Size-Estimation Framework with Applications to Transitive Closure and Reachability

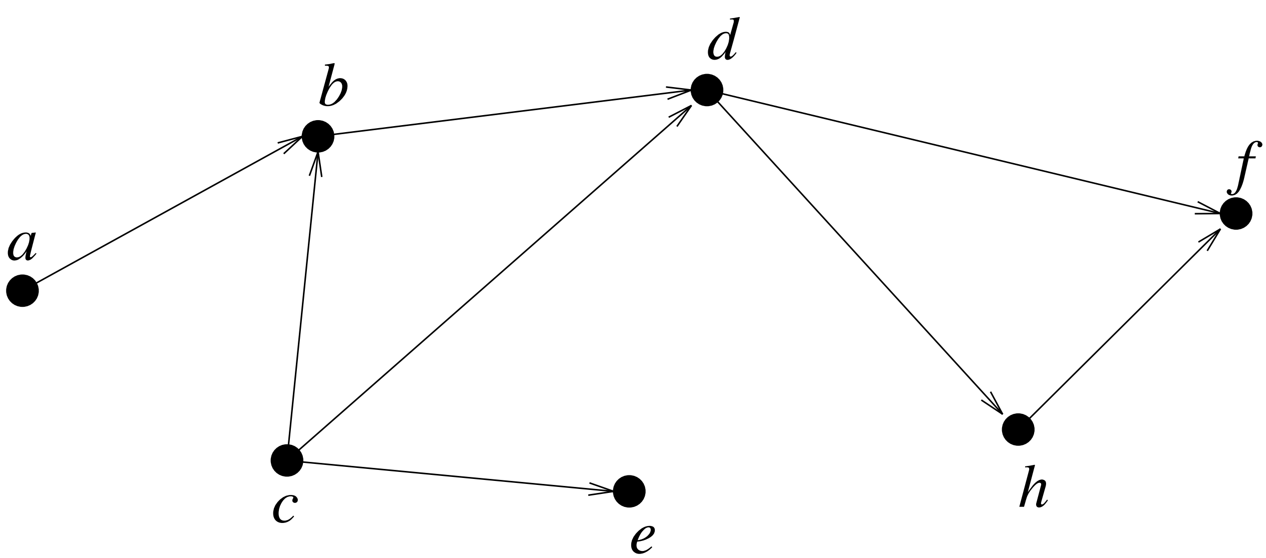
Edith Cohen
AT&T Bell Labs

Reachability and transitive closure

Directed network $G = (V, E)$

- Single source reachability:
  For $v \in V$, compute $S(v) = \{u \in V | v \rightsquigarrow u\}$

- Transitive Closure:
  Find all pairs $(u, v) \in V \times V$ such that $u \rightsquigarrow v$



Reachability sets:

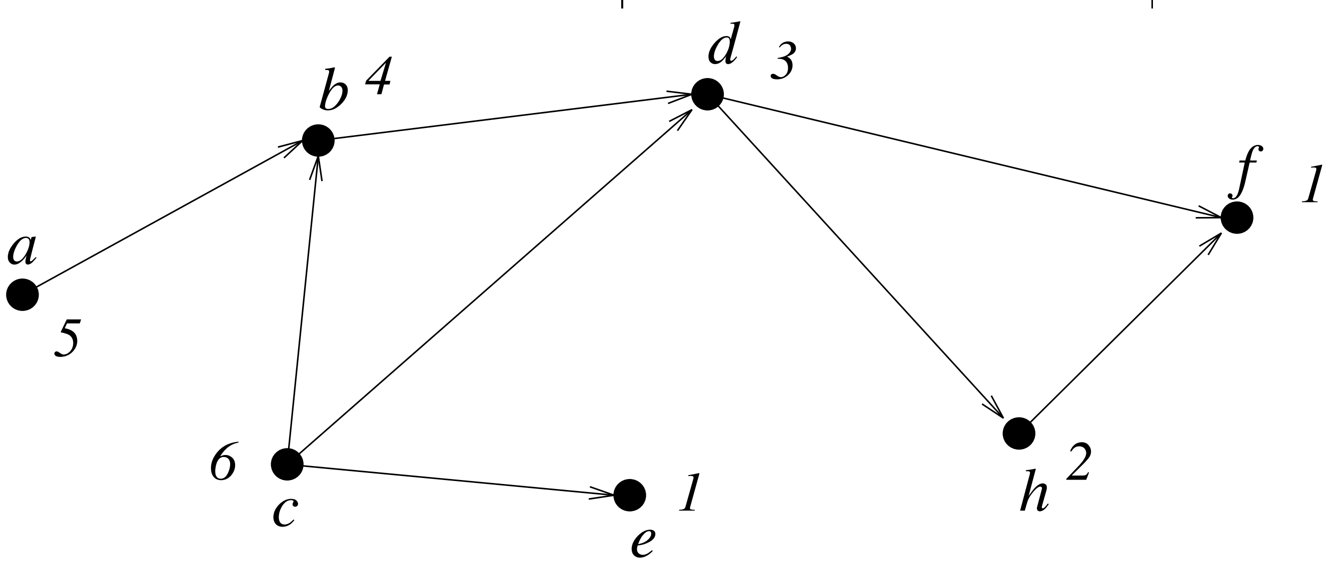
S(a)={a,b,d,f,h} S(c)={b,c,d,e,f,h} S(e)={e}

The transitive closure:

T={(a,b) (a,d) (a,f) (a,h) (b,d) (b,f) (b,h) (c,b) (c,d) (c,e) (c,f) (c,h) (d,f) (d,h) (h,f)}

Size Estimation (Descendant Counting)

- For each node $v \in V$, estimate the size of the reachability set of v (number of descendants) $|S(v)| = |\{u \in V | v \leadsto u\}|$
- Estimate the number of pairs in the transitive closure. $\left|(u, v) \in V \times V | u \leadsto v\right|$



Reachability sets:

S(a)={a,b,d,f,h} S(c)={b,c,d,e,f,h} S(e)={e}

The transitive closure: *15*

T={(a,b) (a,d) (a,f) (a,h) (b,d) (b,f) (b,h) (c,b) (c,d) (c,e) (c,f) (c,h) (d,f) (d,h) (h,f)}

Computing Transitive Closure and Reachability

Networks with n nodes, m edges

- Single source reachability:
  Computing the set $S(v)$ for one node $v \in V$.
  $O(m)$ time (e.g., by DFS, BFS).

- Reachability from each of s sources:
  $O(sm)$ time

- Computing the transitive closure (n sources):
  $O(mn)$ time
  Or, in $O(n^{2.38})$ time using fast matrix multiplication [CW].

$\diamond$ Can we estimate the size faster than explicitly computing the reachability sets?

Applications of fast size estimation

Example 1: Databases: [LN90] [LNS90]

- In query optimization, the size can be used:
  - ⋄ to determine the feasibility of a query
  - ⋄ to optimize the order of operations in performing a complex query.
- the size itself might be the answer to the query.

Example 2: Optimizing the order of multiplications in computing a product of sparse matrices.

Optimizing sparse matrix multiplications

Given matrices $A_{n_1 \times n_2}$, $B_{n_2 \times n_3}$, $C_{n_3 \times n_4}$.
Determine the faster way to compute ABC.
$(AB)C$ or $A(BC)$?

Matrices $A_{n_1 \times n_2}$, $B_{n_2 \times n_3}$ can be multiplied in

$$\sum_{i=1}^{n_2} (\#\text{nonzeros in } A_{\bullet i})(\#\text{nonzeros in } B_{i\bullet})$$

operations.

Example:

$$\begin{pmatrix} 4 & 1 & 2 & 0 \\ 0 & 8 & 0 & -2 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 1 & 7 \end{pmatrix} \begin{pmatrix} 5 & 0 & 0 & 0 \\ 9 & 1 & 0 & 0 \\ 0 & 0 & -3 & 4 \\ 0 & 1 & 6 & 7 \end{pmatrix}$$

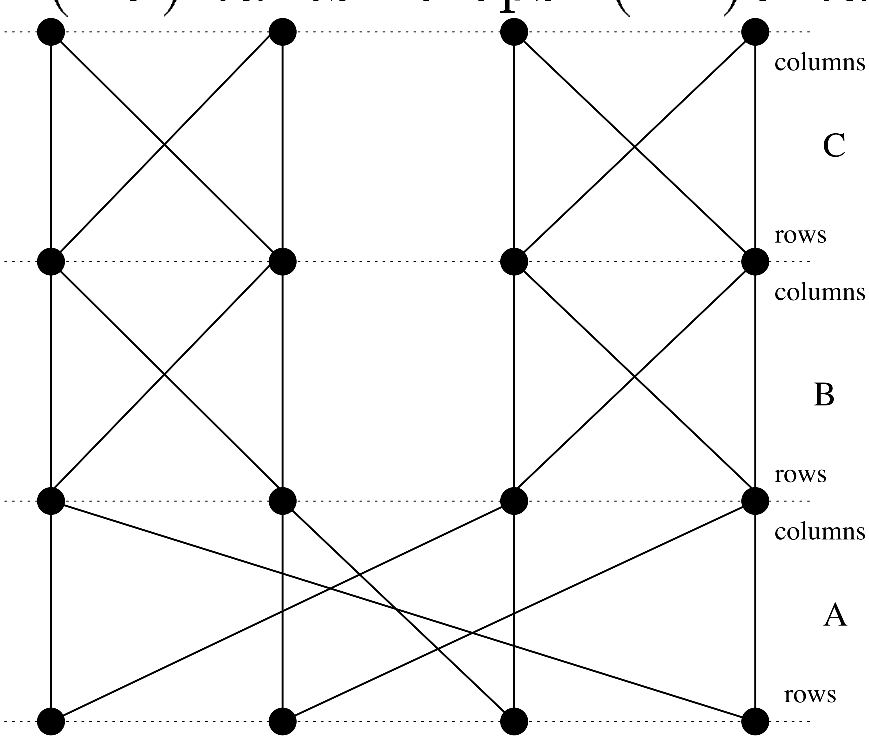
takes $1 \times 1 + 3 \times 2 + 2 \times 2 + 2 \times 3 = 17$ ops.

... Optimizing MM

Example: $ABC =$

$$\begin{pmatrix} 4 & 0 & 2 & 0 \\ 0 & 8 & 0 & -2 \\ 0 & 1 & 7 & 0 \\ 2 & 0 & 0 & 3 \end{pmatrix} \begin{pmatrix} 5 & -2 & 0 & 0 \\ 9 & 1 & 0 & 0 \\ 0 & 0 & -3 & 4 \\ 0 & 0 & 6 & 7 \end{pmatrix} \begin{pmatrix} 6 & -1 & 0 & 0 \\ 4 & 7 & 0 & 0 \\ 0 & 0 & 3 & 2 \\ 0 & 0 & -1 & 6 \end{pmatrix}$$

AB and BC take 16 operations.
The estimation alg. determines how many ops. needed for $A(BC)$and $(AB)C$:
$A(BC)$ takes 16 ops. $(AB)C$ takes 32 ops.



Previous Work

Lipton and Naughton gave an algorithm for estimating the size T of the transitive closure:

1. randomly sample s source nodes
   (s is determined adaptively.)

2. compute the number of descendants for each sampled node.
   D is the total number of descendants

3. Estimate $\hat{T} = nD/s$

Performance: For any fixed $\delta > 0$, $0 < \epsilon \leq 1$,

- time $O(n\sqrt{m})$
- with probability $\geq 1 - \epsilon$, $|T - \hat{T}| \leq \delta T + n(1 + \delta)$

Remarks:

- If $m \gg n$, then $|T - \hat{T}| \leq \delta T$.

- runs in linear time for almost-regular networks (out-degrees of all nodes are within a constant factor of each other)

New estimation algorithm

For any fixed $1 \geq \epsilon > 0$ and $\delta > 0$:

$\diamond$ $O(m)$ time

$\diamond$ computes estimates $\hat{s}(v)$ $(\forall v \in V)$, and $\hat{T}$ s.t.:

- with probability $\geq 1-\epsilon$: $(1-\delta)T \leq \hat{T} \leq (1+\delta)T$
- $Ex(|T - \hat{T}|) \leq \delta T$
- For each $v \in V$, with probability $\geq 1 - \epsilon$: $(1-\delta)|S(v)| \leq \hat{s}(v) \leq (1+\delta)|S(v)|$
- For each $v \in V$, $Ex(||S(v)| - \hat{s}(v)|) \leq \delta |S(v)|$

Improvements:

- Faster, runs in optimal linear time
- Produces better estimates
- Estimates not only the closure size but also the reachability of each node

Asymptotic behavior

For any $1 \geq \epsilon > 0$ and $k > 0$:

$\diamond$ $O(km)$ time

$\diamond$ computes estimates $\hat{s}(v)$ $(\forall v \in V)$, and $\hat{T}$ s.t.:

- with probability $\geq 1 - e^{-O(\epsilon^2 k)}$:
  $(1-\epsilon)T \leq \hat{T} \leq (1+\epsilon)T$

- $Ex(|T - \hat{T}|) \leq T/\sqrt{k}$

- For each $v \in V$,
  with probability $\geq 1 - e^{-O(\epsilon^2 k)}$:
  $(1-\epsilon)|S(v)| \leq \hat{s}(v) \leq (1+\epsilon)|S(v)|$

- $Ex(||S(v)| - \hat{s}(v)|) \leq |S(v)|/\sqrt{k}$

$\diamond$ For $k = O(\epsilon^{-2} \log n)$,
with probability $1 - O(1/\operatorname{poly}(n))$, all estimates are within ϵ.

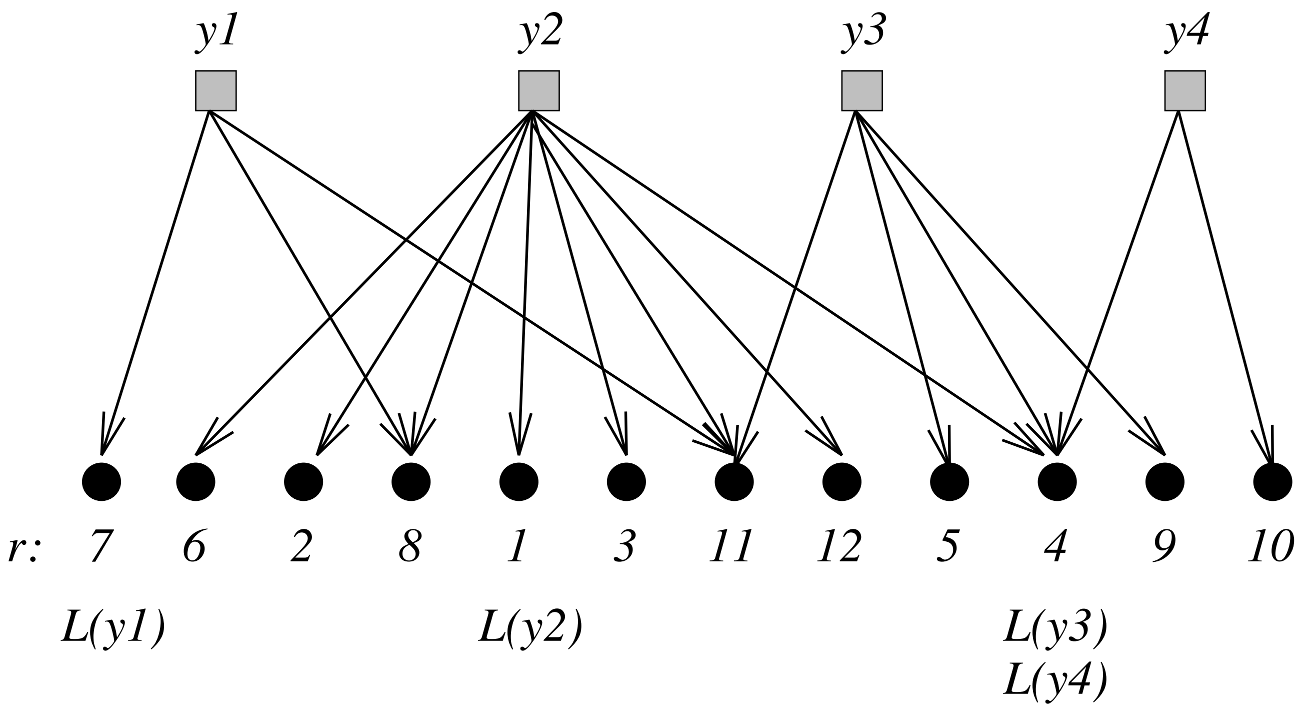
The estimation framework

- Sets Y and X
- $S : Y \rightarrow 2^X$ maps each $y \in Y$ to a subset of X

Goal: compute estimates $\hat{s}(y)$ of $|S(y)|$ $(\forall y \in Y)$

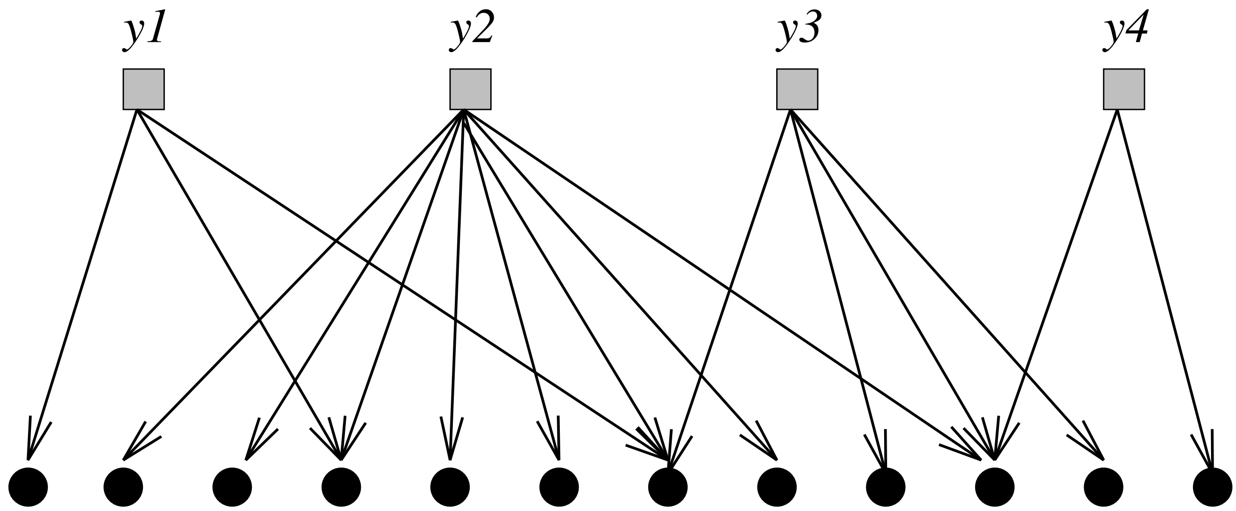
We have access to the following
Least-Element subroutine (LE):
Input: an ordering $r : X \rightarrow \{1, \ldots, |X|\}$ of X,
Output: a mapping $L : Y \rightarrow X$, such that:

- for all $y \in Y$, $L(y) \in S(y)$ and
- $r(L(y)) = \min_{w \in S(y)} r(w)$.



Intuition for use of LE to produce the estimates

- Select ranks $R : X \rightarrow [0,1]$ independently and uniformly at random
- Apply LE with the ordering induced by R
- $R(L(y))$ is the min of $|S(y)|$ values from $U[0,1]$. The expected value of $R(L(y))$ is $\frac{1}{|S(y)|+1}$ Hence, $|S(y)| = \frac{1}{Ex(R(L(y)))} - 1$



|S(y1)|=3 |S(y2)|=8 |S(y3)|=4 |S(y4)|=2

We expect *R(L(y4))* to be large and *R(L(y2))* to be small

The estimation algorithm

Repeat for k iterations ($1 \leq i \leq k$):

1. Select ranks $R_i : X \rightarrow [0,1]$, independently and uniformly at random.

2. Apply LE with the ordering induced by R_i. $L_i : Y \rightarrow X$ is the mapping returned by LE.

For each $y \in Y$:
$\hat{Ex}(y) \leftarrow \frac{\Sigma_{1 \leq i \leq k} R_i(L_i(y))}{k}$
(estimator for the expected value of $R(L(y))$)
$\hat{s}(y) \leftarrow \frac{1}{\hat{Ex}(y)} - 1$
(estimator for $|S(y)|$)

The quality of the estimates increases with k: For larger k (number of iterations), we get a better estimator for the expected minimum rank, and hence a better estimator for $|S(y)|$.

Quality of the estimates

- The running time amounts to $O(k)$ applications of the LE subroutine.

- The estimates $\hat{s}(y)$ (for $y \in Y$) are such that:

  1. For any $\epsilon > 0$, for all $y \in Y$,

$$\text{Prob}\{||S(y)| - \hat{s}(y)| \geq \epsilon|S(y)|\} = e^{-O(\epsilon^2 k)}$$

  2. For all $y \in Y$,

$$Ex(||S(y)| - \hat{s}(y)|/|S(y)|) = O(1/\sqrt{k})$$

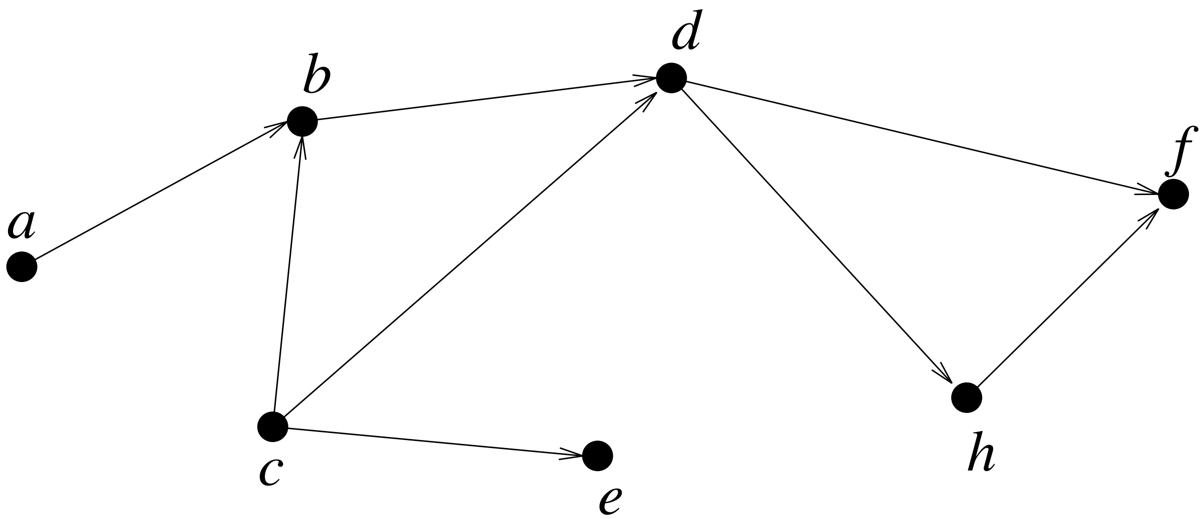
- $\hat{T} \leftarrow \Sigma_{y \in Y} \hat{s}(y)$ is such that:

  1. $Ex(|\hat{T} - T|) = O(T/\sqrt{k})$
  2. $\text{Prob}\{|\hat{T} - T| \geq \epsilon T\} = e^{-O(\epsilon^2 k)}$

Estimating Reachability

$O(m)$ time Least-Element algorithm:

- Assume $r(v_1) < \cdots < r(v_n)$.
  Reverse edge directions. Iterate until $V = \emptyset$:
- $i \leftarrow \min\{j | v_j \in V\}$.
  $V_i \leftarrow \{u \in V | v_i \leadsto u\}$
  For every $u \in V_i$, $L(u) \leftarrow v_i$.
  $V \leftarrow V \setminus V_i$, $E \leftarrow E \setminus V_i \times V_i$.



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}

For the order: *r(e)<r(b)<r(d)<r(a)<r(c)<r(f)<r(h)*

Least-elements are:

L(a)=b L(b)=b L(c)=e L(d)=d L(e)=e L(f)=f L(h)=f

Reachability size estimation in parallel

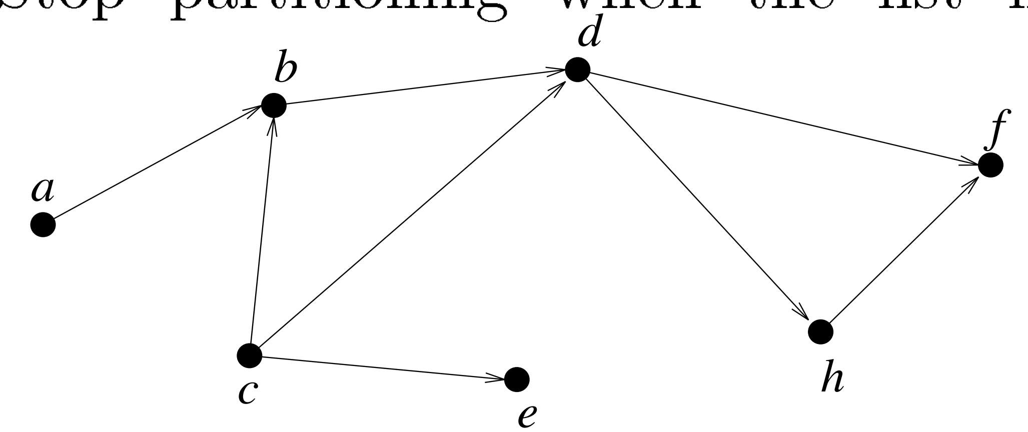
Previous work: For planar graphs, Kao and Klein gave a polylog-time linear-work reduction of descendent counting to single-source reachability (SSR).

New: A parallel algorithm for the least-element problem. The algorithm computes Least-Elements within the time and work bounds of performing $O(\log n)$ SSR computations.
Hence, (apprx.) descendent counting on general graphs has a polylog-time linear-work reduction to SSR.

Known polylog time SSR reachability algorithms are work-intensive ($\Omega(n^{2.38})$ or $\Omega(m^2)$ [KS]). However, SSR and hence size estimation can be solved efficiently when we allow more time or focus on restricted families of graphs.

Computing Least-Elements in parallel

Divide and conquer approach: Maintain a partition to subgraphs. For each subgraph keep a sublist of possible least reachable-nodes. Stop partitioning when the list has size 1.



For the order: *r(e)<r(b)<r(d)<r(a)<r(c)<r(f)<r(h)*

Least-elements are: *L(a)=b L(b)=b L(c)=e L(d)=d L(e)=e L(f)=f L(h)=f*

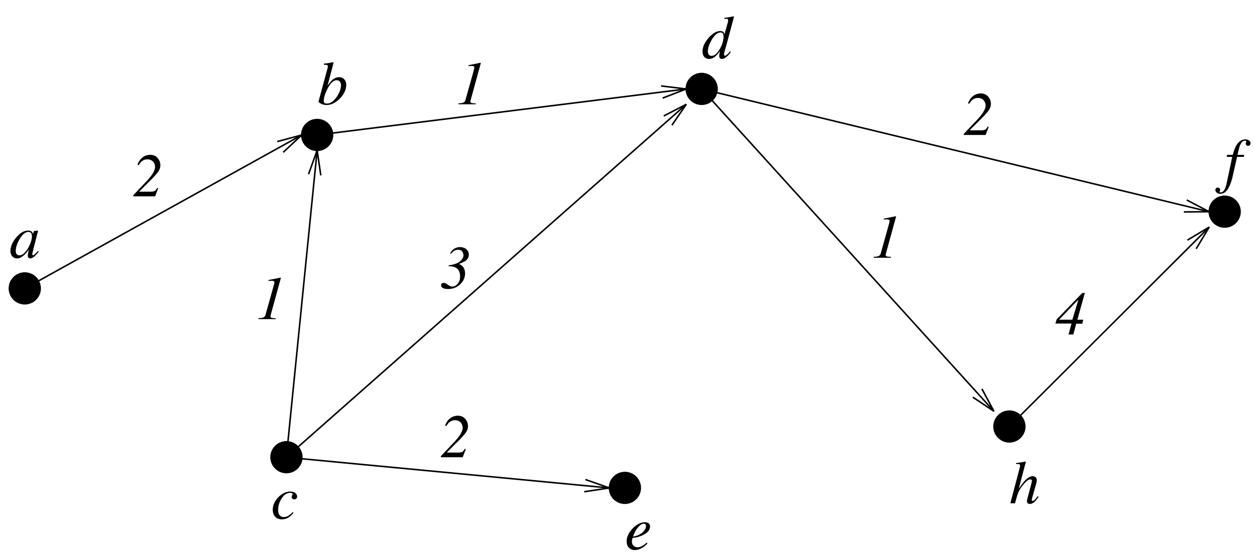
1. Reverse all edges
   One subgraph $G = (V, E)$, list $\leftarrow V$.
2. For $O(\log n)$ phases repeat:
   For each subgraph $G' = (V', E')$:
   - Create a supernode s with edges to the half lowest ranked nodes on the list of G'.
   - Compute SSR from s to reach $\hat{V}$.
     Partition G' to the subgraphs induced.

Estimating neighborhood sizes in weighted graphs

- Directed Network $G = (V, E)$
  positive weights $w : E \to R_+$

- For a pair $v \in V$, $r \in R_+$,
  $N(v, r)$ is the r-neighborhood of v
  (all nodes of distance $\leq r$ from v)

Goal: For query pairs (v, r) $v \in V$, $r \in R$, estimate $|N(v, r)|$



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}

Bounds for estimating neighborhoods' sizes

Previously, to estimate neighborhood sizes we had to compute them explicitly.
The fastest known methods to compute neighborhoods of s nodes is by using Dijkstra's shortest paths algorithm.
The resulting running time is $O(s(m + n \log n))$.

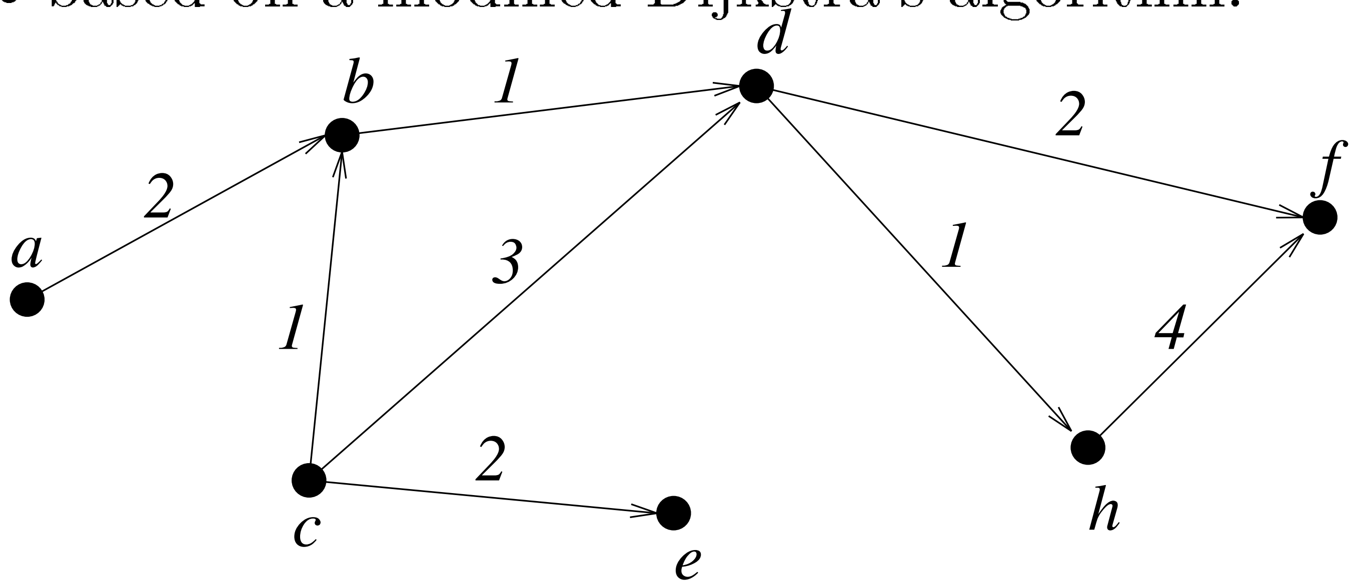
New results:
For any $\delta > 0$, $1 \geq \epsilon > 0$,
after a $O(m \log n + n \log^2 n)$ expected time preprocessing step, we can do as follows:
For each query pair (v, r) we can produce,
in $O(\log \log n)$ expected time,
an estimate $\hat{n}(v, r)$ such that

1. $\mathrm{Prob}\{||N(v,r)| - \hat{n}(v,r)| \geq \delta |N(v,r)|\} \leq 1 - \epsilon$

2. $E(||N(v,r)| - \hat{n}(v,r)|/|N(v,r)|) \leq \delta$

Estimation algorithm for neighborhood sizes

- The LE alg. produces a list for every node.
- requires random order to be efficient.
- based on a modified Dijkstra's algorithm.



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}

For the order: *r(e)<r(b)<r(d)<r(a)<r(c)<r(f)<r(h)*

Least-element lists:

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)

Algorithm for least-element lists

For each $v \in V$, initialize $\ell(v) \leftarrow (0, v)$.

- Assume $r(v_1) < \cdots < r(v_n)$.
  Reverse edge directions.
  Iterate:
- $i \leftarrow \min\{j | v_j \in V\}$.
  Run modified Dijkstra from v_i:
  For each visited node u at distance D do $\ell(u) \leftarrow \ell(u) \cup (v_i, D)$.
  Stop search at nodes v where
  $\exists (v_j, d) \in \ell(v)$ s.t. $d <$ current distance.

Running time:

- Total number of visits (sum of sizes of lists) of all nodes.

- Can be n visits for worst case rankings.

- Since ranks are random, expected number of visits (size of lists) is $O(\log n)$.

More applications

- A new TC algorithm: In each iteration we compute one random descendant for each node. After $O(k \log n)$ with h.p. we compute all reachability sets of size at most k.

- The estimation procedure associates with each node a k-vector (ranks of least-elements in k iterations). These vectors can be used to estimate:

  ◇ whether two nodes $\{v, u\}$ are such that $|S(v) \cap S(u)| \geq \alpha |S(v) \cup S(u)|$

  ◇ number of elements reachable from a subset of nodes $U \subset V$

- Estimating sizes on-line

On-line estimation of weights of growing sets

- X: a set of elements with weights $w : X \to R_+$
- Y: a collection of subsets of X

Operations:

1. Create a new subset $y' \in Y$ init. to $\emptyset, y \in Y$

2. Add a new $x \in X$ to some subsets.

3. Merge two subsets $\{y, y'\} \subset Y$ $(y \leftarrow y \cup y')$.

4. Weight-query: For a $y \in Y$, estimate $w(y)$.

Can be supported with constant/logarithmic time per operation.
Appl: Keeping counts of preceding events in a distributed system.

Open Problems

- ? find more applications to the estimation scheme

- ? better TC algorithms