XCluster Synopses for Structured XML Content

Neoklis Polyzotis[*]
Univ. of California, Santa Cruz
alkis@cs.ucsc.edu

Minos Garofalakis
Intel Research Berkeley
minos.garofalakis@intel.com

Abstract

We tackle the difficult problem of summarizing the path/branching structure and value content of an XML database that comprises both numeric and textual values. We introduce a novel XML-summarization model, termed XClusters*, that enables accurate selectivity estimates for the class of twig queries with numeric-range, substring, and textual IR predicates over the content of XML elements. In a nutshell, an* XCluster *synopsis represents an effective clustering of XML elements based on both their structural and value-based characteristics. By leveraging techniques for summarizing XML-document structure as well as numeric and textual data distributions, our* XCluster *model provides the first known unified framework for handling path/branching structure and different types of element values. We detail the* XCluster *model, and develop a systematic framework for the construction of effective* XCluster *summaries within a specified storage budget. Experimental results on synthetic and real-life data verify the effectiveness of our* XCluster *synopses, clearly demonstrating their ability to accurately summarize XML databases with mixed-value content. To the best of our knowledge, ours is the first work to address the summarization problem for structured XML content in its full generality.*

1. Introduction

The Extensible Mark-up Language (XML) has rapidly evolved to an emerging standard for large-scale data exchange and integration over the Internet. Being self-describing and hierarchical in nature, XML provides a suitable data model that can adapt to several diverse domains and hence enable applications to query effectively the vast amount of information available on the Web.

Within the realm of XML query processing, XML summarization has emerged as an important component for the effective implementation of high-level declarative queries. In brief, a concise XML summary, or *synopsis*, captures (in limited space) the key statistical characteristics of the underlying data and essentially represents a highly-compressed, approximate version of the XML database. By executing a query over the synopsis, the optimizer can efficiently obtain selectivity estimates for different query fragments and thus derive the cost factors of candidate physical execution plans.

One of the key challenges in this important problem stems from the inherent complexity of XML data. More specifically, the information content of a semi-structured data store is encoded in both the *structure* of the XML tree as well as the *values* under different elements. Moreover, the content of XML elements is inherently heterogeneous, comprising of different types of values, e.g., integers, strings, or free text, that can be queried with different classes of predicates. As an example, an application may query an XML database with bibliographic information using the following path expression[1]: `//paper[year>2000][abstract ftcontains( synopsis , XML)]/title[contains(Tree)]`, which will select all titles of papers that were published after 2000, if their abstracts mention the terms “synopsis” and “XML” and their title contains the substring “Tree”. To enable low-error selectivity estimates for such queries, an XML summary clearly needs to capture the key correlations between and across the underlying path structure and value content, and provide accurate approximations for different types of value distributions. Given that real-life XML data sets contain highly heterogeneous content, it becomes obvious that realizing this important and challenging goal will provide crucial support for the effective optimization of XML queries in practice.

Related Work[2]**.** Summarizing a large XML data set for the purpose of estimating the selectivity of complex queries with value predicates is a substantially different and more difficult problem than that of constructing synopses for flat, relational data (e.g., [20, 24]). Recent research studies have targeted specific variants of the XML summarization problem, namely, structure-only summarization [1, 18, 25], or structure and value summarization only for numeric val-

[*] Supported in part by NSF Grant IIS-0447966

[1] In this example, we use the `ftcontains` operator from the Full-Text extensions to XPath [2]

[2] Due to space constraints, a more detailed discussion of related work is deferred to the full version of this paper.

ues [10, 13, 17, 19, 26]. Correlated Suffix Trees (CSTs) [7] and CXHist [14] are recently-proposed techniques that tackle the problem of XML selectivity estimation for substring predicates. CSTs, however, take a straightforward approach, simply treating string values as an extension of the XML structure; on the other hand, CXHist focuses on the simple case of fully-specified linear XPath expressions. It is not at all clear if these techniques can be extended to the more general problem of *twig queries* with predicates on *heterogeneous* value content.

Contributions. In this paper, we address the challenging and important problem of XML summarization in the presence of *heterogeneous value content*. We propose a novel class of XML synopses, termed XCLUSTERs, that capture (in limited space) the key characteristics of the path and value distribution of an XML database and enable selectivity estimates for twig queries with complex path expressions and predicates on element content. In sharp contrast to previous work, our proposed XCLUSTER model provides a unified summarization framework that enables a *single* XML synopsis to effectively support twig queries with predicates on numeric content (range queries), string content (substring queries), and/or textual content (IR-style queries). To the best of our knowledge, ours is the first attempt to explore the key problem of XML summarization in the context of heterogeneous element values. The main contributions of our work can be summarized as follows.

- XCLUSTER **Summarization Model.** Our proposed XCLUSTER synopses rely on a clean, yet powerful model of generalized *structure-value clusters,* a unified, clustering-based framework that can effectively capture the key correlations between and across structure and values of different types. To handle value-based approximations, our framework employs well-known techniques for numeric and string values, and introduces the class of *end-biased term histograms* for summarizing the distribution of unique terms within textual XML content.

- XCLUSTER **Construction Algorithm.** We introduce a set of compression operations for reducing the size of an XCLUSTER synopsis and develop a systematic metric for quantifying the effect of a compression step on the accuracy of the XML summary. Our proposed metric captures the impact on the structure-value clustering of the synopsis by taking into account the *localized* structural and value-based characteristics of the compressed area of the summary. Based on this framework, we propose an efficient, bottom-up construction algorithm that builds an effective XCLUSTER synopsis for a specific space budget by applying carefully selected compression steps on an initial detailed summary.

- **Experimental Study Verifying the Effectiveness of XCLUSTERs.** We validate our approach experimentally with a preliminary study on real-life and synthetic data sets. Our results demonstrate that concise XCLUSTERs constitute an effective summarization technique for XML data with heterogeneous content, enabling accurate selectivity estimates for complex twig queries with different classes of value predicates.

2 Preliminaries

Data Model. Following common practice, we model an XML document as a large, *node-labeled tree* $T(V, E)$. Each node $u \in V$ corresponds to an XML element and is characterized by a *label* (or, *tag*) assigned from some alphabet of string literals, that captures the element's semantics. Edges $(e_i, e_j) \in E$ are used to capture the containment of (sub)element e_j under e_i in the database. (We use label(e_i), children(e_i) to denote the label and set of child nodes for element node $e_i \in V$.) In addition, each element node e_i can potentially also contain a *value* of a certain type (denoted by value(e_i)); we assume the existence of a mapping function type from elements to a set of data types, such that type(e_i) is the data type of value value(e_i). (Elements with no values are mapped to a special null data type.) Our study considers the following set of possible data types for XML-element values:

- NUMERIC: Captures *numeric* element values; for instance, in a bibliographic database, NUMERIC values would include book prices, publication years, and so on. Following the usual conventions for numeric database attributes, we assume the NUMERIC values range in an integer domain $\{0 \ldots M - 1\}$.
- STRING: Captures *(short) string* values – in a bibliographic database, these would include author/publisher names and addresses, book titles, and so on.
- TEXT: Captures *free-text* element values – in our bibliographic database example, these would include book forewords and summaries, paper abstracts, and so on. Such textual values need to support an *IR-style, full-text querying* paradigm based on keyword/index-term search [3, 8]. Based on the traditional set-theoretic, *Boolean model* of IR, TEXT values are essentially Boolean vectors over an underlying dictionary of terms (where the i^{th} entry of the vector is 1 or 0 depending on whether the i^{th} term appears in the free-text data or not).[3]

As an example, Figure 1 depicts a sample XML data tree containing bibliographic data. The document consists of author elements, each comprising a name, and several paper and book sub-elements. Each paper and book comprises a (STRING-valued) title, a (NUMERIC-valued) year of publication, as well as (TEXT-valued) keywords,

[3] In future work, we intend to explore more flexible, *Vector-Space* IR models [4] of representing and querying TEXT values in XML documents, and their impact on our summarization framework.

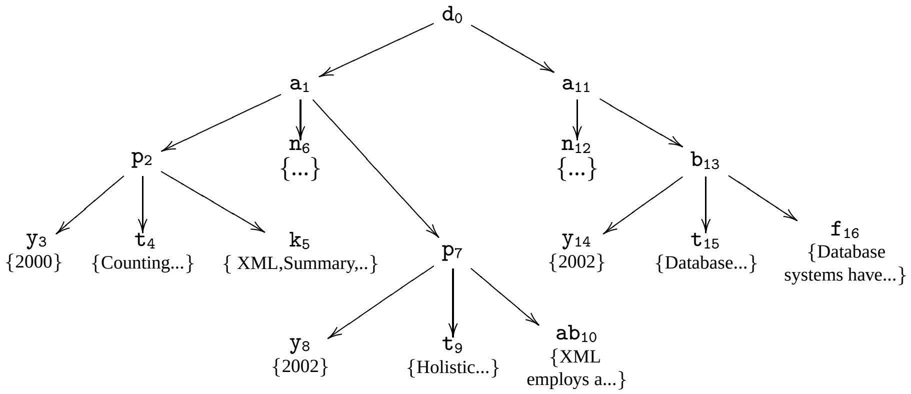


Figure 1. Example XML document.

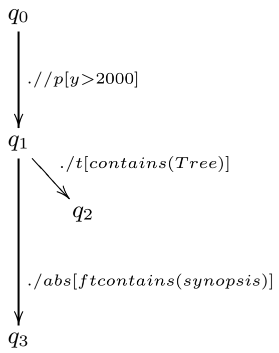


Figure 2. Example query.

abstract, and/or foreword sub-elements. Note that element nodes in the tree are named with the first letter of the element's tag plus a unique identifier.

We believe that the above set of value types adequately captures the bulk of real-world XML content. Textual information (i.e., STRING and TEXT values), in particular, is an integral part of real-life XML documents – this is clearly demonstrated by numerous recent research as well as standardization efforts that attempt to integrate XML query languages and query processing with substring and term-based search models [3, 8, 2]. Our work, however, is the *first* to consider the implications of different types of numeric and textual information on the difficult problem of effective XML summarization [10, 17].

Query Model. The focus of this paper is *twig queries* with *value predicates*. More specifically, a twig query Q is node- and edge-labeled tree $Q(V_Q, E_Q)$ where each node $q_i \in V_Q$ represents a query variable that is bound, during query evaluation, to a set of elements from the input document (we assume that q_0 is the root of the query and is always mapped to the root of the document). Figure 2 shows an example twig query over the sample document of Figure 1. An edge $(q_i, q_j) \in E_Q$ denotes a structural constraint between the elements of the source and target variable, specified by an XPath expression edge-path(q_i, q_j). In our work, we focus on XPath expressions that involve the child and descendant axis, wildcards, and optional predicates on path branches and element values. Conceptually, the evaluation of Q generates all possible assignments of elements to query variables, such that both (a) the *structural constraints* (as specified by the edge labels), and (b) the *value constraints* (as specified by the value predicates attached to query nodes), are satisfied. This set of possible assignments constitutes the set of *binding tuples*, and its cardinality is defined as the *selectivity* $s(Q)$ of the query.

The class of supported value predicates in our twig queries depends on the value types of the queried XML elements, and is defined as follows.

- NUMERIC *range predicates* of the general form $[l, h]$, that specify a certain range $[l, h]$ for the NUMERIC values of the designated XML elements; for example, find all the book elements with prices between \$60 and \$80.
- STRING *substring predicates* of the general form contains(qs), where qs denotes a query string. A substring predicate is satisfied by XML elements with STRING values that contain qs as a substring (i.e., similar to the SQL like predicate); for instance, return all books such that the publisher name contains the (sub)string "ACM".
- TEXT *keyword predicates* of the form ftcontains$(t_1, \ldots, t_k)$ (where $t_1, \ldots, t_k$ denote terms from the underlying term dictionary) specifying exact term matches; for example, find all paper elements with abstracts containing the terms "XML" and "synopsis". (Our techniques can also handle other Boolean-model predicates, such as set-theoretic notions of document-similarity [4, 5].)

3. XCLUSTER Synopsis Model

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

At a high level, several recently-proposed techniques for building statistical summaries for XML databases (including XSKETCHes [17, 19] and TREESKETCHes [18]) are all roughly based on the abstract "node-partitioning" idea described above. Unfortunately, none of these earlier research efforts considered the impact of *element values* of possibly different types (most importantly, strings and unstructured text), and their corresponding querying models on the underlying XML-summarization problem. Obviously, given the prevalence of textual data and the importance of supporting full-text search in modern XML data repositories [3, 8], this restriction severely limits the applicability of such earlier solutions in real-life problem scenarios.

The XCLUSTER **Synopsis Model.** Our proposed XCLUSTER synopses overcome the aforementioned limitations of earlier work by explicitly accounting for different element-value types and querying requirements during the XML-document summarization process. The key idea in XCLUSTERs is to form *structure-value clusters* that, essentially, group together (i.e., summarize) XML elements that are "similar" in terms of *both their XML-subtree structure and their element values*. More formally, given an XML document tree $T(V, E)$ with element values, we say that a node-partitioning graph synopsis of T is *type-respecting* if the underlying equivalence relation $R \subseteq V \times V$ respects both element labels *and* value types; in other words, if $(e_i, e_j) \in R$ then $\texttt{label}(e_i) = \texttt{label}(e_j)$ and $\texttt{type}(e_i) = \texttt{type}(e_j)$. (As previously, given a synopsis node v, $\texttt{label}(v)$ and $\texttt{type}(v)$ denote the common label and value type, respectively, of all data elements in $\texttt{extent}(v)$.) Our general XCLUSTER synopsis model is defined as follows.

Definition 3.1 *An* XCLUSTER *synopsis $\mathcal{S}$ for an XML document T is a (node- and edge-labeled) type-respecting graph-synopsis for T, where each node $u \in \mathcal{S}$ stores: (1) an element count* $\texttt{count}(u) = |\texttt{extent}(u)|$ *(also denoted as $|u|$); (2) per-edge (average) child counters* $\texttt{count}(u, v)$, *for each edge $(u, v) \in \mathcal{S}$, that record the* average number of children in $\texttt{extent}(v)$ *for each element in* $\texttt{extent}(u)$*; and, (3) a value summary* $\texttt{vsumm}(u)$ *that compactly summarizes the distribution of the collection of* $\texttt{type}(u)$ *values under all elements in* $\texttt{extent}(u)$. ▮

Figure 3 shows a possible XCLUSTER synopsis for our example document, where each cluster essentially represents elements of the same tag. Intuitively, each XCLUSTER node u corresponds to a structure-value cluster of $\texttt{count}(u)$ elements in the original document tree T. Similar to the TREESKETCH synopses, the tuple of edge-counters $(\texttt{count}(u, v_1), \texttt{count}(u, v_2), \ldots)$ to the children of u gives the structural *representative* (or, *centroid*) of the u-cluster; moreover, $\texttt{vsumm}(u)$ describes the distribution of element values (or, in a probabilistic sense, the value of an "*average element*") in the u-cluster[4]. The key idea is that all elements in the extent of node u are essentially approximated by $\texttt{count}(u)$ occurrences of the u-cluster centroid, and the key to effective summarization is hence to try to keep these clusters as "tight" as possible. This generalized structure-value clustering model is a key contribution of our work, providing a clean, unified framework for the effective summarization of both path structure and values of different types in XML databases.

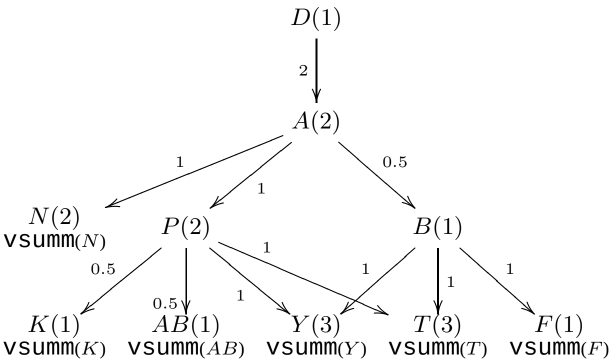


Figure 3. Example XCLUSTER for the data of Figure 1.

XCLUSTER **Value Summaries.** We have thus far been deliberately vague on the specifics of the $\texttt{vsumm}$ value summaries stored within XCLUSTER nodes. Not surprisingly, the specific value-summarization mechanism used differs across different value types; that is, the form of $\texttt{vsumm}(u)$ depends on $\texttt{type}(u)$. In what follows, we briefly discuss the different classes of value summaries implemented within our XCLUSTER framework; of course, it is always possible to extend our framework with support for different value-summarization techniques and/or additional types of element values.

• NUMERIC *value summaries* capture the frequency distribution of all numeric values in the extent of an XCLUSTER node. Summarizing numeric frequency distributions is a well-studied problem in relational approximate query processing, and several known tools can be employed, including histograms [21], wavelets [16], and random sampling [15]. For concreteness, we focus primarily on histograms as our main NUMERIC value summarization tool, although our ideas can easily be extended to other techniques.

• STRING *value summaries* capture the distribution of different substrings in the collection of strings corresponding to the (STRING-valued) XML elements in an XCLUSTER node's extent. Following earlier work on the problem of

[4] Note that, unlike the multi-dimensional histograms of [17], our XCLUSTER value summaries do not explicitly try to capture correlations between sets of values across different synopsis nodes. Uncovering such meaningful correlations across different value sets is itself a challenging problem, and the benefits of the resulting multi-dimensional summaries are often limited by the "curse of dimensionality"; further, designing effective multi-dimensional summaries for correlations spanning different value types (e.g., text and numeric data) is, to the best of our knowledge, an open problem.

substring selectivity estimation [6, 11], we employ *Pruned Suffix Trees (PSTs)* as our main summarization mechanism for STRING values and substring queries. Our model is based on a modification of the original PST proposal, where the PST records at least one node for each symbol that appears in the string distribution (this implies that the pruning threshold [11] is redundant). We have found that this modification yields accurate approximations, while avoiding large errors for negative substring queries.

• TEXT *value summaries* essentially summarize the collection of Boolean term vectors corresponding to the underlying set of elements in an XCLUSTER node's extent. Our basic summarization mechanism here (similar to the case of structure clusters) is the *centroid* of the underlying vector collection; in other words, letting $\mathbf{w}_1, \ldots, \mathbf{w}_k$ denote the set of Boolean term vectors corresponding to a TEXT XCLUSTER node u ($k = \texttt{count}(u)$), then we define $\texttt{vsumm}(u)$ as the vector $\overline{\mathbf{w}}$, where $\overline{\mathbf{w}}[t] = \sum_{i=1}^{k} \mathbf{w}_i[t]/k$ for each term t (thus, $\overline{\mathbf{w}}[t]$ is the fractional frequency of t in the underlying set of texts). In practice, however, the number of terms (i.e., the dimensionality of our vector-centroid summary) can be quite large, thus placing a significant storage burden on TEXT XCLUSTER nodes. Thus, our XCLUSTER synopses also employ a second-level of value summarization for TEXT nodes, where the vector centroid $\overline{\mathbf{w}}$ is itself compressed further using a novel technique, termed *end-biased term histograms*, that we introduce in this paper. Note that conventional histogramming techniques are likely to be ineffective for compressing such term vectors, since they optimize performance for range aggregates, whereas for TEXT values we are primarily interested in term-matching (i.e., "single-point") queries; for instance, by grouping consecutive frequencies in buckets, traditional histograms will almost certainly lose track of zero-valued entries (i.e., non-existent terms). It is interesting to note that recent studies in Information Retrieval have looked into the general problem of compressing the information stored in term vectors. These works, however, focus on dimensionality-reduction techniques for improving the quality of document clustering [12], or techniques for reducing the disk footprint of an IR engine while preserving relevance rankings [9, 23]. It is not clear, therefore, if they can be extended to the problem that we tackle in this paper, namely, selectivity estimation for point queries over term-vectors.

Briefly, our end-biased term histograms compress the term-vector centroid $\overline{\mathbf{w}}$ using a combination of (1) the *top-few term frequencies* in $\overline{\mathbf{w}}$ (which are retained exactly); and, (2) a *uniform bucket* that comprises a (lossless) *run-length compressed encoding* of the binary version of $\overline{\mathbf{w}}$ (i.e., where the entry for t is 1 if $\overline{\mathbf{w}}[t] > 0$ and 0 otherwise), along with an *average frequency* for all the non-zero terms in the bucket. To estimate the frequency $\overline{\mathbf{w}}[t]$ using an end-biased term histogram of $\overline{\mathbf{w}}$, we first try to lookup t in the explicitly-retained top frequencies; if that fails, we lookup t in the run-length compressed uniform bucket returning the average bucket frequency if the corresponding entry is 1, and 0 otherwise. Thus, by avoiding the grouping of values into bucket ranges and retaining a lossless representation of the $0/1$ uniform bucket, our end-biased term histograms circumvent the problems of conventional histograms giving an effective summarization mechanism for term vectors.

4. Building XCLUSTER Summaries

In this section, we tackle the challenging problem of effective synopsis construction, and introduce novel, efficient algorithms for building an accurate XCLUSTER summary for a given space-budget constraint.

At a high level, our goal during XCLUSTER construction is to group XML elements in a manner that results in "tight" structure-value clusters, i.e., XCLUSTER nodes that represent elements with high similarity in both their subtree structures and their value distributions in the original XML document. This natural analogy with clustering, however, highlights the key challenges of our XCLUSTER construction problem. First, clustering problems are generally known to be computationally intractable, even in the simple case of a fixed set of points in low-dimensional spaces [22, 27]. Second, our XCLUSTER nodes capture both *structure and value* distributions — this means that our clustering error metric has to appropriately combine both the structural- and value-summarization errors of a cluster, in a manner that accurately quantifies the effects of such errors on the final approximate query estimates.

Our proposed XCLUSTER construction algorithm is based on a generic *bottom-up clustering* paradigm: starting from a large, detailed *reference* XCLUSTER *summary* of the underlying XML document, our construction process builds a synopsis of the desired size by incrementally *merging* XCLUSTER *nodes* with similar structure and value distributions until the specified space constraint is met; in addition, our construction algorithm can also apply *value-summary compression* operations that further compress the value-distribution information stored locally at synopsis nodes (e.g., by merging similar adjacent buckets in a NUMERIC-values histogram). In what follows, we first describe these two key operations and their impact on the quality of clustering, and then introduce our proposed construction algorithm.

4.1. Merging XCLUSTER Nodes

Consider two nodes u and v with identical labels and value types (i.e., $\texttt{label}(u) = \texttt{label}(v)$ and $\texttt{type}(u) = \texttt{type}(v)$) in a given XCLUSTER synopsis $\mathcal{S}$, and let $\texttt{parents}(u)$ ($\texttt{children}(u)$) denote the set of parent (respectively, children) nodes of u in $\mathcal{S}$ (similarly for v). (Note that, since $\mathcal{S}$ is a general directed-graph structure, the parent and children sets for nodes u and v may overlap, i.e., u and v can have some common parents/children in $\mathcal{S}$.) As previously, we use $\texttt{vsumm}(u)$ and $\texttt{vsumm}(v)$ to denote the value-distribution summaries for the two XCLUSTER nodes. Our XCLUSTER-construction process can choose to apply a *node-merge operation* on such (label/type compatible) node pairs in order to further reduce the space requirements of the $\mathcal{S}$ synopsis.

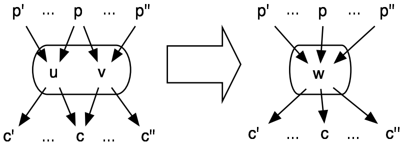


Figure 4. Node-merge operation.

Intuitively, such a node-merge operation creates a smaller synopsis $\mathcal{S}'$ from $\mathcal{S}$ (denoted by $\mathcal{S}' = \texttt{merge}(\mathcal{S}, u, v)$), in which the two original nodes u and v are replaced by a new node w representing the structure-value cluster of the combined collection of XML elements in u and v; that is, $\texttt{extent}(w) = \texttt{extent}(u) \cup \texttt{extent}(v)$. Of course, all edges in the original $\mathcal{S}$ synopsis are maintained in $\mathcal{S}'$, i.e., $\texttt{parents}(w) = \texttt{parents}(u) \cup \texttt{parents}(v)$ and $\texttt{children}(w) = \texttt{children}(u) \cup \texttt{children}(v)$. (This is shown pictorially in Figure 4.) Furthermore, the structural summary information for the new node w is computed in a natural manner (based on the cluster-centroid semantics described in Section 3), as an appropriately-weighted combination of the summary information in the two merged nodes u and v; more formally, we define $\texttt{count}(w) = |w| = |u| + |v|$, and, for each child node c and parent node p of w in the new synopsis, we define the edge counts from/to w as

$$\begin{aligned} \texttt{count}(w, c) &= \frac{|u|\texttt{count}(u, c) + |v|\texttt{count}(v, c)}{|w|} \quad \text{and} \\ \texttt{count}(p, w) &= \texttt{count}(p, u) + \texttt{count}(p, v), \end{aligned}$$

where, of course, $\texttt{count}(u, c) = 0$ ($\texttt{count}(p, u) = 0$) if $c \notin \texttt{children}(u)$ (respectively, $p \notin \texttt{parents}(u)$) (and, similarly for v). Recall that our XCLUSTER edge-counts $\texttt{count}(x, y)$ correspond to the average number of y-children per element of x — it is easy to see that the above formulas for the edge counts of the merged node w retain these average-count semantics in the resulting synopsis $\mathcal{S}'$.

The value-distribution summary information for w in $\mathcal{S}'$ is similarly computed by appropriately "fusing" the value-distribution summaries of u and v to produce a summary for the combined collection of element values in $\texttt{extent}(u) \cup \texttt{extent}(v)$. That is, we define $\texttt{vsumm}(w) = f(\texttt{vsumm}(u), \texttt{vsumm}(v))$, where the specifics of the value-summary fusion function $f()$ depend on the type of element values in nodes u, v.

- *If* $\texttt{type}(u) = \texttt{type}(v) = \text{NUMERIC}$, we build the combined histogram $\texttt{vsumm}(w)$ by merging bucket-count information from the individual histogram $\texttt{vsumm}(u)$ and $\texttt{vsumm}(v)$. This involves an initial *bucket alignment* step, where $\texttt{vsumm}(u)$ and $\texttt{vsumm}(u)$ acquire buckets on the same set of ranges by (potentially) splitting the ranges and counts of their existing buckets (based on conventional histogram-uniformity assumptions [21]); subsequently, the two histograms are merged to produce $\texttt{vsumm}(w)$ by summing the frequency counts across the aligned buckets.
- *If* $\texttt{type}(u) = \texttt{type}(v) = \text{STRING}$, we build the combined PST $\texttt{vsumm}(w)$ by starting out with an empty tree and simply inserting all substrings found in $\texttt{vsumm}(u)$ and $\texttt{vsumm}(v)$. The count for a substring s in $\texttt{vsumm}(w)$ is set equal to the sum of the individual counts for s in $\texttt{vsumm}(u)$ and $\texttt{vsumm}(v)$.
- *If* $\texttt{type}(u) = \texttt{type}(v) = \text{TEXT}$, then, assuming that $\texttt{vsumm}(u)$ and $\texttt{vsumm}(v)$ denote the Boolean term vector centroids for u and v, we define the combined centroid vector for w as a simple weighted combination of the two individual vectors; that is, $\texttt{vsumm}(w) = \frac{|u|}{|w|}\texttt{vsumm}(u) + \frac{|v|}{|w|}\texttt{vsumm}(v)$. (Merging of end-biased term histograms can be defined in a similar manner.)

Quantifying Node-Merging Approximation Error. Applying a node-merge operation on an XCLUSTER synopsis $\mathcal{S}$ to obtain a smaller synopsis $\mathcal{S}' = \texttt{merge}(\mathcal{S}, u, v)$ increases the approximation error in the resulting summary. Intuitively, this comes from fusing two structure-value clusters u, v in $\mathcal{S}$ into a single, "coarser" structure-value cluster w in $\mathcal{S}'$. The increase in approximation error when going from $\mathcal{S}$ to $\mathcal{S}'$ (denoted by $\Delta(\mathcal{S}, \mathcal{S}')$) comprises two key components: (1) *Structural clustering error* due to the fusion of the two structure centroids (i.e., edge-count tuples) for the clusters $\texttt{extent}(u)$ and $\texttt{extent}(v)$ into a single, weighted structure centroid for the combined cluster $\texttt{extent}(w) = \texttt{extent}(u) \cup \texttt{extent}(v)$ in $\mathcal{S}'$; and, (2) *Value clustering error* due to the merging of the two value summaries $\texttt{vsumm}(u)$ and $\texttt{vsumm}(v)$ (for values in $\texttt{extent}(u)$ and $\texttt{extent}(v)$, respectively) into a single value summary $\texttt{vsumm}(w)$ for the union of the two value collections.

One of the key challenges in our XCLUSTER synopsis-construction framework is to appropriately quantify and combine these two forms of error in a meaningful overall approximation-error difference $\Delta(\mathcal{S}, \mathcal{S}')$ between the two synopses $\mathcal{S}$ and $\mathcal{S}'$. Obviously, the problem is further complicated by the several different types of values and value predicates that our synopses need to support,

The basic idea in our approach is to quantify the increase in both structure and value clustering error in $\mathcal{S}'$ by essentially measuring their impact on the estimation errors for a collection of *atomic queries* involving the synopsis nodes affected by the merge operation. The key observation here is that our $\texttt{merge}(\mathcal{S}, u, v)$ operator has a very *localized* effect on the synopsis, essentially changing the edge-count and value-distribution centroids attached to nodes u and v. Thus, it is possible to capture the average behavior of arbitrary queries by measuring the estimation error on *simple, localized query paths* (or, atomic queries) in the affected parts of the summary. (Taking an analog from the domain of measurements, our atomic query paths are essentially a set of micro-benchmarks that allows us to quantify different aspects of a system's performance without a full-blown query benchmark.)

More specifically, to quantify the increase in estimation error, our set of atomic query paths comprises all paths of the form $u[p]/c$ and $v[p]/c$ in $\mathcal{S}$, where $c \in C_u \cup C_v$ (C_x is a shorthand for $\texttt{children}(x)$) and p is a simple, *atomic value predicate* on the underlying value-distribution summaries $\texttt{vsumm}(u)$ and $\texttt{vsumm}(v)$. (And, of course, the corresponding set of $w[p]/c$ paths in $\mathcal{S}' = \texttt{merge}(\mathcal{S}, u, v)$.) The exact definition of the atomic predicates p in our query paths obviously depends on the nature of the summary: For NUMERIC histograms, the atomic predicates correspond to all possible range predicates of the form $[0, h]$ over the domain of the summary;[5] for STRING PSTs, atomic predicates correspond to all substrings in the summary; and, for TEXT summaries, atomic predicates refer to all individual terms. Without going into the details of our estimation algorithms (Section 5), it is not difficult to see that (based on the average-count semantics for our synopsis edge counts) the average number of c elements reached per element of u by the $u[p]/c$ path in $\mathcal{S}$ is exactly $e_{\mathcal{S}}(u, p, c) = \sigma_p(u) \cdot \texttt{count}(u, c)$, where the selectivity $\sigma_p(u)$ of the atomic predicate p at u is estimated from $\texttt{vsumm}(u)$ (and, similarly, for $e_{\mathcal{S}}(v, p, c)$ and $e_{\mathcal{S}'}(w, p, c)$). Summing the squared atomic-query estimation errors over all possible (p, c) combinations for the merged nodes u, v, we obtain the overall increase in approximation error for the compressed summary $\mathcal{S}' = \texttt{merge}(\mathcal{S}, u, v)$ as

$$\begin{aligned}\Delta(\mathcal{S}, \mathcal{S}') &= |u| \sum_{p} \sum_{c \in C_u \cup C_v} (e_{\mathcal{S}}(u, p, c) - e_{\mathcal{S}'}(w, p, c))^2 \\ &\quad + |v| \sum_{p} \sum_{c \in C_u \cup C_v} (e_{\mathcal{S}}(v, p, c) - e_{\mathcal{S}'}(w, p, c))^2.\end{aligned}$$

Thus, by effectively quantifying the impact of a merged structure-value cluster on the error of all localized query estimates affected by the merge operation, our clustering error metric provides us with an intuitive, meaningful measure for unifying structure and value approximation errors and guiding the choice of compression operations during our bottom-up XCLUSTER construction algorithm. It is interesting to note that TREESKETCH construction [18] relies on a similar node-merging approach for building an effective structural summary within a specific space budget. The TREESKETCH algorithm, however, evaluates merge operations based on a *global structural* clustering metric that requires accessing a large, detailed count-stable summary [18] during the build process. Our proposed approach, on the other hand, relies on a *localized structure-value* clustering metric that uses the current synopsis as the point of reference; as a result, it is both efficient to compute and significantly less demanding in terms of memory overhead.

4.2. Compressing Value Summaries

As described above, fusing value-distribution summaries during structural node merges essentially preserves all the detail present in the original value summaries. In order to effectively compress the value information in XCLUSTER nodes (e.g., to meet a specified space budget), we now introduce appropriate *value compression operations* that can be applied to different types of value summaries. For a specific node $u \in \mathcal{S}$ with value summary $\texttt{vsumm}(u)$, a value compression operation results in a new synopsis $\mathcal{S}'$ where $\texttt{vsumm}(u)$ is a coarser approximation of the same value distribution. To quantify the error that is introduced by summary compression, we rely on the same metric as in the case of structural merges, i.e., we quantify $\Delta(\mathcal{S}, \mathcal{S}')$ as the sum-squared estimation error for the set of atomic queries $u[p]/c$. The key difference here, of course, is that the structure of the summary remains unchanged, so we only need the first summand in the above formula for $\Delta(\mathcal{S}, \mathcal{S}')$ (with $w = u$).

We introduce three value compression operations in our framework that cover the different types of value summaries: (a) hist_cmprs for NUMERIC nodes, (b) tv_cmprs for TEXT nodes, and (c) st_cmprs for STRING nodes.

• $\underline{\texttt{hist\_cmprs}(u, b)}$. Here, u is a node with a histogram value summary $\texttt{vsumm}(u)$ and b is a positive integer. This operation results in a new histogram $\texttt{vsumm}(u)$ that contains b buckets less than the original value summary. The new histogram can be constructed from the original distribution, if it is available, or it can be formed by performing b merge operations on adjacent bucket-pairs in $\texttt{vsumm}(u)$ (the latter can be implemented without storing the original distribution and is thus more efficient.)

• $\underline{\texttt{tv\_cmprs}(u, b)}$. Here, u is a summary node with a end-biased term histogram $\texttt{vsumm}(u)$ and b is a positive integer. Similar to hist_cmprs, this operation essentially reduces the number of singleton "buckets" for vsumm(u), i.e., the number of terms for which the synopsis records exact frequency information, by moving the b lowest-frequency indexed terms to the uniform term bucket and appropriately adjusting the corresponding average frequency (used to ap-

[5] Using atomic predicates over *ranges* of the histogram is needed in order to avoid introducing "holes" (i.e., zero-count ranges) in the merged histograms.

proximate all frequencies in the uniform bucket).

- st_cmprs(u, b). Here, u is a node with a PST summary vsumm(u) and b is a positive integer. This operation results in the pruning of b leaf nodes from the vsumm(u) PST, and hence in a coarser approximation of the underlying string distribution. We propose a novel pruning scheme for PST summaries that removes a specified number of nodes while trying to minimize the resulting estimation error. More concretely, we associate with each leaf node x a *pruning error* that quantifies the difference in estimates, before and after the pruning of x from the PST, for the sub-string that x represents. Our observation is that this pruning error is a good indicator of the importance of node x in the PST structure, or equivalently, how well the Markovian estimation assumption [11] applies at x. Our pruning scheme removes b nodes based on their pruning error (thus attempting to minimize the impact on estimation accuracy), while ensuring that the monotonicity constraint of the PST is always preserved. In the interest of space, the complete details of this scheme can be found in the full version of this paper.

4.3 XClusterBuild Algorithm

Having introduced our key component operations for reducing the size of an XCluster summary, we now describe our construction algorithm, termed XClusterBuild, for building an accurate XCluster synopsis within a specific storage budget.

The pseudo-code for XClusterBuild is depicted in Figure 5. The algorithm receives as input the XML data tree T, and two parameters, namely, B_{str} and B_{val}, that define the *structural*- and *value-storage* budget. In a nutshell, B_{str} specifies the storage for recording information on the graph synopsis (nodes + edges + edge-counts), while B_{val} specifies the space that will be devoted to value summaries. The algorithm constructs an initial reference synopsis (line 1), that represents a very detailed clustering of the XML database, and subsequently reduces its size using the operations that were introduced in the previous sections (lines 2-18). As shown, the last step proceeds in two phases :(1) a *structure-value merge* phase, where merge operations are used in order to reduce the structure of the synopsis within B_{str} space units, and (2) a subsequent *value-summary compression* phase, where the algorithm applies value-based operations in order to compress the storage of value-distribution summaries within B_{val} units. In short, the first phase generates a "tight" structure-value clustering of input elements, while the second phase builds value summaries that accurately approximate the content of different element clusters. As we will discuss later, these two properties have a strong connection to the assumptions of the XCluster estimation framework and are effectively the key for the construction of accurate synopses.

```
Procedure XClusterBuild($T$, $B_{str}$, $B_{val}$)
Input: XML Tree $T$; Structural budget $B_{str}$; Value budget $B_{val}$
Output: XCluster synopsis $S$
begin
1.Initialize $\mathcal{S}$ with the reference synopsis
/** (1) Structure-Value Merge **/
2.$l$ := 1; $Cand_{str}$ := build_pool($\mathcal{S}$, $H_m$, $l$)
3.while structural information in $S > B_{str}$ do
4.    while $|Cand_{str}| > H_l$ do
5.        Apply merge $m \in Cand_{str} : \frac{\Delta(\mathcal{S},\mathcal{S}')}{|\mathcal{S}|_{str} - |\mathcal{S}'|_{str}}$ is minimized
6.        Remove $m$ from $Cand_{str}$ and recompute losses
7.    end
8.    $l$ := 1 + (max level of new vertices in this stage)
9.    $Cand_{str}$ := build_pool($S$,$H_m$,$l$) // Replenish pool
10.end
/** (2) Value-Summary Compression **/
11.for each $u \in S$ with values do // Init heap
12.  $Cand_{val} \leftarrow$ (compression onvsumm$(u)$)
13.end
14.while value information is $S > B_{val}$ do
15.  Apply value compression $m \in Cand_{val} : \frac{\Delta(\mathcal{S},\mathcal{S}')}{|\mathcal{S}| - |\mathcal{S}'|}$ is minimized
16.  Let vsumm$(u)$ be the summary that $m$ compressed
17.  $Cand_{val} -= m$; $Cand_{val} += $ (compression on vsumm$(u)$)
18.done
19.return $S$
end
```

Figure 5. Algorithm XClusterBuild.

The key components of XClusterBuild, namely, the computation of the reference synopsis and the two compression stages, are discussed in detail in the paragraphs that follow. Before proceeding with our detailed discussion, we note that it is possible to invoke XClusterBuild with a unified total space budget B and let the construction process determine automatically the ratio of structural- to value-storage budget. One plausible approach, for instance, would be to perform a binary search in the range of possible B_{str}/B_{val} ratios, based on the observed estimation error on a sample workload. This topic, however, is beyond the scope of our current work and we intend to investigate it further in our future research.

Reference Synopsis Construction. Our reference synopsis is essentially a refinement of the lossless *count-stable* summary of the input data [18]. More concretely, each cluster groups elements that have the same number of children in any other summary node and is associated with a detailed content summary that approximates the distribution of element values with low error. Moreover, each cluster has exactly one incoming path in order to capture potential path-to-value correlations. This detailed clustering in our (large) reference summary obviously provides a very accurate approximation of the combined structural and value-based distribution information of the input XML tree. (Due to space constraints, we defer the complete details for our reference-synopsis construction to the full paper.)

Structure-Value Merge. The goal of this phase is to compress the structure of the computed summary within B_{str} space units while preserving the key correlations between and across the structural and value-based distribution. To achieve this, our algorithm applies a sequence of node-merge operations that are selected based on a *marginal losses* heuristic (line 5): among the candidate operations, the algorithm selects the merge m that yields the least distance $\Delta(\mathcal{S}, \mathcal{S}')$ (i.e., loss in accuracy) per unit of saved storage, that is, m minimizes $\Delta(\mathcal{S}, \mathcal{S}')/(|\mathcal{S}|_{str} - |\mathcal{S}'|_{str})$, where $\mathcal{S}'$ is the resulting synopsis and $|\mathcal{S}|_{str} - |\mathcal{S}'|_{str}$ is the space savings for structural information. In order to reduce the number of possible operations, our algorithm employs two key techniques. First, it only considers operations from a pool $Cand_{str}$ of at most H_m candidate merges, where H_m is a parameter of the construction process. The pool is maintained as a priority queue based on marginal gains, thus making it very efficient to select the most effective operation at each step. Once the top operation is applied, the algorithm re-configures the queue by computing the new marginal losses for operations in the neighborhood of the merged nodes, and continues this process until the size of $Cand_{str}$ falls below a specific threshold H_l. At that point, it rebuilds the pool with a new set of candidate operations (line 9) and begins a new round of cluster merges.

```
Procedure build_pool($\mathcal{S}$, $H_m$, $l$)
Input: XCLUSTER $\mathcal{S}$; Max heap size $H_m$; level $l$
Output: The pool $Cand_{str}$ of candidate merges
begin
1.$Cand_{str}$ := $\emptyset$
2.for each $(u, v) : u, v \in S$ do
3.  if $\texttt{label}(u) = \texttt{label}(v) \wedge level(u), level(v) \leq l$ then
4.      $m \leftarrow$ (merge operation on $u, v$)
5.      $Cand_{str}.push$(m)
6.      if $|Cand_{str}| > H_m$ then
7.          $Cand_{str} -= \left\{ m : m \text{ maximizes } \frac{\Delta(\mathcal{S},\mathcal{S}')}{|\mathcal{S}|-|\mathcal{S}'|} \right\}$
8.      endif
9.  endif
10.end
11.return $Cand_{str}$
end
```

Figure 6. Function build_pool.

The second technique concerns the initialization of the pool with candidate merge operations (function build_pool shown in Figure 6). Clearly, the number of candidate merges grows quadratically with the number of synopsis nodes and an exhaustive exploration of all possible operations may be prohibitively expensive (particularly during the first steps of the construction process that operate on the large reference synopsis). To address this issue, we employ a heuristic that considers merge operations in a bottom-up-fashion, starting from the leaf nodes of the summary and gradually moving closer to the root. More concretely, each synopsis node is assigned to a *level* based on the shortest outgoing path that leads to a leaf descendant. The key idea is that a merge of two nodes at level $l + 1$ is more likely to have a low Δ if their respective children have been merged at level l (this matches the intuition that two clusters are similar and hence can be merged if they point to similar children.) The first initialization of the pool considers merges among nodes at levels 0 (leafs) and 1 (parents of leafs); when the pool needs to be replenished with new candidate operations, the algorithm considers merges up to level $maxlevel + 1$, where $maxlevel$ is the maximum level of a newly created node from the previous pool. This heuristic is based again on the same intuition, namely, that merges at level l may increase the effectiveness of merges at level $l + 1$ (that holds the parents of the merged nodes).

Value-Summary Compression. The goal of this stage is to compress the *value-distribution information* of the synopsis within the specified value-storage budget. More concretely, the algorithm maintains a priority queue $Cand_{val}$ that contains the minimum marginal loss value-compression operation m (Section 4.2) for each $\texttt{vsumm}(u)$ in the synopsis. To ensure efficiency, our algorithm only considers candidate operations for a fixed value of b (typically, $b = 1$ in our experiments). Similar to the previous case, the candidate operations are ranked and applied according to their marginal losses, i.e., the value-structure distance $\Delta(S, S')$ between the current ($\mathcal{S}$) and the updated synopsis ($\mathcal{S}$'), normalized by the savings in storage $|\mathcal{S}| - |\mathcal{S}'|$ (line 20). Once an operation is applied, the algorithm updates the queue with a new operation for the modified value summary $\texttt{vsumm}(u)$, and repeats the process until the specific budget constraint is met (or, the queue becomes empty.)

5. XCLUSTER Estimation

Our proposed estimation framework is based on an extension of the estimation algorithm for the structural TREE-SKETCH synopses. More concretely, XCLUSTER estimation relies on the key concept of a query *embedding*, that is, a mapping from query nodes to synopsis nodes that satisfies the structural and value-based constraints specified in the query. As an example, Figure 7 shows the embedding of a sample query over a given XCLUSTER synopsis. Each node is annotated with the query variable that it maps to, while the edge-counts represent the average number of descendants per source element (we discuss their computation in the next paragraph). Overall, the set of unique embeddings provides the possible evaluations of Q on the underlying database, and the overall selectivity can thus be approximated as the sum of individual embedding selectivities.

To estimate the selectivity of an embedding, the

XCLUSTER algorithm employs the stored statistical information coupled with a generalized Path-Value Independence assumption that essentially de-correlates path distribution from the distribution of value-content. More formally, Path-Value Independence approximates the selectivity of a simple synopsis path $u[p]/c$ as $|u| \cdot \sigma_p(u) \cdot \texttt{count}(u, c)$, where the fraction $\sigma_p(u)$ can be readily estimated from the value summary $\texttt{vsumm}(u)$. (Note that we have already hinted at the above formula in our discussion of the distance metric in Section 4.) Based on this approximation, our estimation algorithm traverses the query embedding and combines edge-counts and predicate selectivities at each step in order to compute the total number of binding tuples. Returning to our example, consider nodes $\texttt{A} : q_1$ and $\texttt{C} : q_2$. Using path-value independence, the number of descendants in q_2 per element in q_1 is computed as $\texttt{count}(\texttt{A}, \texttt{B}) \cdot \texttt{count}(\texttt{B}, \texttt{C}) \cdot \sigma_{\texttt{C}}(p) = 10 \cdot 5 \cdot 0.1$, assuming that $\sigma_{\texttt{C}}(p) = 0.1$ based on $\texttt{vsumm}(\texttt{C})$. Similarly, the total descendant count from $\texttt{A} : q1$ to $\texttt{E}^{\texttt{a}} : q_3$ is estimated to be $\texttt{count}(\texttt{A}, \texttt{D}^{\texttt{a}}) \cdot \texttt{count}(\texttt{D}^{\texttt{a}}, \texttt{E}^{\texttt{a}}) = 10$. Hence, each element in $\texttt{A} : q_1$ will generate $10 \cdot 5 = 50$ binding tuples as it will be combined with every descendant in variables q_2 and q_3. Given that the root element in q_0 has 10 descendants in $\texttt{A} : q_1$, this will bring the total estimated number of binding tuples to $50 \cdot 10 = 500$.

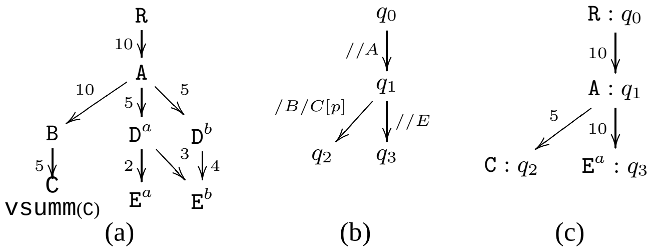


Figure 7. (a) XCLUSTER **(b) Query, (c) Embedding.**

Clearly, the accuracy of XCLUSTER estimation depends heavily on the validity of Path-Value Independence with respect to the underlying data. This assumption, however, is satisfied if element clusters are tight, i.e., they group together elements that have the same structure and similar distribution of values. As outlined in Section 4, this is exactly the criterion used by our construction algorithm in order to compress a synopsis within a specific storage budget. This observation forms the key link between the construction process and the proposed estimation framework, and provides an intuitive justification for the effectiveness of our construction algorithm.

6. Experimental Study

In this section, we present results from an empirical study that we have conducted using our novel XCLUSTER synopses over real-life and synthetic XML data sets.

6.1. Methodology

Techniques. We have completed a prototype implementation of the XCLUSTER model that is outlined in Section 3. Our prototype considers the construction of value-summaries under specific paths of the underlying XML, and supports histograms, PSTs, and end-biased term histograms as approximation methods for value distributions.

Data Sets. We use two data sets in our evaluation: a subset of the real-life *IMDB* data set, and the *XMark* synthetic benchmark data set. The main characteristics of our data sets are summarized in Table 6.1. The table lists the size of the XML document, the number of elements in each data set, the size of the reference synopsis, and the number of nodes (total and nodes with value summaries only.) As mentioned earlier, the reference synopsis contains value summaries for specific paths only which are provided as input to the construction algorithm. In our experiments, we included at least one path for each different type of values, for a total of 7 paths for IMDB and 9 for XMark. In both data sets, we observe that the reference synopsis is considerably smaller than the input data but may still be too large for the time and memory constraints of a query optimizer.

Workloads. We evaluate the accuracy of each synopsis on a workload of random positive twig queries, i.e, queries with non-zero selectivity. The workload is generated by sampling twigs from the reference synopsis and attaching random predicates at nodes with values. (The sampling of paths and predicates is biased toward high-counts.) Table 6.1 summarizes the characteristics of our workloads for the two data sets. We note that we have also performed experiments with negative workloads, i.e., queries with zero selectivity. We omit the results as they have shown that XCLUSTERs consistently yield close to zero estimates for all space budgets.

Evaluation Metric. We quantify the accuracy of an XCLUSTER summary with the *average absolute relative error* of result estimates over the queries of a workload. Given a query q with true result size c and estimated selectivity e, the absolute relative error is computed as $|c-e|/\max(c, s)$, where parameter s represents a *sanity bound* that essentially equates all low counts with a default count s and thus avoids inordinately high contributions from low-count path expressions. Following previous studies [17, 18], we set this bound to the 10-percentile of the true counts in the workload (i.e., 90% of the path expressions in the workload have a true result size $\geq s$).

6.2. Experimental Results

In this section, we present the results of our experimental study for evaluating the effectiveness of our novel

	File Size (MB)	# Elements	Ref. Size (KB)	# Nodes: Value/Total
IMDB	7.1	236822	473448	2037 / 3800
XMark	10	206130	890745	3593 / 16446

Table 1. Data Set Characteristics.

	Avg. Result Size	
	Struct	Pred
IMDB	6727	123
XMark	286341	1005

Table 2. Workload Characteristics.

XCluster summaries. In all the experiments that we present, we vary the structural summarization budget from 0KB to 50KB while keeping the value summarization budget fixed at 150KB. (Hence, the total summary size varies from 150KB to 200KB). We have empirically verified that these settings provide a good balance between structural and value-based summarization for the two data sets that we have used. As we have discussed in Section 4, the automated allocation of a total space budget remains an interesting problem that we intend to investigate further in our future research. We note that a structural space budget of 0KB represents the smallest possible structural summary that clusters elements based solely on their tags. In all experiments, we set the maximum and minimum size of the candidate merge pool to $H_m = 10000$ and $H_l = 5000$ respectively.

Figure 8 shows the average estimation error as a function of the structural budget size, for the two data sets IMDB and XMark. For each data set, the plot depicts the overall estimation error (Overall), the estimation error for queries with predicates on different types of value content (Numeric, Text, String), and queries without predicates (Struct). We discuss these results in more detail below.

Overall Estimation Error. Overall, the results indicate that our novel XClusters synopses constitute a very effective technique for estimating the selectivity of complex twig queries over structured XML content. For both IMDB and XMark, the overall estimation error falls below 10% for a modest total budget of 200KB, which represents a tiny fraction of the original data set size (Table 6.1). This level of performance is very promising, considering the complexity of our workload: multi-variable twig queries, with value predicates on different types of XML content. We also observe that the final error is considerably lower than the starting error of the smallest structural summary (73% for IMDB and 63% for XMark), which clearly demonstrates the effectiveness of our novel structure-value clustering metric in capturing the key correlations between the path and value distribution of the underlying data.

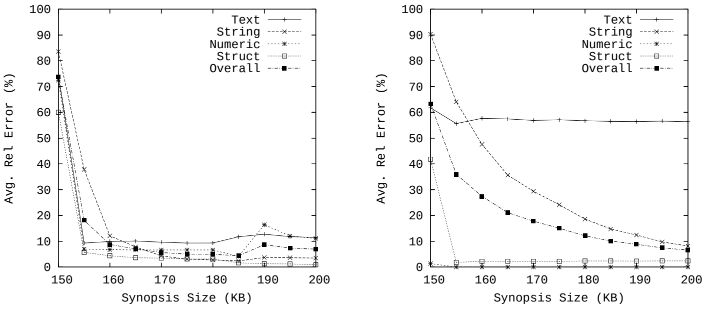
Estimation Error for Value Predicates. The estimation error for individual classes of value predicates follows the same decreasing trend as the previous case, demonstrating again the effectiveness of our summarization mechanism. A notable exception is the considerably increased error of more than 50% for queries with Text predicates over the XMark data set. This result, however, is an artifact of our test workload on XMark. More concretely, the XMark Text predicates have very low selectivities, thus leading to an artificially high relative error even though the absolute error is low. To verify this, Figure 9 lists the average absolute error for low-count queries over the two data sets, when the synopsis size is fixed at 200KB. (A query is included in these results if its true selectivity is below the sanity bound s.) As the results indicate, Text queries on XMark have an average absolute error of 1.09 tuples per query, which, combined with the average true selectivity of 3 tuples, yields the artificially high relative error.

Another interesting point concerns the estimation error for numeric predicates in the IMDB data set. More concretely, we observe an increase in the average error when the structural storage budget is varied from 35KB to 40KB. In this case, the compressed structure in the 35KB synopsis effectively leads to a lower number of nodes with numeric values, that are however assigned the same budget of 150KB for value-based summarization. This essentially results in an increased number of buckets per numeric histogram, thus yielding lower estimation errors for the smaller synopsis.

Error for Structural Queries. Finally, it is interesting to note the effectiveness of our novel localized clustering metric in the structural summarization of XML data. For both data sets, the estimation error for structural queries remains well below 5% for modest structural budgets (10KB–50KB), and is comparable to the original TreeSketch summaries [18] that target specifically the structural summarization problem. Moreover, the results indicate that our *localized* Δ metric is equally effective to the *global* TreeSketch clustering metric, which quantifies the differences between successive compression steps based on a count-stable summary and hence requires repeated accesses to the (potentially large) reference synopsis.

7. Conclusions

The accurate summarization of XML data with heterogeneous value content remains a critical problem in the effective optimization of real-world XML queries. In this paper, we propose the novel class of XCluster synopses that enable selectivity estimates for complex twig queries with predicates on numeric (range queries), string (substring queries), and textual content (IR queries). We de-



(a) (b)

Figure 8. XCLUSTER **relative estimation error for complex twig queries: (a) IMDB, (b) XMark.**

	Avg. Absolute Error	
	IMDB	XMark
Numeric	0.015	0
String	5.12	0.5
Text	0.18	1.09

Figure 9. Absolute estimation error for low-count queries: (a) IMDB, (b) XMark.

fine the XCLUSTER model and develop a systematic construction algorithm for building accurate summaries within a specific space budget. Our results on real-life and synthetic data sets verify the effectiveness of our approach.

References

[1] A. Aboulnaga, A. R. Alameldeen, and J. F. Naughton. Estimating the Selectivity of XML Path Expressions for Internet Scale Applications. In *VLDB*, 2001.

[2] S.Amer-Yahia, C.Botev, S.Buxton, P. Case, J. Doerne, D. McBeath, M. Rys, and J. Shanmugasundaram. XQuery 1.0 and XPath 2.0 Full-Text. W3C Working Draft.

[3] S. Amer-Yahia, L. V. S. Lakshmanan, and S. Pandit. FleXPath: Flexible Structure and Full-Text Querying for XML. In *ACM SIGMOD* , 2004.

[4] R. Baeza-Yates and B. Ribeiro-Neto. *"Modern Information Retrieval"*. ACM Press, Addison-Wesley Publishing Company, 1999.

[5] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. "Syntactic Clustering of the Web". In *WWW*, 1997.

[6] S. Chaudhuri and V. Ganti and L. Gravano. Selectivity Estimation for String Predicates: Overcoming the Underestimation Problem. In *ICDE*, 2004.

[7] Z. Chen, H. V. Jagadish, F. Korn, N. Koudas, S. Muthukrishnan, R. Ng, and D. Srivastava. Counting Twig Matches in a Tree. In *ICDE*, 2001.

[8] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. Xsearch: A semantic search engine for xml. In *VLDB*, pages 45–56, 2003.

[9] M Franz and J. S. McCarley. How Many Bits are Needed to Store Term Frequencies?. In *SIGIR*, 2002.

[10] J. Freire, J. R. Haritsa, M. Ramanath, P. Roy, and J. Siméon. StatiX: Making XML Count. In *ACM SIGMOD* , 2002.

[11] H. Jagadish, R. T. Ng, and D. Srivastava. Substring Selectivity Estimation. In *PODS*, 1999.

[12] X. He and D. Cai and H. Liu and W.Y. Ma. Locality preserving indexing for document representation. In *SIGIR*, 2004.

[13] L. Lim, M. Wang, S. Padmanabhan, J. Vitter, and R. Parr. XPathLearner: An On-Line Self-Tuning Markov Histogram for XML Path Selectivity Estimation. In *VLDB*, 2002.

[14] L. Lim, M. Wang, J. Vitter. CXHist : An On-line Classification-Based Histogram for XML String Selectivity Estimation. In *VLDB*, 2005 (To appear).

[15] R. J. Lipton, J. F. Naughton, D. A. Schneider, and S. Seshadri. Efficient sampling strategies for relational database operations. *Theoretical Comput. Sci.*, 116(1 & 2), 1993.

[16] Y. Matias, J. S. Vitter, and M. Wang. Wavelet-Based Histograms for Selectivity Estimation. In *ACM SIGMOD* , 1998.

[17] N. Polyzotis and M. Garofalakis. Structure and Value Synopses for XML Data Graphs. In *VLDB*, 2002.

[18] N. Polyzotis, M. Garofalakis, and Y. Ioannidis. Approximate XML Query Answers. In *ACM SIGMOD* , 2004.

[19] N. Polyzotis, M. Garofalakis, and Y. Ioannidis. Selectivity Estimation for XML Twigs. In *ICDE*, 2004.

[20] V. Poosala and Y. E. Ioannidis. Selectivity Estimation Without the Attribute Value Independence Assumption. In *VLDB*, 1997.

[21] V. Poosala, Y. E. Ioannidis, P. J. Haas, and E. J. Shekita. Improved Histograms for Selectivity Estimation of Range Predicates. In *ACM SIGMOD* , 1996.

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

[23] T Sakai and K Sparck-Jones. Generic summaries for indexing in information retrieval. In *SIGIR*, 2001.

[24] J. S. Vitter and M. Wang. Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets. In *ACM SIGMOD* , 1999.

[25] W. Wang, H. Jiang, H. Lu, and J. X. Yu. Containment Join Size Estimation: Models and Methods. In *ACM SIGMOD*, 2003.

[26] Y. Wu, J. M. Patel, and H. Jagadish. Estimating Answer Sizes for XML Queries. In *EDBT'02*, 2002.

[27] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. In *ACM SIGMOD* , 1996.