Selectivity Estimation of XPath for Cyclic Graphs

Yun Peng Byron Choi Jianliang Xu
Department of Computer Science
Hong Kong Baptist University
{ypeng, choi, xujl}@comp.hkbu.edu.hk

ABSTRACT

Recent interests on the Semantic Web, Web ontology and XML, among other topics, have sparked a renewed interest on graph-structured databases. Path queries are often used to retrieve graph-structured data (or simply graphs) from the database. A crucial and classical query optimization has been query selectivity estimation. However, the majority of existing works on selectivity estimation focuses on relational and tree data. In this paper, we investigate selectivity estimation on path queries on possibly cyclic graph data. To facilitate selectivity estimation on cyclic graphs, we propose a matrix representation of graphs extended from prime labeling, an index for reachability queries on directed acyclic graphs. With this representation, we exploit consecutive ones property (C1P) of matrix. As a consequence, a node is mapped to a point in a 2-dimensional plane and a query is mapped into multiple points. We adopt histograms for selectivity estimation. We perform an experimental evaluation on the space and time efficiency and accuracy of the proposed technique.

1. INTRODUCTION

Graph-structured databases have a wide range of emerging applications, e.g., the Semantic Web, eXtensible Markup Language (XML), biological databases and network topologies. To-date, there has already been voluminous real-world (possibly cyclic) graph-structured data []. To retrieve subgraphs from a large graph-structured database efficiently, various query optimization techniques have been proposed. Among others, selectivity estimation of queries has been a crucial query optimization technique in databases. In a nutshell, given a query, we want to determine the number of results of the query, without invoking potentially costly query evaluation. Selectivity estimation has been built in the query optimizer of all commercial relational databases. However, the majority of previous research on selectivity estimation, with few exceptions (see Section 2), focuses on relational and tree-structured data. In this paper, we propose an accurate and efficient selectivity estimation for graph-structured databases.

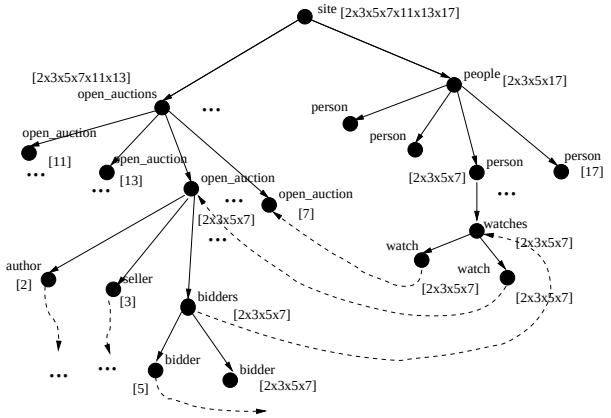

Figure 1: An example graph of auction information

Path queries have been a popular and classical tool for retrieving subgraphs from a graph-structured database. To facilitate our technical discussions, we consider a fragment of XPath query (see Section 3.2). Let us consider a simplified synthetic XMark data graph [14] shown in Figure 1, where . The graph encodes auction information, where people watch over auctions and bidders bid items. Consider an example query `//open_auction[//bidder//open_auction]`, which selects the auctions that are still open and have bidders who watch some `open_auction`s. Note that the XMark data is cyclic and the XPath query is recursive, where the query selects `open_auction`s that have `open_auction` descendants. In XMark with a scaling factor 0.1, there are 120, 708 and `bidder`s, `open_auction`s and `open_auction//bidder`s, respectively. To minimize intermediate result size, the query `//open_auction//bidder` is evaluated prior to `//bidder` from the root.

In recent years, there have been studies on selectivity estimation of XML data and XPath queries. In particular, Wu *et al.* [19] proposes to adopt histograms to selectivity estimation of XML queries. An advantage of this technique is that histograms are by far the most popular technique for query result estimation. The proposed technique relies on an interval representation of nodes, which presumes tree data. While a previous technique [2] extends the interval representation to support directed acyclic graphs, each node is potentially represented by multiple intervals. The storage requirement of multiple intervals per node can be prohibitvie.

In comparison, we represent a node with a single interval but a query with multiple intervals.

There has been a stream of work on summarizing cyclic graphs, namely XSketch [12] and TreeSketch [13]. The main idea of the work is to compute minimal bisimulation of a graph, which is used as a summary structure of the graph. However, the bisimiluation graph can still be large. Local bisimulation has been adopted to further reduce the size of the bisimulation graph. However, the estimation accuracy relies on a strong statistical assumption (uniform distribution) of data. In addition, bisimulation is not yet available in commercial databases.

In this paper, we propose selectivity estimation technique for cyclic graphs. The novelities of this paper are two. (i) We propose to adopt an index for descendant-ancestor queries that can support cyclic graphs. (ii) The second technique is to keep the index of the cyclic data simple while expanding a query for estimation. The intuitions of the two techniques can be described as follows.

We adopt prime number labeling scheme (or simply *prime labeling*) [17, 18] in this work. Prime labeling was originally proposed to determine descendant-ancestor relationship among nodes of trees. This scheme associates each node with an exclusive prime number and labels each node with the product of its parents' labels and the exclusive prime number of itself. Reachability between nodes is mapped to divisibility test of labels. In later work [17] this labeling scheme is extended to support directed acyclic graphs (DAGs). However, to the best of our knowledge, no further work has been presented to extend this idea to cyclic graphs, not mention to selectivity estimation for graph-structured data. In addition, the size of prime labeling is often smaller than that of transitive closure of a graph. We propose a binary matrix representation of prime labeling to further reduce the size of prime labeling.

Next, given a matrix representation of a graph, we transform it into a matrix with the consecutive ones property (C1P). Some columns are duplicated in the transformation. Subsequently, a node of a cyclic graph can be represented as an interval of column ID (*start*, *end*), as opposed to multiple intervals [2]. The nodes are summarized in a positional histogram. The complexity of querying is intuitively "pushed" into the mapping between duplicated column IDs. Given a query, we expand it into multiple query points using the mapping of duplicated column IDs.

The contributions of this paper are as follows.

- We propose to extend prime labeling of DAGs to cyclic graphs. We propose a binary matrix representation of prime labeling of cyclic graphs.
- We propose matrix transformation to map a node of a graph to an interval and a query to possibly multiple intervals. A two-dimensional histogram is adopted to summarize the graph. We propose an estimation procedure with the histogram.
- We performed a preliminary experimental study on the proposed techniques. It verifies that our techniques are efficient and accurate for real and synthetic data graphs.

The rest of this paper is organized as follows. Section 2 discusses related works. Section 3 presents the terminologies and notations of this work. Section 4 proposes the prime labeling and its matrix representation for cyclic graphs. Section 5 presents matrix transformation for interval representation of the graph. We present a preliminary experimental study in Section 6. We present a list of future works in Section 7.

2. RELATED WORK

There has been a host of recent works on selectivity estimation on path queries. The techniques can be roughly classified into two categories: graph-based approaches and relational-based approaches.

While graph-structured data model has its root at network data model, it was revisited in Tsimmis project, where Object Exchange Model (OEM) is proposed. DataGuide [11] is proposed to optimize query evaluation on graphs. Graphs are considered as non-deterministic automata and their DataGuide is the deterministic automata of the graph. DataGuide has been extended to support approximate query processing [8]. [6] proposes a straight-line grammar (STL), which is a special form of context-fre grammar, to summarize a data graph. To reduce the size of the grammar, [6] proposes to use a wildcard to simplify some non-terminals in a production.

Another graph-based approach [12,13] is derived from the notion of bisimulation of graphs. [12] proposes to use the minimum/minimal bisimulation as the summary structure of a data graph. To further reduce the size of bisimulation, a notion of local bisimulation [9] has been applied. To recover the path information from a local bisimilar graph, graph stability is exploited and uniform distribution of nodes are assumed.

Histograms in relational databases have been adapted to support selectivity estimation of queries on graphs. Chen *et al* [4] proposes an interval representation of echo node of a graph. The start and end position of the interval is used as the coordinates of a point in a 2-dimensional plane. Next, a positional histogram is used to summarize the point for estimation.

3. BASIC DEFINITIONS

In this section, we present the notations used.

3.1 Data Model

In this paper, we study directed node-labeled rooted data graphs, or simply *graphs* in the subsequent discussions. A graph can be denoted as $G = (V, E, r, \Sigma, \lambda, oid)$, where V is a set of nodes and E: $V \times V$ is a set of edges, $r \in V$ is a root node, Σ is a set of labels and λ:$V \rightarrow \Sigma$ is a function that returns the label of a node and *oid* is a function that returns a unique identifier of a node. For simplicity, we may denote a graph as (V,E) when other components are not relevant to the discussions.

3.2 Path Queries

Among the queries on graphs, XPath has been studied more extensively recently than others and it has been a indispensable part of eXtensible Markup Language (XML) – the *de facto* standard for electronic data exchange. Hence, we consider a fragment of XPath. The syntax is given in BNF below.

$$p ::= \epsilon \mid A \mid * \mid // \mid p/p \mid p[q],$$
$$q ::= p \mid q \wedge q \mid q \vee q,$$

where ϵ, A, $*$ and '/' denote the *self-axis*, a label (tag), a wildcard and the *child*-axis, and '//' stands for */descendant-or-self::node()/*, respectively; and q in $p[q]$ is called a *filter*, in which s is a constant (string value), and '$\wedge$' and '$\vee$' denote conjunction and disjunction, respectively. For //, we abbreviate $p_1/$ // as $p_1//$ and // $/p_2$ as $//p_2$. We use $r[|p|]$ to denote the evaluation of the query p from the node r.

Let R be the set of nodes of the evaluation, where $R = r[|p|]$. In this paper we want to determine $|R|$ efficiently and accurately.

4. REPRESENTATION OF CYCLIC GRAPHS

To efficiently evaluate the *descendant-or-self* axis on cyclic graphs, we propose our index in this section. We extend prime labeling [18] to our problem. The benefits are twofold: (i) Prime labeling can support cyclic graphs with minor modifications and the labels are simple; and (ii) Prime labeling of a graph is often smaller than the transitive closure of the graph, in practice.

4.1 The Original Prime Labeling

Prime labeling is originally proposed for indexing trees not cyclic graphs. The main idea of prime labeling is that each node is labeled with a product of prime numbers such that the ancestor-descendent relationship between nodes could be determined by using the division of the labels. A node n_1 is an ancestor of another node n_2 if and only if the label of n_2 is *divisible* by that of n_1. In [18], a unique prime number is assigned to each leaf node. The prime label of a node is the product of the prime label of its children. Obviously, such labeling works only on trees. To extend prime labeling to support DAGs, [17, 18] requires a unique prime number per node.

4.2 Prime Labeling for Cyclic Graphs

To support cyclic graphs, the extensions of the previous work on prime labeling ([18], [17]) are two. (i) Prime labeling needs to support possibly multiple strongly connected components (SCCs) in cyclic graphs. By definition, each node in a SCC can reach any other node in the SCC. Therefore, the nodes in a SCC can be associated with the same prime label. (ii) Previous work [17] on prime labeling uses excessive prime numbers (one prime number per node). We use fewer number of (unique) prime numbers needed for labeling and therefore reduce the overall size of prime labeling.

In particular, we require a new prime number for labeling a node in one of the following situations. (i) The node is a leaf node. (ii) The node has multiple parents.

With the above background, we now give the definition of prime labeling.

Let `get_next()` is a special function which returns a prime number that it has not returned before. Assume that a cyclic graph G has been preprocessed by Tarjan's algorithm [15], where each SCC is reduced to a supernode. Denote the reduced graph (DAG) to be $G'(V', E')$. Each node n is associated with a prime label ℓ as defined in Definition 4.1.

Definition 4.1: The *prime label* ℓ of a node n of the reduced graph $G'(V', E')$ can be defined as follows.

Input: A data graph G
Output: the matrix representation of a graph M

```
01 G' = tarjan(G)
02 for each n in G'.V in reverse topological order
03   if n is a leaf node                    /* Definition 4.1 */
04     n.ℓ⃗[get_next()] = 1
05   else
06     if |n.parent| ≥ 1
07       n.ℓ⃗[get_next()] = 1
08     for each c in n.children
09       n.ℓ⃗ = n.ℓ⃗ || c.ℓ⃗
```

Figure 2: Prime Labeling Construction `prime-construct`

1. If n is a leaf node, then $n.\ell =$ `get_next()`.
2. If n is a non-leaf node and n has multiple parents, then $n.\ell =$ `get_next()` $\times \prod_{c \in C} c.\ell$, where C is the set of child nodes of n.
3. Otherwise, $n.\ell = \prod_{c \in C} c.\ell$.

□

The prime label can be assigned to nodes of the reduced graph G' in a reverse-topological order, i.e., a bottom-up traversal. The pseudo-code of the construction (`prime-construct`) is shown in Figure 2. The time complexity of `prime-construct` is $O(|V'| + |E'|)$.

Assume we have a set of A-nodes and B-nodes, denoted as S_A and S_B, respectively. A naive way to determine the number of B-descendants in S_B of the nodes in S_A takes $O(|S_a| \times |S_b|)$. With prime labeling, this can be done by first computing the product of the prime labels of S_A, denoted as M_A and then check the divisibility between the product M_A and the prime label of each node in S_B. This requires $O(|S_a| + |S_b|)$.

4.3 Vector Representation of Prime Labeling

A known issue of prime labeling is that it often results in very large integers. Nowadays, there has been voluminous graph data, such as protein data, social network and XML. In response to these, we propose a vector representation of prime labeling and map division of integers to logic operators of vectors.

Definition 4.2: Suppose that the prime label ℓ of a node n of a graph G is $p_{i_1} \times p_{i_2} \times ... \times p_{i_m}$, where p_{i_j} is the i_j-th prime number. ℓ is then presented by a vector $\vec{\ell}$ where $\vec{\ell}[i_j] = 1$ if and only if p_{i_j} is a factor of *ell* and 0, otherwise. The dimension of the vector is the number of prime numbers used in labeling G. □

Then, a graph can be represented as a matrix. In this work, we always discuss binary vectors and matrices. For simplicity, we may omit the term "binary".

With this representation, division and multiplication of prime labels can be mapped into logical operations on the vector representation of the prime labels. In addition, the property of prime labeling that a set of

Definition 4.3: Given two nodes n_1 and n_2, $n_1.\ell$ is divisible by $n_2.\ell$ if and only if $\neg n_1.\vec{\ell} \wedge n_2.\vec{\ell} = \vec{0}$. □

Definition 4.3 can alternatively be understood that the vector $\neg n_1.\vec{\ell}$ and $n_2.\vec{\ell}$ are *orthogonal*, where the product of the two vectors is 0.

Definition 4.4: Given a set of nodes V and n_2, $\prod_{n\in V} n.\ell$ is divisible by $n_2.\ell$ if and only if $\neg(\bigwedge_{n\in V} n.\vec{\ell}) \wedge n_2.\vec{\ell} = \vec{0}$. □

Therefore, our problem is translated into the following.

Selectivity Estimation. Assume that m prime numbers have been used in labeling and a graph with n nodes. Given a vector representation of a query $\vec{v}$ and a $n\times m$ matrix M, we want to estimate the number of rows in M that are orthogonal to $\vec{v}$. □

Hence, the remaining task is to summarize the matrix for the selectivity estimation problem.

5. AN OVERVIEW OF OUR SOLUTION

In this section, we present an overview of our solution by exploiting the matrix representing proposed in Section 4.

5.1 Consecutive One Property

Given the matrix representation discussed Section 4, we perform a few transformations on the matrix for summarization. First, we convert a matrix into a matrix with a consecutive one property (C1P).

Definition 5.1: A matrix M has the *weak Consecutive Ones Property (C1P)* if its columns can be permuted such that in each row, the ones are adjacent. A matrix M has *strong C1P* if the ones of each row are adjacent. □

Unfortunately, the conversion of a matrix into a C1P matrix is intractable. Worst still, there is no polynomial time approximation algorithm for determining C1P submatrix of a given matrix. Here, we propose a linear time algorithm to convert a matrix into a C1P matrix. Obviously, the matrix returned is not minimum. The algorithm is presented in Figure 3, namely Algorithm `convex_bipartite`.

First, we convert a matrix M into a bipartite graph $G(U, V, E)$, where U and V are two *sequences* of nodes. We create a node in U, for each row of M. Similarly, we create a node in V for each column of M. In addition, we introduce $(u, v) \in E$ if and only if $M[u][v]$ is non-zero. A *convex* bipartite graph is a bipartite graph where the neighbors of each node in U are adjacent. An example of convex bipartite graph is shown in Figure 5. It is obvious that a matrix with C1P can be derived from a convex bipartite graph. However, to obtain a C1P matrix, we will need to introduce new nodes in U. Therefore, a main task of Algorithm `convex_bipartite` is to introduce few new nodes when generating a convex bipartite graph.

Second, we discuss the details of `convex_bipartite`. In Line 01, we simply order the nodes in V by the number of neighbors. In Lines 03 - 08, we expand the bipartite graph fully. Each node has an unique id (assigned by an incremental counter) In Lines 09 - 13, we merge the bipartite graph from left to right. In Lines 14 - 19, we merge the bipartite graph from right to left. The neighbors of a node in v can then represented by the id of its first and last neighbor nodes, which forms an interval of ids. The interval of nodes is that summarized by a two dimensional histogram. In addition, after we obtained a C1P matrix, a node u in U may have been copied multiple times. Hence, we need to record the ids that a node u in U represents. In particular,

```
Input: A bipartite graph G=(U, V, E)
Output: A convex bipartite graph G'=(U ∪ U', V, E')
/* sort v ∈ V by v.N.length from smallest to largest */
01 G=sort(G)
/* duplication */
02 count = 0
03 for i in [1 ... |V|]
04   for each u_j in v_i.N
05     construct a new node u' in U'
06     construct a new edge (v_i, u') in E'
07     set u'.cid = u_j.cid
08     set u'.id = count++
/* two passes merging */
/* merging from left to right */
09 for i in [1 ... |V| − 1]
10   for each u_j in v_i.N
11     p = find(u_j, v_{i+1}.N)
12     if p ≠ −1
13       mergeL2R(u_j, u_p, i)
/* merging from right to left */
14 for i in [|V| ... 2]
15   for each u_j in v_i.N
16     if u_j.N.length == 1
17       p' = find(u_j, v_{i−1}.N)
18       if p' ≠ −1 and mergable(u_j, u_p', i)
19         mergeR2L(u_j, u_p', i)
20 returnG'
```

Figure 3: Transformation to a convex bipartite graph `convex_bipartite`

we store this information in two relations f and f^{-1}, where $f(u)$ returns the ids of nodes that u represent and $f^{-1}(x)$ returns u if $x \in f(u)$.

A property of the the intervals is that the interval of a node's descendants must be contained in the interval of the node. In the two dimensional histogram representation, the descendant point must be on the bottom right direction of the point of the node. To determine the count of the descendants of a node, we simply count the number of points in the bottom right direction of the data point of the node. Two-dimensional histogram divides the two-dimensional space (plane) into grids. Each grid stores the number of data points contained in the grid.

In all, given a matrix M, `convex_bipartite` returns a C1P M' and a map that maps a node u to the ids that u represents f and the inverse of this map f^{-1}.

5.2 Estimation Framework

We assume that the input of an estimation step is (i) a descendant step $//A$ in a path query, where A is a label, (ii) the graph represented as discussed in Section 5.1 and (iii) a set of grids in the two dimensional histogram, which contain the set of input nodes of the query $//A$.

For each grid d, we determine the A-grid on the bottom right of the grid. For a gird that is completely in the bottom right region of d, all its nodes are A descendants of the node. For the grid that is partly in the bottom right region of d, we assume the number of A descendants is proportional to the area that is in the bottom right region – we assume data

```
/* search in a list for a particular element */
find(u,list)
01 for i in [1 ... list.length]
02    if list[i].cid == u.cid
03       return i
04 return -1

/* merge u_j and u_p from left to right */
mergeL2R(u_j, u_p, i)
01 remove u_j from v_i.N
02 add u_p to v_i.N
03 move u_p to left most position of v_{i+1}.N

/* merge u_j and u_{p'} from right to left */
mergeR2L(u_j, u_{p'}, i)
01 remove u_j from v_i.N
02 add u_{p'} to v_i.N
03 move u_{p'} to right most position of v_{i-1}.N

/* determine whether u_j and u_{p'} are mergable */
mergable(u_j, u_{p'}, i)
/* recall that after mergeL2R v_i.N is an ordered list */
01 for k in [p' + 1 ... v_{i-1}.N]
02    if u_k.N.length < u_{p'}.N.length
03       return false
04 return true
```

Figure 4: Algorithm `merge`

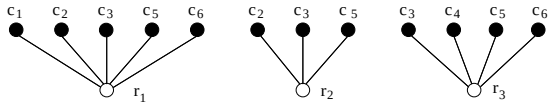

Figure 5: A simple example of convex bipartite graph

is evenly distributed in the grid. This gives us the count of $//A$ and a set of grids that contain some $//A$.

To estimate a path query $//A_1...//A_n$. Initially, we start with the grid containing the root node. We repeat the estimation described above for $//A_1$, ... and $//A_n$.

6. EXPERIMENTAL EVALUATION

In this section, we present a preliminary experimental evaluation that verifies the accuracy and efficiency of our proposed techniques.

We run our experiments on a machine with a Dual 4-core 2.93GHz with 30GB memory running Solaris OS. The implementation is written in `Java`. We use `Matrix` to refer to our technique.

We used `XMark` [14] to obtain a set of data graphs. The scaling factor was ranged from 0.1 - 0.5. We implement a query generator based on the description in Polyzotis *et al.* [12]. The average length of queries is 4. We generate 10,000 queries on the `XMark` data.

The estimation error of the queries on various `XMark` graphs is shown in Figure 6. The x-axis is the gird size of the two-dimensional histogram. The estimation error increases as the grid size increases. This is because more details of the data points have been summarized by larger grids. The estimation error drops for larger grids because we exploit an query evaluation (which produces exact answer) on the last

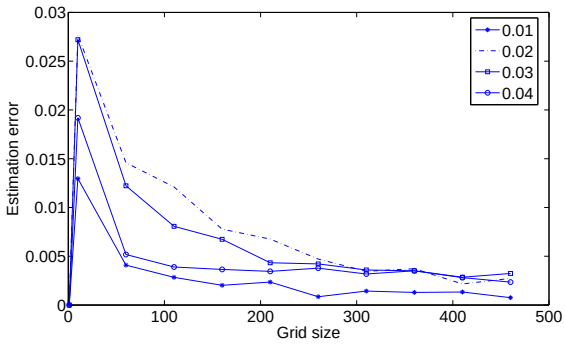

Figure 6: Estimation error

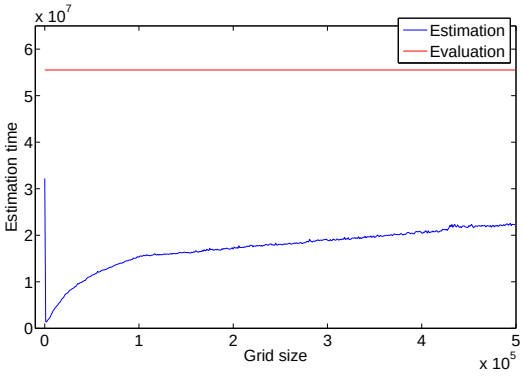

Figure 7: Estimation time

query step. The reason is that the last step often influential to estimation. To avoid introducing large error in estimation, in the last query step, we employ a query evaluation. For large grids, many grids are involved in the last query step for query evaluation. Hence, the estimation error is low.

We compared our result with a previous work [12]. The estimation errors under these settings are ranged from 8% to 10%. In comparison, our estimation never exceeds 3%.

Next, we compare the time for query estimation and query evaluation. The result is shown in Figure 7. The figure shows query estimation is always smaller than 33% of query evaluation time.

The next experiment shows a problem that we are currently working on. While the node is neatly summarized in two-dimension history. However, it assumes the existence of two mappings f and f^{-1}. Figure 8 shows the average size of $f(u)$ under different scaling factor of `XMark`. As shown, the average size of $f(u)$ is large, at least 100. And the average size increases linearly with s.f. Therefore, we are currently working on compression techniques for f and f^{-1}.

While f and f^{-1} are large, not the entire map is used in estimating the selectivity of a query. We use a simple indexing technique on f and f^{-1} to skip lookups that would always generate empty results. Figure 9 shows the number of lookups that are skipped. This shows that the skipping is linearly to the scaling factor of `XMark`.

Next we investigated the effect of optimizations that we used to reduce the estimations. First, we used query evaluation on the last query step. We ran an experiment with

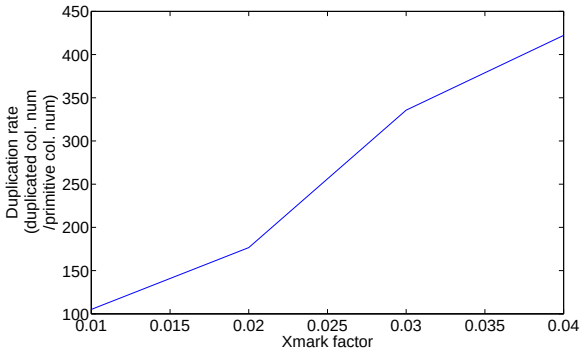

Figure 8: Number of copies of a nodes in f and f^{-1}

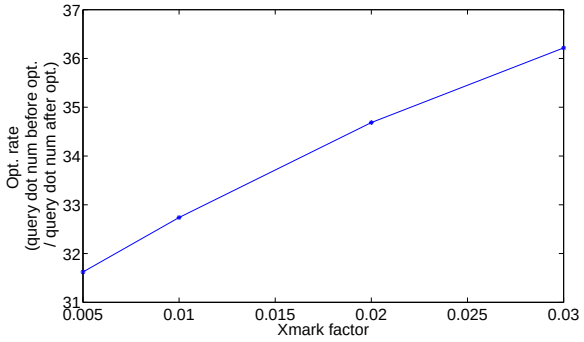

Figure 9: Performance of query dot generation opt.

and without this optimization to illustrate the effect. The result is presented in Figure 10. The figure shows that the optimization clearly reduces estimation errors.

Finally, we explore another optimization to reduce the estimation errors. The optimization is to maintain a bounding rectangle of the points in a grid. Hence, instead of selecting the entire grid or using uniform data distribution on data, we use the data points to limit the region where there are data points. The result is shown in Figure 11. It shows that this optimization clearly reduces estimation errors.

7. FUTURE WORKS

At the time this report is drafted, we are optimizing the proposed techniques. We are extending this report in a few directions. (i) We are compressing the representation of f and f^{-1}. The idea is the to detect frequent parttens in the maps and replace patterns with pattern ids. (ii) We are extending the experimental evaluation with larger datasets. The current dataset can support up to `XMark` up to a s.f. 0.5. (iii) We are deriving estimation upper and lower bounds of our method. (iv) Lastly, we are writing up the algorithms and optimizations that have been used in our experiments.

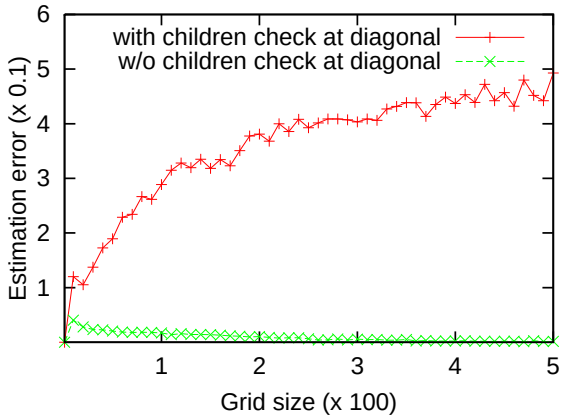

Figure 10: Performance of children check at diagonal grids at last step

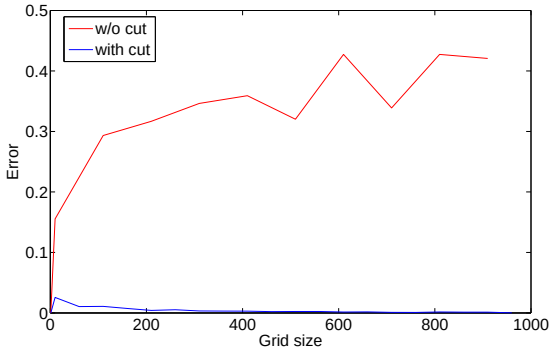

Figure 11: Performance of interval-cut operation

8. REFERENCES

[1] A. Aboulnaga, A. R. Alameldeen, and J. F. Naughton. Estimating the selectivity of xml path expressions for internet scale applications. In *VLDB*, pages 591–600, 2001.

[2] R. Agrawal, A. Borgida, and H. V. Jagadish. Efficient management of transitive relationships in large data and knowledge bases. In *SIGMOD*, pages 253–262, 1989.

[3] A. Chaudhary, D. Chen, X. Hu, M. Niemier, R. Ravichandran, and K. Whitton. Fabricatable interconnect and molecular qca circuits. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 26(11):1978–1991, Nov. 2007.

[4] Z. Chen, H. V. Jagadish, F. Korn, N. Koudas, S. Muthukrishnan, R. Ng, and D. Srivastava. Counting twig matches in a tree. In *ICDE*, pages 595–604, 2001.

[5] D. Fisher and S. Maneth. Structural selectivity estimation for xml documents. In *ICDE*, pages 626–635, 2007.

[6] D. K. Fisher and S. Maneth. Structural selectivity estimation for xml documents. In *ICDE*, pages 626–635, 2007.

[7] J. Freire, J. R. Haritsa, M. Ramanath, P. Roy, and J. Siméon. Statix: making xml count. In *SIGMOD*, pages 181–191, 2002.

[8] R. Goldman and J. Widom. Approximate dataguides. In *In Proceedings of the Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats*, volume 97, pages 436–445, 1999.

[9] R. Kaushik, P. Shenoy, P. Bohannon, and E. Gudes. Exploiting local similarity for indexing paths in graph-structured data. In *ICDE*, page 129, 2002.

[10] L. Lim, M. Wang, S. Padmanabhan, J. S. Vitter, and R. Parr. Xpathlearner: an on-line self-tuning markov

histogram for xml path selectivity estimation. In *VLDB*, pages 442–453, 2002.

[11] J. McHugh and J. Widom. Query optimization for xml. In *VLDB*, pages 315–326, 1999.

[12] N. Polyzotis and M. Garofalakis. Xsketch synopses for xml data graphs. *ACM Trans. Database Syst.*, 31(3):1014–1063, 2006.

[13] N. Polyzotis, M. Garofalakis, and Y. Ioannidis. Approximate xml query answers. In *SIGMOD*, pages 263–274, 2004.

[14] A. Schmidt, F. Waas, M. Kersten, M. J. Carey I. Manolescu, and R. Busse. Xmark: A benchmark for xml data management. In *VLDB*, pages 974–985, 2002.

[15] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[16] W. Wang, H. Jiang, H. Lu, and J. X. Yu. Bloom histogram: Path selectivity estimation for xml data with updates. In *VLDB*, pages 240–251, 2004.

[17] G. Wu, K. Zhang, C. Liu, and J.-Z. Li. Adapting prime number labeling scheme for directed acyclic graphs. In *DASFAA*, pages 787–796, 2006.

[18] X. Wu, M. L. Lee, and W. Hsu. A prime number labeling scheme for dynamic ordered xml trees. In *ICDE*, page 66, 2004.

[19] Y. Wu, J. M. Patel, and H. V. Jagadish. Using histograms to estimate answer sizes for xml queries. *Inf. Syst.*, 28(1-2):33–59, 2003.

[20] N. Zhang, M. Ozsu, A. Aboulnaga, and I. Ilyas. Xseed: Accurate and fast cardinality estimation for xpath queries. In *ICDE*, pages 168–197, 2006.

Acknowledgements. We are grateful to Neoklis Polyzotis for providing the implementation of [12, 13]. We thank Haibo Hu and Jintian Deng for their comments on the earlier drafts.