Faster Fully Dynamic Transitive Closure in Practice

Kathrin Hanauer Monika Henzinger Christian Schulz

universität wien

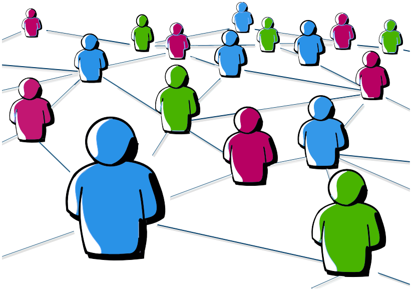
June 17, 2020

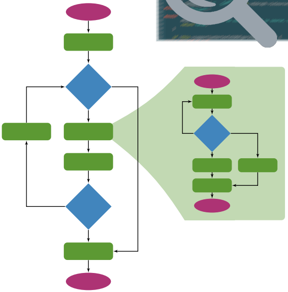
kathrin.hanauer@univie.ac.at

All-Pairs Reachability

a.k.a.

Transitive Closure





Fully Dynamic Transitive Closure

directed graph +
sequence of operations:

queries $s \overset{?}{\rightsquigarrow} t$
edge insertions & deletions

Query & Update Times (in $\mathcal{O}$)

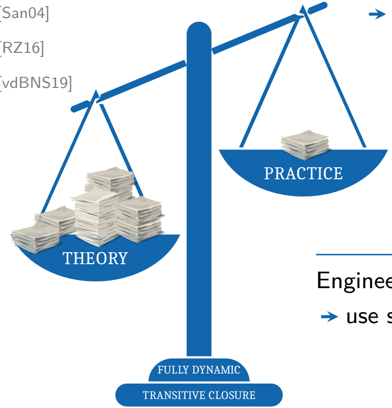
m	1	static graph traversal
1	n^2	[DI08, Rod08, San04]
$\sqrt{n}$	$m\sqrt{n}$	[RZ08]
$m^{0.43}$	$m^{0.58}n$	[RZ08]
$n^{0.58}$	$n^{1.58}$	[San04]
$n^{1.495}$	$n^{1.495}$	[San04]
n	$m + n\log n$	[RZ16]
$n^{1.407}$	$n^{1.407}$	[vdBNS19]

2 Very Large Studies [FMNZ01, KZ08]

➔ Distinctly fastest on most instances: static graph traversal algorithms

➔ Strongest competitors: two SCC-maintaining algorithms

➔ Few "real-world" graphs



Our Idea

Engineering algorithms that ...

➔ use single-source reachability

➔ don't maintain SCCs

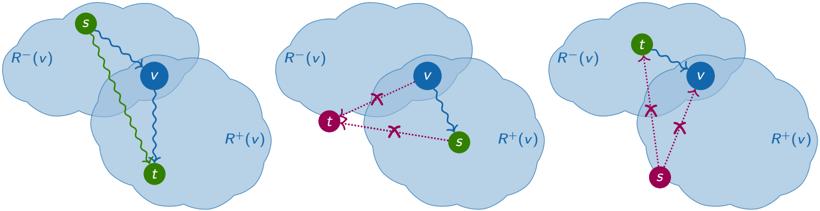
➔ profit from SCCs

Observations

Let v, s, t be vertices.

$R^+(v)/R^-(v)$: vertices reachable from/that can reach v

Consider query $s \overset{?}{\rightsquigarrow} t$:



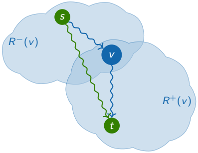
(O1) (O2) (O3)

v is a supportive vertex: $R^+(v)/R^-(v)$ *can* help to answer $s \overset{?}{\rightsquigarrow} t$

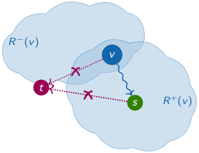
General Outline

- Store list of supportive vertices $\mathcal{L}_{SV}$
  $\forall v \in \mathcal{L}_{SV}$: maintain $R^+(v)$ and $R^-(v)$ via SSR; algorithms
- Updates (edge insertions & deletions):
  forward to SSR algorithms
- Query:
  $\forall v \in \mathcal{L}_{SV}$: try to answer via (O1), (O2), (O3)
  fallback to static graph traversal

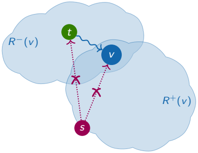
single-source/
single-sink
reachability



(O1)



(O2)



(O3)

Supportive Vertices Algorithms

$\texttt{SV}(k)$	$\texttt{SVA}(k, c)$ with adjustments	$\texttt{SVC}(z, c)$ with SCC cover

Initialization:

$\texttt{SV}(k)$ / $\texttt{SVA}(k, c)$	$\texttt{SVC}(z, c)$
pick k vertices as supportive vertices uniformly at random	compute SCCs $\{S_0, \ldots, S_\ell\}$ if $\lvert S_i \rvert \geq z$: pick supportive vertex for S_i as representative map: vertex $\rightarrow$ representative

Updates:

$\texttt{SVA}(k, c)$ / $\texttt{SVC}(z, c)$
re-initialize every c updates

Queries:

$\texttt{SV}(k)$ / $\texttt{SVA}(k, c)$	$\texttt{SVC}(z, c)$
try supportive vertices in order of $\mathcal{L}_{\mathsf{SV}}$ fallback: static graph traversal	lookup & use representatives remove invalid entries from map fallback: mode of $\texttt{SV}/\texttt{SVA}$

Single-Source Reachability Subalgorithms

Extended Simple Incremental Algorithm (SI)

Maintains reachability tree:

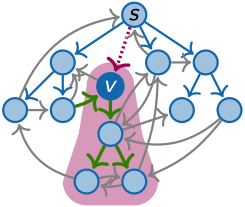
Insertions:

extend tree via BFS

Deletions:

reconstruct tree via backward/forward BFS

Queries: $\mathcal{O}(1)$ time



Simplified Extended Even-Shiloach Trees (SES)

Maintains BFS tree:

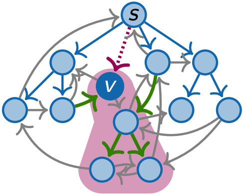
Insertions:

update BFS levels

Deletions:

simplified ES tree routine

Queries: $\mathcal{O}(1)$ time



All algorithms implemented in C++17 as part of the open-source algorithms library Algora.

Code available publicly on Gitlab & Github:

libAlgora
libAlgora

Algorithms

- BFS, DFS, DBFS (DFS-BFS hybrid)
- BiBFS (bidirectional BFS)
- SV with $k = 1$, $k = 2$, $k = 3$ ⋆
- SVA with $k = 1$ and $c = 1\text{k}$, $c = 10\text{k}$, $c = 100\text{k}$ ⋆
- SVC with $z = 25$ or $z = 50$ and $c = 10\text{k}$, $c = 100\text{k}$ ⋆

⋆ Fallback: BiBFS; SSR algorithms: SES, SI [HHS20]

Experiments: Instances

Random dynamic instances

ER graphs:

$n = 100\mathrm{k}$ and $n = 10\mathrm{m}$, $m_{\mathsf{init}} = d \cdot n$, $d \in [1.25 \ldots 50]$

$\sigma = 100\mathrm{k}$, different ratios of insertions/deletions/queries

Stochastic Kronecker graphs with random update sequences:

$n \approx 130\mathrm{k}$ and $n \approx 30 \ldots 130\mathrm{k}$, $m_{\mathsf{avg}} = d \cdot n$, $d = 0.7 \ldots 16.5$

$\sigma_{\pm} = 1.6\mathrm{m} \ldots 702\mathrm{m}$ and $\sigma_{\pm} = 282\mathrm{k} \ldots 82\mathrm{m}$ (updates only)

Real-world dynamic instances

. . . with real-world update sequences:

$n = 100\mathrm{k} \ldots 2.2\mathrm{m}$, $m_{\mathsf{avg}} = d \cdot n$, $d = 5.4 \ldots 7.8$

$\sigma_{\pm} = 1.6\mathrm{m} \ldots 86.2\mathrm{m}$ (updates only)

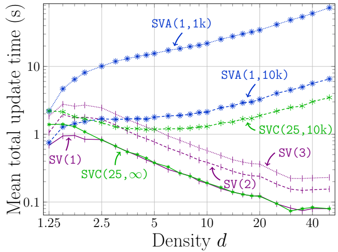
. . . with randomized update sequences:

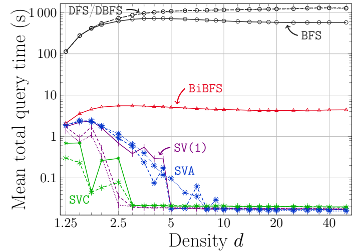
$n = 31\mathrm{k} \ldots 2.2\mathrm{m}$, $m_{\mathsf{avg}} = d \cdot n$, $d = 4.7 \ldots 10.4$

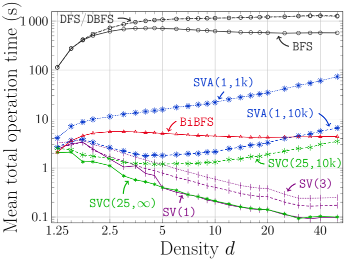
$\sigma_{\pm} = 1.4\mathrm{m} \ldots 76.4\mathrm{m}$ (updates only)

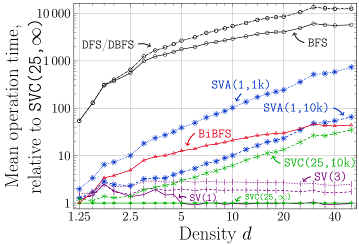
Experiments: Random Instances

$n = \sigma = 100\text{k}, \; \rho_{IDQ} = 1 : 1 : 1$



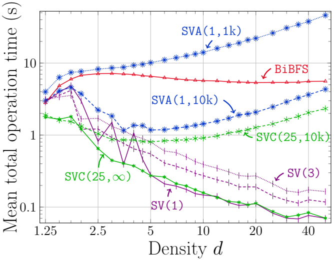


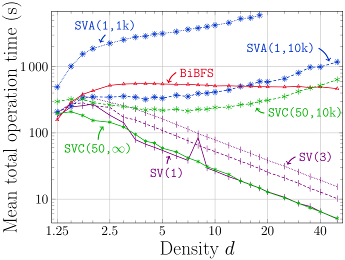


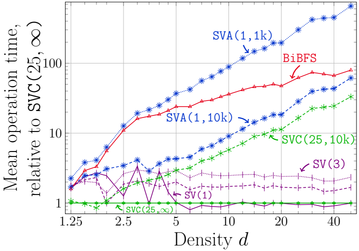


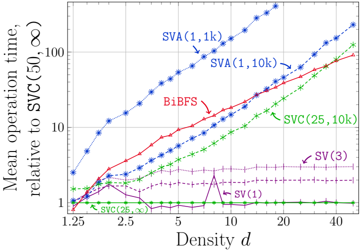
Experiments: Random Instances

$n = \sigma = 100\text{k}$, $\rho_{IDQ} = 1:1:2$ | $n = 10\text{m}$, $\sigma = 100\text{k}$, $\rho_{IDQ} = 1:1:1$



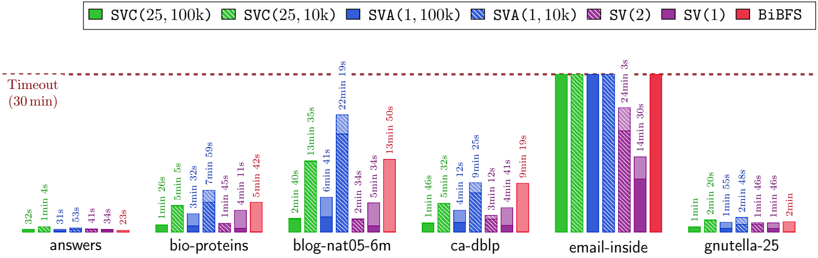






Experiments: Kronecker Instances

$n \approx 130\text{k},\ \sigma_\pm = 1.6\text{m} \ldots 702\text{m},\ \rho_{UQ} = 2:1$



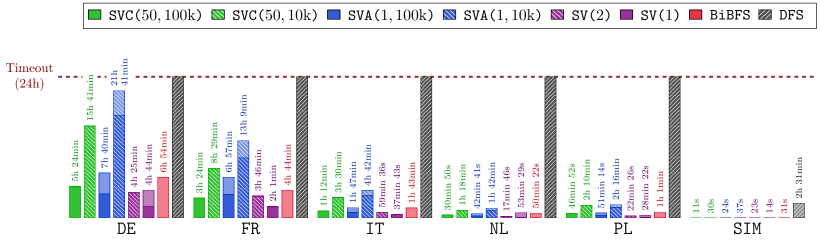
Fastest: `SV(1)`, `SV(2)`, `SVC(25,100k)`

`BFS`, `DFS`, `DBFS`: $> 6\,\text{h}$ on $\geq 13/20$ instances

similar picture for $n \approx 30 \ldots 130\text{k}$

Experiments: Real-World Instances

$n = 31\text{k} \ldots 2.2\text{m}$, $\sigma_{\pm} = 1.6\text{m} \ldots 86.2\text{m}$, $\rho_{UQ} = 2:1$

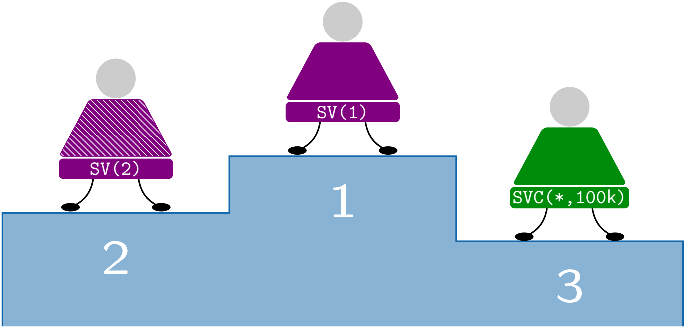


Fastest: SV(1), SV(2)

BFS, DFS, DBFS: ≈ 6 % in 24 h on DE instance

similar picture on set with randomized updates

Conclusion



+ more stable query time

– doubled update time

+ more stable query time

+ fast on sparse instances

– – *considerably* increased update time

?? recompute less often

Slower by *several orders of magnitude*: `BFS`, `DFS`, `DBFS`, `BiBFS`

Thank you! – Questions?

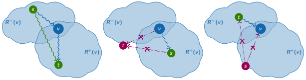
Supportive Vertices

Observations

Let v, s, t be vertices.

$R^+(v)/R^-(v)$: vertices reachable from/that can reach v

Consider query $s \overset{?}{\rightsquigarrow} t$:

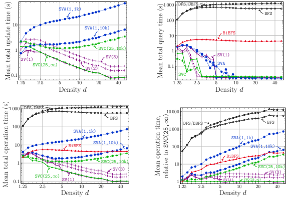


(O1) (O2) (O3)

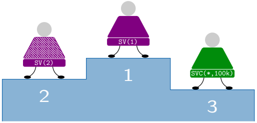
v is a supportive vertex: $R^+(v)/R^-(v)$ *can* help to answer $s \overset{?}{\rightsquigarrow} t$

Experiments: Random Instances

$n = \sigma = 100\text{k}$, $\rho_{IDQ} = 1:1:1$



Conclusion



+ more stable query time
− doubled update time

+ more stable query time
+ fast on sparse instances
− − *considerably* increased update time
?? recompute less often

Slower by *several orders of magnitude*: BFS, DFS, DBFS, BiBFS

Dynamic instances & source code:

https://dyreach.taa.univie.ac.at/transitive-closure

Algora

libAlgora libAlgora

Bibliography I

[DI08] C. Demetrescu and G. F. Italiano. Mantaining dynamic matrices for fully dynamic transitive closure. *Algorithmica*, 51(4):387–427, 2008.

[FMNZ01] D. Frigioni, T. Miller, U. Nanni, and C. Zaroliagis. An experimental study of dynamic algorithms for transitive closure. *Journal of Experimental Algorithmics (JEA)*, 6:9, 2001.

[HHS20] K. Hanauer, M. Henzinger, and C. Schulz. Fully dynamic single-source reachability in practice: An experimental study. In *Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX 2020, Salt Lake City, UT, USA, January 5-6, 2020*, pages 106–119, 2020.

[KZ08] I. Krommidas and C. D. Zaroliagis. An experimental study of algorithms for fully dynamic transitive closure. *ACM Journal of Experimental Algorithmics*, 12:1.6:1–1.6:22, 2008.

[Rod08] L. Roditty. A faster and simpler fully dynamic transitive closure. *ACM Trans. Algorithms*, 4(1), March 2008.

Bibliography II

[RZ08] L. Roditty and U. Zwick. Improved dynamic reachability algorithms for directed graphs. *SIAM Journal on Computing*, 37(5):1455–1471, 2008.

[RZ16] L. Roditty and U. Zwick. A fully dynamic reachability algorithm for directed graphs with an almost linear update time. *SIAM Journal on Computing*, 45(3):712–733, 2016.

[San04] P. Sankowski. Dynamic transitive closure via dynamic matrix inverse. In *45th Symposium on Foundations of Computer Science (FOCS)*, pages 509–517. IEEE, 2004.

[vdBNS19] J. van den Brand, D. Nanongkai, and T. Saranurak. Dynamic matrix inverse: Improved algorithms and matching conditional lower bounds. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 456–480, 2019.