Size-Estimation Framework with Applications to Transitive Closure and Reachability*

Edith Cohen
AT&T Bell Laboratories
Murray Hill, NJ 07974
edith@research.att.com

Revised: July 8, 1996

Abstract

Computing the transitive closure in directed graphs is a fundamental graph problem. We consider the more restricted problem of computing the number of nodes reachable from every node and the size of the transitive closure. The fastest known transitive closure algorithms run in $O(\min\{mn, n^{2.38}\})$ time, where n is the number of nodes and m the number of edges in the graph. We present an $O(m)$ time randomized (Monte Carlo) algorithm that estimates, with small relative error, the sizes of all reachability sets and the transitive closure. Another ramification of our estimation scheme is a $\tilde{O}(m)$ time algorithm for estimating sizes of neighborhoods in directed graphs with nonnegative edge lengths. Our size-estimation algorithms are much faster than performing the respective explicit computations.

1. Introduction

Computing the transitive closure and reachability information in directed graphs is a fundamental graph problem that has many applications, in particular in database systems (see, e.g., Dar [9] and Yannakakis [24]). Consider directed graphs $G = (V, E)$ where $n = |V|$ and $m = |E|$. In the all-pairs problem, the goal is to find all pairs of nodes $(u, v) \in V \times V$, such that u can reach v via a directed path between them. In the single-source problem, the goal is to compute all nodes reachable from a specified source node. In the (symmetric) single-sink problem, the goal is to compute all nodes that can reach the sink. The single-source (or single-sink) problems can be solved using linear-time $O(m + n)$ graph searching techniques such as depth-first search or breadth-first search. The best known bounds for the

*Preliminary version appeared in the proceedings of FOCS '94

all-pairs problem is $O(mn)$, by solving a single-source problem for each node, or alternatively $O(n^{2.38})$ [7] using fast binary matrix multiplication. (see e.g. [8] for background).

The objective, for many applications, is to solve the more restricted *descendant counting* problem, that is, to compute (or *estimate*) for each node the number of nodes reaching or reachable from it and the total size of the transitive closure. Descendant counting is a fundamental problem and hence, there is interest in designing dedicated algorithms that are faster than performing the explicit computations. For example, in query optimization in databases, estimates of query result size are useful both when the size itself is the desired answer or as a way to determine the feasibility of a query. (Papers by Lipton and Naughton [21], and by Lipton et al. [22] provide results and a survey of other work.) Size estimation may also be useful in applications involving multiplications of large matrices with real-valued entries ([R. Lipton, personal communication]). In [5] the author applied (and experimentally evaluated) the use of size estimation for computing sparse matrix products. Sparse matrices can be multiplied using much fewer operations than dense ones. Hence, prior knowledge of the number of non-zero entries in a product of matrices can be used to speed up the multiplication. The problems of estimating the number of nonzero entries in each column, each row, or the whole matrix product of (two or more) matrices reduces naturally to estimating sizes of reachability sets.

For each $v \in V$, denote by $S(v)$ the *reachability set* of v (set of nodes reachable from v) and denote by $T = \sum_{v \in V} |S(v)|$ the size of the transitive closure. We present a Monte Carlo linear time algorithm that produces $\hat{s}(v)$ $(v \in V)$ and $\hat{T}$, high confidence estimates with small relative error on $S(v)$ $(v \in V)$ and T, respectively. By reversing edge directions, the results carry over to the symmetric problem of estimating the number of nodes that can reach each node. Our method introduces a general size-estimation technique. Another important application is estimating sizes of neighborhoods in directed graphs with nonnegative edge lengths. For a node $v \in V$ and $d > 0$, the *d-neighborhood* of v, denoted $N(v,d) = \{u \in V | \text{dist}(v,u) \leq d\}$, is the set of nodes within distance of at most d from v. We present a $O(m \log n + n \log^2 n)$ time Monte Carlo algorithm that for each pair $(v,d) \in V \times \mathcal{R}_+$ produces $\hat{n}(v,d)$, a high confidence, small relative error estimate on the size of the neighborhood $N(v,d)$. (The algorithm outputs a compact representation of all neighborhood sizes such that for a given pair (v,d), an estimate can be obtained in $O(\log \log n)$ time.) Previously, it was not known that sizes of reachability sets can be estimated faster than a transitive closure computation and that neighborhood sizes can be estimated faster than explicit all pairs shortest paths computation ($\tilde{O}(mn)$ time by applying Dijkstra's algorithm for each $v \in V$ as a source [8]). Our algorithms are also simple to implement and we expect them to be useful in practice. Further applications of our scheme are a new approach for computing the transitive closure, fast estimations of sizes of unions of reachability sets, and an online method for counting events in a distributed setting.

For any fixed $\epsilon > 0$ and $\delta \in (0,1]$, our estimates for neighborhoods, reachability sets, and transitive closure sizes (the quantities $\hat{n}(v,d)$, $\hat{s}(v)$, and $\hat{T}$) are such that with probability at least $1-\delta$, the respective relative error ($\frac{||N(v,d)|-\hat{n}(v,d)|}{|N(v,d)|}$, $\frac{||S(v)|-\hat{s}(v)|}{|S(v)|}$, or $\frac{|T-\hat{T}|}{T}$) is at most ϵ. Furthermore, the expected relative error and the variance can be made smaller than any fixed ϵ. More precise tradeoffs between running time and accuracy are presented later on.

We sketch our algorithm and provide some intuition. Our approach utilizes a linear time procedure that for each node $v \in V$, returns a random sample from a distribution with parameter $|S(v)|$. We apply the procedure several times and for each $v \in V$, deduce (estimate) the value of $|S(v)|$ from the samples. The problem of estimating the parameter of a distribution from samples, knowing the family (e.g., Poisson or Exponential distributions), arises frequently in many contexts (see, e.g. [3]). Karger [16] used a similar approach to estimate the size of the min-cut in weighted graphs. To produce the samples, we employ a subroutine for the *least-descendent* mapping: For a directed graph and a ranking of its nodes (a bijection of V onto $\{1, \ldots, n\}$), compute a mapping of every node $v \in V$ to the least-ranked node in $S(v)$. The least-descendent mapping can be computed in $O(m)$ time using any standard linear-time graph searching method (e.g., Depth-First or Breadth-First search). Consider the least-descendent mapping when the ranking is a random permutation. For each $v \in V$, the lowest rank of a node in the set $S(v)$ is highly correlated with the size of $S(v)$. For example, if $S(v)$ contains at least half the nodes, it is very likely that the lowest rank of a node in $S(v)$ is very small, and if $S(v)$ contains only one node, we would expect the lowest rank to be around $n/2$. The estimation algorithm computes the least-descendent mapping for random rankings and uses these lowest-ranks to produce, for each $v \in V$, an estimate of $|S(v)|$. The confidence level and the accuracy of the estimates can be increased by considering least-descendent mappings produced by several such iterations (each utilizing a different random permutation). In effect, in each iteration random *keys* for the nodes are selected independently from some distribution. The least-descendent mapping is computed with respect to the ranking induced by sorting the keys. For each $v \in V$, an estimate $\hat{s}(v)$ is obtained by applying an *estimator* to the keys of the least-descendants of v in the different iterations. We analyze several distributions and estimators.

We provide a parallel version of our reachability sets size estimation algorithm. We reduce the problem to $O(\log n)$ single-source reachability computations on subgraphs of our input graph. Descendant counting plays an important role in parallel depth-first search algorithms for planar and general graphs [1, 15]. Kao and Klein provided a polylog time linear processor algorithm for descendant counting on rooted planar graphs [15]. Their algorithm is based on a fairly complicated reduction to the single-source reachability problem and an efficient parallel algorithm for planar single-source reachability. By combining our results with Kao and Klein's (or the improved Guattery and Miller [12]) planar reachability algorithm, we obtain a parallel algorithm for approximate descendant counting on general planar graphs.

The previous best known algorithm for estimating the size of the transitive closure was given by Lipton and Naughton [20, 21]. For any fixed $\epsilon > 0$ and $\delta \in (0, 1]$, in time $O(n\sqrt{m})$ their algorithm computes an estimate $\hat{T}$ such that with probability at least $1 - \delta$, $|T - \hat{T}| \leq \epsilon T + n(1 + \epsilon)$. When $T = \omega(n)$, their estimate has small relative error (with probability at least $1 - \delta$, $|T - \hat{T}| \leq \epsilon T$). They also showed that for *almost-regular graphs*, where the out-degrees of all nodes are within a constant factor of each other, the algorithm runs in linear $O(m)$ time. Lipton and Naughton posed as an open question the existence of a linear time estimation algorithm for general graphs. Previous algorithms for estimating the size of the transitive closure were based on randomly sampling source-nodes, solving the single-source reachability problem for the sampled nodes, and using this information to estimate the size of the transitive closure. Lipton and Naughton introduced adaptive sampling of source-nodes, that is, instead of initially fixing the number of sampled nodes, according to the specified confidence and accuracy levels, it initially allocates time bounds and samples source-nodes adaptively until the allocated time is used. Our estimation algorithm answers affirmatively the open question posed by Lipton and Naughton. Furthermore, it overcomes other drawbacks of previous algorithms. The source-node-sampling based approaches inherently can not estimate sizes of reachability sets of nodes other than the sampled source-nodes. Our algorithm estimates sizes of all reachability sets, runs in linear time, and has small expected relative error for all input graphs.

In Section 2 we present our estimation framework. In Section 3 we apply the estimation method to sizes of reachability sets and transitive closure. Section 4 is concerned with estimating reachability in parallel. In Section 5 we apply our framework to estimate neighborhood sizes in directed graphs with nonnegative edge lengths. Section 6 contains a discussion of estimators. We consider estimators based on averaging or on the $\lfloor k(1-1/e) \rfloor$th smallest of the keys of least-descendants obtained in k iterations. Section 7 contains the performance analysis for selection-based estimators and Section 8 considers averaging-based estimators. In particular, we provide exact tradeoffs between the estimate quality and the number of iterations. Section 9 is concerned with some implementation issues. In Section 10 we discuss further applications.

2. The estimation framework

We present our size estimation framework. An analysis is provided later on.

Let X and Y be sets and let $S : Y \rightarrow 2^X$ be a mapping from the elements of Y to subsets of X. Let $w : X \rightarrow \mathcal{R}_+$ be nonnegative weights associated with the elements of X. Our objective is to compute estimates on $w(S(y)) = \sum_{x \in S(y)} w(x)$ for all $y \in Y$. We assume that the elements of X and Y and the weights are given, but it is costly to compute $w(S(y))$ for all $y \in Y$. The following *Least-Element Subroutine* (LE) is provided as an oracle. When LE is presented with a ranking $r : X \rightarrow \{1, \ldots, |X|\}$ of the elements of X, it returns a mapping $\mathrm{le} : Y \rightarrow X$, such that for all $y \in Y$, $\mathrm{le}(y) \in S(y)$ and $r(\mathrm{le}(y)) = \min_{x \in S(y)} r(x)$. That is, for each element $y \in Y$, LE computes the least element in $S(y)$ with respect to the ranking r. See Figure 1 for an example of such sets X and Y, some ranking r on the elements of X, and the corresponding mapping le. The applications in this paper utilize only a non-weighted version, where all the elements of X have unit weights and our goal is to compute estimates on $|S(y)|$ for all $y \in Y$. With unit weights, LE is applied with rankings that are random permutations.

The estimation algorithm performs n iterations, where n is determined by the desired accuracy of the estimate. After the iterations, estimates are produced by applying an *estimator* to the values obtained. The ith iteration ($1 \leq i \leq n$) is as follows:

i. Select keys $R_i : X \rightarrow \mathcal{R}_+$, independently for each $x \in X$. The distribution from which $R_i(x)$ is selected is determined by $w(x)$. We analyze the *Exponential* family of distributions, with parameter $w(x)$: The exponential distribution has probability density

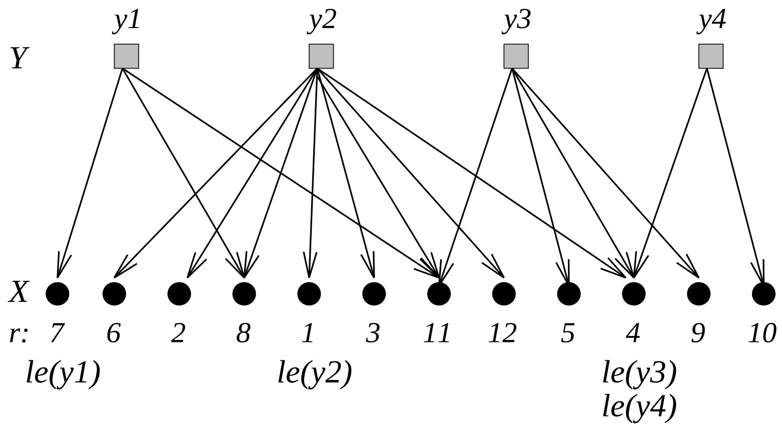


Figure 1: Example of sets X and Y, a ranking r, and $\mathrm{le} : Y \rightarrow X$.

function $w(x)\exp(-w(x)t)$ and distribution function $1 - \exp(-w(x)t)$ $(t \geq 0)$. (One way to obtain samples is to sample z from Uniform $[0, 1]$ and output $-(\ln z)/w(x)$).

ii. Apply LE with the ranking on X induced by sorting the keys R_i. Denote by $\mathrm{le}_i : Y \rightarrow X$ the mapping returned by LE.

The following claims are immediate.

Proposition 2.1 *For all $y \in Y$, the distribution of the minimum key $R(le(y))$ depends only on $w(S(y))$.*

Proof: The minimum of k r.v.'s with distributions with parameters $w_1, \ldots, w_k$ has distribution with parameter $\sum_{j=1}^{k} w_j$. ∎

Proposition 2.2 *For all $y \in Y$ and $x \in S(y)$,* $\mathrm{Prob}\{x = le(y)\} = \frac{w(x)}{w(S(y))}$. *In unweighted settings, $le(y)$ has a uniform distribution over $S(y)$.*

For each element $y \in Y$, we estimate $w(S(y))$ by applying an estimator to the values $R_i(\mathrm{le}_i(y))$ $(1 \leq i \leq n)$. We consider estimators based on:

i. averaging the n samples.

$$\hat{s}(y) \equiv \frac{n}{\sum_{1 \leq i \leq n} R_i(\mathrm{le}_i(y))}, \frac{n-1}{\sum_{1 \leq i \leq n} R_i(\mathrm{le}_i(y))}$$

ii. selection from the samples. Let $\tilde{\mathrm{le}}(y)$ be the $\lfloor n(1-1/e) \rfloor$-smallest value in the sequence $R_i(\mathrm{le}_i(y))$ $(1 \leq i \leq n)$.

$$\hat{s}(y) \equiv \frac{1}{\tilde{\mathrm{le}}(y)}$$

Using the Uniform distribution. An alternate family of distributions that provides the same asymptotic convergence is *Minimum of Uniform distributions:* The key of $x \in X$ is sampled from a distribution $U^{(w(x))}$ with probability density function $w(x)(1-t)^{w(x)-1}$ and distribution function $1-(1-t)^{w(x)}$ $(0 \le t \le 1)$. For unit weights, $U^{(1)}$ is the uniform distribution on the interval $[0,1]$. (One way to sample from $U^{(w)}$ is to sample z from Uniform $[0,1]$ and output $1-z^{1/w}$.) For our estimation framework to be effective, we require $w(S(y)) \ge 1$ for all $y \in Y$. With this family of distribution we consider the averaging estimator

$$\hat{s}(y) \equiv \frac{n}{\sum_{1 \le i \le n} R_i(\mathrm{le}_i(y))} - 1$$

and selection estimator

$$\hat{s}(y) \equiv \frac{1}{\tilde{\mathrm{le}}(y)} - 1 \ .$$

Complexity. Computing the estimates $\hat{s}(y)$ for all $y \in Y$ amounts to producing n (sorted) random key assignments and performing n calls to LE. For simplicity of presentation and analysis we assume that the selected keys are real numbers. Simple considerations show, however, that it is sufficient to use $O(\log \epsilon^{-1})$ significant bits, where ϵ is the tolerated relative error. The expected number of other bits is constant, hence $O(\log \epsilon^{-1})$ random bits per key are sufficient. Furthermore, randomized-rounding type analysis establishes that for $n \gg \epsilon^{-1}$, a constant number of significant bits suffices. Standard arguments show that a list of independent identically distributed numbers can be sorted in expected linear time (e.g., using hashing). Hence, for unit weights, the sorted key assignment is selected in $O(|X| \log \epsilon^{-1})$ time. Keys can also be sorted in linear time when, for example, $\log(\max_{x \in X} w(x) / \min_{x \in X} w(x))$ is small or when the weights $w(x)$ $(x \in X)$ are provided sorted.

Convergence. In later sections we establish that the estimates $\hat{s}(y)$ have the following confidence and accuracy levels. These asymptotic bounds are applicable to other variants of the estimators listed above.

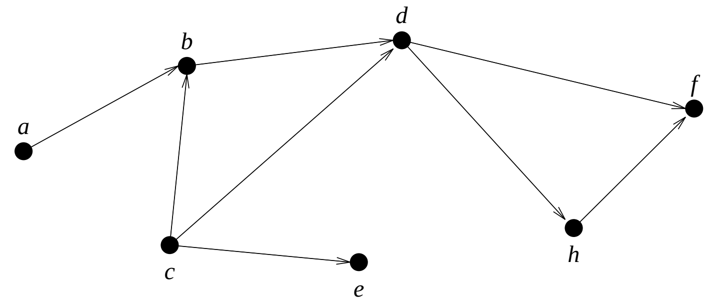
Theorem 2.3 *For all $y \in Y$,*

$$\begin{aligned}
&\textit{for } 0 < \epsilon < 1\ , \quad \mathrm{Prob}\{|w(S(y)) - \hat{s}(y)| \ge \epsilon w(S(y))\} &=& \exp(-\Omega((\epsilon^2 n)))\ . \\
&\textit{for } \epsilon \ge 1\ , \quad \mathrm{Prob}\{|w(S(y)) - \hat{s}(y)| \ge \epsilon w(S(y))\} &=& \exp(-\Omega((\epsilon n)))\ .
\end{aligned}$$

Theorem 2.4 *For all $y \in Y$,*

$$E(|w(S(y)) - \hat{s}(y)|/w(S(y))) = O(1/\sqrt{n})\ .$$

In Section 7 we establish the correctness for the selection based estimators (using Chernoff bounds). In Section 8 we provide exact analytic expressions for the dependence of the confidence on the accuracy, the bias, the variance, and the relative error for the averaging based estimators with exponentially distributed keys.



Reachability sets: *S(a)={a,b,d,f,h} S(b)={b,d,f,h} S(c)={b,c,d,e,f,h}*
S(d)={d,f,h} S(e)={e} S(f)={f} S(h)={f,h}

Example: For ranks such that *r(e)<r(b)<r(d)<r(a)<r(c)<r(f)<r(h)*
we have *le(a)=b,le(b)=b,le(c)=e,le(d)=d,le(e)=e,le(f)=f,le(h)=f*

Figure 2: Example of a graph, a ranking of the nodes, and a corresponding mapping

3. Estimating reachability

We apply the framework of Section 2 to estimate sizes of reachability sets and the transitive closure. The objective is to compute for each $v \in V$ an estimate $\hat{s}(v)$ for $|S(v)|$ (the number of descendants of v) and an estimate $\hat{T}$ for $T = \sum_{v \in V} |S(v)|$ (the size of the transitive closure). The sets X and Y correspond to the vertex set V. The elements of X have unit weights. The mapping S maps each node v to the set of nodes reachable from v. The mapping le maps each $v \in V$ to the least ranked descendent of v. See Figure 2 for an example of a graph, the corresponding reachability sets, and the mapping le with respect to some ranking.

The following algorithm inputs an arbitrary ranking of the nodes and computes the mapping le in $O(m)$ time. The algorithm may employ any linear-time graph search (e.g., depth-first-search or breadth-first search). Suppose the nodes $v_1, \ldots, v_n$ are sorted in increasing order according to their key.

Algorithm 3.1 *(Least-Descendant Subroutine)*

i. Reverse the edge directions of the graph.
   Iterate Step ii until $V = \emptyset$.

ii. Let $i \leftarrow \min\{j | v_j \in V\}$.
   Perform a search to find all nodes $V_i \subset V$ reachable from v_i.
   For every $v \in V_i$, let $\mathrm{le}(v) \leftarrow v_i$.
   Let $V \leftarrow V \setminus V_i$.
   Remove from E all edges incident to nodes in V_i.

Each iteration of the estimation algorithm amounts to selecting a sorted key assignment (can be performed in $O(n)$ time) and applying Algorithm 3.1. Therefore, each iteration takes $O(m)$ time.

Suppose that we utilize k iterations. It follows from Theorems 2.3 and 2.4 that the estimation algorithm runs in time $O(km)$ and produces estimates such that

i. For any $v \in V$, for $0 < \epsilon \leq 1$,

$$\text{Prob}\{||S(v)| - \hat{s}(v)| \geq \epsilon |S(v)|\} = \exp(-\Omega((\epsilon^2 k))) .$$

ii. For any $v \in V$,

$$E\left(\frac{||S(v)| - \hat{s}(v)|}{|S(v)|}\right) = O(1/\sqrt{k}) .$$

Note that if we choose $k = O(\epsilon^{-2} \log n)$ then with probability $1 - O(1/\operatorname{poly}(n))$ our estimates are such that

$$\text{for all } v \in V, \ \frac{||S(v)| - \hat{s}(v)|}{|S(v)|} \leq \epsilon .$$

We use the estimator $\hat{T} \equiv \sum_{v \in V} \hat{s}(v)$ for T, the size of the transitive closure.

Corollary 3.2 $E\left(\frac{|\hat{T}-T|}{T}\right) = O(\frac{1}{\sqrt{k}})$

Proof: Note that $T = \sum_{v \in V} |S(v)|$. We have

$$E(|\hat{T} - T|) \leq E(\sum_{v \in V} |\hat{s} - |S(v)||) = \sum_{v \in V} E(|\hat{s} - |S(v)||) = O(1/\sqrt{k}) \sum_{v \in V} |S(v)| = O(T/\sqrt{k}) .$$

∎

Standard considerations establish the following:

Proposition 3.3

$$\text{Prob}\{|\hat{T} - T| \geq \epsilon T\} = \exp(-\Omega((\epsilon^2 k))) .$$

Remark 3.4 The estimate $\hat{T}$ on the size of the closure has the same worst-case performance as estimates on sizes of individual reachability sets. This is indeed tight if there is a large correlation between reachability sets (for example, when all (a large fraction of) the nodes have (almost) identical reachability sets). In practice, however, it is reasonable to expect the estimate on the size of the closure to converge much faster than the estimates $\hat{s}(v)$.

4. Estimating reachability in parallel

We present a linear-processor polylog-time reduction of the reachability estimation problem to the single-source reachability problem. Hence, a parallel estimation algorithm is advantageous on networks and running time bounds where single-source reachability can be solved more efficiently than transitive closure. Efficient polylog-time single-source reachability algorithms are known for some restricted families of graphs, for example planar graphs [15] or layered graphs when the number of layers is small. For general graphs, Ullman and Yannakakis [23] presented a reachability algorithm with tradeoffs between the time and the work and Klein and Sairam [18] gave a $\tilde{O}(m^2)$ work polylog time single-source reachability algorithm (for sparse graphs it outperforms the $O(n^{2.38})$ [7] transitive closure algorithm).

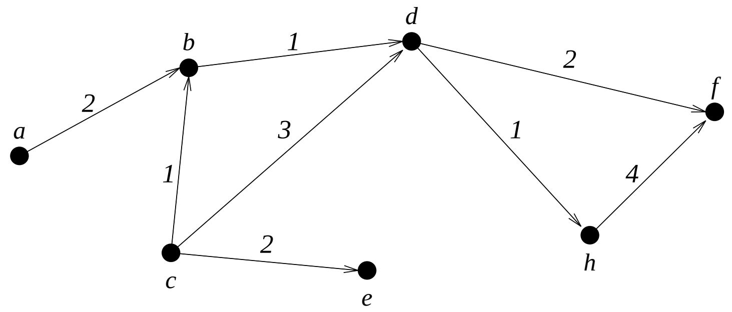
Remark 4.1 Our parallel estimation algorithm utilizes the *multi-source reachability* problem, where for $U \subset V$, the goal is to compute $\bigcup_{u \in U} S(u)$, the set of nodes reachable from U. This problem can be reduced to single source reachability by adding a "super source" with outgoing edges to all nodes in U. However, adding a super source may alter a desirable structure of the graph such as planarity, that allows for efficient parallel single source reachability computations. We show that the multi-source problem reduces to the single source problems on subgraphs of the input graph. We sketch the reduction. Perform $O(\log |U|)$ phases sequentially. In phase i: select $O(2^i)$ random nodes from U, compute single source reachability from each of the selected nodes, and remove all nodes and edges traversed from the graph. Standard arguments establish that each edge of the graph participates in a constant expected number of single-source computations. Hence the multi-source problem has the same work bound and is $O(\log U)$ times slower than the single-source problem.

The k iterations of the sequential estimation algorithm are independent, and therefore can be performed in parallel. Each iteration amounts to computing a sorted key assignment and the respective least-descendant mapping le. The sequential algorithm of Section 3 computes le by performing a sequence of n dependent partial graph searches. Below we present a parallel least-descendant algorithm. The parallel algorithm performs $O(\log n)$ phases. In each phase we have a collection of disjoint subgraphs of the original graph. Each phase amounts to performing a multi-source reachability computation on each of the subgraphs. These computations result in further partitioning of the subgraphs. Hence, the time and work bounds of the parallel least-descendant computation are $O(\log n)$ times the time and work bounds of a multi-source reachability computation, and $O(\log^2 n)$ times the time and $O(\log n)$ times the work of a single-source computation on the input graph. The algorithm partitions the graph recursively. For each subgraph $H = (V_H, E_H)$ in the partition we maintain a list ℓ_H of nodes ($\ell_H \subset V_H$) that contains, for every $v \in V_H$, the lowest-ranked node in $S(v)$. Initially the partition includes only the input graph and its associated list contains all nodes, sorted by key in decreasing order. In each phase, the algorithm considers every subgraph H in the partition. If $|\ell_H| = 1$ (the list of H contains a single node), we map each node in V_H to $v \in \ell_H$ (for all $u \in V_H$, $\mathrm{le}(u) \leftarrow v$) and remove H. Otherwise, if $|\ell_H| > 1$, we apply a divide-and-conquer subroutine that partitions H into two subgraphs H_1, H_2 with associated lists ℓ_{H_1}, ℓ_{H_2} as follows:

i. $|\ell_{H_1}| \le \lceil |\ell_H|/2 \rceil$ and $|\ell_{H_2}| \le \lfloor |\ell_H|/2 \rfloor$

ii. for $i = 1, 2$, for each $v \in V_{H_i}$, the lowest-ranked node in $S(v)$ is contained in ℓ_{H_i}.

We sketch the subroutine that partitions H: Perform a multi-source reachability computation from the set comprising the $\lceil |\ell_H|/2 \rceil$ lowest-ranked nodes in the list ℓ_H. The first subgraph H_1 consists of all reachable nodes and the edges incident to these nodes. Its associated list ℓ_{H_1} contains the first $\lceil |\ell_H|/2 \rceil$ nodes in ℓ_H. The second subgraph H_2 contains all the nodes that were not reachable and their incident edges. The list $\ell_{H_2} \leftarrow \ell_H \setminus H_1$ is contained in the highest ranked $\lfloor |\ell_H|/2 \rfloor$ nodes of ℓ_H. It is easy to verify that if ℓ_H contained $\mathrm{le}(v)$ for all $v \in V_H$, then the two subgraphs H_1, H_2 possess the desired properties. Since the claim holds in the first phase, by induction it holds when the partitioning is halted and the lists contain



Some neighborhoods: *N(c,4)={b,c,d,e,f,h} N(d,1)={d,h} N(h,3)={h}*
N(a,1)={a} N(a,5)={a,b,d,f,h} N(c,2)={b,c,d,e}

Example: For ranks such that *r(e)<r(b)<r(d)<r(a)<r(c)<r(f)<r(h)*
the associated lists are:
a: (2,b) (0,a) *b: (0,b)* *c: (2,e) (1,b) (0,c)* *d: (0,d)*
e: (0,e) *f: (0,f)* *h: (4,f) (0,h)*

Figure 3: Example of a weighted graph, a ranking, and the associated lists

a single node. Hence, at termination, each subgraph $H = (V_H, E_H)$ is such that the single node ℓ_H is the least ranked node in $S(v)$ for all $v \in V_H$. It is easy to see that the algorithm terminates after at most $\lceil \log n \rceil$ phases.

5. Estimating neighborhood sizes

The computation of all neighborhoods amounts to an all-pairs shortest paths computation. We present an algorithm for estimating the sizes of all neighborhoods in directed graphs with nonnegative edge-lengths. The algorithm outputs for each node $v \in V$, an *estimate list* that captures the estimated sizes of all neighborhoods. The estimate list of v is a list of pairs $(D(v)_i, S(v)_i)$ $(i = 0, \ldots, n(v))$ such that:

- $D(v)_i$ and $S(v)_i$ are increasing, $0 = D(v)_0 < D(v)_1 < \cdots < D(v)_{n(v)+1} \equiv \infty$ and $1 \le S(v)_0 < S(v)_1 < \cdots, S(v)_{n(v)} \equiv n$.
- For all pairs $(v, d) \in V \times \mathcal{R}_+$, we use $\hat{n}(v, d) = S(v)_j$, where $D(V)_j \le d < D(V)_{j+1}$.

Therefore, when the estimate lists are provided, an estimate on the neighborhood size of a query pair (v, d), is obtained in time $O(\log n(v))$, using a binary search.

We apply the estimation framework of Section 2 where the set X corresponds to the vertex set V and has unit weights, the set Y is the collection of all pairs $(v, d) \in V \times \mathcal{R}_+$, and S maps each pair (v, d) to $N(v, d) \subset V$. For a given ranking of the nodes, le maps each pair (v, d) to the least ranked node in $N(v, d)$. For every key assignment R and $v \in V$, the value $R(\mathrm{le}(v, d))$ is a piecewise constant nonincreasing function of d. We represent le for all values $d \in \mathcal{R}_+$ as a *least-element list* of labeled intervals. The least-element list of a node $v \in V$ is a list of pairs $(a_v(i), u_v(i)) \in \mathcal{R}_+ \times V$ $(1 \le i \le \ell_v)$ such that:

- $a_v(1) > \cdots > a_v(\ell_v) = 0$ $(a_v(0) \equiv \infty)$, and

- for all $1 \le i \le \ell_v$ and $a_v(i-1) > d \ge a_v(i)$, $\text{le}(v,d) = u_v(i)$ ($u_v(i)$ is the least rank node in $N(v,d)$).

See Figure 3 for an example of a weighted graph, some neighborhoods, a ranking of the nodes, and the corresponding least-element lists. In Subsection 5.1 we present a least-element lists algorithm and establish the following:

Proposition 5.1 *If the ranking $r : V$ of the nodes is a random permutation then*

i. the algorithm runs in $O(m \log n + n \log^2 n)$ expected time (for unit edge lengths in $O(m \log n)$ expected time), and

ii. the expected size of each least-element list is $O(\log n)$.

We perform k iterations of selecting keys and computing least-element lists. Let le_i be the least-element mapping, R_i the key assignment, and $(a^i_v(j), u^i_v(j))$ $(1 \le j \le \ell^i_v)$ be the least-element lists of the ith iteration. The value of the averaging-based estimators for $N(v,d)$ is determined by the sum $s_v(d) = \sum_{i=1}^k R_i(\text{le}_i(v,d))$. Similarly, the value of the selection based estimators is determined by $m_v(d)$, the $\lfloor k(1-1/e) \rfloor$ smallest of $\text{le}_1(v,d), \ldots, \text{le}_k(v,d)$. It is easy to verify that the functions $s_v(d)$ and $m_v(d)$

i. are piecewise constant functions of d,

ii. are nonincreasing (since for all i and v, the least key $R_i(\text{le}_i(v,d))$ is nonincreasing with d),

iii. and have $\sum_i \ell^i_v$ breakpoints $\{a^i_v(j) | 1 \le i \le k, 1 \le j \le \ell^i_v\}$.

It is easy to see that an interval representation of $s_v(d)$ and $m_v(d)$ (and hence, the estimate list of v) can be computed in $O(\sum_i \ell^i_v \log k)$ time by first merging the k sorted lists $a^i_v(j)$ $1 \le i \le k$ and performing $O(1)$ operations per breakpoint of the merged list.

The expected size of the least-element lists is $O(\log n)$. Hence, the estimate lists have an expected number of $O(k \log n)$ breakpoints. Note that for accuracy ϵ, the estimate lists can be reduced to size $O(\epsilon^{-1} \log n)$. For each query pair (v,d), the estimate $\hat{n}(v,d)$ can be computed in $O(\log k + \log\log n)$ time, using a binary search on the estimate list of v.

The theorem follows from Proposition 5.1 and Theorems 2.3 and 2.4.

Theorem 5.2 *The k-iteration algorithm produces the estimate lists in $O(k(m \log n + n \log^2 n))$ expected time ($O(km \log n)$ for unit lengths). For a query pair $(v,d) \in V \times \mathcal{R}_+$, an estimate $\hat{n}(v,r)$ can be computed in $O(\log k + \log\log n)$ expected time. The estimates are such that:*

i. For $\epsilon > 0$,

$$\text{Prob}\{||N(v,d)| - \hat{n}(v,d)| \ge \epsilon |N(v,d)|\} = \exp\left(-\Omega(\epsilon^2 k)\right) .$$

ii.

$$E(||N(v,d)| - \hat{n}(v,d)|/|N(v,d)|) = O(1/\sqrt{k}) \ .$$

Choosing $k = O(\epsilon^{-2}\log n)$ *guarantees that with probability* $1 - O(1/\operatorname{poly}(n))$,

$$\textit{for all } (v,d) \in V \times \mathcal{R}_+,\ \frac{||N(v,d)| - \hat{n}(v,d)|}{|N(v,d)|} \leq \epsilon \ .$$

5.1. Computing least-element lists

We present the least-element-lists algorithm and prove Proposition 5.1. The algorithm is based on a modified Dijkstra's algorithm (see, e.g. [8]). We assume that the nodes $v_1, \ldots, v_n$ are sorted by key in increasing order. We denote by e_{ij} the edge from v_i to v_j. Let $D : E \rightarrow \mathcal{R}_+$ be the lengths of the edges.

Algorithm 5.3 *(Compute least-element lists)*

i. Reverse the edge directions of the graph.
   For $i = 1, \ldots, n$: $d_i \leftarrow \infty$
   For $i = 1, \ldots, n$: initialize the list of v_i to the empty list.

ii. For $i = 1, \ldots, n$: (modified Dijkstra's algorithm):

   (1) Start with an empty heap. Place v_i on the heap with label 0.

   (2) Iterate the following until the heap is empty:
   Remove the node v_k of minimum label from the heap. Let d be the label of v_k.
   Place the pair (d, v_i) on v_k's list.
   Let $d_k \leftarrow d$.
   For each out-neighbor v_j of v_k do as follows:

   - If v_j is in the heap, update its label to the smaller of the current label and $d + D(e_{kj})$.
   - If v_j is not in the heap, then if $d + D(e_{kj}) < d_j$ place v_j on the heap with label $d + D(e_{kj})$.

The following Proposition establishes the correctness of the algorithm.

Proposition 5.4 *i. A node v_k is placed on the heap in iteration i if and only if*

$$\operatorname{dist}(v_i, v_k) < \operatorname{dist}(v_j, v_k) \textit{ for all } j < i.$$

ii. If v_k is placed on the heap during iteration i, then the pair $(\operatorname{dist}(v_i, v_k), v_i)$ is placed on v_k's list and the value of d_k is updated to be $\operatorname{dist}(v_i, v_k)$.

Proof: We first establish that the proposition holds if at the beginning of iteration i, for every $1 \leq k \leq n$, $d_k = \min_{j<i} \mathrm{dist}(v_j, v_k)$. Consider v_k such that $\mathrm{dist}(v_i, v_k) < \mathrm{dist}(v_j, v_k)$ for all $j < i$. We show that v_k is placed on the heap and before the end of the iteration has label $\mathrm{dist}(v_i, v_k)$. The proof is by induction on the number of edges in the shortest path from v_i to $\mathrm{dist}(v_j, v_p)$, then $\mathrm{dist}(v_i, v_k) \geq \mathrm{dist}(v_j, v_k)$, and we get a contradiction. Let v_q be the next-to-last node on the path. The induction hypothesis asserts that v_q was placed on the heap, and was removed when it had label $\mathrm{dist}(v_i, v_q)$. Therefore, when the neighbors of q are scanned, k is placed on the heap with label $\mathrm{dist}(v_i, v_k)$ or if already in the heap, gets its label updated to $\mathrm{dist}(v_i, v_k)$. It remains to prove the assumption that at the end of iteration i, for every $1 \leq k \leq n$, $d_k = \min_{j \leq i} \mathrm{dist}(v_j, v_k)$. The proof is straightforward by induction on the iterations, using the claim proved above. ∎

We analyze the running time of the algorithm when implemented with Fibonacci heaps (see, e.g. [8]). In each iteration, for each placement of a node v in the heap, the algorithm examines each of v's out-neighbors and performs at most one label update for each neighbor. When v is removed from the heap, the algorithm performs one operation of finding the minimum labeled element in the heap. Fibonacci heaps use $O(\log n)$ time to find a minimum element and $O(1)$ time for an insertion or an update. Let ℓ_i denote the number of iterations in which the node v_i was placed on the heap ($1 \leq i \leq n$). It follows that the running time of the algorithm is $O\left(\sum_{1 \leq i \leq n} \ell_i(\log n + \mathrm{outdeg}(v_i))\right)$. Note that ℓ_i is also the size of v_i's list, since a new pair is added in each iteration that places v_i on the heap.

Proposition 5.5 *If the ranking is a random permutation, the expected size of ℓ_i is $O(\log n)$ (for all $1 \leq i \leq n$). Furthermore, the ℓ_i's are all $O(\log n)$ with probability $1 - 1/\mathrm{poly}(n)$.*

Proof: Consider a ranking $v_{i_1}, \ldots, v_{i_n}$ of the nodes according to their distance from v_i. It follows from Proposition 5.4 that the node v_i is placed on the heap at iteration i_j if and only if for all $k < j$, $i_k > i_j$. A standard well-studied Quicksort-type analysis (see, e.g. [19]) concludes the proof. (The permutation order can be viewed as the order of elements chosen to partition the sequence, and all elements larger than the current one constitute a sublist in the partitioning process.) ∎

Therefore, if the ranking is a random permutation, the expected running time of Algorithm 5.3 is $O(m \log n + n \log^2 n)$. When the graph has unit edge length, modified breadth-first searches can replace the modified Dijkstra's algorithm in Step ii of Algorithm 5.3. The latter yields $O(m \log n)$ expected time.

6. Estimating the least-key distribution

We analyze the performance of the estimation framework when keys of elements $x \in X$ are selected according to the exponential or uniform distributions. For all $y \in Y$, the minimum key of an element in $S(y)$ has distribution $M^{(w(S(y)))}$ that depends only on $w(S(y))$. We discuss the form of $M^{(k)}$ for both families of distributions. Each iteration of the estimation framework supplies us with a random sample from $M^{(w(S(y)))}$. The estimation algorithm estimates $k = w(S(y))$ by applying an *estimator* $\hat{k} : \mathcal{R}^n \rightarrow \mathcal{R}$ to n independent samples from $M^{(k)}$ (obtained in n iterations). We discuss criteria for measuring the performance of different estimators. In Section 7 we establish the asymptotic bounds for the selection-based estimators and in Section 8 we analyze the averaging-based estimators.

6.1. Exponentially distributed keys

The exponential distribution with parameter w has density function we^{-wx} $(x \geq 0)$, distribution function $1 - e^{-wx}$ $(x \geq 0)$, expected value $1/w$, and variance $1/w^2$. Consider the r.v. that is the minimum of ℓ independent exponential r.v.'s with weights $w_1, \ldots, w_\ell$, where $k = \sum_{i=1}^{\ell} w_i$. This r.v. is exponentially distributed with parameter k (see,e.g, Feller [10]). Hence, when keys are selected according to the exponential distribution, $M^{(k)}$ is exponentially distributed with parameter k, has density ke^{-kx}, distribution $1-e^{-kx}$ $(x \geq 0)$, expected value $\mu = 1/k$ and variance $1/k^2$.

6.2. Uniformly distributed keys

For a parameter $k > 1$, consider the distribution $U^{(k)}$ such that for $0 \leq t \leq 1$ $\text{Prob}\{M^{(k)} \geq t\} = (1-t)^k$. For integral values of k, $U^{(k)}$ is the distribution of the minimum of k independent r.v.'s uniformly sampled from the interval $[0, 1]$. $U^{(k)}$ has distribution function $1 - (1-t)^k$, probability density function $k(1-t)^{k-1}$ $(0 \leq t \leq 1)$, expectation

$$E(U^{(k)}) = \int_0^1 kx(1-x)^{k-1}dx = 1/(1+k) \ ,$$

second moment

$$E(U^{(k)^2}) = \int_0^1 kx^2(1-x)^{k-1}dx = 2/((2+k)(1+k)) \ ,$$

and hence, variance

$$\text{Var}(U^{(k)}) = E(U^{(k)^2}) - E(U^{(k)})^2 = k/((2+k)(1+k)^2) \ .$$

Consider a r.v. that is the minimum of ℓ elements independently sampled from distributions $U^{(w_i)}$ $(w_i > 0,\ i = 1, \ldots, \ell)$. For $0 \leq t \leq 1$ we have

$$\text{Prob}\{x \geq t\} = \prod_{i=1}^{\ell}(1-t)^{w_i} = (1-t)^{\sum_{i=1}^{\ell} w_i} \ .$$

Hence, this r.v. has distribution $U^{(\sum_{i=1}^{\ell} w_i)}$. Therefore, when keys are sampled from $U^{(w(x))}$, $M^{(k)}$ has distribution $U^{(k)}$.

6.3. Criteria for choosing estimators

We review some basic notions from the theory of statistical inference (see, e.g., Kiefer [17] or Bickel and Doksum [2]) in the context of our framework. Consider the following problem. We are given n independent values $M_i^{(k)}$ $(1 \leq i \leq n)$ from a distribution $M^{(k)}$, for some unknown k. We would like to estimate k, knowing only that $k > 0$. The estimator $\hat{k}$ is a mapping of the form $\hat{k} : \mathcal{R}_+^n \to \mathcal{R}_+$. Our objective is to find an estimator that minimizes the maximum, over $k > 0$ ($k \geq 1$ for the uniform distribution), of the expected value of $W(k, \hat{k})$, where W is the "loss" when the real answer is k, but the estimate is $\hat{k}$. In other words, we would like an optimal minimax estimator according to the loss function $W(k, \hat{k})$. An optimal estimator $\hat{k}$ would minimize

$$\max_{k>0} \int_0^\infty \cdots \int_0^\infty W(k, \hat{k}(x_1, \ldots, x_n))(\prod_{i=1}^n f^{(k)}(x_i))dx_1 \cdots dx_n \ ,$$

where $f^{(k)}(x)$ is the probability density function of $M^{(k)}$.

To tie us back to the framework of Section 2, note that for each $y \in Y$, the minimum key $R(\mathrm{le}(y))$ has distribution $M^{(w(S(y)))}$. Hence, our estimation problem is: given values from the distribution $M^{(w(S(y)))}$, estimate $w(S(y))$.

One loss function that we consider is

$$W(k, \hat{k}) = \begin{cases} 0 & \text{when } k(1-\epsilon) \leq \hat{k} \leq k(1+\epsilon) \\ 1 & \text{otherwise} \end{cases} \tag{1}$$

This loss function means that we are equally happy with all estimates such that $|\hat{k} - k| \leq k\epsilon$ and unhappy, and equally so, with all other estimates. It measures the confidence level in our estimate having a relative error of at most ϵ. (Corresponds to the bound in Theorem 2.3.)

We also consider the loss function

$$W(k, \hat{k}(x_1, \ldots, x_n)) = |\hat{k}(x_1, \ldots, x_n) - k|/k \tag{2}$$

that measures the average relative error (corresponds to the bound in Theorem 2.4), and the loss function

$$W(k, \hat{k}) = \left(\frac{\hat{k} - k}{k}\right)^2 \tag{3}$$

that measures the variance of the estimator. For some applications we would like the estimator to be *unbiased* (have expectation equal to the value estimated). The estimator is unbiased if

$$\forall k > 0, \int_0^\infty \cdots \int_0^\infty (k - \hat{k}(x_1, \ldots, x_n))(\prod_{i=1}^n f^{(k)}(x_i))dx_1 \cdots dx_n = 0 \ .$$

7. Selection-based estimators

We provide an asymptotic analysis of the performance of selection-based estimators and establish that Theorems 2.3 and 2.4 hold. The estimation algorithm performs n iterations and we select the $\lfloor n(1-1/e)\rfloor$-smallest value as an estimator of the expected value. We provide the analysis for the exponential distribution. A very similar argument establishes that the same asymptotic bounds hold when sampling from the uniform distribution, provided that the weight of the estimated quantities is larger than some constant (e.g., $k \geq 1$).) For an integer $n \geq 1$ and $k > 0$, denote by $E^{(n,k)}$ the $\lfloor n(1-1/e)\rfloor$-smallest value amongst n independent random variables distributed according to $M^{(k)}$.

Proposition 7.1

$$\mathrm{Prob}\{E^{(n,k)} \geq (1+\epsilon)\mu\} = \begin{cases} \textit{if } \epsilon < 1, & \exp(-\Omega(\epsilon^2 n)) \\ \textit{if } \epsilon > 1, & \exp(-\Omega(ne^{\epsilon})) \end{cases}$$

Proof: Let $p = \mathrm{Prob}\{M^{(k)} \geq (1+\epsilon)\mu\}$. Note that $p = \exp(-(1+\epsilon))$. Let the r.v. X_n be the number of successful trials among n Bernoulli trials with probability of success p. We have

$$\mathrm{Prob}\{E^{(n,k)} \geq (1+\epsilon)\mu\} = \mathrm{Prob}\{X_n \geq \lceil n/e\rceil\} = \sum_{i=\lceil n/e\rceil}^{n} b(i;n,p) \ .$$

Applying Chernoff's bound [4] we obtain

$$\begin{aligned}
\mathrm{Prob}\{X_n \geq n/e\} &\leq \mathrm{Prob}\{|X_n - np| \geq np(1-1/(pe))\} \\
&\leq \exp(-(1-1/(pe))^2 np/2) \\
&= \exp(-(1-e^{\epsilon})^2 ne^{-1-\epsilon}/2) \ .
\end{aligned}$$

The proof follows. ▮

Proposition 7.2

$$\mathrm{Prob}\{E^{(n,k)} \leq (1-\epsilon)\mu\} = \begin{cases} \textit{if } \epsilon \leq 1/2, & \exp(-\Omega(\epsilon^2 n)) \\ \textit{if } 1 > \epsilon \geq 1/2, & \exp(-\Omega(n/(1-\epsilon))) \\ \textit{if } \epsilon = 1, & 0 \end{cases}$$

Proof: Let $p = \mathrm{Prob}\{M^{(k)} \leq (1-\epsilon)\mu\}$. We have $p = 1-\exp(-(1-\epsilon))$. Let the r.v. X_n be the number of successful trials among n Bernoulli trials with probability of success p. Note that

$$\mathrm{Prob}\{E^{(n,k)} \leq (1-\epsilon)\mu\} = \mathrm{Prob}\{X_n \geq \lfloor n(1-1/e)\rfloor\} = \sum_{i=\lfloor n(1-1/e)\rfloor}^{n} b(i;n,p) \ .$$

Applying Chernoff's bound we obtain

$$\begin{aligned}
\mathrm{Prob}\{X_n \geq n(1-1/e)-1\} &\leq \mathrm{Prob}\left\{|X_n - np| \geq np\left(1-\frac{e-1}{pe}-\frac{1}{np}\right)\right\} \\
&\leq \exp\left(-\left(1-\frac{e-1}{pe}-\frac{1}{np}\right)^2 np/2\right) \\
&= \exp\left(-\frac{(1-e^{\epsilon}-e/n)^2 n}{2e(e-e^{\epsilon})}\right) .
\end{aligned}$$

The proof follows. ∎

Consider the estimator $\hat{k} = 1/E^{(n,k)}$. We bound the relative error of $\hat{k}$:

Corollary 7.3

For $\epsilon < 1$, $\mathrm{Prob}\{\frac{|k-\hat{k}|}{k} \geq \epsilon\} = \exp(-\Omega(\epsilon^2 n))$
For $\epsilon \geq 1$, $\mathrm{Prob}\{\frac{|\hat{k}-k|}{k} \geq \epsilon\} = \exp(-\Omega(\epsilon n))$

The bound on the expected relative error of the estimator follows from Corollary 7.3 using elementary calculus:

Proposition 7.4 $E(|k-\hat{k}|/k) = O(1/\sqrt{n})$

8. Averaging-based estimators

Let $x_1, \ldots, x_n$ be independent samples from the distribution $M^{(k)}$ (for some unknown $k > 0$). For an integer $n \geq 1$ and $k \geq 1$, denote by $S^{(n,k)}$ the r.v. $\sum_{i=1}^n x_i$ (the sum of n independent random variables distributed according to $M^{(k)}$). We estimate k by applying an estimator to $s = \sum_{i=1}^n x_i$. When the Exponential distribution is used to set the keys we consider the estimators

$$\hat{k}(s) = \frac{n}{s}, \frac{(n-1)}{s}$$

We bound the maximum, over $k > 0$, of the expected loss and the variance incurred by the estimator $\hat{k}$ when the true value is k. (The expected loss and variance turns out to be independent of k.) We consider the loss functions stated in Subsection 6.3 and provide exact analytic expressions for the bias, variance, relative error, and the interdependence of the confidence, accuracy, and the number of rounds. We establish that Theorems 2.3 and 2.4 hold when the Exponential distribution is used to set the keys. Similar arguments apply when the keys are sampled according to the uniform distribution, when $k \geq 1$ with the estimator $\hat{k}(s) = \max\{1, n/s - 1\}$.

We now consider the probability density and distribution functions of $S^{(n,k)}$ and deduce the bias, the relative error, and the variance of the estimators. $S^{(n,k)}$ has density and distribution functions

$$
\begin{aligned}
g_{n,k}(s) &= k\frac{(ks)^{n-1}}{(n-1)!}e^{-ks} \\
G_{n,k}(s) &= 1-e^{-ks}\left(1+\sum_{i=1}^{n-1}\frac{(ks)^i}{i!}\right) \ .
\end{aligned}
$$

where $s \geq 0$ (see, e.g. [11]).

Consider the estimator $\hat{k} = (n-1)/s$ and the random variable $y = \hat{k}/k$. The density and distribution functions of y are independent of k and are given by

$$
\begin{aligned}
f_n(y) &= -\frac{(n-1)^n}{(n-1)!y^{n+1}}e^{-\frac{(n-1)}{y}} \\
F_n(y) &= e^{-\frac{(n-1)}{y}}\left(1+\sum_{i=1}^{n-1}\frac{(n-1)^i}{i!y^i}\right)
\end{aligned}
$$

for $y \geq 0$. The following expressions follow for the relative error

$$
\begin{aligned}
\mathrm{Prob}\{\hat{k} \geq k(1+\epsilon)\} &= \mathrm{Prob}\{y \geq 1+\epsilon\} \\
&= 1-e^{-\frac{n-1}{1+\epsilon}}\left(1+\sum_{i=1}^{n-1}\frac{(n-1)^i}{(1+\epsilon)^i i!}\right) \\
\mathrm{Prob}\{\hat{k} \leq k(1-\epsilon)\} &= \mathrm{Prob}\{y \leq 1-\epsilon)\} \\
&= e^{-\frac{n-1}{1-\epsilon}}\left(1+\sum_{i=1}^{n-1}\frac{(n-1)^i}{(1-\epsilon)^i i!}\right)
\end{aligned}
$$

Hence,

$$
\mathrm{Prob}\left\{\frac{|\hat{k}-k|}{k} \geq \epsilon\right\} = 1-e^{-\frac{n-1}{1+\epsilon}}\left(1+\sum_{i=1}^{n-1}\frac{(n-1)^i}{(1+\epsilon)^i i!}\right) + e^{-\frac{n}{1-\epsilon}}\left(1+\sum_{i=1}^{n-1}\frac{n^i}{(1-\epsilon)^i i!}\right) \ .
$$

By integrating we obtain the following. The bias of the estimator is

$$
E\left(\frac{\hat{k}-k}{k}\right) = E(y)-1 = \int_0^\infty y f_n(y)dy - 1 = 0 \ .
$$

(the estimator is unbiased). The expected relative error is

$$
E\left(\frac{|\hat{k}-k|}{k}\right) = E(|1-y|) = \int_0^\infty |1-y|f_n(y)dy = \frac{2(n-1)^{(n-2)}}{(n-2)!e^{n-1}} \approx \sqrt{\frac{2}{\pi(n-2)}} \ .
$$

(using Stirling's formula.) The variance of the estimator is

$$
E\left(\frac{(\hat{k}-k)^2}{k}\right) = E((1-y)^2) = \int_0^\infty (1-y)^2 f_n(y)dy = \frac{1}{n-2} \ .
$$

Consider the estimator $\hat{k}' = n/s$ and the random variable $y = \hat{k}'/k$. Similarly, the density and distribution functions of y are

$$\begin{aligned} f_n(y) &= -\frac{n^n}{(n-1)!y^{n+1}}e^{-\frac{n}{y}} \\ F_n(y) &= e^{-\frac{n}{y}}\left(1+\sum_{i=1}^{n-1}\frac{n^i}{y^i i!}\right) \end{aligned}$$

for $y \geq 0$. It follows that

$$\text{Prob}\left\{\frac{|\hat{k}-k|}{k} \geq \epsilon\right\} = 1 - e^{-\frac{n}{1+\epsilon}}\left(1+\sum_{i=1}^{n-1}\frac{n^i}{(1+\epsilon)^i i!}\right) + e^{-\frac{n}{1-\epsilon}}\left(1+\sum_{i=1}^{n-1}\frac{n^i}{(1-\epsilon)^i i!}\right) ,$$

the bias is $\frac{1}{n}$, and the variance is $\frac{n+2}{(n-2)(n-1)}$.

We sketch the proof of Theorem 2.3 for the estimators $\hat{k}, \hat{k}'$. Note that $\text{Prob}\{\hat{k} \geq k(1+\epsilon)\}$ equals the probability that a Poisson distribution with expectation $\Lambda = \frac{n}{1+\epsilon}$ has n or more successes. $\text{Prob}\{\hat{k} \leq k(1-\epsilon)\}$ equals the probability that a Poisson distribution with $\Lambda = \frac{n}{1-\epsilon}$ has n or fewer successes. The asymptotic bounds can be derived by approximating the Poisson distribution as a limit of Binomial distributions, and applying the Chernoff bounds in a similar manner to Section 7.

Remark 8.1 Billingsley ([3], pp. 368) considers as an example in his book our exact problem of estimating the unknown parameter k of an exponential distribution, given n independent samples from that distribution. As a consequence of the Central Limit theorem and of Skorohod's theorem he obtains that the distribution of the r.v. $\frac{\sqrt{n}}{k}(n/S^{(n,k)} - k)$ converges to the Normal distribution with zero mean and unit variance. That is, for every η,

$$\lim_{n\to\infty} \text{Prob}\{\frac{\sqrt{n}}{k}(n/S^{(n,k)} - k) \leq \eta\} = \Phi(\eta) ,$$

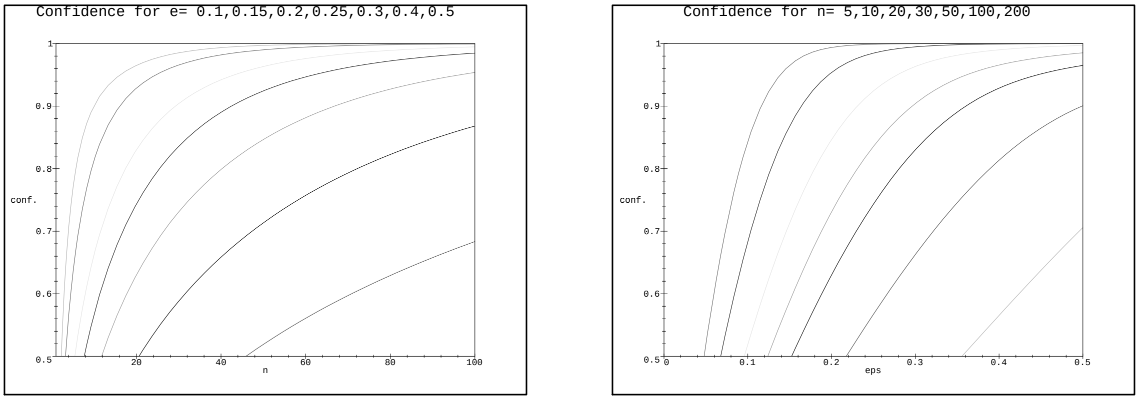
where $\Phi(t) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{t} \exp(-x^2/2)dx$ is the distribution function of the normal distribution.

$$\begin{aligned} \text{Prob}\{\frac{|\hat{k}-k|}{k} \geq \epsilon\} &= \text{Prob}\{\sqrt{n}\frac{|\hat{k}-k|}{k} \geq \sqrt{n}\epsilon\} \\ &\Rightarrow 2\Phi(-\epsilon\sqrt{n}) \leq \frac{2\exp(n\epsilon^2/2)}{\epsilon\sqrt{2\pi n}} \\ &= e^{-\Omega(\epsilon^2 n)} \end{aligned}$$

The last inequality follows from bounds on the tail of the Normal distribution (see, e.g., Feller [10] pp. 175). The expected relative error converges to

$$\begin{aligned} E(\frac{|n/S^{(n,k)} - k|}{k}) &= \frac{1}{\sqrt{n}}E(\sqrt{n}\frac{|\hat{k}-k|}{k}) \\ &\Rightarrow \frac{2}{\sqrt{2\pi n}}\int_0^\infty x\exp(-x^2/2)dx = \sqrt{\frac{2}{\pi n}} \end{aligned}$$

The convergence in probability to the Normal distribution demonstrates that the asymptotic bounds we obtained for our estimators are essentially optimal.



Various accuracy levels *Various numbers of rounds*

Figure 4: Performance of the estimator $(n-1)/s$ with expo. dist. keys

9. Implementation issues

The estimate quality of the estimation framework of Section 2 increases with the number of iterations (LE calls). We considered sampling based on either the uniform or exponential distributions and several estimators based on selection or averaging. These schemes and other variants exhibit the same asymptotic bounds. However, for practical applications it is desirable to determine the precise tradeoffs between number of iterations and accuracy, for each scheme. We studied these tradeoffs using both simulations and analysis. We concluded that our averaging-based estimators perform significantly better than the selection-based ones (require much fewer iterations for similar estimate quality). As for averaging-based estimators, sampling from the exponential distribution yields better performance than the uniform distribution. The choice of an appropriate estimator should depend on the application. Different objectives such as unbiasedness or a particular loss function call for different estimators. For example: the estimator $(n-1)/s$ is unbiased and has smaller variance and expected relative error (the loss functions (2) and (3)) than n/s. However, for small values of n and the loss function (1), it incurs a larger loss. For unit weights, the uniform distribution may be simpler to work with and justify a larger number of iterations.

Plots of the performance of the $(n-1)/s$ estimator with exponentially distributed keys are provided in Figure 4. The figure plots the confidence level as a function of the number of iterations (rounds) n and the accuracy ϵ. The curves were plotted using the derivations in Section 8.

We comment on sampling the keys. In our experiments, the pseudo random generators supplied in standard C (UNIX) environment achieved the expected performance of the estimators. Small precision suffices to represent the keys since we need only a fixed number of significant bits. Also, it is well known that samples from the exponential distribution require an expected small number of random bits. Karger's thesis [16] contains a discussion and bibliography on sampling efficiently from the exponential distribution.

10. Further applications

We discuss additional applications and extensions of this work.

Consider our estimation algorithm. In each iteration, for each node $v \in V$, the algorithm computes a node in $S(v)$, selected uniformly at random (see Proposition 2.2). After k iterations we have for each $v \in V$, a list of k nodes in $S(v)$.

Computing the transitive closure. After k linear time iterations, we obtain for each node v, k independent random samples from the set $S(v)$. It follows that for each $v \in V$, after $k = O(|S(v)| \log |S(v)|)$ iterations, with probability $1 - O(1/\operatorname{poly} n)$, we computed all of $S(v)$. Hence, we obtain a new algorithm to compute the transitive closure, partially or in whole. If $P = \max_{v \in V} |S(v)|$, the transitive closure is computed with probability $1 - O(1/\operatorname{poly} n)$ after $k = P \log P$ iterations (that is, in $O(Pm \log n)$ time). In the worst case, $P = O(n)$ and like previous methods, our algorithm has a $\tilde{O}(mn)$ bound. In some cases, however, it is faster than other transitive closure algorithms. This is of interest since conceivably small *random sample* of elements from each reachability set may be useful for some applications. and furthermore, an abundance of papers in the database literature are concerned with faster transitive closure algorithms for some families of graphs (e.g., Jakobsson gave faster algorithms for graphs with certain connectivity properties [13, 14], also see Dar [9] for an experimental comparison of the performance of different transitive-closure algorithms).

Estimating similarities between reachability sets. The lists of different nodes are correlated and can be used to estimate "similarities" between the reachability sets of these nodes. That is, we can determine with high confidence whether two nodes $\{v, u\}$ are such that

$$|S(v) \cap S(u)| \geq \alpha |S(v) \cup S(u)| \ ,$$

for some fixed constant $\alpha > 0$. In each iteration, the probability that $\mathrm{le}(v) = \mathrm{le}(u)$ is

$$|S(v) \cap S(u)| / |S(v) \cup S(u)| \ .$$

This probability can be estimated by counting the number of equal components in v's and u's lists and dividing it by k.

Estimating sizes of unions of neighborhoods. The lists can be used to estimate for any given subset of nodes $U \subset V$, the number of nodes reachable from U, $|\bigcup_{u \in U} S(u)|$. For each node u, we consider the k-vector of least-keys. The estimate on $|\bigcup_{u \in U} S(u)|$ is produced in $O(|U|k)$ time by applying the estimator to the k-vector obtained by a coordinate-wise minima of the k-vectors of the nodes in U. It is easy to see that the confidence and accuracy levels of the estimate are the same as for single nodes. Similarly, for directed graphs with edge lengths we can estimate sizes of unions of neighborhoods.

On-line estimation of weights of growing sets. Our estimation framework is admissible in on-line settings where the goal is to produce estimates on-line of the weights of dynamically growing sets. Consider the following scenario. Let X be a set of elements with positive weights $w : X \rightarrow \mathcal{R}_+$. Let Y be a collection of subsets of X. The admissible operations on the subsets are:

i. Create a new subset (initialized to $\emptyset$ or a copy of another subset).

ii. Add a new weighted element x to one or more subsets.

iii. Merge two subsets $\{y, y'\} \subset Y$ (replace y by $y \cup y'$).

iv. Weight-query: For a subset $y \in Y$, produce an estimate of $w(y)$.

A straightforward way to support these operations is by explicitly maintaining the contents of each subset. The estimation framework allows us to support this data structure and operations efficiently while providing high confidence and accuracy estimates for the weights. We outline the method. We maintain a small size vector for each subset. The entries of a vector that corresponds to the empty subset are initialized to $+\infty$. A merge of two subsets amounts to performing coordinate-wise minima of the two vectors. Adding a new element x to some subsets amounts to drawing a random keys vector for x and replacing the vector of each subset x is added to by the coordinate-wise minima of the vector of x and the vector of the subset. A weight estimate for a subset y is obtained by applying an estimator to the entries of the vector of y. This application was conceived with Yi-Min Wang and Gaurav Suri, in the context of counting preceding events in a distributed system of communicating processes [6].

Considering ith least-ranked elements. In some situations it is worthwhile to apply an estimator to the keys of the nth least-ranked elements of a set in a single iteration instead of considering the least-ranked element in n different iterations. A more general statement is to perform n/i iterations where in each iteration we consider the key of the ith-least-ranked element. Typically we would like to choose i to be as large as possible under the condition that it is quite unlikely that for the estimated sets we obtain repetitions of the least-ranked element when using i different iterations. For the reachability and neighborhood sizes estimation algorithms, the computation involved in obtaining the ith-least elements is comparable to performing i iterations, and therefore, the only benefit of using ith least-ranked element is to save random bits. In some other applications, however, the use of ith least-ranked elements reduces the amount of computation as well. For example, in on-line applications (as sketched above) when the number of sets is much smaller than the number of elements. For each new element, we draw a single random number instead of a vector of random numbers. For each subset, we maintain a list of the n smallest keys of elements in the subset. Merging two subsets amounts to taking the n smallest keys in the union of the two lists.

Acknowledgements. I am grateful to Shaul Dar, Yoav Freund, Phil Klein, and especially to Mihalis Yannakakis for helpful discussions and bibliographic pointers. I would like to thank David Karger for suggesting to use and analyze the exponential distribution, and to David Karger and David Zukerman for insisting that I should be able to obtain tighter bounds on the estimate quality. I would like to thank the anonymous referees for many thoughtful suggestions.

References.

[1] A. Aggarwal, R. Anderson, and Kao M-Y. Parallel depth-first-search in general directed graphs. *SIAM J. Comput.*, 19:397–409, 1990.

[2] P. J. Bickel and K. A. Doksum. *Mathematical Statistics.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1977.

[3] P. Billingsley. *Probability and measure.* John Wiley & Sons, New York, 1986.

[4] H. Chernoff. A measure of the asymptotic efficiency for test of a hypothesis based on the sum of observations. *Annals of Math. Statistics*, 23:493–509, 1952.

[5] E. Cohen. Optimizing multiplications of sparse matrices. In W. H. Cunningham, S. T. McCormick, and M. Queyranne, editors, *Proc. of the 5th International Conference on Integer Programming and Combinatorial Optimization*, pages 219–233. Springer-Verlag, Lecture Notes in Computer Science Vol. 1084, 1996.

[6] E. Cohen, Y.-M. Wang, and G. Suri. When piecewise determinism is almost true. In *Proc. Pacific Rim International Symposium on Fault-Tolerant Systems*, pages 66–71, December 1995.

[7] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9:251–280, 1990.

[8] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to algorithms.* McGraw-Hill Book Co., New York, 1990.

[9] S. Dar. *Augmenting databases with generalized transitive closure.* PhD thesis, Department of Computer Science, University of Wisconsin, Madison, 1994.

[10] W. Feller. *An introduction to probability theory and its applications*, volume 1. John Wiley & Sons, New York, 1968.

[11] W. Feller. *An introduction to probability theory and its applications*, volume 2. John Wiley & Sons, New York, 1971.

[12] S. Guaterry and G. Miller. A contraction procedure for planar directed graphs. In *Proc. 4th annual ACM Symposium on Parallel Algorithms and Architectures*, pages 431–441. ACM, 1992.

[13] H. Jakobsson. Mixed-approach algorithms for transitive closure. In *Proc. of the 10th ACM SIGACT-SIGMOND-SIGART Symposium on Principles of Database Systems*, pages 199–205. ACM, 1991.

[14] H. Jakobsson. On tree-based techniques for query evaluation. In *Proc. of the 11th ACM SIGACT-SIGMOND-SIGART Symposium on Principles of Database Systems*, pages 380–392. ACM, 1992.

[15] M-Y Kao and P. N. Klein. Towards overcoming the transitive-closure bottleneck: efficient parallel algorithms for planar digraphs. *J. Comput. System Sci.*, 47:459–500, 1993.

[16] D. R. Karger. *Random Sampling in Graph Optimization Problems.* PhD thesis, Stanford University, 1994. available by ftp from `theory.stanford.edu`, directory `pub/karger`.

[17] J. C. Kiefer. *Introduction to statistical inference.* Springer-Verlag, New York, 1987.

[18] P. N. Klein and S. Sairam. A linear-processor polylog-time algorithm for shortest paths in planar graphs. In *Proc. 34th IEEE Annual Symposium on Foundations of Computer Science*, pages 259–270. IEEE, 1993.

[19] D. E. Knuth. *The art of computer programming*, volume 3. Addison-Wesley Publishing Co., Reading, MA, 1973.

[20] R. J. Lipton and J. F. Naughton. Estimating the size of generalized transitive closures. In *Proc. of the 15th Int. Conference on Very Large Databases*, pages 165–172, 1989.

[21] R. J. Lipton and J. F. Naughton. Query size estimation by adaptive sampling. In *Proc. of the 9th ACM SIGACT-SIGMOND-SIGART Symposium on Principles of Database Systems*, pages 40–46. ACM, 1990.

[22] R. J. Lipton, J. F. Naughton, and D. A. Schneider. Practical selectivity estimation through adaptive sampling. In *Proc. of the 1990 ACM SIGMOND Int. Conference on Management of Data*, pages 1–11. ACM, 1990.

[23] J. D. Ullman and M. Yannakakis. High-probability parallel transitive closure algorithms. *SIAM J. Comput.*, 20:100–125, 1991.

[24] M. Yannakakis. Graph-theoretic methods in database theory. In *Proc. of the 9th ACM SIGACT-SIGMOND-SIGART Symposium on Principles of Database Systems*, pages 230–242. ACM, 1990.