A Deterministic Almost-Tight Distributed Algorithm for Approximating Single-Source Shortest Paths

Monika Henzinger[1] Sebastian Krinninger[2] Danupon Nanongkai[3]

[1]University of Vienna

[2]Max Planck Institute for Informatics

[3]KTH Royal Institute of Technology

STOC 2016

Introduction

The problem:

- Single-source shortest paths
- Undirected graphs
- Positive edge weights $\in \{1, \ldots, poly(n)\}$
- Goal: $(1+\epsilon)$- or $(1+o(1))$-approximation ($\epsilon = 1/polylogn$)

Introduction

The problem:

- Single-source shortest paths
- Undirected graphs
- Positive edge weights $\in \{1, \ldots, poly(n)\}$
- Goal: $(1+\epsilon)$- or $(1+o(1))$-approximation ($\epsilon = 1/polylogn$)

Distributed setting:

- Network modeled as undirected graph
- Processors can communicate with neighbors
- **CONGEST** model: synchronous rounds, message size $O(\log n)$
- Running time = number of rounds
- Goal: every node knows distance to source

Overview

Upper bounds:

exact	$O(n)$	det.	[Bellman-Ford]

Overview

Upper bounds:

exact	$O(n)$	det.	[Bellman-Ford]
$O(\epsilon^{-1} \log \epsilon^{-1})$	$O(n^{1/2+\epsilon} + Diam)$	rand.	[Lenzen, Patt-Shamir '13]

Overview

Upper bounds:

exact	$O(n)$	det.	[Bellman-Ford]
$O(\epsilon^{-1} \log \epsilon^{-1})$	$O(n^{1/2+\epsilon} + Diam)$	rand.	[Lenzen, Patt-Shamir '13]
$1 + \epsilon$	$O(n^{1/2} Diam^{1/4} + Diam)$	rand.	[Nanongkai '14]

Overview

Upper bounds:

exact	$O(n)$	det.	[Bellman-Ford]
$O(\epsilon^{-1} \log \epsilon^{-1})$	$O(n^{1/2+\epsilon} + Diam)$	rand.	[Lenzen, Patt-Shamir '13]
$1 + \epsilon$	$O(n^{1/2} Diam^{1/4} + Diam)$	rand.	[Nanongkai '14]
$1 + o(1)$	$O(n^{1/2+o(1)} + Diam^{1+o(1)})$	det.	**[Our result]**

Overview

Upper bounds:

exact	$O(n)$	det.	[Bellman-Ford]
$O(\epsilon^{-1} \log \epsilon^{-1})$	$O(n^{1/2+\epsilon} + Diam)$	rand.	[Lenzen, Patt-Shamir '13]
$1 + \epsilon$	$O(n^{1/2} Diam^{1/4} + Diam)$	rand.	[Nanongkai '14]
$1 + o(1)$	$O(n^{1/2+o(1)} + Diam^{1+o(1)})$	det.	**[Our result]**

Lower bound: $\Omega(n^{1/2}/\log n + Diam)$ for any reasonable approximation [Das Sarma et al. '11]

Overview

Upper bounds:

exact	$O(n)$	det.	[Bellman-Ford]
$O(\epsilon^{-1} \log \epsilon^{-1})$	$O(n^{1/2+\epsilon} + Diam)$	rand.	[Lenzen, Patt-Shamir '13]
$1 + \epsilon$	$O(n^{1/2} Diam^{1/4} + Diam)$	rand.	[Nanongkai '14]
$1 + o(1)$	$O(n^{1/2+o(1)} + Diam^{1+o(1)})$	det.	**[Our result]**

Lower bound: $\Omega(n^{1/2} / \log n + Diam)$ for any reasonable approximation [Das Sarma et al. '11]

Our approach:

1. Compute overlay network
2. Compute hop set and approximate SSSP on overlay network

Overview

Upper bounds:

exact	$O(n)$	det.	[Bellman-Ford]
$O(\epsilon^{-1} \log \epsilon^{-1})$	$O(n^{1/2+\epsilon} + Diam)$	rand.	[Lenzen, Patt-Shamir '13]
$1 + \epsilon$	$O(n^{1/2} Diam^{1/4} + Diam)$	rand.	[Nanongkai '14]
$1 + o(1)$	$O(n^{1/2+o(1)} + Diam^{1+o(1)})$	det.	**[Our result]**

Lower bound: $\Omega(n^{1/2} / \log n + Diam)$ for any reasonable approximation [Das Sarma et al. '11]

Our approach:

1. Compute overlay network
   Derandomization of "hitting paths" argument at cost of approximation
2. Compute hop set and approximate SSSP on overlay network
   Deterministic hop set using greedy hitting set heuristic

Summary of Results

Theorem (CONGEST)

There is a deterministic distributed algorithm that, on any weighted undirected network, computes $(1+o(1))$*-approximate shortest paths between a given source node s and every other node in* $O(n^{1/2+o(1)} + D^{1+o(1)})$ *rounds.*

Summary of Results

Theorem (CONGEST)

There is a deterministic distributed algorithm that, on any weighted undirected network, computes $(1 + o(1))$-approximate shortest paths between a given source node s and every other node in $O(n^{1/2+o(1)} + D^{1+o(1)})$ rounds.

Theorem (Congested Clique)

There is a deterministic distributed algorithm that, on any weighted congested clique, computes $(1 + o(1))$-approximate shortest paths between a given source node s and every other node in $O(n^{o(1)})$ rounds.

Summary of Results

Theorem (CONGEST)

There is a deterministic distributed algorithm that, on any weighted undirected network, computes $(1 + o(1))$-approximate shortest paths between a given source node s and every other node in $O(n^{1/2+o(1)} + D^{1+o(1)})$ rounds.

Theorem (Congested Clique)

There is a deterministic distributed algorithm that, on any weighted congested clique, computes $(1 + o(1))$-approximate shortest paths between a given source node s and every other node in $O(n^{o(1)})$ rounds.

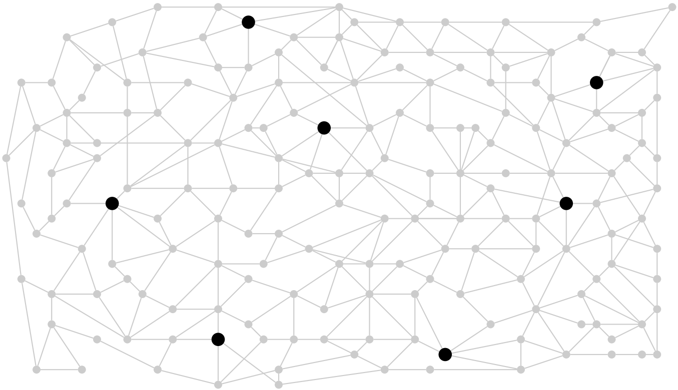
Theorem (Streaming)

There is a deterministic streaming algorithm that, given any weighted undirected graph, computes $(1 + o(1))$-approximate shortest paths between a given source node s and every other node in $O(n^{o(1)} \log W)$ passes with $O(n^{1+o(1)} \log W)$ space.

Computing Overlay Network

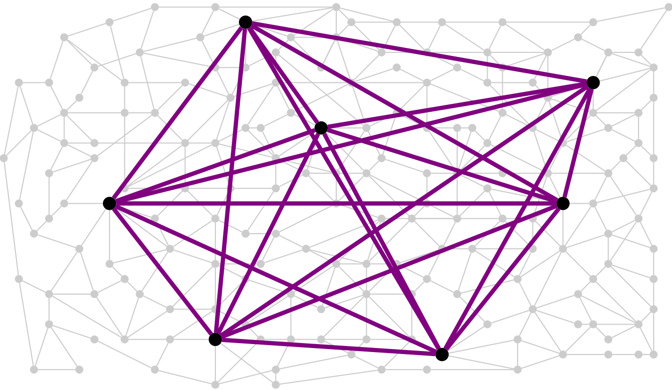
Overlay Network

Overlay Network



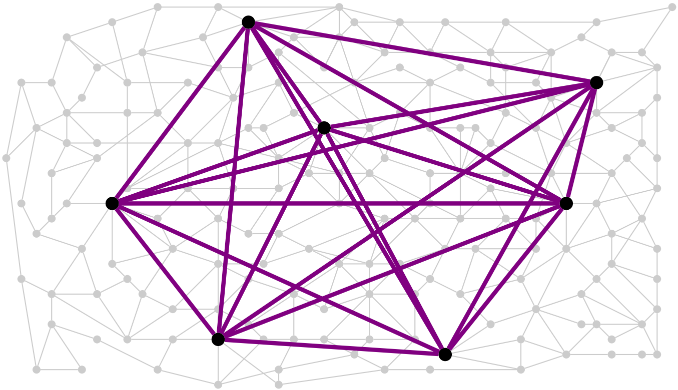
1. Sample $N = O(\sqrt{n} \log n)$ centers (+ source s)
   $\Rightarrow$ Every shortest path with $\geq \sqrt{n}$ edges contains center whp

Overlay Network



1. Sample $N = O(\sqrt{n} \log n)$ centers (+ source s)
   $\Rightarrow$ Every shortest path with $\geq \sqrt{n}$ edges contains center whp
2. For every node: compute approx. shortest paths to centers within $\sqrt{n}$ edges in $O(\sqrt{n}\epsilon^{-1})$ rounds (**source detection** [Lenzen/Peleg '13])

Overlay Network

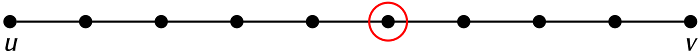


1. Sample $N = O(\sqrt{n} \log n)$ centers (+ source s)
   $\Rightarrow$ Every shortest path with $\geq \sqrt{n}$ edges contains center whp
2. For every node: compute approx. shortest paths to centers within $\sqrt{n}$ edges in $O(\sqrt{n}\epsilon^{-1})$ rounds (**source detection** [Lenzen/Peleg '13])
3. Sufficient to solve SSSP on overlay network using hop set

Derandomization

Property from randomization

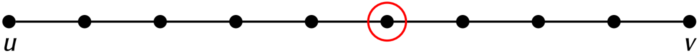
$O(\sqrt{n} \log n)$ centers that hit every shortest path with $\geq \sqrt{n}$ edges



Derandomization

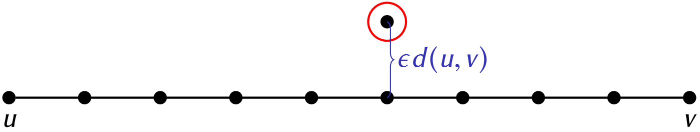
Property from randomization

$O(\sqrt{n} \log n)$ centers that hit every shortest path with $\geq \sqrt{n}$ edges



Deterministic relaxation

$O(\sqrt{n}\epsilon^{-1} \log n)$ centers that **almost** hit every path with $\geq \sqrt{n}$ edges



Ruling sets for deterministic centers

First: Explanation for unweighted graphs

Ruling sets for deterministic centers

First: Explanation for unweighted graphs

Definition

(α, β)-**ruling set** R of U is a set of **rulers** such that

- Every pair of rulers in R is at distance $\geq \alpha$ from each other
- Every node in U has a ruler in R at distance $\leq \beta$

Ruling sets for deterministic centers

First: Explanation for unweighted graphs

Definition

(α, β)-**ruling set** R of U is a set of **rulers** such that

- Every pair of rulers in R is at distance $\geq \alpha$ from each other
- Every node in U has a ruler in R at distance $\leq \beta$

Lemma ([Goldberg et al. '88])

A $(c, c \log n)$*-ruling set can be computed in* $O(c \log n)$ *rounds.*

Ruling sets for deterministic centers

First: Explanation for unweighted graphs

Definition

(α, β)-**ruling set** R of U is a set of **rulers** such that

- Every pair of rulers in R is at distance $\geq \alpha$ from each other
- Every node in U has a ruler in R at distance $\leq \beta$

Lemma ([Goldberg et al. '88])

A $(c, c \log n)$-ruling set can be computed in $O(c \log n)$ rounds.

Our setting:

- U = all nodes v with $|Ball(v, \sqrt{n})| \geq \sqrt{n}$
- $c = \epsilon \sqrt{n}$

Ruling sets for deterministic centers

First: Explanation for unweighted graphs

Definition

(α, β)-**ruling set** R of U is a set of **rulers** such that

- Every pair of rulers in R is at distance $\geq \alpha$ from each other
- Every node in U has a ruler in R at distance $\leq \beta$

Lemma ([Goldberg et al. '88])

A $(c, c \log n)$-ruling set can be computed in $O(c \log n)$ rounds.

Our setting:

- U = all nodes v with $|Ball(v, \sqrt{n})| \geq \sqrt{n}$
- $c = \epsilon \sqrt{n}$
- Any shortest $u - v$ path with $\geq \sqrt{n}$ edges: ruler in distance $\leq \epsilon dist(u, v)$

Ruling sets for deterministic centers

First: Explanation for unweighted graphs

Definition

(α, β)-**ruling set** R of U is a set of **rulers** such that

- Every pair of rulers in R is at distance $\geq \alpha$ from each other
- Every node in U has a ruler in R at distance $\leq \beta$

Lemma ([Goldberg et al. '88])

A $(c, c \log n)$-ruling set can be computed in $O(c \log n)$ rounds.

Our setting:

- U = all nodes v with $|Ball(v, \sqrt{n})| \geq \sqrt{n}$
- $c = \epsilon\sqrt{n}$
- Any shortest $u - v$ path with $\geq \sqrt{n}$ edges: ruler in distance $\leq \epsilon dist(u, v)$
- Uniquely assign $\epsilon\sqrt{n}/2$ nodes to every ruler $\Rightarrow |T| \leq 2\sqrt{n}/\epsilon$

Ruling sets for deterministic centers

First: Explanation for unweighted graphs

Definition

(α, β)-**ruling set** R of U is a set of **rulers** such that

- Every pair of rulers in R is at distance $\geq \alpha$ from each other
- Every node in U has a ruler in R at distance $\leq \beta$

Lemma ([Goldberg et al. '88])

A $(c, c \log n)$-ruling set can be computed in $O(c \log n)$ rounds.

Our setting:

- U = all nodes v with $|Ball(v, \sqrt{n})| \geq \sqrt{n}$
- $c = \epsilon \sqrt{n}$
- Any shortest $u - v$ path with $\geq \sqrt{n}$ edges: ruler in distance $\leq \epsilon dist(u, v)$
- Uniquely assign $\epsilon \sqrt{n}/2$ nodes to every ruler $\Rightarrow |T| \leq 2\sqrt{n}/\epsilon$

Crucial: "weight = #edges" in unweighted graphs

Weighted graphs

Goal: Make graph locally "look unweighted" s.t. weight $\approx$ #hops

Weighted graphs

Goal: Make graph locally "look unweighted" s.t. weight ≈ #hops

Well-known weight rounding [Bernstein '09/13, Madry '10, ...]

G_i: round up edge weights to next multiple of $\epsilon 2^i / \sqrt{n}$ ($\forall i = 1$ to $\log(nW)$)
$(1+\epsilon)$-approximation of shortest paths with $\sqrt{n}$ edges and weight $2^i \ldots 2^{i+1}$
Intuition: "weight $\leq$ #edges"

Weighted graphs

Goal: Make graph locally "look unweighted" s.t. weight $\approx$ #hops

Well-known weight rounding [Bernstein '09/13, Madry '10, ...]

G_i: round up edge weights to next multiple of $\epsilon 2^i / \sqrt{n}$ ($\forall i = 1$ to $\log(nW)$)
$(1+\epsilon)$-approximation of shortest paths with $\sqrt{n}$ edges and weight $2^i \ldots 2^{i+1}$
Intuition: "weight $\leq$ #edges"

Not enough: we also want "#edges $\leq$ weight"

Weighted graphs

Goal: Make graph locally "look unweighted" s.t. weight $\approx$ #hops

Well-known weight rounding [Bernstein '09/13, Madry '10, ...]

G_i: round up edge weights to next multiple of $\epsilon 2^i / \sqrt{n}$ ($\forall i = 1$ to $\log(nW)$)
$(1+\epsilon)$-approximation of shortest paths with $\sqrt{n}$ edges and weight $2^i \dots 2^{i+1}$
Intuition: "weight $\leq$ #edges"

Not enough: we also want "#edges $\leq$ weight"

Type $t(v)$ of node v: minimum i such that $|Ball_{G_i}(v, (2+\epsilon)\sqrt{n})| \geq \epsilon\sqrt{n}$

Intuition: type gives scale s.t. local neighborhood "looks unweighted"

Weighted graphs

Goal: Make graph locally “look unweighted” s.t. weight ≈ #hops

Well-known weight rounding [Bernstein ’09/13, Madry ’10, . . .]

G_i: round up edge weights to next multiple of $\epsilon 2^i / \sqrt{n}$ ($\forall i = 1$ to $\log(nW)$)
$(1+\epsilon)$-approximation of shortest paths with $\sqrt{n}$ edges and weight $2^i \dots 2^{i+1}$
Intuition: “weight ≤ #edges”

Not enough: we also want “#edges ≤ weight”

Type $t(v)$ of node v: minimum i such that $|Ball_{G_i}(v, (2+\epsilon)\sqrt{n})| \geq \epsilon\sqrt{n}$

Intuition: type gives scale s.t. local neighborhood "looks unweighted"

Lemma

Every path π with $\sqrt{n}$ edges contains a node v such that $2^{t(v)} \leq 2\epsilon w(\pi)$.

Weighted graphs

Goal: Make graph locally "look unweighted" s.t. weight ≈ #hops

Well-known weight rounding [Bernstein '09/13, Madry '10, ...]

G_i: round up edge weights to next multiple of $\epsilon 2^i / \sqrt{n}$ ($\forall i = 1$ to $\log(nW)$)
$(1+\epsilon)$-approximation of shortest paths with $\sqrt{n}$ edges and weight $2^i \dots 2^{i+1}$
Intuition: "weight ≤ #edges"

Not enough: we also want "#edges ≤ weight"

Type $t(v)$ of node v: minimum i such that $|Ball_{G_i}(v, (2+\epsilon)\sqrt{n})| \geq \epsilon \sqrt{n}$

Intuition: type gives scale s.t. local neighborhood "looks unweighted"

Lemma

Every path π with $\sqrt{n}$ edges contains a node v such that $2^{t(v)} \leq 2\epsilon w(\pi)$.

⇒ Determine centers by computing ruling set for all type classes

Computing Hop Set on Overlay Network

Hop Sets

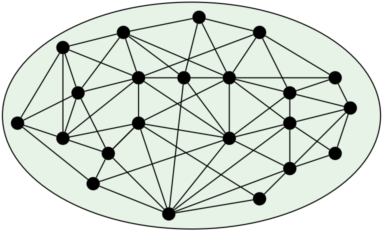
Definition

An (h, ϵ)-hop set is a set of weighted edges F such that, for all pairs of nodes u and v, in the 'shortcut graph' $G \cup F$ there is a path from u to v with **at most h edges** of weight at most $(1+\epsilon)dist(u, v)$.

Hop Sets

Definition

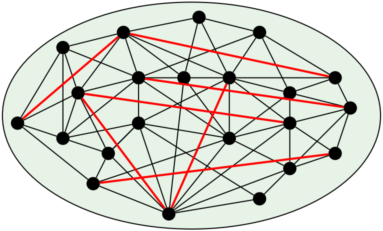
An (h, ϵ)-hop set is a set of weighted edges F such that, for all pairs of nodes u and v, in the 'shortcut graph' $G \cup F$ there is a path from u to v with **at most h edges** of weight at most $(1 + \epsilon)dist(u, v)$.



Hop Sets

Definition

An (h, ϵ)-hop set is a set of weighted edges F such that, for all pairs of nodes u and v, in the 'shortcut graph' $G \cup F$ there is a path from u to v with **at most h edges** of weight at most $(1 + \epsilon)dist(u, v)$.

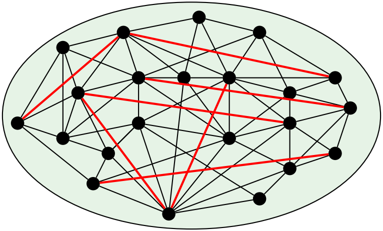


Application: SSSP up to small #edges can be done fast in overlay network

Hop Sets

Definition

An (h, ϵ)-hop set is a set of weighted edges F such that, for all pairs of nodes u and v, in the 'shortcut graph' $G \cup F$ there is a path from u to v with **at most h edges** of weight at most $(1+\epsilon)dist(u, v)$.

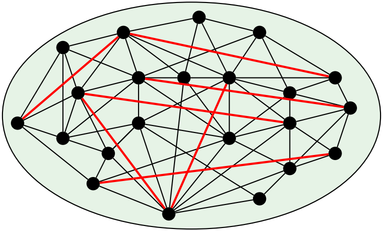


Application: SSSP up to small #edges can be done fast in overlay network
A: $(\log^{O(1)} n, \epsilon)$-hop set of size $n^{1+o(1)}$ [Cohen '94]
B: $(n^{o(1)}, \epsilon)$-hop set of size $n^{1+o(1)}$ [Bernstein '09]
C: (n^{α}, ϵ)-hop set of size $O(n)$ [Miller et al. '15]

Hop Sets

Definition

An (h, ϵ)-hop set is a set of weighted edges F such that, for all pairs of nodes u and v, in the 'shortcut graph' $G \cup F$ there is a path from u to v with **at most h edges** of weight at most $(1 + \epsilon) dist(u, v)$.



Application: SSSP up to small #edges can be done fast in overlay network

A: $(\log^{O(1)} n, \epsilon)$-hop set of size $n^{1+o(1)}$ [Cohen '94]

B: $(n^{o(1)}, \epsilon)$-hop set of size $n^{1+o(1)}$ [Bernstein '09]

C: (n^{α}, ϵ)-hop set of size $O(n)$ [Miller et al. '15]

Our contribution: Fast computation of **B** on overlay network

Hop Set Based on Clusters [Thorup/Zwick '01]

$V = A_0 \supseteq A_1 \supseteq \cdots \supseteq A_k = \emptyset$ where node of A_i goes to A_{i+1} with probability $1/n^{1/k}$

v has **priority** i iff $v \in A_i \setminus A_{i+1}$

Hop Set Based on Clusters [Thorup/Zwick '01]

$V = A_0 \supseteq A_1 \supseteq \cdots \supseteq A_k = \emptyset$ where node of A_i goes to A_{i+1} with probability $1/n^{1/k}$

v has **priority** i iff $v \in A_i \setminus A_{i+1}$

For every node u of priority i:

$Cluster(v) = \{u \in V \mid dist(u, v) < dist(u, A_{i+1})\}$

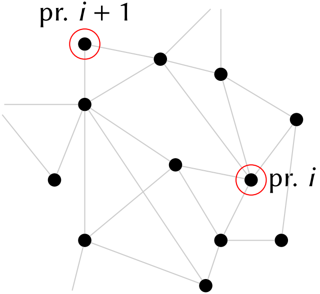
Hop Set Based on Clusters [Thorup/Zwick '01]

$V = A_0 \supseteq A_1 \supseteq \cdots \supseteq A_k = \emptyset$ where node of A_i goes to A_{i+1} with probability $1/n^{1/k}$

v has **priority** i iff $v \in A_i \setminus A_{i+1}$

For every node u of priority i:

$Cluster(v) = \{u \in V \mid dist(u, v) < dist(u, A_{i+1})\}$



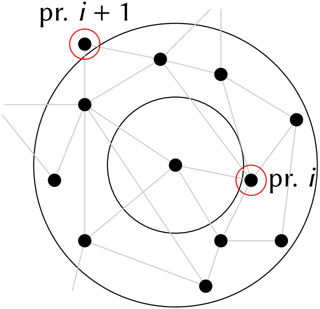
Hop Set Based on Clusters [Thorup/Zwick '01]

$V = A_0 \supseteq A_1 \supseteq \cdots \supseteq A_k = \emptyset$ where node of A_i goes to A_{i+1} with probability $1/n^{1/k}$

v has **priority** i iff $v \in A_i \setminus A_{i+1}$

For every node u of priority i:

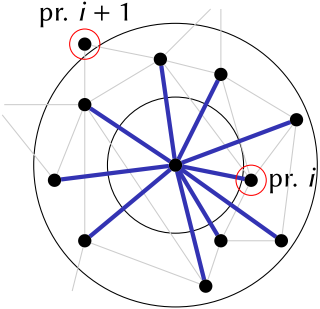
$Cluster(v) = \{u \in V \mid dist(u, v) < dist(u, A_{i+1})\}$



Hop Set Based on Clusters [Thorup/Zwick '01]

$V = A_0 \supseteq A_1 \supseteq \cdots \supseteq A_k = \emptyset$ where node of A_i goes to A_{i+1} with probability $1/n^{1/k}$

v has **priority** i iff $v \in A_i \setminus A_{i+1}$



For every node u of priority i:

$Cluster(v) = \{u \in V \mid dist(u, v) < dist(u, A_{i+1})\}$

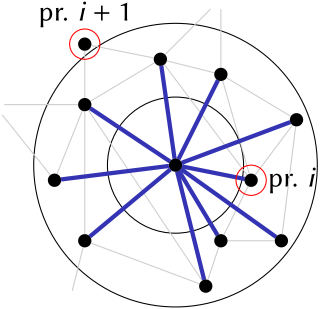
Hop set:

- $(u, v) \in F$ iff $u \in Cluster(v)$
- $w(u, v) = dist_G(u, v)$

Hop Set Based on Clusters [Thorup/Zwick '01]

$V = A_0 \supseteq A_1 \supseteq \cdots \supseteq A_k = \emptyset$ where node of A_i goes to A_{i+1} with probability $1/n^{1/k}$

v has **priority** i iff $v \in A_i \setminus A_{i+1}$



For every node u of priority i:

$Cluster(v) = \{u \in V \mid dist(u,v) < dist(u, A_{i+1})\}$

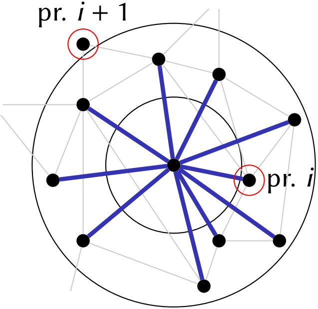
Hop set:

- $(u,v) \in F$ iff $u \in Cluster(v)$
- $w(u,v) = dist_G(u,v)$
- Guarantee: $((4/\epsilon)^k, \epsilon)$-hop set [Bernstein '09, Thorup/Zwick '06]
- Expected size: $O(kn^{1+1/k})$ [Thorup/Zwick '01]

Hop Set Based on Clusters [Thorup/Zwick '01]

$V = A_0 \supseteq A_1 \supseteq \cdots \supseteq A_k = \emptyset$ where node of A_i goes to A_{i+1} with probability $1/n^{1/k}$

v has **priority** i iff $v \in A_i \setminus A_{i+1}$



For every node u of priority i:

$Cluster(v) = \{u \in V \mid dist(u,v) < dist(u, A_{i+1})\}$

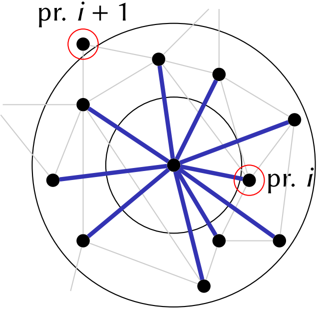
Hop set:

- $(u,v) \in F$ iff $u \in Cluster(v)$
- $w(u,v) = dist_G(u,v)$
- Guarantee: $((4/\epsilon)^k, \epsilon)$-hop set [Bernstein '09, Thorup/Zwick '06]
- Expected size: $O(kn^{1+1/k})$ [Thorup/Zwick '01]
- With $k = \sqrt{\log n}/\sqrt{\log 4/\epsilon}$: $(n^{o(1)}, \epsilon)$-hop set of size $n^{1+o(1)}$

Hop Set Based on Clusters [Thorup/Zwick '01]

$V = A_0 \supseteq A_1 \supseteq \cdots \supseteq A_k = \emptyset$ where node of A_i goes to A_{i+1} with probability $1/n^{1/k}$

v has **priority** i iff $v \in A_i \setminus A_{i+1}$



For every node u of priority i:

$Cluster(v) = \{u \in V \mid dist(u, v) < dist(u, A_{i+1})\}$

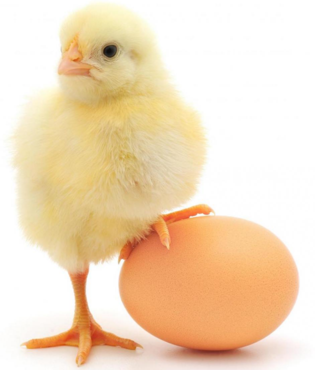
Hop set:

- $(u, v) \in F$ iff $u \in Cluster(v)$
- $w(u, v) = dist_G(u, v)$
- Guarantee: $((4/\epsilon)^k, \epsilon)$-hop set [Bernstein '09, Thorup/Zwick '06]
- Expected size: $O(kn^{1+1/k})$ [Thorup/Zwick '01]
- With $k = \sqrt{\log n}/\sqrt{\log 4/\epsilon}$: $(n^{o(1)}, \epsilon)$-hop set of size $n^{1+o(1)}$
- **Derandomization:** choose A_{i+1} from A_i by greedy hitting set heuristic *(Sequential, but affordable in overlay network)*

Chicken-Egg Problem?

1. Goal: Faster SSSP via hop set
2. Compute hop set by computing clusters
3. Computing clusters at least as hard as SSSP

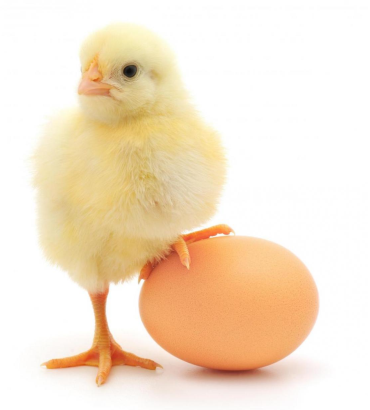
⇒ Back at problem we wanted to solve initially?



Chicken-Egg Problem?

1. Goal: Faster SSSP via hop set
2. Compute hop set by computing clusters
3. Computing clusters at least as hard as SSSP

$\Rightarrow$ Back at problem we wanted to solve initially?



No! Iterative computation starting with

- SSSP up to small #hops is cheap in overlay network
- Clusters up to small #hops provide sufficient shortcutting to make progress in each iteration

Computing $(n^{o(1)}, \epsilon)$-hop set

Iterative computation

In each iteration number of hops is reduced by a factor of $n^{1/k}$

Computing $(n^{o(1)}, \epsilon)$-hop set

Iterative computation

In each iteration number of hops is reduced by a factor of $n^{1/k}$

Algorithm:

for $i = 1$ **to** k **do**

$$H_i = G \cup \bigcup_{1 \le j \le i-1} F_j$$

Compute clusters with k priorities in H_i up to $n^{2/k}$ hops

$$F_i = \{(u, v) \mid u \in Cluster(v)\}$$

end

return $F = \bigcup_{1 \le i \le k} F_i$

Computing $(n^{o(1)}, \epsilon)$-hop set

Iterative computation

In each iteration number of hops is reduced by a factor of $n^{1/k}$

Algorithm:

for $i = 1$ **to** k **do**

$$H_i = G \cup \bigcup_{1 \le j \le i-1} F_j$$

Compute clusters with k priorities in H_i up to $n^{2/k}$ hops

$$F_i = \{(u, v) \mid u \in Cluster(v)\}$$

end

return $F = \bigcup_{1 \le i \le k} F_i$

Error amplification: $(1 + \epsilon')^k \le (1 + \epsilon)$ for $\epsilon' = 1/(2\epsilon \log n)$

Computing $(n^{o(1)}, \epsilon)$-hop set

Iterative computation
In each iteration number of hops is reduced by a factor of $n^{1/k}$

Algorithm:

for $i = 1$ **to** k **do**

$$H_i = G \cup \bigcup_{1 \le j \le i-1} F_j$$

Compute clusters with k priorities in H_i up to $n^{2/k}$ hops

$$F_i = \{(u, v) \mid u \in Cluster(v)\}$$

end

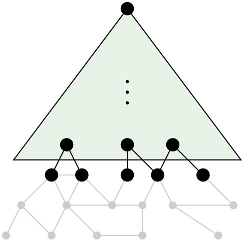
return $F = \bigcup_{1 \le i \le k} F_i$

Error amplification: $(1 + \epsilon')^k \le (1 + \epsilon)$ for $\epsilon' = 1/(2\epsilon \log n)$

Omitted detail: weighted graphs, use rounding technique

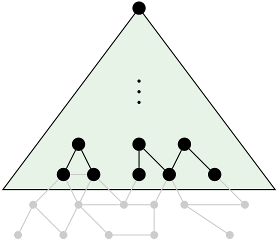
Computing Hop Set on Overlay Network

Shortest paths from source *s* **up to distance** d**:**



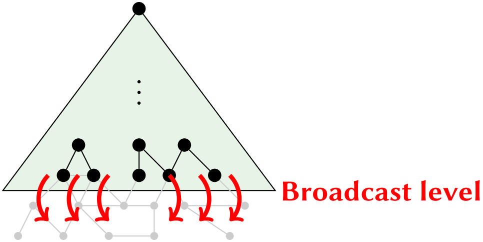
Computing Hop Set on Overlay Network

Shortest paths from source s **up to distance** d:



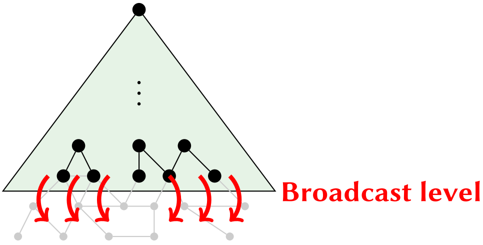
Computing Hop Set on Overlay Network

Shortest paths from source *s* **up to distance** *d***:**



Computing Hop Set on Overlay Network

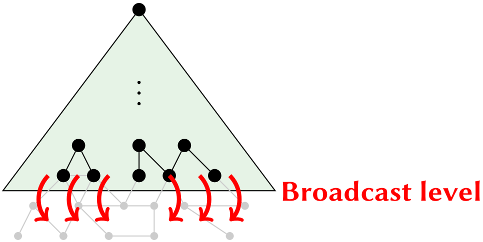
Shortest paths from source s **up to distance** d**:**



d iterations, each $O(Diam + N_\ell)$ rounds where N_ℓ = #nodes at level ℓ
Running time: $O(d \cdot Diam + \sum_{l \leq d} N_\ell) = O(d \cdot Diam + N)$

Computing Hop Set on Overlay Network

Shortest paths from source s **up to distance** d**:**



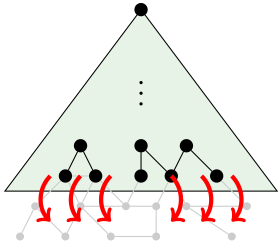
d iterations, each $O(Diam + N_\ell)$ rounds where N_ℓ = #nodes at level ℓ

Running time: $O(d \cdot Diam + \sum_{l \leq d} N_\ell) = O(d \cdot Diam + N)$

Computing clusters: $\widetilde{O}(n^{1/k} \cdot Diam + \sum_v |Cluster(v)|) = \widetilde{O}(n^{1/k} \cdot Diam + N^{1+1/k})$

Computing Hop Set on Overlay Network

Shortest paths from source s **up to distance d:**



d iterations, each $O(Diam + N_\ell)$ rounds where N_ℓ = #nodes at level ℓ
Running time: $O(d \cdot Diam + \sum_{l \leq d} N_\ell) = O(d \cdot Diam + N)$

Computing clusters: $\widetilde{O}(n^{1/k} \cdot Diam + \sum_{v} |Cluster(v)|) = \widetilde{O}(n^{1/k} \cdot Diam + N^{1+1/k})$

$\Rightarrow$ Hop Set and approximate SSSP: $O(n^{1/2+o(1)} + Diam^{1+o(1)})$ $(N \approx \sqrt{n})$

Conclusion

Main contributions:

- Almost tight algorithm
- Deterministic overlay network and deterministic hop set

Conclusion

Main contributions:

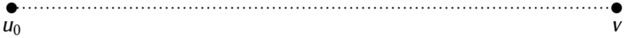
- Almost tight algorithm
- Deterministic overlay network and deterministic hop set

Open problems:

- $n^{o(1)} \rightarrow \log^{O(1)} n$
  Better hop set?
- Improve dependence on ϵ
- $O(n)$ rounds optimal for exact SSSP?

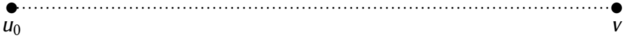
Example: $(n^{1/2+o(1)}, \epsilon)$-hop set

Case 1: $dist(u_0, v) \leq n^{1/2+1/k}/\epsilon$



Example: $(n^{1/2+o(1)}, \epsilon)$-hop set

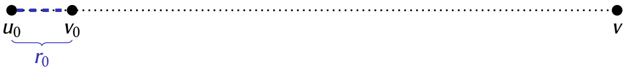
Case 2: $dist(u_0, v) > n^{1/2+1/k}/\epsilon$



Example: $(n^{1/2+o(1)}, \epsilon)$-hop set

Case 2: $dist(u_0, v) > n^{1/2+1/k}/\epsilon$

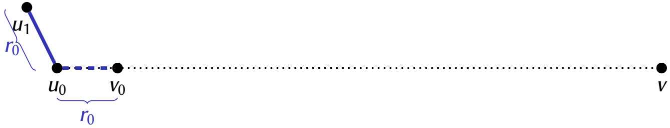
$r_0 = n^{1/2}$



Example: $(n^{1/2+o(1)}, \epsilon)$-hop set

Case 2: $dist(u_0, v) > n^{1/2+1/k}/\epsilon$

$r_0 = n^{1/2}$



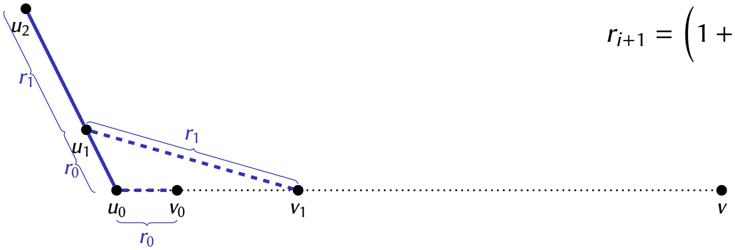
For every node u of priority i and every node v, either $(u, v) \in H$, or $\exists u'$ of priority $i + 1$ s. t. $dist(u, u') \leq dist(u, v)$.

Example: $(n^{1/2+o(1)}, \epsilon)$-hop set

Case 2: $dist(u_0, v) > n^{1/2+1/k}/\epsilon$

$$r_0 = n^{1/2}$$

$$r_{i+1} = \left(1 + \frac{2}{\epsilon}\right) \sum_{0 \le j \le i} r_j$$



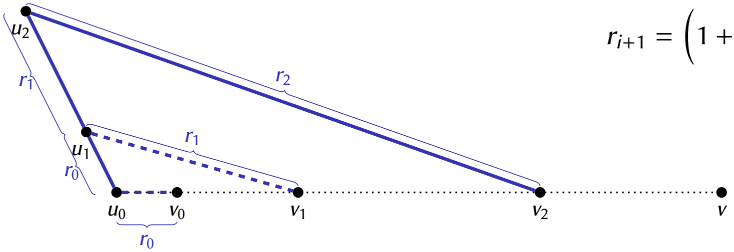
For every node u of priority i and every node v, either $(u, v) \in H$, or $\exists u'$ of priority $i + 1$ s. t. $dist(u, u') \le dist(u, v)$.

Example: $(n^{1/2+o(1)}, \epsilon)$-hop set

Case 2: $dist(u_0, v) > n^{1/2+1/k}/\epsilon$

$$r_0 = n^{1/2}$$

$$r_{i+1} = \left(1 + \frac{2}{\epsilon}\right) \sum_{0 \le j \le i} r_j$$



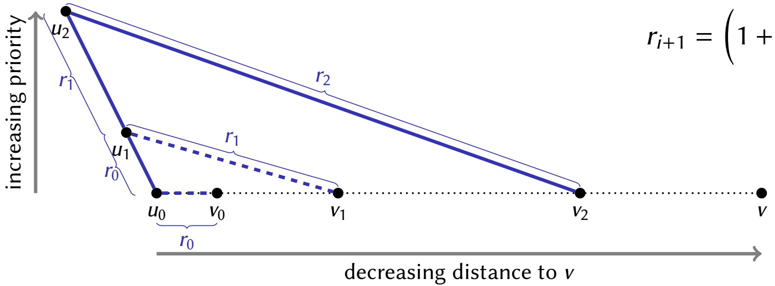
For every node u of priority i and every node v, either $(u, v) \in H$, or $\exists u'$ of priority $i+1$ s. t. $dist(u, u') \le dist(u, v)$.

Example: $(n^{1/2+o(1)}, \epsilon)$-hop set

Case 2: $dist(u_0, v) > n^{1/2+1/k}/\epsilon$

$$r_0 = n^{1/2}$$

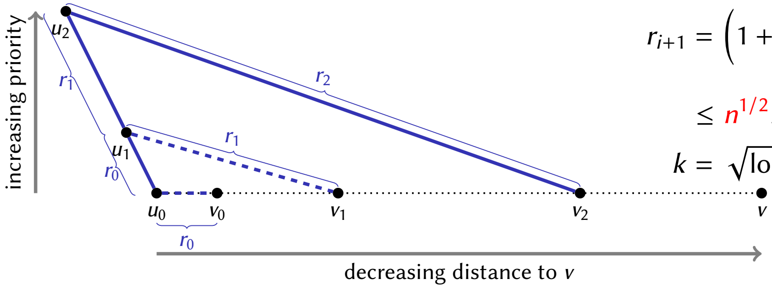
$$r_{i+1} = \left(1 + \frac{2}{\epsilon}\right) \sum_{0 \le j \le i} r_j$$



For every node u of priority i and every node v, either $(u, v) \in H$, or $\exists u'$ of priority $i + 1$ s. t. $dist(u, u') \le dist(u, v)$.

Example: $(n^{1/2+o(1)}, \epsilon)$-hop set

Case 2: $dist(u_0, v) > n^{1/2+1/k}/\epsilon$



$$r_0 = n^{1/2}$$

$$r_{i+1} = \left(1 + \frac{2}{\epsilon}\right) \sum_{0 \le j \le i} r_j$$

$$\le n^{1/2} n^{1/k}$$

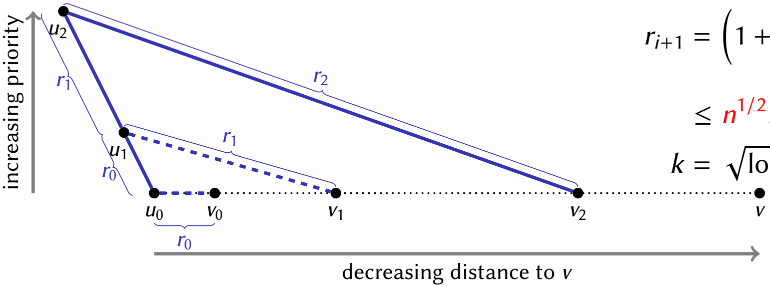
$$k = \sqrt{\log n} / \sqrt{\log 4/\epsilon}$$

For every node u of priority i and every node v, either $(u, v) \in H$, or $\exists u'$ of priority $i + 1$ s. t. $dist(u, u') \le dist(u, v)$.

$$\textit{Weight} \le (1 + \epsilon) dist(u_0, v)$$

Example: $(n^{1/2+o(1)}, \epsilon)$-hop set

Case 2: $dist(u_0, v) > n^{1/2+1/k}/\epsilon$



$$r_0 = n^{1/2}$$

$$r_{i+1} = \left(1 + \frac{2}{\epsilon}\right) \sum_{0 \le j \le i} r_j \le n^{1/2} n^{1/k}$$

$$k = \sqrt{\log n} / \sqrt{\log 4/\epsilon}$$

For every node u of priority i and every node v, either $(u, v) \in H$, or $\exists u'$ of priority $i+1$ s. t. $dist(u, u') \le dist(u, v)$.

$$\textit{Weight} \le (1+\epsilon) dist(u_0, v)$$

$$\#\textit{Edges} \le \frac{k \cdot dist(u, v)}{n^{1/2}} \le \frac{k \cdot n}{n^{1/2}} = kn^{1/2}$$