Query Size Estimation by Adaptive Sampling (Extended Abstract)

Richard J. Lipton*
Department of Computer Science
Princeton University

Jeffrey F. Naughton†
Department of Computer Sciences
University of Wisconsin

Abstract

We present an adaptive, random sampling algorithm for estimating the size of general queries. The algorithm can be used for any query Q over a database D such that 1) for some n, the answer to Q can be partitioned into n disjoint subsets $Q_1, Q_2, \ldots, Q_n$, and 2) for $1 \leq i \leq n$, the size of Q_i is bounded by some function $b(D, Q)$, and 3) there is some algorithm by which we can compute the size of Q_i, where i is chosen randomly. We consider the performance of the algorithm on three special cases of the algorithm: join queries, transitive closure queries, and general recursive Datalog queries.

1 Introduction

Query size estimation has a number of uses in database systems. Perhaps the most common use of query size estimation is query planning. Powerful, logic-based database query languages (such as LDL [TZ86] and NAIL! [MUVG86, MNS+87]) require sophisticated query planning if the are to have acceptable performance; unfortunately, the query size estimation techniques that have been developed for relational database systems [Chr83, Dem80, SAC+79, HOT88, Lyn88, OR86, PSC84, Row83] do not extend to provide size estimation procedures for these languages.

In this paper we present a general, adaptive sampling algorithm for query size estimation. Intuitively, the algorithm is applicable to any query for which answer to the query can be partitioned into disjoint subsets such that computing the size of a subset is cheaper than computing the original query. This includes Datalog queries that are not expressible by any relational algebraic query.

A problem with most sampling estimation algorithms is that they statically determine the number of samples needed based on the desired accuracy for the estimate. This works well when the time to take a sample is small and doesn't vary much, as is the case in sampling common relational queries. But if the cost of computing a sample varies widely and can be large, such algorithms have unpredictable running times and can be impractically slow.

Our sampling algorithm is adaptive in that the number of samples taken depends upon the size of the samples. If the cost of a sample is some function of its size (as is commonly the case) then our algorithm tends to keep the running time constant rather than the number of samples.

We focus on three applications of the algorithm: join queries, transitive closure queries, and general recursive Datalog queries.

We show that for the natural join $R \bowtie S$, the algorithm estimates the size of $R \bowtie S$ in time linear in the size of the smaller of the two relations. If the join is such that each tuple of R joins with at most a constant number of tuples in S (or vice-versa), the estimating algorithm is constant time.

For the transitive closure, we prove that the algorithm estimates the size of the transitive closure of a digraph with n nodes and m edges in time $O(n\sqrt{m})$ and $\Omega(n\sqrt{m})$. Furthermore, we show prove that the algorithm runs in linear time if the graph is of almost-uniform degree ("almost-uniform" is defined in Section 4.)

*Work supported by DARPA and ONR contracts N00014-85-C-0456 and N00014-85-K-0465, and by NSF Cooperative Agreement DCR-8420948

†Work supported by NSF grant IRI-8909795

Finally, we show that the algorithm can be used to estimate the size of arbitrary recursive Datalog queries, and that it can be expected to be especially efficient (with respect to the cost of computing the query) for recursions such that selections on the recursively defined relation can be evaluated significantly more efficiently than the entire relation can be computed.

The most closely related work is that which we presented in [LN89], since that algorithm can be regarded as a special case of the algorithm presented here. This paper presents the following new results.

- Section 2 presents an urn model significantly more general than that presented in [LN89]. This allows the estimating algorithm of this paper to be used on queries where the algorithm of [LN89] does not apply, such as arbitrary Datalog recursions and joins.
- The application of the estimation algorithm to join queries, and a proof that it runs in constant time for a useful subset of joins.
- The algorithm for the transitive closure presented in [LN89] gave estimates that were valid to a predetermined accuracy and confidence. The algorithm presented here takes the desired accuracy and confidence as a parameter.
- For the case of the transitive closure, we present tight asymptotic time bounds (upper and lower) for the algorithm.
- We prove that estimating the size of regular chain-rule queries over sparse relations is at least as hard as estimating the size of the transitive closure of arbitrary relations.

Section 2 gives a general statement of our estimating algorithm and discusses its performance. Section 3 analyses the algorithm as applied to join queries, while Section 4 considers the transitive closure and Section 5 considers arbitrary Datalog recursions.

2 General Size Estimation

In this section we present our query size estimation algorithm in its most general form. The theoretical underpinnings of the algorithm can be understood best in terms of the following urn model. This is a generalization of the urn model we presented in [LN89].

Consider an urn U in which there are n *balls*. Associated with ball i is a number $1 \leq a_i \leq b(n)$, for $1 \leq i \leq n$. In [LN89] we considered the special case $b(n) = n$; by considering a more general case here we are able to provide a significantly more powerful estimation procedure.

Algorithm 2.1 Repeatedly sample with replacement until S, the sum of the costs on the balls sampled, is greater than $\alpha b(n)$. Let the number of samples this takes be m. Estimate that $A = nS/m$.

The main utility of this urn model is the following theorem.

Theorem 2.1 *Let $\tilde{A}$ be the estimate of A in Algorithm 2.1. Then for $0 \leq p < 1$ and $d > 0$, if $\alpha = d(d+1)/(1-\sqrt{p})$, then Algorithm 2.1 estimates A to within A/d with probability p.*

Rather than prove the theorem directly, we first prove the following related theorem.

Theorem 2.2 *Suppose in Algorithm 2.1, the bound $\alpha b(n)$ is replaced with $\alpha V/E$, where V and E are the variance and mean of a random sample from the urn. Furthermore, let $\tilde{A}$ be the estimate of A in Algorithm 2.1. Then for $0 \leq p < 1$ and $d > 0$, if $\alpha = d(d+1)/(1-\sqrt{p})$, then Algorithm 2.1 estimates A to within A/d with probability p.*

We shall prove that $V/E \leq b(n)$, so Theorem 2.2 implies Theorem 2.1. The proof of Theorem 2.2 uses the following sequence of lemmas. We use the notation that the probability of an event x is $P[x]$.

The algorithm could fail to give a good estimate in two ways: by stopping too early (doing too little sampling) to produce a good estimate, or by doing enough sampling but still producing a bad estimate. The following lemma bounds the probability that the algorithm will stop too early.

Lemma 2.1 *Let $m = \beta V/E^2$ be a positive integer. Then*

$$P\left[\sum_{i=1}^{m} X_i \geq \frac{\alpha V}{E}\right] \leq \frac{\beta}{(\alpha-\beta)^2}$$

The next lemma bounds the probability of error, given enough sampling.

Lemma 2.2 *Let $m \geq \beta V/E^2$ and $d > 0$. Then*

$$P\left[\left|\frac{n}{m}\left(\sum_{i=1}^{m} \tilde{X}_i\right) - A\right| \geq A/d\right] \leq \frac{d^2}{\beta}$$

To prove Theorem 2.1, we need one more lemma:

Lemma 2.3 *Let $b(n)$ be the maximum size of any sample X. Then $b(n) \geq V/E$.*

When using this urn model to develop a query size estimation algorithm, a fundamental notion is that of *partitioning* the relation defined by the query.

Definition 2.1 A query $\mathcal{Q}$ over a database D is n-partitionable if

- The answer to $\mathcal{Q}$ can be partitioned into n disjoint subsets $\mathcal{Q}_i$, for $1 \leq i \leq n$;
- There is some function $b(D, \mathcal{Q})$ such that for $1 \leq i \leq n$, the size of $\mathcal{Q}_i$, written $|\mathcal{Q}_i|$, satisfies $0 \leq |\mathcal{Q}_i| \leq b(D, \mathcal{Q})$;
- It is possible to randomly select a subset $\mathcal{Q}_i$ and compute its size.

The function $b(D, \mathcal{Q})$ is purposely left vague; the intended meaning is that $b(D, \mathcal{Q})$ is some value that can be determined from properties of the query $\mathcal{Q}$ and the database D.

Example 2.1 Consider a query $\mathcal{Q}$ defining a k-ary relation r defined by an arbitrary Datalog program over some domain $\mathcal{D}$, and let $\mathcal{D}$ contain n distinct constants.

Then $\mathcal{Q}$ is n-partitionable as follows: choose one column of r, say column i, and partition the relation r into subsets based on the constants appearing in that column. That is, for each element $d \in \mathcal{D}$, there is a partition $\mathcal{Q}_d$ defined by the selection $\sigma_{i=d}(r)$. In general, some of these partitions will be empty.

The size of each partition is at most n^{k-1}, so here $b(D, \mathcal{Q}) = n^{k-1}$; the size of a partition $\mathcal{Q}_d$ can be computed by evaluating the selection query $\sigma_{i=d}(r)$. □

Example 2.2 Consider the natural join query $\mathcal{Q} = R \bowtie S$. If R contains n_R tuples, the query is n_R-partionable as follows: for each tuple r in R, the corresponding partition of $\mathcal{Q}$ is all tuples t in $\mathcal{Q}$ such that t was generated by joining r with some tuple of S.

If S has n_S tuples, then the size of each partition is at most n_S, so $b(D, \mathcal{Q}) = n_S$. We denote the partition of $\mathcal{Q}$ corresponding to the tuple r by $\mathcal{Q}_r$. If the join column of R is i, and the join column of S is j, and the constant appearing in column i of the tuple r is r_i, then the size of a partition $\mathcal{Q}_r$ can be computed by evaluating the selection $\sigma_{j=r_i}(S)$. □

Our estimation algorithm in its most general form appears below.

Algorithm 2.2 Suppose that we wish to estimate $|\mathcal{Q}|$ to within $|\mathcal{Q}|/d$ with probability p, and that $\mathcal{Q}$ is n-partitionable. Then set $\alpha = d(d+1)/(1-\sqrt{p})$ do the following.

1. set $S \leftarrow 0$;
2. repeat
   (a) randomly choose some $i \in \{1 \ldots n\}$;
   (b) compute $s = |\mathcal{Q}_i|$;
   (c) set $S \leftarrow S + \max(1, s)$;

   until $S \geq \alpha b(D, \mathcal{Q})$;
3. return nS/m.

Theorem 2.3 *Let $\mathcal{Q}$ be n-partitionable. Then Algorithm 2.2 estimates $|\mathcal{Q}|$ to within $(|\mathcal{Q}|+n)/d+n$ with probability p.*

The error term n in the statement of Theorem 2.3 arises because while the balls in the urn have associated numbers that are strictly greater than zero, the query partitions can have zero size. There are two logical ways to handle this difference; one (which we have chosen in this abstract) adds the error term n; the other (explored in the full paper) eliminates this error term at the cost of potentially increasing the running time if there are many zero-size samples.

The estimating algorithm presented in this abstract is most appropriate for estimating the size of queries that are "large" when compared to the partitioning constant n, as noted by the following corollary to Theorem 2.3.

Corollary 2.1 *Let $\mathcal{Q}$ be n-partitionable, and $|\mathcal{Q}|$ be $\Omega(n^{1+\epsilon})$. Then, asymptotically, Algorithm 2.2 estimates $|\mathcal{Q}|$ to within $|\mathcal{Q}|/d$ with probability p.*

Any computable query is n-partitionable with $n = 1$. This corresponds to computing all of $\mathcal{Q}$ some number of times, and will certainly produce a good estimate. The problem is one of efficiency.

The general problem is 1) to identify a means of non-trivially partitioning the query, then 2) to determine an algorithm for computing the sizes of the $\mathcal{Q}_i$, and 3) to prove that, given this partitioning, running Algorithm 2.2 is more efficient than computing the answer to $\mathcal{Q}$.

Suppose that a sample $|\mathcal{Q}_i|$ can be computed in time $g(|\mathcal{Q}_i|)$. Since $|\mathcal{Q}_i|$ is at most $b(D, \mathcal{Q})$, and the algorithm samples until its running sum reaches $\alpha b(D, \mathcal{Q})$, we immediately have a crude upper bound of $\alpha b(D, \mathcal{Q}) g(b(D, \mathcal{Q}))$ for the running time. In many cases, a much better bound applies, as formalized by the following theorem.

Theorem 2.4 *Suppose that a sample $\mathcal{Q}_i$ can be computed in time $g(\max(|\mathcal{Q}_i|, 1))$, and that for $x, y > 0$, $g(x) + g(y) = O(g(x+y))$. Then Algorithm 2.2 runs in time $O(g(b(D, \mathcal{Q})))$.*

Note that most functions g of interest in evaluating database queries — n, n^2, $n \log n$ — all satisfy $g(x) + g(y) = O(g(x+y))$ if $x > 0$ and $y > 0$.

In Sections 3, 4, and 5, we give examples of queries such that the size of the query can be estimated far more efficiently than it can be computed.

3 Join Queries

We begin by considering estimating the size of a binary natural join, say $R \bowtie S$. Let n_R be the number of tuples in R, and n_S be the number of tuples in S. As noted in Example 2.2 the query can be partitioned as follows. Each answer tuple must have been generated by the joining of a pair of tuples, one from R and one from S. For each tuple $r \in R$, we form the partition $\mathcal{Q}_r$ by selecting out all tuples in $R \bowtie S$ such that the tuple was generated by r from R and some tuple from S.

The size of each partition is trivially bounded by n_S; if we assume an appropriate index on S, then the size of a partition $\mathcal{Q}_r$ can be computed in $|\mathcal{Q}_r|$. This gives a bound on the running time of $O(n_S)$.

This bound is significantly better than the bound on computing the join itself, which is $O(n_R n_S)$. However, for an important subclass of joins, we can do much better. The bound given above on the size of the partitions of $\mathcal{Q}$ is a worst-case bound, which assumes that some tuple in R joins with every tuple of S. In many cases we will empirical or external domain knowledge that there is some constant k such that a tuple in R joins with at most k tuples of S.

Example 3.1 For example, consider a database with relations $Courses(CourseNum, Name)$ and $Students(Student, CourseNum)$. Suppose the join in question is $Students \bowtie Courses$. Then if we know that students don't take more than 10 courses, the partitions of the query will be of size at most 10. □

Using $b(D, \mathcal{Q}) = k$, we get that the algorithm must sample until $S \geq \alpha k$. This is a constant, independent of the size of the relations being joined.

The preceding discussion is formalized by the following theorems.

Theorem 3.1 *Let $\mathcal{Q}$ be the join query $R_1 \bowtie R_2 \bowtie \ldots \bowtie R_m$, and for $1 \leq i \leq m$, let relation R_i contain n_i tuples. Then by partitioning $\mathcal{Q}$ on the tuples of R_1, Algorithm 2.2 runs in time $O(\Pi_{2 \leq i \leq n} n_i)$.*

Theorem 3.2 *Let $\mathcal{Q}$ be the join query $R_1 \bowtie R_2 \bowtie \ldots \bowtie R_m$. Furthermore, suppose that for $1 \leq i \leq m-1$, a tuple in relation R_i joins with at most k_i tuples of relation R_{i+1}. Then by partitioning $\mathcal{Q}$ on the tuples of R_1, Algorithm 2.2 runs in time $O(\Pi_{2 \leq i \leq n} k_i)$, that is, in constant time.*

4 Transitive Closure Queries

In this section we show that Algorithm 2.2 can be specialized to estimate the size of the transitive closure of a directed graph. A variant of the resulting algorithm for the transitive closure was first presented in [LN89]. In that variant, a particular value for α was "hardwired" into the code, which meant that the accuracy and confidence intervals were fixed. In [LN89] it was also noted that for bounded degree graphs, the algorithm runs in linear time. In this section we extend that analysis to provide tight asymptotic upper and lower bounds on the running time, and to extend the class of graphs over which the algorithm is known to be linear time.

Consider the problem of estimating the size of the transitive closure of a graph G, which we will denote by G^*. The transitive closure of G can be partitioned into n disjoint subsets, where n is the number of nodes in the graph, by partitioning on the node that appears in the first column of the tuples of G.

We will let $\mathcal{Q}_i$ be the partition determined by node v_i. Using this partition, we have that $b(D, \mathcal{Q}) = n$, and that $|\mathcal{Q}_i|$ is just the size of the connected component of G rooted at v_i. Hence we use the notation $|G_v|$ to represent $|\mathcal{Q}_i|$. Then the following specialization of Algorithm 2.2 will estimate the size of the closure.

Algorithm 4.1 Let $\mathcal{Q}$ be the transitive closure of a graph G, let V be the set of all vertices in G, let n be the number of vertices in G. Furthermore, suppose that we wish to estimate $|\mathcal{Q}|$ within $(|\mathcal{Q}| + n)/d + n$ with confidence p. Then choose $\alpha = d(d+1)/(1-\sqrt{p})$ and execute the following.

1. set $S \leftarrow 0$;
2. repeat
   (a) randomly choose some $v \in \mathcal{G}$;
   (b) compute $s = |G_v|$;
   (c) set $S \leftarrow S + \max(1, s)$;

   until $S \geq \alpha n$;
3. return nS/m.

Note that by Corollary 2.1, if $|G^*|$ is $\Omega(n^{1+\epsilon})$, then the asymptotic accuracy of the estimate is $|G^*|/d$.

To discuss upper and lower bounds on Algorithm 4.1, we need to specify the algorithm used to compute $|G_v|$. Assuming that $|G_v|$ is computed by the standard depth-first search traversal of the connected component rooted at v, we have the following theorems.

Theorem 4.1 *Over the class of all directed graphs with n nodes and m edges, where $n \leq m \leq n^2$, for any $\epsilon > 0$, Algorithm 4.1 runs in time $\Omega(n\sqrt{m})$ with probability $1 - \epsilon$.*

The proof of Theorem 4.1 is constructive — that is, for any n and m, it specifies a graph such that the expected running time of Algorithm 4.1 is $\Omega(n\sqrt{m})$.

We now turn to an upper bound on the running time.

Theorem 4.2 *Over the class of all directed graphs with n nodes and m edges, where $n \leq m \leq n^2$, Algorithm 4.1 runs in time $O(n\sqrt{m})$.*

The graphs used in the lower bound argument of Theorem 4.1 have nodes of widely varying degrees. We say that a graph G is *almost-uniform* if there exists a function $d(x)$ such that if G has n nodes, for any vertex v appearing in G, the out-degree of v is at least $d(n)$ and at most $kd(n)$. For almost-uniform graphs, the algorithm is provably much faster.

Theorem 4.3 *Let k be fixed, and let G be almost-uniform and have m edges. Then Algorithm 4.1 is $O(m)$ on G.*

5 General Recursive Datalog Queries

As noted in Example 2.1, any Datalog query can be estimated by partitioning the relation specified by the query on the constants appearing in some column. This means that our estimation algorithm can be used to estimate the size of general recursive queries.

Discussing the performance of our algorithm on general recursive queries is difficult, since currently there are no techniques for giving tight bounds on the runtime for evaluation of general recursive queries. In general, using our estimation algorithm means that a query defining a relation R is estimated by some number of selection queries on R.

For some important classes of recursive queries, if n is the size of the database being queried, one can prove that a selection query on the recursively defined relation R can be evaluated with an asymptotic running time that is a factor of n faster than the asymptotic running time for evaluation all of R (see [BKBR87, NRSU89] for examples of such classes.) For any query Q such that the selection queries defining the partitions Q_i falls into such a class, our estimation algorithm runs asymptotically a factor of n faster than the query can be computed.

One particularly interesting subset of recursions is the chain-rule recursions. Chain-rule recursions have especially intuitive interpretations because of their similarity to context-free grammars.

A recursion R is a *chain-rule* recursion if

- Every predicate appearing in R is binary, and
- As the body of a rule r is read from left-to-right,
  - The first variable in the body of the rule appears in the first column of the predicate instance in the head of the rule,
  - The last variable in the body of the rule appears in the last column of the predicate instance in the head of the rule,
  - For all variables other than the first in the first predicate of the rule or the last in the last predicate of the rule, if the variable appears in column one of one predicate instance, it appears in column two of the instance to its left, and vice-versa.

Since all predicates in chain-rule queries are binary, in addition to being interpreted as a grammar, chain-rule queries have a natural interpretation as specifying path relations in digraphs.

Example 5.1 The program

```
t(X,Y) :- a(X,W),t(W,Z),c(Z,Y).
t(X,Y) :- b(X,Y).
```

is a chain-rule program. The program

```
t(X,Y) :- a(X,W),t(Z,W),c(Z,Y).
t(X,Y) :- b(X,Y).
```

is not a chain-rule program, since the variables in the instance of `t` in the recursive rule do not satisfy the chain-rule ordering. □

To interpret a chain-rule program as a grammar, ignore the variables in the rule and replace the implication :- by the symbol $\rightarrow$. Let the formal language so defined be L. To interpret this program as a graph problem,

- The constants in the relations represent nodes in the graph, and

- If there is a tuple $r(a, b)$ in some relation r in the chain query, then there is an edge from a to b labeled r in the graph, and
- The query itself specifies vertices (v_1, v_2) in G such that the concatenation of the labels of some directed path from v_1 to v_2 forms a word of L.

Because of this interpretation, we will use the following terminology.

Let L be the relation defined by the grammar corresponding to the chain-rule query, and let G be the graph corresponding to the database being queried. Let $R(L, G)$ be the set of all pairs (v_1, v_2) such that there is a path in G such that the concatenation of the edges in the path form a word of L. Let $R_v(L, G)$ be all pairs of the path relation $R(L, G)$ of the form (v, v') for some vertex v'. We denote the size of the path relation by $|R(L, G)|$ and the size of $R_v(L, G)$ by $|R_v(L, G)|$. Then the following specialization of Algorithm 2.2 is an estimation algorithm for path relations.

Algorithm 5.1 Let G be a directed, labeled graph with n nodes, and let L be some language. Suppose we wish to estimate $|R(L, G)|$ to within $(|R(L, G)| + n)/d + n$ with probability p. Then set $\alpha = 2d(1 + d)/(1 - \sqrt{p})$, and do the following.

1. Set $x \leftarrow 0$.
2. Repeatedly choose a random vertex v from G, and set $x \leftarrow x + \max(|R_v(L, G)|, 1)$, until $x \geq \alpha n$.
3. Let m be the number of vertices chosen in Step 2. Estimate $|R(L, G)| = nx/m$.

Perhaps the most natural generalization of the transitive closure estimation problem is the estimation of the size of path relations where the language in question is regular. We now consider the running time for the estimation algorithm on such queries.

Section 4 showed that for almost-uniform graphs the size of the transitive closure can be estimated in linear time using Algorithm 4.1. Whether there is such a linear time algorithm for estimating the size of path relations for regular languages over almost-uniform graphs is to our knowledge an open question. However, the following theorem shows that such a linear time estimating algorithm would imply a linear time estimating algorithm for the size of the transitive closure over arbitrary graphs.

Theorem 5.1 *Suppose that there exists an linear time estimating algorithm for regular path relations over almost-uniform graphs G. Then there exists a linear estimating algorithm for the transitive closure over arbitrary digraphs.*

The proof is by a reduction that, for an arbitrary digraph G, constructs a language L and a labeled digraph G' of almost-uniform degree such that the size of the path relation of L over G' is the same as the transitive closure of G.

Theorem 5.1 implies that estimating the size of a regular path relation over almost-uniform graphs is at least as hard as estimating the size of the transitive closure over arbitrary digraphs.

6 Conclusion

We have presented a general framework for estimating the size of queries. The task of applying the framework to provide an estimating algorithm for a given query requires a way of partitioning the answer to the query into disjoint sets, and an algorithm for computing the size of these subsets. The running time of the algorithm depends upon the relationship between the size of the samples and the cost of computing the samples.

A number of interesting questions remain. The general problem is to identify more classes of queries for which there are partitioning strategies and algorithms for computing the size of the resulting partitions such that the query size can be estimated far more quickly than it can be computed. Some more specific questions follow.

While the time bounds for our algorithm on the transitive closure are tight, there may be some other estimating algorithm that achieves the same time bounds with a lower asymptotic running time. The challenge is to find such an algorithm, or to prove that no such algorithm exists.

Another challenge is to identify the classes of Datalog queries such that the estimating algorithm runs provably much faster than the best evaluation algorithm for the query. One such class is the subset of linear recursions for which "factoring" yields a factor of n speedup, as described in [NRSU89].

References

[BKBR87] Catriel Beeri, Paris Kanellakis, Francois Bancilhon, and Raghu Ramakrishnan. Bounds on the propagation of selection into logic programs. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 214–226, San Diego, California, March 1987.

[Chr83] Stavros Christodoulakis. Estimating block transfers and join sizes. In *Pro-*

ceedings of the ACM SIGMOD International Conference on Management of Data, pages 40–54, San Jose, California, May 1983.

[Dem80] Robert Demolombe. Estimation of the number of tuples satisfying a query expressed in predicate calculus language. In *Proceedings of the Sixth International Conference on Very Large Databases*, pages 55–63, Montreal, Canada, 1980.

[HOT88] Wen-Chi Hou, Gultekin Ozsoyoglu, and Baldeao K. Taneja. Statistical estimators for relational algebra expressions. In *Proceedings of the Seventh ACM Symposium on Principles of Database Systems*, pages 276–287, Austin, Texas, March 1988.

[LN89] Richard J. Lipton and Jeffrey F. Naughton. Estimating the size of generalized transitive closures. In *Proceedings of the Fifteenth International Conference on Very Large Databases*, pages 165–172, Amsterdam, The Netherlands, August 1989.

[Lyn88] Clifford A. Lynch. Selectivity estimation and query optimization in large databases with highly skewed distributions of column values. In *Proceedings of the Fourteenth International Conference on Very Large Databases*, pages 240–251, Los Angeles, California, August 1988.

[MNS+87] Katherine Morris, Jeffrey F. Naughton, Yatin Saraiya, Jeffrey D. Ullman, and Allen Van Gelder. YAWN! (Yet Another Window on NAIL!). *Database Engineering*, December 1987.

[MUVG86] Katherine Morris, Jeffrey D. Ullman, and Allen Van Gelder. Design overview of the NAIL! system. In *Proceedings of the Third International Conference on Logic Programming*, 1986.

[NRSU89] Jeffrey F. Naughton, Raghu Ramakrishnan, Yehoshua Sagiv, and Jeffrey D. Ullman. Argument reduction through factoring. In *Proceedings of the Fifteenth International Conference on Very Large Databases*, pages 173–182, Amsterdam, The Netherlands, August 1989.

[OR86] Frank Olken and Doron Rotem. Simple random sampling for relational databases. In *Proceedings of the Twelfth International Conference on Very Large Databases*, pages 160–169, Kyoto, Japan, August 1986.

[PSC84] Gregory Piatetsky-Shapiro and Charles Connell. Accurate estimation of the number of tuples satisfying a condition. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 256–276, Boston, Massachusets, June 1984.

[Row83] Neil C. Rowe. Top-down statistical estimation on a database. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 135–144, San Jose, California, May 1983.

[SAC+79] P. Griffiths Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Proceedings of the ACM SIGMOD International Symposium on Management of Data*, pages 23–34, 1979.

[TZ86] Shalom Tsur and Carlo Zaniolo. LDL: A logic-based data-language. In *Proceedings of the Twelfth International Conference on Very Large Data Bases*, pages 33–41, Kyoto, Japan, August 1986.