Parallel Reachability in Square Root Depth

Yang P. Liu
Stanford University

Joint work with Arun Jambulapati, Aaron Sidford
Stanford University

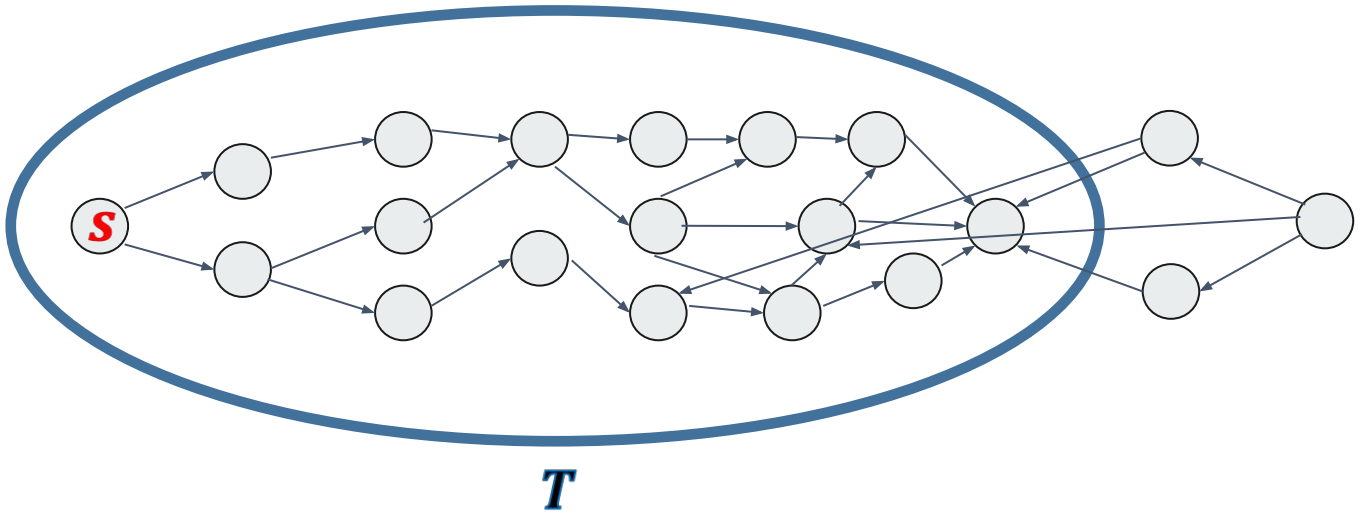
Reachability Problem in Directed Graphs

Reachability Problem in Directed Graphs

Input: directed graph G = (V, E) and source vertex s.
Output: vertices T that are reachable from s in G.

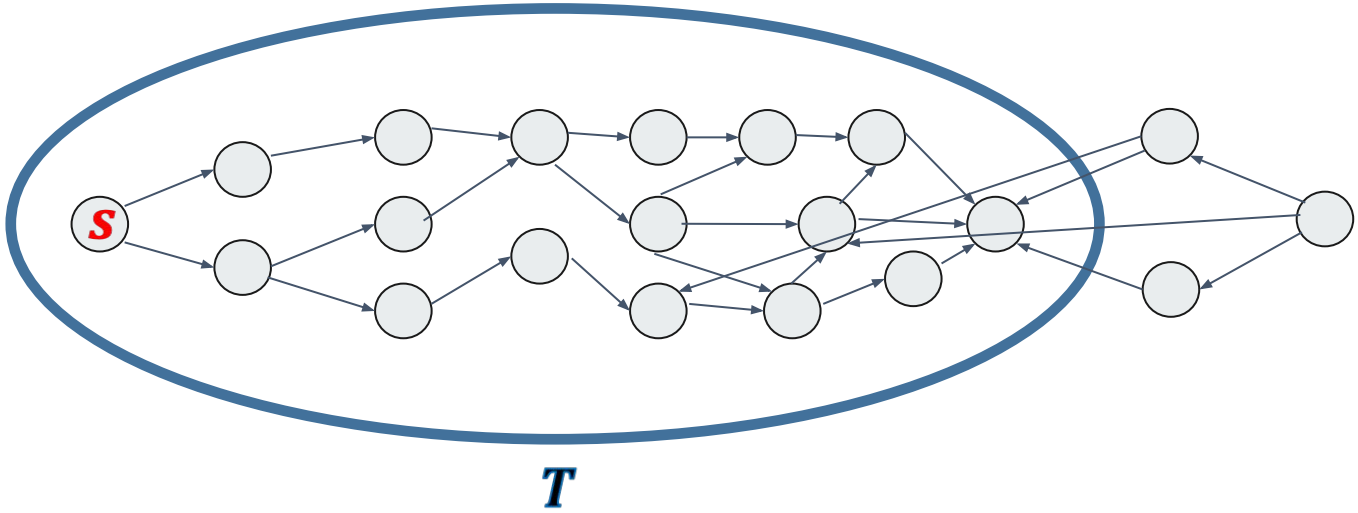
Reachability Problem in Directed Graphs

Input: directed graph G = (V, E) and source vertex s.
Output: vertices T that are reachable from s in G.



Reachability Problem in Directed Graphs

Input: directed graph G = (V, E) and source vertex s.
Output: vertices T that are reachable from s in G.
Setting: *parallel* and *distributed* models of computation

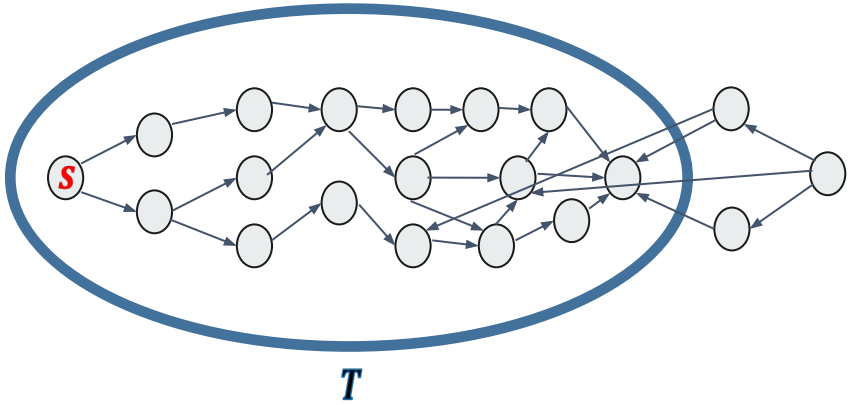


Reachability Problem in Directed Graphs

Input: directed graph G = (V, E) and source vertex s.
Output: vertices T that are reachable from s in G.
Setting: *parallel* and *distributed* models of computation

Why?

- Simplest directed graph problem
- Barrier towards efficient algorithms when edge asymmetry



Work on Parallel Reachability Before 2017

Work on Parallel Reachability Before 2017

Algorithm	**Work**	**Depth**
Parallel BFS	O(m)	O(D)
Parallel Trans. Closure	$O(n^{\omega})$	$\tilde{O}(1)$
Spencer's [Spe97]	$\tilde{O}(m + n\rho^2)$	$\tilde{O}(n/\rho)$
UY [UY91]	$\tilde{O}(m\rho + \rho^4/n)$	$\tilde{O}(n/\rho)$

Work on Parallel Reachability Before 2017

Algorithm	Work	Depth
Parallel BFS	O(m)	O(D)
Parallel Trans. Closure	$O(n^{\omega})$	$\tilde{O}(1)$
Spencer's [Spe97]	$\tilde{O}(m + n\rho^2)$	$\tilde{O}(n/\rho)$
UY [UY91]	$\tilde{O}(m\rho + \rho^4/n)$	$\tilde{O}(n/\rho)$

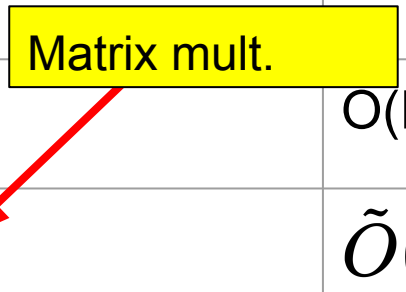
Graph diameter

Work on Parallel Reachability Before 2017

Algorithm	**Work**	**Depth**
Parallel BFS	O(m)	O(D)
Parallel Trans. Closure	$O(n^{\omega})$	$\tilde{O}(1)$
Spencer's [Spe97]	$\tilde{O}(m + n\rho^2)$	$\tilde{O}(n/\rho)$
UY [UY91]	$\tilde{O}(m\rho + \rho^4/n)$	$\tilde{O}(n/\rho)$

Work on Parallel Reachability Before 2017

Algorithm	Work	Depth
Parallel BFS	O(m)	O(D)
Parallel Trans. Closure	$O(n^\omega)$	$\tilde{O}(1)$
Spencer's [Spe97]	$\tilde{O}(m + n\rho^2)$	$\tilde{O}(n/\rho)$
UY [UY91]	$\tilde{O}(m\rho + \rho^4/n)$	$\tilde{O}(n/\rho)$



Work on Parallel Reachability Before 2017

Algorithm	**Work**	**Depth**
Parallel BFS	O(m)	O(D)
Parallel Trans. Closure	$O(n^{\omega})$	$\tilde{O}(1)$
Spencer's [Spe97]	$\tilde{O}(m + n\rho^2)$	$\tilde{O}(n/\rho)$
UY [UY91]	$\tilde{O}(m\rho + \rho^4/n)$	$\tilde{O}(n/\rho)$

Work on Parallel Reachability Before 2017

Algorithm	**Work**	**Depth**
Parallel BFS	O(m)	O(D)
Parallel Trans. Closure	$O(n^\omega)$	$\tilde{O}(1)$
Spencer's [Spe97]	$\tilde{O}(m + n\rho^2)$	$\tilde{O}(n/\rho)$
UY [UY91]	$\tilde{O}(m\rho + \rho^4/n)$	$\tilde{O}(n/\rho)$

None are linear work and sublinear depth.

Shortcutting and Hopsets

Shortcutting and Hopsets

Parallel BFS uses O(m) work and O(D) depth

Shortcutting and Hopsets

Parallel BFS uses O(m) work and O(D) depth

But: D can be up to n.

Shortcutting and Hopsets

Parallel BFS uses O(m) work and O(D) depth

But: D can be up to n.

Idea: add O(m) edges to G that *don't affect reachability* and reduce diameter to o(n), i.e. we can add edges u -> v where u can already reach v in G

Shortcutting and Hopsets

Parallel BFS uses O(m) work and O(D) depth

But: D can be up to n.

Idea: add O(m) edges to G that *don't affect reachability* and reduce diameter to o(n), i.e. we can add edges u -> v where u can already reach v in G

Hopset: such a set of edges

Shortcut: edge in a hopset

Algorithm for Hopsets

Algorithm for Hopsets

Theorem [UY91]: We can add $\tilde{O}(n)$edges to G and reduce the diameter to $O(\sqrt{n})$.

Algorithm for Hopsets

Theorem [UY91]: We can add $\tilde{O}(n)$edges to G and reduce the diameter to $O(\sqrt{n})$.

Pick $O(\sqrt{n}\log n)$ random vertices, call the set S

Algorithm for Hopsets

Theorem [UY91]: We can add $\tilde{O}(n)$edges to G and reduce the diameter to $O(\sqrt{n})$.

Pick $O(\sqrt{n}\log n)$ random vertices, call the set S

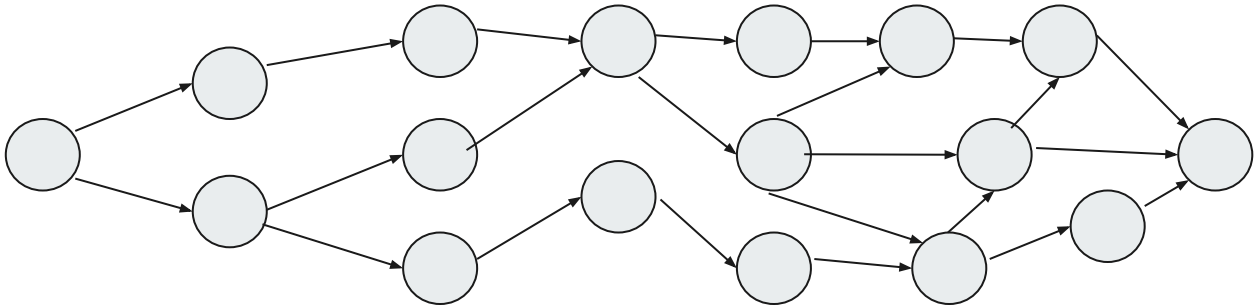
For all u, v in S add a shortcut u -> v if u can reach v in G.

Algorithm for Hopsets

Theorem [UY91]: We can add $\tilde{O}(n)$edges to G and reduce the diameter to $O(\sqrt{n})$.

Pick $O(\sqrt{n}\log n)$ random vertices, call the set S

For all u, v in S add a shortcut u -> v if u can reach v in G.

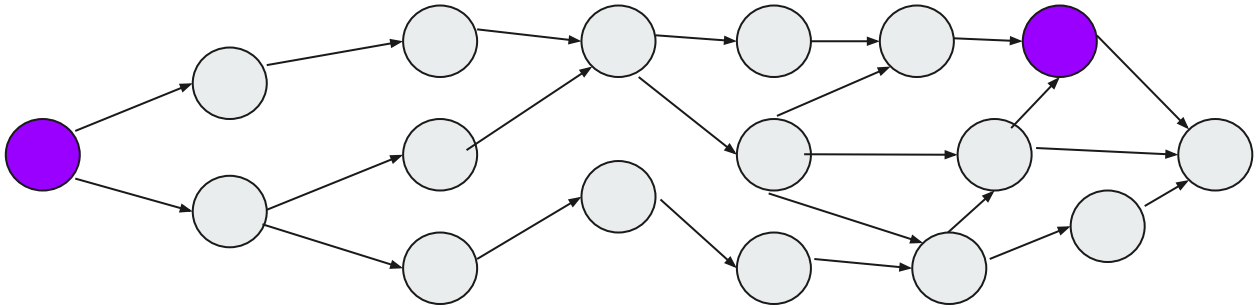


Algorithm for Hopsets

Theorem [UY91]: We can add $\tilde{O}(n)$edges to G and reduce the diameter to $O(\sqrt{n})$.

Pick $O(\sqrt{n}\log n)$ random vertices, call the set S

For all u, v in S add a shortcut u -> v if u can reach v in G.

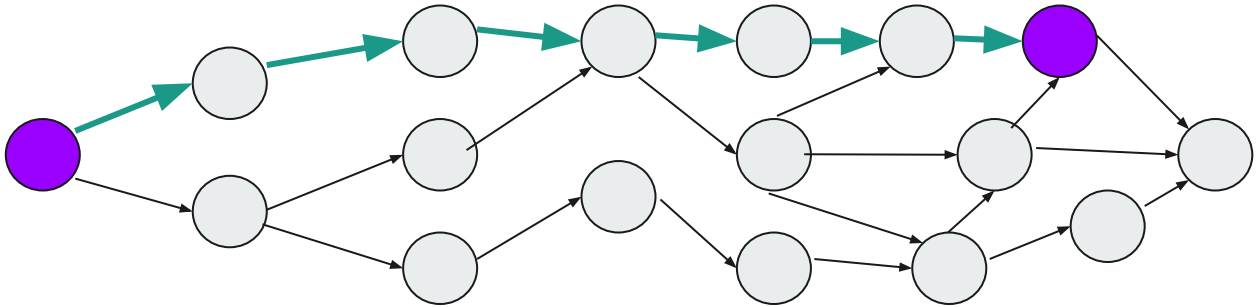


Algorithm for Hopsets

Theorem [UY91]: We can add $\tilde{O}(n)$edges to G and reduce the diameter to $O(\sqrt{n})$.

Pick $O(\sqrt{n}\log n)$ random vertices, call the set S

For all u, v in S add a shortcut u -> v if u can reach v in G.

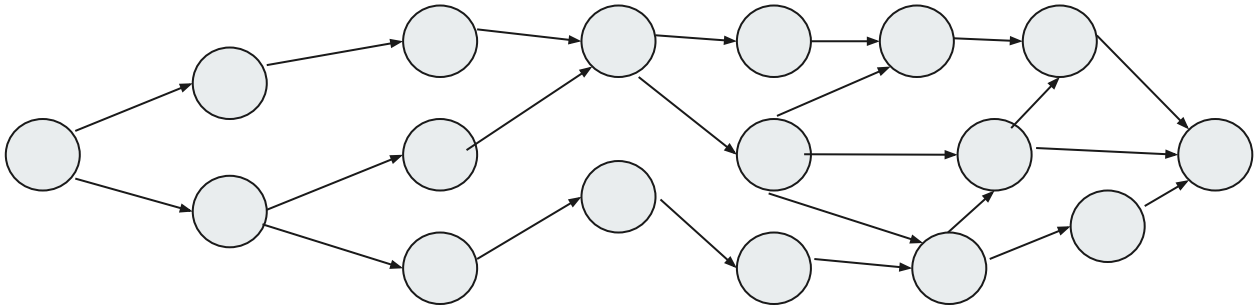


Algorithm for Hopsets

Theorem [UY91]: We can add $\tilde{O}(n)$edges to G and reduce the diameter to $O(\sqrt{n})$.

Pick $O(\sqrt{n}\log n)$ random vertices, call the set S

For all u, v in S add a shortcut u -> v if u can reach v in G.

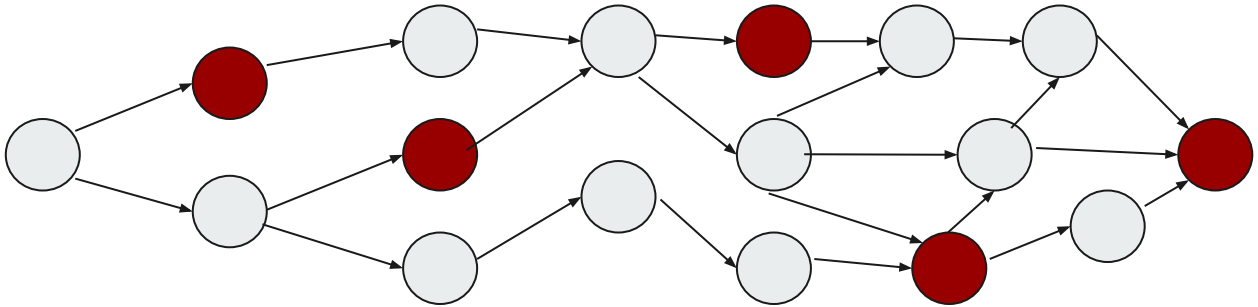


Algorithm for Hopsets

Theorem [UY91]: We can add $\tilde{O}(n)$edges to G and reduce the diameter to $O(\sqrt{n})$.

Pick $O(\sqrt{n}\log n)$ random vertices, call the set S

For all u, v in S add a shortcut u -> v if u can reach v in G.

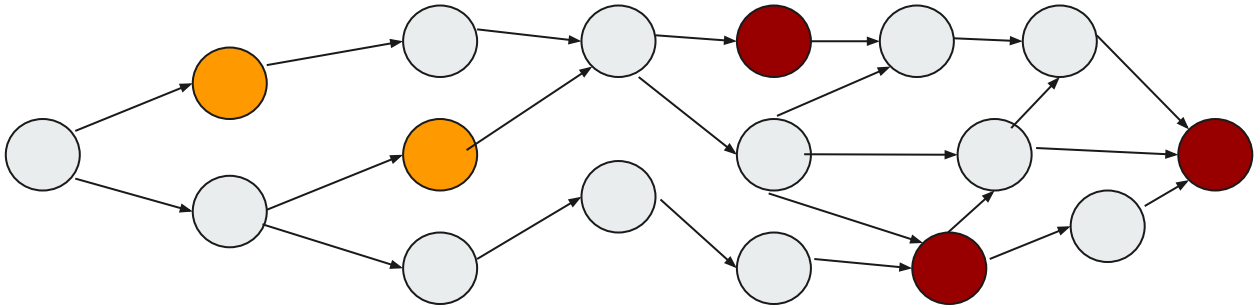


Algorithm for Hopsets

Theorem [UY91]: We can add $\tilde{O}(n)$edges to G and reduce the diameter to $O(\sqrt{n})$.

Pick $O(\sqrt{n}\log n)$ random vertices, call the set S

For all u, v in S add a shortcut u -> v if u can reach v in G.

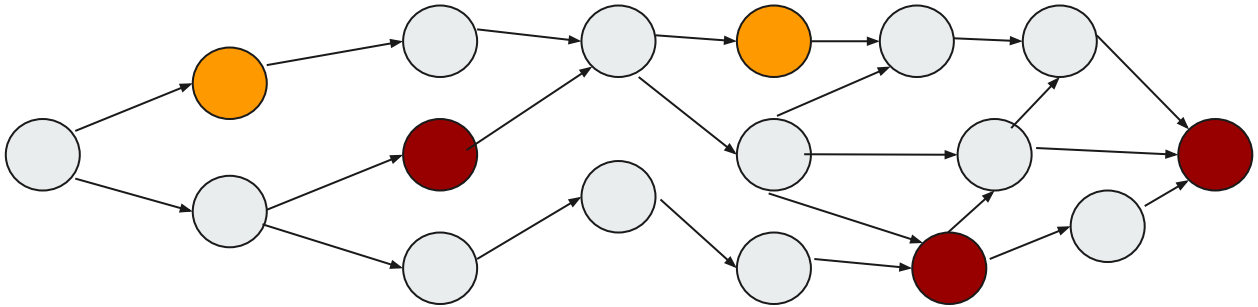


Algorithm for Hopsets

Theorem [UY91]: We can add $\tilde{O}(n)$edges to G and reduce the diameter to $O(\sqrt{n})$.

Pick $O(\sqrt{n}\log n)$ random vertices, call the set S

For all u, v in S add a shortcut u -> v if u can reach v in G.

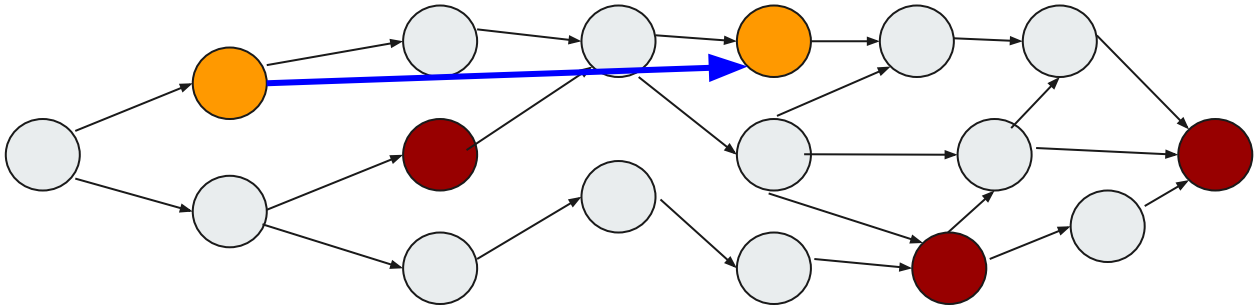


Algorithm for Hopsets

Theorem [UY91]: We can add $\tilde{O}(n)$edges to G and reduce the diameter to $O(\sqrt{n})$.

Pick $O(\sqrt{n}\log n)$ random vertices, call the set S

For all u, v in S add a shortcut u -> v if u can reach v in G.

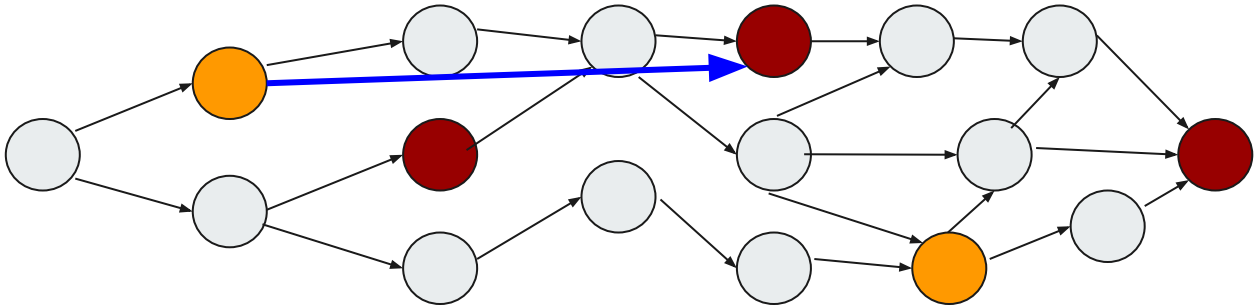


Algorithm for Hopsets

Theorem [UY91]: We can add $\tilde{O}(n)$edges to G and reduce the diameter to $O(\sqrt{n})$.

Pick $O(\sqrt{n}\log n)$ random vertices, call the set S

For all u, v in S add a shortcut u -> v if u can reach v in G.

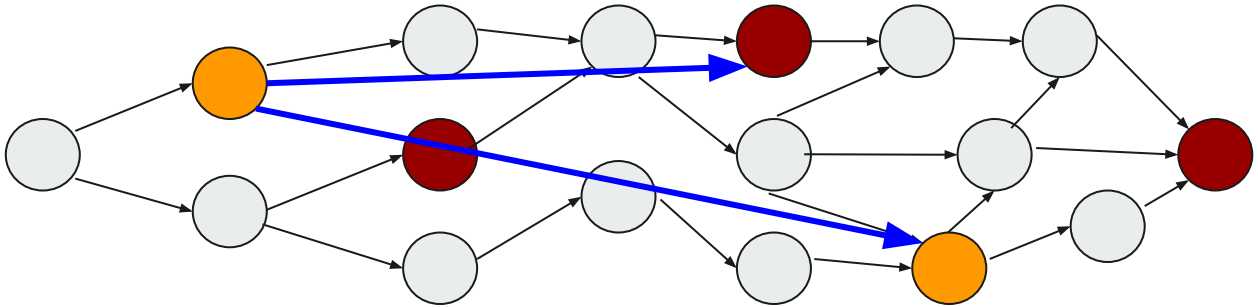


Algorithm for Hopsets

Theorem [UY91]: We can add $\tilde{O}(n)$edges to G and reduce the diameter to $O(\sqrt{n})$.

Pick $O(\sqrt{n}\log n)$ random vertices, call the set S

For all u, v in S add a shortcut u -> v if u can reach v in G.

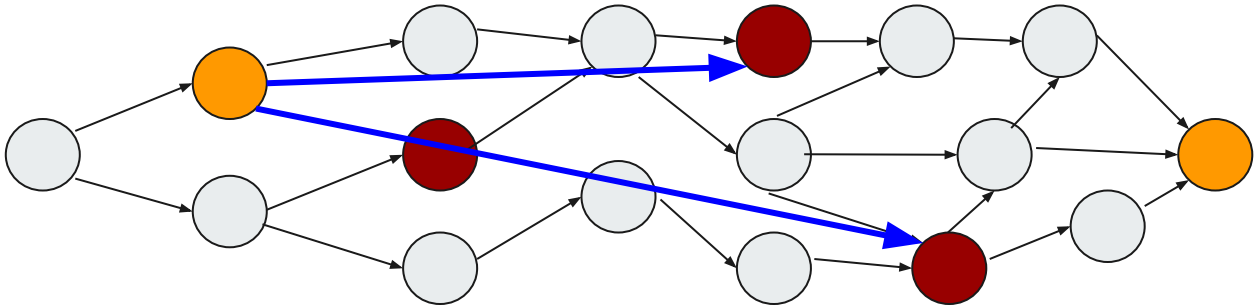


Algorithm for Hopsets

Theorem [UY91]: We can add $\tilde{O}(n)$edges to G and reduce the diameter to $O(\sqrt{n})$.

Pick $O(\sqrt{n}\log n)$ random vertices, call the set S

For all u, v in S add a shortcut u -> v if u can reach v in G.

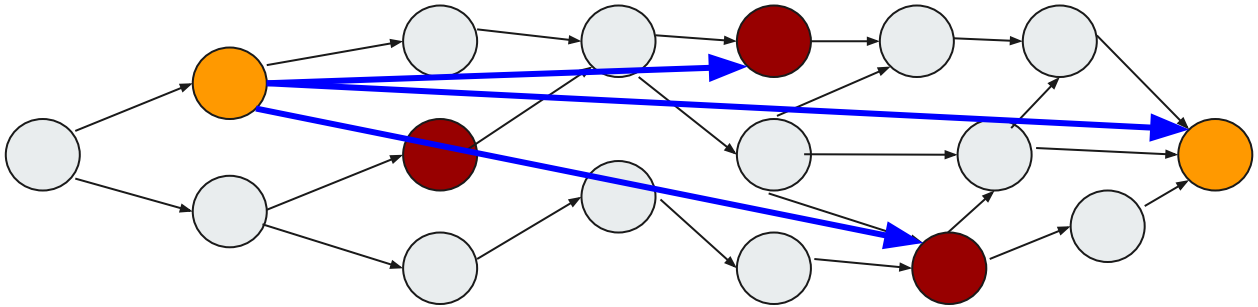


Algorithm for Hopsets

Theorem [UY91]: We can add $\tilde{O}(n)$edges to G and reduce the diameter to $O(\sqrt{n})$.

Pick $O(\sqrt{n}\log n)$ random vertices, call the set S

For all u, v in S add a shortcut u -> v if u can reach v in G.

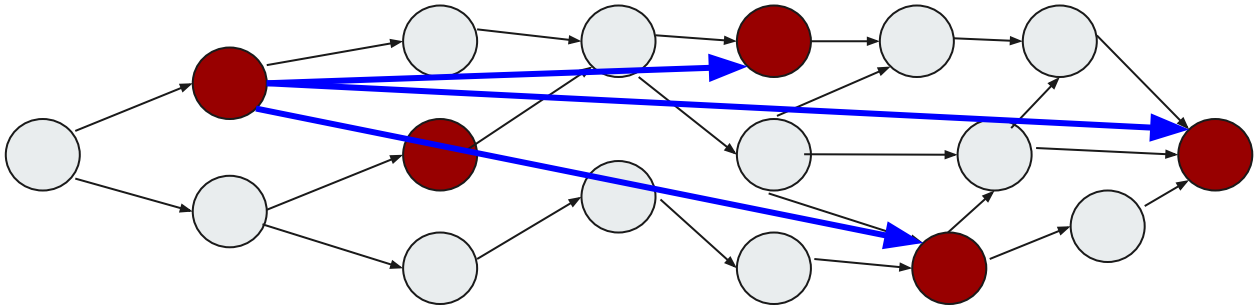


Algorithm for Hopsets

Theorem [UY91]: We can add $\tilde{O}(n)$edges to G and reduce the diameter to $O(\sqrt{n})$.

Pick $O(\sqrt{n}\log n)$ random vertices, call the set S

For all u, v in S add a shortcut u -> v if u can reach v in G.

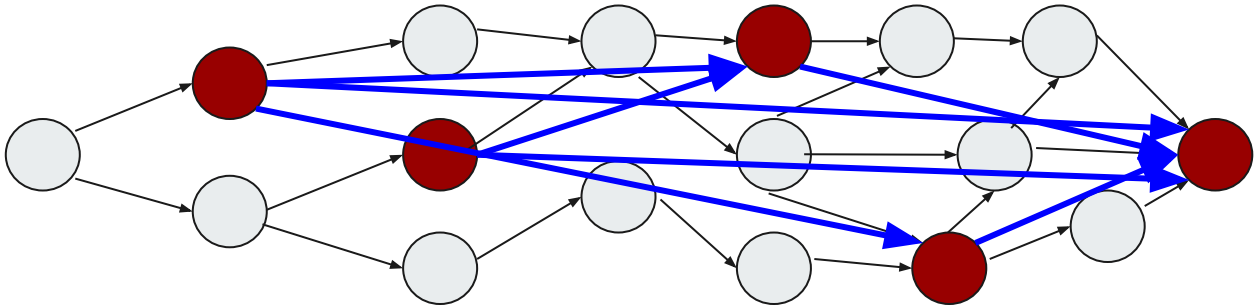


Algorithm for Hopsets

Theorem [UY91]: We can add $\tilde{O}(n)$edges to G and reduce the diameter to $O(\sqrt{n})$.

Pick $O(\sqrt{n}\log n)$ random vertices, call the set S

For all u, v in S add a shortcut u -> v if u can reach v in G.

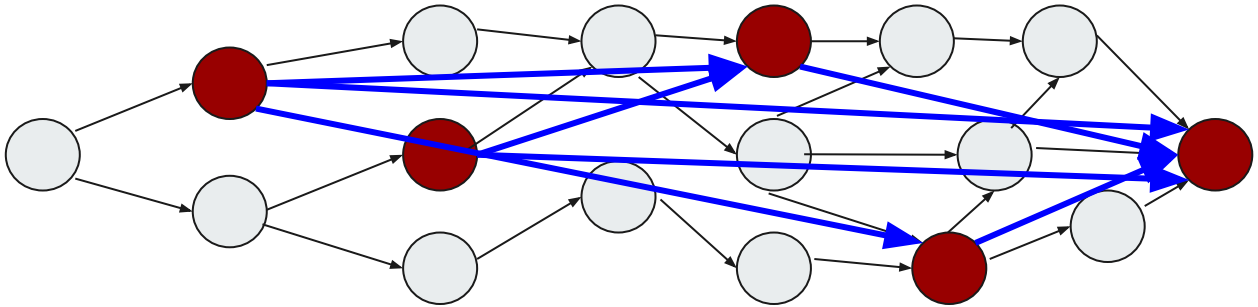


Algorithm for Hopsets

Theorem [UY91]: We can add $\tilde{O}(n)$edges to G and reduce the diameter to $O(\sqrt{n})$.

Pick $O(\sqrt{n}\log n)$ random vertices, call the set S

For all u, v in S add a shortcut u -> v if u can reach v in G.



Blue edges form a hopset.

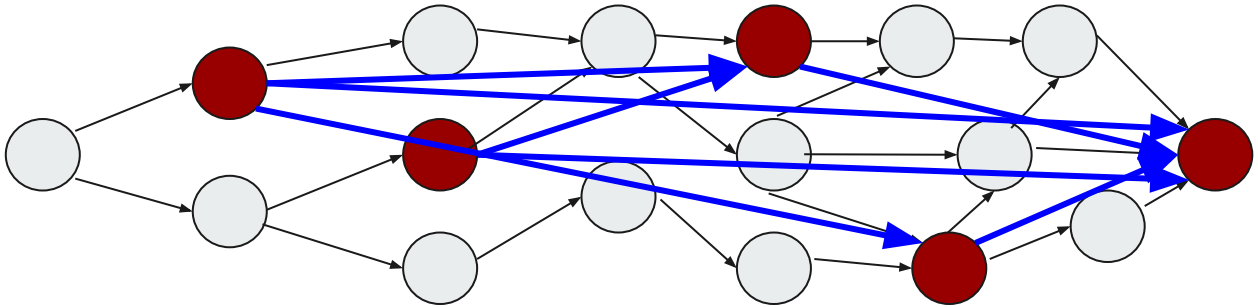
Each blue edge is a shortcut.

Algorithm for Hopsets

Theorem [UY91]: We can add $\tilde{O}(n)$edges to G and reduce the diameter to $O(\sqrt{n})$.

Pick $O(\sqrt{n}\log n)$ random vertices, call the set S

For all u, v in S add a shortcut u -> v if u can reach v in G.

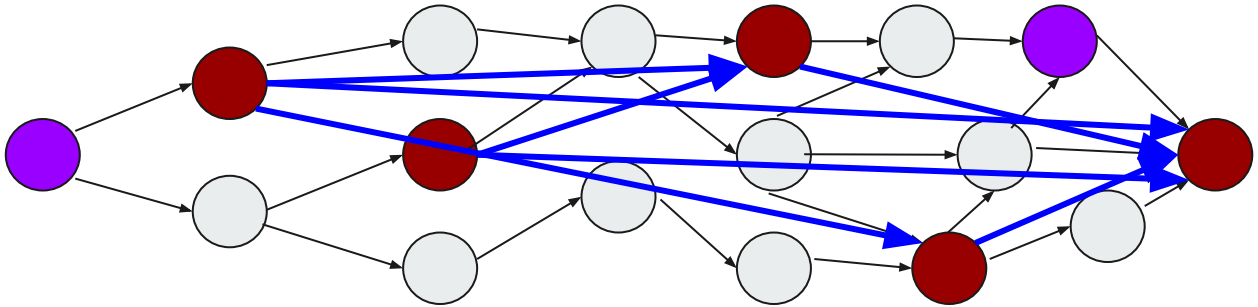


Algorithm for Hopsets

Theorem [UY91]: We can add $\tilde{O}(n)$edges to G and reduce the diameter to $O(\sqrt{n})$.

Pick $O(\sqrt{n}\log n)$ random vertices, call the set S

For all u, v in S add a shortcut u -> v if u can reach v in G.

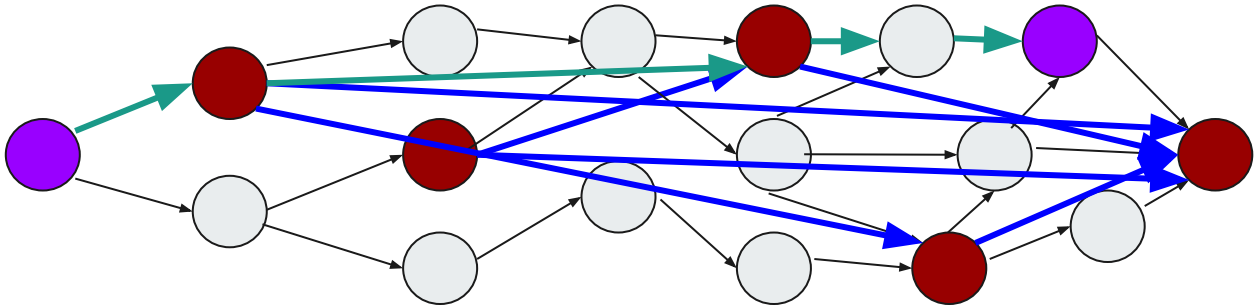


Algorithm for Hopsets

Theorem [UY91]: We can add $\tilde{O}(n)$edges to G and reduce the diameter to $O(\sqrt{n})$.

Pick $O(\sqrt{n}\log n)$ random vertices, call the set S

For all u, v in S add a shortcut u -> v if u can reach v in G.

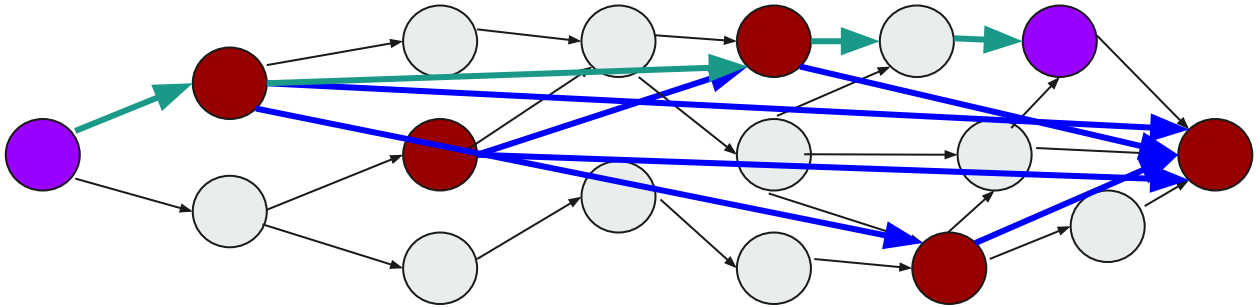


Algorithm for Hopsets

Theorem [UY91]: We can add $\tilde{O}(n)$edges to G and reduce the diameter to $O(\sqrt{n})$. **Work:** $\tilde{O}(m\sqrt{n})$ **Depth:** $O(\sqrt{n})$

Pick $O(\sqrt{n}\log n)$ random vertices, call the set S

For all u, v in S add a shortcut u -> v if u can reach v in G.



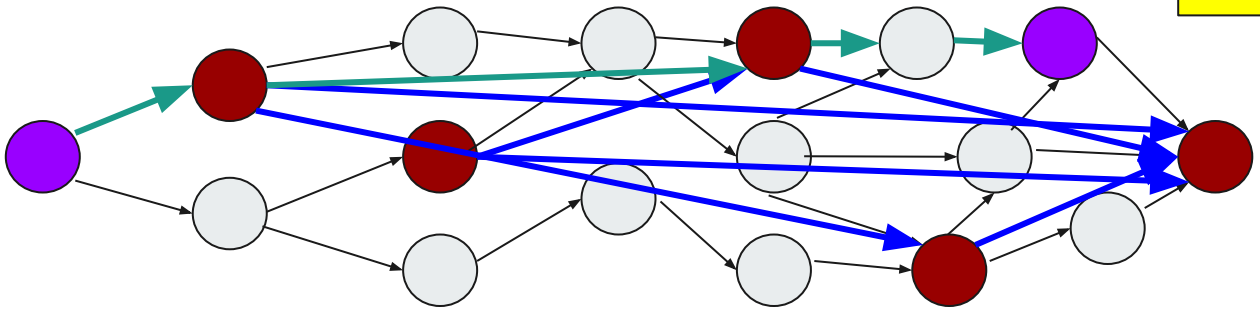
Algorithm for Hopsets

Theorem [UY91]: We can add $\tilde{O}(n)$edges to G and reduce the diameter to $O(\sqrt{n})$. **Work:** $\tilde{O}(m\sqrt{n})$ **Depth:** $O(\sqrt{n})$

Pick $O(\sqrt{n}\log n)$ random vertices, call the set S

For all u, v in S add a shortcut u -> v if u can reach v in G.

Good depth, but superlinear work.



Algorithm for Hopsets

Theorem [UY91]: We can add $\tilde{O}(n)$edges to G and reduce the diameter to $O(\sqrt{n})$. **Work:** $\tilde{O}(m\sqrt{n})$ **Depth:** $O(\sqrt{n})$

Algorithm for Hopsets

Theorem [UY91]: We can add $\tilde{O}(n)$edges to G and reduce the diameter to $O(\sqrt{n})$. **Work:** $\tilde{O}(m\sqrt{n})$ **Depth:** $O(\sqrt{n})$

Best known *combinatorial construction* (even ignoring algorithmic efficiency)

Fineman's Algorithm

Fineman's Algorithm

Theorem [Fin18]: There is an algorithm which computes a hopset with $\tilde{O}(n)$ edges which reduces the diameter to $\tilde{O}(n^{2/3})$.
Work: $\tilde{O}(m)$ **Depth:** $\tilde{O}(n^{2/3})$

Fineman's Algorithm

Theorem [Fin18]: There is an algorithm which computes a hopset with $\tilde{O}(n)$ edges which reduces the diameter to $\tilde{O}(n^{2/3})$.
Work: $\tilde{O}(m)$ **Depth:** $\tilde{O}(n^{2/3})$

Theorem [UY91]: We can add $\tilde{O}(n)$edges to G and reduce the diameter to $O(\sqrt{n})$. **Work:** $\tilde{O}(m\sqrt{n})$ **Depth:** $O(\sqrt{n})$

Fineman's Algorithm

Theorem [Fin18]: There is an algorithm which computes a hopset with $\tilde{O}(n)$ edges which reduces the diameter to $\tilde{O}(n^{2/3})$.
Work: $\tilde{O}(m)$ **Depth:** $\tilde{O}(n^{2/3})$

Theorem [UY91]: We can add $\tilde{O}(n)$edges to G and reduce the diameter to $O(\sqrt{n})$. **Work:** $\tilde{O}(m\sqrt{n})$ **Depth:** $O(\sqrt{n})$

Can we match the *hopset bound*, i.e. nearly linear work and depth $O(\sqrt{n})$?

Our Results

Our Results

Theorem [JLS19]: There is an algorithm which computes a hopset with $\tilde{O}(nk)$ edges which reduces the diameter to $n^{1/2+O(1/\log k)}$.
Work: $\tilde{O}(mk)$ **Depth:** $k^2 \cdot n^{1/2+O(1/\log k)}$

Our Results

Theorem [JLS19]: There is an algorithm which computes a hopset with $\tilde{O}(nk)$ edges which reduces the diameter to $n^{1/2+O(1/\log k)}$.
Work: $\tilde{O}(mk)$ **Depth:** $k^2 \cdot n^{1/2+O(1/\log k)}$

Theorem [JLS19]: There is an algorithm which computes a hopset with $\tilde{O}(n)$ edges which reduces the diameter to $n^{1/2+o(1)}$.
Work: $\tilde{O}(m)$ **Depth:** $n^{1/2+o(1)}$

Distributed Computation

Distributed Computation

CONGEST model

Distributed Computation

CONGEST model

Directed graph G.

Distributed Computation

CONGEST model

Directed graph G.

Each vertex is a processor with infinite computational power.

Vertices only know their in and out neighbors.

Distributed Computation

CONGEST model

Directed graph G.

Each vertex is a processor with infinite computational power.

Vertices only know their in and out neighbors.

Each round, a vertex can send a message of length O(log n) to each of its in and out neighbors (not necessarily the same message).

Distributed Computation

Distributed Computation

Complexity is measured in number of rounds of communication.

Distributed Computation

Complexity is measured in number of rounds of communication.

Complexity depends on n (number of vertices) and D (undirected diameter of the underlying graph G)

Distributed Reachability Problem

Distributed Reachability Problem

For source vertex s, every other vertex t must learn whether s can reach it.

Distributed Reachability Problem

For source vertex s, every other vertex t must learn whether s can reach it.

s does not have to know which vertices it can reach

Distributed Reachability Problem

For source vertex s, every other vertex t must learn whether s can reach it.

s does not have to know which vertices it can reach

Every vertex knows s to start

Distributed Reachability: What's Known

Distributed Reachability: What's Known

Theorem [GU15]: There is an algorithm for reachability in the **CONGEST** model which takes $\tilde{O}(D + \sqrt{n}D^{1/4})$ rounds

Distributed Reachability: What's Known

Theorem [GU15]: There is an algorithm for reachability in the **CONGEST** model which takes $\tilde{O}(D + \sqrt{n}D^{1/4})$ rounds

Theorem [DSHK+11]: Any algorithm for reachability in the **CONGEST** model requires $\tilde{\Omega}(D + \sqrt{n})$ rounds.

Distributed Reachability: What's Known

Distributed Reachability: Our Results

Theorem [JLS19]: There is an algorithm for reachability in the **CONGEST** model which takes $\tilde{O}(n^{1/2} + n^{1/3+o(1)} D^{2/3})$ rounds.

Distributed Reachability: Our Results

Theorem [JLS19]: There is an algorithm for reachability in the **CONGEST** model which takes $\tilde{O}(n^{1/2} + n^{1/3+o(1)} D^{2/3})$ rounds.

Matches the lower bound of **[DSHK+11]** for $D = O(n^{1/4-\epsilon})$

Distributed Reachability: Our Results

Theorem [JLS19]: There is an algorithm for reachability in the **CONGEST** model which takes $\tilde{O}(n^{1/2} + n^{1/3+o(1)} D^{2/3})$ rounds.

Matches the lower bound of **[DSHK+11]** for $D = O(n^{1/4-\epsilon})$

First algorithm matching the lower bound for $D = \Omega(n^{\delta})$ for some $\delta > 0$

Distributed Reachability: Our Results

Theorem [JLS19]: There is an algorithm for reachability in the **CONGEST** model which takes $\tilde{O}(n^{1/2} + n^{1/3+o(1)} D^{2/3})$ rounds.

Matches the lower bound of **[DSHK+11]** for $D = O(n^{1/4-\epsilon})$

First algorithm matching the lower bound for $D = \Omega(n^{\delta})$ for some $\delta > 0$

Our algorithm in the **CONGEST** model is a fairly general way of converting "reasonable" low work and depth parallel reachability algorithms into distributed algorithms.

Distributed Reachability: Our Results

Theorem [JLS19]: There is an algorithm for reachability in the **CONGEST** model which takes $\tilde{O}(n^{1/2} + n^{1/3+o(1)} D^{2/3})$ rounds.

Matches the lower bound of **[DSHK+11]** for $D = O(n^{1/4-\epsilon})$

First algorithm matching the lower bound for $D = \Omega(n^{\delta})$ for some $\delta > 0$

Our algorithm in the **CONGEST** model is a fairly general way of converting "reasonable" low work and depth parallel reachability algorithms into distributed algorithms.

Using Fineman's algorithm instead of our algorithm still improves over the previous state of the art.

Organization for Remainder of the Talk

Organization for Remainder of the Talk

Remainder of the Talk:
Proof sketch of our parallel reachability algorithm.

Organization for Remainder of the Talk

Remainder of the Talk:
Proof sketch of our parallel reachability algorithm.

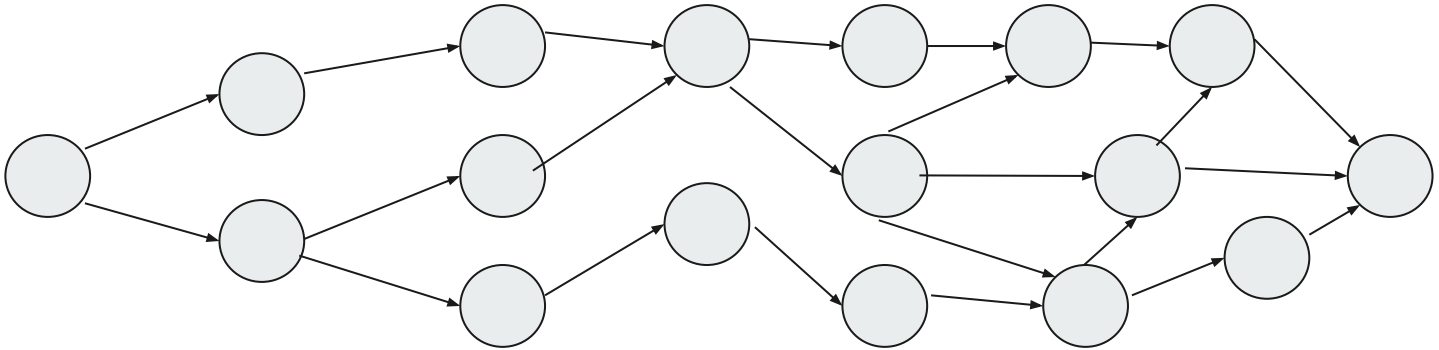
Theorem [JLS19]: There is an algorithm which computes a hopset with $\tilde{O}(nk)$ edges which reduces the diameter to $n^{1/2+O(1/\log k)}$.
Work: $\tilde{O}(mk)$ **Depth:** $k^2 \cdot n^{1/2+O(1/\log k)}$

Theorem [JLS19]: There is an algorithm which computes a hopset with $\tilde{O}(n)$ edges which reduces the diameter to $n^{1/2+o(1)}$.
Work: $\tilde{O}(m)$ **Depth:** $n^{1/2+o(1)}$

How to Construct Hopsets?

We will ignore strongly connected components.

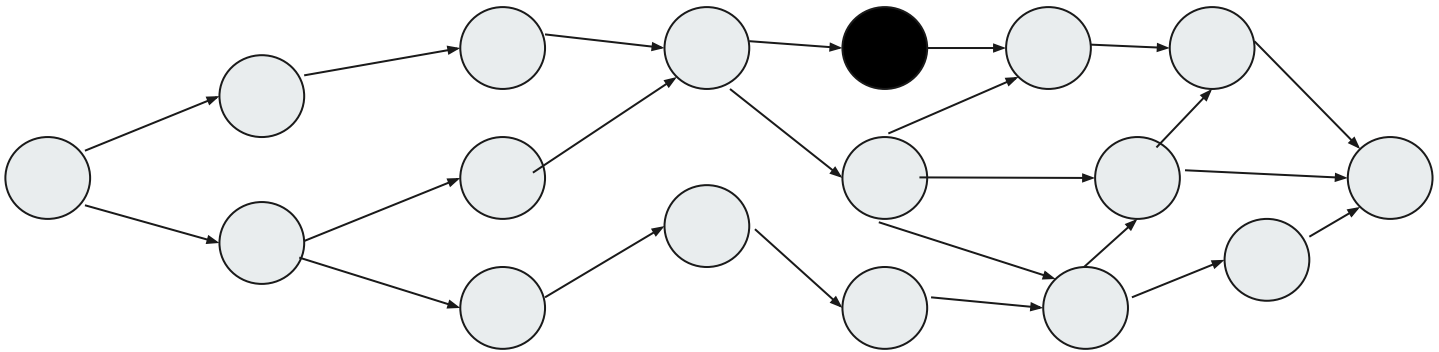
We first give a serial linear-time construction. We'll discuss parallelization later.



How to Construct Hopsets?

We will ignore strongly connected components.

[Fineman 2018] Pick a uniformly random vertex **v**.

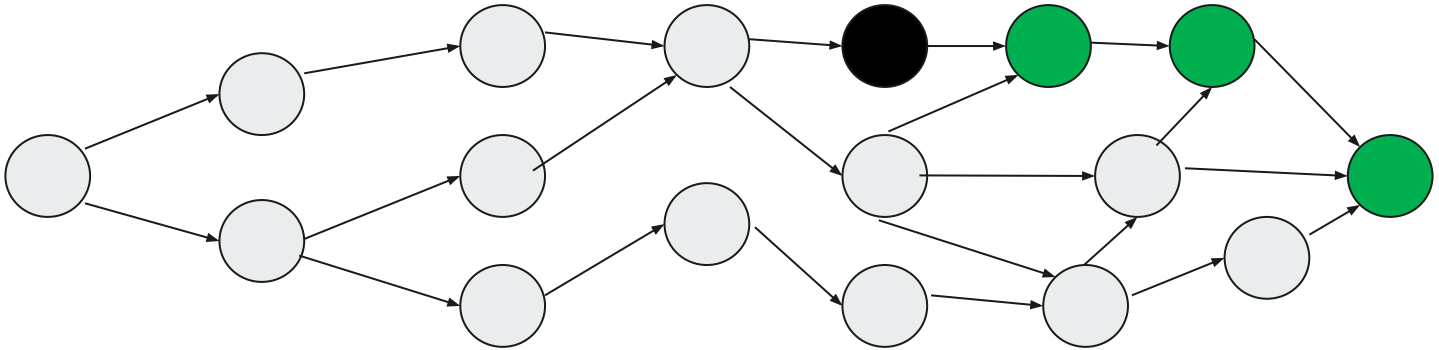


How to Construct Hopsets?

We will ignore strongly connected components.

[Fineman 2018] Pick a uniformly random vertex **v**. Call it the *shortcutter.*

Add a shortcut from **v** to every vertex it can reach (**D = Descendant**)



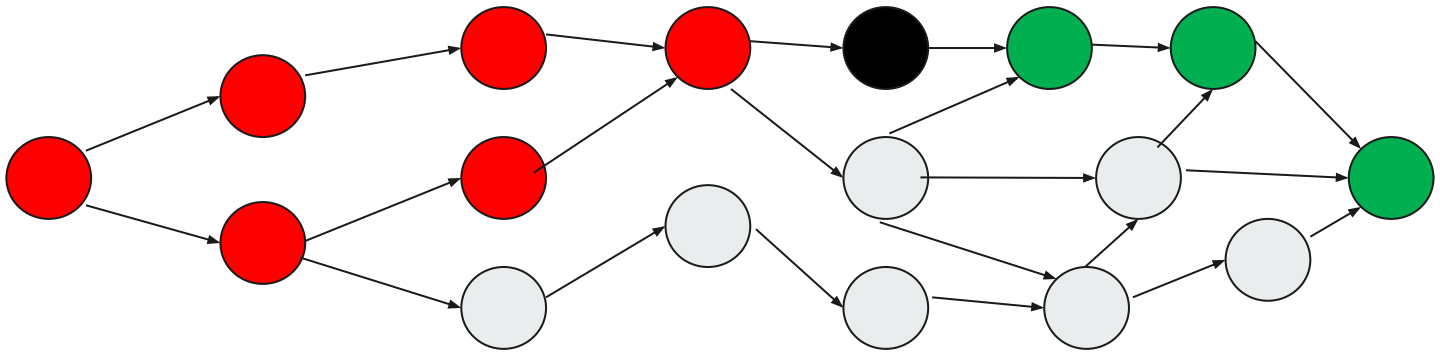
How to Construct Hopsets?

We will ignore strongly connected components.

[Fineman 2018] Pick a uniformly random vertex **v**. Call it the *shortcutter.*

Add a shortcut from **v** to every vertex it can reach (**D = Descendant**)

Add a shortcut to **v** from every vertex it can reach (**A = Ancestor**)



How to Construct Hopsets?

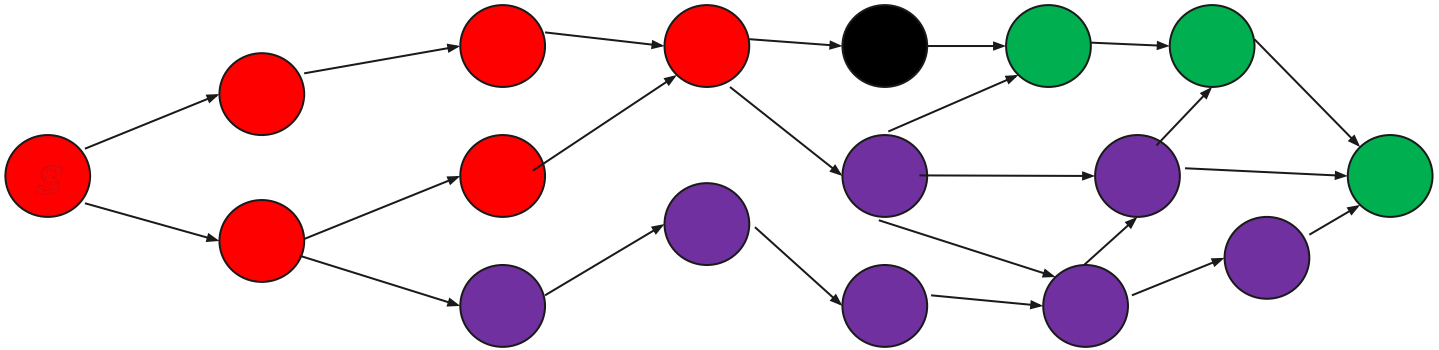
We will ignore strongly connected components.

[Fineman 2018] Pick a uniformly random vertex **v**. Call it the *shortcutter.*

Add a shortcut from **v** to every vertex it can reach (**D = Descendant**)

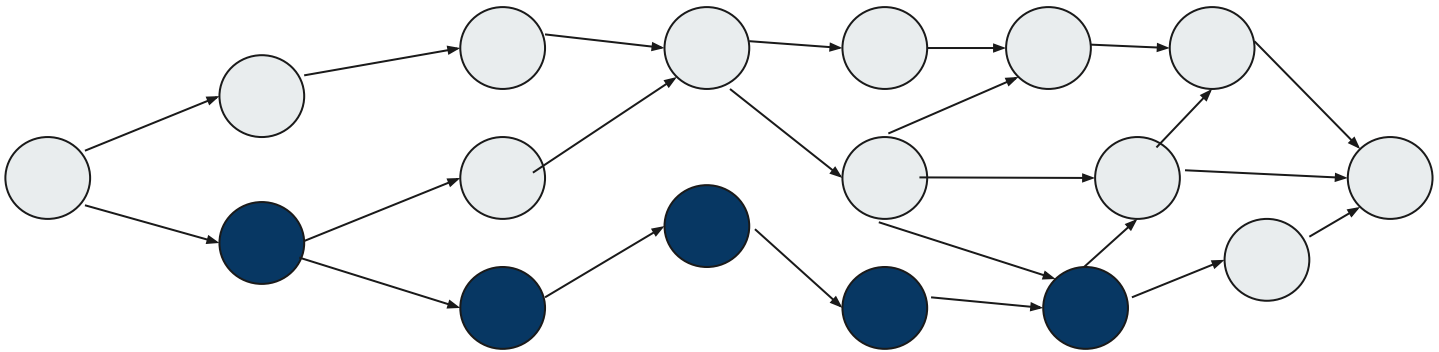
Add a shortcut to **v** from every vertex it can reach (**A = Ancestor**)

Recurse on **D**, **A**, and **U = V \ D \ A = Unrelated**.



How to Analyze?

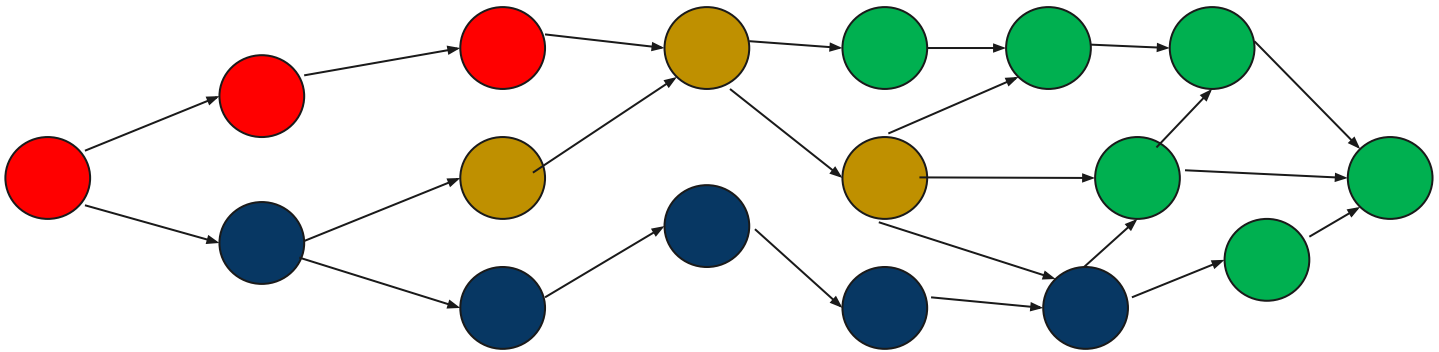
Fix a simple **path P** in the graph.



How to Analyze?

Fix a simple **path P** in the graph.

Look at **D = Descendant**, **A = Ancestor**, and **B = D ∩ A = Bridges**

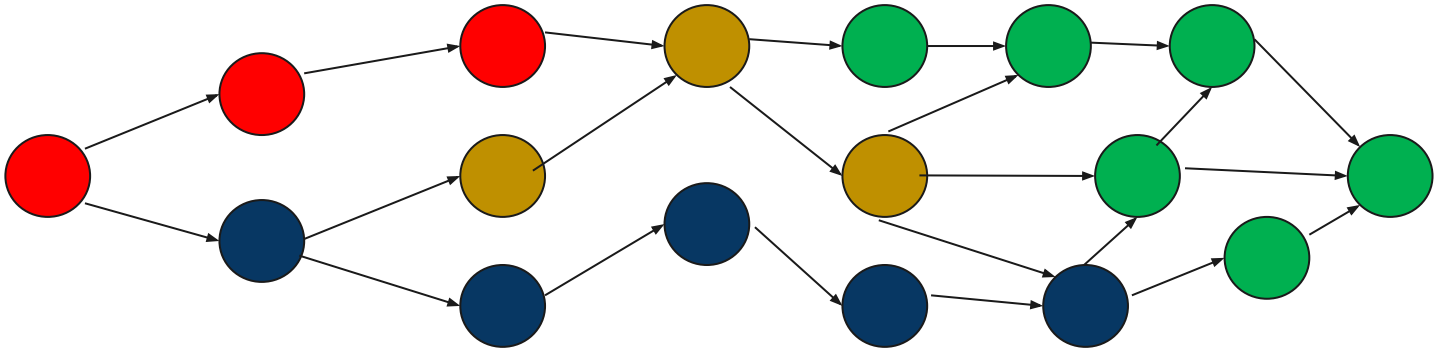


How to Analyze?

Fix a simple **path P** in the graph.

Look at **D = Descendant**, **A = Ancestor**, and **B = D ∩ A = Bridges**

Observation:
Picking a bridge to shortcut from shortcuts the path to length 2.

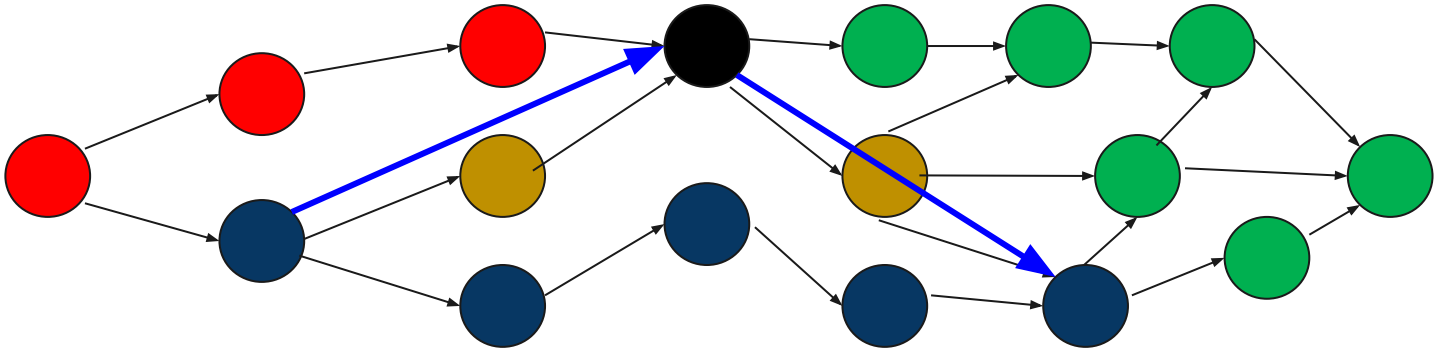


How to Analyze?

Fix a simple **path P** in the graph.

Look at **D = Descendant**, **A = Ancestor**, and **B = D ∩ A = Bridges**

Observation:
Picking a bridge to shortcut from shortcuts the path to length 2.

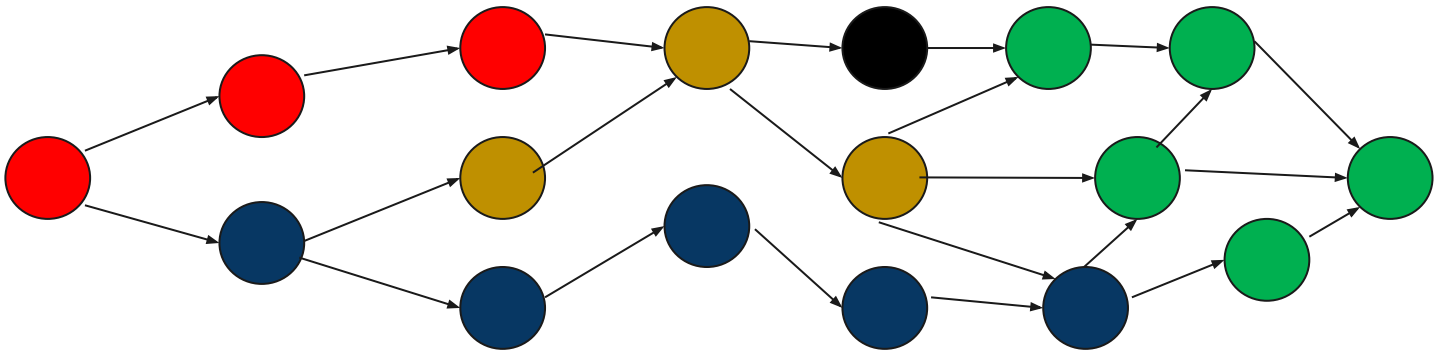


How to Analyze?

Fix a simple **path P** in the graph.

Look at **D = Descendant**, **A = Ancestor**, and **B = D ∩ A = Bridges**

Of course, we cannot guarantee that we pick a bridge:



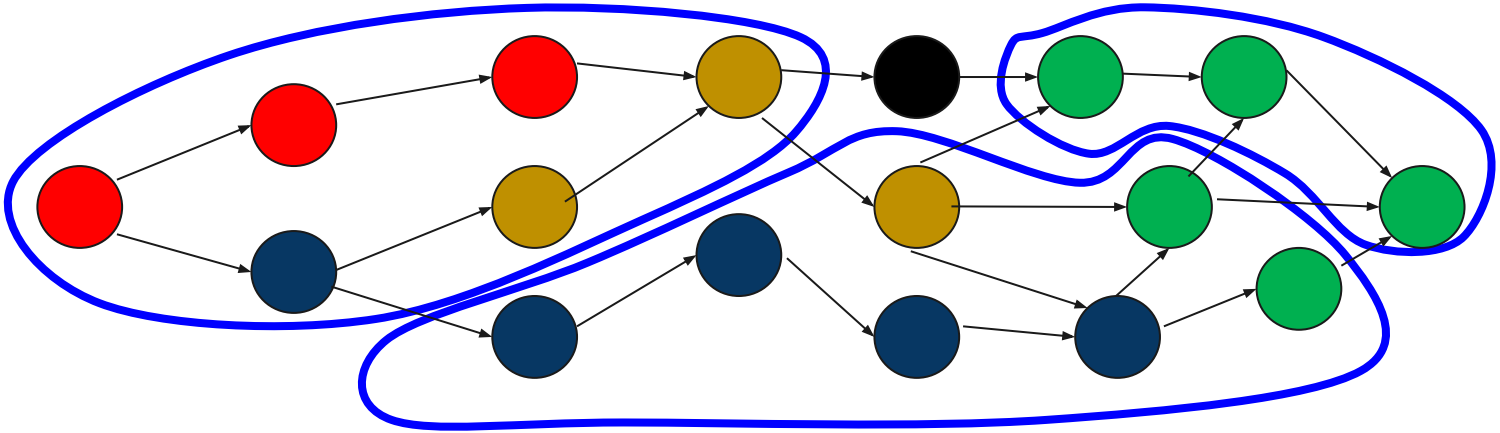
How to Analyze?

Fix a simple **path P** in the graph.

Look at **D = Descendant**, **A = Ancestor**, and **B = D ∩ A = Bridges**

Of course, we cannot guarantee that we pick a bridge:

Path gets split among different vertex sets: we recurse on each separately.

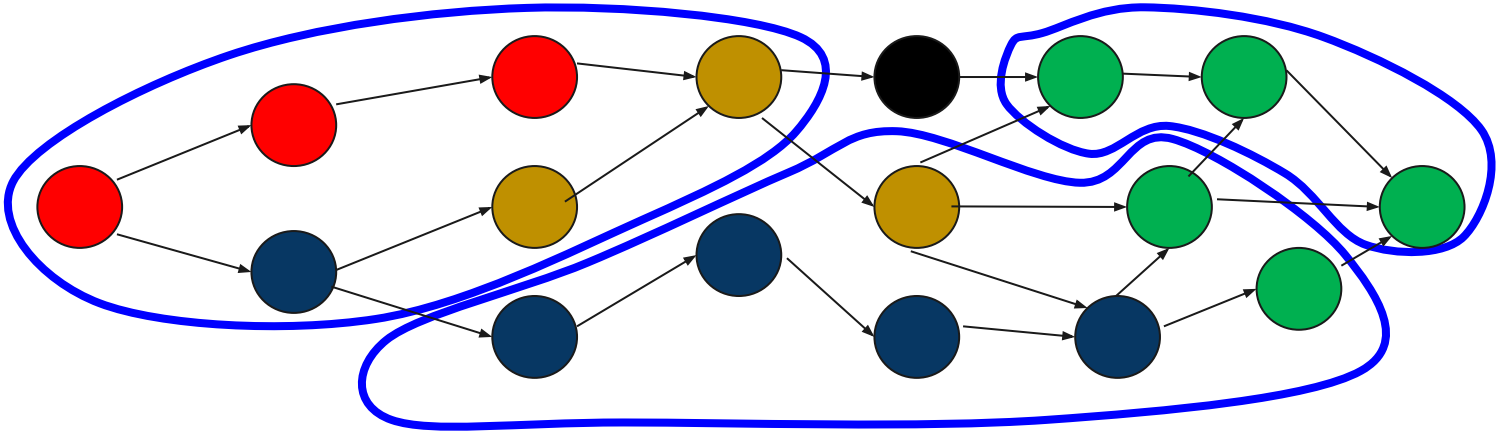


How to Analyze?

Fix a simple **path P** in the graph.

Look at **D = Descendant**, **A = Ancestor**, and **B = D ∩ A = Bridges**

Call **D ∪ A ∪ B** the set of *path-related* vertices. Let $\mathbf{L_P(V) = |D| + |A| + |B|}$.



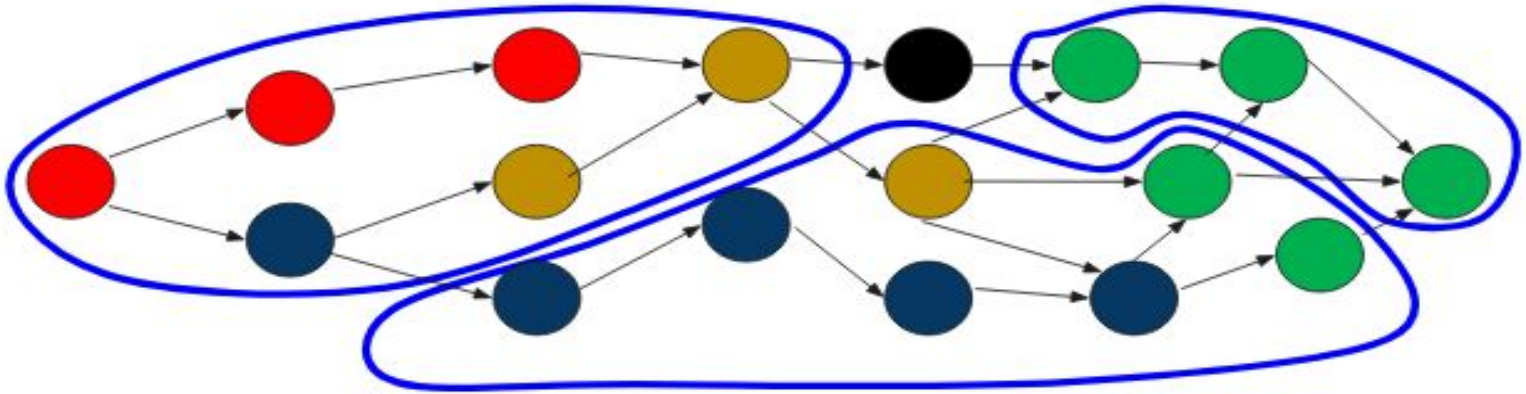
How to Analyze?

Observation 1: If we choose a shortcutter which is unrelated to **P**, the path stays intact in one of the recursively generated subproblems.

How to Analyze?

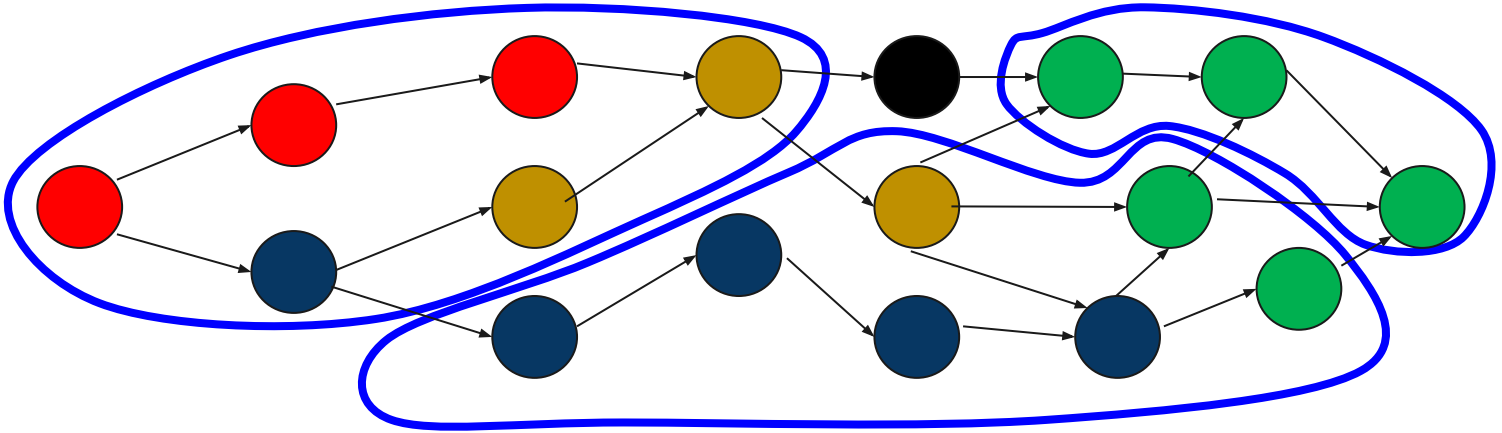
Observation 1: If we choose a shortcutter which is unrelated to **P**, the path stays intact in one of the recursively generated subproblems.

Observation 2: If we chose a shortcutter which is path related for **P,** the path either gets shortcut to length 2 or divided among at most two subproblems.



How to Analyze?

Observe that in the example below, shortcutting means that 3 of the remaining 5 path descendants are separated from any part of the path.

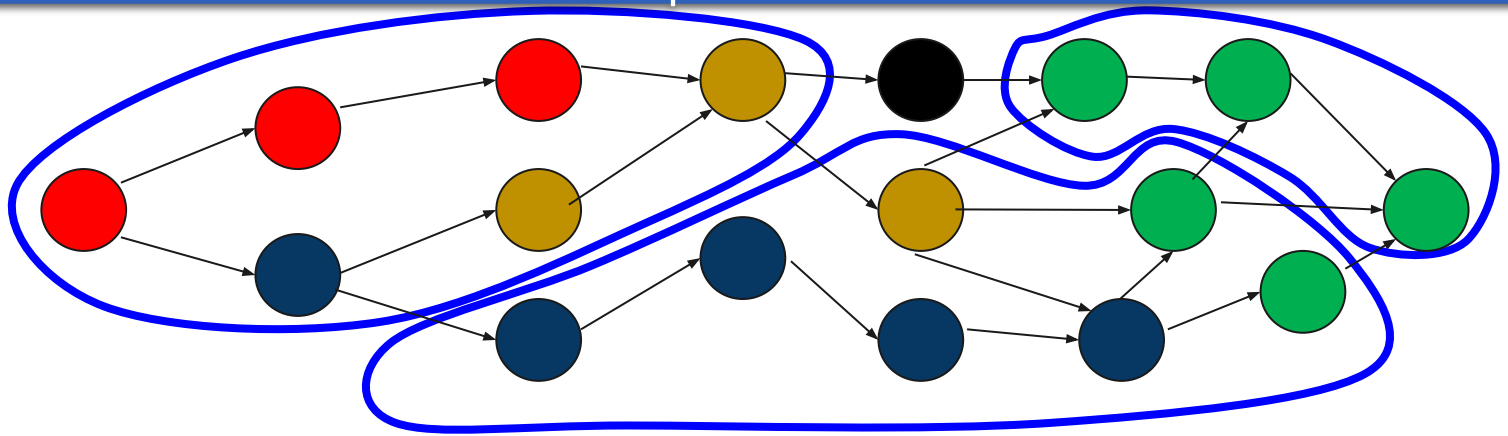


How to Analyze?

Observe that in the example below, shortcutting means that 3 of the remaining 5 path descendants are separated from any part of the path.

Key Lemma

Conditioned on picking an ancestor (resp. descendant) the expected # of ancestors (resp. descendants) related to any part of P decreases by ½ in expectation.



A Crude Initial Bound

Corollary

Conditioned on shortcutting with a path related-vertex, the expected # of vertices related to any of the path decreases by ¾ in expectation.

A Crude Initial Bound

Corollary

Conditioned on shortcutting with a path related-vertex, the expected # of vertices related to any of the path decreases by ¾ in expectation.

Observe $|P| < L_P(V) = |D| + |A| + |B| < n$.

A Crude Initial Bound

Corollary

Conditioned on shortcutting with a path related-vertex, the expected # of vertices related to any of the path decreases by ¾ in expectation.

Observe **|P| < $L_P(V)$ = |D| + |A| + |B|** < n.

Let $f(L)$ be the expected shortcutted length of a path with L related vertices.

By combining our observations with the above we obtain the recursion

$$f(L) \leq \max_{a+b \leq \frac{3L}{4}} f(a) + f(b) + 1$$

whenever our randomly selected shortcutter is related to **P.**

A Crude Initial Bound

Corollary

Conditioned on shortcutting with a path related-vertex, the expected # of vertices related to any of the path decreases by ¾ in expectation.

$$f(L) \leq \max_{a+b \leq \frac{3L}{4}} f(a) + f(b) + 1$$

As clearly $f(L) \leq L$, running this recursion for *r* levels and optimizing gives

$$f(L_P(V)) \leq f(n) \leq 2^r + (\tfrac{3}{4})^r n \leq \tilde{O}(n^{1/\log_2(8/3)}) \leq \tilde{O}(n^{0.707})$$

A Crude Initial Bound

Key Lemma

Conditioned on picking an ancestor (resp. descendant) the expected # of ancestors (resp. descendants) decrease by ½ in expectation.

Thus, we get hopsets in near-linear work with diameter $\tilde{O}(n^{1/\log_2(8/3)}) = \tilde{O}(n^{.707})$

A Crude Initial Bound

Key Lemma

Conditioned on picking an ancestor (resp. descendant) the expected # of ancestors (resp. descendants) decrease by ½ in expectation.

Thus, we get hopsets in near-linear work with diameter $\tilde{O}(n^{1/\log_2(8/3)}) = \tilde{O}(n^{.707})$

Improved analysis achieves diameter $\tilde{O}(n^{2/3})$.

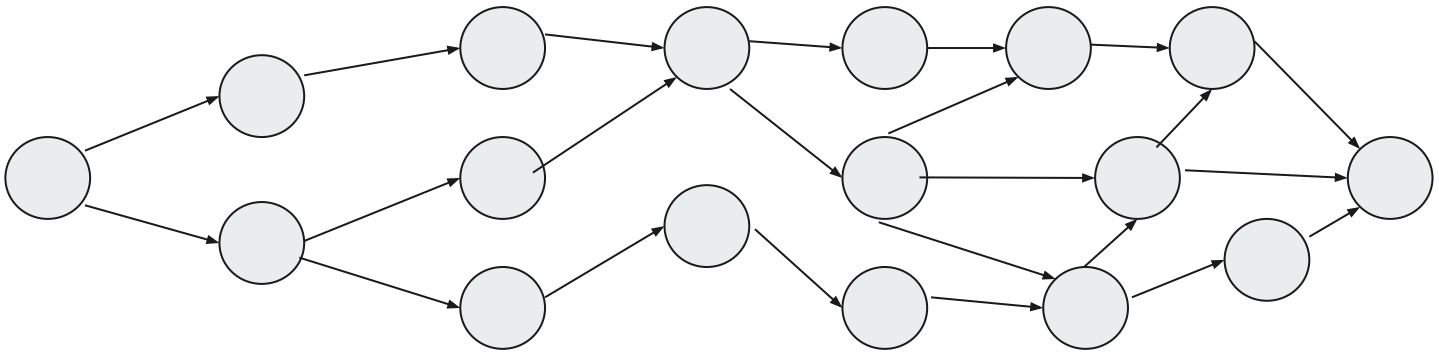
How to improve?

Improved Construction

Pick k random vertices $v_1, v_2, \ldots, v_k$.

Add a shortcut from v_i to every **Descendant** of v_i

Add a shortcut to v_i from every **Ancestor** of v_i

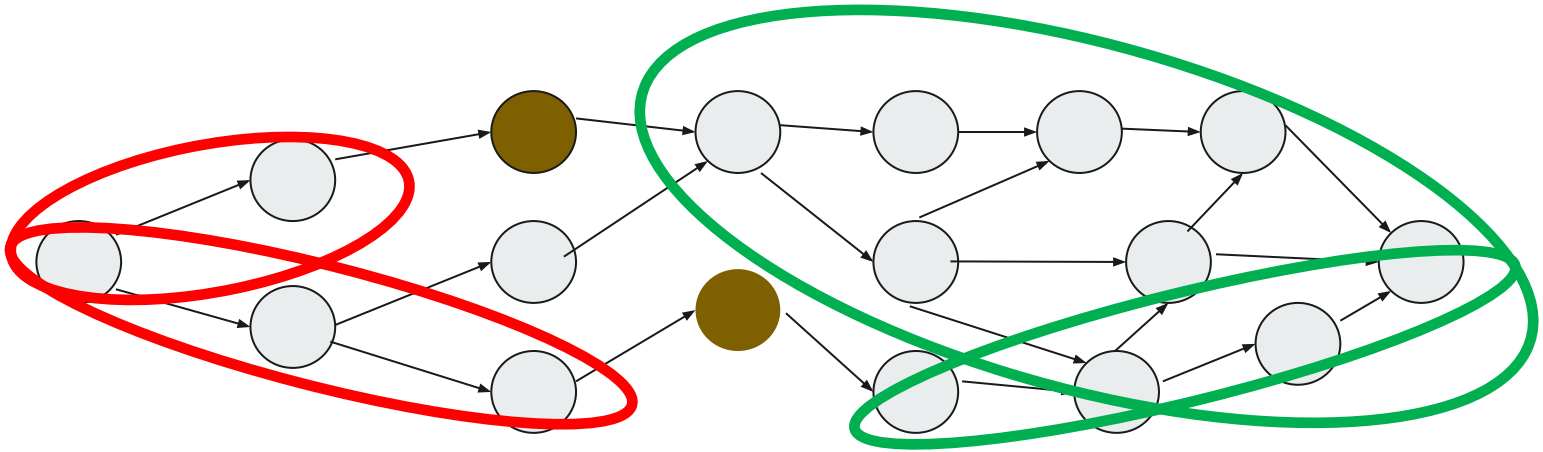


Improved Construction

Pick k random vertices $v_1, v_2, \ldots, v_k$.

Add a shortcut from v_i to every **Descendant**

Add a shortcut to v_i from every **Ancestor**



Improved Construction

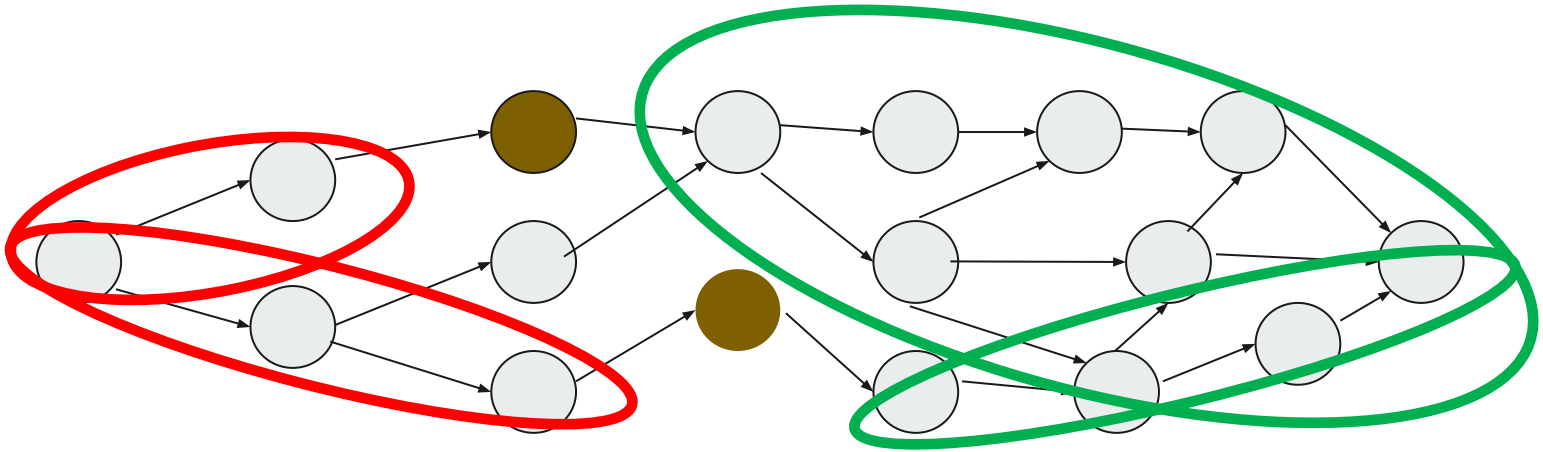
How to partition and recurse

Partition vertices based on whether they had edges added to/from same vertices.

Pick k random vertices $v_1, v_2, \ldots, v_k$.

Add a shortcut from v_i to every **Descendant**

Add a shortcut to v_i from every **Ancestor**



Improved Construction

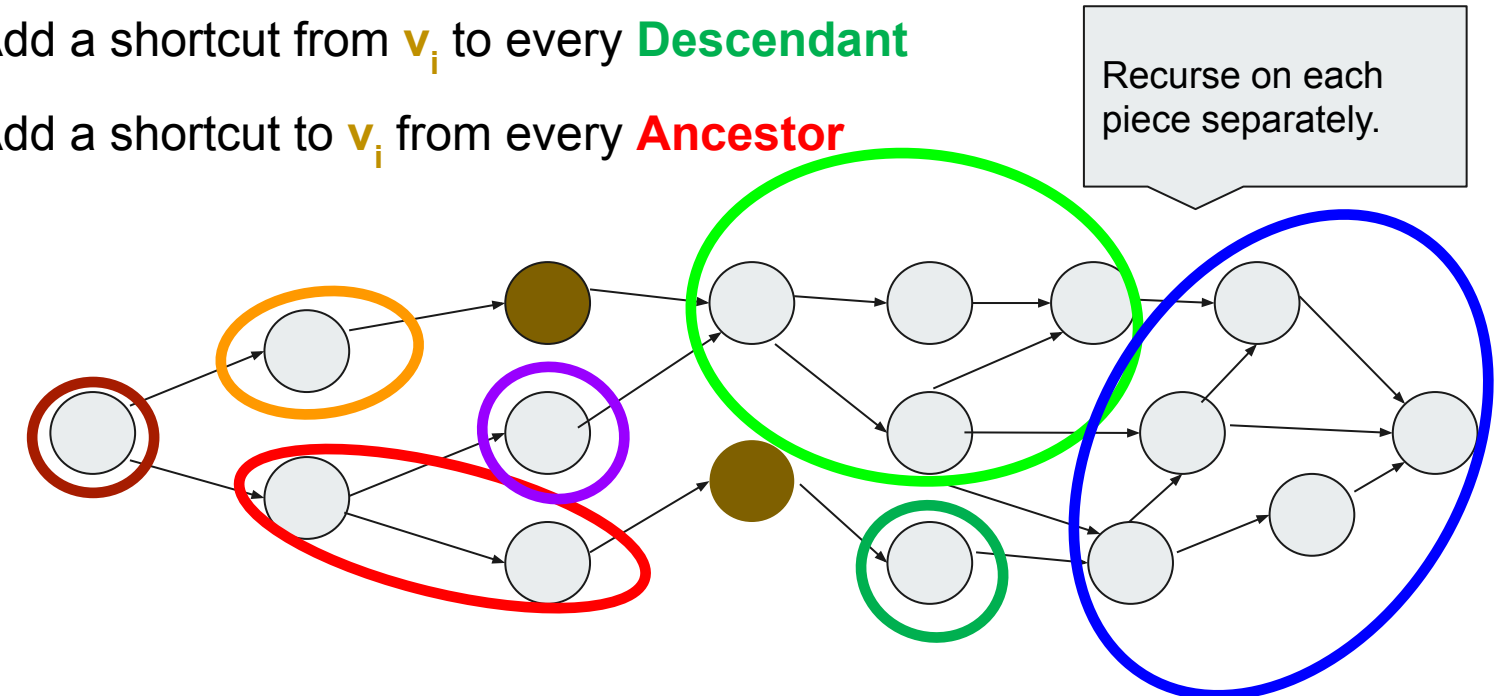
How to partition and recurse

Partition vertices based on whether they had edges added to/from same vertices.

Pick k random vertices $v_1, v_2, \ldots, v_k$.

Add a shortcut from v_i to every **Descendant**

Add a shortcut to v_i from every **Ancestor**



Optimistic Analysis

We start with an incorrect algorithm based on this idea. We can prove the following result:

Optimistic Analysis

Key Lemma Generalization

Conditioned on picking a k ancestors (resp. descendant) the expected # of ancestors (resp. descendants) decrease by 1/(k+1) in expectation.

Optimistic Analysis

Key Lemma Generalization

Conditioned on picking a k ancestors (resp. descendant) the expected # of ancestors (resp. descendants) decrease by 1/(k+1) in expectation.

At every level of recursion, pick k path-related shortcutters for each piece of the path in a subproblem.

Optimistic Analysis

Key Lemma Generalization

Conditioned on picking a k ancestors (resp. descendant) the expected # of ancestors (resp. descendants) decrease by 1/(k+1) in expectation.

At every level of recursion, pick k path-related shortcutters for each piece of the path in a subproblem.

Path splits into k+1 pieces each time a path-related vertex is picked.

After **r** levels, total shortcutted path length = $O((k+1)^r) + \left(\frac{2}{k+1}\right)^r n$.

Yields $\tilde{O}(m)$ time and $\tilde{O}(n^{1/2+o(1)})$ diameter!

Optimistic Analysis

Key Lemma Generalization

Conditioned on picking a k ancestors (resp. descendant) the expected # of ancestors (resp. descendants) decrease by 1/(k+1) in expectation.

Do **r** levels of recursion, where we pick k path-related vertices at each level.

Path splits into k+1 pieces each time a path-related vertex is picked.

Total shortcutted path length = $O((k+1)^r) + \left(\frac{2}{k+1}\right)^r n$.

Yields $\tilde{O}(m)$ time and $\tilde{O}(n^{1/2+o(1)})$ diameter.

Question: What is wrong with this analysis?

Optimistic Analysis

Key Lemma Generalization

Conditioned on picking a k ancestors (resp. descendant) the expected # of ancestors (resp. descendants) decrease by 1/(k+1) in expectation.

Do **r** levels of recursion, where we pick k path-related vertices at each level.

Path splits into k+1 pieces each time a path-related vertex is picked.

Total shortcutted path length = $O((k+1)^r) + \left(\frac{2}{k+1}\right)^r n$.

Yields $\tilde{O}(m)$ time and $\tilde{O}(n^{1/2+o(1)})$ diameter.

Question: What is wrong with this analysis?

How to ensure that we can pick k path related vertices?

Optimistic Analysis

Recall that in each step of Fineman we pick either zero or one related shortcutters to **P.**

Optimistic Analysis

Recall that in each step of Fineman we pick either zero or one related shortcutters to **P.**

Whenever we picked a path-related shortcutter we got our recursion, but when we picked something unrelated *nothing happened* to **P.**

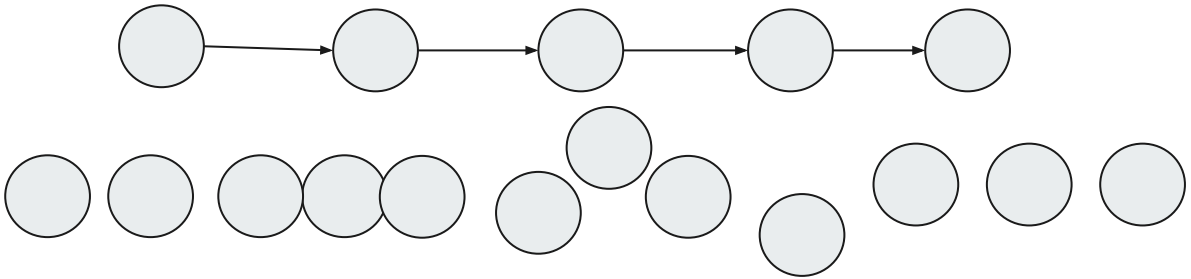
Optimistic Analysis

Recall that in each step of Fineman we pick either zero or one related shortcutters to **P.**

Whenever we picked a path-related shortcutter we got our recursion, but when we picked something unrelated *nothing happened* to **P.**

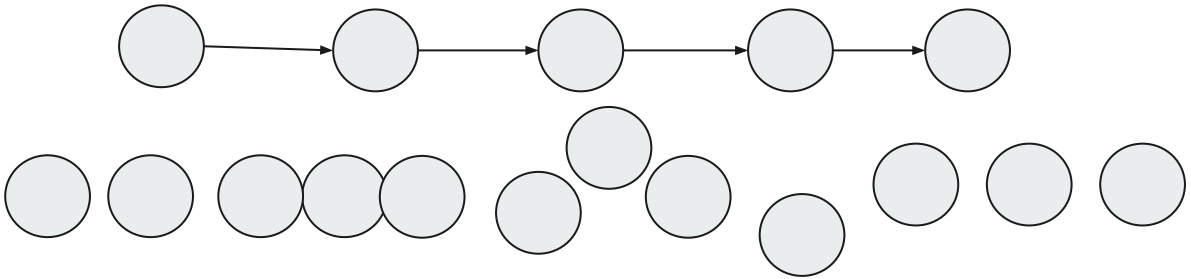
To get similar behavior for this optimistic idea we need to always pick 0 or k path related shortcutters...

Optimistic Analysis



An example bad graph: the “hidden path”. Think length of path is $n^{0.99}$.

Optimistic Analysis



An example bad graph: the “hidden path”. Think length of path is $n^{0.99}$.

If we want to see more than one path-related vertex per round of shortcutting, we need to pick $n^{0.01}$ shortcutters!

Our Algorithm

Is there a way to pick lots of shortcutters in a level of recursion?

Our Algorithm

Key Observation:

As the graph gets partitioned, there are less related pairs of vertices. So when we sample k random vertices, very few are path related in expectation.

Our Algorithm

Key Observation:
As the graph gets partitioned, there are less related pairs of vertices. So when we sample k random vertices, very few are path related in expectation.

Key Observation Part 2:
As our recursion level deepens, we need to *increase* the number of shortcutter vertices we choose.

Our Algorithm

At recursion depth r, include every vertex into shortcutter set S with probability $\tilde{O}(k^r/n)$.

Shortcut to and from all vertices in S.

Our Algorithm

At recursion depth r, include every vertex into shortcutter set S with probability $\tilde{O}(k^r/n)$.

Shortcut to and from all vertices in S.

Partition vertices based shortcuts to/from vertices in S.

Recurse on partitions.

Our Algorithm

At recursion depth r, include every vertex into shortcutter set S with probability $\tilde{O}(k^r/n)$.

Shortcut to and from all vertices in S.

Partition vertices based shortcuts to/from vertices in S.

Recurse on partitions.

Key Lemma:
With high probability, after round r, every vertex has at most n/k^r total ancestors and descendants.

Total Work:
Thus, total work at depth r+1 is $m \times k^{r+1}/n \times n/k^r = mk$ as desired.

Proving Correctness

We use the generalization of the Key Lemma from before:

Key Lemma Generalization

Conditioned on picking a k ancestors (resp. descendant) the expected # of ancestors (resp. descendants) decrease by 1/(k+1) in expectation.

Proving Correctness

We use a generalization of the key corollary from Fineman:

Key Lemma

Conditioned on picking k path-related shortcutters the expected # of path related vertices decrease by 2/(k+1) in expectation.

Further, if we pick k path-related shortcutters the path is divided into at most k+1 recursive subproblems.

Proving Correctness

We use a generalization of the key corollary from Fineman:

Key Lemma

Conditioned on picking t path-related shortcutters the expected # of path related vertices decrease by 2/(t+1) in expectation.

Further, if we pick t path-related shortcutters the path is divided into at most t+1 recursive subproblems.

Resulting recursion is more complicated (t is a random variable), but we can prove $\tilde{O}(n^{1/2+o(1)})$ diameter!

Parallel Implementation

Parallel Implementation

Algorithm described thus far uses full BFS traversals to find hopset: not parallel.

Parallel Implementation

Algorithm described thus far uses full BFS traversals to find hopset: not parallel.

We generalize the parallel implementation in the work of Fineman.

Parallel Implementation

Main idea: do *distance-limited* searches.

Parallel Implementation

Main idea: do *distance-limited* searches.

If we only run BFS searches to depth $\tilde{O}(n^{1/2+o(1)})$, we only pay that much in our parallel depth.

Parallel Implementation

Main idea: do *distance-limited* searches.

If we only run BFS searches to depth $\tilde{O}(n^{1/2+o(1)})$, we only pay that much in our parallel depth.

Goal: reduce the length of a path of length $\tilde{O}(n^{1/2+o(1)})$ by a constant factor in expectation: we can repeat this log number of times to reduce the length of any path.

Parallel Implementation

Unfortunately, distance-limited BFS is not transitive: if X can reach Y and Y can reach Z, X may not be able to reach Z.

Parallel Implementation

Unfortunately, distance-limited BFS is not transitive: if X can reach Y and Y can reach Z, X may not be able to reach Z.

Without taking this into account, distance-limited searches will cut paths without getting decrease in path-relevant vertices.

Parallel Implementation

Unfortunately, distance-limited BFS is not transitive: if X can reach Y and Y can reach Z, X may not be able to reach Z.

Without taking this into account, distance-limited searches will cut paths without getting decrease in path-relevant vertices.

Fix: duplicate vertices at boundary of searches. If vertex x is near the end of a BFS search, place a copy of it in the ancestor/descendant set and another in the unrelated set.

Parallel Implementation

Unfortunately, distance-limited BFS is not transitive: if X can reach Y and Y can reach Z, X may not be able to reach Z.

Without taking this into account, distance-limited searches will cut paths without getting decrease in path-relevant vertices.

Fix: duplicate vertices at boundary of searches. If vertex x is near the end of a BFS search, place a copy of it in the ancestor/descendant set and another in the unrelated set.

Guarantees that there *exists* some copy of a recursively generated subpath that we didnt cut without meaning to.

Conclusion

Conclusion

Nearly linear work parallel reachability algorithm in square root depth.

Conclusion

Nearly linear work parallel reachability algorithm in square root depth.

Almost matches the known combinatorial hopset construction.

Conclusion

Nearly linear work parallel reachability algorithm in square root depth.

Almost matches the known combinatorial hopset construction.

Improved distributed reachability algorithms.

Conclusion

Nearly linear work parallel reachability algorithm in square root depth.

Almost matches the known combinatorial hopset construction.

Improved distributed reachability algorithms.

First to match the known lower bound for small polynomial hop diameter D.

Open Problems

Open Problems

Improve on the known combinatorial hopset construction
OR show lower bound?

Open Problems

Improve on the known combinatorial hopset construction
OR show lower bound?

Remove the $n^{o(1)}$ in the depth? Break $\sqrt{n}$ depth bound via new methods?

Open Problems

Improve on the known combinatorial hopset construction
OR show lower bound?

Remove the $n^{o(1)}$ in the depth? Break $\sqrt{n}$ depth bound via new methods?

Extend the machinery to parallel shortest paths?

Open Problems

Improve on the known combinatorial hopset construction
OR show lower bound?

Remove the $n^{o(1)}$ in the depth? Break $\sqrt{n}$ depth bound via new methods?

Extend the machinery to parallel shortest paths?

Get any sublinear depth, subquadratic work *deterministic* algorithm for reachability?

Questions?