Estimating the Size of Generalized Transitive Closures*

Richard J. Lipton and Jeffrey F. Naughton
Computer Science Department
Princeton University

Abstract

We present a framework for the estimation of the size of binary recursively defined relations. We show how the framework can be used to provide estimating algorithms for the size of the transitive closure and generalizations of the transitive closure, and also show that for bounded degree relations, the algorithm runs in linear time. Such estimating algorithms are essential if database systems that support recursive relations or fixpoints are to be able to optimize queries and avoid infeasible computations.

1 Introduction

Deductive database or knowledge base systems require estimates of relation sizes so that a good order of evaluation can be found for conjunctive queries [IW87,KBZ86,SG88,SG85]. Avoiding a bad choice of evaluation order is critical to the efficiency of the query evaluation procedure. For example, in the query

```
richCASenator(X) :- livesInCA(X) &
                    rich(X) &
                    senator(X).
```

first finding all senators (100) then finding those that live in California (2) then checking to see if those two are rich would be far more efficient than first finding all California residents (millions) then checking these to see if they are rich and senators.

Some progress has been made toward estimating the size of relations that are defined by relational expressions rather than being stored in the database [Chr83,Dem80,HOT88,Lyn88,PSC84, Row83]. However, currently there are no known techniques for estimating the size of relations that are recursively defined. Such relations arise in not only in deductive database systems, but in any relational system that allows one to take the fixpoint of a relational expression. This paper considers a method for estimating the size of an important subset of such relations, which we term *generalized transitive closures.*

Previous work on generalized closures [AJ87,Ioa86, IR88,LMR87,Lu87,NSRU89,Nau87,RHDM86] has focussed on efficiently constructing the answer to the query rather than on answer size estimation. Of course, constructing the relation is one way to estimate its size; our goal is to get a rough estimate of the size much more cheaply.

The problem of estimating the size of a recursively defined relation is difficult. For concreteness, consider the transitive closure of a binary relation e. We may interpret e as being the edge relation of a digraph. Figure 1 shows a graph, with uniform out-degree 1, such that the closure of the graph is $O(n)$. Figure 2 shows another graph, also with uniform out-degree 1, such that the closure of the graph is $O(n^2)$. An estimation algorithm based on local (rather than global) properties of the graph will miss the difference between the two.

As another example of a hard case, consider the graph in Figure 3. Here, the graph again has uniform in- and out-degree 1. But while the transitive closure of the graph is $O(n^2)$, the closure of the graph formed by deleting the clique at the right is $O(n)$. Note that the portion of the graph that makes the difference between $O(n)$ and $O(n^2)$ here involves only $1/k$ of

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

Proceedings of the Fifteenth International
Conference on Very Large Data Bases

Amsterdam, 1989

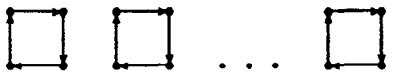

Figure 1: A bounded degree graph with closure $O(n)$.

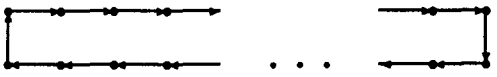

Figure 2: A bounded degree graph with closure $O(n^2)$.

nodes in the graph.

In this paper we give an adaptive sampling algorithm for estimating the size of generalized transitive closures. It is a sampling algorithm in that it estimates the size of the complete closure by sampling subsets of the closure; it is adaptive in that the number of samples made varies with the information obtained from each sample.

For the specific case of the transitive closure, if the underlying graph over which the transitive closure is being computed is of bounded degree, our sampling algorithm runs in linear time and, for $0 < \epsilon < 0.5$, estimates the size of the closure to within a factor of $1/\epsilon$ with probability $1-2\epsilon$. An algorithm for bounded degree graphs is especially important, as we expect that the bulk of relations encountered in practice are of bounded degree when interpreted as graphs.

When we move from the simple transitive closure to generalized closures, the error and accuracy of our estimation algorithm are unchanged. However, the running time will depend on the specific recursion in question.

The estimate produced by our algorithm is admittedly crude — by taking $\epsilon = 0.1$ we see that the algorithm estimates the closure to within a factor of 10 with 80% certainty. However, in optimizing queries, the key is to identify gross differences between the sizes of relations (e.g., the difference between the `livesInCA` relation and the `senator` relation in the example above.)

There is another reason why even rough estimates are essential in database systems that support recursion or fixpoints: even on modest sized databases, recursively defined relations can be too large to be computed. If a system is to be robust, it must detect such computations and produce a warning rather than attempting to compute the relation.

For example, suppose again that we wish to compute the transitive closure of a 100K tuple relation, and that each tuple of the relation takes 100 bytes. (Such a relation is small by database standards — it will easily fit in the memory of many workstations today — but it will serve to illustrate the point.) Then the transitive closure will fall somewhere between $\approx 10^5$ tuples ($\approx$ 10Mbyte) and $\approx 10^{10}$ tuples ($\approx$ a terabyte.) Computing a terabyte relation will almost certainly be infeasible, no matter how clever an algorithm we use to compute the closure. The estimating algorithm presented in this paper can determine quickly whether the proposed relation can sensibly be computed.

In Section 2 we present an urn model that establishes the theoretical underpinnings for the estimation algorithm. Section 3 discusses a general approach to estimating the size of binary recursively defined relations based on the results of Section 2. Section 4 considers the special case of the transitive closure. We conclude in Section 5.

2 An Urn Model

In this section we consider an urn model that abstracts the estimation problem.

Consider an urn U in which there are n *balls*. Associated with ball i is a number $1 \leq a_i \leq n$, for $1 \leq i \leq n$. The *cost* of sampling ball i is a_i; the cost of a set of samples is the sum of the cost of the samples in the set. The quantity we wish to estimate is $A = \sum_{i=1}^{n} a_i$. Consider doing so with the following procedure:

Algorithm 2.1 Repeatedly sample with replacement until S, the sum of the costs on the balls sampled, is greater than $2n$. Let the number of samples

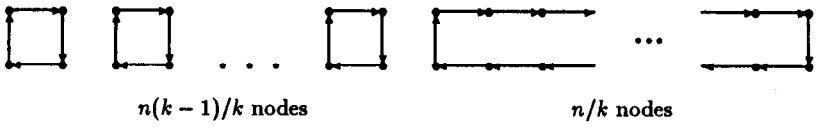

Figure 3: A hard example for size estimation.

this takes be m. Estimate that $A = nS/m$.

The main result of this section is the following theorem.

Theorem 2.1 *Let $\tilde{A}$ be the estimate of A in the above procedure, and let $A > 0$. Then for $0 < \epsilon < 0.5$, with probability $\geq 1 - 2\epsilon$ Algorithm 2.1 estimates A to within a factor of $1/\epsilon$.*

Before proving the theorem, we establish a series of lemmas. Let $\tilde{X}_i$ be a random sample from the urn.

Lemma 2.1 *Let $\tilde{X}_i$ be a random sample from the urn. Then*

1. $E[\tilde{X}_i] = A/n$.
2. $E[\tilde{X}_i^2] \leq A$.
3. $Var[\tilde{X}_i] \leq A$.

Proof:

1. $E[\tilde{X}_i] = \sum_{i=1}^{n} a_i/n = A/n$.
2. $E[\tilde{X}_i^2] = \sum_{i=1}^{n} a_i^2/n \leq \sum_1^n a_i = A$ (since $a_i/n \leq 1$).
3. $Var[\tilde{X}_i] = E[\tilde{X}_i^2] - E[\tilde{X}_i]^2 \leq A$.

□

The next lemma is useful in proving a lower bound on the expected number of samples made before stopping. We represent the probability of an event x by $Pr[x]$.

Lemma 2.2 *Let $m = \epsilon n^2/A$. Then*

$$Pr[\tilde{X}_1 + \ldots + \tilde{X}_m \geq cn] \leq \epsilon/(c-\epsilon)^2$$

Proof: Then

$$\begin{aligned} &Pr[\tilde{X}_1 + \ldots + \tilde{X}_m \geq cn] \\ &\quad = Pr[(\tilde{X}_1 - A/n) + \ldots + (\tilde{X}_m - A/n) \geq cn - \epsilon n] \\ &\quad \leq \frac{1}{(c-\epsilon)^2 n^2} m Var[\tilde{X}_i] \end{aligned}$$

But by Lemma 2.1, $Var[\tilde{X}_i] \leq A$, so

$$\begin{aligned} Pr[\tilde{X}_1 + \ldots + \tilde{X}_m \geq cn] &\leq \frac{1}{(c-\epsilon)^2 n^2} \frac{\epsilon n^2}{A} A \\ &\leq \frac{\epsilon}{(c-\epsilon)^2} \end{aligned}$$

□

We bound the error in the estimate with the following lemma.

Lemma 2.3 *Let $m \geq \epsilon n^2/A$ and $d > 0$. Then*

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

Proof:

$$\begin{aligned} &Pr\left[\left|\frac{n}{m}\left(\sum_{i=1}^{m} \tilde{X}_i\right) - A\right| \geq t\right] \\ &\quad = Pr\left[\left|\left(\sum_{i=1}^{m} \tilde{X}_i\right) - \frac{m}{n}A\right| \geq mt/n\right] \\ &\quad = Pr\left[\left|\left(\sum_{i=1}^{m} \tilde{X}_i - \frac{1}{n}A\right)\right| \geq mt/n\right] \\ &\quad \leq \frac{n^2}{m^2 t^2} m Var[\tilde{X}_i] \end{aligned}$$

Let $m = \lambda n^2/A$, where $\lambda \geq \epsilon$. Then the above probability is less than or equal to

$$\frac{n^2}{m^2 t^2} m Var[\tilde{X}_i] \leq \frac{n^2}{\lambda \frac{n^2}{A} t^2} A$$

which is just $A^2/\lambda t^2$. Setting $Ad = t$ gives

$$\frac{1}{\lambda d^2} \leq \frac{1}{\epsilon d^2}$$

□

We can now prove Theorem 2.1.

Proof: (Theorem 2.1) There are two ways that the algorithm can fail — it can stop too early to guarantee a good error bound, or it can stop after enough samples but with a bad estimate.

First we claim that the procedure is unlikely to stop with $m \leq n^2/A$. We have that

$$Pr[((\exists j)(j \leq m) \wedge (\tilde{X}_1 + \ldots + \tilde{X}_j \geq cn))]$$
$$\leq Pr[\tilde{X}_1 + \ldots + \tilde{X}_m \geq cn]$$

where $m = \epsilon n^2/A$, because the event to the right of the inequality implies the event to the right. But by Lemma 2.2, the right side of this equation is at most $\epsilon/(c-\epsilon)^2$. Substituting $c = 2$ and noting that $0 < \epsilon < 0.5$, we get that this probability is less than ϵ.

Next we turn to the accuracy of the estimate. If $m = \epsilon n^2/A$, by Lemma 2.3 the estimate,

$$\tilde{A} = \frac{n}{m} \sum_{i=1} m\tilde{X}_i$$

is within dA of A with probability $\geq 1/(\epsilon d^2)$. Letting $d = 1/\epsilon$, this is just ϵ.

Putting the two ways of failure together, we get that the total probability of failure is less than $\epsilon + (1-\epsilon)\epsilon$, which is less than 2ϵ. Finally, note that because $A > 0$, there must be at least one i such that $a_i > 0$, so the algorithm will terminate. □

An interesting aspect of this urn theorem is that the sampling is adaptive: usually such sampling procedures perform a fixed number of samples. Here it is critical that the procedure adapt its behavior.

3 Binary Relation Estimation

First some notation. We let $R(Q, D)$ represent the binary relation defined by the query Q over the database D. The size of $R(Q, D)$ is just the number of tuples in D, and is represented by $|R(Q, D)|$. Using notation from relational algebra, the tuples in relation R with some constant c in column i is $\sigma_{i=c}(R(Q, D))$. In the remainder of this paper we will assume without loss of generality that $i = 1$. Such a binary query Q can be considered a generalization of the transitive closure, as it is expressing a "reachability" relationship between the constants of D.

We can map the urn problem of Section 2 to the problem of estimating $|R(Q, D)|$ as follows. We let the balls represent the constants appearing in D. If ball i represents a constant c_i, then $a_i = |\sigma_{1=c_i}(R(Q, D))|$. Sampling ball i corresponds to evaluating the query $\sigma_{1=c}(R(Q, D))$. Clearly, for any binary relation $R(Q, D)$, if there are n distinct constants in D, $|\sigma_{1=c}(R(Q, D))| \leq n$, and the relation $R(Q, D) = \cup_{c \in D} \sigma_{1=c}(R(Q, D))$. Note, however, that in the urn model we have $1 \leq a_i \leq n$, whereas here we have $0 \leq |\sigma_{1=c}(R(Q, D))| \leq n$.

We can now state the adaptive sampling algorithm for estimating $|R(Q, D)|$.

Algorithm 3.1 Let $R(Q, D)$ be a binary relation.

1. Set $s \leftarrow 0$.
2. Repeatedly choose a random constant c from D, and set $s \leftarrow s + \max(1, |\sigma_{1=c}(R(Q, D)|)$, until $s \geq 2n$.
3. Let m be the number of vertices chosen in Step 2. Estimate $|R(Q, D)| = ns/m$.

Theorem 3.1 *Let $0 < \epsilon < 0.5$. Then algorithm 3.1 estimates the size of $R(Q, D)$ to within $|R(Q, D)|/\epsilon + n(1 + 1/\epsilon)$ with probability $1 - 2\epsilon$.*

Proof: First, note that if $|\sigma_{1=c_i}(R(Q, D))| \geq 1$ for all i, then the correspondence between Algorithm 2.1 and Algorithm 3.1 is exact. Then by Theorem 2.1, Theorem 3.1 holds.

Next, note that for any sample $|\sigma_{1=c_i}(R(Q, D))|$, we have that

$$|\sigma_{1=c_i}(R(Q, D))| \leq \max(1, |\sigma_{1=c_i}(R(Q, D))|)$$

and

$$\max(1, |\sigma_{1=c_i}(R(Q, D))|) \leq |\sigma_{1=c_i}(R(Q, D))| + 1$$

This means that the quantity actually being estimated, say $|\tilde{R}(Q, D)|$, is such that

$$|R(Q, D)| \leq |\tilde{R}(Q, D)|$$

and

$$|\tilde{R}(Q, D)| \leq |R(Q, D)| + n$$

Since by Theorem 2.1 Algorithm 3.1 estimates $|\tilde{R}(Q, D)|$ to within $|\tilde{R}(Q, D)|/\epsilon$ with probability $1 - 2\epsilon$, Algorithm 3.1 estimates $|R(Q, D)|$ to within $(|R(Q, D)| + n)/\epsilon + n$, or $|R(Q, D)|/\epsilon + n(1 + 1/\epsilon)$. □

If $|R(Q, D)| > O(n)$, then the $O(n)$ term in the error of the estimate is asymptotically neglible.

We now turn to the running time of the algorithm. Because the algorithm is guaranteed to make at most $O(n)$ samples, if the time to perform an individual sample is $O(f(n))$, the time for the entire algorithm is $O(nf(n))$. However, if the problem $R(Q, D)$ has the property that the time to perform a sample is a well-behaved function of the size of the sample, we can prove a tighter bound.

Theorem 3.2 *Suppose that the running time of obtaining a sample $\max(1, |\sigma_{\$j=c_i}(R(Q, D))|)$ is bounded by $f(\max(1, |\sigma_{\$j=c_i}(R(Q, D))|))$, where $f(x)$ is such that for all $a, b > 0$, $f(a) + f(b) \leq f(a + b)$. Then Algorithm 3.1 runs in time $O(f(|R(Q, D)|))$.*

Proof: The proof follows from the observation that, for any function $f(x)$ such that $a, b > 0$ implies that $f(a) + f(b) \leq f(a+b)$, and any set x_i such that $x_i > 0$ for all i, we have $\sum_i f(x_i) = O(f(\sum_i x_i))$. □

The functions of interest in this paper (including n^k, where $k > 0$) are all well-behaved in this way.

Since in Algorithm 3.1 a sample is just a selection on the query, if we are to use our framework to estimate a query Q, we must have an efficient algorithm to answer selection queries on the relation defined by Q.

In the remainder of this section, we use Datalog notation. Informally, the Datalog rule

```
t(X,Y) :- t(X,W), e(W,Y).
```

may be read "the tuple `(X,Y)` is in `t` if there is some tuple `(X,W)` in `t` and some tuple `(W,Y)` in `e`." We consider only "safe" Datalog programs consisting of a linear recursive rule and a nonrecursive rule. Following Prolog conventions, we use uppercase letters to denote variables and lowercase to denote constants. Thus, the relational algebra expression $\sigma_{\$1=c}(t)$ may be rendered `t(c,Y)?`.

Consider estimating the size of the following trivial generalization of the transitive closure:

```
t(X,Y) :- e(X,W),t(W,Y).
t(X,Y) :- t0(X,Y).
```

(It is a generalization in that the body of the nonrecursive rule contains the predicate `t0` rather than `e`.) We might choose to sample the relation `t` based on its first argument, that is, we would evaluate `t(c,Y)` for constants `c` appearing in `t0` and `e`.

The straightforward way of evaluating the query `t(c,Y)` is to begin at the constant `c`, performing a breadth-first search through `e`. For each constant `a` encountered in this search, we check if there is a tuple `(a,b)` in `t0`, and if so, add `b` to the result.

This breadth-first search evaluates `t(c,Y)` in time linear in the sizes of `e` and `t0` hence can be considered "efficient." However, this cost of evaluating `t(c,Y)` is not any function of the answer size, because we could trace arbitrarily long paths through `e` such that there is no `t0` edge reachable from any node along that path.

On the other hand, if we sample on the second column of `t` instead, one can verify that if the number of answers returned by a sample is k, the time to evaluate the sample is $O(k^2)$.

Definition 3.1 Let r be a Datalog rule in which some predicate, say t, appears both in the head and in the body. Then a variable appearing in the same argument position both in the instance of t in the rule head and the instance of t in the rule body is a **`stable variable`**. An argument position of t is a *stable argument position* if it contains a stable variable.

First, we note that if the recursive predicate in a recursion has a stable argument position, a good heuristic is that the stable argument position is the column that should be sampled. This is because zero-size samples can be detected in $O(1)$ (a sample on a constant c is of zero size if and only if no tuple exists in the relation defined by the nonrecursive rule with the constant c in the sampled column.) Unfortunately, having a stable argument position is not sufficient to guarantee that the running time of a sample is a function of the size of the sample.

Example 3.1 Consider the recursion

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

and suppose that the database is

```
a(1,1).     b(1,1).
a(1,2).     b(2,1).
a(1,3).     b(3,1).
.           .
.           .
.           .
a(1,n).     b(n,1).
```

where the relation `e` is empty. Then the sample `t(1,Y)?` will take time $O(n)$, but will only return a single answer. □

Identifying classes of recursions for which the time to compute a sample is a function of the sample size is an interesting open question. The following section shows that the standard transitive closure is one important example of a recursive query for which the running time of a sample is indeed a function of the sample size.

4 Transitive Closure

We will now use theorem 3.1 to provide an estimation result for the transitive closure of binary relations. For the purposes of this section, we consider the binary relation over which the closure is computed to be the edge relation of a digraph G. Hence we will refer to graphs rather than relations, edges rather than tuples, and vertices rather than constants.

Here "sampling the relation" just means choosing a vertex v, then counting the number of vertices reachable from v. If there are m edges in the graph, this

clearly can be done in time $O(m)$. So, a trivial bound on the running time is $O(nm)$. Because $m < n^2$ this is better than the standard n^3 algorithm for computing the closure. However, a much better bound can be proven as a corollary to Theorem 3.2.

Corollary 4.1 *Let G be a digraph with n vertices. Then Algorithm 3.1 estimates the size of the transitive closure of G in time $O(n^2)$.*

Proof: Recall that a sample of a vertex v is the number of vertices in the connected component with v at the root. This quantity can be computed (via depth-first search, for example) in time bounded by the number of edges in the component. But the number of edges is at most the number of vertices squared. So the sample $|R_v(G,Q)|$ can be computed in time $O(|R_v(G,Q)|^2)$. The sum over all samples is $O(n)$, so we have by Theorem 3.2 that the total time is bounded by $O(n^2)$, as required. □

Note that the constant n here is not the size of the relation over which the closure is being performed — rather, it is the number of distinct constants that appear in that relation. In general, the size of the relation varies from n to n^2. For a relation of size n^2, the above bound is linear in the size of the input relation. For bounded degree graphs, we can prove a still better bound.

Corollary 4.2 *Let G be a bounded degree digraph with n vertices. Then Algorithm 3.1 estimates the size of the transitive closure of G in time $O(n)$.*

Proof: The proof follows that of Corollary 4.1, except that for a bounded degree graph there is a constant d such that no node has more than d out-edges. This in turn implies that the number of edges in a connected component is linear in the number of nodes in the component (instead of quadratic in the general case.) □

In the preceding proofs we discussed the sampling evaluation procedure in standard graph-processing terms. In particular, we mention the use of any standard search method, without reference to the number of disk I/O's necessary to perform the computation. If the relation is too big to fit in memory, specialized evaluation algorithms, which attempt to limit the number of disk I/O's, can be used to compute the $\sigma_{\$i=c}(R(Q,D))$. (See, for example, [IR88].)

A more detailed analysis of the special case of the transitive closure appears in [LN89]. The analysis shows that in general the algorithm is $\Theta(n\sqrt{m})$ on digraphs with n nodes and m edges, but is $O(m)$ if the graph is of almost uniform degree.

5 Conclusion

In this paper we have presented a framework for the estimation of the size of recursively defined relations. We have shown how the framework can be used to provide size estimation algorithms for the transitive closure and its generalizations. In a database system that supports recursion, such estimation algorithms are essential in order to optimize conjunctive queries and to avoid computations that are infeasible because the result is too large.

Many interesting questions remain. In general terms, the question is whether good estimation algorithms exist for arbitrary datalog recursions. While the work presented here represents a first step toward such an algorithm, it is possible that datalog is sufficiently powerful that no such general estimation procedure exists. Even if no such algorithm exists, the sampling algorithm presented here will be useful if we are correct in our conjecture that a significant fraction of the recursive relations encountered in practice will be generalized transitive closures.

References

[AJ87] Rakesh Agrawal and H. V. Jagadish. Direct algorithms for computing the transitive closure of database relations. In *Proceedings of the 13th VLDB Conference*, pages 255–266, Brighton, England, September 1987.

[Chr83] Stavros Christodoulakis. Estimating block transfers and join sizes. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 40–54, San Jose, California, May 1983.

[Dem80] R. Demolombe. Estimation of the number of tuples satisfying a query expressed in predicate calculus language. In *Proceedings of the Sixth VLDB Conference*, 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.

[Ioa86] Yannis E. Ioannidis. On the computation of the transitive closure of relational operators. In *Proceedings of the Twelfth*

VLDB Conference, pages 403–411, Kyoto, Japan, August 1986.

[IR88] Yannis E. Ioannidis and Raghu Ramakrishnan. Efficent transitive closure algorithms. In *Proceedings of the Fourteenth VLDB Conference*, pages 382–394, Los Angeles, California, August 1988.

[IW87] Yannis E. Ioannidis and Eugene Wong. Query optimization by simulated annealing. In *Proceedings of the ACM-SIGMOD Conference on the Management of Data*, pages 9–22, San Fransisco, California, May 1987.

[KBZ86] Ravi Krishnamurthy, Haran Boral, and Carlo Zaniolo. Optimization of nonrecursive queries. In *Proceedings of the Twelfth VLDB Conference*, pages 128–137, Kyoto, Japan, August 1986.

[LMR87] Hongjun Lu, Krishna Mikkilineni, and James P. Richardson. Design and evaluation of algorithms to compute the transitive closure of a database relation. In *Proceedings of the Third International Conference on Data Engineering*, pages 112–119, 1987.

[LN89] Richard J. Lipton and Jeffrey F. Naughton. Estimating the size of the transitive closure of a digraph. Submitted for publication, May 1989.

[Lu87] H. Lu. New strategies for computing the transitive closure of a database relation. In *Proceedings of the Thirteenth VLDB Conference*, pages 267–274, Brighton, England, September 1987.

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

[Nau87] Jeffrey F. Naughton. One sided recursions. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 340–348, San Diego, California, March 1987.

[NSRU89] Jeffrey F. Naughton, Yehoshua Sagiv, Raghu Ramakrishnan, and Jeffrey D. Ullman. Efficient evaluation of right-, left-, and combined-linear rules. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Portland, Oregon, May 1989.

[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.

[RHDM86] Arnon Rosenthal, Sandra Heiler, Umeshwar Dayal, and Frank Manola. Traversal recursion: A practical approach to supporting recursive applications. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data*, Washington, D.C., June 1986.

[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.

[SG85] David E. Smith and Michael R. Genesereth. Ordering conjunctive queries. *Artificial Intelligence*, 26:171–215, 1985.

[SG88] Arun Swami and Anoop Gupta. Optimization of large join queries. In *Proceedings of the ACM-SIGMOD Conference on Management of Data*, pages 8–17, Chicago, Illinois, May 1988.