Estimating Result Size and Execution Times for Graph Queries

Silke Trißl and Ulf Leser

Humboldt-Universität zu Berlin, Institut für Informatik, Unter den Linden 6,
10099 Berlin, Germany
{trissl,leser}@informatik.hu-berlin.de

Abstract. In recent years several languages have been proposed to pose queries on graphs. These languages allow to state graph queries that contain multiple node and path variables. Nodes and paths of the graph are incrementally bound to these variables when evaluating the query. For an efficient execution the order of the bindings is important. To optimize this order we must be able to estimate the sizes of intermediate result sets and the time required to produce these. Therefore, in this paper we present estimation functions for reachability and path queries. We show that it is possible to estimate the sizes and times using easy to pre-compute key features of a graph, such as number of nodes and edges, number of nodes without outgoing edges, and the outdegree of the node with highest degree.

1 Introduction

Graphs occur in many areas of life. In our work, we specifically target graphs used in molecular biology. A human being has, according to current estimates, about 250,000 different proteins in his or her body. Each protein may interact with numerous other proteins or some of the hundreds of thousands organic and inorganic substances. Biologists have studied these complex interactions and their gained knowledge is stored as graphs in publicly available data sources [14]. The size of biological graphs ranges from a few hundreds to millions of nodes and edges. These biological graphs are often stored in relational database management systems (RDBMS), where they are usually represented by two relations, `NODE` and `EDGE`. Relation `NODE` represents all nodes of a graph, possibly with additional information on each node. Relation `EDGE` represents edges by giving the start and end node of the edge.

In [9] van Helden and colleagues identified several questions that are important for biologists working with biological graphs. We can divide those into two categories, *reachability queries*, where only nodes are of interest that are reachable from a given start node, and *path queries*, where paths including their intermediate nodes or lengths of paths are required. For instance, a user may be interested in fatty acids that lie on a path from Glucose to Acetyl-CoA with the condition that the path from the fatty acid to Acetyl-CoA is shorter than k.

Biologists can use specialized graph viewing tools to display these graphs [15]. But when manually navigating through images of pathways a biologist might not find a path although there exists one. Thus, tools are required that allow a user to pose a query. In [11], we introduced the Pathway Query Language (PQL), a language to formulate queries on biological graphs stored in an RDBMS. In similar fashion, He and Singh [8] and Dries et al. [6] also propose graph query languages. All languages have in common that in order to evaluate a graph query they have to bind nodes and paths from the graph to node and path variables of the query. Combinations of bindings are incrementally built to find all correct answers for a query. Clearly, the order in which these combinations are built is essential for high performance [7], especially when path predicates are involved, which are notoriously costly to compute [17]. In the previous example, should one first bind nodes to node variables, or first search fatty acids that are reachable on a path from glucose, or first find paths from a fatty acid to Acetyl-CoA that are shorter than k? To decide this, one must be able to estimate the time required to execute each step and the sizes of the result set.

In this work we present cardinality estimates to predict the size of the result set and cost functions to approximate the time required to compute reachability and path queries. These functions represent a key feature of our graph query optimizer [16]. Our estimates are merely based on properties of the graph, such as the number of nodes and edges, the number of nodes without outgoing edges, and the maximum outdegree of the node with highest degree. These properties are easy to compute as a single pass over relations `NODE` and `EDGE` is sufficient. We show in Section 4 that good estimation functions for result sizes and query times can be found using these predicates.

1.1 Related Work

Estimating the cardinality is a well established technique in RDBMS, usually performed by building histograms [10], as it is important for query optimization [7]. However, the methods established for RDBMS only target relational operations, such as selection, projection, join, and grouping. In graph queries, such methods can directly be used for estimating the number of bindings for node variables, but they cannot be applied for path variables.

There are approaches to estimate the sizes of result sets for reachability and path queries. Lipton and Naughton propose in [12] to estimate the size of the transitive closure by randomly select a set of start nodes and compute for each of these nodes the number of reachable nodes. This way, the result set for reachability queries may be estimated, but no algorithms or estimates for path queries are given.

In the area of queries on XML data some effort has been put into estimating the size of the result set for path queries. In [13] McHugh and Widom present a method based on pre-computed paths of length less than k. However, this approach has disadvantages. First, estimating works only well for paths of length k or less, and second, computing and storing paths requires time and a large amount of storage space. There exist several methods [1, 3, 18] to estimate the

size of the result set of a query on an XML document based on the number and positions of occurrences of elements in an XML document. The disadvantage of these methods is that they only work on XML documents with a tree structure. In contrast, our work addresses general graphs where the problem of reachability and path computation is considerably more complex.

We are not aware of any prior work that tried to estimate the size of the result of reachability or path queries merely based on general and easy to compute properties of the graph. To do this, we first give an introduction to graph queries in Section 2. Section 3 then presents algorithms for reachability and path queries. In Section 4 we develop functions to estimate cardinality and query time and evaluate our proposed functions. Section 5 concludes the paper.

2 Graphs and Graph Queries

We assume directed multi-graphs $G = (V, E)$ with $n = |V|$ and $m = |E|$. The number of incoming and outgoing edges of a node is its degree. Based on the distribution of the node degree in a graph we distinguish between random and scale-free graphs. Random graphs have a binomial degree distribution, while the distribution of scale-free graphs follows a power-law [2]. Graphs in biology are typically scale-free.

In this work we study reachability and path queries. To describe these queries, we first need to define the terms *path* and *path length*.

Definition 1 (Path and path length). *Let $G = (V, E)$ be a graph. A* path *p is a sequence of nodes $\langle v_0, v_1, v_2, \ldots, v_k \rangle$, $v_i \in V$ such that $(v_{i-1}, v_i) \in E$ for $i = 1, 2, \ldots, k$. The* length *of the path is the number of edges in the path.*

If there exists a path p from u to w we say w is *reachable* from u, written as $u \rightsquigarrow w$. A path is *cycle-free* if all nodes in p are distinct, otherwise p is said to contain a *cycle*. In this work we only consider cycle-free paths. Of course, if w is reachable from u, there might exist more than one path from u to w.

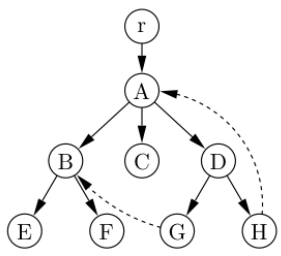
Definition 2 (Cardinality of reachability and path queries). *Let $G = (V, E)$ be a graph and $P_1, \ldots P_n$ be properties of the graph. Let $S \subseteq V$ be the set of start nodes and $T \subseteq V$ be the set of target nodes, with $u \in S$ and $w \in T$. The cardinality of a* reachability query*, $Card_{reach}(|S|, |T|, P_1, \ldots P_n)$, is the number of node pairs u, w for which $u \rightsquigarrow w$ holds. The cardinality of* path queries*, $Card_{path}(|S|, |T|, P_1, \ldots P_n)$ is the number of tuples to represent all paths between u and w.*

Cardinality estimates the size of intermediate results using prior knowledge of the data $(P_1, \ldots, P_n)$. However, in query optimization we are more interested in the time it takes to compute such an intermediate result, which, of course, depends on the size of the set, but also on the method used to compute it. Therefore, in the next chapter we introduce several algorithms for computing reachability and path queries. In Chapter 4 we compare measured execution times with those for our cardinality and time estimates.

3 Algorithms for Evaluation of Graph Queries

To answer graph queries we can for example traverse the graph at query time or use index structures, such as the transitive closure (TC)or GRIPP [17]. Graph traversal can be done using depth-first or breadth-first search [4]. The size of TC is in the order of $O(n^2)$, but can be queried in constant time to answer reachability queries. We presented in [17] GRIPP, an index structure whose size is in the order of $O(n + m)$ and can be queried in almost constant time for reachability queries. But GRIPP may also be used for path queries as we show in this section.

GRIPP is based on pre- and postorder labeling of the graph [5]. Each node in the graph receives as many pre- and postorder and depth values as it has incoming edges, but at least one value triple. Figure 1(a) shows the graph for indexing, while Figure 1(b) shows the resulting GRIPP index table with pre- and postorder and depth labelled instances of nodes. In this example, nodes A and B have two instances in GRIPP, one tree instance and one non-tree instance. This distinction is important for querying.



(a) Graph, G

node	pre	post	depth	type
r	0	21	0	tree
A	1	20	1	tree
B	2	7	2	tree
E	3	4	3	tree
F	5	6	3	tree
C	8	9	2	tree
D	10	19	2	tree
G	11	14	3	tree
B	12	13	4	non-tree
H	15	18	3	tree
A	16	17	4	non-tree

(b) GRIPP index table

Fig. 1. The graph to index and the resulting GRIPP index table. Nodes A and B have two instances in GRIPP, one tree instance and one non-tree instance

3.1 Reachability Queries

We can answer reachability queries for a given pair of nodes u, w using either a depth-first traversal, querying GRIPP, or the TC. For the depth-first traversal we start at u and query `EDGE` recursively until we find w. The number of recursive calls to answer reachability queries using depth-first traversal may range from 1 call for nodes without outgoing edges to n calls when we have to traverse all nodes of the graph. In contrast, querying the transitive closure only requires one lookup in the index table.

For GRIPP the situation is more complicated [17]. When querying GRIPP we use the tree instance of u, u^T to retrieve the reachable instance set of u, *RIS(u)*. *RIS(u)* contains all instances v' , for which $u^T_{pre} < v'_{pre} < u^T_{post}$ holds. We know that all nodes v, which have at least one instance in *RIS(u)* are reachable from

u. But in *RIS(u)* we may also find non-tree instances, for which we know that there exists some tree instance in the GRIPP index table. To find all reachable nodes of u we have to hop to the tree instance and use this instance and query the GRIPP index table again. We basically have to traverse the GRIPP index recursively until we find w or no further hop node can be used. To make the search more efficient we developed in [17] heuristics.

3.2 Path Queries

To answer path queries we have to define how to represent cycle-free paths. As we want to answer graph queries inside an RDBMS, we represent paths as relation `PATHS`, which is given in Figure 2(a). To give you an example how this relation is filled, consider Figure 2(b), which contains the tuples for all paths between D and B in the graph from Figure 1(a).

Paths
path_id
start
end
length
node_id
position

(a) `PATHS`

start	end	length	node_id	position
D	B	2	D	0
D	B	2	G	1
D	B	2	B	2
D	B	3	D	0
D	B	3	H	1
D	B	3	A	2
D	B	3	B	3

(b) Instance for paths from D to B

Fig. 2. The relation `PATHS` to represent all paths of a graph and the resulting relation for all paths between D and B in the graph from Figure 1(a)

To fill relation `PATHS` we can recursively traverse the graph or query the GRIPP index. We can not use TC, as it does not store intermediate nodes. When using the recursive query strategy we basically traverse the graph using depth-first search starting at u. When we find w during the search, we return the node pair and all intermediate nodes, which are stored during the search, together with their length information.

But we can also use GRIPP to answer path queries. To find paths with GRIPP we basically use the information already present in the index structure and use every hop node in a *RIS*. We use the example from Figure 2(b) to illustrate the search strategy. We start the search at the tree instance of D and add D at position 0 to list path_nodes, which stores all intermediate nodes. In *RIS(D)* we first find an instance of G, which we add to path_nodes at $0-2+3 = 1$. It is neither a non-tree instance nor B, therefore we use the next instance in *RIS(D)*, which is B, our target node. We return all intermediate nodes of that path and continue by removing B from path_nodes. Next, we find H, add it a position 1 and then we find the non-tree instance of A. We add A at position 2 to path_nodes and use A as hop node. We now explore all paths in *RIS(A)*, where we find another instance of B and return that path as well. If a length restriction for the paths is given, we only explore paths with up to the given length.

4 Estimating Cardinality and Execution Times

We estimate the size of the result set and the time required to compute it based on simple properties of the graph. In particular, we only use the number of nodes, the number of edges, the highest outdegree, and the number of nodes without any outgoing edge.

4.1 Evaluation Method and Data

We evaluate our estimation using synthetic random and scale-free graphs with varying sizes and densities. On each graph, we executed 1,000 reachability and path queries using randomly chosen node pairs and gathered the size of the result set and query time. We compare these measurements with our estimation functions. Table 1 shows properties of these graphs. Note that the maximum outdegree for random graphs only increases slightly with growing number of nodes, while it grows constantly for scale-free. In contrast, the number of nodes with no outgoing edges increases linearly with growing graph sizes in both types.

Table 1. Properties of random and scale-free graphs used for evaluation. Figures are averaged over five different graphs for each type and size

No. nodes	No. edges	Random graphs		Scale-free graphs	
		Max. degree	Zero degree	Max. degree	Zero degree
100	200	6.2	12.0	10.8	14.6
1,000	2,000	7.8	135.6	28.2	156.4
10,000	20,000	9.6	1,346.8	91.8	1,567.2
100,000	200,000	11.2	13,520.2	237.8	15,839.2
1,000,000	2,000,000	11.6	135,198.0	748.2	158,780.8
100	100	4.2	37.6	7.2	35.8
100	200	6.2	12.0	10.8	14.6
100	300	7.0	4.4	15.4	6.6
100	400	9.4	1.0	17.6	1.4
100	500	10.6	0.6	20.0	1.6

4.2 Reachability Queries

We estimate for a given set of start and end nodes, with $|S| = s$ and $|T| = t$, the size of the result set. Clearly, the larger both sets are, the more tuples are returned. But the density of the graph also influences the size of the result set. We can expect that with increasing average node degree the probability of $u \rightsquigarrow w$ will increase, until a saturation occurs. A saturation function is described by $(1 - e^x)$. In our case x is the fraction between additional edges $(m - n)$ and the number of nodes, n. Equation 1 shows the formula for cardinality estimation of reachability queries.

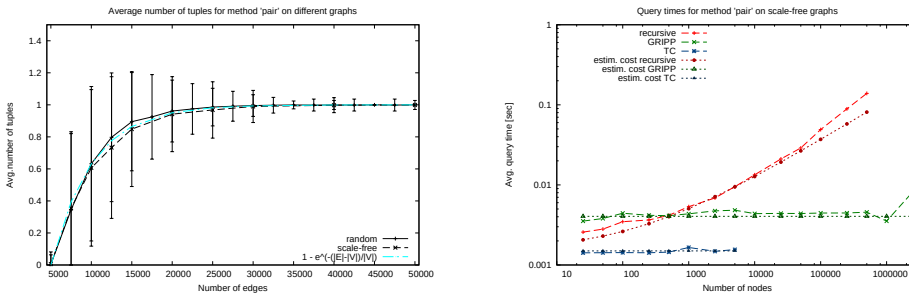
$$Card_{reach}(s, t, n, m) = s \cdot t \cdot (1 - e^{-\frac{m-n}{n}}) \tag{1}$$

The cost estimates must resemble the time required for each algorithm to answer reachability queries for a given pair of nodes. This means, we require three cost functions. Querying the transitive closure is done in constant time. The

same is true for GRIPP, as we require on average 2.7 recursive calls, regardless the size or shape of the graph. For the recursive query strategy the situation is more complicated. The query time is proportional to the number of calls. This number in turn depends on the number of nodes in the graph (calls $\sim \sqrt{n}$), the degree and maximum outdegree, x (calls $\sim 1/\ln((m+x)/n)$), and the number of nodes with outdegree 0, z (calls $\sim (n-z)/n$). Equation 2 gives the cost estimates for reachability queries. The constant factors c_1 (access cost) and c_2 (CPU cost), which depend on the system setup, may be estimated using test runs on the system.

$$\begin{aligned} Cost_{reach(TC)}(s,t) &= s \cdot t \cdot c_1 \\ Cost_{reach(GRIPP)}(s,t) &= s \cdot t \cdot 2.7 \cdot c_1 \\ Cost_{reach(recursive)}(s,t,n,m,x,z) &= s \cdot t \cdot (c_1 + (c_2 \cdot \sqrt{n} \cdot \frac{1}{\ln \frac{m+x}{n}} \cdot \frac{n-z}{n})) \end{aligned} \tag{2}$$

We evaluate our cardinality and cost functions experimentally. Figures 3(a) and 3(b) show that the given functions correspond well with the actually observed values for $c_1 = 0.0015$ and $c_2 = 0.00017$. Note, there is no distinction between random and scale-free graphs for the size of the result set.



(a) Cardinality Estimates (b) Cost Estimates

Fig. 3. The figures show the estimated and actual result sizes and query times for different graphs to answer reachability queries for a given pair of nodes

4.3 Path Queries

To estimate the size of the result set for path queries we consider the following. Assume, the graph has an average outdegree of $d = 2$, thus, for a single start node we find on average 2 nodes, i.e., 2 paths for path length 1, $4 + 2$ paths for length 2, $8 + 4 + 2$ paths for length 3, and so on. In general for length l we find $\sum_1^l d^l$ paths. This would be true for infinite binary trees, but not for general graphs. In general graphs three factors influence the number of paths, which are given in Equation 3. First, we may hit a node with high outdegree during

the search, which will open many new paths. Therefore, the number of outgoing edges, x, of the node with highest degree increases the average outdegree by $f(m/n, x) = (x - (m/n))/16$. Second, if we find a node without outgoing edges, we must stop. Thus, the number of nodes without outgoing edges z, reduces the number of paths by the factor of $(1 - pZero(n, z, l))$. The same is true, if we find a node for a second time on a path, which reduces the number of paths by the factor of $(1 - pDupl(n, x, l))$.

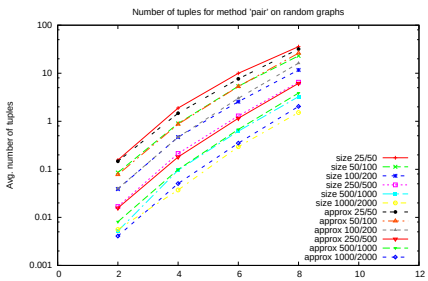
$$
\begin{aligned}
&Card_{paths}(s, t, n, m, x, z, l) = s \cdot t \cdot \\
&\sum_{i=1}^{l} \left((l+1) \cdot \left(\frac{m}{n} + f(\frac{m}{n}, x) \right)^i \cdot (1 - pZero(n, z, l)) \cdot (1 - pDupl(n, x, l)) \right) \\
&\text{with} \\
&pZero(n, z, l) = \sum_{i=1}^{l} \frac{l!}{i!(l-i)!} \cdot \left(\frac{z}{n} \right)^i \cdot \left(\frac{n-z}{n} \right)^{l-i} \\
&pDupl(n, x, l) = \sum_{i=2}^{l} \frac{l!}{i!(l-i)!} \cdot \left(\frac{x}{n} \right)^i \cdot \left(\frac{n-x}{n} \right)^{l-i}
\end{aligned}
\tag{3}
$$

For both correction factors we assume that each node has the same probability of being added to the path. Thus, we can use the binomial distribution to model the two correction factors. $pDupl(n, x, l)$ represents the probability that we find a node twice in a path of length l, while $pZero(n, z, l)$ represents that we find a node without outgoing edges on a path of length l.

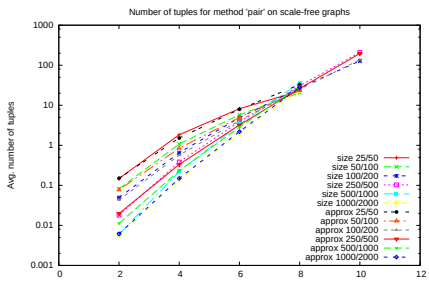
To answer path length queries we have two different implementations. We can either recursively traverse the graph or query GRIPP. For both implementations it is important how often we read relation `EDGE` or the GRIPP index. The number of reads is related to the number of paths found and thus, the correction factors applied for the cardinality estimates also apply here. Equation 4 shows the functions to estimate the cost to compute paths up to a certain length. GRIPP requires only $\sqrt{(m-n)/m}$ times the number of reads of the recursive strategy as some path information is already stored in the index.

$$
\begin{aligned}
Cost_{paths(GRIPP)}(s, t, n, m, x, z, l) &= s \cdot t \cdot c_1 \cdot No\_reads_{paths}(n, m, x, z, l) \\
&\quad \cdot \sqrt{(m-n)/m} \\
Cost_{paths(recursive)}(s, t, n, m, x, z, l) &= s \cdot t \cdot c_1 \cdot No\_reads_{paths}(n, m, x, z, l) \\
&\text{with} \\
No\_reads_{paths}(n, m, x, z, l) &= \sum_{i=0}^{l-1} \left(\frac{m}{n} + f(\frac{m}{n}, x) \right)^i \cdot (1 - pZero(n, z, i)) \\
&\quad \cdot (1 - pDupl(n, x, i+1))
\end{aligned}
\tag{4}
$$

As for reachability queries we experimentally evaluate our cardinality estimates and cost functions. Figure 4(a) and 4(b) show the experimentally and predicted result sizes. Note, there is a clear difference between the sizes of random and scale-free graphs. We capture this difference in Equation 3 by x, the outdegree of the node with highest degree, which basically means that in scale-free graphs more paths up to a certain length are found due to the nodes with high outdegree. This difference is also reflected in the query time (data not shown), where a query to find all paths up to length 7 for a given pair of nodes requires in random graphs with 1,000 nodes and 2,000 edges on average 90 ms and on scale-free graphs 800 ms using GRIPP. This difference is due to the size of the resulting relation `Paths` (1.5 tuples compared to 23.3 tuples on average). In comparison, a reachability query for a pair of nodes using GRIPP requires some 4 ms, regardless the size and shape of the graph. Clearly, this fact opens many possibilities for query optimization by query rewriting as it is much cheaper to execute a reachability query for the same set of input nodes than a path query.



(a) Cardinality on random graphs



(b) Cardinality on scale-free graphs

Fig. 4. The figures show the estimated and actual result sizes for different graphs to answer path queries for a given pair of nodes

5 Conclusion

In this paper we provide functions to estimate the sizes of result sets and the times to compute these for reachability and path queries. The presented functions require as parameters only easy to compute key features of a graph, such as the number of nodes and edges, the outdegree of the node with highest degree, and the number of nodes without outgoing edges. We verified our proposed functions experimentally on random and scale-free graphs.

These functions are a key requirement in query optimization. We are currently working to include the presented functions in our graph query optimizer [16]. This optimizer has rewrite rules that state, in which cases a path query may be rewritten to a reachability query. Based on the cardinality and cost estimates presented here, the optimizer can then decide if query rewriting is beneficial.

References

1. A. Aboulnaga, A. R. Alameldeen, and J. F. Naughton. Estimating the Selectivity of XML Path Expressions for Internet Scale Applications. In *Proceedings of VLDB*, pages 591–600, 2001. Morgan Kaufmann.
2. A.-L. Barabási and Z. N. Oltvai. Network biology: understanding the cell's functional organization. *Nature Reviews Genetics*, 5(2):101–113, Feb 2004.
3. J. Cheng, J. X. Yu, and B. Ding. Cost-Based Query Optimization for Multi Reachability Joins. In *DASFAA*, volume 4443 of *Lecture Notes in Computer Science*, pages 18–30, 2007. Springer.
4. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 2001.
5. P. Dietz and D. Sleator. Two algorithms for maintaining order in a list. In *Proceedings of STOC*, pages 365–372, 1987. ACM Press.
6. A. Dries, S. Nijssen, and L. De Raedt. A query language for analyzing networks. In *Proceedings of ACM SIGMOD*, pages 485–494, 2009. ACM Press.
7. L. M. Haas, J. C. Freytag, G. M. Lohman, and H. Pirahesh. Extensible Query Processing in Starburst. In *Proceedings of the ACM SIGMOD*, pages 377–388, 1989. ACM Press.
8. H. He, and A. K. Singh. Graphs-at-a-time: query language and access methods for graph databases. In *Proceedings of the ACM SIGMOD*, pages 405–418, 2008. ACM Press.
9. J van Helden, A Naim, R Mancuso*et. al.* Representing and analysing molecular and cellular function using the computer. *Journal of Biological Chemistry*, 381(9-10):921–935, 2000.
10. Y. E. Ioannidis. The History of Histograms (abridged). In *Proceedings of VLDB*, pages 19–30, 2003.
11. U. Leser. A query language for biological networks. *Bioinformatics*, 21 Suppl 2:ii33–ii39, Sep 2005.
12. R. J. Lipton and J. F. Naughton. Estimating the Size of Generalized Transitive Closures. In *Proceedings of VLDB*, pages 165–171, 1989. Morgan Kaufmann.
13. J. McHugh and J. Widom. Query Optimization for XML. In *Proceedings of VLDB*, pages 315–326, 1999. Morgan Kaufmann.
14. C. F Schaefer. Pathway databases. *Annals of the New York Academy of Sciences*, 1020:77–91, May 2004.
15. M. Suderman, and M. Hallett. Tools for visually exploring biological networks. *Bioinformatics*, 23: 2651–2659. 2007.
16. S. Trißl. Cost-based Optimization of Graph Queries. In IDAR, 2007.
17. S. Trißl and U. Leser. Fast and Practical Indexing and Querying of Very Large Graphs. In *Proceedings of ACM SIGMOD*, pages 63–79, 2007. ACM Press.
18. Y. Wu, Jignesh M. Patel, and H. V. Jagadish. Using histograms to estimate answer sizes for XML queries. *Inf. Syst.*, 28(1-2):33–59, 2003.