Approximate XML Query Answers

Neoklis Polyzotis
University of California, Santa Cruz
alkis@cs.ucsc.edu

Minos Garofalakis
Bell Labs, Lucent Technologies
minos@research.bell-labs.com

Yannis Ioannidis
University of Athens, Hellas
yannis@di.uoa.gr

ABSTRACT

The rapid adoption of XML as the standard for data representation and exchange foreshadows a massive increase in the amounts of XML data collected, maintained, and queried over the Internet or in large corporate datastores. Inevitably, this will result in the development of on-line decision support systems, where users and analysts interactively explore large XML data sets through a declarative query interface (e.g., XQuery or XSLT). Given the importance of remaining interactive, such on-line systems can employ approximate query answers as an effective mechanism for reducing response time and providing users with early feedback. This approach has been successfully used in relational systems and it becomes even more compelling in the XML world, where the evaluation of complex queries over massive tree-structured data is inherently more expensive.

In this paper, we initiate a study of approximate query answering techniques for large XML databases. Our approach is based on a novel, conceptually simple, yet very effective XML-summarization mechanism: TREESKETCH synopses. We demonstrate that, unlike earlier techniques focusing solely on selectivity estimation, our TREESKETCH synopses are much more effective in capturing the complete tree structure of the underlying XML database. We propose novel construction algorithms for building TREESKETCH summaries of limited size, and describe schemes for processing general XML twig queries over a concise TREESKETCH in order to produce very fast, approximate *tree-structured query answers*. To quantify the quality of such approximate answers, we propose a novel, intuitive error metric that captures the quality of the approximation in terms of both the overall structure of the XML tree and the distribution of document edges. Experimental results on real-life and synthetic data sets verify the effectiveness of our TREESKETCH synopses in producing fast, accurate approximate answers and demonstrate their benefits over previously proposed techniques that focus solely on selectivity estimation. In particular, TREESKETCHes yield faster, more accurate approximate answers and selectivity estimates, and are more efficient to construct. To the best of our knowledge, ours is the first work to address the timely problem of producing fast, approximate tree-structured answers for complex XML queries.

1. INTRODUCTION

Since its introduction six years ago, XML has evolved from a mark-up language for web documents to an emerging standard for data exchange and integration over the Internet. Being self-describing and hierarchical in nature, the XML data model is suitable for representing a diverse range of data sources and promises to enable the next-generation of search applications that will allow users to query effectively the information available on the Web.

SIGMOD 2004 June 13-18, 2004, Paris, France.

With the rapid growth of available XML data, one can expect a proliferation of on-line decision support systems that enable the interactive exploration of large-scale XML repositories. In a typical exploratory session, a domain expert poses successive queries in a declarative language, such as XQuery [4] or XSLT [7], and uses an appropriate visualization of the results in order to detect interesting patterns in the stored data. Obviously, the successful deployment of decision-support systems depends crucially on their ability to provide timely feedback to users' queries. This requirement, however, conflicts with the inherently expensive evaluation of XML queries which involve complex traversals of the data hierarchy, coupled with non-trivial predicates on the path structure and the value content.

Generating *approximate answers* is a cost-effective solution for offsetting the high evaluation cost of XML queries. In short, the system processes the query over a concise synopsis of the XML data and returns an approximation of the true result. Ideally, this approximate answer is computed very fast and is accurate in the sense that it preserves with low error the statistical traits of the true result. The user can then examine this "preview", assess the information content of the true answer, and decide whether it needs to be retrieved by executing the query over the base data. Overall, by providing the user with fast and accurate feedback on the form of the results, the system can reduce the number of queries that need to be evaluated in order to support effectively the data exploration task.

In a typical scenario, the result of an XML query is an XML fragment that is constructed by appropriate projections on the original data; an approximate answer, therefore, is an XML document that resembles the true answer in terms of hierarchical structure and value content. Clearly, the effectiveness of an approximate answering system hinges upon the existence of accurate synopsis structures that capture the key statistical characteristics of the base XML data and can thus produce low-error approximate answers to queries that project parts of it. Note that the problem of efficient XML summarization also arises in the context of selectivity estimation, where the synopsis is only used to estimate the *size* of the result. Approximating the *structure* of the result, however, is a strictly more complex problem since there are documents where the same query produces results of equal size but with *very different structure*. Summarizing, therefore, an XML document in order to compute approximate answers is more involved than building synopses for selectivity estimation, which in itself is known to be a hard problem [18].

Related Work[1]**.** Previous studies on approximate query answering [3, 10] have focused on the relational model, where the result

[1]Due to space constraints, a more detailed overview of related work can be found in the full version of this paper [17]

of a query is typically a multi-set of values. The key idea is to process a query over an appropriate relational synopsis (such as, histograms, wavelets, or sample-based summaries) and compute an approximation of the true value set. The proposed techniques and summarization methods, however, are suitable for flat relational data and are not easily extended to the case of general XML hierarchies.

As noted earlier, approximate XML query answering is closely tied to the problem of building effective XML synopses. Recent studies have looked at the related problem of summarizing XML data for estimating the selectivity of single XPath expressions [1, 12, 15, 16, 21, 22], or the number of binding tuples for twig queries [6, 9, 18]. Even though a selectivity estimate is essentially an approximate answer to an aggregate query (COUNT), the proposed summarization techniques do not store detailed enough information in order to approximate the *structure* of the query result.

Buneman et al. [2] have recently introduced a query-able compression scheme for tree-structured XML data. The proposed technique compresses the XML tree by using an appropriate bisimulation relation and evaluates an XPath query directly over the compressed instance. The goal, therefore, is to compute an *exact* answer to a path query, whereas our focus is on computing an *approximate* answer to a *twig query*, which typically involves the joint evaluation of multiple path expressions.

Our Contributions. In this paper, we initiate the study of approximate query answering for XML queries. In order to gain intuition on the complexity of the problem, this initial study focuses on approximate answers for twig queries with branching path expressions, i.e., we consider the structural part of the problem and ignore for now the value content of the document. As we show in this paper, even this constrained version is quite complex and requires non-trivial solutions. Our approach is based on a novel type of structural XML synopses, termed TREESKETCHes, that capture, in limited space, the key properties of the underlying path distribution and enable low-error approximate answers for a large class of interesting XML queries. We develop a systematic query evaluation framework for generating approximate answers over concise TREESKETCH synopses and describe an efficient construction algorithm for building an accurate TREESKETCH summary within the constraints of a limited space budget. Finally, we present experimental results on real-life and synthetic data sets that demonstrate the effectiveness of our approach and its benefits over previously proposed techniques, not only for generating approximate answers, but also for enabling accurate selectivity estimation. To the best of our knowledge, ours is the first study to look into the problem of computing approximate answers for complex XML queries. More concretely, the key contributions of our work can be summarized as follows:

• TREESKETCH **Summarization Model and Query Evaluation Framework.** Our TREESKETCH summarization model is based on the novel concept of *count-stability* which captures very effectively the intrinsic similarity of sub-structures in an XML document. Briefly, a TREESKETCH summary represents a clustering of document elements, where each cluster represents elements with similarly structured sub-trees. We develop an efficient evaluation algorithm that processes a query over a concise TREESKETCH and produces another TREESKETCH synopsis that summarizes the structure of the result. Futhermore, we discuss how the same algorithm can be used to estimate the result size of a complex twig query.

• **Efficient** TREESKETCH **Construction Algorithm.** We describe an efficient heuristic algorithm that starts from a detailed summary and incrementally merges element clusters that are "close" in terms of element sub-structure. To make our algorithm applicable on large data sets, we devise an effective heuristic that limits the number of possible merges in every step, without compromising the quality of the resulting synopsis.

• **New Distance Metric for XML Documents.** We argue that traditional graph-theoretic distance metrics, such as tree-edit distance, are not suitable for evaluating the quality of an approximate answer relative to the true result. To overcome this difficulty, we introduce a novel distance metric that quantifies the differences between two trees in terms of both the overall path structure and the distribution of document edges.

• **Experimental Study Verifying the Effectiveness of** TREESKETCH**es.** We validate our approach experimentally with an extensive study on real-life and synthetic data sets. Our results demonstrate that TREESKETCHes perform consistently better than previously proposed summarization techniques: they enable more accurate approximate answers and selectivity estimates, and at the same time are more efficient to construct. Moreover, our scaling experiments with large data sets show that even small-size TREESKETCHes are extremely effective in enabling low error selectivity estimates to complex twig queries (e.g., less than 5% estimation error for a 10KB summary of a 100MB input document). Combined with the affordable construction times of TREESKETCH summaries, these results indicate that TREESKETCHes constitute an effective and viable in practice solution for the structural summarization of large XML data sets.

2. BACKGROUND

XML Data Model. Following common practice, we model an XML document as a large, *node-labeled tree* $T(V, E)$. Each node $u \in V$ corresponds to an XML element and is characterized by a *unique object identifier (oid)* and a *label* (or, *tag*) assigned from some alphabet of string literals, that captures the element's semantics. Edges $(e_i, e_j) \in E$ are used to capture the containment of (sub)element e_j under e_i in the database. (We use $\texttt{label}(e_i)$, $\texttt{children}(e_i)$ to denote the label and set of child nodes for element node $e_i \in V$.) As an example, Figure 1 depicts a sample XML data tree containing bibliographical data. The document consists of `author` elements, each comprising a `name`, and several `paper` and `book` sub-elements. Each `paper` contains a `title`, a `year` of publication and one or more `keywords`, whereas a `book` just gives its `title`. Note that element nodes in the tree are named with the first letter of the element's tag plus a unique identifier. Leaf elements in T typically contain *values*, but our primary focus in this work is on approximately capturing and querying the *label structure* of an XML data tree, rather than the relevant value distributions.

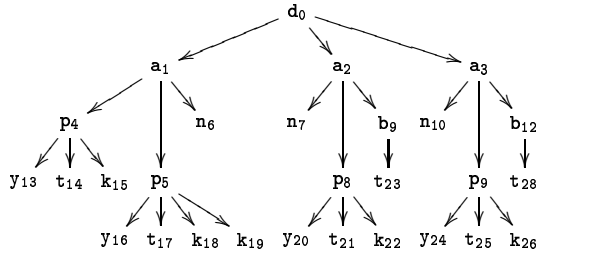


Figure 1: Example XML Document.

XML Query Model. We focus on *XML twig queries*, which represent the basic building block of declarative query languages for XML (including the XQuery [4] and XSLT [7] standards). Briefly, a twig query describes a complex traversal of the XML data tree and returns a *tree-structured* XML result constructed through the intertwined evaluation (i.e., structural join) of multiple path expressions (expressed in XPath [8]). Figure 2(a) depicts an example twig query over the document of Figure 1, where the q_i's denote variable names that are bound to specific data elements during query evaluation. We model a twig query Q as a node-labeled *query tree* T_Q, where (1) each node of T_Q is labeled with a variable name q_i in Q (with q_0 being a distinguished root node always bound to the XML document root); and, (2) each edge (q_i, q_j) of T_Q is annotated with an XPath expression $\texttt{path}(q_i, q_j)$ that describes the specific structural constraints specified in Q between the data elements bound to q_i and q_j during evaluation. For instance, the query tree corresponding to our example twig query above is shown in Figure 2(b). Following the generalized tree pattern notation [5], we use dashed edges to separate paths that are specified in the twig's **return** clause and can thus have empty results without nullifying the result of the query.

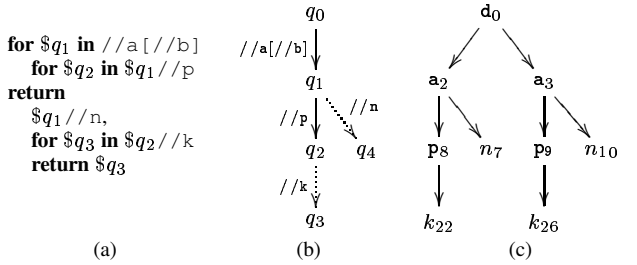


Figure 2: (a) Twig Query, (b) Query Tree, (c) Nesting Tree.

We consider twig queries using XPath expressions involving only the child and descendant-or-self axes (i.e., "/" and "//" operators) and may include existential branching predicates of the form "[$\bar{l}$]", where $\bar{l}$ is, in general, a label path whose existence is required under a given parent node in the XPath expression. As an example, the "//a[//b]" predicate in Figure 2 specifies author tree nodes that are located at any depth under the current binding of variable q_0 (the document root) and have *at least one* book descendant. Intuitively, the evaluation of a twig query Q proceeds by jointly evaluating all XPath expressions in Q over the XML tree, and generating the full set of *binding element tuples* for Q's variables. Each such binding tuple essentially specifies an assignment of element nodes to all the q_i query variables such that all structural constraints specified in the query's (q_i, q_j) edges are met. We will represent the binding tuples of a query Q with a *nesting tree* $N_T(Q)$, which contains all the elements of T that appear in the bindings of different variables and in addition preserves their ancestor/descendant relationships as specified by the query paths. Figure 2(c) shows the nesting tree for the example query of Figure 2(b). Obviously, the nesting tree can be used to reproduce the binding tuples of a query and ultimately its result.

3. TREESKETCH SYNOPSIS MODEL

3.1 General Graph-Synopsis Model

Abstractly, our general graph-synopsis model for an XML data tree $T(V, E)$ is defined by a *partitioning* of the element nodes in V (or, equivalently, by an *equivalence relation* $R \subseteq V \times V$) that respects element labels; that is, if $(e_i, e_j) \in R$ then $\texttt{label}(e_i) = \texttt{label}(e_j)$. The graph synopsis defined for T by such an equivalence relation R, denoted by $\mathcal{S}_R(T)$, can be represented as a *graph*, where: (1) each node v in $\mathcal{S}_R(T)$ corresponds to an equivalence class of R, i.e., a subset of (identically-labeled) data elements in T (termed the *extent* of v and denoted by $\texttt{extent}(v)$); and, (2) an edge (u, v) exists in $\mathcal{S}_R(T)$ if and only if some element node in $\texttt{extent}(u)$ has a child element in $\texttt{extent}(v)$. (We use $\texttt{label}(v)$ to denote the common label of all data elements in $\texttt{extent}(v)$.)

At a high level, several recently-proposed techniques for building path-index structures for XML (including 1-indexes [14] and A(k)-indexes [11]), as well as statistical summaries for XML databases (including XSKETCHes [15, 16] and twig-XSKETCHes [18]) are all based on the abstract "node-partitioning" idea described above. As an example, the basic twig-XSKETCH summary mechanism, which targets selectivity-estimation of complex twig queries, augments our general graph-synopsis model with (1) per-node count information that records the size of each synopsis node's extent, (2) localized per-edge stability information, indicating whether the synopsis edge is *backward- and/or forward-stable*, and (3) *edge distribution information*, that captures the distribution of child counts for the elements in the node's extent, across different stable ancestor or descendant edges. These localized edge distributions are maintained selectively on a per-node basis in the form of *edge histograms*, and essentially enable the computation of selectivity estimates for twig queries. For a simple example, consider a synopsis node u and two emanating synopsis edges $u \rightarrow v$ and $u \rightarrow w$. A two-dimensional edge histogram $H_u(c_1, c_2)$ would capture the fraction of data elements in $\texttt{extent}(u)$ that have exactly c_1 children in $\texttt{extent}(v)$ and c_2 children in $\texttt{extent}(w)$.

Limitations of Selectivity-Estimation Synopses. Given the amount of earlier work on XML summarization and the number of already-existing synopsis data structures for XML, a natural question that arises is whether there is a real need for a new summarization mechanism for approximate XML query answering. Our key observation here is that the focus of all earlier work in the area has been on the problem of *selectivity estimation* (for XPath expressions [15, 16] or twig queries [6, 18]) and, unfortunately, even the state-of-the-art solutions for XML selectivity estimates prove to be inadequate in accurately capturing the complete tree structure of the underlying document.

We illustrate our observation with a simple example on twig-XSKETCH synopses (we focus on the twig-XSKETCH model since it also uses a graph-synopsis and it is applicable in the general case of schema-less documents.) Consider the two XML document trees T_1 and T_2 shown in Figure 3(a,b). Both documents have the same set of distinct label-paths and differ only in the number of c children for the different b elements (the corresponding counts/multiplicities are shown along the edge). It is straightforward to verify that any twig query will have the same selectivity in either of the two documents and, in effect, both documents map to the same, *zero-error twig-*XSKETCH *synopsis*, shown in Figure 3(c), with the (exact) edge histograms for nodes A and B depicted in Figure 3(d). Consider, for instance, the twig query Q shown in Figure 3(e). Using the twig-XSKETCH and the methodology in [18], we can estimate its selectivity $sel(Q)$ with the expression $sel(Q) = |\texttt{extent}(A)| \cdot \sum_{b,c} H_A(b) \cdot H_B(c \mid b) \cdot b \cdot c$, which yields the same (accurate) estimate of 10 for both documents T_1 and T_2. Note, however, that the tree structure for the binding tuples of Q is in fact *very different* across our two example documents. For example, looking at the edge distribution in the query result, for document T_1, each A element appears in 5 binding tu-

ples, while for document T_2, one element (a_1) appears in 2 tuples and the other (a_2) appears in 8 tuples. This type of information is not captured by the twig-XSKETCH synopsis, since it does not affect the overall selectivity estimate.

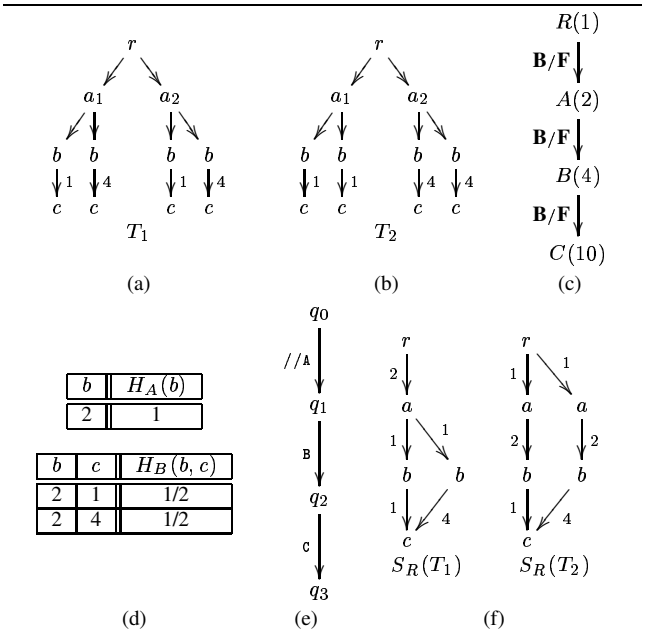


Figure 3: (a) Document T_1, (b) Document T_2, (c) Twig-XSKETCH, (d) Edge-histograms, (e) Twig query Q, (f) Count-Stable Synopses.

Again, the key observation here is that, while twig-XSKETCHes and edge histograms provide an accurate summarization mechanism for twig selectivity estimation, they cannot model the details of the tree structure for the twig query's binding tuples; thus, we expect them to be inadequate as a general-purpose approximate query answering tool (the results of our empirical study in Section 6 clearly verify our expectations.) Furthermore, as this paper demonstrates, our new synopses are also conceptually simpler, significantly easier to build, and provide more accurate results than twig-XSKETCHes *even for the simpler selectivity estimation problem*.

3.2 Count-Stability and the TREESKETCH Synopsis

Our proposed TREESKETCH synopsis data structure is a specific instantiation of the generic graph-synopsis model outlined earlier in this section. TREESKETCHes rely on a novel, intuitive concept of localized stability, termed *count stability*, defined formally as follows.

DEFINITION 3.1. *Let $R \subseteq V \times V$ denote a (label-respecting) equivalence relation over the nodes of $T(V, E)$, and let (u, v) denote a pair of equivalence classes (i.e., element-node partitions) induced by R. We say that the pair (u, v) is* k-stable *(where $k \geq 0$) if and only if each element $e \in u$ has exactly k child elements in v. The relation R and the graph synopsis $\mathcal{S}_R(T)$ resulting from the corresponding element partitioning are said to be* count stable *if and only if, for every possible pair of element partitions (u, v) there exists some $k \geq 0$ such that (u, v) is k-stable.* ∎

Note that the element partitions u, v in the above definition essentially correspond to the `extents` of synopsis nodes in $\mathcal{S}_R(T)$; furthermore, for k-stability, we treat the special case $k = 0$ as the absence of child elements (i.e., no synopsis edge between u and v). As an example, the count stable summaries for the XML trees of Figure 3(a,b) are shown in Figure 3(f), where the summary edges are annotated with the corresponding k. It is easy to see that our notion of count stability is a *refinement* of the traditional F-stability relation for trees employed by both XSKETCHes [15, 16] and twig-XSKETCHes [18]; in other words, the equivalence classes for the count-stability relation are generated by further partitioning the equivalence classes for F-stability.

Intuitively, our concept of count stability tries to define a class of equivalence relations where element nodes are grouped together only if the data sub-tree structures underneath them are identical. As the following lemma shows, the count-stable graph-synopsis for a data tree T is uniquely defined and, furthermore, it accurately captures the structure of T.

LEMMA 3.1. *Given a data tree $T(V, E)$, there exists a unique minimal (in terms of the number of equivalence classes) count-stable equivalence relation $R \subseteq V \times V$. Furthermore, there exists a function* Expand *from stable relations to XML trees, such that* $\mathrm{Expand}(R)$ *is isomorphic to the original document tree T.* ∎

Thus, the tree structure of the original document T can be retrieved *with zero-error* from a synopsis $S_R(T)$ if R is stable. The problem, of course, is that the size of a count-stable synopsis can become very large – it can easily be in the order of the original document size. Given the stringent time and storage limitations typically associated with interactive approximate query answering, it is clear that perfect count-stable summaries cannot be very useful as a data-approximation tool for real-time XML data exploration. Instead, our proposed TREESKETCH synopses try to *approximately* capture the underlying document-tree structure within a predefined space budget. Intuitively, the key idea behind TREESKETCHes is to *locally approximate* count-stable relations in the graph-synopsis wherever structural correlations exist in the underlying data, while relaxing the count-stability requirement where such correlations are not dominant and independence/uniformity assumptions are sufficient. Our TREESKETCH synopsis model is simply defined as follows.

DEFINITION 3.2. *A* TREESKETCH *synopsis $\mathcal{TS}$ for an XML data tree T is a node- and edge-labeled graph-synopsis for T, where: (1) each node u in $\mathcal{TS}$ stores an element count* $\mathtt{count}(u) = |\mathtt{extent}(u)|$*; and, (2) each edge (u, v) in $\mathcal{TS}$ stores an (average) child count* $\mathtt{count}(u, v)$ *equal to the* average number of children in $\mathtt{extent}(v)$ *for each element in* $\mathtt{extent}(u)$. ∎

Thus, instead of storing complex histograms for edge combinations in a B/F-stable neighborhood of a node (like twig-XSKETCHes [18]), our TREESKETCHes simply maintain a localized average child count for each edge in the synopsis (without requiring any stability properties for that edge). The interpretation of the stored average is simple: *all* elements in the extent of u have $\mathtt{count}(u, v)$ child elements in the extent of v. Obviously, this is trivially satisfied in a stable synopsis where each edge (u, v) is count stable for $k = \mathtt{count}(u, v)$.

There is an interesting and intuitive connection between TREESKETCHes and the clustering of points in multi-dimensional spaces. More specifically, let u be a synopsis node with outgoing edges $u \rightarrow v_1, \ldots, u \rightarrow v_n$. The set of outgoing edges defines a n-dimensional space, where an element $e \in u$ is mapped to a point $(c_1(e), \ldots, c_n(e))$ if it has $c_i(e)$ children to node v_i, $1 \leq i \leq n$. The recorded average edge counts essentially map all points in this

space to point $(\texttt{count}(u, v_1), \ldots, \texttt{count}(u, v_n))$, which actually represents the *centroid* of the cluster. We can thus characterize the quality of a TREESKETCH synopsis by using a metric that quantifies the quality of the induced clustering. The metric that we adopt in our work is the *squared error* of the clustering, which essentially measures the euclidean distance between points and their corresponding centroid. The squared error for a single cluster u is defined as $sq(u) = \sum_{e \in u} \sum_{1 \leq i \leq n} (c_i(e) - \texttt{count}(u, v_i))^2$, while the squared error $sq(\mathcal{TS})$ for a synopsis $\mathcal{TS}$ is simply the sum of squared errors for all the induced clusters. Note, of course, that the squared error for a count-stable synopsis is zero since all edge-count centroids are *exact*, i.e., the child counts for any element in a given synopsis-node `extent` are identical (and equal to the corresponding edge `counts`). We have chosen the squared error metric since it captures a notion of weighted variance, but it is possible to use other metrics such as the Manhattan distance or the pairwise intra-cluster distance. Irrespective of the actual choice, the existence of a workload-independent TREESKETCH-quality metric is a major difference from earlier summarization techniques which are also based on graph synopses, but quantify the quality of summaries on a per-workload basis (examples include both XSKETCHes and twig-XSKETCHes.) As we will see later, this feature will enable fast construction times, since the quality of a summary in the space of possible TREESKETCHes can be determined very efficiently, without requiring the costly evaluation of a query workload (as in the case of XSKETCH and twig-XSKETCH construction).

4. SYNOPSIS CONSTRUCTION AND QUERY PROCESSING

In this section, we start by describing novel, efficient bottom-up construction procedures for count-stable summaries and our TREESKETCH synopses (for a given space budget). We then introduce algorithms for approximating the results as well as the selectivities of XML twig queries over TREESKETCH synopses.

4.1 Building the Count-Stable Summary

Our algorithm for constructing the complete count-stable summary of an input XML tree T (termed BUILDSTABLE) is depicted in Figure 4. In a nutshell, BUILDSTABLE processes element nodes in a post-order traversal of T and constructs the count-stable synopsis graph $\mathcal{S}$ in a bottom-up fashion. A hash table $H[l, C]$ is used to maintain the collection of equivalence classes (i.e., synopsis nodes) built thus far, hashed on the (common) class label l and the identifying tuple of child counts C to other equivalence classes. The key observation here is that, by virtue of the post-order traversal, when visiting an element node e, its children in T have already been assigned to equivalence classes in $\mathcal{S}$; thus, the equivalence clas for e can be determined (with the help of $H[]$) based on its label and the classes and counts of its children (Step 3). If a class for e does not already exist, a new class/synopsis node is created and the appropriate edges and `counts` are added to $\mathcal{S}$ (Steps 4–8). Finally, e is added to the extent of the corresponding synopsis node (Step 9).

Algorithm BUILDSTABLE constructs the count-stable summary of an XML tree in linear $O(|T|)$ time; note that, for building the "child-count signature" in Step 3, only the element's child classes are necessary, and these can be easily accessed using a stack during the post-order traversal.

4.2 Building TREESKETCH Synopses

As already mentioned in Section 3.2, the size of an exact count-stable synopsis typically renders it useless in the context of a real-life approximate query processing system. Such systems usually place tight limits on the space budget for building synopses of the underlying data collection. Thus, there is a clear need for effectively constructing *compressed* TREESKETCH synopses under a given space budget, while maintaining a high-quality XML-data approximation in order to enable meaningful approximate answers.

Procedure BUILDSTABLE(T)
Input: XML Document T.
Output: Count-Stable synopsis $\mathcal{S}$ of T.
begin
1. $H := \phi$; $\mathcal{S} := \phi$
2. **for each** element $e \in T$ in post-order **do**
3. $C := \{(u_i, c_i) : u_i$ is a node in $\mathcal{S}$ and $|\texttt{children}(e) \cap \texttt{extent}(u_i)| = c_i > 0\}$
4. **if** $(H[\texttt{label}(e), C] = \phi)$ **then**
5. Add node u to $\mathcal{S}$ with $\texttt{label}(u) = \texttt{label}(e)$
6. $H[\texttt{label}(e), C] := u$
7. **for** $(u_i, c_i) \in C$ **do** add edge $u \xrightarrow{c_i} u_i$ to $\mathcal{S}$
8. **endif**
9. $u := H[\texttt{label}(e), C]$; $\texttt{extent}(u) := \texttt{extent}(u) \cup \{e\}$
10. **endfor**
end

Figure 4: Algorithm BUILDSTABLE.

Given the aforementioned natural analogy between TREESKETCHes and data clustering (Section 3.2), our goal of constructing an effective synopsis can be translated to computing an effective clustering of the XML elements. Here, of course, an element cluster is "tight" if it encompasses data elements with similar sub-trees, and "tightness" can be quantified using the *squared error* for the clustering (as discussed in Section 3.2). Thus, our goal is to build a TREESKETCH synopsis $\mathcal{TS}$ that fits within a given space budget, such that the overall square error $sq(\mathcal{TS})$ for the synopsis is *minimized*. The analogy with clustering also highlights the difficulty of TREESKETCH construction, since such clustering problems are known to be $\mathcal{NP}$-hard even in the simple case of points in a low-dimensional space [19, 23]. Furthermore, TREESKETCH construction typically deals with a high-dimensional space which is defined by the clustering itself (i.e., the space itself changes as elements are assigned to clusters)! Thus, the problem is significantly more complex and existing clustering algorithms are not directly applicable.

Our approach is based on a generic *bottom-up* clustering paradigm: starting from the count-stable synopsis, our algorithm (termed TSBUILD) incrementally reduces the synopsis size by merging nodes with similar sub-structures, until the budget constraint is met. This resembles agglomerative hierarchical clustering algorithms, which start with one cluster per input data point and successively reduce the number of clusters by merging neighboring groups (according to some appropriate distance metric). Another possible option is a top-down approach that starts from a coarse summary and gradually expands it by splitting nodes (this is actually the approach taken in the XSKETCH work [15, 16, 18]). In the clustering literature, however, bottom-up algorithms have been shown to perform better than their top-down counterparts; in addition, we have experimentally verified that bottom-up TREESKETCH construction yields much better results, without significantly increasing construction time.

The TSBUILD Algorithm. We now describe our TREESKETCH-construction algorithm in more detail. In a nutshell, TSBUILD maintains a pool of *candidate operations* to be applied to the working TREESKETCH synopsis $\mathcal{TS}$ (initialized to the count-stable graph), where each operation m in the pool merges two nodes of $\mathcal{TS}$ (de-

Procedure TSBUILD(D,S,U_h, L_h)
Input: XML document D; space budget S; upper/lower bounds for heap size (U_h, L_h)
Output: TREESKETCH synopsis $\mathcal{TS}$ of T of size $\leq S$
begin
1. $\mathcal{TS}$:= BUILDSTABLE(T) *// Start with the count-stable summary*
2. $h \leftarrow \phi$
3. **while** ($\texttt{size}(\mathcal{TS}) > S$) **do**
4. h := CREATEPOOL ($\mathcal{TS}$, U_h)
5. **while** ($\texttt{size}(\mathcal{TS}) > S$ **and** $\texttt{size}(h) > L_h$) **do**
6. $m \leftarrow h.popMin()$
7. $\mathcal{TS} := m(\mathcal{TS})$ *// Apply m on $\mathcal{TS}$*
8. Let u_m be the new synopsis node
9. **for each** $m' \in h$ **do**
10. **if** ($m'.nodes \cap m.nodes \neq \phi$) **then**
11. Remove m' from h
12. Add a merge between $m'.nodes - m.nodes$ and u_m to h
13. **endif**
14. Recompute $m'.err_d, m'.size_d$ for all $m' \in$ *affected*(h, m)
15. **endwhile**
16. **endwhile**
17. **return** $\mathcal{TS}$
end

Figure 5: Algorithm TSBUILD**.**

noted by $m.nodes$). If $m(\mathcal{TS})$ denotes the resulting synopsis after applying the merge m on $\mathcal{TS}$, we define $m.err_d = sq(m(\mathcal{TS})) - sq(\mathcal{TS})$ to be the increase in squared error from $\mathcal{TS}$ to $m(\mathcal{TS})$, and $m.size_d = \texttt{size}(\mathcal{TS}) - \texttt{size}(m(\mathcal{TS}))$ to be the corresponding decrease in synopsis size. The pool of candidate operations is organized in a *min-heap* according to the *marginal-gain* ratio $m.err_d/m.size_d$, i.e., the operation at the top of the heap offers the least increase in squared error per unit of space that is saved. At each step of the construction algorithm, the operation at the top of the heap is applied, the pool is updated with new merge operations for the new node, and the err_d, $size_d$ metrics are recomputed for the new pool of candidate merge operations. This process is repeated until the heap is exhausted (i.e., no merge operations are possible) or the size of the $\mathcal{TS}$ synopsis drops below the allotted space budget.

The pseudo-code for our TSBUILD algorithm is shown in Figure 5. TSBUILD initializes the min-heap h of candidate merge operations through function CREATEPOOL (discussed below), and then applies successive merges according to our marginal-gain criterion (Steps 5-15). In order to limit the memory requirements of the algorithm and increase efficiency, the size of the operations heap is bounded by the supplied parameter U_h. As operations are performed, the size of the heap is gradually reduced and when it drops below a supplied threshold L_h, the heap is re-generated and the process repeated.

A potential performance bottleneck for the construction process is the re-computation of the err_d and $size_d$ metrics for the merge operations in the heap. To make this more efficient, our TSBUILD algorithm employs two key techniques. First, re-computation is performed only for a *limited subset* of the candidate merge operations. The key observation here is that the err_d and $size_d$ metrics measure *differences* in the characteristics of the synopsis (rather than absolute quantities) and, thus, most of them can be preserved across merges. More specifically, if m is the merge that was performed last and u_m is the newly created node, then TSBUILD only needs to compute the metrics for operations that merge parent or child nodes of u_m (we denote this set of operations as *affected*(m)); for the remaining operations, the err_d, $size_d$ metrics remain unchanged.

Procedure CREATEPOOL($\mathcal{TS}$, U_h)
Input: Synopsis $\mathcal{TS}$; heap-size upper bound U_h.
Output: Double-ended heap h containing $\leq U_h$ merge operations.
begin
1. $h \leftarrow \phi, level := 0$
2. **while** ($level < \texttt{height}(\mathcal{TS})$ **and** $\texttt{size}(h) < U_h$) **do**
3. $level := level + 1$
4. **for all** $u, v \in \mathcal{TS} : \texttt{label}(u) = \texttt{label}(v)$ **do**
5. **if** ($\max\{depth(u), depth(v)\} = level$) **then**
6. Let m be the operation that merges u,v
7. $h.push(m)$
8. **if** ($\texttt{size}(h) > U_h$) **then** $h.popMax()$
9. **endif**
10. **endfor**
11. **endwhile**
12. **return** h
end

Figure 6: Algorithm CREATEPOOL**.**

Our second technique makes the computation of err_d more efficient by storing *"sufficient" statistics* in each synopsis node. Briefly, each node stores the sum and the sum of squares for the child counts of its elements along each outgoing edge in the synopsis. It is straightforward to show that these statistics are sufficient in order to compute the squared-error metric for the synopsis $sq(\mathcal{TS})$ without accessing the base data. In addition, in certain cases, these statistics can be combined in order to derive the statistics of new nodes (created through merge operations). The complete details are beyond the scope of this presentation and can be found in the full paper [17]. Note that this idea is similar to the one proposed in the BIRCH clustering algorithm [23], where clusters are represented only by a collection of similar sufficient statistics throughout the computation. In our case, however, the stored statistics do not obviate the need to access a small subset of the base data (although this can be done very efficiently, by accessing only the relevant parts of the count-stable summary). Again, we defer the details to the full version of this paper [17].

Generation of Candidate Operations. We now discuss the details of our CREATEPOOL algorithm for initializing a heap of at most U_h merge operations. An obvious approach would be to generate all possible pair-wise merges and keep the top U_h operations according to our ratio metric $err_d/size_d$. Unfortunately, such a solution requires evaluating $O(N^2)$ merge operations, where N is the number of nodes in the count-stable summary and, thus, becomes prohibitively expensive as the size and complexity of the data grows. Given that CREATEPOOL is invoked repeatedly during the TREESKETCH-construction process, this increased complexity has a significant negative impact on construction times. On the other hand, reducing the number of operations considered increases the efficiency of the candidate-generation stage, but it also runs the risk of "polluting" the heap with less effective merge operations that can affect the quality of the generated TREESKETCHes.

To overcome this difficult problem, we adopt a heuristic that limits the number of merge operations considered while ensuring that the heap only contains operations that are likely to be beneficial. The key observation here is that a merge of two nodes u and v leads to a "good" clustering of the elements involved only if u and v have similarly structured sub-trees. Thus, our TREESKETCH-construction algorithm is much more likely to apply merge operations on the children of u and v *first*, before merging u and v themselves. This observation suggests a bottom-up approach for populating the heap with merge operations, starting with nodes close

to the leaves of the current synopsis and proceeding upward to the root.

Figure 6 shows the pseudo-code for our CREATEPOOL algorithm that implements the aforementioned heuristic. CREATEPOOL uses the concept of a node's *depth* in order to examine merge operations in a bottom-up fashion. More specifically, let e be a document element. The depth of e is defined as 0 if e is a leaf, and $1+\max\{depth(e_c)\}$ otherwise, where the maximum is taken over all $e_c \in \texttt{children}(e)$. Intuitively, the depth of an element is the longest path that leads to a leaf descendant. The depth of a *synopsis node* u is defined as $\max_{e \in u}\{depth(e)\}$. CREATEPOOL evaluates merge operations at increasing depths in the current synopsis $\mathcal{TS}$ and only records the best U_h of the operations seen thus far (this can be implemented efficiently through a double-ended heap). Candidate generation terminates when the current depth has been exhausted and the heap holds the maximum allowed number of operations.

4.3 Approximate Query Processing

We now turn our attention to the problem of generating approximate answers from a concise TREESKETCH synopsis. At an abstract level, our query evaluation algorithm, termed EVALQUERY, processes the input query Q over an input TREESKETCH $\mathcal{TS}$ and produces an output TREESKETCH $\mathcal{TS}_Q$ that summarizes the nesting tree $N_T(Q)$ (the full nesting tree can be retrieved by expanding $\mathcal{TS}_Q$). As noted in Section 2, the full nesting tree can be used to reconstruct the binding tuples of Q and ultimately its result. The evaluation algorithm uses the structure information of $\mathcal{TS}$ in order to identify matches of the query's path expressions, while the stored edge counts are used to approximate the cardinalities of the corresponding result sets. Similar to any summarization method, the use of the stored information is coupled with a set of appropriate statistical assumptions that compensate the lack of detailed distribution information at certain parts of the synopsis. As we will see, the validity of these assumptions depends on the quality of element clustering within each synopsis node and is thus directly linked to the heuristics of the TSBUILD algorithm. Intuitively, this direct relationship between the build algorithm and the query processing framework leads to the construction of summaries that compute highly accurate approximate answers.

Figure 7 shows the pseudo-code for algorithm EVALQUERY. The algorithm processes query Q over the input synopsis $\mathcal{TS}$ and incrementally builds the result TREESKETCH $\mathcal{TS}_Q$. Each node $u_Q \in \mathcal{TS}_Q$ corresponds to a set of elements with tag $\texttt{label}(u_Q)$, which come from the extent of a node $u \in \mathcal{TS}$ and will appear in the bindings of query variable $q \in Q$. We will use the notation $u_Q(u, q)$ to denote this association and the shorthand $u_Q(q)$ when no confusion arises. In addition, $bind[q]$ will denote the set of nodes in $\mathcal{TS}_Q$ that contain bindings for q.

Initially, the approximate TREESKETCH contains a root node $r_Q(\texttt{root}(\mathcal{TS}), q_0)$ which specifies the binding of the topmost variable q_0 to the root of the document. The algorithm processes the query nodes in a pre-order traversal and, for each node q, evaluates the path expressions to the children of q, relative to the computed bindings in $bind[q]$. More specifically, for each child q_c and binding $u_Q(u, q) \in bind[q]$, the algorithm computes a list of bindings $B(q_c, u_Q) =< (v, k) >$ for variable q_c (lines 4-9), where $v \in \mathcal{TS}$ and $k > 0$ is a descendant count. Essentially, each $(v, k) \in B(q_c, u_Q)$ specifies that every element in u_Q (the current binding for q) has exactly k descendants in v along path $\texttt{path}(q, q_c)$. The new bindings are recorded with the insertion of a node $v_Q(v, q_c)$ and an edge $u_Q \rightarrow v_Q$. Since an element in u_Q can have descendants in the same node v through multiple paths

procedure EVALQUERY($\mathcal{TS}$,Q)
Input: TREESKETCH $\mathcal{TS}$ of document T; twig query Q
Output: TREESKETCH $\mathcal{TS}_Q$ that approximates the nesting tree $N_T(Q)$
begin
1. Initialize $\mathcal{TS}_Q$ with root $r_Q(\texttt{root}(\mathcal{TS}), q_0)$
2. **for each** $q \in Q$ in a pre-order traversal **do**
3. **for each** $u_Q(u, q) \in bind[q], q_c \in \texttt{children}(q)$ **do**
4. Let p_m be the main path of $\texttt{path}(q, q_c)$.
5. $E := \{e_i \mid e_i = u/\ldots/v_l$ is an embedding of $p_m\}$
6. **for all** $e_i = u/\ldots/v_l \in E$ **do**
7. $k_l :=$ EVALEMBED(p, e_i); $B(q_c, u_Q) \leftarrow (v_l, k_l)$
8. **done**
9. **for** (v, k) **in** $B(q_c, u_Q)$ **do**
10. Add node $v_Q(v, q_c)$ to $\mathcal{TS}_Q$ if it does not exist
11. Add edge $u_Q \rightarrow v_Q$ to $\mathcal{TS}_Q$ if it does not exist
12. $\texttt{count}(u_Q, v_Q) += k$
13. **done**
14. **done**
15. **if** $(\exists q_c \in children[q] : bind[q_c] = \emptyset)$ **then**
16. **return** $\emptyset$ // *The answer is empty*
17. **done**
18. **return** $\mathcal{TS}_Q$
end

Figure 7: Algorithm EVALQUERY.

in the synopsis, all counts k that correspond to the same v are aggregated in $\texttt{count}(u_Q, v_Q)$ (line 12). Note that the algorithm inserts exactly one node $u_Q(u, q)$ for each pair (u, q), thus forming a graph-structured summary $\mathcal{TS}_Q$. This optimization, which guarantees a worst case size of $O(|\mathcal{TS}| \cdot |Q|)$ for the intermediate result synopsis, stems from the interpretation of the TREESKETCH summarization model: all elements in u contain identically structured sub-trees and thus need to be represented only once in the synopsis (regardless of their ancestor nodes.) The query node q is included in the association in order to correctly handle the case where elements of the same node appear in the bindings of different query nodes. In order to compute the set of bindings $B(q_c, u_Q)$ for variable q_c, the algorithm first identifies the synopsis paths that possibly contain descendants of u_Q along $\texttt{path}(q, q_c)$, and the number of descendants along each path is computed with algorithm EVALEMBED. The separate invocations of EVALEMBED essentially apply an independence assumption between the different variables of the query, which translates to an independence assumption on the underlying path distribution. We defer this point to the end of the section, where we discuss the relationship of the processing assumptions to the general TREESKETCH framework.

The pseudo-code for algorithm EVALEMBED is shown in Figure 8. The final descendant count is computed as the number of descendants n_t along the main path of the embedding, scaled by the selectivity factors of the branch embeddings. The count n_t is estimated simply as the product of the corresponding edge counts, using the assumption that every element in source node u_i has $\texttt{count}(u_i, u_{i+1})$ children to target node u_{i+1} (this is the basic interpretation of the TREESKETCH model.) To estimate the selectivity s_i of branching predicate $\bar{\texttt{l}}_i$, the algorithm calls itself recursively to compute the number of descendants for each element of node u_i (the source of the branch) along the different embeddings of $\bar{\texttt{l}}_i$. If there exists at least one embedding such that the descendant count is ≥ 1, then all elements in u_i satisfy the branching predicate and the selectivity is equal to 1. In the opposite case (all descendant counts are strictly less than 1), each count is treated as the fraction of elements in u that have descendants along the corresponding embedding of the branching predicate. Since an element satisfies the branching predicate if it is the root of *at least one*

Procedure EVALEMBED(p,e)
Input: XPath $p = \mathtt{l}_1[\bar{\mathtt{l}}_1]/\ldots/\mathtt{l}_n[\bar{\mathtt{l}}_n]$; synopsis path $e = u_0/u_1/\ldots/u_n$, where $u_1/\ldots/u_n$ is an embedding of $\mathtt{l}_1/\mathtt{l}_2/\ldots/\mathtt{l}_n$
Output: Estimated number of descendants for each element of u along p.
begin
1. $n_t := \prod_{0<i\leq n} \mathtt{count}(u_{i-1}, u_i)$ *// Descendants along main path*
2. **for each** $\bar{\mathtt{l}}_i \in p$ **do** *// Compute the selectivity of branches*
3. $E_i := \{e_j \mid e_j = u_i/v_1/\ldots/v_l$ is an embedding of $\bar{\mathtt{l}}_i\}$
4. **for all** $v_l : \{e_j \mid e_j = u_i/\ldots/v_l \in E_i\} \neq \emptyset$ **do**
5. $k_l := \sum_{e_j \in E_i}$ EVALEMBED$(\bar{\mathtt{l}}_i, e_j)$
6. $N_i \leftarrow k_l$
7. **done**
8. **if** $\exists k_j \in N_i : k_j \geq 1$ **then**
9. $s_i := 1$
10. **else**
11. $s_i := \left(\sum_{k_j \in N_i} k_j - \sum_{k_j,k_l \in N_i, j\neq l} (k_j \cdot k_l) + \sum_{k_j,k_l,k_m, j\neq l\neq m} (k_j \cdot k_l \cdot k_m) - \cdots \right)$ *// inclusion-exclusion*
12. **endif**
13. **done**
14. **return** $n_t \cdot \prod_{1\leq i\leq n} s_i$
end

Figure 8: Algorithm EVALEMBED**.**

matching embedding, the overall selectivity is computed using the inclusion-exclusion principle on the recorded fractions (line 11). We note that the application of the exclusion/inclusion principle essentially makes use of an independence assumption on the distribution of document edges, which, as we discuss below, is derived from the interpretation of the TREESKETCH summarization model and is closely related to the squared error of the synopsis.

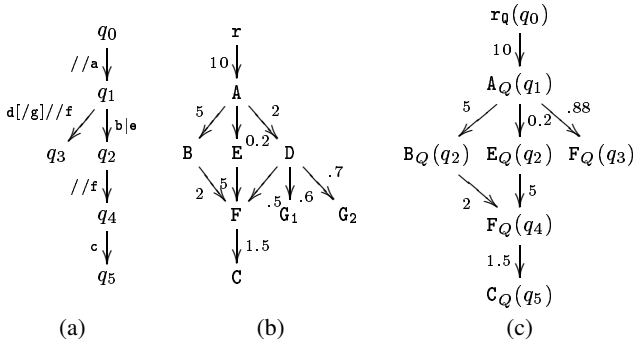


Figure 9: (a) Query Q, (b) TREESKETCH $\mathcal{TS}$ **(c) Result** TREESKETCH $\mathcal{TS}_Q$**.**

EXAMPLE 4.1. *Consider the invocation of* EVALQUERY *on the query Q and synopsis $\mathcal{TS}$ shown in Figure 4.3. Initially, the result synopsis contains a root $r_Q(r, q_0)$ only and $bind[q_0] = r_Q$. On the first iteration of* EVALQUERY*, variable q_0 is processed and the bindings of child variable q_1 are computed. In this case, it is easy to verify that each element in r_Q has 10 descendants along path* `//a` *to node* `A`*. As a result, node $\mathtt{A}_Q(\mathtt{A}, q_1)$ is inserted in $\mathcal{TS}_Q$ along with edge $\mathtt{r}_Q \rightarrow \mathtt{A}_Q$, and $\mathtt{count}(\mathtt{r}_A, \mathtt{A}_Q) = 10$.*

Let us consider now the processing of q_1, and more specifically, the computation of bindings from q_1 to q_3. Starting from node `A`*, which appears in the bindings of q_1, we can identify exactly one simple embedding of $\mathtt{path}(q_1, q_3) = \mathtt{d[/g]//f}$, namely $e =$* `A/D/F`*. The bindings of q_3, therefore, will be the descendants of* `A` *along the given embedding. To compute the number of descendants for each element in* `A` *(algorithm* EVALEMBED*), we first observe that $n_t = \mathtt{count}(\mathtt{A}, \mathtt{D}) \cdot \mathtt{count}(\mathtt{D}, \mathtt{F}) = 2 \cdot 0.5 = 1$. This count needs to be scaled by the selectivity of the branching predicate* `[/g]`*, for which there exist two embeddings:* $\mathtt{G}_1$*, with descendant count 0.6, and* $\mathtt{G}_2$*, with descendant count 0.7. Essentially, 60% of elements in* `D` *have a branching embedding along* $\mathtt{G}_1$ *and 70% have a branching embedding along* $\mathtt{G}_2$*. The overall branch selectivity is computed as $s = 0.6 + 0.7 - 0.6 \cdot 0.7 = 0.88$. Thus, the number of descendants along* `d[/g]//f` *for each binding in q_1 is $1 \cdot 0.88$ and $\mathcal{TS}_Q$ is updated accordingly. The final result synopsis $\mathcal{TS}_Q$ is shown in Figure 4.3(c) (synopsis nodes are annotated with the corresponding query node).* ∎

As noted previously, the evaluation algorithm applies a set of independence assumptions during the processing of an input query over a concise TREESKETCH summary. At a closer inspection, all the processing assumptions can be reduced to a basic independence assumption that de-correlates the distribution of document edges along different paths of the document. This assumption is essentially derived from the interpretation of the TREESKETCH synopsis model: given a synopsis edge $u \rightarrow v$, all elements in u have $\mathtt{count}(u, v)$ children in v, independent of incoming or outgoing paths (Section 3). Obviously, this interpretation is trivially satisfied on a stable synopsis where, by virtue of count-stability, all elements in the extent of a node u have the same edge counts to child nodes. As a result, EVALQUERY will compute the exact nesting tree of a query when the accessed edges of the synopsis are count-stable. In the general case of a compressed TREESKETCH, it is straightforward to observe that the validity of the assumption is directly related to the error of the induced element clustering: if the error is low, i.e., the clusters are tight, then the elements are closer to the centroid (which is defined by the recorded average edge counts), and the assumption becomes more valid. In essence, there is a close relationship between the squared error of the synopsis, which quantifies the tightness of the clusters, and the quality of the generated approximate answers. This observation provides the "missing link" between the construction algorithm and the evaluation framework: although the build process does not use a workload-based approach to ensure high-quality approximate answers, it achieves the same goal by keeping the squared error low and thus making the basic independence assumption more valid.

4.4 Selectivity Estimation

In this section we briefly discuss the use of TREESKETCHes for estimating the selectivity of twig queries. As shown in earlier studies [5, 13], accurate estimation for the number of bindings tuples for twig queries is a key requirement in producing effective query plans for complex declarative queries over XML data.

Our proposed estimation framework uses the result of the EVALQUERY algorithm to efficiently compute an estimate of the query's selectivity. More specifically, the estimation algorithm performs a single post-order traversal of the structural summary $\mathcal{TS}_Q$ and computes, for each node, the average number of binding tuples per element in its extent. Given the bounded size of $\mathcal{TS}_Q$, it becomes clear that the estimation process has low memory requirements and can be performed very efficiently. In the interest of space, we do not discuss the estimation algorithm further. The full details can be found in the full version of this paper.

5. AN ERROR METRIC FOR APPROXIMATE XML QUERY ANSWERS

In order to evaluate the effectiveness of the proposed approximate query answering framework, it is necessary to measure the degree of similarity between the approximate nesting tree $N_{\mathcal{TS}}(Q)$ that is computed over a concise synopsis $\mathcal{TS}$, and the true nesting tree $N_T(Q)$ of the query. More formally, this translates to computing a distance $dist_A(N_{\mathcal{TS}}(Q), N_T(Q))$ between the two XML trees which essentially quantifies the error of approximation. There are numerous proposals for distance metrics over trees, the most widely used being the *tree-edit distance metric* [20]. As we will see next, however, the proposed metrics essentially measure the *syntactic* differences between the two XML trees and thus fail to capture the semantics of approximate answers. We note that our discussion will focus on the tree-edit distance metric, but our observations hold for other graph-theoretic metrics as well.

The tree-edit distance $dist_E(T_1, T_2)$ between two XML trees T_1 and T_2 measures the minimum cost sequence of edit operations that transform T_1 to T_2 (or vise versa). The basic edit operations include adding, deleting, or relabeling a tree node, while more complex operations (such as copying whole sub-trees) are usually modelled as a composition of simple operations. Consider, for instance, the example of Figure 10, where S_c and S_d denote sub-trees of sizes $|S_c|$ and $|S_d|$ respectively and numbers along edges denote child cardinalities. We will assume that T is the true nesting tree of the query, and T_1, T_2 are two possible approximations. If we limit the edit operations to node insertion and deletion, and assuming that each operation has unit cost, it is straightforward to show that $dist_E(T, T_1) = 3 \cdot |S_c| + 3 \cdot |S_d|$ (essentially, we have to add 3 S_c sub-trees to the left a element of T_1 and delete 3 S_b sub-trees from the right a element in order to transform T_1 to T). Similarly, $dist_E(T, T_2) = 3 \cdot |S_c| + 3 \cdot |S_d|$. According to tree-edit distance, therefore, T_1 and T_2 are equally good approximations of the true result. Intuitively, however, we expect T_2 to be a better approximation since it maintains the correlation between the number of S_c and S_d subtrees under the same parent (few S_c are combined with several S_d and vise versa); answer T_1, on the other hand, conveys exactly the opposite trait, that there is an equal number of S_c and S_d sub-trees under every a.

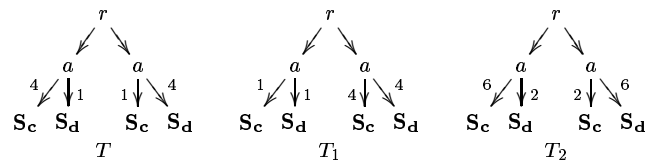


Figure 10: Query answer T and two approximations T_1, T_2

The previous example illustrates that the syntactic difference between two documents, as measured by tree-edit distance or other similar graph-theoretic metrics, is not a suitable similarity metric for approximate answers. Intuitively, an approximate answer is useful if it preserves the statistical traits of the true answer, without necessarily being identical to it, and the distance metric should capture this type of "approximate" similarity. Similar observations have been made in the context of approximate answers for relational queries [3, 10], where the result of a query is a multi-set of values. In short, these studies have argued convincingly that set-theoretic metrics, which correspond to syntax-oriented metrics in the XML world, do not yield intuitive results when comparing two value sets (the approximate and the true answer). This has led to the introduction of new distance metrics, such as the MAC [10] and the EMD [3], in order to measure effectively the quality of approximate answers to relational queries.

A New Distance Metric for XML Trees. We introduce a novel distance metric, termed *Element Simulation Distance (ESD)*, that avoids the shortcomings of syntax-oriented metrics by capturing regions of *approximate similarity* between the compared XML trees. To the best of our knowledge, ours is the first metric that considers both the overall path structure *and* the distribution of document edges, when computing the distance between two XML trees.

We now describe the ESD metric in more detail. Let $u \in T_1$ and $v \in T_2$ be elements of the compared trees with $\texttt{label}(u) = \texttt{label}(v)$. We wish to define a function $ESD(u, v)$ that measures the degree of "simulation", or sub-tree similarity, between the two elements. Let U_t and V_t denote the children of u and v respectively that have tag t. If we treat U_t, V_t as two sets of "values", where the distance between any two elements $u' \in U_t, v' \in V_t$ can be measured as $ESD(u', v')$ (i.e., a recursive application of the metric to the children of u, v), then we can measure the distance $dist_S(U_t, V_t)$ between U_t, V_t by using any existing value-set distance metric, like MAC [10] or EMD [3]. The result is an indication of how well u's children of tag t simulate v's children of the same tag. The ESD distance between u and v can now be measured as the sum of distances for children of matching tags: $ESD(u, v) = \sum_t dist_S(U_t, V_t)$. In effect, two elements are more (or less) similar if their children with matching tags are more (or less) similar themselves, which recursively extends to the whole sub-structure underneath the two elements. In the case where one of U_t, V_t is empty, we apply a straightforward transformation so that the computation of $dist_S(U_t, V_t)$ is well defined. More concretely, assume without loss of generality that $V_t = \emptyset$. For each element $e \in U_t$, we insert a unique (artificial) element e_v in V_t with distance $ESD(e, e_v) = |e|$, where $|e|$ is the sub-tree size of e, and $ESD(e', e_v) = \infty, \forall e' \in U_t, e' \neq e$. This transformation essentially models the insertion of the missing sub-trees under v and allows the set-distance metric to be computed on the new non-empty set V_t.

EXAMPLE 5.1. *Consider the example of Figure 10 and let u and v be the left a elements of T and T_1 respectively. Elements u, v have children of tags c and d (the roots of sub-trees S_c and S_d) and thus $ESD(u, v) = dist_S(U_c, V_c) + dist_S(U_d, V_d)$. In order to compute $dist_S(U_c, V_c)$, we observe that the pairwise distances $ESD(c_i, c_j), c_i \in U_c, c_j \in V_c$ are equal to 0, since the elements have identical sub-trees. Essentially, the two value sets contain equal values but at different multiplicities. If we use the MAC metric [10], then the computed distance $dist_S(U_c, V_c)$ is equal to 8 due to the difference in value frequencies. On the other hand, sets U_d and V_d have the same elements at the same frequencies and thus $dist_S(U_d, V_d) = 0$. Overall, $ESD(u, v) = 8 + 0 = 8$. Now, assume that v' is the left a element of T_2. It is straightforward to show that, under the same MAC metric, $ESD(u, v') = 6$ and thus, as expected intuitively, the element of T_2 simulates better the element of the true result.* ∎

Having defined the ESD metric between any two elements, we define the ESD metric between two trees T_1, T_2 as $ESD(T_1, T_2) = ESD(\texttt{root}(T_1), \texttt{root}(T_2))$. We note that $ESD(T_1, T_2)$ does not lend itself to a meaningful interpretation, except that a lower value represents increased similarity between T_1 and T_2. This, however, is a common characteristic of metrics that measure the approximate distance between complex objects (e.g., a similar observation holds for the MAC and EMD metrics). We note that it is possible to compute the ESD metric efficiently by first building the stable

Data Set	Elements	File Size (MB)	Stable Synopsis Size (KB)
IMDB-TX	102,754	3	77
XMark-TX	103,135	5	276
SProt-TX	182,300	4	265
IMDB	236,822	7	149
XMark	2,048,180	100	2,652
SProt	473,031	10	645
DBLP	1,594,443	48	204

Table 1: Data set characteristics

summaries of T_1 and T_2 on the fly and then evaluating the metric on the stable synopses. The key observation is that a stable summary preserves the path structure and the edge distributions of the original document, while containing fewer nodes. A detailed description of the computation of ESD on stable summaries can be found in the full version of the paper [17].

6. EXPERIMENTAL STUDY

In this section, we present an extensive experimental study of TREESKETCHes on real-life and synthetic data sets. Our results verify the effectiveness, in terms of accuracy and construction time, of the TREESKETCH synopses as structural summaries for large XML data sets. These benefits become even more apparent in a comparison to previously proposed techniques, where TREESKETCHes perform consistently better in all aspects. Overall, this empirical study indicates that TREESKETCHes are a viable and effective solution for the structural summarization of large XML data sets in real-world applications.

6.1 Testbed and Methodology

Techniques. We have experimented with two techniques.
TREESKETCHes. We have implemented a fully functional prototype of the TREESKETCH framework that we describe in this paper. Throughout our experiments, the construction algorithm uses an upper limit of $U_h = 10,000$ operations and rebuilds the heap when its size is reduced below $L_h = 100$ operations.
Twig-XSKETCHes. Twig-XSKETCHes [18] have been proposed as a summarization technique for estimating the selectivity of complex twig queries. Since the original proposal focused solely on selectivity estimation, we have developed an algorithm for producing approximate answers from a twig-XSKETCHThe algorithm traverses the query tree and uses the distribution information of the recorded edge histograms in order to sample the number of descendants for each element in the approximate result tree. For the construction of twig-XSKETCH summaries, we have used the same parameters that were reported in the original study [18].

Data Sets. We have used four data sets in our experiments: IMDB, a real-life data set from the Internet Movie Database Project; XMark, a synthetic data set that models transactions on a on-line auction site; Swiss Prot, a real-life data set with annotations on proteins; and DBLP, a real-life data set with bibliographical data. The main characteristics of the corresponding XML documents are summarized in Table 1. The TX documents have been used in the twig-XSKETCH study [18], and we include them here for the comparison of TREESKETCHes against twig-XSKETCHes. Looking at the sizes of the stable summaries, we observe that count-stability is very effective in compressing, without loss, the structural information of the original documents. Still, processing a query over so large a summary becomes prohibitively expensive relative to the stringent time requirements of an approximate answering system.

Query Workloads. For each data set, we evaluate the performance

	IMDB-TX	XMark-TX	SProt-TX
Avg Number of Binding Tuples	3,477	2,436	104,592

	IMDB	XMark	SProt	DBLP
Avg Number of Binding Tuples	13,039	145,577	365,493	78,784

Table 2: Workload characteristics

of the generated summaries against a workload of 1000 positive queries, i.e., queries that have non-empty results sets. Our experiments with negative workloads have shown that TREESKETCHes consistently produce empty answers as approximations and we therefore omit these workloads from our presentation in the interest of space. The workload is generated by sampling sub-trees from the stable synopsis and converting them to twig queries. Table 2 contains the average number of binding tuples per query in the workloads that we have generated.

Evaluation Metrics. We quantify the accuracy of approximate answers with the ESD metric which was defined in Section 5. More specifically, we compute the ESD between the approximate and the true nesting tree of each query in the workload and report the average over all queries. Our implementation uses a slightly revised version of MAC (kindly provided by Y. Ioannidis and V. Poosala) as the underlying set-distance metric, and limits comparisons to the binding elements of the same query variables. As always, the complete details can be found in the full paper [17].

For experiments on selectivity estimation, we measure the accuracy of the synopses with the average absolute relative error over all queries in the workload. More formally, if r is the true and e the estimated selectivity for a query in the workload, the absolute relative error is defined as $|r-e|/max(e,s)$. The sanity bound s is used to avoid the artificially high percentages of low-count queries. Following common practice [16, 18], we set s to the 10-percentile of true query counts.

6.2 Results

Approximate Query Answers. In this experiment, we evaluate the effectiveness of our novel TREESKETCH synopses as a practical solution for generating approximate answers to complex twig queries. We present a comparison against the previously proposed twig-XSKETCH synopses, focusing on two measures: the quality of the generated approximate answers, and the efficiency of the construction process.

Figure 11 shows the average ESD metric for approximate answers computed with TREESKETCHes and twig-XSKETCHes on a workload of 1000 twig queries, and for the XMark-TX, IMDB-TX, and SwissProt-TX data sets. We note that the increased distance numbers are partly due to the underlying MAC metric, which assigns a heavy penalty if the compared element sets contain the same sub-tree in different multiplicities. The interpretation of the results is therefore based on the relative performance of the two techniques, rather than on the absolute distances. Clearly, our novel TREESKETCH synopses consistently produce approximate answers of lower error. In all three data sets, the average distance for twig-XSKETCHes is at least four times higher than the one for TREESKETCHes, and the error for a 10KB TREESKETCH synopsis (lowest budget) is less than the error for a 50KB twig-XSKETCH (highest budget). The effectiveness of TREESKETCHes can be attributed to our novel clustering-based summarization model, which captures very accurately the intrinsic sub-structure similarity found in XML data. The edge-histogram model used by twig-XSKETCHes,

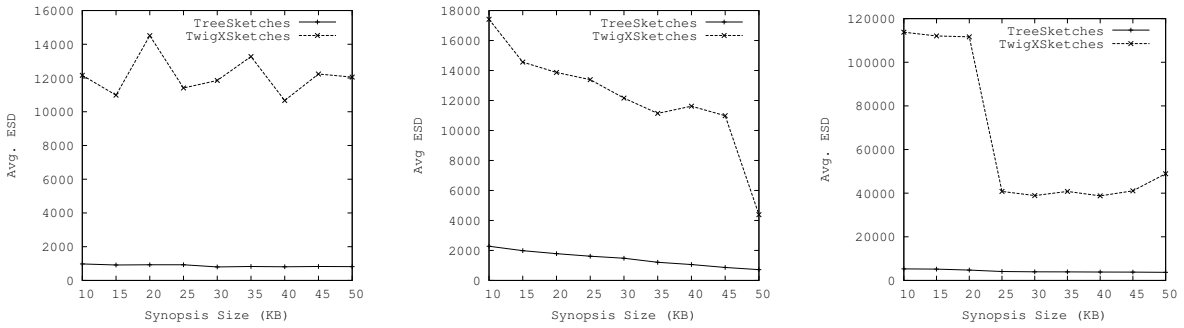


Figure 11: Average ESD metric for approximate answers: (a) XMark-TX, (b) IMDB-TX, (c) SwissProt-TX

on the other hand, can capture correlations within limited neighborhoods of synopsis nodes, while the typically high dimensionality of edge distributions affects negatively the quality of histogram approximation.

In terms of construction efficiency, we present a qualitative comparison between the two techniques since the twig-XSKETCH code base is not optimized for speed. The twig-XSKETCH construction algorithm starts from a coarse label-split graph, which contains exactly one node for all elements of the same tag, and gradually expands it through incremental refinement operations (basically, node splits, and histogram refinements). To evaluate the benefit of a candidate refinement, the algorithm measures the accuracy of the resulting twig-XSKETCH on a sample workload of twig queries (workload-based evaluation). Our proposed TSBUILD algorithm, on the other hand, compresses the stable summary down to the available space budget, using the squared error as a workload-independent quality metric.

	IMDB-TX	XMark-TX	SwissProt-TX
TREESKETCHes	0.7	8	10
Twig-XSKETCHes	13	47	55

Table 3: Construction times (in minutes) for TREESKETCHes and twig-XSKETCHes

Table 3 compares the construction time for TREESKETCHes and twig-XSKETCHes for the IMDB-TX, XMark-TX, and SwissProt-TX data sets. All times are reported in minutes and were measured on an unloaded Pentium4 3GHz machine, running Linux. For TREESKETCH synopses, we measure the time to compress the stable summary down to the smallest summary possible, the label split graph; for twig-XSKETCHes, we measure the time needed to expand the original coarse summary to 10KB of storage. This represents a worst case scenario for TREESKETCHes since the distance from the stable summary to the label-split graph is certainly "longer" than the distance from the label-split-graph to 10KB. Still, a qualitative comparison of the measured times indicates that TREESKETCH construction is much more efficient. As we described in Section 4.2, the TSBUILD algorithm uses effective heuristics to explore limited, yet promising regions of the search space, while the squared error metric, which is workload-independent, avoids the most expensive step of the twig-XSKETCH algorithm, namely evaluating the accuracy of candidate summaries against sample workloads.

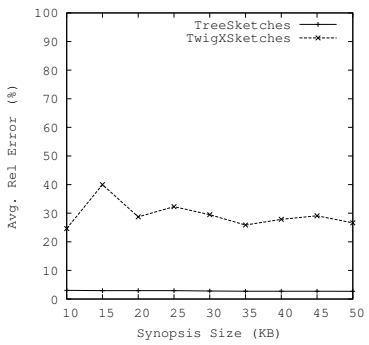
We have also evaluated the accuracy of TREESKETCH-generated approximate answers for the large datasets of Table 1. The results remain qualitatively the same as for the smaller data sets and we omit them in the interest of space. A detailed presentation can be found in the full version of this paper [17]. Note that we were not able to evaluate the performance of the twig-XSKETCH approach on the large data sets due to the high construction times.

Selectivity Estimation. In this experiment, we evaluate the effectiveness of our proposed synopses in estimating the selectivity of complex twig queries with branching path expressions. Figure 12 shows the average relative estimation error on a workload of 1000 queries for TREESKETCHes and twig-XSKETCHes, and for the XMark-TX and SwissProt-TX data sets. The results for the IMDB-TX data set are similar to XMark-TX and are omitted in the interest of space. As in the previous experiment, the results show that TREESKETCHes are effective in summarizing the key properties of the underlying path distribution. We observe that the estimation error remains well below 10% for all three data sets, even for small space budgets of 10KB-20KB that represent a small fraction of the original document sizes. Compared to twig-XSKETCHes, our new TREESKETCH synopses produce significantly more accurate estimates and exhibit more stable behavior.

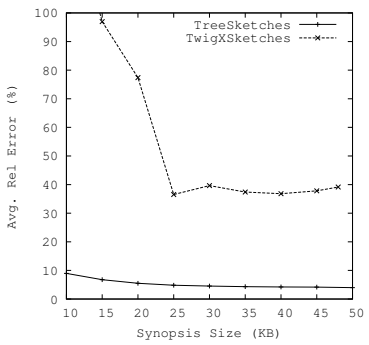
Figure 13 shows the TREESKETCH estimation error over a workload of 1000 queries and for the XMark, IMDB, SwissProt, and DBLP data sets (the large data sets of Table 1). The results verify the effectiveness of TREESKETCHes in computing accurate selectivity estimates for complex twig queries and demonstrate their nice scaling properties in terms of data size. In all four data sets, the estimation error drops below 5% for a space budget of 50KB, which in turn represents an extremely small fraction of the original document size. At the same time, the construction times remain affordable given the complexity and size of the involved data sets: 38 minutes for Swiss Prot, 11 minutes for DBLP, 2.5 minutes for IMDB, while the largest XMark data set required 4 hours.

7. CONCLUSIONS

Approximate answers constitute an effective solution for offsetting the high execution cost of complex XML queries in an interactive data exploration environment. In this paper, we have initiated the study of approximate query answering for XML data. We have proposed the TREESKETCH synopses, a novel class of structural summaries that capture very effectively the sub-structure similarity that is commonly found in XML data sets. We have developed a systematic evaluation algorithm for computing approximate answers over a concise TREESKETCH summary, and we have described an efficient heuristic construction algorithm for building an effective TREESKETCH for a limited space budget. To quantify the



(a)



(b)

Figure 12: Average selectivity estimation error: (a) XMark-TX, (b) SwissProt-TX.

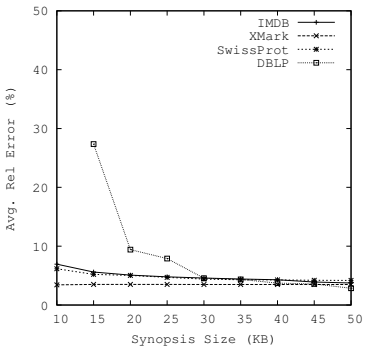


Figure 13: TREESKETCH **estimation error on large data sets.**

quality of the generated approximate answers, we have proposed a novel distance metric between XML trees that avoids the shortcomings of existing graph-theoretic metrics. Experimental results on real-life and synthetic data sets have verified the effectiveness of our approach and have demonstrated its benefits over previously proposed techniques.

8. REFERENCES

[1] Ashraf Aboulnaga, Alaa R. Alameldeen, and Jeffrey F. Naughton. "Estimating the Selectivity of XML Path Expressions for Internet Scale Applications". In *Proceedings of the 27th Intl. Conf. on Very Large Data Bases*, 2001.

[2] P. Buneman, M. Grohe, and C. Koch. "Path Queries on Compressed XML". In *Proceedings of the 29th Intl. Conf. on Very Large Data Bases*, 2003.

[3] Kaushik Chakrabarti, Minos Garofalakis, Rajeev Rastogi, and Kyuseok Shim. "Approximate Query Processing Using Wavelets". In *Proceedings of the 26th Intl. Conf. on Very Large Data Bases*, 2000.

[4] Don Chamberlin, James Clark, Daniela Florescu, Jonathan Robie, Jérôme Siméon, and Mugur Stefanescu. "XQuery 1.0: An XML Query Language". W3C Working Draft, 2001.

[5] Zhimin Chen, H.V. Jagadish, Laks V.S. Laksmanan, and Stelios Paparizos. "From Tree Patterns to Generalized Tree Patterns: On Efficient Eavluation of XQuery". In *Proceedings of the 29th Intl. Conf. on Very Large Data Bases*, 2003.

[6] Zhiyuan Chen, H. V. Jagadish, Flip Korn, Nick Koudas, S. Muthukrishnan, Raymond Ng, and Divesh Srivastava. "Counting Twig Matches in a Tree". In *Proceedings of the 17th Intl. Conf. on Data Engineering*, 2001.

[7] James Clark. "XSL Transformations (XSLT), Version 1.0". W3C Recommendation, November 1999.

[8] James Clark and Steve DeRose. "XML Path Language (XPath), Version 1.0". W3C Recommendation, November 1999.

[9] Juliana Freire, Jayant R. Haritsa, Maya Ramanath, Prasan Roy, and Jéróme Siméon. "StatiX: Making XML Count". In *Proceedings of the 2002 ACM SIGMOD Intl. Conf. on Management of Data*, 2002.

[10] Yannis E. Ioannidis and Viswanath Poosala. "Histogram-Based Approximation of Set-Valued Query Answers". In *Proceedings of the 25th Intl. Conf. on Very Large Data Bases*, 1999.

[11] Raghav Kaushik, Pradeep Shenoy, Phillip Bohannon, and Ehud Gudes. "Exploiting Local Similarity for Efficient Indexing of Paths in Graph Structured Data". In *Proceedings of the 18th Intl. Conf. on Data Engineering*, 2002.

[12] L. Lim, M. Wang, S. Padmanabhan, J.S. Vitter, and R. Parr. XPathLearner: An On-Line Self-Tuning Markov Histogram for XML Path Selectivity Estimation. In *Proceedings of the 28th Intl. Conf. on Very Large Data Bases*, 2002.

[13] Jason McHugh and Jennifer Widom. "Query Optimization for XML". In *Proceedings of the 25th Intl. Conf. on Very Large Data Bases*, 1999.

[14] Tova Milo and Dan Suciu. "Index structures for Path Expressions". In *Proceedings of the 7th Intl. Conf. on Database Theory (ICDT'99)*, 1999.

[15] N. Polyzotis and M. Garofalakis. "Statistical Synopses for Graph Structured XML Databases". In *Proceedings of the 2002 ACM SIGMOD Intl. Conf. on Management of Data*, 2002.

[16] N. Polyzotis and M. Garofalakis. "Structure and Value Synopses for XML Data Graphs". In *Proceedings of the 28th Intl. Conf. on Very Large Data Bases*, 2002.

[17] Neoklis Polyzotis, Minos Garofalakis, and Yannis Ioannidis. "Approximate XML Query Answers". *University of California Santa Cruz, Technical Report*, 2004.

[18] Neoklis Polyzotis, Minos Garofalakis, and Yannis Ioannidis. "Selectivity Estimation for XML Twigs". In *Proceedings of the 20th Intl. Conf. on Data Engineering*, 2004.

[19] C. M. Procopiuc. *Geometric Techniques for Clustering: Theory and Practice*. PhD thesis, Duke Univ., 2001.

[20] D. Sasha and K. Zhang. Fast algorithms for the unit cost editing distance between trees. *Jnl. of Algorithms*, 11, 1990.

[21] Wei Wang, Haifeng Jiang, Hongjun Lu, and Jeffrey Xu Yu. Containment join size estimation: Models and methods. In *Proceedings of the 2003 ACM SIGMOD Intl. Conf. on Management of Data*, 2003.

[22] Yuqing Wu, Jignesh M. Patel, and H.V. Jagadish. "Estimating Answer Sizes for XML Queries". In *Proceedings of the 8th Intl. Conf. on Extending Database Technology*, 2002.

[23] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. In *Proceedings of the 1996 ACM SIGMOD Intl. Conf. on Management of Data*, 1996.