Online Appendix to: XSKETCH Synopses for XML Data Graphs

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

A. REFINEMENT OPERATION DETAILS

A.1 Structural Refinements

A.1.1 *The* `B_Stabilize`($\mathcal{X}S(G)$, u, v) *Operation.* The pseudocode for the `B_Stabilize` operation is depicted in Figure A.1. The procedure basically backward-stabilizes the set of data elements in `extent`(u) with respect to the set `extent`(v) (Definition 3.2) by taking the intersection with `succ(extent`(v)), the set of child elements of nodes in `extent`(v) in the data graph. This results in a node u_1 that contains all elements in u reached by v, and a second node u_2 with the remaining elements. Note that the `B_Stabilize` operation can affect the B- or F-stability of other edges that were attached to node u in the original synopsis; for example, although u may not have been F-stable with respect to a child node w, u_1 or u_2 may very well be F-stable with respect to w. The `add_node`() subroutine takes care of connecting u_1 and u_2 to the appropriate parent and child nodes of u in the final synopsis and correctly setting the stability labels on the resulting edges.

To complete the operation, the algorithm performs two postprocessing steps, termed split_value_summaries() and validate_value_summaries(), that update value distribution information in the area of the refined node. In particular, the split of an existing synopsis node u introduces two issues for the recorded value summaries: (a) if u has an attached value summary, then it becomes necessary to initialize the value summaries for the newly created nodes, and (b) the split can affect the stabilities of other incoming or outgoing edges, and thus may modify the correlation scopes of value summaries in the neighborhood of u. Our two postprocessing steps (described in more detail below) address exactly these two points and are obviously included in all the structural refinements that we introduce.

—split_value_summaries(u,$\{u_i\}$). This step initializes the value summaries of the newly formed nodes u_i. Each new summary is assigned one bucket worth of memory and if the histogram of u contains more than $|\{u_i\}|$ buckets, then the additional memory is distributed among the new histograms using a greedy procedure based on marginal gains (similar to Deshpande et al.

procedure B_Stabilize($\mathcal{X}S(G)$, u, v)
Input: XSKETCH synopsis $\mathcal{X}S(G)$; summary graph node u and parent node v of u such that $\mathbf{B} \notin \texttt{label}(v, u)$.
Output: Refined XSKETCH $\mathcal{X}S(G)$ that results from splitting node u into u_1 and u_2 to make it B-stable with respect to v.
begin
1. u_1 := new_node($\mathcal{X}S(G)$); u_2 := new_node($\mathcal{X}S(G)$) // create new nodes to replace u
2. extent(u_1) := succ(extent(v)) ∩ extent(u) // (v, u_1) will be **B**-stable
3. extent(u_2) := extent(u) − extent(u_1) // u_2 contains the remaining elements
4. remove_node($\mathcal{X}S(G)$, u)
5. add_node($\mathcal{X}S(G)$, u_1) // add new nodes and check stabilities with neighbors
6. add_node($\mathcal{X}S(G)$, u_2)
7. split_value_summaries(u, $\{u_1, u_2\}$) // allocate value summaries for new nodes
8. validate_value_summaries(u) // validate value summaries of neighboring nodes
end

Fig. A.1. The B_Stabilize operation.

[2001]). We note that the actual meaning of *bucket* depends on the implementation of the value summaries: in distinct sampling, for example, one bucket corresponds to a prespecified number of samples, while in range histograms it carries the conventional meaning of a value partition.

—validate_value_summaries(u). This step examines the value summaries in the neighborhood of the split node u, and modifies their dimensions according to the (possibly) modified stability conditions. More specifically, u's split can "break" edge stabilities to parent and children nodes and thus may shrink the stable neighborhoods of ancestors and descendants. For each affected node, the validate_value_summaries() step examines the corresponding value histogram (if it exists) and removes any dimensions that become invalid, that is, dimensions that refer to nodes outside of the new stable neighborhood. The new histogram is computed simply by taking the marginal on the remaining valid dimensions, which form the new correlation scope. This postprocessing step, therefore, is applied only to nodes in the "reverse" stable neighborhood of u, that is, nodes that contain u in their original stable neighborhood. Formally, this is defined as $F \cup B$, where F is the set of nodes that reach u through forward-stable paths (including u itself) and B is the set of nodes that can be reached from a node in F through a backward-stable path.

A.1.1.1 *Time and Space Complexity.* Let odeg(x) denote the *out-degree* of a node x in the XML data graph, and, for an XSKETCH node u, define odeg(u) as the *average out-degree* of data nodes in extent(u), that is, $\sum_{x \in u}$ odeg(x)/count(u). The average *in-degree* ideg(u) of an XSKETCH node u is defined similarly. The key data structures that our B_Stabilize algorithm needs to maintain in memory (not necessarily at the same time) are (1) a hash table for the data elements in the extent of the split XSKETCH node u (to efficiently compute the set intersection/difference operations in B_Stabilize); and, (2) hash tables for the elements in succ(extent(u)) and pred(extent(u)) (to efficiently add the

procedure F_Stabilize($\mathcal{X}S(G)$, u, w)
Input: XSKETCH synopsis $\mathcal{X}S(G)$; summary graph node u and child node w of u where $\mathbf{F} \notin \texttt{label}(u, w)$.
Output: Refined XSKETCH $\mathcal{X}S(G)$ that results from splitting node u into u_1 and u_2 to make it F-stable with respect to w.
begin
1. u_1 := new_node($\mathcal{X}S(G)$); u_2 := new_node($\mathcal{X}S(G)$) // create new nodes to replace u
2. extent(u_1) := pred(extent(w)) $\cap$ extent(u) // (u_1, w) will be **F**-stable
3. extent(u_2) := extent(u) $-$ extent(u_1) // u_2 contains the remaining elements
4. remove_node($\mathcal{X}S(G), u$)
5. add_node($\mathcal{X}S(G), u_1$) // add new nodes and check stabilities with neighbors
6. add_node($\mathcal{X}S(G), u_2$)
7. split_value_summaries(u, $\{u_1, u_2\}$) // allocate value summaries for new nodes
8. validate_value_summaries(u) // validate value summaries of neighboring nodes

end

Fig. A.2. The F_Stabilize operation.

required edges in the add_node() computations). Thus, the space complexity of the B_Stabilize procedure is $O(\sum_{x \in u} \max\{\texttt{odeg}(x), \texttt{ideg}(x)\}) = O(\texttt{count}(u) \max\{\texttt{odeg}(u), \texttt{ideg}(u)\})$. Given such hash-table structures, all the structural operations in B_Stabilize can be performed in a single pass over the data elements in the parents and children of node u (including v) in the XSKETCH summary. Consequently, the overall time complexity for all structural operations in B_Stabilize is $O(\texttt{count}(u) \max\{\texttt{odeg}(u), \texttt{ideg}(u)\} + \sum_{v_i \in \texttt{parents}(u)} \texttt{count}(v_i) + \sum_{w_j \in \texttt{children}(u)} \texttt{count}(w_j))$. The complexity of the value-summary operations obviously depends on the specific type of summary employed in the XSKETCH. Letting b_u denote the number of buckets in node u and $C_{N,b}$ be the cost of building a summary with b buckets out of N "base" data values,[5] the cost of the split_value_summaries() operation can be upper-bounded by $O(b_u C_{N,b_u})$. For validate_value_summaries(), our B_Stabilize algorithm, in the worst case, may need to project/marginalize each value summary in $F \cup B$ to a smaller correlation scope (an operation with cost linear in the number of buckets). Letting $b_{F \cup B}$ denote the average number of value-summary buckets per XSKETCH node in $F \cup B$, this cost can be upper bounded by $O(|F \cup B| b_{F \cup B})$. Thus, the overall time complexity of the value-summary operations in B_Stabilize is $O(b_u C_{N,b_u} + |F \cup B| b_{F \cup B})$.

A.1.2 *The* F_Stabilize($\mathcal{X}S(G), u, w$) *Operation.* Figure A.2 shows the pseudocode for the F_Stabilize operation. The basic idea is to forward-stabilize the set of data elements in extent(u) with respect to the set extent(w) (Definition 3.2) by taking the intersection with pred(extent(w)), the set of all parent elements of nodes in extent(v) in the data graph. This results in a node u_1 that

[5]As an example, for one-dimensional *V-optimal* histograms $C_{N,b} = O(N^2 B)$ [Jagadish et al. 1998], whereas for a distinct sample summary (of any size b) $C_{N,b} = O(N)$ [Gibbons 2001]. As we discuss in Section H.2, N here corresponds to the number of bucketsin a detailed *"reference synopsis"* used in our construction algorithm.

contains all elements in u that have at least one child in w, and a second node u_2 representing the remaining elements. Once again, an `add_node()` subroutine takes care of connecting u_1 and u_2 to the appropriate neighbors of u and correctly setting the stability labels on the resulting edges. The time and space complexities of our `F_Stabilize` procedure are essentially identical to those for `B_Stabilize`.

A.1.3 *The* `B_Split`$(\mathcal{X}S(G), u, \{v_i\})$ *Operation.* Figure A.3 shows the detailed pseudocode for the `B_Split` operation. The procedure starts by backward-stabilizing the data elements in `extent`(u) into subsets e_i, such that each e_i is backward-stable with respect to the same subset of parents from $\{v_1, \ldots, v_n\}$; that is, all elements in e_i have parent elements in the same subset $parents(e_i)$ of $\{v_1, \ldots, v_n\}$ (Steps 1–9). The next stage of the operation merges together those partitions e_i and e_j that share parent summary nodes, that is, $parents(e_i) \cap parents(e_j) \neq \phi$; the result of this merge stage is a (possibly) smaller set of partitions $\{e_i\}$ that have disjoint *parent* sets (i.e., the node clusters of $\{v_i\}$) (Steps 10–20). At the final step, the operation groups the new e_i subsets into two buckets that represent the final result nodes u_1 and u_2 of the `B_Split` operation (Steps 21–25). This grouping process for u_i nodes relies on a simple clustering algorithm that tries to minimize the overall variance of the metric $\texttt{ratio}(v_k, e_i) = \frac{sel(v_k/e_i)}{\texttt{count}(v_k)}$ in both of the resulting partitions; this is done by sorting the e_i's based on their average `ratio` values and selecting the best partition boundary in that sequence (Steps 21–23). (Note that estimation inaccuracies due to large `ratio` variances *within individual* e_i *groups* can be resolved through future backward-stabilization operations.) From a relational perspective, the `B_Split` operation simulates a histogram-bucket split, where similar frequencies are grouped together, so that the uniformity assumption is more accurate within each bucket [Poosala and Ioannidis 1997; Poosala et al. 1996]

A.1.3.1 *Time and Space Complexity.* As with our previous refinement operations, the main data structures required by `B_Split` are hash tables on the data elements in the extent of the node u being split, as well as the parents and children of those elements in the data graph; thus, the working space complexity of `B_Split` is $O(\texttt{count}(u)\ \max\{\texttt{odeg}(u), \texttt{ideg}(u)\})$. Given such hash-table structures, the required stabilization and edge-addition operations can be done in a single pass over the elements in parent and child nodes of u in the XSKETCH. Among the remaining structural `B_Split` operations, the dominant time complexity comes from the partition-merging loop—since there can be at most `count`(u) partitions, this merging cost can be upper-bounded by $O(\texttt{count}(u)^2)$. (The sorting of the partitions requires $O(\texttt{count}(u)\log(\texttt{count}(u)))$ time, whereas the best-variance split can be found in linear $O(\texttt{count}(u))$ time using a linear preprocessing step to enable the computation of the variance in constant time for each possible split.) The overall time complexity for the structural operations in our `B_Split` procedure is $O(\texttt{count}(u)\ \max\{\texttt{odeg}(u), \texttt{ideg}(u)\} + \sum_{v_i \in \texttt{parents}(u)} \texttt{count}(v_i) + \sum_{w_j \in \texttt{children}(u)} \texttt{count}(w_j) + \texttt{count}(u)^2)$. The cost of the value-summary operations in `B_Split` is essentially identical to that in our earlier analysis for the `B_Stabilize` refinement.

procedure B_Split($\mathcal{X}S(G)$, u, $\{v_i\}$)
Input: XSKETCH synopsis $\mathcal{X}S(G)$; summary graph node u that will be split;
parent nodes $\{v_i\}$ of u where $\mathbf{B} \notin \texttt{label}(v_i, u)$.
Output: Refined XSKETCH synopsis $\mathcal{X}S(G)$ that results from splitting node u into
u_1 and u_2 to improve count uniformity.
begin
1. t := extent(u); $parents(t) := \phi$
2. $p_u := \{t\}$
3. **for each** $v \in \{v_1, \ldots, v_n\}$ **do**
4. $tmp := p_u$
5. **for each** $e \in p_u$ **do** // stabilize e with respect to v
6. $e^s := e \cap$ succ(extent(v)); $parents(e^s) := parents(e) \cup \{v\}$
7. $e^r := e - e^s$; $parents(e^r) := parents(e) - \{v\}$
8. $tmp := tmp \cup \{e^s, e^r\} - \{e\}$
9. **endfor**
10. $p_u := tmp$
11. **endfor**
12. **if** $(\exists e_\phi \in p_u : parents(e_\phi) = \phi)$ **then** // elements that do not descend from any of $\{v_i\}$
13. u_3 := new_node($\mathcal{X}S(G)$); extent(u_3) := e_ϕ; add_node($\mathcal{X}S(G)$,u_3)
14. $p_u := p_u - \{e_\phi\}$
15. **endif**
16. **for each** $e_i, e_j \in p_u, e_i \neq e_j$ **do** // merge partitions with overlapping parent sets
17. **if** $(parents(e_i) \cap parents(e_j) \neq \phi)$ **then**
18. $e_{ij} := e_i \cup e_j$
19. $parents(e_{ij}) := parents(e_i) \cup parents(e_j)$
20. $p_u := p_u \cup \{e_{ij}\} - \{e_i, e_j\}$
21. **endif**
22. **endfor**
23. let $\texttt{ratio}(e_i) = \frac{\sum_{v_k \in parents(e_i)} sel(v_k/e_i)}{\sum_{v_k \in parents(e_i)} \texttt{count}(v_k)}$, and let $< e_1, \ldots, e_l >$ be the list of partitions in p_u in order of decreasing ratio's
24. u_1 := new_node($\mathcal{X}S(G)$); u_2 := new_node($\mathcal{X}S(G)$) // two new nodes that replace u
25. $m := \underset{1 \leq m < l}{\text{minarg}} \left(Var(\texttt{ratio}(e_1), \ldots, \texttt{ratio}(e_m)) + Var(\texttt{ratio}(e_{m+1}), \ldots, \texttt{ratio}(e_l)) \right)$
26. extent(u_1) := $\bigcup_{1 \leq i \leq m} e_i$; extent($u_2$) := $\bigcup_{m < i \leq l} e_i$
27. remove_node($\mathcal{X}S(G)$,u);
28. add_node($\mathcal{X}S(G)$,u_1); add_node($\mathcal{X}S(G)$,u_2);
29. split_value_summaries(u,$\{u_1, u_2, u_3\}$) // allocate value summaries for new nodes
30. validate_value_summaries(u) // validate value summaries of neighboring nodes
end

Fig. A.3. The B_Split operation.

A.2 Value Refinements

A.2.1 *The* Value_Refine(u, k) *Operation.* This operation allocates k additional "buckets" to the value histogram of node u in order to capture more effectively the underlying distribution of element values. In this manner, it attacks the implicit assumption that the recorded value summaries are accurate approximations of the XML value content. Note that the interpretation of

buckets and the actual use of extra memory depends on the implementation of the value summary.

A.2.1.1 *Time and Space Complexity.* Following the notation that we have introduced previously, we use B to denote the number of existing "buckets" and N for the number of base values. Clearly, the complexity of `Value_Refine` depends on the approximation method that implements the value-summaries in the XSKETCH synopsis. A crucial point, in particular, is the possibility of an *incremental* update to the refined value summary versus its reconstruction from the N base values and the updated number of buckets. As an example of the former, consider the application of `Value_Refine` on a single-dimensional *max-diff* histogram [Poosala et al. 1996]. When the histogram for u is first initialized, it is possible to compute the pair-wise frequency differences of the N values and store them in a min-heap along with the value-summary. In this fashion, `Value_Expand` can select very efficiently the boundaries for the additional k buckets by popping from the heap the k smallest such differences. This implies a space complexity of $O(N)$ and a time complexity of $O(klogN)$. A *V-Optimal* histogram [Jagadish et al. 1998], on the other hand, cannot accommodate an incremental update of the value-summary and requires complete reconstruction; this translates to a space complexity of $O(N(B+k))$ and a time-complexity of $O(N^2(B+k))$. (The tradeoff in this case is that *V-Optimal* histograms enable an optimal approximation of the underlying value distribution.)

A.2.1 *The* `Value_Expand`(u, v, k) *Operation.* Here, v is an XSKETCH node (whose elements have values) that lies within the stable twig neighborhood but outside the correlation scope of u; that is, $v \in (\texttt{STN}(u) - \texttt{dep}(u))$. By our estimation process (Section 4), since $v \notin \texttt{dep}(u)$, the number of elements in $\texttt{extent}(u)$ that descend from (or lead to) elements in $\texttt{extent}(v)$ with specific values is approximated based on the Value Independence Outside Correlation Scope assumption (Assumption 3), which postulates independence between the distributions in u and v. The `Value_Expand`(u, v, k) operation lifts this assumption by explicitly adding v to the correlation scope of u, which effectively amounts to adding a new dimension to u's histogram for the values under v. To support this expansion, the operation allocates k additional buckets to the histogram in order to refine the approximation of the resulting distribution. (Once again, these histogram refinements depend on the actual implementation of the value summary.)

A.2.1.2 *Time and Space Complexity.* Similar to `Value_Refine`, the complexity of `Value_Expand` depends on the specific approximation method that implements the value-summaries of the XSKETCH synopsis. The difference, however, is that `Value_Expand` introduces an additional dimension to the summarized values and this change typically excludes the possibility of an incremental update—in other words, the histogram at u must be reconstructed. Therefore, the space and time complexity of `Value_Expand` follow directly from the corresponding bounds of the underlying value-summarization technique.

B. COMPLEXITY OF XSKETCH EMBEDDING AND CONSTRUCTION ALGORITHMS

B.1 Time and Space Complexity of COMPUTEEMBEDDINGS

We use P_S for the number of root-to-node paths in the synopsis, f_Q for the fanout of query nodes, and N_S and N_Q for the number of nodes in the synopsis and the query respectively. The algorithm is called P_S *times in total* as it performs a depth-first traversal of the synopsis starting from the root; out of these, only N_S invocations actually compute bindings. The latter requires two steps: (a) identifying the candidate query nodes, and (b) iterating over each candidate and checking the existence of witness paths. The complexity is $O(N_Q)$ and $O(N_Q f_Q)$, respectively, as the worst case is for v to match all query nodes (this is the pathological case where the query only contains wild cards). Hence, the computation of bindings has complexity $O(N_S N_Q f_Q)$. The update of path information to ancestors happens on every invocation of the algorithm, and it requires accessing the traversal stack and checking the candidate sets of ancestors; this implies $O(N_Q N_S)$ iterations for the final loop (Steps 12–14). Each iteration performs two checks: whether $q_j \in bind[v]$ is a child of the current candidate q_i of the ancestor, and whether $TS[u..v]$ is compatible with $axis(q_i, q_j)$. The first check can be performed in constant time by maintaining $bind[v]$ as a hash table based on the parent node of each q_j; the second is a simple check, as $TS[u..v]$ is compatible if $axis(q_i, q_j) = //$ or if $axis(q_i, q_j) = /$ and v is a child of u. The total complexity of the algorithm is therefore $O(P_S N_Q N_S + N_S N_Q f_Q)$, which simplifies to $O(P_S N_Q N_S)$, since, typically, $f_Q \ll P_S$.

Regarding space complexity, the algorithm stores two pieces of information: the candidates and bindings of a node (sets $cand$ and $bind$, respectively), and the witness paths. The former requires $O(N_S N_Q)$ of total space. For $paths$, our observation is that the last loop can add at most one path in each iteration; this amounts to a total of $O(P_S N_Q N_S)$ paths of length $O(N_S)$ each. The total space complexity is therefore $O(P_S N_Q N_S^2)$.

B.2 Time and Space Complexity of BUILDXSKETCH

The total number of iterations for the BUILDXSKETCH algorithm depends on the target space budget and of course on the structural and value characteristics of the input data. Hence, we analyze the space and time complexity of performing a single iteration, focusing on the two main components that we discussed previously, namely, the generation of candidate refinements and the scoring of each candidate. In what follows, we use N_S to denote the number of nodes in the synopsis, $\texttt{ideg}(u)$ and $\texttt{odeg}(u)$ for the in- and out-degree, respectively, of node u, and $C_r(x, u)$ for the cost of performing refinement x on node u. (We have derived time and space bounds for these cost factors in Section 5.1.)

The generation of candidate operations requires obtaining a biased sample of synopsis nodes and computing refinement operations on the sample. The first part involves an initial pass over the synopsis nodes to compute their sample weights, and a second pass to actually obtain the sample; hence, the time and space complexity is $O(N_S)$. For the second part, we assume that each node u in the sample V has $\texttt{ideg}(u)$ incoming edges, $\texttt{odeg}(u)$ outgoing edges,

and carries a value-summary for which $|STN(u)| - |dep(u)| = n(u)$. We can therefore express the cost of this part as $\sum_{u \in V} \texttt{ideg}(u) \cdot C_r(\texttt{B\_Stabilize}, u) + \texttt{odeg}(u) \cdot C_r(\texttt{F\_Stabilize}, u) + C_r(\texttt{B\_Split}, u) + n(u) \cdot C_r(\texttt{Value\_Expand}, u) + C_r(\texttt{Value\_Refine}, u)$. (The space complexity can be expressed in a similar fashion.)

The scoring of a candidate refinement comprises again two steps, namely, gathering a sample $P(u)$ of path expressions around the targeted node u, and estimating the selectivity of each sampled path before and after the refinement. For the first step, the algorithm essentially samples a set of embeddings that include u and subsequently converts them to the corresponding path expressions. To sample a single embedding, BUILDXSKETCH starts with u in the main branch and gradually expands it by sampling either a parent from the first node of the main branch, or a child from any of the already included nodes. (These two options correspond to augmenting the main branch and the branching predicates, respectively.) Augmenting the embedding by one node is therefore equivalent to sampling one out of $N_Q\,\texttt{odeg} + \texttt{ideg}$ items, where N_Q is the number of nodes in the embedding, and odeg and ideg is the average out- and in-degree, respectively, of each synopsis node. The time complexity for generating an embedding of N_Q nodes is thus $O(N_Q^2(\texttt{odeg} + \texttt{ideg}))$, whereas the space complexity is $O(N_Q(\texttt{odeg} + \texttt{ideg}))$. For the second part, namely, approximating the selectivity of each path expression, the time complexity can be expressed as $O(|P(u)| \cdot C_e(N_Q))$, where N_Q denotes the average number of steps in each sampled path expression, and $C_e(N_Q)$ the complexity of estimating the selectivity of a path expression with N_Q steps (Section G). (The space complexity can be bounded in a similar fashion.)

C. XSKETCH SENSITIVITY TO CONSTRUCTION PARAMETERS

We discuss a set of experiments that examines the effectiveness of the construction algorithm when we vary two key parameters of the build process: V, the size of the node sample over which refinements are considered at each step of the greedy construction process, and P, the size of the localized path sample that is used to evaluate the score of a candidate synopsis. To isolate the effect of these parameters on the quality of the constructed XSKETCHes, we consider the simpler problem of structural summarization, that is, we ignore element values and thus remove any errors introduced by value-based summaries. We base our experiments on the graph-structured versions of the IMDB and XMark data sets.

We first consider the effect of the size of the node sample V on XSKETCH quality. Parameter V essentially controls the number of XSKETCH-node refinements that the construction algorithm considers at each step and essentially defines the portion of the construction search space that our forward-selection algorithm examines. We keep the size of the path sample P fixed to 50, and vary V as a percentage of the total number of nodes in the (current) synopsis. We experiment with two values for the node-sample size: 1% and 10%.

Figure C.1 shows the performance of XSKETCHes for branching path expressions as a function of the synopsis size for the different values of the node-sample size V. Note that, in all the graphs that we present, the estimation

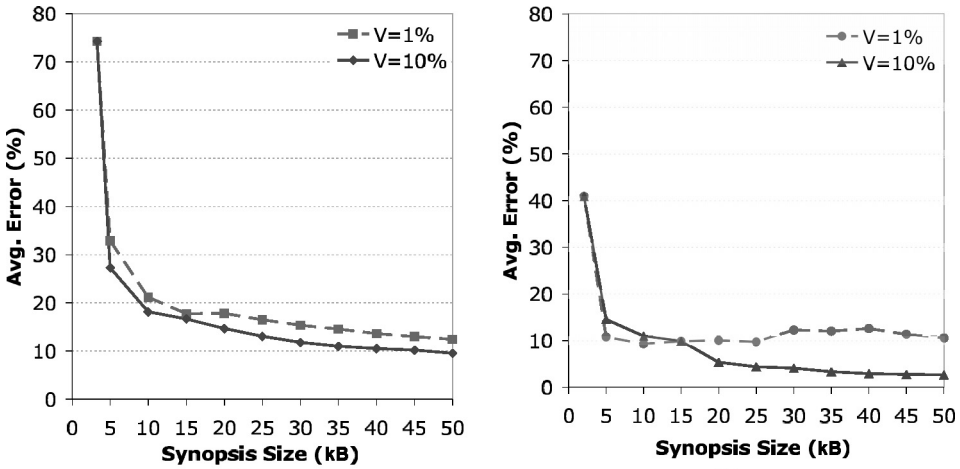

Fig. C.1. Performance of XSKETCHes for branching path expressions for varying V: (a) IMDB, (b) XMark.

error at the smallest summary size corresponds to the label-split graph synopsis. Our results clearly indicate that XSKETCHes constitute an efficient and accurate summarization method for graph-structured XML databases. Even with a small node-sample size of $V = 10\%$ for synopsis refinements and an allotted space of 25–35 kB, the estimation error drops to 10% and is substantially lower than the error of the coarsest summary, the label-split graph. This is most noticeable in the IMDB data set that is the most irregular of the two: the starting summary yields an average error of 70%, which rapidly drops below 20% after the first iterations of the construction algorithm. Furthermore, XSKETCHes achieve a low estimation error while using only a very small fraction of the space required by the corresponding "perfect" B/F-bisimilar graph for both data sets (Table 1). Overall, our XSKETCH synopses yield accurate selectivity estimates with low space overhead and can be efficiently constructed using only a small sample of the nodes for refinement.

The next set of experiments studies the effect of the size of the path sample P on the estimation accuracy of the constructed XSKETCH summary. Parameter P determines the number of sampled paths against which each refinement is evaluated; in essence, it represents the size of the training set of our forward-selection construction algorithm and can affect the quality of the generated synopsis. We keep the node-sample size V fixed to 10% of the total nodes in the summary and we experiment with two sizes for the path-sample size P: 10 and 50.

Figure C.2 depicts the XSKETCH estimation error for the IMDB and XMark data sets as a function of the synopsis size for the two different values of the localized path-sample size P. The results clearly show that, in all cases, our XSKETCH construction algorithm captures effectively the important statistical characteristics of the underlying path and branch distribution in the data. Once again, the XSKETCH estimation error is reduced significantly during the first few

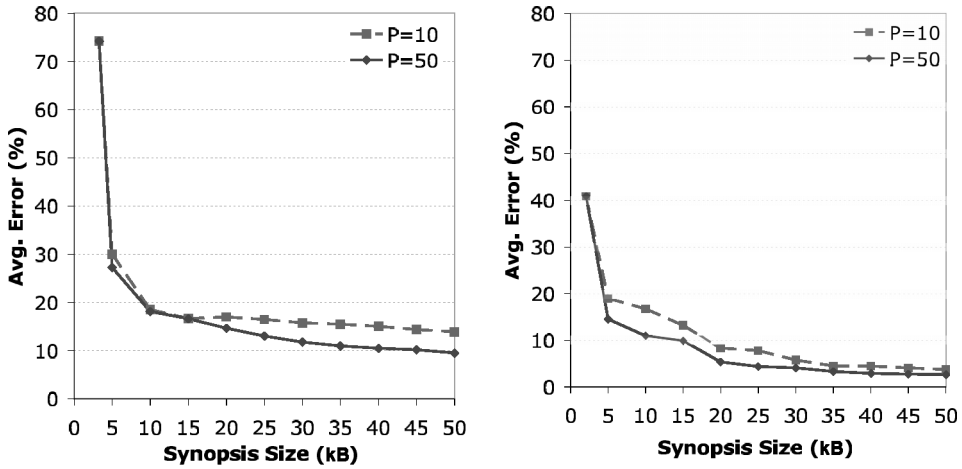

Fig. C.2. Performance of XSKETCHes for branching path expressions for varying P: (a) IMDB, (b) XMark.

iterations and improves gradually thereafter. We observe that the convergence of our algorithm is faster for a larger path sample size, especially in the case of the more irregular IMDB data set, where the small sample size of 10 exhibits a more "unstable" behavior with increased errors. In effect, the limited size of the training set prevents the algorithm from identifying the important correlations in the structure of the graph and leads to the application of less effective refinements at each step. Still, the overall conclusion is that reasonable path-sample sizes are adequate to construct accurate XSKETCH synopses, and an XSKETCH that occupies only 25–30 kB (0.5%–2% of the space required by the corresponding "perfect" B/F-bisimilar graph) is sufficient to enable low selectivity-estimation errors.

It is interesting, at this point, to examine the distribution of XSKETCH estimation error for our test workloads. Figure C.3 depicts the percentage of queries in each workload for different ranges of XSKETCH estimation error. (In both data sets, the XSKETCH error is computed for the 50 kB synopsis, using $P = 50$ and $V = 10\%$.) As shown, XSKETCHes enable a low estimation error for the vast majority of path queries; more concretely, 60% of IMDB paths and 87% of XMark paths have an XSKETCH error of less than 10%. The increased skew of the error distribution in Figure 3(b) corroborates the findings of the previous experiments, namely, that XMark has a more regular path structure and hence estimates are more accurate on the average; the complexity of the IMDB data set, on the other hand, causes higher estimation errors for a relatively larger fraction of the test path expressions.

D. PROOFS OF THEORETICAL RESULTS

D.1 Proof of Theorem G.1

Case (1). The proof follows by induction on the length of the label path n.

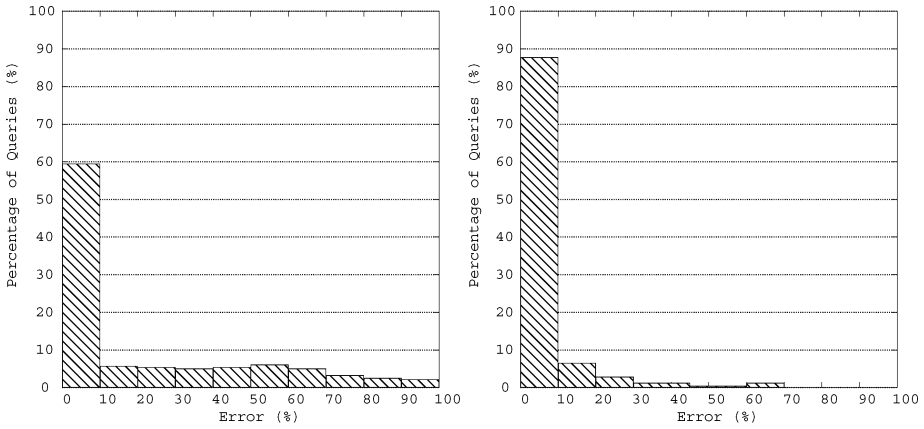

Fig. C.3. Distribution of XSKETCH estimation error for branching path expressions on graph-structured data: (a) IMDB, (b) XMark.

Base case ($1 \leq i \leq 2$). We know that v_2 is B-stable with respect to v_1 and therefore, by Definition 3.2, it holds that all elements in v_2 have an ancestor in v_1. Since all elements in v_1 have the same label, and the same holds for v_2, we conclude that all elements in v_2 are discovered by label path $\texttt{label}(v_1)/\texttt{label}(v_2)$.

Induction hypothesis. The theorem holds for $1 \leq i \leq n-1$.

Induction step. The induction hypothesis states that all elements in v_{n-1} are discovered by the label path $\texttt{label}(v_1)/\cdots/\texttt{label}(v_{n-1})$. We also know that v_n is B-stable with respect to v_{n-1}, which means that all elements in v_n have a parent in v_{n-1} and, therefore, all elements in v_n are discovered by the label path $\texttt{label}(v_1)/\cdots/\texttt{label}(v_{n-1})/\texttt{label}(v_n)$.

Case (2). The proof follows by induction on the length of the label path n.

Base case ($1 \leq i \leq 2$). We know that v_1 is F-stable wrt v_2 and therefore, by Definition 3.2, it holds that all elements in v_1 have at least one child in v_2. Since all elements in v_1 have the same label, and the same holds for v_2, we conclude that all elements in v_1 reach an element in v_2 by path $\texttt{label}(v_2)$.

Induction hypothesis. The theorem holds for $1 \leq i \leq n-1$.

Induction step. The induction hypothesis states that all elements in v_1 reach an element in v_{n-1} by the label path $\texttt{label}(v_1)/\cdots/\texttt{label}(v_{n-1})$. We also know that v_{n-1} is F-stable with respect to v_n, which means that all elements in v_{n-1} reach at least one element in v_n and, therefore, all elements in v_1 reach at least one element in v_n by the label path $\texttt{label}(v_1)/\cdots/\texttt{label}(v_{n-1})/\texttt{label}(v_n)$.

This completes the proof. □

D.2 Proof of Theorem H.1

We demonstrate the $\mathcal{NP}$-hardness of the XSKETCH construction problem with a reduction from the following two-dimensional clustering problem that aims to minimize the total normalized mean-squared error (MSE) for each of the

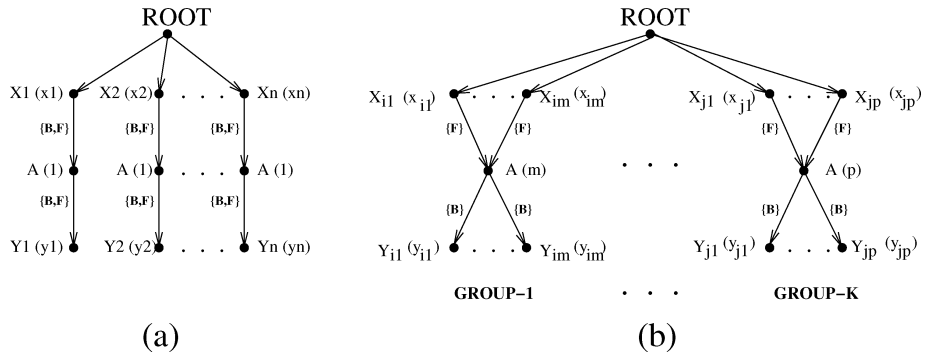

Fig. D.1. (a) Perfect synopsis (B/F-bisimilar graph). (b) XSKETCH synopsis after grouping A nodes.

resulting clusters (we term this problem 2*d-NMSE*). The details of the construction that proves the $\mathcal{NP}$-hardness of 2d-NMSE clustering are very similar to those given in earlier work on geometric clustering and covering (see, e.g., Fowler et al. [1981], Gonzalez [1985], Megiddo and Supowit [1984]).

Normalized MSE clustering on the plane (*2d-NMSE*). Given a set of points $S = \{(x_i, y_i) : i = 1, \ldots, n\}$ on the Euclidean plane, determine a partition of S into k clusters $C_1, \ldots, C_K$ such that the total normalized MSE from the cluster centroids

$$\sum_{j=1}^{K} \sum_{(x_i, y_i) \in C_j} \left[\left(\frac{x_i - \mathtt{xavg}(j)}{\mathtt{xavg}(j)} \right)^2 + \left(\frac{y_i - \mathtt{yavg}(j)}{\mathtt{yavg}(j)} \right)^2 \right]$$

is *minimized*, where $\mathtt{xavg}(j) = \sum_{(x,y) \in C_j} x/|C_j|$ and $\mathtt{yavg}(j) = \sum_{(x,y) \in C_j} y/|C_j|$.

We now demonstrate how, given a 2d-NMSE instance, we can build a (polynomially sized) instance of the optimal XSKETCH construction problem whose solution gives exactly the desired two-dimensional clustering that minimizes the normalized MSE objective. This obviously establishes the $\mathcal{NP}$-hardness of optimal XSKETCH construction.

Given a set of points on the plane $S = \{(x_i, y_i) : i = 1, \ldots, n\}$, consider the (exact) distribution of XML data paths given in the B/F-bisimilar graph depicted in Figure D.1(a). Here, we assume that all the X_i and Y_j labels in the (perfect) B/F-bisimilar synopsis are *distinct* with corresponding counts given by the corresponding point coordinates x_i and y_j. Note that the exact path-distribution information given by the B/F-bisimilar graph in Figure D.1(a) requires a total of $3n + 1$ nodes. Our XSKETCH construction instance seeks to find the XSKETCH synopsis with at most $2n + 1 + K$ nodes that minimizes the total squared error in the estimates for the queries in the workload $\mathcal{Q}$ defined as $\mathcal{Q} = \{//X_i/A : i = 1, \ldots, n\} \cup \{//A[Y_i] : i = 1, \ldots, n\}$. Note that, clearly, the exact answer to each of the $2n$ queries in $\mathcal{Q}$ is 1.

By our construction, it is obvious that the only possible way of reducing the number of nodes in the synopsis by $n - K$ ($= 3n + 1 - (2n + 1 + K)$) is to have an XSKETCH that groups the A-labeled nodes into K distinct XSKETCH nodes, as shown in Figure D.1(b). Of course, grouping two or more A-labeled nodes under

an XSKETCH node would ruin some of the stability properties of the original nodes, thus mandating the use of our estimation framework and assumptions to produce estimates for the queries in $\mathcal{Q}$. More specifically, consider the first A-node group in Figure D.1(b). Applying Backward- and Forward-Edge Uniformity, we estimate the number of elements reached by the queries $//X_{i_j}/A$ and $//A[Y_{i_j}]$ in $\mathcal{Q}$ as follows:

$$est(//X_{i_j}/A) = m \cdot \frac{x_{i_j}}{\sum_{l=1}^{m} x_{i_l}} = \frac{x_{i_j}}{\texttt{xavg}(i)}, \qquad est(//A[Y_{i_j}]) = m \cdot \frac{y_{i_j}}{\sum_{l=1}^{m} y_{i_l}} = \frac{y_{i_j}}{\texttt{yavg}(i)},$$

where $\texttt{xavg}(i) = \sum_{l=1}^{m} x_{i_l}/m$ and $\texttt{yavg}(i) = \sum_{l=1}^{m} y_{i_l}/m$. Thus, since the exact answer is 1, summing the squared errors over all the queries in $\mathcal{Q}$ we get the total squared error:

$$\sum_{j=1}^{K} \sum_{(x_i,y_i)\in \text{GROUP}-j} \left[\left(1 - \frac{x_i}{\texttt{xavg}(j)}\right)^2 + \left(1 - \frac{y_i}{\texttt{yavg}(j)}\right)^2 \right] =$$
$$\sum_{j=1}^{K} \sum_{(x_i,y_i)\in \text{GROUP}-j} \left[\left(\frac{x_i - \texttt{xavg}(j)}{\texttt{xavg}(j)}\right)^2 + \left(\frac{y_i - \texttt{yavg}(j)}{\texttt{yavg}(j)}\right)^2 \right].$$

The correspondence should now be obvious. Clearly, the XSKETCH synopsis that minimizes the total squared error in the above-described instance of the problem also uniquely determines the optimal solution to the corresponding input instance of the 2d-NMSE clustering problem (with the grouping of A-labeled nodes giving the desired clusters $C_1, \ldots, C_k$). This completes the proof. □