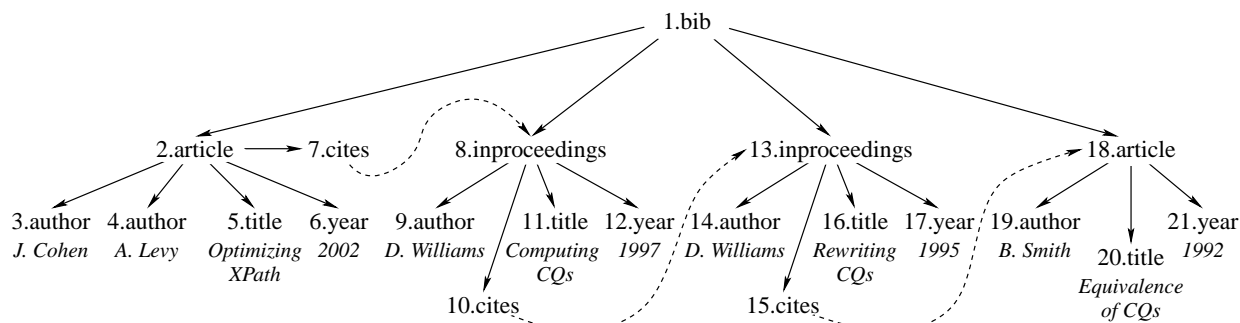


Figure 1: An XML graph G

not address the issue of dealing with many answers that have varying degrees of semantic strength. In particular, they neither considered ranking nor provided a formal upper bound on the running time of their algorithms. A language that extends XPath to general XML graphs was discussed in [9]. That language enables the user to bound the length of a path by specifying how many *steps* the path can follow, where a step is an XPath axis. However, that language does not handle weights or ranking functions. Furthermore, the complexity of the language is not stated. The work of [21] is fundamental to evaluating regular path expression over graph databases. It should be noted that defining edge conditions by means of regular expressions can be easily incorporated in our query language, while preserving all of our complexity results. In a recent paper [17], we considered ordered evaluation of acyclic conjunctive queries. Some of the techniques presented there are needed for proving the complexity results of this paper.

2. XML GRAPHS AND TWIG PATTERNS

An XML graph is directed and rooted. The nodes have labels and possibly values. There are two types of edges: *element* edges and *reference* edges. The element edges form a spanning tree that has the same root as the XML graph itself. We use the terminology of XML nodes and XML edges to distinguish them from nodes and edges of trees that represent queries.

EXAMPLE 2.1. Figure 1 shows an XML graph G that represents bibliographic data. Values are written in italic and reference edges are depicted by dotted lines. This graph comprises publications of various types that are described by labels, e.g., *article*, *inproceedings*, etc. The label of the root is *bib*. Citations are represented by reference edges. Each node is identified by a unique integer. \square

We now describe twig patterns and their conventional evaluation over XML graphs. A *twig* is a directed tree T with *child* edges and *descendant* edges. Each node n of T has an attached unary predicate, denoted by $cond(n)$, that is defined over XML nodes. For example, $cond(n)$ can specify that the label of the XML node is *article*, or that the string “query optimization” should appear in the value attached to the XML node. It can also be any boolean combination of the two. In general, $cond(n)$ can be any condition that is checkable in polynomial time.

Following [24], a match of a twig in an XML graph is defined in the usual way, except that element edges and reference edges are treated equally. Formally, a *match* of a twig T in an XML graph G is a mapping M from the nodes of T to the nodes of G , such that: (1) For each node n of T , the XML node $M(n)$ satisfies $cond(n)$; (2) For each child edge $e = (n_1, n_2)$ of T , the XML graph G has either

an element or a reference edge from $M(n_1)$ to $M(n_2)$; and (3) For each descendant edge $e = (n_1, n_2)$ of T , the XML graph G has a directed path (comprising edges of any type) from $M(n_1)$ to $M(n_2)$. The path must have at least one edge, but could start and end in the same XML node, due to cycles.³ We use $M(T, G)$ to denote the set of all matches from the twig T to the XML graph G .

EXAMPLE 2.2. Consider the bibliographic data of Figure 1. Intuitively, an evidence for a potential connection between two publications is the existence of a directed path that starts in one publication and leads to the second publication through a number of citations. Accordingly, the twig T_1 of Figure 2(a) is a query about potential connections between the publications written by B. Smith on equivalence of queries and other work that has been done since 1997. Note that a child edge is represented by a single line whereas a descendant edge is depicted by a double line. The direction of edges is from top to bottom.

All the predicates of T_1 have at least one conjunct that refers to the label of the corresponding XML node. This conjunct is represented by either an explicit label (e.g., *article*) or the wild-card symbol $*$ that is always satisfied (i.e., the same as **true**). Some predicates have a second conjunct (the \wedge symbol is not used explicitly).

A match of T_1 in G must map the node labeled with *article* to node 18 of G , in order to satisfy the two predicates attached to the children of the former node. The predicate of the root of T_1 has one conjunct, namely $*$, and it must have children labeled with *title* and *year*, where the value of the year node is at least 1997. Thus, the root of T_1 can be mapped to either node 2 or 8, since there is a directed path from each of them to node 18. \square

3. DBTWIG QUERIES

When applying twigs to XML graphs (rather than trees), the user might be overwhelmed with a large number of answers. Moreover, the semantic strength varies widely between these answers. That is, some matches are meaningful while many others have low or no semantic value. For example, consider again the twig T_1 of Figure 2(a). A match of this twig in a large XML document (e.g., DBLP) can connect some publication to an article by Smith through a very long sequence of citations or through a book that has a huge bibliography on a wide range of topics.

In this paper, we propose an approach that uses both *filtering* and *ranking*. Filtering excludes matches that are not likely to be meaningful answers. Ranking is used to produce answers in the order

³In principle, we can allow *descendant-or-self* edges and also let the user specify that a path must start and end in different nodes.

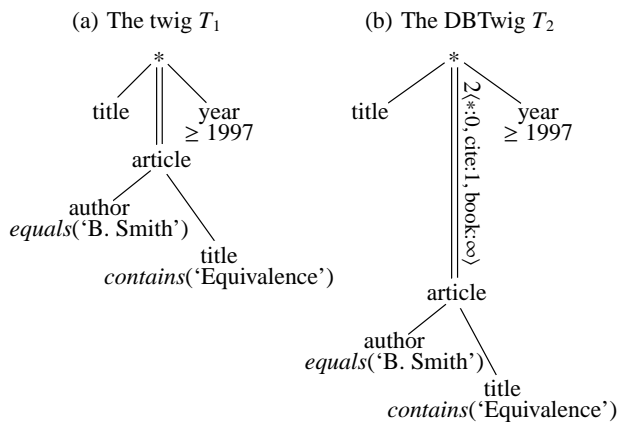


Figure 2: A twig and a DBTwig

of decreasing significance. The challenge is to develop effective filtering and ranking mechanisms that retain the simplicity and efficiency of twigs. These mechanisms should be built into the system, so that formulating queries will be an easy task. However, since the notion of a “semantically significant” answer varies from one person to another, the query language should enable users to tweak the filtering and ranking mechanisms.

3.1 DBTwig Patterns

As illustrated in Example 2.2, a simple yet effective way of filtering out semantically weak matches is by specifying upper bounds on the length of paths that correspond to descendant edges. Other types of conditions are also possible provided that they can be checked in polynomial time. For example, a user can specify that the path corresponding to a given descendant edge should have at most one node labeled with cites and no node labeled with book. Some seemingly simple conditions cannot be checked efficiently (e.g., it is NP-complete to determine whether two given nodes of an XML graph are connected by a path that has no repeated labels).

To fully utilize conditions on lengths of paths, we enrich our data model by adding weights to the nodes and edges of XML graphs. These weights are also used by the ranking functions that are discussed later. Various considerations can be used in order to determine the weights. For example, the weight of a specific XML node can indicate the importance of that node. Similarly, the weight of an XML edge could be derived from the strength of the semantic connection represented by that edge. We will not get into further details, since this has already been investigated (e.g., [4]).

We denote the weights of an XML node v and an XML edge e by $w(v)$ and $w(e)$, respectively; note that these weights are non-negative numbers. The weight of a path is the sum of the weights of all the edges and all the interior nodes.

To simplify the presentation, we use only upper bounds on the weights of paths as filtering conditions. We also provide a mechanism that enables users to fine-tune the weights of nodes that are predefined in XML graphs, by means of specifying weight schemes in the queries that they pose. Each descendant edge may have its own weight scheme that applies only to paths corresponding to that edge. The formal definitions are as follows.

We introduce *distance-bounding twigs* (abbr. *DBTwigs*) that generalize ordinary twigs with two additional features. First, each edge e has a *distance bound* that is denoted by $db(e)$ and is a non-negative number (the default value is ∞). Second, each descendant edge may have a *weight scheme*, denoted by $ws(e)$, that assigns

nonnegative weights to XML nodes according to their labels. The weight scheme $ws(e)$ is a sequence of the form $\langle l_1:w_1, \dots, l_k:w_k \rangle$, where each l_i is either a label or $*$, and each w_i is a non-negative number.

Consider an XML graph G and a DBTwig T . The definition of a match M of T in G is modified as follows. In Part (2), if $e = (n_1, n_2)$ is a child edge of T , then there must be an edge from $M(n_1)$ to $M(n_2)$ that has a weight of no more than $db(e)$. In Part (3), if $e = (n_1, n_2)$ is a descendant edge of T , then there must be a path from $M(n_1)$ to $M(n_2)$ that has a weight of no more than $db(e)$. If, in addition, the descendant edge e has the weight scheme $ws(e) = \langle l_1:w_1, \dots, l_k:w_k \rangle$, then the weight of the corresponding path is calculated after changing the weights of the interior nodes as follows. Suppose that v is an XML node with the label l . The weight of v is replaced with the weight w_i , where l_i is either l or $*$. If more than one l_i matches the label l of v , then the last one in $ws(e)$ is used. If no l_i matches l , the weight of v remains unchanged.

EXAMPLE 3.1. Consider the DBTwig T_2 of Figure 2(b) and the XML graph G of Figure 1. Suppose that all the XML nodes have weight 1 and all the XML edges have weight 0. Let e denote the only descendant edge of T_2 . The weight scheme of e assigns 0 to all the labels of G , except for the labels *cite* and *book* that get the weights 1 and ∞ , respectively. Since $db(e) = 2$, a path P of G matches e if and only if it has at most two interior nodes labeled with *cite* and no interior node labeled with *book*. Thus, there is only one match of T_2 in G and it maps the root of T_2 to node 8. The root cannot be mapped to node 2 of G , because the path of G from node 2 to node 18 contains three interior nodes that are labeled with *cite*. \square

In practice, users do have to formulate explicit DBTwigs. The actual query language may consist of simpler (and less expressive) features that can be translated into DBTwigs. For example, the user may specify, for a given descendant edge, a set of forbidden labels (i.e., labels that cannot appear in a matching path). She can also specify a set of labels and an upper bound on the total number of occurrences of labels from that set in a matching path.

In [9], filtering conditions are upper bounds on the number of *steps* needed to connect two XML nodes, where a step is an XPath axis. We believe that our approach of using weights and upper bounds makes it easier to express natural conditions for eliminating semantically weak matches.

3.2 Ranking of Matches

DBTwigs, in comparison to twigs, eliminate matches that do not satisfy the distance bounds. This could still leave a large number of answers with varying degrees of semantic strength. Hence, the answers should be presented to users in ranked order. In this section, we discuss ranking functions.

Formally, a *ranking function* ρ defines a numerical value, denoted by $\rho(M, T, G)$, where M is a match of a DBTwig T in an XML graph G . The value $\rho(M, T, G)$ determines the *quality* of M . The matches should be presented to the user in *ranked order*, that is, if $\rho(M_1, T, G) > \rho(M_2, T, G)$, then M_1 should appear before M_2 .

We now give several examples of ranking functions. Note that some of these functions may be unintuitive, but they are needed later in order to demonstrate our results. Consider an XML graph G and a DBTwig T . Let e be a descendant edge of T . We use G_e to denote the XML graph that is obtained from G by replacing the weights of nodes according to the weight scheme of e (if e does not have a weight scheme, then $G_e = G$). Given two nodes v_1 and v_2 of G , the *e-distance* from v_1 to v_2 , denoted by $D_G^e(v_1, v_2)$, is the minimum weight among all paths of G_e from v_1 to v_2 (recall that a

path must have at least one edge and, hence, is a cycle if $v_1 = v_2$). If e is a child edge of T , then $D_G^e(v_1, v_2)$ is just the weight of the XML edge from v_1 to v_2 , if this edge exists, and is ∞ otherwise.

Note that, in principle, a descendant edge of a DBTwig may have two distinct weight schemes: one is used for filtering out matches, as described earlier, and the other is used for ranking.

The first ranking function that we define, $\rho_{\Sigma d}$, is the sum of the e -distances, between the corresponding images under M , over all edges e of T . Formally, we use $\mathcal{E}(T)$ to denote the set of edges of T and define

$$\rho_{\Sigma d}(M, T, G) = - \sum_{(n, n') \in \mathcal{E}(T)} D_G^{(n, n')}(M(n), M(n')).$$

Note that the values returned by $\rho_{\Sigma d}$ are negative, since a larger weight indicates a weaker semantic connections and, hence, the ranking should be lower.

For the next ranking function, $\mathcal{P}(T)$ denotes the set of all paths P of T , such that P starts at the root of T and ends at a leaf. Given a path $P \in \mathcal{P}(T)$, we use $(n, n') \in P$ to denote that the edge (n, n') is in P . The ranking function ρ_h returns the maximum weight among all paths of G that correspond, under M , to paths from the root of T to some leaf. It returns a negative value that is defined as follows.

$$\rho_h(M, T, G) = - \max_{P \in \mathcal{P}(T)} \sum_{(n, n') \in P} D_G^{(n, n')}(M(n), M(n'))$$

Given a match M of a DBTwig T in an XML graph G , let $N(M, T, G)$ denote the minimal number N' , such that G contains a subgraph G' with N' nodes and M is a match of T in G' . Then we define

$$\rho_{ms}(M, T, G) = -N(M, T, G).$$

The next ranking function, $\rho_{\#}(M, T, G)$, counts the number of distinct labels that appear in the image of M , i.e.,

$$\rho_{\#}(M, T, G) = \{ |l \mid n \text{ is a node of } T \text{ and } l \text{ is the label of } M(n) \}.$$

Finally, suppose that the nodes of T contain vague conditions, e.g., “about(‘XML’)” as used in NEXI [23]. We assume that there is a ranking function f that, given a node n of T and a node v of G , returns a numeric value $f(n, v)$ that measures the extent to which v matches n . We use $\mathcal{N}(T)$ to denote the set of nodes of T and define

$$\rho_{fn}(M, T, G) = \sum_{n \in \mathcal{N}(T)} f(n, M(n)).$$

Note that the function f is predefined by the system, but it could use some specific information that is attached to node n (for example, the keywords that are specified in $cond(n)$).

Ranking is not merely sorting according to the filtering conditions. A ranking function can be defined over the whole image of a match; for example, the ranking function $\rho_{\Sigma d}$ is defined as the sum of the e -distances. A filtering condition, on the other hand, refers just to a single edge (n_1, n_2) of a DBTwig (e.g., the distance between the images of the nodes n_1 and n_2 is no more than 5). Thus, ranking functions are not just for sorting the answers. In fact, they are also an advanced form of filtering, since they can be used for eliminating answers that have a rank that is below a given threshold. The user may also specify that only the top- k answers should be generated, e.g., by using mechanisms like SQL’s ORDER BY clause combined with the STOP AFTER operator [7, 8].

3.3 Projections

In many cases, the user is not interested in seeing the whole match of a DBTwig in an XML graph, but rather only wants to get a subset of the nodes. Formally, given a DBTwig T , the user may specify a *sequence of projected nodes*, i.e., a sequence $p =$

$\langle n_1, \dots, n_k \rangle$ of nodes of T . For a match M of T in an XML graph G , we use $\pi_p(M)$ to denote the tuple $\langle M(n_1), \dots, M(n_k) \rangle$. Given T , p and G , the goal is to generate the tuples $\pi_p(M)$ for all matches M of T in G .

One way of handling projections is by applying them as a post-processing phase, i.e., after generating the matches (in a ranked order). This approach, however, is inherently inefficient, since (exponentially) many matches may result in the same answer. Therefore, it is important to develop algorithms that can apply projections (and eliminate duplicate answers) as early as possible.

3.4 DBTwig Queries

Formally, a *DBTwig query* is a triple $Q = \langle T, p, \rho \rangle$, such that T is a DBTwig, p is a projection sequence and ρ is a ranking function. Given a DBTwig query $Q = \langle T, p, \rho \rangle$ and an XML graph G , the *result* of Q in G , denoted by $Q(G)$, consists of all projections $\pi_p(M)$, where M is a match of T in G , i.e., $Q(G) = \{ \pi_p(M) \mid M \in \mathcal{M}(T, G) \}$. To take the ranking function into account, the answers should be given to the user in the order obtained from the following process.

1. Compute $\mathcal{M}(T, G)$.
2. Sort $\mathcal{M}(T, G)$ according to ρ and let M_1, \dots, M_n be the resulting sequence.
3. Apply projection to obtain the sequence $\pi_p(M_1), \dots, \pi_p(M_n)$.
4. Remove duplicates from $\pi_p(M_1), \dots, \pi_p(M_n)$, i.e., delete every $\pi_p(M_i)$, such that $\pi_p(M_i) = \pi_p(M_j)$ for some $j < i$.

The above order can be equivalently defined as follows. Let t_1, \dots, t_m be a sequence that consists of all the tuples of $Q(G)$ without duplicates. First, we define ρ_i ($1 \leq i \leq m$) to be the maximal rank of any match that produces t_i by projection. That is, ρ_i is the maximal number r , such that there exists a match $M \in \mathcal{M}(T, G)$ that satisfies $\pi_p(M) = t_i$ and $\rho(M, T, G) = r$. Now, we say that the sequence t_1, \dots, t_m is in *ranked order* if for all $1 \leq i < j \leq m$, it holds that $\rho_i \geq \rho_j$. The goal is to generate the tuples of $Q(G)$ in ranked order.

4. COMPLEXITY

A complexity analysis of query languages over XML graphs must take into account the fact that the number of answers could be huge. Thus, an evaluation algorithm should be deemed efficient only if it computes the answers incrementally in ranked order, rather than merely generating the whole result before sorting it.

In this section, we consider the complexity of evaluating DBTwig queries. Consider a query Q and an XML graph G . *Polynomial running time* is not a suitable yardstick for measuring the efficiency of algorithms for evaluating Q , since the number of tuples in $Q(G)$ could be exponential in the size of Q (even if G is a tree). The results of [25] imply that $Q(G)$ can be evaluated in *polynomial total time*, i.e., the running time is bounded by a polynomial in the combined size of the input (i.e., Q and G) and the output (i.e., $Q(G)$). (We assume that the conditions attached to nodes can be computed in polynomial time in the size of the input.) Polynomial total time, however, is not good enough for the task of evaluating DBTwig queries, for the following reasons. An algorithm that runs in polynomial total time (e.g., the one of [25]) requires generating all the answers before we can be certain that the top-ranked answer (or the top- k answers) have already been found, let alone before we can start producing the answers in ranked order. Thus, the user gets the first few answers only after the whole result (which could be very large) is generated and sorted. Moreover, recall that the position of a tuple in the ranked order is determined by the maximal rank

over all matches that yield this tuple when projection is applied. Consequently, in order to sort $Q(G)$, we need an additional step of computing for each answer the maximal rank among all matches that generate it. So, we should develop evaluation algorithms that enumerate the answers in ranked order with *polynomial delay* [12], that is, the time between two consecutive tuples is polynomial in the size of the input.

Consider a ranking function ρ . The ρ -EVAL problem is defined as follows. Given a DBTwig query $Q = \langle T, p, \rho \rangle$ and an XML graph G , enumerate all the tuples of $Q(G)$ in ranked order. Another problem of interest, ρ -TOP, is the restriction of ρ -EVAL to the first tuple, namely, given a DBTwig T and an XML graph G , find a top-ranked match, i.e., a match $M \in \mathcal{M}(T, G)$, such that $\rho(M) \geq \rho(M')$ for all $M' \in \mathcal{M}(T, G)$.

To explain our complexity results, let us first consider ranking functions that do not have efficient evaluation algorithms.

PROPOSITION 4.1. *The problems ρ_{ms} -EVAL and $\rho_{\#}$ -EVAL cannot be solved with polynomial delay, unless $P=NP$.*

The above proposition is a direct corollary of the fact that the problems ρ_{ms} -TOP and $\rho_{\#}$ -TOP are NP-hard. The first ranking function, ρ_{ms} , is even intractable to compute; that is, the following problem is NP-complete: given an XML graph (or tree) G , a DBTwig T , a match $M \in \mathcal{M}(T, G)$ and a number r , determine whether $\rho_{\text{ms}}(M, T, G) \geq r$. Note, however, that the ranking function $\rho_{\#}$ is clearly computable in polynomial time. In the sequel, we only consider ranking functions that can be computed in polynomial time in the size of the input (i.e., Q and G).

Next, we show that, under reasonable assumptions, tractability of finding the top-ranked match is not only necessary but also sufficient for enumerating all the answers in ranked order with polynomial delay. As a corollary, in the next section, we show that there are efficient evaluation algorithms for the other ranking functions presented earlier.

The following theorem shows that, for a ranking function ρ , the ρ -EVAL problem can be reduced to the ρ -TOP problem. This theorem can be proved by adapting the procedure of Lawler [18] (which is a generalization of the work by Yen [26, 27]) for computing the top- k solutions to discrete optimization problems. To obtain this result, we need to assume the following. First, each XML node has a unique *id* and the conditions attached to a node n of a DBTwig can include conjuncts of the form: $id(n) = i$ and $id(n) \neq i$ (note that these conjuncts can be computed in polynomial time in the size of the input). Second, the ranking of an answer is not changed as a result of adding conjuncts of the above form. More formally, we assume that if the DBTwig T' is obtained from T by adding conjuncts of this type, then $\rho(M, T, G) = \rho(M, T', G)$ for every match $M \in \mathcal{M}(T, G) \cap \mathcal{M}(T', G)$.

THEOREM 4.2. *The following are equivalent, under query-and-data complexity, for a ranking function ρ .*

- ρ -TOP can be solved in polynomial time.
- ρ -EVAL can be solved with polynomial delay.

4.1 Monotonic Ranking Functions

In this section, we present a family of ranking functions ρ , such that a top-ranked match of a DBTwig in an XML graph G can be computed efficiently. Thus, by Theorem 4.2, there are efficient evaluation algorithms for DBTwig queries that use these ranking functions. First, we need some notation.

Consider a DBTwig T and a node n of T . An n -branch B of T is a subtree of T that consists of n and all the descendants of

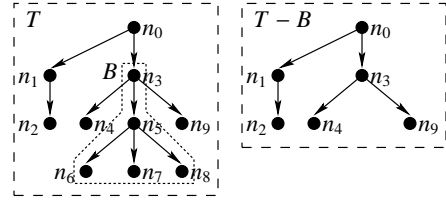


Figure 3: An example of a branch

one child of n . In other words, B is obtained from the subtree of T that is rooted at n by pruning all the children of n except one. We use $T - B$ to denote the DBTwig that is obtained from T by removing all the nodes of B , except for its root. As an example, the left part of Figure 3 shows a DBTwig T and one n_3 -branch B that is surrounded by a dotted polygon. The right side of this figure shows the DBTwig $T - B$.

Consider a DBTwig T , an XML graph G and a ranking function ρ . Let n be a node of T and B be an n -branch of T . Note that each of T , B and $T - B$ is a DBTwig. Also note that n is the only node that belongs to both B and $T - B$. Let M_{T-B} be a match of $T - B$ in G and M_B be a match of B in G , such that $M_{T-B}(n) = M_B(n)$. We use $M_B \oplus M_{T-B}$ to denote the match of T in G that is obtained by combining M_B and M_{T-B} as follows. If \hat{n} belongs to $T - B$, then $(M_B \oplus M_{T-B})(\hat{n}) = M_{T-B}(\hat{n})$; otherwise (i.e., \hat{n} belongs to B), $(M_B \oplus M_{T-B})(\hat{n}) = M_B(\hat{n})$.

Consider a DBTwig T and an XML graph G . We consider ranking functions ρ that are monotonic in the following sense. If, in a given match, we replace the mapping of one branch of T with a mapping that has a higher rank (over the given branch), then the ranking of the whole match can only improve. Formally, a ranking function ρ is *branch-monotonic* w.r.t. G and T if the following holds. For all nodes n of T and n -branches B of T , if $M_B^1, M_B^2 \in \mathcal{M}(B, G)$, $M' \in \mathcal{M}(T - B, G)$, $M_B^1(n) = M_B^2(n) = M'(n)$ and $\rho(M_B^1, B, G) \geq \rho(M_B^2, B, G)$, then $\rho(M_B^1 \oplus M', T, G) \geq \rho(M_B^2 \oplus M', T, G)$. We say that ρ is branch-monotonic if it is monotonic w.r.t. all XML graphs G and all DBTwig T . For example, the ranking functions $\rho_{\Sigma D}$, ρ_h and ρ_{fn} are branch-monotonic.

If ρ is branch-monotonic, then we can compute the top-ranked match in polynomial time in the size of Q and G . As a corollary to Theorem 4.2, we can also enumerate answers of DBTwig queries in ranked order with polynomial delay.

THEOREM 4.3. *Consider an XML graph G , a DBTwig query $Q = \langle T, p, \rho \rangle$ and a ranking function ρ that is branch-monotonic w.r.t. G and T . Then the following hold under query-and-data complexity.*

- The top-ranked match in $\mathcal{M}(T, G)$ can be found in polynomial time;
- $Q(G)$ can be enumerated in ranked order with polynomial delay.

COROLLARY 4.4. *If ρ is branch-monotonic, then ρ -EVAL can be solved with polynomial delay, under query-and-data complexity.*

5. CONCLUSION

When querying XML graphs, one has to deal with a huge number of answers that have varying degrees of semantic strength. The solution that we have presented incorporates several ideas. First, the nodes and edges of XML graphs have weights, but the user is able to do some fine tuning by overriding these weights. Second, the weights are used not just for ranking, but also for an a priori elimination of matches that map some pairs of adjacent nodes (from

the DBTwig) to XML nodes that do not satisfy the filtering conditions (i.e., distance bounds) of the corresponding edges. Third, for a wide range of ranking functions, the answers can be enumerated in ranked order with polynomial delay, under query-and-data complexity, even if projection is used and duplicates are eliminated.

The branch-monotonic ranking functions can combine a variety of measures. The weights introduce a database point of view, since they measure the semantic strength of connections. We can combine this measure with a function f (such as the one given toward the end in Section 3.2) that determines, for a node n of a DBTwig T , the relevance of the image $M(n)$. Note that f is a function of both n and $M(n)$ and, in particular, it can use some information attached to n (e.g., keywords) in order to calculate an IR score (e.g., by applying to $M(n)$ a formula based on tf/idf). The function f can also take into account a score based on a link (i.e., XLink or ID reference) analysis, similarly to PageRank [5]. Thus, the following branch-monotonic ranking function ρ can combine a variety of IR methods, which are based on the relevance of XML nodes to keywords (e.g., [14, 22]), with the notion of graph proximity (e.g., [4, 11, 15]) that is derived from the weights. Note that in the formula below, λ is a constant parameter that satisfies $0 < \lambda < 1$.

$$\rho(T, M, G) = \lambda \rho_{\Sigma d}(M, T, G) + (1 - \lambda) \rho_{\text{fn}}(M, T, G)$$

The expressiveness of DBTwigs can be easily extended while preserving our complexity results. In particular, it is possible to attach constraints of any type to edges, provided that these constraints (on pairs of XML nodes) can be computed in polynomial time in the size of the input. Thus, the constraints could be, for example, regular path expressions. We believe, however, that it is simpler and more flexible to specify paths (that match edges of the DBTwig) by using weight schemes rather than regular path expressions.

Our notion of a match is a natural generalization of the usual one. Namely, a match is a mapping from the nodes of a given DBTwig to XML nodes, such that the images of adjacent nodes (from the DBTwig) satisfy certain conditions. These conditions could refer to the paths connecting the XML nodes (e.g., the images are connected by a path with a weight of at most 5). Sometimes the user might want to see not just the match itself but also the *witness* paths, that is, paths showing that the conditions are indeed satisfied. Similarly, some ranking functions (e.g., $\rho_{\Sigma d}$) refer to paths and the user might want to see the witness paths that actually determine the ranking of the match. An answer is derived (by projection) from a match that satisfies the filtering conditions and has some ranking. However, the witness paths of the filtering are not necessarily the same as those of the ranking. In many cases, the witness paths of the ranking also satisfy the distance bounds. This is the case, for example, if the ranking function is of the form described above (and the same weight schemes are used for the filtering and ranking). But this is not always the case. An interesting research problem is to develop a notion of a match guaranteeing that the witness paths of the ranking always satisfy the filtering conditions.

Finally, we note that the work on keyword proximity search of [15, 16] deals with a different problem. There, a query is just a list of keywords and answers need not match any specific pattern.

6. REFERENCES

- [1] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: enabling keyword search over relational databases. In *SIGMOD*, 2002.
- [2] S. Amer-Yahia, S. Cho, and D. Srivastava. Tree pattern relaxation. In *EDBT*, 2002.
- [3] S. Amer-Yahia, N. Koudas, A. Marian, D. Srivastava, and D. Toman. Structure and content scoring for XML. In *VLDB*, 2005.
- [4] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *ICDE*, 2002.
- [5] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7), 1998.
- [6] N. Bruno, N. Koudas, and D. Srivastava. Holistic twig joins: optimal XML pattern matching. In *SIGMOD*, 2002.
- [7] M. J. Carey and D. Kossmann. On saying "enough already!" in SQL. In *SIGMOD*, pages 219–230, 1997.
- [8] M. J. Carey and D. Kossmann. Reducing the braking distance of an SQL query engine. In *VLDB*, pages 158–169, 1998.
- [9] S. Cassidy. Generalizing XPath for directed graphs. In *Extreme Markup Languages*, 2003.
- [10] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4), 2003.
- [11] V. Hristidis, Y. Papakonstantinou, and A. Balmin. Keyword proximity search on XML graphs. In *ICDE*, 2003.
- [12] D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27, March 1988.
- [13] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB*, 2005.
- [14] R. Kaushik, R. Krishnamurthy, J. F. Naughton, and R. Ramakrishnan. On the integration of structure indexes and inverted lists. In *SIGMOD*, 2004.
- [15] B. Kimelfeld and Y. Sagiv. Efficient engines for keyword proximity search. In *WebDB*, 2005.
- [16] B. Kimelfeld and Y. Sagiv. Finding and approximating top-k answers in keyword proximity search. In *PODS*, 2006.
- [17] B. Kimelfeld and Y. Sagiv. Incrementally computing ordered answers of acyclic conjunctive queries. In *NGITS*, 2006.
- [18] E. L. Lawler. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science*, 18, 1972.
- [19] W.-S. Li, K. S. Candan, Q. Vu, and D. Agrawal. Retrieving and organizing web pages by "information unit". In *WWW*, 2001.
- [20] A. Marian, S. Amer-Yahia, N. Koudas, and D. Srivastava. Adaptive processing of top-k queries in XML. In *ICDE*, 2005.
- [21] A. O. Mendelzon and P. T. Wood. Finding regular simple paths in graph databases. *SIAM J. Comput.*, 24(6), 1995.
- [22] M. Theobald, R. Schenkel, and G. Weikum. An efficient and versatile query engine for TopX search. In *VLDB*, 2005.
- [23] A. Trotman and B. Sigurbjörnsson. Narrowed Extended XPath I (NEXI). In *INEX*, pages 16–40, 2004.
- [24] Z. Vagena, M. M. Moro, and V. J. Tsotras. Twig query processing over graph-structured XML data. In *WebDB*, 2004.
- [25] M. Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, 1981.
- [26] J. Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17, 1971.
- [27] J. Y. Yen. Another algorithm for finding the k shortest loopless network paths. In "Proc. 41st Mtg. Operations Research Society of America", volume 20, 1972.