Fully Dynamic Reachability – in Practice!

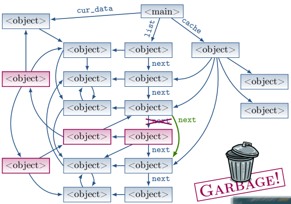
Kathrin Hanauer

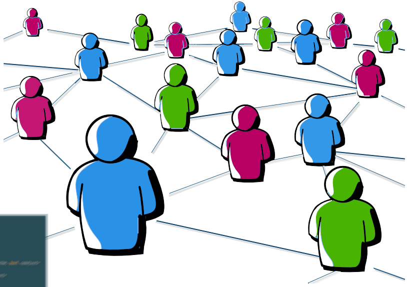
joint work with Monika Henzinger and Christian Schulz

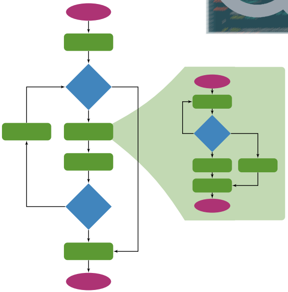
universität wien

November 24, 2021

kathrin.hanauer@univie.ac.at







Fully Dynamic Reachability

directed graph +
sequence of operations:

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

single-source reachability: s fixed

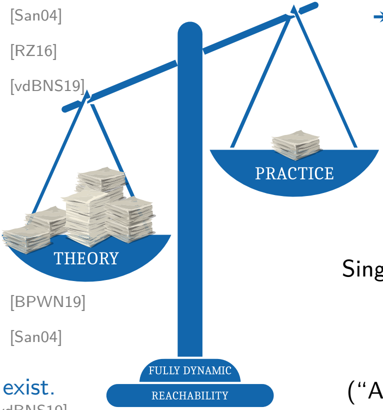
Query & Update Times (in $\mathcal{O}$)

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]

Single Source:

inc.	1	
dec.	$\log^4 n$	[BPWN19]
fully-dyn.	$n^{1.575}$	[San04]

Conditional lower bounds exist.
[DHZ00, WW10, AW14, HKNS15, vdBNS19]



2 Very Large Studies [FMNZ01, KZ08]

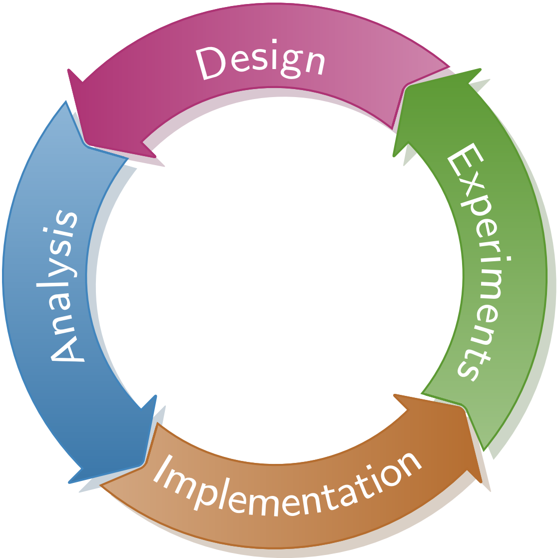
➔ Distinctly fastest on most instances:
static graph traversal algorithms

➔ Strongest competitors:
two SCC-maintaining algorithms

➔ Few "real-world" graphs
➔ All-pairs only

This Talk
Algorithms for
Single-Source Reachability
↓
Algorithms for
Transitive Closure
("All-Pairs Reachability")

Algorithm Engineering



Single-Source Reachability

Algorithms for Single-Source Reachability

Group I : "Dynamized" static algorithms

Group II : Dynamic maintenance of a reachability tree

Group III : Dynamic maintenance of a *BFS* tree
(➔ reachability tree with *minimal* vertex depths)

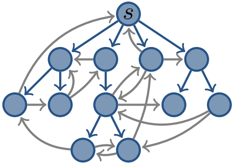
Features:

Reachability proof: Algorithms can return path (upon request)

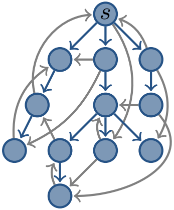
Concentrate on deterministic or Las Vegas-style randomized algorithms

Algorithms: Dynamized Static Algorithms

BFS



DFS



Three flavors:

`BFS`	Static:	Pure static algorithm, called for each QUERY.	`DFS`
`CBFS`	Caching:	Cache reachability of *all* vertices, recompute *entirely* upon QUERY if necessary.	`CDFS`
`LBFS`	Lazy:	Cache only reachability of vertices encountered during a QUERY, resume traversal if cache is still *valid*.	`LDFS`

Algorithms: Maintenance of Reachability Tree $\mathcal{T}$

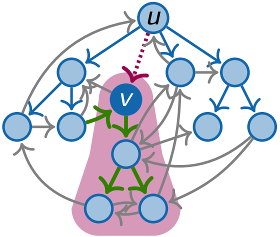
Extended Simple Incremental algorithm (`SI`):

$\textsc{Query}(v)$

Maintain for each vertex: v `treeEdge: <edge>/null`

$\textsc{initialize}()$, $\textsc{edgeInserted}((u, v))$: build/extend $\mathcal{T}$ via BFS

$\textsc{edgeDeleted}(e = (u, v))$:



If v.`treeEdge` $= e$:

- $\mathcal{L}$ too large? ➔ recompute from scratch
- Reconstruct ➔ use backward BFS
- Optional: additionally use forward BFS

reverse $\mathcal{L}$ forward BFS

➔ Algorithm: `SI(R?/SF?/`ρ`)`

threshold: $|\mathcal{L}| \leq \rho \cdot n$

Algorithms: Maintenance of BFS Tree

Extended Even-Shiloach trees (ES):

$\textsc{Query}(v)$

Maintain for each vertex: v

level: depth(v) $\vee\ \infty$

inEdges: <list of in-edges>

treeEdge: <index in inEdges>

minimum level, minimum index

v.inEdges: | e_0 | e_1 | e_2 | e_3 | e_4 | e_5 | e_6 |

tails' levels $>$ e_3's

v.treeEdge

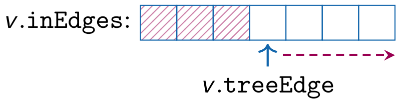
$\textsc{Initialize}()$, $\textsc{edgeInserted}((u, v))$: build/update $\mathcal{T}$ via BFS

$\textsc{edgeDeleted}(e = (u, v))$:

If e is tree edge: FIFO queue $\mathcal{Q} = \langle v \rangle$; $\textsc{process}(\mathcal{Q})$;

Algorithms: Maintenance of BFS Tree $\mathcal{T}$

PROCESS$(\mathcal{Q} = \langle v, \ldots \rangle)$:



- → `v.level += 1`
- → re-enqueue v and children

Thresholds:

#re-enqueuings per vertex $> \beta$ ↘ abort update and
total #vertices processed $> \rho \cdot n$ ↗ recompute $\mathcal{T}$ from scratch

➔ Algorithm: ES(β/ρ)

Variants:

Multi-Level: Scan v.inEdges completely, re-enqueue only children.

➔ Algorithm: MES(β/ρ)

Simplified: Abandon v.inEdges, scan in-edges in arbitrary order.

➔ Algorithm: SES(β/ρ)

Experiments

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, CBFS, LBFS, DFS, CDFS, LDFS
- SI with $(\texttt{R?}/\texttt{SF?}/\rho) = (\overline{\texttt{R}}/\texttt{SF}/.25), (\overline{\texttt{R}}/\overline{\texttt{SF}}/.25)$
- ES, MES, SES with $(\beta/\rho) = (5/.5), (100/1), (\infty, \infty)$

Random dynamic instances

ER graphs:

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

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

Stochastic Kronecker graphs with random update sequences:

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

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

Real-world dynamic instances

. . . with real-world update sequences:

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

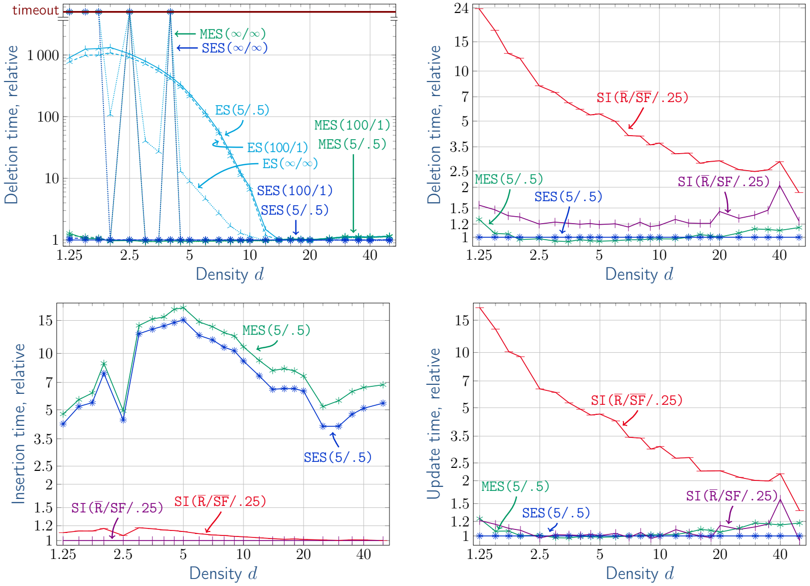
$\sigma_{\pm} = 1.6\text{m} \ldots 86.2\text{m}$ (updates only)

. . . with randomized update sequences:

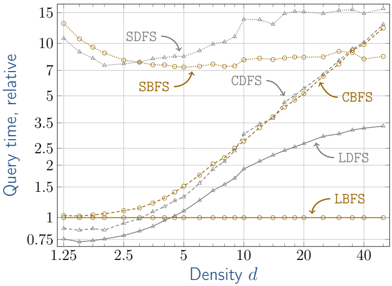
$n = 31\text{k} \ldots 2.2\text{m}$, $m_{\text{avg}} = d \cdot n$, $d = 4.7 \ldots 10.4$

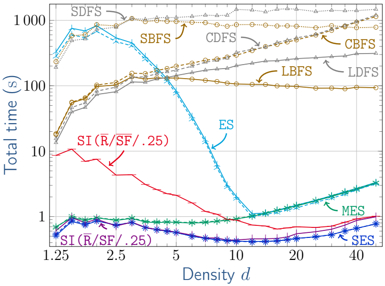
$\sigma_{\pm} = 1.4\text{m} \ldots 76.4\text{m}$ (updates only)

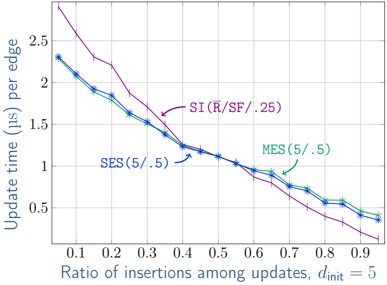
Experiments: Random Instances, $n = 100\text{k}$

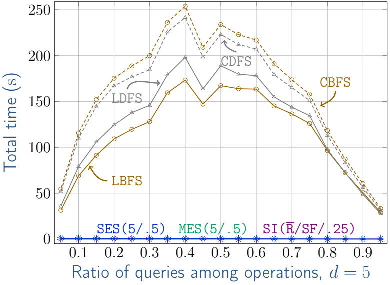


Experiments: Random Instances, $n = 100\text{k}$

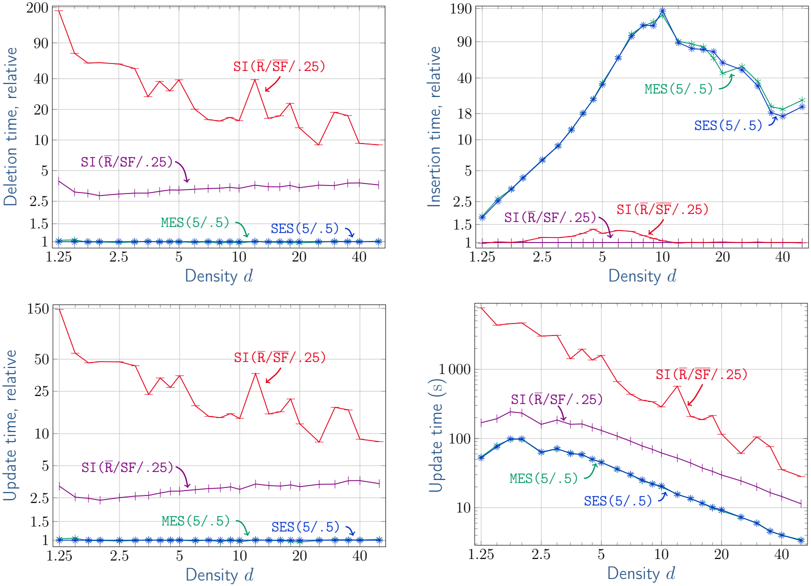




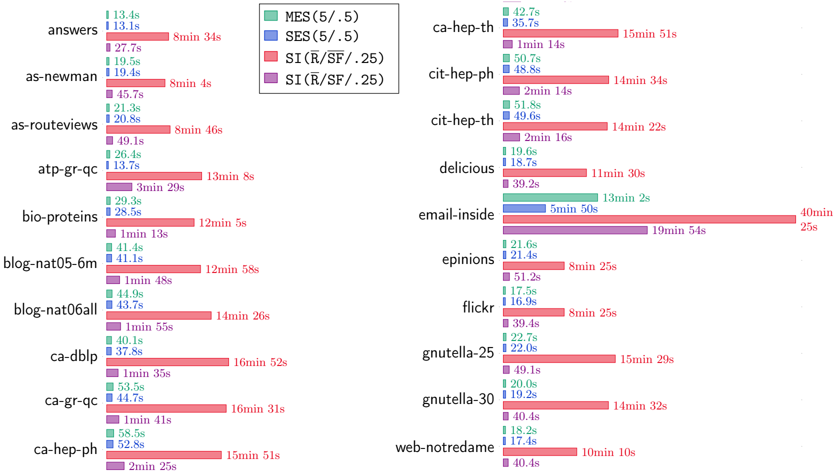




Experiments: Random Instances, $n = 10\mathrm{m}$



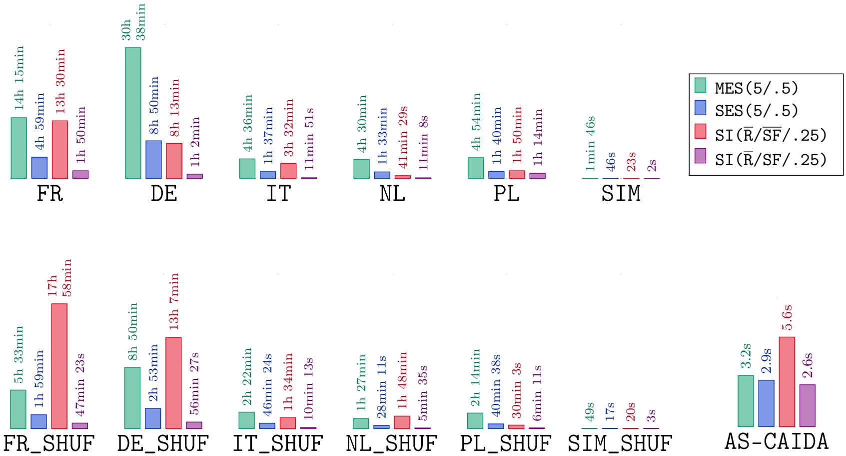
Experiments: Kronecker Instances, $n \approx 130k$



$\approx$ 50% insertions among updates –

$\geq$ 71% of update time spent on deletions (except email-inside, 51%)

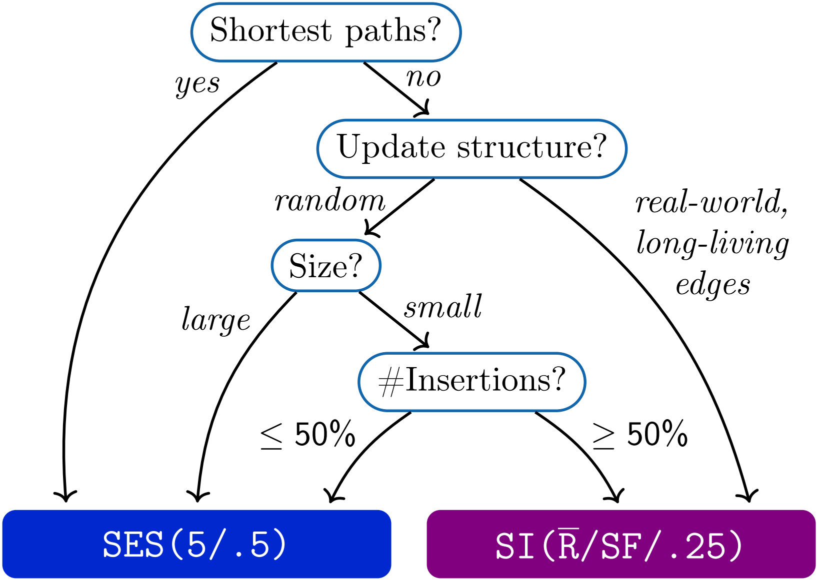
Experiments: Real-World Instances, $n = 31\mathrm{k} \ldots 2.2\mathrm{m}$



51 – 85% insertions among updates –

$>$ 89% of update time spent on deletions

Which algorithm is best?



SSR Algorithms: Overview and Time Complexities

Algorithm	Insertion	Deletion	Query
`BFS`, `DFS`	0	0	$\mathcal{O}(n+m)$
`CBFS`, `CDFS`, `LBFS`, `LDFS`	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(n+m)$
`SI(R?/SF?/`ρ`)` ∟ $\rho = 0$	$\mathcal{O}(n+m)$	$\mathcal{O}(n \cdot m)$ $\mathcal{O}(n+m)$	$\mathcal{O}(1)$
`ES(`β/ρ`)`, `MES(`β/ρ`)` ∟ $\beta \in \mathcal{O}(1) \vee \rho = 0$	$\mathcal{O}(n+m)$	$\mathcal{O}(n \cdot m)$ $\mathcal{O}(n+m)$	$\mathcal{O}(1)$
`SES(`β/ρ`)` ∟ $\beta \in \mathcal{O}(1) \vee \rho = 0$	$\mathcal{O}(n+m)$	$\mathcal{O}(n \cdot m)$ $\mathcal{O}(n+m)$	$\mathcal{O}(1)$

Bibliography I

[AW14] A. Abboud and V. V. Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Proceedings of the 2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, FOCS '14, pages 434–443. IEEE, 2014.

[BPWN19] A. Bernstein, M. Probst, and C. Wulff-Nilsen. Decremental strongly-connected components and single-source reachability in near-linear time. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing*, STOC '19, 2019.

[DHZ00] D. Dor, S. Halperin, and U. Zwick. All-pairs almost shortest paths. *SIAM Journal on Computing*, 29(5):1740–1759, 2000.

[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.

Bibliography II

[HKNS15] M. Henzinger, S. Krinninger, D. Nanongkai, and T. Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *47th ACM Symposium on Theory of Computing*, STOC'15, pages 21–30. ACM, 2015.

[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.

[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.

Bibliography III

[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.

[WW10] V. V. Williams and R. Williams. Subcubic equivalences between path, matrix and triangle problems. In *51st Symposium on Foundations of Computer Science (FOCS)*, pages 645–654, 2010.