XSKETCH Synopses for XML Data Graphs

NEOKLIS POLYZOTIS
University of California, Santa Cruz
and
MINOS GAROFALAKIS
Intel Research Berkeley

Effective support for XML query languages is becoming increasingly important with the emergence of new applications that access large volumes of XML data. All existing proposals for querying XML (e.g., XQuery) rely on a pattern-specification language that allows (1) *path navigation and branching through the label structure* of the XML data graph, and (2) *predicates on the values* of specific path/branch nodes, in order to reach the desired data elements. Clearly, optimizing such queries requires approximating the result cardinality of the referenced paths and hence hinges on the existence of concise synopsis structures that enable accurate compile-time selectivity estimates for complex path expressions over the base XML data. In this article, we introduce a novel approach to building and using statistical summaries of large XML data graphs for effective path-expression selectivity estimation. Our proposed graph-synopsis model (termed XSKETCH) exploits *localized graph stability* and *value-distribution summaries* (e.g., histograms) to accurately approximate (in limited space) the path and branching distribution, as well as the complex correlation patterns that can exist between and across path structure and element values in the data graph. To the best of our knowledge, ours is the first work to address this timely problem in the most general setting of graph-structured XML data with values, and complex (branching) path expressions.

Categories and Subject Descriptors: H.2.4 [**Database Management**]: Systems—*Query processing*; G.2.1 [**Discrete Mathematics**]: Combinatorics—*Combinatorial algorithms*

General Terms: Algorithms, Performance, Theory

Additional Key Words and Phrases: XML, data synopses, approximate query processing, path expressions

---

Most of this work was done while the first author was a Ph.D. student at the University of Wisconsin and a research intern a Bell Labs, Lucent Technologies, and the second author was with Bell Labs, Lucent Technologies.

Authors' addresses: N. Polyzotis, Department of Computer Science, University of California, Santa Cruz, 1156 High St., Santa Cruz, CA 95064; email: alkis@cs.ucsc.edu; M. Garofalakis, Intel Research Berkeley, 2150 Shattuck Ave., Penthouse Suite, Berkeley, CA 94704; email: garofalakis@intel.com.

1. INTRODUCTION

The Extensible Markup Language (XML) is rapidly emerging as the new standard for data representation and exchange on the Internet. The simple, self-describing nature of the XML standard promises to enable a broad suite of next-generation Internet applications, ranging from intelligent Web searching and querying to electronic commerce. In many respects, XML represents an instance of *semistructured data* [Goldman and Widom 1997]: the underlying data model comprises a labeled graph of *element* nodes, where each element can be either an atomic data item (i.e., raw values stored with elements) or a composite data collection consisting of references (represented as graph edges) to other elements in the graph. Further, *labels* (or *tags*) stored with XML data elements describe the actual semantics of the data rather than simply specifying how the element is to be displayed (as in HTML). Thus, XML data, like semistructured data, is graph-structured and self-describing.

Sophisticated query-processing engines that allow users and applications to effectively tap into the large amounts of data stored in XML databases around the globe are going to be crucial in fulfilling the full potential of XML. Realizing such Internet-scale XML query processors, such as, Xyleme (`www.xyleme.com`), Niagara [Naughton et al. 2001], or TIMBER [Jagadish et al. 2002], in turn, hinges on providing effective support for high-level, declarative XML query languages. A variety of languages have been proposed for querying semistructured and XML databases, including XQuery [Boag et al. 2005], Lorel [Goldman and Widom 1997], and UnQL [Buneman et al. 1996]. A common characteristic of all existing language proposals is the existence of a pattern-specification language (e.g., XPath [Clark and DeRose 1999]) built around *path* and *subtree ("twig") expressions*. These expressions replace the traditional SQL `FROM` clause and enable selections based on *value predicates* as well as *path navigation and branching* through the XML data graph in order to reach the relevant data elements. While simple path queries were popularized in the context of object-oriented databases, the pattern-specification languages proposed for XML data are substantially more complex. In particular, the XPath language [Clark and DeRose 1999] (which lies at the core of XQuery [Boag et al. 2005] and XSLT [Clark 1999], the dominant W3C language proposals for XML querying and transformation) allows *branching regular path expressions* that enable queries to navigate along paths in the data graph using label names, wildcards, value predicates, and branching predicates on the existence of specific sibling paths. As a concrete example, in a bibliography database, the XPath expression `//author[book]/paper/vldb[year > 1997]/title` selects the set of all VLDB article `titles` published after 1997 by `authors` that have published *at least one* `book` (specified by the `author[book]` branch).

Optimizing XML queries with complex path expressions depends crucially on the ability to obtain effective compile-time estimates for the selectivity of these expressions over the underlying (large) graph-structured XML database. Similar to relational query optimization, selecting an efficient query-execution plan relies on the accurate estimation of the number of XML elements that are accessed from (i.e., "satisfy") a path-expression specification. To be feasible

at query-optimization time, this estimation process has to depend on a concise and accurate *statistical synopsis* of the structure and value content of the XML data graph that can provide such selectivity estimates within the memory and time constraints of the optimizer. Of course, such a synopsis can also be an invaluable tool for providing users with fast approximate answers and quick feedback to their queries, either before or during query execution.

In this article, we propose a novel approach to building and using concise statistical synopses for effectively estimating the selectivity of complex XPath expressions with branching and value predicates, over general, graph-structured XML data. Our proposed synopsis model, termed XSKETCH, exploits *localized graph stability* in conjunction with *small-space value summaries* to accurately capture (in limited space) the important statistical characteristics of the path, branching, and value distribution in the XML data graph. Hence, an XSKETCH summary can enable selectivity estimates for complex path expressions that query both the structure and the value content of large XML data graphs. We develop a systematic estimation framework for approximating path-expression selectivities over concise XSKETCH synopses, and propose an efficient algorithm for XSKETCH construction. We should note that the problem of XML summarization has received considerable attention within the database community, and recent studies have introduced a host of relevant techniques [Aboulnaga et al. 2001; Chen et al. 2001; Freire et al. 2002; Lim et al. 2002; Wang et al. 2003; Wu et al. 2002]. To the best of our knowledge, however, our work is the first to deal with the most general version of the summarization problem, namely, graph-structured XML data and path expressions with both branching and value predicates. More concretely, our key contributions can be summarized as follows.

—XSKETCH *synopsis model and estimation framework.* We give a formal definition of our XSKETCH synopses for XML data that exploit the concepts of localized *backward and forward graph stability* [Paige and Tarjan 1987] and *value-distribution summaries* to effectively explore the space between extremely coarse (but inaccurate) and extremely detailed (but large) summarizations of graph-structured data. We develop a systematic estimation framework that uses the information in the XSKETCH synopsis to parse a complex path expression and produce an approximate selectivity estimate. Like any estimation technique that uses concise data synopses (e.g., histograms [Poosala et al. 1996]), our proposed framework relies on a set of appropriate statistical (uniformity and independence) assumptions to compensate for the lack of detailed distribution information.

—XSKETCH *construction: hardness and efficient heuristic algorithm.* Constructing effective XSKETCH synopses turns out to be a difficult optimization problem: we demonstrate that the problem of building an accuracy-optimal XSKETCH for a given space budget is $\mathcal{NP}$-hard, even for the simpler "structure-only" case. Given this intractability result, we propose an efficient heuristic algorithm for XSKETCH construction based on greedy forward selection. Briefly, our algorithm constructs an XSKETCH synopsis in a top-down fashion, by successive refinements of the label-split graph, the coarsest summary

of the XML data graph. Our refinement operations act locally and attempt to capture important statistical correlations between data paths. The end result is an XSKETCH synopsis that, abstractly, is more refined where correlations are stronger and less refined where data paths are independent and uniformity assumptions are valid.

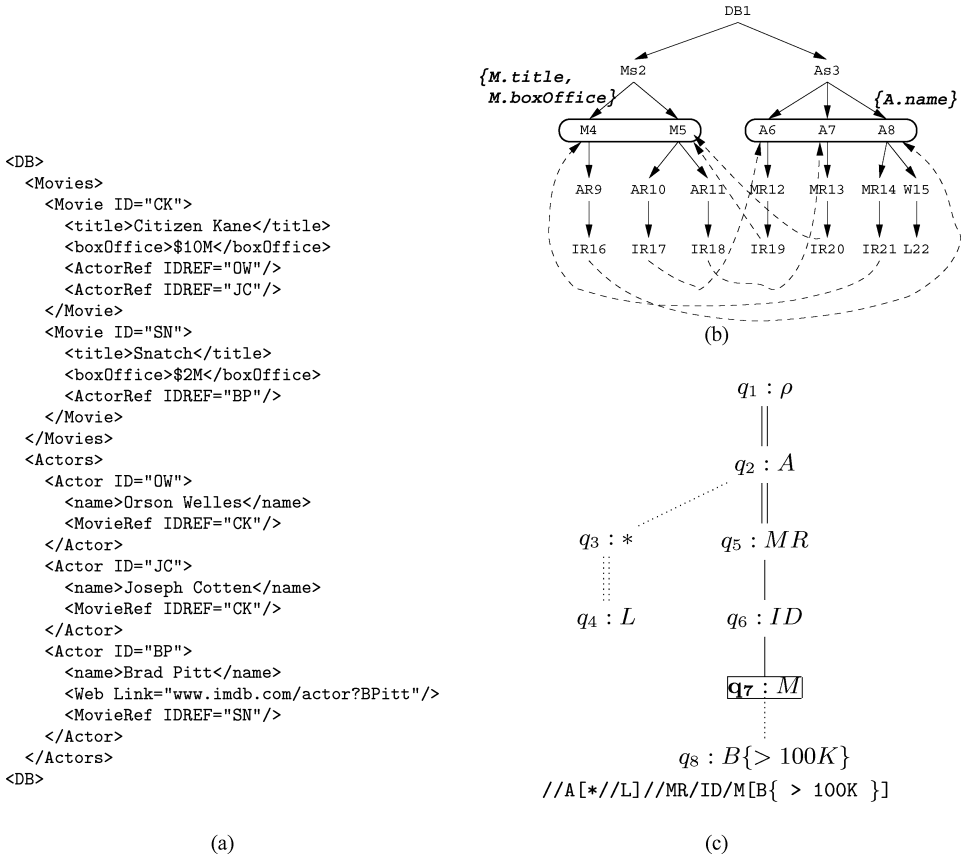
—*Experimental results verifying the effectiveness of* XSKETCH*es.* We present the results of an extensive experimental study of XSKETCHes with several synthetic and real-life data sets that validate our approach. Our results show that XSKETCHes are accurate, concise synopses for general graph-structured XML data, achieving estimation errors as low as 3% for low space budgets around 30–40 kB. The generated summaries are built utilizing small path samples from the original document, thus ensuring the efficiency of the XSKETCH construction algorithm. We experiment with both complex and simple path expressions and show that the constructed summaries yield accurate estimates in all cases; furthermore, our XSKETCHes perform better and more consistently than earlier approaches for the simpler problem of handling simple path expressions over tree-structured XML data.

2. PRELIMINARIES

2.1 XML Data Model

Following previous work on XML and semistructured data [Goldman and Widom 1997; Kaushik et al. 2002b; Milo and Suciu 1999], we model an XML database as a directed, node-labeled *data graph* $G = (V_D, E_D)$. Each node in V_D corresponds to an XML element in the database and is characterized by (a) a unique *object identifier (oid)*, (b) a *label* (assigned from some alphabet of string literals) that captures the element's semantics, and (c) (possibly) a *raw data value* for the element. (We use $\texttt{label}(v)$, $\texttt{value}(v)$ to denote the label and value of $v \in V_D$.) Without loss of generality, we assume that the root of the database is labeled with a distinct tag ρ that does not appear in any other element.

Edges in E_D are used to capture both the element-subelement relationships (i.e., element nesting or explicit element references through id/idref attributes or XLink constructs [Bray et al. 2000; DeRose et al. 2001; Kaushik et al. 2002b; Goldman and Widom 1997]) and the element-value relationships in the XML data. Given a data-graph node u, we define its predecessors as $\texttt{pred}(u) = \{v \mid (v, u) \in E_D\}$ and its successors as $\texttt{succ}(u) = \{w \mid (u, w) \in E_D\}$; this is extended naturally to a set of nodes $U = \{u_i\}$ as $\texttt{pred}(U) = \cup_i \texttt{pred}(u_i)$ and $\texttt{succ}(U) = \cup_i \texttt{succ}(u_i)$. Note that nontree edges, such as those implemented through id/idref constructs, form an integral part of the XML data model and are often of equal importance for the modeling of complex data sets (a typical example is the representation of a part/subpart hierarchy, where nesting is used for the description of individual parts while id/idref edges implement the inclusion relationship). As a result, nontree edges are treated as a "first-class citizen" in our model and are not differentiated from normal tree edges. We therefore focus on the most general case of XML data *graphs* (rather than just trees) for the remainder of this article. Of course, our model implies the existence of




```
<DB>
  <Movies>
    <Movie ID="CK">
      <title>Citizen Kane</title>
      <boxOffice>$10M</boxOffice>
      <ActorRef IDREF="OW"/>
      <ActorRef IDREF="JC"/>
    </Movie>
    <Movie ID="SN">
      <title>Snatch</title>
      <boxOffice>$2M</boxOffice>
      <ActorRef IDREF="BP"/>
    </Movie>
  </Movies>
  <Actors>
    <Actor ID="OW">
      <name>Orson Welles</name>
      <MovieRef IDREF="CK"/>
    </Actor>
    <Actor ID="JC">
      <name>Joseph Cotten</name>
      <MovieRef IDREF="CK"/>
    </Actor>
    <Actor ID="BP">
      <name>Brad Pitt</name>
      <Web Link="www.imdb.com/actor?BPitt"/>
      <MovieRef IDREF="SN"/>
    </Actor>
  </Actors>
<DB>
```

(a)

Fig. 1. (a) Example XML document, (b) data graph of document, (c) example query tree and its XPath notation.

meta-data that enable the discovery of nontree edges, for example, in the case of id/idref links, a DTD is necessary in order to identify and interpret correctly the ID and IDREF attributes. This point is orthogonal to our framework and we henceforth assume that such metadata is readily available.

Figure 1 depicts an example XML document and the corresponding data graph. The sample document is modeled after the Internet Movie Database (IMDB) XML data set (`www.imdb.com`) and includes information on two movies and three actors. The graph node corresponding to a data element is named with an abbreviation of the element's label and a unique id number. Also, as shown, each movie (`M`) element is associated with two values (i.e., `title` and `boxOffice` sales), and each actor (`A`) element is associated with one value (i.e., `name` of the actor); the specific values for each element are not shown to avoid cluttering the figure. Dashed lines are used for graph edges corresponding to `id-idref` relationships.

We say that an XML data graph is *resursive* if there exist at least two elements e and e' that lie on the same path and $\texttt{label}(e) = \texttt{label}(e')$. An example of a recursive data set is our sample data graph of Figure 1, as a `movie` element

can reach another `movie` element through a common actor reference. (The same holds for actor elements.) We note that recursion may appear in tree-structured data as well, and is not exclusive to graph structures.

2.2 XML Query Model

An XML path expression Q (e.g., in XQuery [Boag et al. 2005]) defines a navigational path over the XML data graph, specifying conditions on the labels and (possibly) the value(s) of data elements. More formally, a path expression (or query) Q is defined as a node- and edge-labeled tree $G_Q(V_Q, T_Q)$. Each node (or *step*) $q_i \in V_Q$ $(1 \leq i \leq N_Q)$ carries a label $\texttt{label}(q_i)$ and essentially represents the selection of document elements with a matching tag. In our model, $\texttt{label}(q_i)$ can be either a document tag, or the special wildcard label '$*$' that matches any tag.[1] (We assume that the distinguished root node q_1 always carries the root label ρ.) A step can also be annotated with an optional value predicate σ_i that further restricts the set of selected elements.

An edge $(q_i, q_j) \in E_Q$ denotes a structural relationship between the two steps and is labeled with *child axis* (/) or *descendant axis* (//). (We use $axis(q_i, q_j)$ to denote the edge label.) In a nutshell, a child axis (respectively, descendant axis) specifies that the elements of q_j must have a parent (resp. ancestor) in the elements of q_i. We say that Q is a *recursive query* if at least one edge carries a descendant annotation. We assume that Q contains a distinguished root-to-node path, termed the *main branch* and denoted by $\overline{L}$, whose last node determines the elements returned by the query. The remaining paths are attached to the nodes of the main branch and are termed the *branching predicates* of the query. Essentially, a branching predicate at node q_i requires the existence of sibling paths for the elements that q_i selects.

Figure 1(c) shows an example query over the IMDB data set and its translation in the XPath language. We note that our notation is slightly different from that of XPath, as we are using {} to distinguish value predicates from element-branching predicates (denoted by []). The tree-based representation denotes a descendant axis with a double arc, while a dashed arc is used for edges that belong to branching predicates. As shown, the example query contains two branching predicates at steps q_7 and q_2, and one selection predicate at step q_8. Following the semantics of the XPath language, the results of this query can be defined as follows. The first step q_1 generates the root node of the document, while the second step q_2 of the main branch selects all the `Actor` descendants of the root that have at least one child (step q_3) with a `Link` descendant (step q_4). The following step q_5 selects all the `MovieRef` descendants of such actors, while q_6 returns all the `ID` children of these movie-reference elements. The final step q_7 of the main branch selects all `Movie` children of the previously computed `ID`s that satisfy the branching predicate [$B > 100$K], that is, they have at least one `BoxOffice` child with value greater than 100K. The resulting elements form the result of the query.

[1]It is straightforward to extend our framework so that each step corresponds to a set of document tags. We omit this extension to keep the presentation simple.

More formally, we define the result of a query through the concept of a *data embedding* that represents the matching of a query tree against the data graph. A tuple of elements $(e_{k_1} : e_{k_2} : \ldots : e_{k_{N_Q}})$ forms a data embedding of Q if (a) $\texttt{label}(e_{k_i})$ matches $\texttt{label}(q_i)$, (b) the value of e_{k_i} satisfies the optional predicate σ_i, and (c) elements e_{k_i} and e_{k_j} match $axis(q_i, q_j)$, that is, e_{k_j} is a child (respectively, descendant) of e_{k_i} if $axis(q_i, q_j) = /$ (respectively, $axis(q_i, q_j) = //$). Intuitively, a data embedding is an assignment of elements to query nodes that respects the structural and value constraints of the query. The result of Q is simply defined as the set $\mathcal{M}(Q)$ of elements that are assigned by data embeddings to the last node of the main branch $\overline{L}$.

There are three points of interest with respect to the semantics of XPath expressions. First, branching predicates are essentially existential quantifiers, as they specify the existence of at least one sibling path. To see this, note that the number of data paths that match branching predicates does not affect the selectivity of the query. Second, the result contains only the elements that are mapped to the very last step of the main branch. Finally, the result is a *set* of elements and hence it becomes important to distinguish between the *set of elements* in $\mathcal{M}(Q)$ and the *set of paths* that reach the elements in $\mathcal{M}(Q)$. Even though these two are essentially equivalent for tree-structured data, the same does not hold for graph structures where the same element can be reached by several paths.

3. THE XSKETCH SYNOPSIS MODEL

This section presents our proposed XSKETCH model for summarizing the path and value distribution of large XML databases. At an abstract level, an XSKETCH summary is an instantiation of a generic *graph synopsis*, that captures the basic characteristics of the underlying data graph, augmented with detailed distribution information in order to enable selectivity estimates for complex XPath expressions. An example is presented in Figure 2(c), which depicts an XSKETCH summary for the sample data graph of the previous section. In a nutshell, a synopsis node corresponds to a subset of XML elements with the same tag, for example, `M(3)` denotes a subset of three `Movie` elements, while an edge denotes the existence of document edges between the corresponding element sets. As we discuss later, the statistical information stored in the synopsis comprises the element counts of nodes and the **B** and **F** edge annotations that essentially describe key properties of element connectivity. The goal is to carefully select a partitioning of XML elements to synopsis nodes, so that the induced statistical information models accurately the important properties of the underlying data distribution. The following sections describe the XSKETCH model in more detail. We first introduce a framework for summarizing the graph structure of the XML data, and then extend the proposed model to the general case of data graphs with values.

3.1 Summarizing Graph Structure

3.1.1 *Graph Synopsis Model.* Abstractly, our general model of a synopsis for an XML data graph $G = (V_G, E_G)$ is a node-labeled, directed graph structure

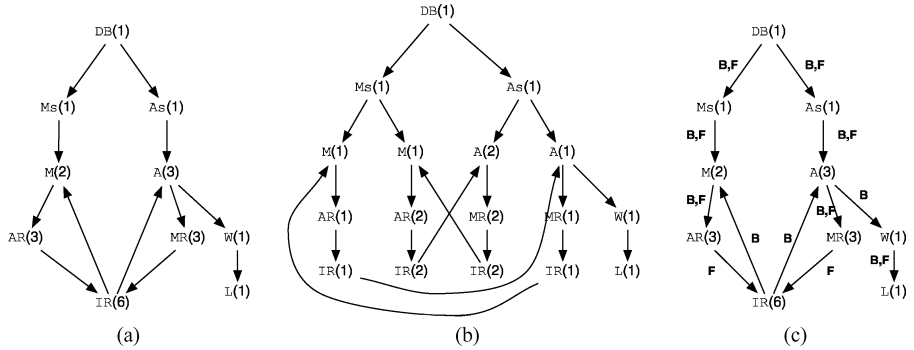


Fig. 2. Synopses for XML data of Figure 1: (a) label-split graph, (b) B/F-bisimilar graph, (c) XSKETCH based on the label-split graph.

$\mathcal{S}(G) = (V_\mathcal{S}, E_\mathcal{S})$, where each node in $v \in V_\mathcal{S}$ corresponds to a subset of identically labeled nodes in a partitioning of V_G (termed the *extent* of v) and an edge in $(e_i, e_j) \in E_G$ is represented in $E_\mathcal{S}$ as an edge between the nodes whose extents contain the two endpoints e_i and e_j. To enable selectivity estimates for complex path expressions, each node v of $\mathcal{S}(G)$ only captures summary information about G in the form of a *count* field (`count`(v)) that records the number of elements in G that map to v, that is, the size of v's extent. (We use v and `extent`(v) interchangeably in what follows.)

Definition 3.1. A (structural) graph synopsis for $G = (V_G, E_G)$ is a node-labeled, directed graph $\mathcal{S}(G) = (V_\mathcal{S}, E_\mathcal{S})$, where each node $v \in V_\mathcal{S}$ corresponds to a set `extent`$(v) \subseteq V_G$ such that (1) all elements in `extent`(v) have the *same label*[2] (denoted by `label`(v), that is, the label of the synopsis node); (2) $\cup_{v \in V_\mathcal{S}}$ `extent`$(v) = V_G$ and `extent`$(u) \cap$ `extent`$(v) = \phi$ for each $u, v \in V_\mathcal{S}$; (3) $(u, v) \in E_\mathcal{S}$ if and only if there exist $u' \in$ `extent`(u) and $v' \in$ `extent`(v) such that $(u', v') \in E_G$; and (4) each node $v \in V_\mathcal{S}$ stores only an element count `count`$(v) = |$`extent`$(v)|$.

Figures 2(a) and 2(b) show two possible graph synopses for our example XML document, where each synopsis node is depicted by its `label` followed by its `count` attribute in parentheses. (The choice of the particular summaries will become apparent later.) Several recently proposed path-index structures for XML data, including 1-indexes [Milo and Suciu 1999], and A(k)-indexes [Kaushik et al. 2002b], are based on the "node-partitioning" technique described in our general graph-synopsis definition. As an example, the 1-index and the A(k)-index are based on the *bisimilarity* and *k-bisimilarity* partition of G, respectively. (Briefly, *bisimilarity* groups together data nodes that have identically structured incoming paths from the document root [Milo and Suciu 1999], whereas *k-bisimilarity* is based on a more relaxed rule that essentially considers incoming paths *of length at most k* [Kaushik et al. 2002b].) However, given the stringent space limitations for our compile-time, selectivity-estimation

[2]In the general case, it is straightforward to parameterize this definition with an abstract function $\mathcal{P}$ that determines whether two elements are "compatible" and can thus appear in the same `extent`. We omit this generalization in order to keep our presentation simple.

problem, the graph synopsis can only store the extent counts (rather than the entire extents, typically stored in the aforementioned index structures). Our goal is to be able to evaluate the selectivity of complex path expressions over the data graph G based solely on a compact graph synopsis of G.

At this point, it is interesting to consider the two extreme points in our model space of graph synopses for summarizing the path structure of XML databases. At one extreme, the *label-split graph* represents a very succinct but, at the same time, coarse and inaccurate synopsis of the data graph; at the other extreme, the *Backward/Forward-bisimilar (B/F-bisimilar)* graph represents an exact but prohibitively large synopsis for branching-path selectivities.

—*The coarsest graph synopsis: label-split graph* ($\mathcal{S}_0(G)$). The label-split (or 0-bisimilar [Kaushik et al. 2002a]) graph synopsis groups data nodes into synopsis nodes based solely on their node labels; that is, all nodes in G sharing the same label are mapped onto a unique node in $\mathcal{S}_0(G)$. The label-split graph is a very succinct representation of the data graph, as the number of nodes in $\mathcal{S}_0(G)$ is exactly the number of distinct labels in G. Unfortunately, the label-split graph also presents a very poor picture of the path distribution in G since, exactly due to its coarseness, it typically contains several *false paths and cycles* (that did not exist in the original data graph).

—*The perfect graph synopsis: B/F-bisimilar graph* ($\mathcal{S}_{B/F}(G)$). The B/F-bisimilar partitioning maps data nodes to the same node of $\mathcal{S}_{B/F}(G)$ only if they share the same set of *incoming and outgoing paths* in G. This represents a refinement of the well-known Backward-bisimilarity (B-bisimilarity or, simply, bisimilarity) partitioning, that has been used, for example, in the exact 1-index for simple paths [Milo and Suciu 1999]). It is easy to verify that such a B/F-bisimilar graph synopsis captures the exact path structure of the underlying data graph and can thus enable zero-error selectivity estimates for any branching path expression. As a previous study [Kaushik et al. 2002a] has shown, however, the size of the B/F-bisimilar graph can grow close to the size of the original data; given the stringent time and memory constraints of a query optimizer, therefore, its use as a concise summary becomes problematic.

Figure 2 depicts the two extremes of synopsis graphs for the example document of Figure 1. Figure 2(a) depicts the label-split graph, the coarsest summary of the document. We observe that it captures only part of the original path structure, while introducing a number of false paths (e.g., `A/MR/IR/A`). Figure 2(b) shows the B/F-bisimilar graph which represents an exact summary. Note that the summary groups together actor elements `A6,A7` since they cannot be distinguished based on their incoming and outgoing paths and the same holds for elements (`MR12, MR13`), (`IR19, IR20`), (`AR10, AR11`), and (`IR17, IR18`). Overall, the B/F-Bisimilar graph contains all the paths of the original document and no false paths, yet its size can be close to that of the original data graph.

3.1.2 *Structural* XSketch *Model.* Our proposed XSketch synopsis structures represent specific instantiations of this general graph-synopsis model. Two key concepts underlying XSketches are those of *backward- and*

forward-stability (B- and F-stability, respectively) [Paige and Tarjan 1987] that we define formally below.

Definition 3.2. Let V, U be sets of elements in an XML data graph G. We say that V is *backward-stable (B-stable)* with respect to U, if and only if, for each $e \in V$ there exists a $e' \in U$ such that the edge (e', e) is in G. Similarly, U is said to be *forward-stable (F-stable)* with respect to V, if and only if for each $e \in U$ there exists a $e' \in V$ such that the edge (e, e') is in G. Given a graph synopsis $\mathcal{S}(G)$ of G, we define a node u in the synopsis to be B-stable (F-stable) with respect to another synopsis node v if and only if $\texttt{extent}(u)$ is B-stable (respectively, F-stable) with respect to $\texttt{extent}(v)$.

Note that, by Definitions 3.1 and 3.2, a node in a graph synopsis $\mathcal{S}(G)$ can only be B-stable (F-stable) with respect to its parent (respectively, child) nodes in $\mathcal{S}(G)$. Thus, a synopsis node u is B-stable with respect to its parent v if and only all data elements mapped to u have a parent data element mapped to v in $\mathcal{S}(G)$; in other words, the number of data elements in $\texttt{extent}(u)$ that are reached by an edge from data elements in $\texttt{extent}(v)$ in G is *exactly* $\texttt{count}(u) = |\texttt{extent}(u)|$. Similarly, the F-stability condition in $\mathcal{S}(G)$ guarantees the count field of a parent synopsis node u is an exact estimate for the number of elements in u's extent that reach (by an edge in G) elements mapping to a child node in the synopsis.

Example 3.1. Consider again the label-split graph of Figure 2(a). We observe that all elements in extent(MR) have a parent element in extent(A) and, therefore, MR is B-stable with respect to A. As a result of this stability, the count associated with MR obviously yields an exact estimate for the number of elements reached by the path expression A/MR; in fact, since we can show that A is in turn B-Stable with respect to IR (all Actor elements are reached by an IDREF attribute), MR's count gives an exact estimate for the selectivity of IR/A/MR as well. Node IR, on the other hand, is not B-stable with any of its parent nodes; thus, the count associated with IR does not give an exact selectivity estimate for any path expression that ends in IR.

We can make similar observations about forward stabilities. All elements, for example, in extent(MR) have a child element in extent(IR) and, therefore, MR is F-stable with respect to IR; as a result, MR's count yields an exact selectivity estimate for the path expression MR[IR]. Note, however, that F-stability does not say anything about the number of elements in IR reached by elements in MR; it only guarantees that the path MR/IR exists for all elements in MR.

It is interesting to note that there is an obvious connection between the concepts of stability and graph bisimilarity. Essentially, stability can be seen as a *localized* notion of bisimilarity since, for a given synopsis node, it only considers paths of length one to/from specific child/parent node(s) in the synopsis. In fact, stability plays a central role in known efficient algorithms for computing the bisimilarity partition of a graph (e.g., Paige and Tarjan [1987]), where the basic operation is to *stabilize* a subset in the partition with respect to other subsets and the final bisimilarity partition is reached when every subset is stable with respect to its neighboring subsets (e.g., parent subsets for B-bisimilarity). As will become clear later in this section, it is precisely this localized character of

stability that we exploit in our XSKETCHes to ensure that the limited space available for the synopsis is judiciously allocated to those portions of the data graph where our estimation assumptions are particularly inappropriate. To allow for such localized refinements at different levels of resolution, our XSKETCH synopses augment our general graph-synopsis model with a 2-bit edge label that is used to indicate possible B-stability, F-stability, or both (i.e., B/F-stability) between neighboring nodes in the synopsis.

Definition 3.3. A *structural* XSKETCH $\mathcal{X}S(G) = (V_{\mathcal{X}S}, E_{\mathcal{X}S})$ for a data graph G is an *edge-labeled* graph synopsis for G, where the label for each edge $(u, v) \in E_{\mathcal{X}S}$ is a 2-bit indicator whose value is defined as follows: (1)$\texttt{label}(u, v) = \{\mathbf{B}\}$, if v is B-stable with respect to u; (2)$\texttt{label}(u, v) = \{\mathbf{F}\}$, if u is F-stable with respect to v; (3)$\texttt{label}(u, v) = \{\mathbf{B}, \mathbf{F}\}$, if both (1) and (2) hold; and (4)$\texttt{label}(u, v) = \phi$ (empty), otherwise.

An example structural XSKETCH synopsis for the XML data graph in Figure 1(b) is depicted in Figure 2(c); note that, in this specific example, the XSKETCH is simply the label-split graph of Figure 2(a) augmented with the appropriate B/F-stability labels.

Overall, edge stabilities are the basic mechanism of our proposed XSKETCH framework for summarizing the path structure of the XML data graph. The key intuition is that, by definition, a stable edge identifies a subset of the XML data with specific properties on element connectivity and thus a careful data-node partitioning, along with the induced stabilities, can capture skew in the underlying data distribution. This idea is analogous to the bucketization of a frequency distribution in histogram-based synopses, where a bucket essentially represents a group of values that have similar frequencies. We will revisit this point in Section 5, where we present a framework for refining an XSKETCH summary and using edge stabilities to capture areas of structural skew.

3.2 Capturing Value-Distribution Information

The previous section introduced our *structural* XSKETCH model, for summarizing the path distribution of a large XML data graph. Clearly, this represents only a partial solution to the general XML summarization problem, as it ignores a key part of XML data, namely, the value content of elements and attributes. In this section, we extend our proposed summarization model to the general case of XML hierarchies with values.

A naive solution to adding information on element values would be to directly apply our structural XSKETCH framework, simply treating different data values as different "labels" in the graph. Of course, the problem with such an approach is that the number of distinct values in an XML database is typically far greater than the number of distinct element labels; thus, such a naive solution is likely to cause an explosion in the size of any structural graph synopsis. Even the coarsest graph synopsis (i.e., the *label-split graph* that simply groups data nodes by label) can become too large to be useful as an optimization-time structure, whereas the perfect graph synopsis (i.e., the fully *B/F-bisimilar graph*) can easily be as large as the database itself. Instead, the key idea of our

proposed approach is to incorporate compressed value-distribution information (using, e.g., histograms) in the nodes of an XSKETCH synopsis in order to effectively capture the distribution of element values in nodes' extents. Despite its deceptive simplicity, this turns out to be a rather difficult problem since, to be effective, the resulting XSKETCH synopses need to accurately capture complex correlation patterns that may exist in the underlying graph-structured data. More specifically, there are two key forms of correlations that our synopses need to model.

—*Path/value correlations.* Given a node v in the XSKETCH, the characteristics of the value distribution for data elements in `extent`(v) can vary drastically depending on the specific *label path(s)* that reach these elements (or, leave from these elements) in the data. For example, consider a bookstore database and a `book`-labeled node v in the synopsis that records book-pricing information; obviously, the prices of elements in `extent`(v) reached through the label path `cs/textbooks/book` will be very different from those reached through `poetry/rare` − `collections/book`. Thus, just maintaining a histogram for the complete set of prices under v is very likely to produce inaccurate estimates for selections on `book` prices that specify either of the two label paths.

—*Value correlations.* Given a node v in the XSKETCH, the distribution characteristics for elements in `extent`(v) that are reached through (or, lead to) a *specific label path*, can depend to a large extent on the *values* of other elements on that path. Continuing with our bookstore example, assume that we have separated out in a node v' of our synopsis all `book` elements that are reached through the label path `publisher/cs/textbooks/book`, and that we have both an "expensive" and a "cheap" publisher in our database; then, clearly, the prices under node v' are correlated to the names at the `publisher` node that lead to them. Thus, the selectivity of the path expression `publisher{name = X}/cs/textbooks/book{price>$100}` estimated at v' can be very different depending on whether `X` = "expensive" or `X` = "cheap".

Correlations in (flat) relational-data synopses are typically modeled using concise, *multidimensional* representations (e.g., histograms, wavelets) for the joint distribution of correlated attributes [Deshpande et al. 2001; Poosala and Ioannidis 1997; Vitter and Wang 1999]. As the above discussion demonstrates, the graph-structured nature of XML data poses additional challenges for the effective summarization of element-value distributions in an XSKETCH, as we need to capture correlations across both data *values* and data *structure*. More specifically, consider the set of all elements in the extent of a specific XSKETCH node v. Different subsets of elements in `extent`(v) (with, possibly, different value characteristics) may be reachable by different label paths in the data (path/value correlations) and different value predicates on different labels on the path (value correlations). Clearly, keeping separate joint-distribution information (e.g., multidimensional histograms or wavelet synopses) for subsets of elements in `extent`(v) for all possible combinations of incoming label paths and value-predicate assignments is impractical—the number of combinations

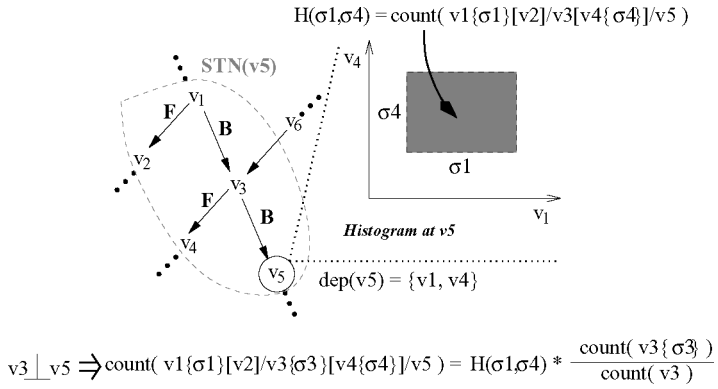


Fig. 3. Example XSKETCH value-distribution information: the histogram at v_5 records correlations along a subset $dep(v_5) = \{v_1, v_4\}$ of $STN(v_5)$.

is simply too large and, for accurate estimates, we would also need to capture overlap information between such subsets of $\texttt{extent}(v)$, thus exploding the complexity and space requirements of the synopsis. Instead, our XSKETCH nodes capture joint-distribution information only along paths and branches of the synopsis that are *common to all elements* in a node's extent. Consider, for instance, the sample XSKETCH synopsis shown in Figure 3. Assume that all nodes correspond to elements with values and consider node v_5. Given the stability properties of edges in its neighborhood, it is straightforward to verify that each element in v_5 has ancestors in nodes v_1 and v_3, which in turn have at least one descendant in nodes v_2 and v_4, respectively; on the other hand, only a subset of elements in $\texttt{extent}(v_5)$ have an ancestor in v_6. Our proposed approach is to record value correlations for the elements of v_5 only within the neighborhood $\{v_1, v_2, v_3, v_4, v_5\}$ that essentially defines a set of common paths and branches; correlations to v_6 are not recorded, since they correspond to only a subset of elements in $\texttt{extent}(v_5)$. More concretely, we introduce the concept of the *stable twig neighborhood* of each node, formally defined as follows:

Definition 3.4. Let v be an XSKETCH node, and let $B(v)$ denote the set of all nodes in the XSKETCH that reach v through a B-stable path (including v itself). Also, let $F(v)$ denote the set of all nodes in the XSKETCH that can be reached starting from any node in $B(v)$ through an F-stable path. The *stable twig neighborhood (*$\texttt{STN}$*)* of v is defined as $\texttt{STN}(v) = B(v) \cup F(v)$.

The key observation here is that, by virtue of stability, the stable twig neighborhood of an XSKETCH node v captures path and branching structure that is common to all data elements in $\texttt{extent}(v)$. The joint-distribution information recorded for v in the XSKETCH tries to capture the frequencies of elements in $\texttt{extent}(v)$ and their possible correlation(s) with the values of (a subset of) other nodes in $\texttt{STN}(v)$ along specific paths (or, in general, twigs) within $\texttt{STN}(v)$. In other words, our XSKETCH synopses rely on (a) *structural B- and F-stability* to model the dependence of element values on path and branching structure (i.e., path/value correlations), and (b) *multidimensional distribution synopses (e.g.,*

histograms) to model value correlations within stable "neighborhoods" of the XSKETCH.

Consider an XSKETCH node v and let T be a twig contained within $\mathtt{STN}(v)$. Let $\mathtt{dep}(v)$ ($\subseteq \mathtt{STN}(v)$) denote the "correlation scope" of v; that is, the set of nodes in the XSKETCH for which correlations with the element distribution in $\mathtt{extent}(v)$ are captured in the joint-distribution information maintained in v. (If v itself contains elements with values then $\mathtt{dep}(v)$ must contain at least v.) We also let $\mathtt{dep}(v, T)$ denote the restriction of $\mathtt{dep}(v)$ in T (i.e., $\mathtt{dep}(v, T) = \mathtt{dep}(v) \cap T$). Then the joint-distribution information recorded in v can give direct estimates for the number of v elements that are reached through the branching path expression $T\{\overline{\sigma}\}$, where the value predicates $\overline{\sigma}$ can be on any subset of $\mathtt{dep}(v, T)$. As a more concrete example, consider the twig $T = v_1[v_2]/v_3[v_4]/v_5$ (shown in Figure 3) that is contained within $\mathtt{STN}(v_5)$, and assume $\mathtt{dep}(v_5) = \{v_1, v_4\}$; then the joint-distribution kept at v_5 can be used to directly estimate the number of v_5 elements discovered by $v_1\{\sigma_1\}[v_2]/v_3[v_4\{\sigma_4\}]/v_5$, where σ_1, σ_4 are value predicates on nodes v_1 and v_4, respectively. We can now give a formal definition for our model of (structure and value) XSKETCH synopses for XML data.

Definition 3.5. An XSKETCH synopsis $\mathcal{XS}(G)$ for an XML data graph G *with element values* is a structural XSKETCH $(V_{\mathcal{XS}}, E_{\mathcal{XS}})$ for G, where each node $v \in V_{\mathcal{XS}}$ can also contain a (possibly) multidimensional synopsis for the joint distribution of elements in $\mathtt{extent}(v)$ and the values of any subset of XSKETCH nodes $\mathtt{dep}(v) \subseteq \mathtt{STN}(v)$ along specific paths/twigs within $\mathtt{STN}(v)$.

Note that, by the above definition, $\mathtt{dep}(v) \subseteq \mathtt{STN}(v)$ since, in general, only a subset of the value distributions in a node's STN will actually be correlated, and it is only these correlations (along a given stable twig) that we need to capture in the XSKETCH node. For noncorrelated value distributions, an *independence assumption* is valid and will give accurate estimates [Deshpande et al. 2001]. Consider once again our example stable twig $T = v_1[v_2]/v_3[v_4]/v_5$ with $\mathtt{dep}(v_5) = \{v_1, v_4\}$, and assume that the distribution of v_5 elements is independent of the values in $v_3 \notin \mathtt{dep}(v_5)$ (i.e., $v_3 \perp v_5$). Even though the distribution information in v_5 cannot directly give an estimate for the count of $v_1\{\sigma_1\}[v_2]/v_3\{\sigma_3\}[v_4\{\sigma_4\}]/v_5$ (which contains a value predicate on v_3), a product estimate based on the independence of the distributions in v_5 and v_3 is going to give a good approximation (Figure 3).

3.2.1 *Value-Distribution Summaries for* XSKETCH *Nodes.* Thus far, we have been deliberately vague about the exact form of distribution summaries maintained in the nodes of our XSKETCH synopsis. A main reason for this is that, as mentioned earlier, several forms of multidimensional data synopses (e.g., histograms or wavelets) can be used to produce a concise description of the distribution of elements in a node's extent across the values in its correlation scope. More specifically, let v be an XSKETCH node and let $\mathcal{D}_v$ denote the cross-product of the value domains for elements in v's correlation scope, that is, $\mathcal{D}_v = \times_{u \in \mathtt{dep}(v)} domain(u)$. Then, the distribution of v's elements within its correlation scope can be described by the joint-frequency table $f_v[c_1, \ldots, c_k]$ that gives the number of elements in $\mathtt{extent}(v)$ that are reached from (or, lead

to) the tuple of values $(c_1, \ldots, c_k) \in \mathcal{D}_v$ in the corresponding nodes of $\texttt{dep}(v)$. This frequency table $f_v[]$ can be summarized in our XSKETCH using conventional multidimensional histograms [Poosala and Ioannidis 1997], Haar wavelets [Vitter and Wang 1999], or even more complex summarization techniques based on statistical modeling [Deshpande et al. 2001]. For concreteness, we use the term *histogram* to refer to the distribution information maintained in XSKETCH nodes in the remainder of this article.

Unlike the relational case, however, the joint-frequency table $f_v[]$ is *not* sufficient to provide accurate estimates for arbitrary value-selection predicates over the nodes of $\texttt{dep}(v)$. The hierarchical nature of XML data once again introduces novel challenges in capturing the element distribution in a node with respect to other nodes in its STN. As a simple example, consider a v-labeled node with 10 u-labeled children $(u_1, \ldots, u_{10})$ in the data graph, and nine other v-labeled nodes all with a single u-labeled child (u_{11}). Further, assume that v nodes carry no values whereas each u_i node carries a value of i $(i = 1, \ldots, 10)$. Clearly, in our XSKETCH synopsis this simple data configuration will result in a single B/F-stable (v, u) edge with $\texttt{count}(v) = \texttt{count}(u) = 10$. Assume that $\texttt{dep}(v) = \{u\}$. Then the joint-frequency table $f_v[]$ will have entries $f_v[i] = 1$ for $i = 1, \ldots, 10$ and $f_v[11] = 9$. Now consider the branch query $v[u\{\sigma\}]$ where $\sigma = (1 \leq \texttt{value}(u) \leq 10)$. Clearly, the correct $\texttt{count}(v[u\{\sigma\}])$ is 1; however, using the $f_v[]$ table in the conventional manner to estimate the selectivity of σ (assuming no summarization whatsoever), we get an erroneous count of 10. The reason, of course, is *"double counting"*: even though the frequency table tells us that each u-value from 1 to 10 is reached by one v-element, it has no way of telling us that they are in fact reached by the *same* v-element!

It is easy to see that this double-counting problem can become much more complicated when the element distribution involves more complex path structures and overlap patterns between elements. Unfortunately, this problem is inherent in all approximation techniques that estimate the selectivity of ranges by summing point frequencies and there is no easy solution based on traditional frequency tables and histogram structures.[3] It is thus necessary to use specialized summarization techniques that take directly into account the existence of duplicates in the estimated cardinalities. In what follows, we describe two approaches for dealing with double-counting: the first one is based on *distinct sampling* [Gibbons 2001], while the second one is a novel summarization method called *range histograms*, which avoids double-counting in XSKETCH nodes by explicitly capturing the overlap between different value ranges.

—*Distinct sampling*. In a recent study, Gibbons has introduced *distinct sampling* [Gibbons 2001], a summarization method for estimating the cardinality of DISTINCT projections over relational queries. Even though this technique has been proposed for relational data, we observe that the relational estimation problem is essentially the same as estimating

[3]A naive approach would be to incorporate element-id information in the dimensions of our frequency table. Such an approach, however, is very inflexible with respect to updates in the database and (perhaps most importantly) it is not at all clear how element-id axes should be handled during the summarization (histograming) of the table.

the selectivity of range predicates over graph-structured data. To illustrate this, consider the previous example and let $R_{vu}(A_v, A_u) = \{(v_1, u_1), \ldots, (v_1, u_{10}), (v_2, u_{11}), \ldots, (v_{10}, u_{11})\}$ be a relation of two attributes that represents the structural join of nodes v and u , that is, all pairs (v_i, u_j) such that v_i is a parent of u_j. It is straightforward to show that the selectivity of path $v[u\{\sigma\}]$ is equal to the result cardinality of the "relational" query (SELECT DISTINCT A_v FROM R_{uv} WHERE $\sigma(A_u)$). We can thus employ a fairly straightforward adaptation of distinct sampling in order to implement the value summaries for XSKETCH nodes and avoid the double counting problem.

—*Range histograms.* We propose an alternative solution for avoiding double-counting in XSKETCH nodes for the special case of simple value-range predicates. Our main idea is to increase the dimensionality of the frequency table $f_v[]$ so that its entries capture the number of v elements within *a range of values* of its reference nodes. Thus, in our "bad" example above, the frequency table for v would be two-dimensional and $f_v[lb, ub]$ would be the number of v elements reaching u values in the interval [*lb,ub*], where $lb \leq ub$; then, of course, $f_v[1, 10] = 1$ would give the correct answer to our example query. In general, an n-dimensional value distribution will generate a $2n$-dimensional range distribution which can be used to accurately estimate the number of elements that satisfy any conjunction of range predicates over the corresponding dimensions. The resulting range-frequency tables can be summarized to give "range histograms" that can provide accurate estimates to queries with range predicates on values. We note that existing summarization techniques, such as wavelets [Vitter and Wang 1999] or histograms [Poosala and Ioannidis 1997], can provide a very effective approximation of the range-frequency table, as adjacent cells correspond to ranges with high overlap and are thus likely to have similar frequencies (i.e., the distribution of frequency counts is expected to be "smooth.")

3.3 Problem Formulation: Construction and Usage of Effective XSKETCHes

Given an amount of space (i.e., size limit) for the XSKETCH synopsis (determined, for instance, by optimizer time and space constraints), we are obviously interested in determining the "most effective" XSKETCH within our specified space budget. Of course, the notion of "effectiveness" for an XSKETCH synopsis needs to be defined based on an *estimation framework* that uses the summary information in the synopsis to parse an input complex path expression and produce an (approximate) selectivity estimate. Given the concise and approximate nature of our XSKETCHes, this estimation process obviously has to rely on a set of *statistical assumptions* that compensate for the lack of detailed information (similar, for example, to the intrabucket uniformity assumptions typically made during histogram-based estimation [Poosala and Ioannidis 1997; Poosala et al. 1996]). Thus, our XML-graph summarization problem comprises two important and interrelated challenges that are addressed in the remainder of this article:

(1) *Estimation framework for complex path queries over* XSKETCH *graph synopsest*. Given a complex, branching path expression Q and an XSKETCH synopsis $\mathcal{X}S(G)$ representing a statistical summary of a large XML data

graph G, process Q over $\mathcal{X}S(G)$ (using a set of well-founded statistical assumptions) to produce an estimate for the selectivity of P over the original data graph G.

(2) *Effective* XSKETCH *synopsis construction*. Given a large XML data graph G and a space budget of B bytes, build an XSKETCH graph synopsis $\mathcal{X}S(G)$ of G that effectively minimizes the approximation error in the selectivity estimates produced based on $\mathcal{X}S(G)$ (and the given estimation model) for complex path expressions over G.

In this work, we tackle these difficult problems for the general case of recursive XPath expressions (with value and branching predicates) over nonrecursive graph-structured data; moreover, our techniques extend to the dual case of nonrecursive queries over recursive graph-structured data. As we show in this article, the summarization problem is already hard for these combinations and requires the development of nontrivial solutions. The existence of combined query- and data-recursion is of course an interesting case, albeit one with significant complications in the computation of accurate selectivity estimates. We revisit this point in Section 4.3, where we describe in more detail the challenges involved in general recursion.

4. XSKETCH ESTIMATION FRAMEWORK

We now define our estimation framework for approximating path-expression selectivities over a compact XSKETCH synopsis. At an abstract level, the proposed framework operates in two steps. The first step maps Q to a set of *synopsis embeddings*, that is, synopsis subgraphs that match the navigation steps of Q and hence correspond to elements that appear in the result of Q. The second step estimates the number of results in each unique embedding and approximates $sel(Q)$ as the sum of these individual selectivities. Our selectivity estimation problem, therefore, essentially reduces to two subproblems: (a) computing the set of synopsis embeddings for Q, and (b) estimating the selectivity for each discovered embedding. The following sections describe the details of our approach. We initially present an algorithm for computing embeddings (Section 4.1), and then describe a framework for estimating the selectivity of individual embeddings (Section 4.2). To simplify the presentation, the aforementioned sections focus on the case of recursive queries over nonrecursive data; we consider recursive data in Section 4.3.

4.1 Computing Path-Expression Embeddings

At a high level, a synopsis embedding of Q instantiates each query edge (q_i, q_j) with a simple synopsis path that respects the corresponding structural and value constraints. The result is a subgraph of the synopsis on which Q has a potentially nonempty result.

More formally, we first introduce the concept of a *root binding* between the nodes of Q and the nodes of an XSKETCH $\mathcal{X}S(G)$. We say that $u \in \mathcal{X}S(G)$ is a root binding for $q_i \in Q$ if (a) $\texttt{label}(u)$ matches $\texttt{label}(q_i)$ and predicate σ_i has nonzero selectivity according to the histogram at u, and (b) for every edge (q_i, q_j)

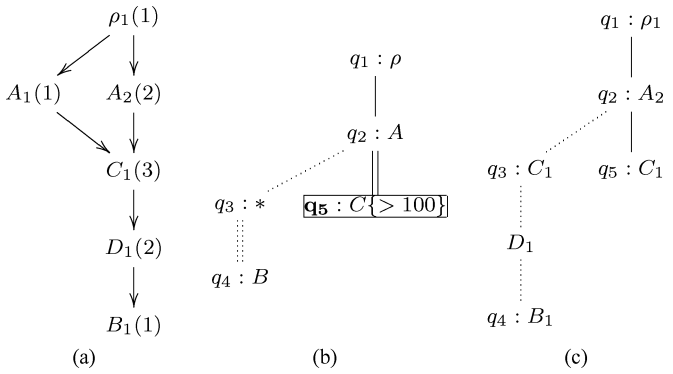


Fig. 4. An example of a query embedding: (a) XSKETCH synopsis, (b) query Q, (c) a possible embedding.

with $axis(q_i, q_j) = /$(respectively, $axis(q_i, q_j) = //$), u has a child $v \in \mathcal{X}S(G)$ (respectively, a descendant $v \in \mathcal{X}S(G)$) that is a root binding for q_j. We call the path between u and v a *witness* of the binding between q_i and u. Intuitively, a root binding implies that the subquery $Q(q_i)$ will have a positive selectivity if it is evaluated starting from the elements of u. A synopsis embedding is simply a pair of mappings (h_n, h_p), where h_n is a mapping from query nodes to synopsis nodes such that $h_n(q_i)$ is a root binding for q_i, and h_p is a mapping between query edges to synopsis paths such that $h_p(q_i, q_j)$ is a witness path between $h_n(q_i)$ and $h_n(q_j)$.

Example 4.1. Figure 4 shows an example embedding for a sample XSKETCH synopsis and query Q. The depicted embedding corresponds to the following mappings: $h_n(q_1) = \rho_1$, $h_n(q_2) = A_2$, $h_p(q_1, q_2) = \rho_1.A_2$, $h_n(q_3) = C_1$, $h_p(q_2, q_3) = A_2.C_1$, $h_n(q_4) = B_2$, $h_p(q_3, q_4) = C_1.D_1.B_2$, $h_n(q_5) = C_1$, $h_p(q_2, q_5) = A_2.C_1$. (Our assumption is that the histogram of C_1 yields a nonzero selectivity for predicate > 100.) The embedding is presented as a tree of synopsis nodes, with a subset of the nodes annotated with their mapped query node. It is interesting to note that the same node C_1 appears in two parts in the embedding, namely, the branching predicate and the main branch. Intuitively, this simply implies that the elements of C_1 can appear in the evaluation of two different steps of the query.

Intuitively, an embedding specifies the nodes whose elements correspond to different query steps and the paths that are followed in between; moreover, the "abstract" information of a query, namely, the wildcard labels and the edges with the descendant axis (//), is substituted with *simple* synopsis paths that match the corresponding constraints. We define the selectivity of an embedding as the count of generated results when the evaluation of the query is constrained to the specified element sets and data paths. Returning to the example of Figure 4, the selectivity of the embedding is defined as the number of elements in $\texttt{extent}(C_1)$ that (a) are reached by a root-to-element path passing through nodes $\rho_1.A_2.C_1$, and (b) their ancestor in A_2 has at least one outgoing path that passes through nodes $C_1.D_1.B_1$. Essentially, estimating the selectivity of an embedding is equivalent to reasoning about the properties of backward and

```
procedure COMPUTEEMBEDDINGS( XS(G), Q, v, TS)
Input: XSKETCH synopsis XS(G); query Q; synopsis node v; depth-first traversal stack TS
begin
1.  if ( visit[v] = ongoing ) return
2.  if ( visit[v] ≠ completed ) then
3.      visit[v] := ongoing; bind[v] := ∅; cand[v] := {q_j ∈ Q | v matches q_j}
4.  TS.push(v)
5.  for each c ∈ children(v) do COMPUTEEMBEDDINGS(XS(G),Q,c,TS)
6.  TS.pop()
7.  if ( visit[v] ≠ completed ) then
8.      for each q_j ∈ cand[v] do
9.          if ( ∄q_k ∈ children(q_j) : paths[v, q_j, q_k] = ∅ ) then bind[v]+ = q_j
10. visit[v] := completed
11. endif
12. for each ancestor u ∈ TS, q_i ∈ cand[u] do
13.     if ∃q_j ∈ bind[v] : q_j ∈ children(q_i) and TS[u..v] matches axis(q_i, q_j) then
14.         paths[u, q_i, q_j]+ = TS[u..v]
end
```

Fig. 5. Algorithm for computing query embeddings.

forward element-connectivity along simple paths. As we discuss in the next section, this general observation forms the basic principle behind our proposed estimation algorithm.

We now shift our attention to the key problem of identifying the embeddings of a query over a concise XSKETCH synopsis. We present an algorithm, termed COMPUTE-EMBEDDINGS, that identifies the complete set of synopsis embeddings with a single depth-first traversal of the synopsis graph. The key idea is to examine synopsis nodes in a postorder fashion and compute the root bindings for each node based on the bindings of its descendants. The pseudocode of COMPUTEEMBEDDINGS is shown in Figure 5. We assume that the initial call to COMPUTEEMBEDDINGS is performed on the root node of the synopsis. When a node v is visited for the first time, the algorithm computes a set $cand[v]$ of query nodes that match with v, that is, their labels are compatible and σ_i has a positive selectivity on v. Essentially, these are the candidate nodes for the root bindings of v. Subsequently, the algorithm recurses on the children of v (Steps 4–6), thus discovering the root bindings of its descendants and the witness paths that link v to the latter. This information is used to filter the set of candidate nodes $cand[v]$ and determine the set $bind[v] \subseteq cand[v]$ of its root bindings. Finally, the filtered nodes in $bind[v]$ are used to compute witness paths for the ancestors of v, using the stack TS of the depth-first traversal.

A key step of the algorithm is the identification of set $bind[v] \subseteq cand[v]$. More concretely, the recursive call to a descendant w has two effects: (a) it determines the root bindings for w, and (b) it computes the witness paths for edges (q_j, q_k), where $q_j \in cand[v]$ is a candidate and $q_k \in bind[w]$ is a root binding for w. This second step is performed at the end of the recursive call to w, by examining the traversal stack TS and determining whether the simple synopsis path $TS[v..w]$ (the path between v and w in the stack) matches $axis(q_j, q_k)$. If the answer is positive, then $TS[v..w]$ is added to a variable $paths[v, q_j, q_k]$ that accumulates

the witnesses of the binding between v and q_j. Clearly, v is a root binding for q_j if the witness set $path[v, q_j, q_k]$ is nonempty for every edge (q_j, q_k); moreover, the embeddings that map q_j to v can be recovered by chasing through the contents of $paths$ for v and its descendants. Hence, the algorithm computes all the embeddings of the query after processing the root synopsis node and determining whether it is a root binding for q_1.

Example 4.2. Consider again the application of COMPUTEEMBEDDINGS on the example synopsis and query of Figure 4. Assume that the depth-first traversal has followed the path $\rho_1.A_2.C_1.D_1.B_1$ and the algorithm has just finished processing node B_1. The following describes the information associated with each node in the traversal stack:

$$
\begin{array}{lcl}
cand[\rho_1] = \{q_1, q_3\} & \rightarrow & paths[\rho_1, q_3, q_4] = \{\rho 1.A_2.C_1.D_1.B_1\} \\
cand[A_2] = \{q_2, q_3\} & \rightarrow & paths[A_2, q_3, q_4] = \{A_2.C_1.D_1.B_1\} \\
cand[C_1] = \{q_3, q_5\} & \rightarrow & paths[C_1, q_3, q_4] = \{C_1.D_1.B_1\} \\
cand[D_1] = \{q_3\} & \rightarrow & paths[D_1, q_3, q_4] = \{D_1.B_1\} \\
bind[B_1] = \{q_4\} & &
\end{array}
$$

Based on this information, the algorithm can determine that D_1 and C_1 are root bindings for q_3, since they match its label and have a witness path to a root binding of q_4. (By the same token, C_1 is also a root binding for q_5.) After finishing the processing of C_1, the algorithm has updated the binding sets for D_1 and C_1, and the witness sets of their ancestors in the traversal stack:

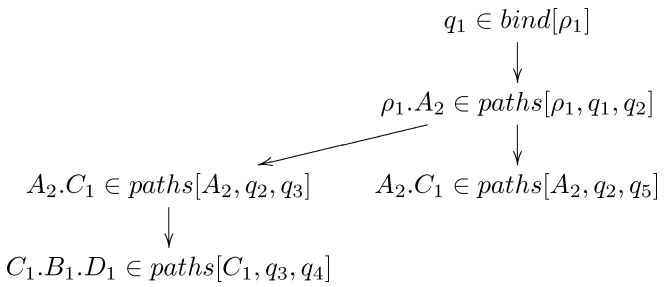
$$
\begin{array}{ll}
cand[\rho_1] = \{q_1, q_3\} & \rightarrow paths[\rho_1, q_3, q_4] = \{\rho 1.A_2.C_1.D_1.B_1\}, \\
cand[A_2] = \{q_2, q_3\} & \rightarrow paths[A_2, q_3, q_4] = \{A_2.C_1.D_1.B_1\}, \\
 & \rightarrow paths[A_2, q_2, q_3] = \{A_2.C_1\} \quad paths[A_2, q_2, q_5] = \{A_2.C_1\} \\
bind[C_1] = \{q_3, q_5\} & \rightarrow paths[C_1, q_3, q_4] = \{C_1.D_1.B_1\} \\
bind[D_1] = \{q_3\} & \rightarrow paths[D_1, q_3, q_4] = \{D_1.B_1\} \\
bind[B_1] = \{q_4\} &
\end{array}
$$

At this point, it can be determined that A_2 is a root binding for both q_2 and q_3. This also implies that ρ_1 has a witness path for its binding to q_1 and hence $\rho_1.A_2$ will be inserted in $paths[\rho_1, q_1, q_2]$. The final state of the algorithm, after processing all nodes, is as follows:

$$
\begin{array}{ll}
bind[\rho_1] = \{q_1, q_3\} & \rightarrow paths[\rho_1, q_3, q_4] = \{\rho_1.A_2.C_1.D_1.B_1,\ \rho_1.A_1.C_1.D_1.B_1\} \\
 & \rightarrow paths[\rho_1, q_1, q_2] = \{\rho_1.A_2,\ \rho_1.A_1\} \\
bind[A_1] = \{q_2, q_3\} & \rightarrow paths[A_1, q_3, q_4] = \{A_1.C_1.D_1.B_1\} \\
 & \rightarrow paths[A_1, q_2, q_3] = \{A_1.C_1\} \quad paths[A_1, q_2, q_5] = \{A_1.C_1\} \\
bind[A_2] = \{q_2, q_3\} & \rightarrow paths[A_2, q_3, q_4] = \{A_2.C_1.D_1.B_1\} \\
 & \rightarrow paths[A_2, q_2, q_3] = \{A_2.C_1\} \quad paths[A_2, q_2, q_5] = \{A_2.C_1\} \\
bind[C_1] = \{q_3, q_5\} & \rightarrow paths[C_1, q_3, q_4] = \{C_1.D_1.B_1\} \\
bind[D_1] = \{q_3\} & \rightarrow paths[D_1, q_3, q_4] = \{D_1.B_1\} \\
bind[B_1] = \{q_4\} &
\end{array}
$$

The individual embeddings can be recovered by starting from the root binding of q_1 and chasing to the other query nodes based on the contents of $paths$.

The embedding of Figure 4, for instance, is generated from the following information:



On a final note, we observe that COMPUTEEMBEDDINGS does not follow any cycles in the synopsis (Step 1) and hence generates witness paths of unique synopsis nodes. This is in agreement with our assumption of nonrecursive data, as a cyclic witness path would essentially imply a cycle in the evaluation of the query. Moreover, this restriction bounds the number of embedding nodes by $N_Q N_S$, since every root-to-leaf path in the embedding must contain unique synopsis nodes. (The complexity of our COMPUTEEMBEDDINGS procedure is discussed in the Electronic Appendix.) We revisit these issues in Section 4.3, where we discuss the extension of our approach to recursive data.

4.2 Estimating Embedding Selectivity

Having described the COMPUTEEMBEDDINGS algorithm for identifying the embeddings of a query, we now introduce an estimation framework for approximating the selectivity of each embedding. As discussed earlier, our goal is to approximate the selectivity $sel(Q)$ of the query as the sum of selectivities of its identified embeddings.

To simplify our presentation, we will assume that Q has a single branching predicate that contains a linear path. (The extension of our analysis to multiple branching predicates is straightforward and is discussed later.) We will use $\overline{v}\{\overline{\sigma}\} = v_1/\cdots/v_n[u_1/\cdots/u_k]/v_{n+k+1}/\cdots/v_{n+k+m}$ to denote an embedding of Q, with $\overline{v}$ referring to the label structure of the embedding (i.e., without the value predicates). Based on this notation, the example of Figure 4 would be written as $\rho_1/A_2[C_1/B_1/D_1]/C_1\{> 100\}$, with $\overline{v}$ corresponding to $\rho_1/A_2[C_1/B_1/D_1]/C_1$. Clearly, the selectivity $sel(\overline{v}\{\overline{\sigma}\})$ of the embedding is equal to the number of data elements for which (a) there exists a root-to-element data path that passes through nodes $v_1/\ldots/v_{n+k+m}$ and satisfies all the selection predicates, and (b) the ancestor that corresponds to v_n is the root of at least one data path that passes through nodes $v_n/u_1/\ldots/u_k$. We express this as $sel(\overline{v}\{\overline{\sigma}\}) = \texttt{count}(v_{n+k+m}) \cdot f(\overline{v}\{\overline{\sigma}\})$, where $f(\overline{v}\{\overline{\sigma}\})$ denotes the estimated *fraction* (i.e., empirical probability) of result elements in $\texttt{extent}(v_{n+k+m})$.

As our discussion indicates, the estimation problem is reduced to approximating the fraction $f(\overline{\{}\sigma\})$ for every embedding of the query. In what follows,

we introduce a framework for estimating the required fractions based on the information stored in the XSKETCH summary $\mathcal{X}S(G)$.

4.2.1 *Parsing Path-Expression Embeddings over an* XSKETCH. Our proposed estimation process is based on the parsing of an embedding in subtwigs, for which estimation accuracy is guaranteed by virtue of the recorded edge stabilities. The following theorem formalizes this notion, demonstrating that the element counts estimated at the two endpoints of a label path embedding in $\mathcal{X}S(G)$ are guaranteed to be *exact* as long as all the edges followed in $\mathcal{X}S(G)$ satisfy the appropriate stability conditions. (The proof can be found in the Electronic Appendix.)

THEOREM 4.1. *Let $\mathcal{X}S(G)$ be an* XSKETCH *synopsis for an XML data graph G, and let $v_1, \ldots, v_n$ be a directed path in $\mathcal{X}S(G)$.*

(1) *If* $\mathbf{B} \in \texttt{label}(v_i, v_{i+1})$ *for each* $i = 1, \ldots, n-1$, *then* all $\texttt{count}(v_n)$ *elements corresponding to v_n are discovered by the label path* $\texttt{label}(v_1)/\cdots/\texttt{label}(v_n)$ *starting from some node in* $\texttt{extent}(v_1)$ *in G.*
(2) *If* $\mathbf{F} \in \texttt{label}(v_i, v_{i+1})$ *for each* $i = 1, \ldots, n-1$, *then* all $\texttt{count}(v_1)$ *elements corresponding to v_1 reach at least one element in* $\texttt{extent}(v_n)$ *by the label path* $\texttt{label}(v_1)/\cdots/\texttt{label}(v_n)$ *in G.*

Theorem 4.1 ensures that the estimates obtained from an XSKETCH for a (structure-only) label path expression are accurate as long as all the edges traversed in the synopsis while parsing the path expression satisfy the appropriate stability constraints (recall that an XSKETCH with all edges labeled $\{\mathbf{B}, \mathbf{F}\}$ is exactly the perfect synopsis, i.e., the B/F-bisimilar graph.) Of course, given the hard space constraints that the XSKETCH synopsis must satisfy, it is impossible to guarantee such an ideal parsing for all possible path expressions over the data graph; in addition, the theorem covers the case of *structural* embeddings, that is, embeddings without value predicates, and does not consider the correlation between value distributions and path structure. We address these issues in the remainder of this section, where we introduce an estimation framework for approximating the selectivities of complex path embeddings over XSKETCHes. As with any form of estimation that uses concise synopses (e.g., histograms or wavelets), our proposed framework also relies on a set of statistical (uniformity and independence) assumptions to compensate for the lack of detailed distribution information.

Consider a single-branch path embedding $\overline{v} = v_1/\cdots/v_n[u_1/\cdots/u_k]/v_{n+k+1}/\cdots/v_{n+k+m}$ (note that we are using u_i for branch nodes to simplify the notation). The first step in our estimation process is to parse the embedding $\overline{v}$ into a sequence of *maximal, nonoverlapping stable twigs*; that is, we break the embedding $\overline{v}$ into a collection of subtwigs $T_1, T_2, \ldots$, such that every XSKETCH node in the embedding is "covered" by exactly one of the T_i's, and each T_i is a maximal stable twig in $\overline{v}$, that is, all path (branch) edges in T_i are B-stable (respectively, F-stable). Conceptually, this parsing can be done as follows. Starting from the last node in the embedding (v_{n+m}), build the first tree T in the decomposition by

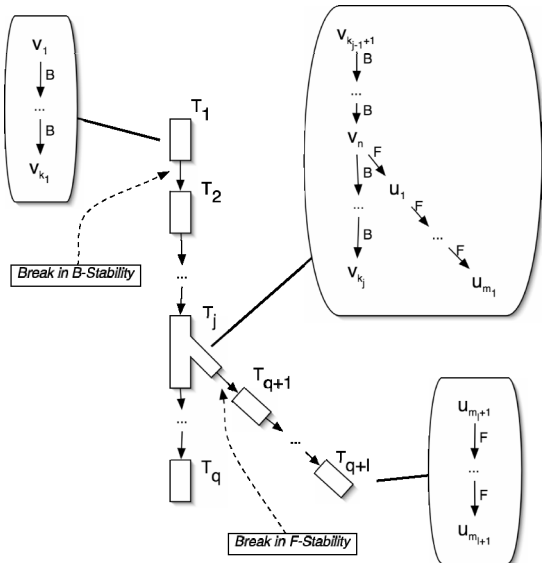


Fig. 6. Parsing of single-branch embedding into multiple stable sub-embeddings.

taking the intersection of the embedding $\overline{v}$ with the stable twig neighborhood of v_{n+m}; that is, $T = \texttt{STN}(v_{n+m}) \cap \overline{v}$. Then, take the nodes of T out of $\overline{v}$ and repeat the process from those nodes of $\overline{v} - T$ that were directly connected to T nodes (i.e., at the outside "border" of T).

It is easy to see that, for the case of our single-branch embedding $\overline{v}$, all the stable twigs resulting from the above decomposition will be simple paths except perhaps for the single twig, say T_j, that contains the only branching node v_n. More concretely, let the stable-twig decomposition of $\overline{v}$ be as follows: $T_1 = v_1/\ldots/v_{k_1}, \ldots, T_j = v_{k_{j-1}+1}/\ldots/v_n[u_1/\ldots/u_{m_1}]/\ldots/v_{k_j}, \ldots, T_q = v_{k_{q-1}+1}/\ldots/v_{k_q}, T_{q+1} = u_{m_1+1}/\cdots/u_{m_2}, \ldots, T_{q+l} = u_{m_l+1}/\cdots/u_{m_{l+1}}$, where $0 < k_1 < k_2 < \cdots < n+m = k_q$ and $0 < m_1 < m_2 < \cdots < k = m_{l+1}$. Note that, in the above decomposition, the stable twigs $T_1, \ldots, T_q$ essentially cover the main path of the expression (with the only "true" twig T_j possibly covering part of the branch), whereas twigs $T_{q+1}, \ldots, T_{q+l}$ cover the remainder of the $u_1/\cdots/u_k$ branch. (This is shown pictorially in Figure 6.) The decomposition can be computed efficiently by performing a single pass over the embedding and marking the breaks of B-stability (respectively, f-stability) for the edges pointing to v_i nodes (respectively u_j nodes). The time complexity is therefore $O(N_Q N_S)$, since any embedding can have at most $N_Q N_S$ nodes (Section 4.1). We note that our definition of decomposition assumes a single linear-path branching predicate, but it extends naturally to multiple tree-structured branching predicates. The only difference is that the decomposition may result in several twig components, instead of a single twig T_j.

Given this decomposition of the embedding $\overline{v}$, we now employ the well-known *chain rule* from probability theory [Feller 1968] to rewrite the required fraction $f(\overline{v}\{\overline{\sigma}\})$ as follows (for simplicity, we use $\overline{\sigma}_i$ for the set of all predicates in twig

T_i and π_j for the predicate on branch node u_j):

$$\begin{aligned} f(\overline{v}\{\overline{\sigma}\}) \;=\; & f(T_q\{\overline{\sigma}_q\}) \cdot \prod_{i=j}^{q-1} f(T_i\{\overline{\sigma}_i\}/v_{k_i+1} \mid T_{i+1}\{\overline{\sigma}_{i+1}\}/\cdots/T_q\{\overline{\sigma}_q\}) \\ & \cdot \prod_{i=1}^{l} f(u_{m_i}[T_{q+i}\{\overline{\sigma}_{q+i}\}] \mid T_j\{\overline{\sigma}_j\}[u_{m_1+1}\{\pi_{m_1+1}\}/\cdots \\ & \qquad \cdots/u_{m_i}\{\pi_{m_i}\}]/T_{j+1}\{\overline{\sigma}_{j+1}\}/\cdots/T_q\{\overline{\sigma}_q\}) \\ & \cdot \prod_{i=1}^{j-1} f(T_i\{\overline{\sigma}_i\}/v_{k_i+1} \mid T_{i+1}\{\overline{\sigma}_{i+1}\}/\cdots/T_j\{\overline{\sigma}_j\}[u_1\{\pi_1\}/\cdots/u_k\{\pi_k\}]/\cdots/T_q\{\overline{\sigma}_q\}), \end{aligned}$$

where the notation $f(T\{\overline{\sigma}\}/v \mid v/T'\{\overline{\sigma}'\})$ denotes the *conditional probability* that a data element in a synopsis node v is discovered by the embedding $T\{\overline{\sigma}\}/v$ given that there exists an embedding $v/T'\{\overline{\sigma}'\}$ "rooted" at that element. (The corresponding conditional fractions for branches $f(v[T'\{\overline{\sigma}'\}] \mid T\{\overline{\sigma}\}/v)$ are defined similarly.) Note that the second product term above captures the selectivity of the complex expression along the "existential" branch $u_1/\cdots/u_k$, whereas the first and third product terms capture the selectivity along the main path. Of course, by the chain rule, the $f()$ frequency terms always condition on the remainder of the complex path as it is parsed in a "bottom-up" fashion; intuitively, the reason for this conditioning is to capture the *correlations* between the various twigs that comprise the overall path embedding $\overline{v}\{\overline{\sigma}\}$. Also, note that the value predicates for the v_{k_i+1} and u_{m_i} nodes are not included in the fraction expressions for the T_i and T_{q+i} terms above, since they are already accounted for in the expression for T_{i+1} and T_{q+i-1} (respectively) and included in the corresponding conditionals for T_i and T_{q+i}. To simplify the above expression, our estimation process makes the following "Twig-Independence" assumption.

Assumption 1 (*Twig Independence*). Given a node v in $\mathcal{X}S(G)$, the distribution of incoming twigs $T\{\overline{\sigma}\}$ to v is *independent* of the distribution of outgoing twigs $T'\{\overline{\sigma}'\}$ from v for any value predicates $\overline{\sigma}$, $\overline{\sigma}'$; more formally, $f(T\{\overline{\sigma}\}/v \mid v/T'\{\overline{\sigma}'\}) \approx f(T\{\overline{\sigma}\}/v)$ and $f(v[T'\{\overline{\sigma}'\}] \mid T\{\overline{\sigma}\}/v) \approx f(v[T'\{\overline{\sigma}'\}])$.

Obviously, the validity of this assumption depends on the underlying data distribution and can directly affect the accuracy of the computed estimates (this is a basic observation that holds for all the statistical assumptions that we introduce). We address this issue in Section 5, where we describe transformations for refining an XSKETCH against possibly erroneous assumptions, thus "tailoring" our synopses to the statistical characteristics of the base data. Returning to our estimation framework, Assumption 1 allows us to eliminate most of the conditionings and simplify our expression for $f(\overline{v}\{\overline{\sigma}\})$ to

$$\begin{aligned} f(\overline{v}\{\overline{\sigma}\}) \;\approx\; & f(T_q\{\overline{\sigma}_q\}) \cdot \prod_{i=1}^{q-1} f(T_i\{\overline{\sigma}_i\}/v_{k_i+1} \mid v_{k_i+1}\{\sigma_{k_i+1}\}) \\ & \cdot \prod_{i=1}^{l} f(u_{m_i}[T_{q+i}\{\overline{\sigma}_{q+i}\}] \mid u_{m_i}\{\pi_{m_i}\}). \end{aligned} \tag{1}$$

That is, the only conditioning that remains over the $f()$ fractions is on the value predicate imposed on the referenced node in our XSKETCH synopsis. Now, let B_{q+i}

denote the *branch expression* corresponding to the F-stable path T_{q+i} in our decomposition of the $u_1/\cdots/u_k$ branch; that is, $B_{q+i} = u_{m_i+1}[u_{m_i+2}/\cdots/u_{m_{i+1}}]$, for $i = 1, \ldots, l$. Applying the chain rule once again for the individual terms in the above products, we have

$$\begin{aligned} f(T_i\{\overline{\sigma}_i\}/v_{k_i+1} \mid v_{k_i+1}\{\sigma_{k_i+1}\}) &= f(v_{k_i}/v_{k_i+1}|v_{k_i+1}\{\sigma_{k_i+1}\}) \cdot f(T_i\{\overline{\sigma}_i\}|v_{k_i}/v_{k_i+1}\{\sigma_{k_i+1}\}) \\ &\approx f(v_{k_i}/v_{k_i+1} \mid v_{k_i+1}\{\sigma_{k_i+1}\}) \cdot f(T_i\{\overline{\sigma}_i\}), \end{aligned} \quad (2)$$

where the last derivation follows from Twig Independence (Assumption 1). Similarly,

$$f(u_{m_i}[T_{q+i}\{\overline{\sigma}_{q+i}\}] \mid u_{m_i}\{\pi_{m_i}\}) \approx f(u_{m_i}[u_{m_i+1}] \mid u_{m_i}\{\pi_{m_i}\}) \cdot f(B_{q+i}\{\overline{\sigma}_{q+i}\}). \quad (3)$$

To simplify the resulting fractions further, we make one more independence assumption that aims to compensate for the lack of statistical-correlation information across nonstable edges.

Assumption 2 (*Edge-Value Independence Across Nonstable Edges*). Consider a node v in $\mathcal{X}S(G)$ and let u (w) be a non-B-stable parent (respectively, non-F-stable child) of v in $\mathcal{X}S(G)$. Then the distribution of incoming (outgoing) edges from u (respectively, to w) across the elements in $\texttt{extent}(v)$ is *independent of the elements' values*; that is, for any predicate σ on the values of v elements, we have $f(u/v \mid v\{\sigma\}) \approx f(u/v)$ and $f(v[w] \mid v\{\sigma\}) \approx f(v[w])$.

Assumption 2 essentially simplifies path-value correlations along nonstable edges. Combining Equations (1)–(3) and Assumption 2, we obtain the following expression for the selectivity estimate of our branching-path embedding:

$$f(\overline{v}\{\overline{\sigma}\}) \approx \prod_{i=1}^{q} f(T_i\{\overline{\sigma}_i\}) \cdot \prod_{i=1}^{l} f(B_{q+i}\{\overline{\sigma}_{q+i}\}) \cdot \prod_{i=1}^{q} f(v_{k_i}/v_{k_i+1}) \cdot \prod_{i=1}^{l} f(u_{m_i}[u_{m_i+1}]).$$

As a final step, we separate the structural and value constraints in the first two terms, and use Theorem 4.1 in order to eliminate the predicates on path structure. More specifically, the chain rule allows us to rewrite term $f(T_i\{\overline{\sigma}_i\})$ as $f(T_i) \cdot f(\overline{\sigma}_i \mid T_i)$, where the first probability captures the structural predicate (elements that are reached by synopsis twig T_i), while the second covers the value predicates (elements that in addition satisfy the constraints in $\overline{\sigma}_i$). Given that T_i represents a stable twig, that is, all edges are backward- and forward-stable, Theorem 4.1 guarantees that $f(T_i) = 1$; that is, the structural predicate holds for every element in the last node of T_i. The branching terms $f(B_{q+i}\{\overline{\sigma}_{q+i}\})$ can be simplified using the same methodology, which leads to the following final expression:

$$f(\overline{v}\{\overline{\sigma}\}) \approx \prod_{i=1}^{q} f(\overline{\sigma}_i \mid T_i) \cdot \prod_{i=1}^{l} f(\overline{\sigma}_{q+i} \mid B_{q+i}) \cdot \prod_{i=1}^{q} f(v_{k_i}/v_{k_i+1}) \cdot \prod_{i=1}^{l} f(u_{m_i}[u_{m_i+1}]). \quad (4)$$

Example 4.3. We present an example of our estimation methodology on the XSketch summary of Figure 2(c). We assume that each actor element records the name of the corresponding actor and, similarly, each movie element contains a value for the respective title. As a result, the XSketch records two value

histograms, namely, H_A for the distribution of names in node A, and H_M for the distribution of movie titles in node M.

Consider the embedding A{n = "x"}[W/L]/MR/IR/M{t = "y"}, where n,t denote the name of an actor and the title of a movie respectively. Based on our discussion, we approximate its selectivity as count(M) $\times$ f(A{n = "x"}[W/L]/MR/IR/M{t = "y"}) (to simplify the description, we use f_Q to denote the fraction term). According to the recorded stabilities, the embedding can be parsed in three stable components: A/MR, W/L, and IR/MR. Applying the Chain rule on this parsing yields the following expression for the fraction term:

$$f_Q = f(\text{IR/M\{t = "y"\}}) \cdot f(\text{MR/IR} \mid \text{IR/M\{t = "y"\}}) \cdot f(\text{A\{n = "x"\}/MR} \mid \text{MR/IR/M\{t = "y"\}}) \cdot f(\text{A[W]} \mid \text{A\{n = "x"\}/MR/IR/M\{t = "y"\}}) \cdot f(\text{W[L]} \mid \text{A\{n = "x"\}[W]/MR/IR/M\{t = "y"\}}).$$

Based on Assumptions 1 and 2 (Twig Independence and Edge-Value Independence), our framework makes the following approximations:

$$\begin{aligned} f(\text{MR/IR} \mid \text{IR/M\{t = "y"\}}) &\approx f(\text{MR/IR}), \\ f(\text{A\{n = "x"\}/MR} \mid \text{MR/IR/M\{t = "y"\}}) &\approx f(\text{A\{n = "x"\}/MR}), \\ f(\text{A[W]} \mid \text{A\{n = "x"\}/MR/IR/M\{t = "y"\}}) &\approx f(\text{A[W]}), \\ f(\text{W[L]} \mid \text{A\{n = "x"\}[W]/MR/IR/M\{t = "y"\}}) &\approx f(\text{W[L]}). \end{aligned}$$

It is interesting to note that, since edge (W, L) is F-stable, all W elements reach at least one L element independent of outgoing or incoming paths; hence, the last approximation is exact as both fractions are actually equal to 1. Returning to the expression for f_Q, it can be rewritten as follows:

$$f_Q = f(\text{IR/MR\{t = "y"\}}) \cdot f(\text{MR/IR}) \cdot f(\text{A\{n = "x"\}/MR}) \cdot f(\text{A[W]}) \cdot f(\text{W[L]}).$$

At this point, our estimation framework relies on one final application of the chain rule, in order to separate the structurally stable components from value predicates:

$$\begin{aligned} f(\text{IR/MR\{t = "y"\}}) &= f(\text{IR/MR}) \cdot f(\{\text{t = "y"}\} \mid \text{IR/MR}), \\ f(\text{A\{n = "x"\}/MR}) &= f(\text{A/MR}) \cdot f(\{\text{n = "x"}\} \mid \text{A/MR}). \end{aligned}$$

By virtue of **B**-stability, $f(\text{IR/MR}) = f(\text{A/MR}) = 1$, while, as mentioned earlier, **F**-stability guarantees that $f(\text{W[L]}) = 1$. Thus, our final estimation expression simplifies to the following form:

$$f_Q = f(\{\text{t = "y"}\} \mid \text{IR/MR}) \cdot f(\text{MR/IR}) \cdot f(\{\text{n = "x"}\} \mid \text{A/MR}) \cdot f(\text{A[W]}).$$

Overall, Equation (4) computes the selectivity estimate for the embedding $\overline{v}\{\overline{\sigma}\}$ as a product of two key components: (a) the fractions of elements in the stable parts of the embedding that satisfy the value predicates in $\overline{\sigma}$ (first two product terms), and (b) the selectivities of path or branch edges along the *"stability breaks"* in the $\overline{v}$ embedding (last two product terms). (Once again, it is easy to extend our estimation formula to the most general case of complex path expressions with *multiple* branches.) In the next two sections, we describe our methodology for approximating these two components in Equation (4) over a concise XSKETCH synopsis.

procedure TWIGEST($\mathcal{X}S(G)$,$T\{\overline{\sigma}\}$)
Input: XSKETCH $\mathcal{X}S(G)$; stable twig $T = v_1/\cdots/v_{n-k-1}/v_n[v_{n-1}/\cdots/v_{n-k}]/v_{n+1}/\cdots$ $\cdots/v_{n+m}$ with value predicates $\overline{\sigma}$.
Output: Estimate of the selectivity fraction $f(T\{\overline{\sigma}\})$.
begin
1. `Covered` := ϕ; `Uncovered` := { nodes with value predicate in $\overline{\sigma}$}
2. result := 1; index := $n+m$; $v := v_{\text{index}}$
3. **while** `Uncovered` $\neq \phi$ **do** // check for predicates covered by node v's statistics
4. **if** (`Uncovered` $\cap$ `dep`$(v,T) \neq \phi$) **then**
5. // find covered and uncovered ancestors/descendants of v that are in its
6. // direct "correlation scope"
7. C := `Covered` $\cap$ `dep`(v,T)
8. U := `Uncovered` $\cap$ `dep`(v,T)
9. Using the element distribution information (histogram) at v, compute the conditional fraction:
10. $f^* := f(\texttt{twig}(v,U)\{\overline{\sigma}(U)\} \mid \texttt{twig}(v,C)\{\overline{\sigma}(C)\}) = \frac{f(\texttt{twig}(v,U\cup C)\{\overline{\sigma}(U\cup C)\})}{f(\texttt{twig}(v,C)\{\overline{\sigma}(C)\})}$.
11. // update result estimate and covered/uncovered nodes
12. result := result $\cdot f^*$; `Covered` := `Covered` $\cup U$; `Uncovered` := `Uncovered` $- U$
13. **endif**
14. index := index -1; $v := v_{\text{index}}$ // move to next twig node
15. **endwhile**
end

Fig. 7. The TWIGEST algorithm for selectivity estimation over stable XSKETCH twigs.

4.2.2 *Selectivity Estimation for Stable Twigs with Value Predicates.* Consider a *stable twig embedding* T in an XSKETCH synopsis $\mathcal{X}S(G)$, and let $\overline{\sigma}$ denote a collection of value predicates over the nodes of T. Our goal is to utilize the statistical information in $\mathcal{X}S(G)$ to obtain an estimate for $f(\overline{\sigma} \mid T)$, the fraction of data elements that are discovered by the "value-restricted" twig embedding $T\{\overline{\sigma}\}$.

Once again, to simplify the exposition, we consider a single-branch stable twig T; our discussion can easily be extended to the general, multi-branch case. (We also use T as the *set* of XSKETCH nodes in the twig when no confusion arises.) Let v be a node in T. Given a set S of ancestor and/or descendant nodes of v in T, let `twig`(v, S) denote the subtwig of T that connects v to all the nodes in S, and let $\overline{\sigma}(S)$ denote the set of value predicates on nodes of S in our twig query (i.e., the restriction of $\overline{\sigma}$ to S).

Our algorithm for estimating the selectivity of a stable twig embedding $T\{\overline{\sigma}\}$ (termed TWIGEST) is depicted in Figure 7. Briefly, our TWIGEST algorithm examines each node v in the main path of the twig embedding (in reverse order) and considers the set of value predicates that can be directly *"covered"* by the node's joint-distribution histogram based on its correlation scope `dep`(v). When a branching node (e.g., v_n) is encountered, TWIGEST traverses the branch top-down once again using nodes' correlation scopes to cover value predicates in the branch. This covering of value predicates is based on successive applications of the chain rule as we parse the twig embedding; of course, since each node carries only limited value-correlation information, our estimate relies on one final independence assumption for element-value distributions.

Assumption 3 (*Value Independence Outside Correlation Scope*). The distribution of elements in the `extent` of an XSKETCH node v is *independent* of the values in other XSKETCH nodes u that are *not* in v's direct correlation scope, that is, $u \notin \texttt{dep}(v)$.

Assumption 3 allows us to simplify the chain-rule conditionals to obtain the conditional probability f^* in Step 10 of TWIGEST, which can be directly estimated using the histogram information in v. Once all value predicates in the embedding $T\{\overline{\sigma}\}$ have been covered, TWIGEST returns the accumulated selectivity estimate for $f(\overline{\sigma} \mid T)$.

As shown, TWIGEST examines each node of the embedding at most once and hence performs at most N_T iterations for the loop of lines 3-15. (N_T denotes the number of nodes in the embedding.) For each node, the algorithm performs standard set operations on $dep(v)$, which has size $O(N_S)$, and computes two selectivity estimates using the histogram of v. The complexity of the former is $O(N_S)$ provided that sets are maintained in sorted order, while the complexity of the latter step obviously depends on the specific summarization method used in the value-summary of v (e.g., conventional histogram, wavelet, sample-based summary, etc.). Here, we use $O(h)$ to denote the time complexity of computing a single selectivity estimate from the histogram of v, and thus express the time complexity of TWIGEST as $O(N_T N_S \cdot h)$. Hence, the total complexity of all the invocations of TWIGEST for a single query embedding is $O(N_Q N_S^2 \cdot h)$.

4.2.3 *Selectivity Estimation across Nonstable Edges.* We now shift our attention to the problem of estimating path selectivities across nonstable edges of a XSKETCH synopsis $\mathcal{X}S(G)$, that is, the fraction terms $f(v_{k_i}/v_{k_i+1})$ and $f(u_{m_i}[u_{m_i+1}])$ in Equation (4). Given the absence of detailed distribution information in $\mathcal{X}S(G)$, our estimation framework approximates the required selectivities by applying the following uniformity assumptions on the path structure of the data graph:

A.4 (*Backward-edge Uniformity*). Given a node v in $\mathcal{X}S(G)$, the incoming edges to v from all parent nodes u of v such that v is *not* B-stable with respect to u are *uniformly distributed* across all such parents in proportion to their `counts`; that is, if we let $\mathcal{N}B(v)$ denote the set of all "non-B-stable" parents of v then the fraction of elements in $\texttt{extent}(v)$ that are reached by $u \in \mathcal{N}B(v)$ is approximately $\texttt{count}(u)/\sum_{w \in \mathcal{N}B(v)} \texttt{count}(w)$.

A.5 (*Forward-edge Uniformity*). Given a node v in $\mathcal{X}S(G)$, the outgoing edges from v to all children u of v such that v is *not* F-stable with respect to u are *uniformly distributed* across all such children in proportion to their `counts`, and the total number of such edges is *at most* equal to the total of these `counts`; that is, if we let $\mathcal{N}F(v)$ denote the set of all "non-F-stable" children of v and $s = \sum_{w \in \mathcal{N}F(v)} \texttt{count}(w)$, then the fraction of elements in $\texttt{extent}(v)$ that reach $u \in \mathcal{N}F(v)$ is approximately $\texttt{count}(u)/\max\{s, \texttt{count}(v)\}$.

Note that our "forward" Assumption 5 above is slightly different from its "backward" analog (Assumption 5), due to the max{} in the normalizing constant for the fractions. The key intuition for this differentiation is as follows.

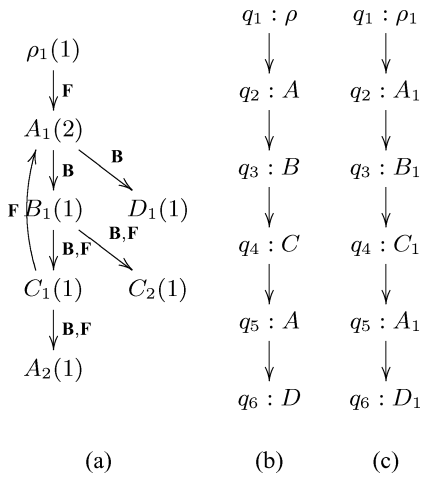


Fig. 8. Identifying embeddings in recursive data: (a) XSKETCH synopsis, (b) query, (c) embedding.

Backward-Edge Uniformity is basically estimating "up" from a given synopsis node and, since the node has parents, every node in its extent also *must* have parents in the data (root XML elements are grouped separately). Forward-Edge Uniformity tries to estimate "down" from a node, and the situation is not symmetric as not every element in the node's extent has to have children. Omitting the max{} from the denominator in Assumption 5 and using only the summation over children would essentially force every element in the node's extent to have a child which seems an excessively strong assumption to make (especially for nodes with very large counts compared to their child counts).

4.3 Selectivity Estimation on Recursive Data

Up to this point, our presentation of the XSKETCH estimation framework has focused on the case of nonrecursive data, that is, where every document path consists of uniquely labeled elements. In this section, we consider recursive data sets and discuss the application of our techniques to nonrecursive and recursive queries.

4.3.1 *Nonrecursive Queries.* We first present an extension of the XSKETCH estimation framework to the case of nonrecursive queries over recursive data. At a high level, the basic approach remains essentially the same: a query if first mapped to its set of embeddings, and the total selectivity is subsequently approximated as the sum of embedding selectivities. The existence of data recursion affects the first step only (algorithm COMPUTEEMBEDDINGS), as it now becomes necessary to handle synopsis cycles in the identification of embeddings; the second stage of estimating embedding selectivity remains exactly the same, as its specifics do not depend on the existence of recursion.

Before describing the extensions to algorithm COMPUTEEMBEDDINGS, we illustrate the issue of synopsis cycles with a simple example. Figure 8 depicts a sample XSKETCH synopsis of a recursive data set and a simple linear path query. Consider the execution of COMPUTEEMBEDDINGS on this sample input, and

assume that the depth first traversal follows the path $\rho_1.A_1.B_1.C_1$. After identifying the candidates of C_1, the algorithm has computed the following information for nodes on the traversal stack:

$$cand[\rho_1] = \{q_1\},\ \ cand[A_1] = \{q_2, q_5\},\ \ cand[B_1] = \{q_3\},\ \ cand[C_1] = \{q_4\}\,.$$

Assume that the traversal follows edge $C_1.A_1$ next and hence discovers the cycle in the synopsis. This cycle essentially reveals a cyclic dependency in the discovery of embeddings: the binding of A_1 to q_2 depends on the binding of C_1 to q_4, which in turn depends on the binding of A_1 to q_5. The latter, however, will be determined only after all the descendants of A_1 have been processed and popped off the stack, including C_1. The current algorithm will therefore fail to identify the embedding of Figure 8, as it will miss the witness path $C_1.A_1$ for the query edge (q_4, q_5).

To address this issue, we augment the information recorded by COMPUTEEMBEDDINGS with *pending* witness paths. These paths are identified when a cycle is detected in the traversal, and they essentially represent tentative witnesses that will be verified *after* the processing of the node that they reach. (Contrast this with normal witness paths, which are created and recorded when their last node is processed.) We illustrate this idea with the same example of Figure 8. When the cycle is detected, the algorithm examines the *cand* sets of the nodes on the traversal stack and inserts pending witness paths as follows:

$$cand[\rho_1] = \{q_1\},\ \ cand[A_1] = \{q_2, q_5\},\ \ cand[B_1] = \{q_3\},\ \ cand[C_1] = \{q_4\}\,,$$
$$pend[A_1, q_2, q_3] = \{A_1.B_1\},\ \ pend[B_1, q_3.q_4] = \{B_1.C_1\},\ \ pend[C_1, q_4, q_5] = \{C_1.A_1\}\,.$$

Intuitively, pending witnesses encode the cyclic dependencies between the bindings of synopsis nodes, and enable the algorithm to determine *tentative* root bindings for nodes in the cycle. The confirmation of these tentative bindings occurs when the first node of the cycle is eventually processed. Returning to the running example, C_1 is identified as a tentative root binding for q_4, pending the confirmation of $C_1.A_1$ as a witness path. (The same happens for B_1 and $pend[B_1, q_3, q_4]$.) After processing D_1, the algorithm resumes the processing of A_1 (the first node in the cycle), having recorded the following information:

$$cand[A_1] = \{q_2, q_5\},\ \ paths[A_1, q_5, q_6] = \{A_1.D_1\}\,,$$
$$pend[A_1, q_2, q_3] = \{A_1.B_1\},\ \ pend[B_1, q_3.q_4] = \{B_1.C_1\},\ \ pend[C_1, q_4, q_5] = \{C_1.A_1\}\,.$$

The confirmation of pending paths proceeds in two stages. First, COMPUTEEMBEDDINGS processes the nodes in $cand[A_1]$ in descendant to ancestor order and determines the root bindings of A_1 based on *paths* only. In this example, it can be verified that A_1 is a root binding for q_1 since $paths[A_1, q_5, q_6] \neq \emptyset$. Subsequently, the algorithm examines the list of pending paths based on the nesting of query steps, and verifies the validity of witnesses. Hence, $pend[C_1, q_4, q_5] = \{C_1.A_1\}$ is verified first, followed by $pend[B_1, q_3, q_4]$ and $pend[A_1, q_2, q_3]$. The latter provides the witness for the binding between A_1 and q_2, thus modifying the bind set to $bind[A_1] = \{q_2, q_5\}$.

Overall, this extension introduces two additional steps to our algorithm: the identification of witness paths, and their verification. The fist step increases

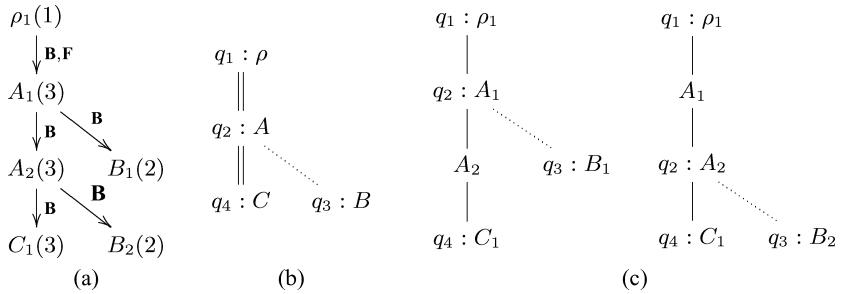


Fig. 9. Estimating the selectivity of a recursive query over recursive data: (a) XSKETCH of recursive data, (b) recursive query Q, (c) two embeddings of Q with overlapping results.

the complexity by a factor of $O(N_Q N_S^2)$ for the first node of a cycle in the depth-first traversal, as there are $O(N_S)$ nodes in the cycle, each node has $O(N_Q)$ candidates in $cand$, and each node has to be compared against its ancestors in the cycle. The second step involves the verification of $O(N_S N_Q)$ paths per cycle, as the number of pending paths per node can be at most equal to the number of query edges. (The latter corresponds to the uncommon case where every query step has a wild-card label.)

4.3.2 *Recursive Queries.* As hinted earlier, it is not straightforward to extend the XSKETCH framework to the case of combined data and query recursion. The key issue is that recursion complicates the identification of duplicates in the result-set of the query, thus increasing the difficulty of handling the set-based semantics of XPath expressions. Figure 9 illustrates this problem with a simple recursive query over an example synopsis. As shown in part (c), the query has two distinct synopsis embeddings that both reach the same summary node C_1; in turn, this implies that the individual selectivities may overlap, that is, certain elements in $\texttt{extent}(C_1)$ may be reachable by both embeddings. The naive aggregation of the individual selectivities will therefore overestimate the true selectivity of the query, due exactly to the double counting of result elements. A more methodical approach is to combine the two selectivities using the principle of inclusion-exclusion, thus accounting explicitly for their overlap and generating an estimate of improved accuracy.

As the previous example indicates, generalized recursion introduces double counting in the naive aggregation of embedding selectivities and therefore requires testing for overlap (and in a more general context, for inclusion) among a potentially large set of embeddings. This clearly increases the complexity of the estimation function, thus conflicting with the requirement of obtaining fast selectivity estimates during query optimization. Overall, these observations only hint at possible new approaches for this general case, and potentially open up an interesting direction for future work in the area of XML summarization.

5. XSKETCH SYNOPSIS CONSTRUCTION

In this section, we turn our attention to the important problem of effective XSKETCH construction for a given synopsis space budget. Briefly, our approach is based on using successive, localized *refinement operations* to gradually evolve

an initial, coarse XSKETCH into a more detailed synopsis that captures the important path and value correlations in the data. We first discuss the specific set of XSKETCH refinement operations that we use, and then present our construction algorithm in more detail.

5.1 Refinement Operations

In order to approximate path-expression selectivities, our XSKETCH estimation framework (Section 4) relies on a number of statistical (uniformity and independence) assumptions that compensate for the lack of detailed path and value information in the synopsis. Clearly, the accuracy of an XSKETCH (and the resulting selectivity estimates) depends crucially on the validity of these assumptions and the degree to which they reflect the statistical characteristics of the underlying path/value distribution in the actual data. To build an effective XSKETCH, we need to be able to appropriately *refine* the synopsis structure for regions of the data graph where our estimation assumptions fail, since these regions are likely to result in high estimation errors (the relational-world analog would be allocating more buckets to "difficult" data regions during histogram construction [Poosala and Ioannidis 1997]). In this section, we introduce such localized refinement operations for XSKETCH synopses. We categorize our refinement operations into two types: (1) *structural refinements* that refine the path structure in the XSKETCH, and (2) *value refinements* that refine the value-distribution information maintained in XSKETCH nodes. In what follows, we describe the operations in more detail and discuss their connection to the assumptions of our estimation framework. Due to space constraints, the details of our refinement operations (including pseudocode descriptions and complexity analyses) are provided in the *Electronic Appendix*.

5.1.1 *Structural Refinements.* At an abstract level, each structural refinement operation uses a partitioning criterion to *split* an XSKETCH node u into a set of new nodes $\{u_i\}$, such that $\cup_i \texttt{extent}(u_i) = \texttt{extent}(u)$ and $\texttt{extent}(u_i) \cap \texttt{extent}(u_j) = \phi$ for all $i \neq j$. Our node-partitioning criteria aim to either completely eliminate some uniformity/independence assumption(s) for the new synopsis nodes $\{u_i\}$, or to at least make such assumptions much more realistic for the new nodes $\{u_i\}$ than the old node u (similar to histogram-bucket splits). Thus, successive refinements evolve the synopsis to a larger and more precise summary. We define three different structural refinement operations for XSKETCH nodes, namely, `B_Stabilize`, `F_Stabilize`, and `B_Split`, which we describe in more detail below. To simplify the presentation, our description assumes that the base XML data graph is accessed and used to refine the XSKETCH synopsis; in reality, all the refinement operations only need to use the (potentially smaller) B/F-bisimilar graph that provides the exact path and branching distribution information for the data.[4]

—`B_Stabilize`($\mathcal{X}S(G)$, u, v). Here, v is a parent of node u in the $\mathcal{X}S(G)$ synopsis, and $\mathbf{B} \notin \texttt{label}(v, u)$. Clearly, when estimating the selectivity of any

[4]The key difference for our refinement operations is that the `extents` of XSKETCH nodes are defined as sets of nodes *of the B/F-bisimilar graph* rather than the data graph.

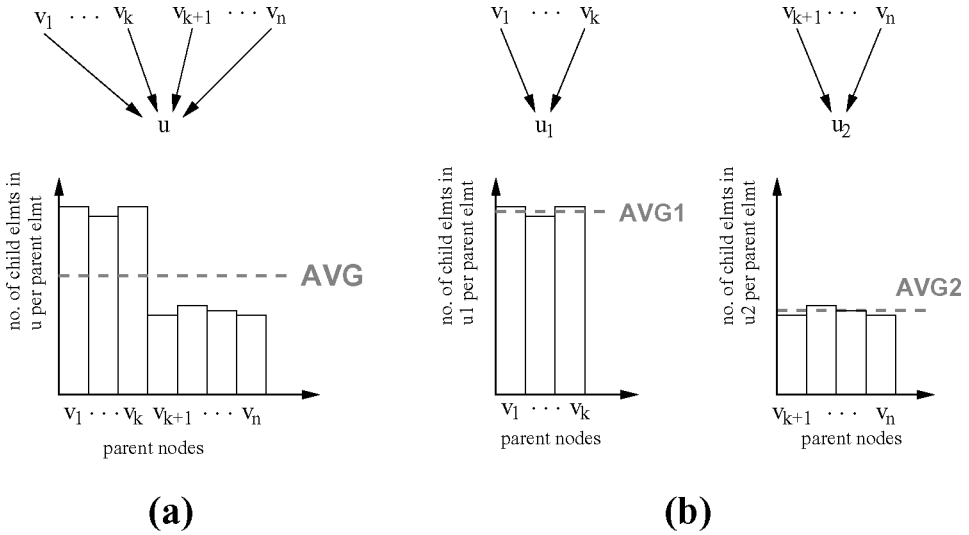


Fig. 10. The B_Split operation: (a) original "bad" summary; (b) after applying B_Split at node u.

twig embedding in $\mathcal{X}S(G)$ that contains the edge (v, u) in the main path, this edge constitutes a breakpoint in the parsing of the embedding into maximal stable subtwigs. This essentially forces the application of Twig Independence (Assumption 1) and Backward-Edge Uniformity (Assumption 4) in order to estimate $f(v/u)$, that is, the fraction of data elements in u that descend from v (Section 4). A B_Stabilize operation eliminates the need for such assumptions by refining the u node into two element partitions u_1, u_2 with the same label, of which only u_1 is connected to v with a B-stable edge. Hence, edge (v, u) is substituted with a new edge (v, u_1) where $\texttt{label}(v, u_1) = \texttt{label}(v, u) \cup \{\mathbf{B}\}$ and $f(v/u_1) = 1$.

—F_Stabilize($\mathcal{X}S(G)$, u, w). The F_Stabilize operation represents the "forward" equivalent of B_Stabilize. Here u is a parent of node w in the $\mathcal{X}S(G)$ synopsis, and $\mathbf{F} \notin \texttt{label}(u, w)$. When estimating the selectivity of a path embedding whose branch contains the edge (u, w), the break in the F-stability chain mandates the use of Twig Independence (Assumption 1) and Forward-Edge Uniformity (Assumption 5) in order to estimate $f(u[w])$, that is, the fraction of data elements in u that have a child in w (Section 4). The F_Stabilize operation separates out exactly those elements of u in a new synopsis node u_1, so that $\texttt{label}(u_1, w) = \texttt{label}(u, w) \cup \{\mathbf{F}\}$ and P[ ]$u_1[w]) = 1$.

—B_Split($\mathcal{X}S(G)$, u, $\{v_i\}$). Here, u is a node in the $\mathcal{X}S(G)$ synopsis and $\{v_i\}$ is the set of parent nodes of u such that $\mathbf{B} \notin \texttt{label}(v_i, u)$ (that is, u is not B-stable with respect to parent v_i). When estimating the selectivity of any twig embedding in $\mathcal{X}S(G)$ that contains any of the (v_i, u) edges in the main path, the break in the B-stability chain forces the use of Backward-Edge Uniformity (Assumption 4) in order to estimate the fraction $f(v_i/u)$ of elements in u that descend from elements in v_i. Such a scenario is depicted in Figure 10(a) where the node u in the XSketch is shown along with an

example histogram that summarizes the exact count-distribution information for the number of children in u per element in each parent v_i. According to Backward-Edge Uniformity, the number of elements in u that descend from elements in $v_k \in \{v_i\}$ is expressed as follows:

$$sel(v_k/u) = \texttt{count}(u) \times \frac{\texttt{count}(v_k)}{\sum_{v_i} \texttt{count}(v_i)}, \quad \text{for all } k,$$

which, of course, implies that

$$\frac{sel(v_k/u)}{\texttt{count}(v_k)} = \frac{\texttt{count}(u)}{\sum_{v_i} \texttt{count}(v_i)} = \texttt{constant}, \quad \text{for all } k.$$

Essentially, the Backward-Edge Uniformity assumption approximates this count distribution (that is, the ratio of the number of child elements in u per parent element v_k) using a single average, indicated by the dashed line in our example histogram. As our example figure shows, this may result in a poor approximation when the count-distribution of the $\{v_i\}$ parents is somewhat skewed. Our `B_Split` operation tries to intelligently partition the elements in u across two new XSKETCH nodes so that (1) the set of parents $\{v_i\}$ of u is also partitioned across the two new nodes, and (2) the Backward-Edge Uniformity assumption *within each partition* gives a much more accurate approximation. A possible `B_Split` for our example summary is shown in Figure 10(b) where, without loss of generality, the parent set $\{v_i\}$ has been partitioned into $\{v_1, \ldots, v_k\}$ and $\{v_{k+1}, \ldots, v_n\}$. The key idea is that, by intelligently partitioning based on count information, `B_Split` manages to produce much more uniform count histograms and, thus, substantially improve the accuracy of the average approximation within each of the resulting nodes u_1 and u_2. The benefit of the `B_Split` operation, therefore, is that it does not lift Backward-Edge Uniformity but, instead, tries to make the summary fit it better. Of course, the situation is slightly more complicated than the depiction in Figure 10, as the children of elements in $\{v_i\}$ may actually *overlap* in u and such overlaps should be accounted for when partitioning $\{v_i\}$. The approach followed in our `B_Split` operation is to initially group the nodes in the parent set $\{v_i\}$ into disjoint *node clusters* whose children are nonoverlapping in u, and then partition $\{v_i\}$ at this "coarser" level of node clusters.

5.1.2 *Value Refinements.* Our value-refinement operations try to improve estimation accuracy for value predicates by increasing the granularity of the value-distribution information maintained at individual XSKETCH nodes. Abstractly, there are two ways to improve the accuracy of the distribution information at an XSKETCH node u: (1) give more space (that is, additional buckets) to the histogram(s) already in u, and (2) expand the correlation scope of u, so that additional value correlations within u's stable twig neighborhood are captured. Thus we introduce two new value-refinement operations for XSKETCH nodes u with histogram information: (1) `Value_Refine`, which allocates more space to histograms, and (2) `Value_Expand`, which expands the correlation scope of a node. (The details can be found in the Electronic Appendix.)

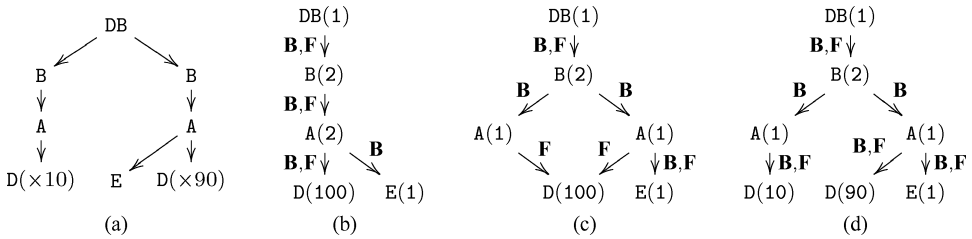


Fig. 11. Handling Twig Independence: (a) XML document, (b) XSKETCH synopsis, (c) After F_Stabilize, (d) After B_Stabilize.

5.1.3 *Discussion.* Having defined our XSKETCH-refinement operations, we now describe how they attack each of the assumptions in our estimation framework to improve accuracy. As we have already discussed, certain correspondences are obvious from the definition of the operations: B_Stabilize attacks Backward-Edge Uniformity (Assumption 4), since it inserts B-stable edges in the synopsis and thus lifts the need for the assumption along a nonstable edge; B_Split handles the same assumption by making the distribution of incoming edges more uniform across the new nodes; F_Stabilize attacks Forward-Edge Uniformity (Assumption 5) by adding F-stable edges; Value_Expand handles an invalid assumption of Value Independence Outside Correlation Scope (Assumption 3), since it expands the scope of a value summary to a larger subset of the stable neighborhood; and, Value_Refine attacks the implicit assumption that value histograms are good approximations of the underlying value distributions, by allocating more memory to the value-summarization model. In what follows, we discuss how the refinement operations cover the remaining assumptions of our estimation framework, namely, Twig Independence and Edge-Value Independence.

Twig Independence (Assumption 1) is applied across edges that are not stable during path parsing and is lifted with the help of B_Stabilize and F_Stabilize operations that introduce more B-stable and F-stable edges in the summary. As an example, consider the XML document of Figure 11(a) and the corresponding XSKETCH synopsis in Figure 11(b) (we use $A(\times n)$ in the document to denote n elements of label A). Suppose that we wish to estimate the selectivity of expression B/A/[/E]/D over the given synopsis. As described in Section 4, the expression will be parsed in two stable twigs, namely, B/A/D and [E], and its selectivity will be calculated as $f(\text{B/A/[/E]/D}) = f(\text{B/A/D}) \cdot f(\text{A[/E]} \mid \text{B/A/D})$. The first term captures the fraction of D elements that are reached by B/A/D, while the second term represents the fraction of those elements whose A parent satisfies the branch [/E]. At this point, the estimation framework applies the Twig-Independence assumption that decorrelates the branching predicate from the main path and allows the second term to be approximated as follows: $f(\text{A[/E]} \mid \text{B/A/D}) \approx f(\text{A[/E]})$. Essentially, this corresponds to the following assumption: the fraction of B/A/D elements whose parent satisfies the branching predicate is equal to the fraction of B/A elements that satisfy [/E]. This assumption, however, is clearly wrong in our example XML data, as 90% of the B/A/D elements have an A parent with an E child, while the selectivity of $f(\text{A[E]})$ is 50%. To handle this erroneous Twig-Independence assumption, the XSKETCH

framework can refine the synopsis with two stabilization refinements: an initial F_Stabilize operation, which separates the A elements with a E child, and a followup B_Stabilize, which separates the D elements according to their A parent. This process is shown pictorially in Figures 11(c) and 11(d) and, as the final synopsis shows, it essentially isolates the skewed regions of the distribution in a completely stable part of the summary. As a result, the initial path expression is mapped to a single stable twig and the need for Twig Independence is lifted. The same technique can be applied when path correlations extend over a larger region of the summary, at the expense, of course, of more F_Stabilize and B_Stabilize operations. It is interesting to note that our approach is similar in spirit to histogram-based techniques for relational data, especially compressed histograms [Poosala et al. 1996], which use singleton buckets in order to isolate values of high frequency and thus capture skew in the underlying data distribution.

Edge-Value Independence (Assumption 2) can be handled in a similar fashion, by essentially isolating skewed regions of the path distribution in stable parts of the summary. As described in Section 4, the assumption is applied along nonstable edges to decorrelate the path distribution from the distribution of values in other parts of the document. The key idea, therefore, is to employ F_Stabilize and B_Stabilize operations that completely stabilize the skewed region and thus obviate the need for an independence assumption across graph edges and values.

5.2 XSKETCH Construction Algorithm

Given the selectivity-estimation framework of Section 4, we now address the difficult problem of building an XSKETCH synopsis that effectively summarizes a large XML data graph within a given space budget. In many respects, XSKETCH construction is similar to other *statistical-model inference* problems, where the goal is to infer an "optimal" statistical model (e.g., Bayesian or Markov network) from an underlying data set. Most such problems have been shown to be computationally hard and can be solved exactly only by exhaustive search [Pearl 1988]. As the following theorem demonstrates, our effective XSKETCH construction problem is also computationally intractable; thus, it is unlikely that we can build accuracy-optimal XSKETCHes in an efficient manner. Our reduction is based on demonstrating that a specific, rather simple, instance of the optimal (structural) XSKETCH construction problem is actually equivalent to an optimal clustering problem for a set of points on the Euclidean plane [Fowler et al. 1981; Gonzalez 1985; Megiddo and Supowit 1984]; the detailed proof argument can be found in the Electronic Appendix.

THEOREM 5.1. *Let $\mathcal{Q}$ be a fixed set of path expressions to be evaluated over an XML data graph G. The problem of building an* XSKETCH *synopsis $\mathcal{XS}(G)$ of G with at most K nodes that minimizes the mean-squared error in the selectivity estimates of path expressions in $\mathcal{Q}$ is $\mathcal{NP}$-hard.*

Based on this intractability result, we propose a computationally-efficient heuristic algorithm, termed BUILDXSKETCH, for building XSKETCH synopses. At

procedure BUILDXSKETCH(G, B)
Input: XML data graph G; space budget B.
Output: XSKETCH synopsis $\mathcal{X}S(G)$ of G of size at most B.
begin
1. `minima` := ϕ; $\mathcal{X}S(G)$:= Label-split graph synopsis with 1-d, single-bucket histograms
2. **while** (**true**) **do**
3. R := generate candidate refinements on $\mathcal{X}S(G)$ that do not cause our space budget B to be exceeded
4. **if** ($R = \phi$) **break**
5. `bestSketch` := **nil**; `bestGain` := 0
6. **for each** $r \in R$ **do**
7. $\mathcal{X}$:= XSKETCH resulting from applying r on $\mathcal{X}S(G)$
8. $\texttt{gain}(r) := \frac{\texttt{score}(\mathcal{X}) - \texttt{score}(\mathcal{X}S(G))}{\texttt{size}(\mathcal{X}) - \texttt{size}(\mathcal{X}S(G))}$
9. **if** ($\texttt{gain}(r) >$ `bestGain`) **then**
10. `bestGain` := $\texttt{gain}(r)$; `bestSketch` := $\mathcal{X}$
11. **endfor**
12. **if** (`bestSketch` = **nil**) **then**
13. `minima` := `minima` $\cup \{\mathcal{X}S(G)\}$
14. take a number of random refinement steps
15. **else**
16. $\mathcal{X}S(G)$:= `bestSketch`
17. **endwhile**
18. **return** the XSKETCH in `minima` $\cup \{\mathcal{X}S(G)\}$ with the best `score`()
end

Fig. 12. The BUILDXSKETCH algorithm.

an abstract level, our algorithm views XSKETCH construction as a search problem over the space of all possible XSKETCH synopses, and uses our localized XSKETCH refinement operations to effectively explore this space. More specifically, BUILDXSKETCH is based on a *greedy, forward-selection* paradigm that starts out with a very coarse synopsis model and incrementally adds more complexity using our localized XSKETCH refinements. The initial (coarse) synopsis is basically the *label-split graph*, which partitions data element nodes into synopsis node based solely on their label, augmented with minimal distribution information that consists of a trivial (i.e., single-bucket) one-dimensional histogram for each node v with values.

Figure 12 depicts the pseudocode for our BUILDXSKETCH algorithm. The refinement strategy is based on the idea of *marginal gains* [Fox 1966]: at each iteration of the main loop, the algorithm selects and applies the refinement operation that results in the largest increase in accuracy per unit of extra space required (and, of course, does not violate our overall space budget for the synopsis) (Steps 2–17). To avoid getting trapped in local minima, BUILDXSKETCH takes a number of random steps (that is, random node refinements) when no accuracy-improving neighbor can be found (Steps 12–14). We now discuss two of the key components of our construction algorithm in more detail, namely: (1) the generation of candidate refinements for a given XSKETCH (Step 3), and (2) the scoring metric (`score`()) for evaluating the accuracy of an XSKETCH synopsis and its computation during the search (Step 8).

5.2.1 *Candidate Refinement Generation.* Rather than exhaustively examining all single-step refinements over the entire set of nodes in the XSKETCH, our BUILDXSKETCH algorithm considers only a restricted set of candidate refinements that are more likely to have significant impact on the estimation error. More specifically, we only consider refinements over a small sample V of the synopsis nodes which attempts to capture frequent portions of the data graph that are not yet refined (as a heuristic, we bias the sample toward nodes with high element counts and high numbers of incoming and outgoing unstable edges). Once this biased node sample V is formed, the set of candidate refinements includes, for each node $u \in V$: (1) all possible `B_Stabilize`, `F_Stabilize`, and `B_Split` operations with unstable parents or children of u; (2) one `Value_Refine`$(u, 1)$ operation to give one extra bucket to u's histogram; and, (3) one `Value_Expand`$(u, v, 1)$ operation, where v is the XSKETCH node in $(\texttt{STN}(u) - \texttt{dep}(u))$ that exhibits the highest degree of correlation with u, as determined by a statistical correlation criterion.

5.2.2 XSKETCH *Scoring Metric and Its Computation.* Our scoring metric for XSKETCH synopses is based on the *average absolute relative error* between the estimated and real counts over the set $\mathcal{P}$ of all complex path expressions in G and $\mathcal{X}S(G)$:

$$\texttt{score}(\mathcal{X}S(G)) = \frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} \frac{|\texttt{count}_{\mathcal{X}S(G)}(p) - \texttt{count}_G(p)|}{\max\{\texttt{count}_G(p), s\}},$$

where $\texttt{count}_{\mathcal{X}S(G)}(p)$ and $\texttt{count}_G(p)$ denote the XSKETCH estimate and the true count of expression $p \in \mathcal{P}$, respectively. The *sanity bound* parameter s used in the above definition essentially equates all zero or low counts with a default count s, thus avoiding inordinately high contributions from low-count path expressions to our relative-error metric. We typically set s equal to a small percentile of the actual count distribution in order to ensure good relative accuracy for most values in the distribution; for example, using a 5-percentile means that 95% of the counts in the distribution are larger than s.

As defined above, evaluating our scoring metric would require a huge number of path expression evaluations over the data graph G in order to get their actual result `counts`. Obviously, this becomes prohibitively expensive for large XML databases. To keep the computational overhead of XSKETCH construction manageable, our BUILDXSKETCH algorithm employs two key techniques:

(1) *Using a "reference synopsis."* Instead of using the data graph G, we approximate the actual path counts $\texttt{count}_G(p)$ using a reasonably accurate "reference synopsis" of G. (Since, in the presence of values, a perfect summary can easily be as large as the data, the reference summary needs to be approximate to keep the construction process tractable.) Our reference synopsis is based on augmenting the nodes of the perfect *structural* summary (that is, the B/F-bisimilar graph) with distribution information on their extents. The histograms on the distribution of node v try to capture all correlations within a neighborhood of v, and are built to guarantee an average error less than a specified threshold in the approximation. As our

results demonstrate, such a reference synopsis can grow quite large but is still significantly smaller than the full data graph.

(2) *Reducing the expression set $\mathcal{P}$.* preds All our refinement operations have a very localized effect on the XSKETCH synopsis. Further, our BUILDXSKETCH algorithm is only interested in the *difference* of `scores` resulting before and after a refinement (Step 8). Clearly, since $\texttt{count}_{\mathcal{X}S(G)}$ estimates are not going to change for paths that are not affected by a refinement operation, it is sufficient to estimate the effects of that refinement on only the affected paths. More specifically, when applying a refinement operation at an XSKETCH node u, we restrict the "gain" computation in Step 8 to only a small sample $P(u)$ of the label paths that contain node u; once again, this sample is biased toward high-count paths, since these are more likely to significantly impact estimation accuracy.

It is interesting to note that our algorithm can also use a representative workload as the set $\mathcal{P}$ of evaluated path expressions. Of course, this approach would bias the refinement process in favor of the supplied workload, thus tuning the characteristics of the synopsis to the path expressions that are expected to occur. (We discuss the time and space complexity of BUILDXSKETCH in the Electronic Appendix.)

6. EXPERIMENTAL STUDY

In this section we present the results of an extensive empirical study that we have conducted using our novel XSKETCH synopses described in this article. The objective of this study is twofold: (1) to establish the effectiveness of XSKETCHes as summaries for graph-structured XML documents, and (2) to demonstrate the benefits of our methodology compared to earlier approaches for the estimation of simple path expressions over tree XML documents. Our experiments consider a wide range of queries over synthetic and real-life data and demonstrate the ability of XSKETCHes to capture important path and value correlations in the underlying data using only limited space.

6.1 Testbed and Methodology

6.1.1 *Techniques.* We considered the following estimation techniques in our study:

—XSKETCHes. We have implemented the XSKETCH framework that we have presented in this article. Our prototype uses one-dimensional histograms to summarize value distributions under synopsis nodes with values and hence our estimation algorithm relies on value independence to approximate path expressions with value predicates. We found that, even though such independence assumptions are not valid in general, they are sufficiently accurate for the data sets used in our evaluation.

Our construction algorithm considers refinements on a biased 10% sample of all summary nodes and determines the score of each operation based on a biased sample of 50 label paths. We have chosen these values based on sensitivity analysis experiments that we discuss in the next section. Note

Table I. Characteristics of the Data Sets

		IMDB	XMark	DBLP
No. of elements		102,755	87,480	1,594,444
Document size (MB)		2.94	5.48	48.18
Label split graph	No. nodes	164	80	605
	Total size	9.1 KB	4.1 KB	31 KB
	Size of hists.	1.7 KB	688B	888B
Reference synopsis	No. nodes	49,181	83,466	6,573
	Total size	1.9 MB	3.3 MB	310 KB
	Size of hists.	388 KB	667 KB	105 KB

that, in our current implementation, XSKETCH construction does not consider `Value_Expand` refinement operations, since all node histograms are one-dimensional.

—*Markov Tables (MTs).* Aboulnaga et al. [2001] introduced several summary structures for estimating the selectivity of simple path expressions over *tree-structured* XML documents. We compare XSKETCHes against second-order Suffix-* Markov Tables, which were shown to be the most accurate synopses on the real-life data sets tested in Aboulnaga et al. [2001]. We note that recent studies [Aboulnaga and Naughton 2003; Chen et al. 2001; Freire et al. 2002; Lim et al. 2002; Wang et al. 2003, 2004; Wu et al. 2002] have proposed several other techniques for path selectivity estimation; we have chosen the Markov Tables technique for our comparison as it has been shown to be very effective in a "baseline" summarization problem, namely, estimating the selectivity of simple path expressions over tree-structured data. (A qualitative comparison of XSKETCHes to other summarization techniques is presented in Section 7.)

6.1.2 *Data Sets.* We use two real-life and one synthetic data set in our evaluation.

—*IMDB.* This is a real-life, graph-structured data set from the Internet Movie Database (`www.imdb.com`). It contains a large number of `IDREF` edges resulting in a highly irregular and cyclic path structure.

—*XMark.* This is a synthetic, graph-structured data set, modeling the activities of an on-line auction site (`www.xml-benchmark.org`). The path structure is highly irregular but the data set does not contain any cycles.

—*DBLP.* This is a real-life, tree-structured data set that contains bibliographic data from the DB&LP database (`www.informatik.uni-trier.de/ley/db`). It does not contain any cycles and is relatively regular in structure. We note that we use DBLP mainly for the comparison of XSKETCHes to Markov Tables and hence we show performance results only for simple path expressions on this data set.

Table I summarizes the main characteristics of our data sets. As expected, the DBLP data set has the smallest perfect summary since it is the most regular data set of the three. IMDB and XMark, on the other hand, have large perfect summaries, thus, motivating the need for concise synopses. Note that the sizes reported do not include the space needed to store the actual text of the element

Table II. Average Query Workload Result Sizes

		IMDB	XMark	DBLP
Branching paths	W/o predicates	1901	1057	—
	With predicates	478	254	—
Simple paths	W/o predicates	933	771	4667
	With predicates	483	302	12

labels; each label is hashed to a unique integer and the mapping is stored in a separate structure that is not part of the summary.

6.1.3 *Workload.* We evaluate the accuracy of each synopsis against a workload of positive path expressions, that is, expressions that have a nonzero result size in the original data. We form the workload by sampling document twigs from the reference graph and converting them to the corresponding label paths. The length of path expressions is uniformly distributed between 2 and 5, and the workload contains 500 path expressions with no predicates and 500 expressions with a single value range predicate. Each value predicate covers a random 10% range of the value domain of the tag it is attached to. We form two types of workloads: (1) *branching paths*, where half of the path expressions (with and without a value predicate) contain a branching predicate, and (2) *simple paths*, where no path expression contains branching predicates. Table II summarizes the average result size of each type of path expression across our three data sets.

We have also experimented with negative workloads, containing queries that have a zero count in the original data. Our XSKETCH summaries consistently produced close to zero estimates with negligible error, and, therefore, we omit these results from our presentation.

6.1.4 *Evaluation Metric.* We quantify the accuracy of an XSKETCH summary based on the *average absolute relative error* of result estimates over path expressions in our workload. Given a path expression p with true result size c, the absolute relative error of the estimated count e is computed as $|e - c|/\max(c, s)$. Parameter s represents a *sanity bound* that essentially equates all zero or low counts with a default count s and thus avoids inordinately high contributions from low-count path expressions. We set this bound to the 10-percentile of the true counts in the workload (that is, 90% of the path expressions in the workload have a true result size $\geq s$). In our results, we report the estimation error for path expressions with value predicates, termed *Pred*, the estimation error for path expressions without value predicates, termed *No Pred*, and the overall error for both types of path expressions in the workload, termed *Overall*.

6.2 Experimental Results

6.2.1 *Performance of* XSKETCH*es on Graph-Structured Data.* In this section, we present a set of experiments that evaluated the performance of our proposed XSKETCH synopses over the graph-structured data sets of IMDB and XMark. We performed an initial set of experiments to examine the sensitivity of the build process to the parameters of the construction algorithm; due

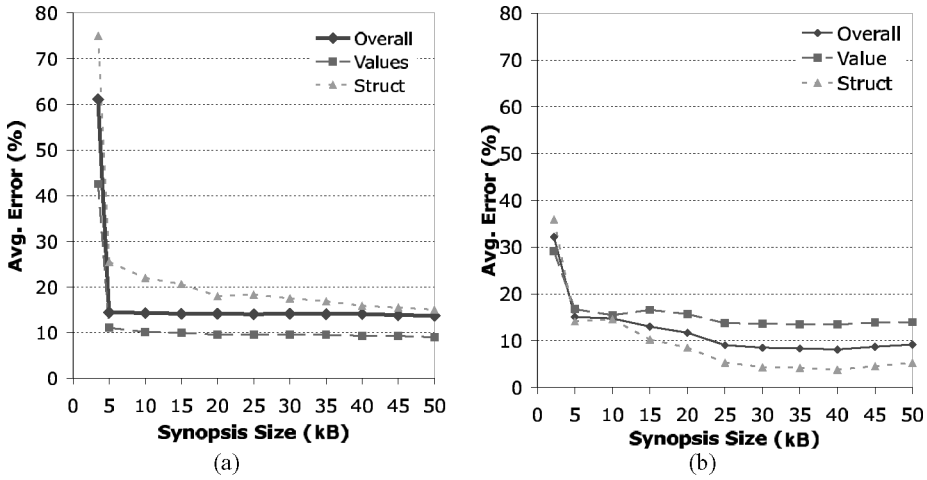


Fig. 13. Performance of XSKETCHes for branching path expressions on graph-structured data: (a) IMDB, (b) XMark.

to space constraints, a detailed discussion of our sensitivity findings can be found in the Electronic Appendix. Our second set of experiments focused on the effectiveness of XSKETCH synopses in summarizing the path structure and value content of the XML data graph, by measuring the accuracy of XSKETCH selectivity estimates for branching path expressions with value predicates.

Figure 13 depicts the estimation error of our XSKETCH synopses for branching path expressions over graph-structured data. For each data set, we show the overall estimation error, as well as the error for path expressions with and without value predicates. Overall, our results indicate that XSKETCHes are effective in providing accurate estimates for branching path expressions with value predicates. In the XMark data set, for instance, a moderate space budget of 25 kB was adequate to provide an overall estimation error of less than 10%. For the IMDB data set, on the other hand, we observed slightly higher estimation errors due to the more complex structure of the underlying XML graph; still, the overall error dropped at 15% for a fraction of the size of the perfect synopsis. Similar to previous experiments, we observed that the error was reduced rapidly during the first iterations of the construction process, and dropped gradually for larger space budgets.

6.2.2 *Performance of* XSKETCH*es on Tree-Structured Data.* Up to this point, our study has focused on the general case of graph-structured XML data, where nesting edges and id/idref edges are treated as first-class citizens in the XML data model. In this section, we restrict our attention to the *tree-structured* XML data model, that is, we consider only element containment in the XML data graph.

6.2.2.1 *Estimation Accuracy on Tree-Structured Data.* In this set of experiments, we evaluated the performance of XSKETCHes on estimating the selectivity of branching path expressions over tree-structured data sets. Similar to our

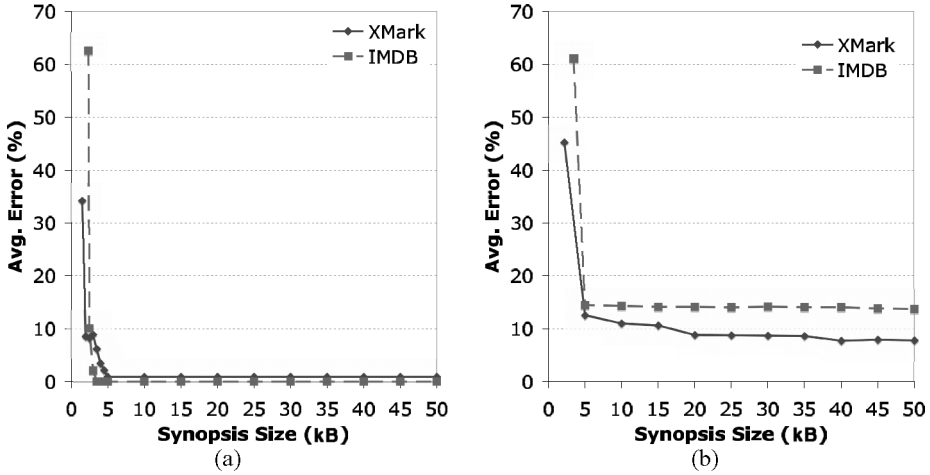


Fig. 14. Performance of XSKETCHes for branching path expressions on tree-structured data: (a) workload without value predicates, (b) workload with value predicates.

previous experiments on graph-structures, we set $V = 10\%$ and $P = 50$ during the construction of XSKETCH synopses.

Figure 14 shows the estimation error of XSKETCHes for branching path expressions with value predicates, over the tree-structured IMDB, and XMark data sets. (Note that IMDB and XMark are the same data sets that were used in our graph-structured experiments, except that we ignore any ID/IDREF edges.) Overall, our results exhibit similar trends to the case of graph-structured data: XSKETCHes enabled accurate selectivity estimates for low space budgets, with the estimation error dropping faster during the first steps of the build process and decreasing gradually thereafter. The results of Figure 14(a) indicate that XSKETCHes are especially effective as structural summaries for tree-structured data. In particular, we observe that a very small space budget of 5 kB sufficed to enable an average estimation error of less than 4% for both data sets. This is clearly a very effective tradeoff with respect to the corresponding perfect summaries, which provide estimates of zero error but require significantly more storage space (Table I.)

In the next set of experiments, we evaluated the accuracy of XSKETCHes on a workload of complex path expressions that comprise branching and value predicates, the descendant and child axes, and the wild-card label ("*"). Essentially, our goal was to evaluate the efficacy of the XSKETCH framework under the full complexity of our supported query model. We generated such workloads by post-processing the simpler workloads used in the previous experiments, as follows: for each query, we chose at random sequences of child steps replaced them with a single descendant step, and replaced certain labels (chosen again at random) with the wild-card label. This process resulted in a workload of approximately 500 unique complex path expressions for each data set, biased again in favor of high result counts. We note that we focused on tree-structured data sets for this type of complex queries, since our estimation framework supports recursive queries over nonrecursive data only (Section 4.1).

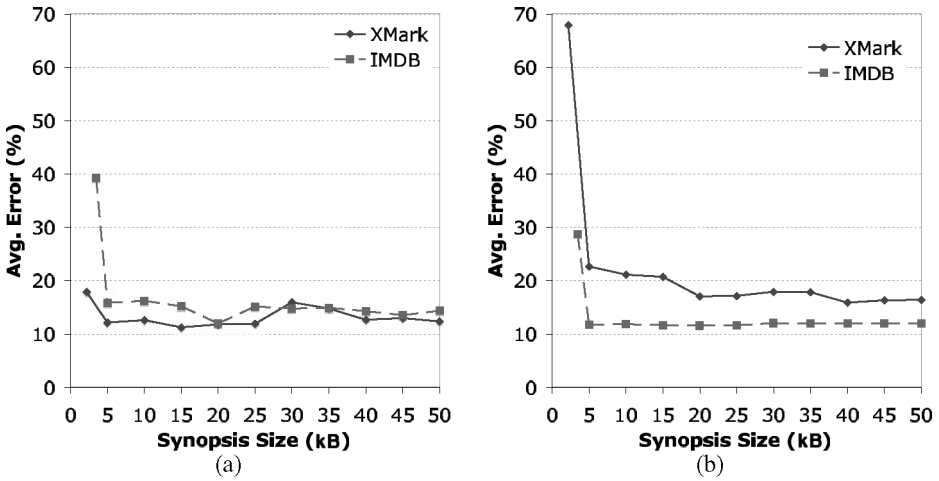


Fig. 15. Performance of XSKETCHes for complex path expressions (with // and *) on tree-structured data: (a) workload without value predicates, (b) workload with value predicates.

Figure 15 shows the performance of XSKETCHes on a workload with complex path expressions over the tree-structured IMDB and XMark data sets. (Similarly to the previous set of experiments, we show the estimation error separately for queries with and without value predicates.) Our results verify again the efficacy of XSKETCHes as accurate synopses for tree-structured XML data sets. As shown, XSKETCHes enabled low-error selectivity estimates even for the most complex queries in our model—queries that combined branching and value predicates coupled with the descendant axis ("//") and wild-card tags ("*"). In the IMDB data set, for instance, a moderate space budget of 30 kB sufficed to drop the estimation error below 15%. We note that the overall estimation error was increased compared to the previous set of experiments, but this is expected given the higher complexity of the test workloads.

6.2.2.2 *Estimation Accuracy for Simple Path Expressions.* Several recent studies [Aboulnaga et al. 2001; Wang et al. 2003; Wu et al. 2002] have looked at a constrained version of the general XML summarization problem, namely, estimating the selectivity of simple path expressions (that is, expressions without branching or value predicates) over tree-structured XML data. Such queries arise frequently in practice (e.g., in publish/subscribe systems for XML [Altinel and Franklin 2000]), and are also used by XML query optimizers in order to estimate the cost factors of physical evaluation plans [Wu et al. 2003]. In this section, we present a set of experiments that evaluated the effectiveness of XSKETCHes on simple path expressions over tree-structured data, and compare the performance of our synopses against the Markov Tables technique proposed by Aboulnaga et al. We used the second order Suffix-* Markov Table summary that was shown to have the best performance on the real-life data sets tested in [Aboulnaga et al. 2001].

Figure 16 shows the estimation error of XSKETCHes and Markov Tables as a function of the synopsis size on the (tree-structured) data sets of IMDB, XMark,

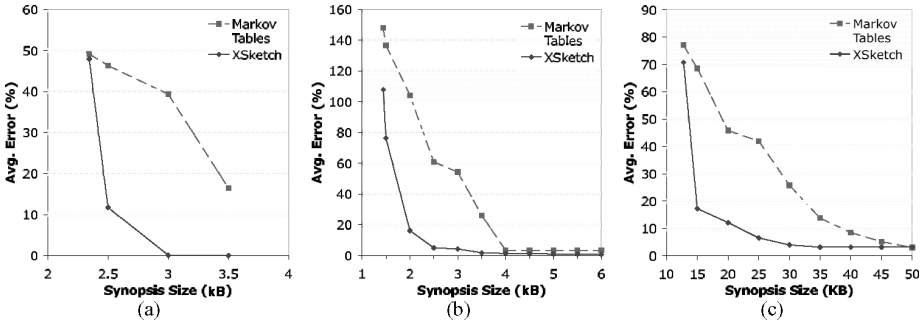


Fig. 16. Accuracy of XSKETCHes versus MTs: (a) IMDB, (b) XMark, (c) DBLP.

and DBLP. (We have included DBLP in this part of our study as Markov Tables have been shown to perform very well on this data set [Aboulnaga et al. 2001].) In all cases, XSKETCHes were more effective in capturing the key structural dependencies using the limited storage space, and consistently provided estimates with very low error after a few iterations of the construction algorithm. In the DBLP data set, for instance, an XSKETCH of 30 kB—a relatively small space budget compared to the size of the original data—yielded an estimation error of 4%, compared to 25% for MT summaries. The construction of MT summaries is based on the summarization of low-frequency paths with special *-paths, which approximate the pruned frequencies with an average and thus enforce a uniformity assumption. In addition, the MT-estimation model is based on a "Markovian memory" assumption in order to compute the count of a path from the counts of shorter paths (of length up to 2). The MT-construction algorithm, however, prunes paths in a greedy fashion based solely on their frequency and does not consider the validity of the two assumptions with respect to the underlying path distribution. Our approach, on the other hand, is more methodical as XSKETCH construction directly takes into account the structural dependencies that exist in the paths of the XML data graph. As a side note, it is interesting to point out that the highest values for the space budget in IMDB and XMark (3.5 and 8 kB respectively) correspond to the size of the perfect XSKETCH synopsis, that is, the summary that is derived from the B/F-Bisimilar graph. (More precisely, the perfect summary corresponds to the *path-tree* [Aboulnaga et al. 2001] of the data, since we focus on tree-structures.) In data sets where the tree structure is quite regular, therefore, the perfect XSKETCH is a viable option for the statistics component of a query optimizer, enabling fast, zero-error estimates for any simple path expression.

7. RELATED WORK

The problem of constructing accurate statistical synopses for flat, relational data sets has received a significant amount of attention in the context of both relational query optimization and, more recently, approximate query processing [Garofalakis and Gibbons 2001], and several effective solutions have been proposed, including histograms [Poosala and Ioannidis 1997; Poosala et al. 1996], wavelets [Vitter and Wang 1999], and random sampling [Chaudhuri

et al. 1999]. However, summarizing a large XML data graph for the purpose of estimating the selectivity of arbitrary path expressions with value predicates is a substantially different and more difficult problem than that of constructing synopses for flat, relational data. At an abstract level, the desired synopsis should be able to accurately capture (in a limited amount of space) the important statistical characteristics of both (a) the *label path and branching structure*, and (b) the *element value distributions and correlations* in a large, labeled data graph. This is clearly a nontrivial problem, and none of the existing synopsis techniques for flat data tables is immediately applicable. As a result, several recent studies have considered the problem of XML summarization and have proposed specialized techniques for semistructured data.

Previously proposed techniques for path selectivity estimation have considered a specialized variant of the general problem, either focusing on tree-structured data or on simple path expressions without branching or value predicates. Aboulnaga et al. [2001] introduced Markov Tables and path trees for estimating the selectivity of simple path expressions (that is, without branching or value predicates) over tree-structured data. Recent studies have proposed two extensions of Markov Tables, namely, *XPathLearner* [Lim et al. 2002] and *On-line Annotated Path Tables* [Aboulnaga and Naughton 2003], that use query feedback, instead of the base data, in order to construct a synopsis of the XML tree. In a nutshell, a Markov Table estimates the selectivity of a simple path by combining frequency information on its subpaths of a specific length (typically, two or three steps). This technique, however, can only be applied to fully specified paths, and it is not clear if it can be extended to handle the descendant axis (//). Wu et al. [2002] proposed the use of Position Histograms in order to estimate the selectivity of path expressions over XML trees. In brief, a Position Histogram is a two-dimensional histogram synopsis that summarizes the (*pre*, *post*) ids for elements of a specific tag. Estimating the selectivity of a path expressions thus amounts to performing a spatial join among the histograms that correspond to the steps of the path. The use of (*pre,post*) identifiers, however, limits this technique to tree-structured data; moreover, it is not clear if histograms are an effective compression method for the (*pre,post*) plane, since the latter essentially corresponds to a binary frequency matrix. Wang et al. [2004] proposed the Bloom Histogram, a randomized summarization technique that provides probabilistic guarantees on the estimation error for simple path expressions. It is unclear, however, if this model can be extended to handle the general case of graph-structured XML databases and of XPath expressions with branching and value predicates.

Recent studies have considered the problem of selectivity estimation for *twig queries*, which represent the joint evaluation of multiple path expressions and are essentially equivalent to the `for` clause of a query in the XML Query language. Chen et al. [2001] proposed the use of Pruned Suffix Trees (PSTs), where a trie is used to encode the paths in the document and special hash signatures within each trie node capture correlations among paths. The specifics of this technique, however, are tied to tree structures and thus cannot handle the general case of graph-structured XML data that we tackle in this article. StatiX [Freire et al. 2002] (and the followup IMAX [Ramanath et al. 2005]

study) relies on histograms in order to summarize the path structure and value content of an XML tree. The key idea is to generate one histogram for each type in the schema of the database, approximating the frequency counts of parent element ids for the elements of the specific type. To estimate the selectivity of a query, StatiX retrieves the histograms of matching types and performs a histogram-based join to approximate the size of the result. Clearly, StatiX requires a schema for the input data and it is not clear how it can be applied to schema-less XML documents; moreover, the semantics of histogram-based joins are compatible with tree structures only and it is not clear if StatiX can be applied to graph structured data. More specifically, graph structures allow multiple parents per element which essentially means that a single element can contribute to the frequency of several parent element ids in the same histogram. This in turn introduces duplicates in the result of the histogram-based join and requires some nontrivial mechanism of duplicate elimination. The XSKETCH estimation framework, on the other hand, relies on set semantics and can thus readily handle graph-structured data. Furthermore, XSKETCH requires a minimal amount of metadata pertaining to nontree edges, and is thus applicable to a wide class of XML documents that are not equipped with a full schema. Wang et al. [2003] introduced two techniques for estimating the result size of a containment join, that is, the number of (a, d) tuples produced when processing the simplified twig query `for $a in //A, $d in $a//D`. Their techniques, however, target a small fragment of the XPath query language, namely, expressions of two steps that do not contain branching or value predicates, whereas the proposed XSKETCH framework covers the significantly more complex class of branching path queries with value predicates. Finally, recent studies [Polyzotis et al. 2004a, 2004b; Polyzotis and Garofalakis 2006; Zhang et al. 2006], introduced summarization models that employ the graph-synopsis framework presented in this article in order to estimate the selectivity of complex twig queries. Still, these works focus exclusively on the case of tree-structured data and cannot be applied to the more general case of XML data graphs that we consider in this article.

Path-index structures for XML data, like strong DataGuides [Goldman and Widom 1997] and 1- and T-indexes [Milo and Suciu 1999], also try to capture the path structure in the underlying XML data graph. The basic idea is to group element nodes in the data graph into "coarser" index-graph nodes based on the set of incoming label paths at each data element. A key problem with such path indexes is their size, which can grow to a fairly large proportion of the data-graph size [Milo and Suciu 1999; Kaushik et al. 2002a]. Obviously, this fact severely limits their usefulness as an optimization-time data synopsis. The recently proposed A(k)-path index [Kaushik et al. 2002b] tries to limit the index size by using a more relaxed grouping rule that essentially groups data elements based on their incoming label paths *of length at most k*. Still, the usefulness of an A(k)-index graph structure as a synopsis is unclear: even though the k parameter can be adjusted so that the index fits in the allotted amount of space, using the *same, fixed value of k for the entire data graph* can result in a synopsis that does not capture the essential statistical characteristics of the data-graph distribution. Kaushik et al. [2002a] also introduced the

$F + B$ structural index for evaluating different classes of branching path expressions. Similarly to the A(k)-index, the proposed technique controls index size by relaxing the grouping rule and essentially trading index generality for size. As discussed earlier, however, it is not clear if this approach is suitable for *statistical* summaries of the underlying data, where the goal is to capture the statistical characteristics of the XML data graph and not to reduce the number of I/O operations during the evaluation of path queries.

8. CONCLUSIONS

The optimization of complex declarative queries over XML data depends crucially on the existence of effective synopses that enable accurate estimates for the selectivities of query constructs. In this article, we have described the XSKETCH graph-synopsis model for XML data graphs with raw data values. Our proposed summarization model is based on the generic graph-synopsis model, augmented with localized stability and value-distribution summaries to accurately capture the complex correlation patterns that can exist between and across path structure and element values in the data graph. We have developed a systematic XSKETCH estimation framework for complex path expressions as well as efficient XSKETCH construction algorithms. Results from our implementation have verified the effectiveness of our XSKETCH synopses, demonstrating their ability to yield consistent, low-error estimates for complex XPath expressions with branching and value predicates.

REFERENCES

ABOULNAGA, A., ALAMELDEEN, A. R., AND NAUGHTON, J. F. 2001. Estimating the Selectivity of XML Path Expressions for Internet Scale Applications. In *Proceedings of the 27th International Conference on Very Large Data Bases*.

ABOULNAGA, A. AND NAUGHTON, J. F. 2003. Building XML statistics for the hidden web. In *Proceedings of the Twelfth International Conference on Information and Knowledge Management*. 358–365.

ALTINEL, M. AND FRANKLIN, M. J. 2000. Efficient filtering of XML documents for selective dissemination of information. In *Proceedings of the 26th International Conference on Very Large Data Bases*. 53–64.

BOAG, S., CHAMBERLIN, D., FERNNDEZ, M. F., FLORESCU, D., ROBIE, J., AND SIMON, J. 2005. XQuery 1.0: An XML query language. W3C Candidate Recommendation. Go online to `www.w3.org`.

BRAY, T., PAOLI, J., SPERBERG-MCQUEEN, C. M., AND MALER, E. 2000. Extensible Markup Language (XML) 1.0 (second edition). W3C Recommendation. Go online to `www.w3.org/`.

BUNEMAN, P., DAVIDSON, S., HILLEBRAND, G., AND SUCIU, D. 1996. A query language and optimization techniques for unstructured data. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*.

CHAUDHURI, S., MOTWANI, R., AND NARASAYYA, V. 1999. On random sampling over joins. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*. 263–274.

CHEN, Z., JAGADISH, H. V., KORN, F., KOUDAS, N., MUTHUKRISHNAN, S., NG, R., AND SRIVASTAVA, D. 2001. Counting twig matches in a tree. In *Proceedings of the 17th International Conference on Data Engineering*. 595–604.

CLARK, J. 1999. XSL Transformations (XSLT), Version 1.0. W3C Recommendation. Go online to `www.w3.org`.

CLARK, J. AND DEROSE, S. 1999. XML path language (XPath), version 1.0. W3C Recommendation. Go online to `www.w3.org`.

DEROSE, S., MALER, E., AND ORCHARD, D. 2001. XML linking language (XLink), version 1.0. W3C Recommendation. Go online to `www.w3.org`.

DESHPANDE, A., GAROFALAKIS, M., AND RASTOGI, R. 2001. Independence is good: Dependency-based histogram synopses for high-dimensional data. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*.

FELLER, W. 1968. *An Introduction to Probability Theory and its Applications—Volume I*, John Wiley & Sons, New York, NY.

FOWLER, R. J., PATERSON, M. S., AND TANIMOTO, S. L. 1981. Optimal packing and covering in the plane are NP-complete. *Inform. Process. Lett. 12*, 3 (Jun.), 133–137.

FOX, B. 1966. Discrete Optimization Via Marginal Analysis. *Manage. Sci. 13*, 3, 211–216.

FREIRE, J., HARITSA, J. R., RAMANATH, M., ROY, P., AND SIMÉON, J. 2002. StatiX: Making XML count. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*. 181–191.

GAROFALAKIS, M. N. AND GIBBONS, P. 2001. Approximate Query processing: Taming the terabytes. In *Proceedings of the 27th International Conference on Very Large Data Bases*.

GIBBONS, P. B. 2001. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In *Proceedings of the 27th International Conference on Very Large Data Bases*. 541–550.

GOLDMAN, R. AND WIDOM, J. 1997. DataGuides: Enabling query formulation and optimization in semistructured databases. In *Proceedings of the 23rd International Conference on Very Large Data Bases*.

GONZALEZ, T. F. 1985. Clustering to minimize the maximum intercluster distance. *Theoret. Comput. Sci. 38*, 293–306.

JAGADISH, H., KOUDAS, N., MUTHUKRISHNAN, S., POOSALA, V., SEVCIK, K. C., AND SUEL, T. 1998. Optimal histograms with quality guarantees. In *Proceedings of the 24th International Conference on Very Large Data Bases*.

JAGADISH, H. V., AL-KHALIFA, S., CHAPMAN, A., LAKSHMANAN, L. V. S., NIERMAN, A., PAPARIZOS, S., PATEL, J. M., SRIVASTAVA, D., WIWATWATTANA, N., WU, Y., AND YU, C. 2002. TIMBER: A native XML database. *Internat. J. Very Large Data Bases 11*, 4, 274–291.

KAUSHIK, R., BOHANNON, P., NAUGHTON, J. F., AND KORTH, H. F. 2002a. Covering indexes for branching path queries. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*. 133–144.

KAUSHIK, R., SHENOY, P., BOHANNON, P., AND GUDES, E. 2002b. Exploiting local similarity for efficient indexing of paths in graph structured data. In *Proceedings of the 18th International Conference on Data Engineering*. 129–140.

LIM, L., WANG, M., PADMANABHAN, S., VITTER, J., AND PARR, R. 2002. XPathLearner: An on-line self-tuning markov histogram for XML path selectivity estimation. In *Proceedings of the 28th International Conference on Very Large Data Bases*. 442–453.

MEGIDDO, N. AND SUPOWIT, K. J. 1984. On the complexity of some common geometric location problems. *SIAM J. Comput. 13*, 1, 182–196.

MILO, T. AND SUCIU, D. 1999. Index structures for path expressions. In *Proceedings of the 7th International Conference on Database Theory* (*ICDT'99*). 277–295.

NAUGHTON, J. F., DEWITT, D. J., AND ET AL 2001. The niagara internet query system. *IEEE Data Eng. Bull. 24*, 2.

PAIGE, R. AND TARJAN, R. E. 1987. Three partition refinement algorithms. *SIAM J. Comput. 16*, 6 (Dec.), 973–989.

PEARL, J. 1988. *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. Morgan Kaufmann, San Francisco, CA.

POLYZOTIS, N. AND GAROFALAKIS, M. 2006. Xcluster synopses for structured xml content. In *Proceedings of the 22nd International Conference on Data Engineering*.

POLYZOTIS, N., GAROFALAKIS, M., AND IOANNIDIS, Y. 2004a. Approximate XML query answers. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*. 263–274.

POLYZOTIS, N., GAROFALAKIS, M., AND IOANNIDIS, Y. 2004b. Selectivity estimation for XML twigs. In *Proceedings of the 20th International Conference on Data Engineering*. 264–275.

POOSALA, V. AND IOANNIDIS, Y. E. 1997. Selectivity estimation without the attribute value independence assumption. In *Proceedings of the 23rd International Conference on Very Large Data Bases*. 486–495.

POOSALA, V., IOANNIDIS, Y. E., HAAS, P. J., AND SHEKITA, E. J. 1996. Improved histograms for selectivity estimation of range predicates. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*.

RAMANATH, M., ZHANG, L., FREIRE, J., AND HARITSA, J. 2005. IMAX: The big picture of dynamic XML statistics. In *Proceedings of the 21st International Conference on Data Engineering*. 273–284.

VITTER, J. S. AND WANG, M. 1999. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*. 193–204.

WANG, W., JIANG, H., LU, H., AND YU, J. X. 2003. Containment join size estimation: Models and methods. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*. 145–156.

WANG, W., JIANG, H., LU, H., AND YU, J. X. 2004. Bloom histogram: Path selectivity estimation for XML data with updates. In *Proceedings of the 30th International Conference on Very Large Data Bases*. 240–251.

WU, Y., PATEL, J. M., AND JAGADISH, H. 2002. Estimating answer sizes for XML queries. In *Proceedings of the 8th International Conference on Extending Database Technology* (EDBT '02). 590–608.

WU, Y., PATEL, J. M., AND JAGADISH, H. 2003. Structural join order selection for XML query optimization. In *Proceedings of the 19th International Conference on Data Engineering*. 443–454.

ZHANG, N., OZSU, M. T., ABOULNAGA, A., AND ILYAS, I. 2006. Xseed: Accurate and fast cardinality estimation for xpath queries. In *Proceedings of the 22nd International Conference on Data Engineering*.

Received March 2005; revised December 2005, May 2006; accepted May 2006