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

Monika Henzinger\textsuperscript{1}  Sebastian Krinninger\textsuperscript{2}  Danupon Nanongkai\textsuperscript{3}

\textsuperscript{1}University of Vienna
\textsuperscript{2}Max Planck Institute for Informatics
\textsuperscript{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 ($\epsilon = 1/polylog n$)

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
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 ($\epsilon = 1/polylog n$)

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. \[\text{[Bellman-Ford]}\]

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

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

### Upper bounds:

<table>
<thead>
<tr>
<th>Type</th>
<th>Exact Bound</th>
<th>Approximate Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$O(n)$</td>
<td>$O(n^{1/2+\epsilon} + Diam)$</td>
</tr>
<tr>
<td></td>
<td>$O(\epsilon^{-1} \log \epsilon^{-1})$</td>
<td>$O(n^{1/2+\epsilon} + Diam)$</td>
</tr>
<tr>
<td>det.</td>
<td>[Bellman-Ford]</td>
<td></td>
</tr>
<tr>
<td>rand.</td>
<td>[Lenzen, Patt-Shamir ’13]</td>
<td></td>
</tr>
<tr>
<td>Our result</td>
<td></td>
<td>$\Omega\left(n^{1/2}/\log n + Diam\right)$</td>
</tr>
</tbody>
</table>
Overview

Upper bounds:

<table>
<thead>
<tr>
<th>Exact</th>
<th>O(n)</th>
<th>Deterministic</th>
<th>[Bellman-Ford]</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(\epsilon^{-1} \log \epsilon^{-1})$</td>
<td>$O(n^{1/2+\epsilon} + Diam)$</td>
<td>Random</td>
<td>[Lenzen, Patt-Shamir '13]</td>
</tr>
<tr>
<td>$1 + \epsilon$</td>
<td>$O(n^{1/2} Diam^{1/4} + Diam)$</td>
<td>Random</td>
<td>[Nanongkai '14]</td>
</tr>
</tbody>
</table>
Overview

Upper bounds:

- exact
  - $O(n)$
  - $O(\epsilon^{-1} \log \epsilon^{-1})$
  - $1 + \epsilon$
  - $1 + o(1)$

- $O(n^{1/2+\epsilon} + Diam)$
- $O(n^{1/2} Diam^{1/4} + Diam)$
- $O(n^{1/2+o(1)} + Diam^{1+o(1)})$

det. [Bellman-Ford]
rand. [Lenzen, Patt-Shamir ’13]
rand. [Nanongkai ’14]

Our approach:
1. Compute overlay network
2. Derandomization of “hitting paths” argument at cost of approximation
3. Compute hop set and approximate SSSP on overlay network
Overview

Upper bounds:

<table>
<thead>
<tr>
<th>Exact</th>
<th>$O(n)$</th>
<th>Deterministic</th>
<th>[Bellman-Ford]</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(\epsilon^{-1} \log \epsilon^{-1})$</td>
<td>$O(n^{1/2+\epsilon} + Diam)$</td>
<td>Random</td>
<td>[Lenzen, Patt-Shamir ’13]</td>
</tr>
<tr>
<td>$1+\epsilon$</td>
<td>$O(n^{1/2}Diam^{1/4} + Diam)$</td>
<td>Random</td>
<td>[Nanongkai ’14]</td>
</tr>
<tr>
<td>$1+o(1)$</td>
<td>$O(n^{1/2+o(1)} + Diam^{1+o(1)})$</td>
<td>Deterministic</td>
<td>[Our result]</td>
</tr>
</tbody>
</table>

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

Upper bounds:

<table>
<thead>
<tr>
<th>Type</th>
<th>Formulation</th>
<th>Deterministic</th>
<th>Randomized</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact</td>
<td>$O(n)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$O(\varepsilon^{-1} \log \varepsilon^{-1})$</td>
<td>$O(n^{1/2+\varepsilon} + \text{Diam})$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$1 + \varepsilon$</td>
<td>$O(n^{1/2} \text{Diam}^{1/4} + \text{Diam})$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$1 + o(1)$</td>
<td>$O(n^{1/2+o(1)} + \text{Diam}^{1+o(1)})$</td>
<td>Deterministic</td>
<td></td>
</tr>
</tbody>
</table>

[Our result]

[Lenzen, Patt-Shamir ’13]

[Our result]

Lower bound: $\Omega(n^{1/2}/\log n + \text{Diam})$ 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:** $O(n)$
- $O(\varepsilon^{-1} \log \varepsilon^{-1})$: $O(n^{1/2+\varepsilon} + Diam)$
- $1 + \varepsilon$: $O(n^{1/2} Diam^{1/4} + Diam)$
- $1 + o(1)$: $O(n^{1/2+o(1)} + Diam^{1+o(1)})$

### Deterministic
- [Bellman-Ford]

### Randomized
- [Lenzen, Patt-Shamir ’13]
- [Nanongkai ’14]
- [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 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
Sample

\[ N = O(\sqrt{n \log n}) \] centers (+ source(s))

\[ \Rightarrow \]

Every shortest path with \( \geq \sqrt{n} \) edges contains center whp

For every node: compute approx. shortest paths to centers within \( \sqrt{n} \) edges in \( O(\sqrt{n}/\log n) \) rounds (source detection [Lenzen/Peleg '13])

Sufficient to solve SSSP on overlay network using hop set
Sample $N = O(\sqrt{n \log n})$ centers (+ source $s$)
$
\Rightarrow$ Every shortest path with $\geq \sqrt{n}$ edges contains center whp
Sample $N = O(\sqrt{n \log n})$ centers (+ source $s$)

⇒ Every shortest path with $\geq \sqrt{n}$ edges contains center whp

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

For every node: compute approx. shortest paths to centers within $\sqrt{n}$ edges in $O(\sqrt{n e^{-1}})$ rounds (source detection [Lenzen/Peleg ’13])

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 \textbf{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**

\((\alpha, \beta)\)-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**

$(\alpha, \beta)$-**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**

\((\alpha, \beta)\)-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 \(|\text{Ball}(v, \sqrt{n})| \geq \sqrt{n}\)
- \(c = \epsilon \sqrt{n}\)
Ruling sets for deterministic centers

**First:** Explanation for unweighted graphs

**Definition**

\((\alpha, \beta)\)-**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 \(|\text{Ball}(v, \sqrt{n})| \geq \sqrt{n}\)
- \(c = \epsilon \sqrt{n}\)
- Any shortest \(u - v\) path with \(\geq \sqrt{n}\) edges: ruler in distance \(\leq \epsilon \text{dist}(u, v)\)
Ruling sets for deterministic centers

First: Explanation for unweighted graphs

Definition

\((\alpha, \beta)\)-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 \(\leq \epsilon \text{dist}(u, v)\)
- 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**

(\(\alpha, \beta\))-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
Goal: Make graph locally “look unweighted” s.t. weight $\approx$ #hops
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”
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 ≤ #edges”

Not enough: we also want “#edges ≤ 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 \( 2^i \ldots 2^{i+1} \)

**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 \(\approx \#\text{hops}\)

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

\(G_i: \) round up edge weights to next multiple of \(\epsilon 2^i / \sqrt{n} \) (\(\forall i = 1 \text{ to } \log (nW)\))

(1 + \(\epsilon\))-approximation of shortest paths with \(\sqrt{n}\) edges and weight \(2^i \ldots 2^{i+1}\)

**Intuition:** “weight \(\leq \#\text{edges}”

Not enough: we also want “\#edges \(\leq \) weight”

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

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

**Lemma**

*Every path \(\pi\) 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 $\approx \#\text{hops}$

**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 \#\text{edges}”

Not enough: we also want “$\#\text{edges} \leq \text{weight}”

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

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

**Lemma**

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

$\Rightarrow$ Determine centers by computing ruling set for all type classes
Computing Hop Set on Overlay Network
Hop Sets

Definition

An \((h, \epsilon)\)-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 \((1 + \epsilon)\text{dist}(u, v)\).
Hop Sets

Definition
An \((h, \epsilon)\)-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 \((1 + \epsilon)\) \(\text{dist}(u, v)\).
Hop Sets

Definition
An $(h, \epsilon)$-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 $(1 + \epsilon) \text{dist}(u, v)$.

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

Definition

An \((h, \epsilon)\)-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 \((1 + \epsilon)\text{dist}(u,v)\).

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: \((n^\alpha, \epsilon)\)-hop set of size \(O(n)\) [Miller et al. ’15]
Hop Sets

**Definition**
An \((h, \varepsilon)\)-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 \((1 + \varepsilon)\text{dist}(u, v)\).

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

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

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

C: \((n^\alpha, \varepsilon)\)-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 \( 1/n^{1/k} \)

\( \nu \) has priority \( i \) iff \( \nu \in A_i \setminus A_{i+1} \)
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 \( i \):

\[
\text{Cluster}(v) = \{ u \in V \mid \text{dist}(u, v) < \text{dist}(u, A_{i+1}) \}
\]
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 \( i \):

\[ \text{Cluster}(v) = \{ u \in V \mid \text{dist}(u, v) < \text{dist}(u, A_{i+1}) \} \]
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 \( i \):

\[ \text{Cluster}(v) = \{ u \in V \mid \text{dist}(u, v) < \text{dist}(u, A_{i+1}) \} \]
Hop Set Based on Clusters [Thorup/Zwick ’01]

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

\( v \) has priority \( i \) iff \( v \in A_i \setminus A_{i+1} \)

For every node \( u \) of priority \( i \):
\[ \text{Cluster}(v) = \{ u \in V \mid \text{dist}(u,v) < \text{dist}(u,A_{i+1}) \} \]

Hop set:
\[ (u,v) \in F \text{ iff } u \in \text{Cluster}(v) \]
\[ w(u,v) = \text{dist}_G(u,v) \]
Hop Set Based on Clusters \[\text{[Thorup/Zwick '01]}\]

\[V = A_0 \supseteq A_1 \supseteq \cdots \supseteq A_k = \emptyset\] 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 \(i\):

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

Hop set:

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

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

Hop set:
- \((u, v) \in F\) iff \( u \in \text{Cluster}(v)\)
- \( w(u, v) = \text{dist}_G(u, v) \)
- 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 \( i \):

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

**Hop set:**

- \( (u, v) \in F \) iff \( u \in \text{Cluster}(v) \)
- \( w(u, v) = \text{dist}_G(u, v) \)
- **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

⇒ 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
    $H_i = G \cup \bigcup_{1 \leq j \leq i-1} F_j$
    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 $F = \bigcup_{1 \leq i \leq k} F_i$
```
Computing \((n^{o(1)}, \varepsilon)\)-hop set

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

**Algorithm:**

\[
\text{for } i = 1 \text{ to } k \text{ do}
\]

\[
H_i = G \cup \bigcup_{1 \leq j \leq i-1} F_j
\]

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

\[
F_i = \{ (u, v) \mid u \in \text{Cluster}(v) \}
\]

end

**return** \(F = \bigcup_{1 \leq i \leq k} F_i\)

Error amplification: \((1 + \varepsilon')^k \leq (1 + \varepsilon)\) for \(\varepsilon' = 1/(2\varepsilon \log n)\)
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:**

```plaintext
for i = 1 to k do
    \[ H_i = G \cup \bigcup_{1 \leq j \leq i-1} F_j \]
    Compute clusters with \(k\) priorities in \(H_i\) up to \(n^{2/k}\) hops
    \[ F_i = \{(u, v) \mid u \in \text{Cluster}(v)\} \]
end

return \(F = \bigcup_{1 \leq i \leq k} F_i\)
```

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

**Omitted detail:** weighted graphs, use rounding technique
Computing Hop Set on Overlay Network

Shortest paths from source $s$ \textbf{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 $d$:

Broadcast level
Computing Hop Set on Overlay Network

Shortest paths from source \( s \) up to distance \( d \):

**Broadcast level**

\( d \) iterations, each \( O(Diam + N_\ell) \) rounds where \( N_\ell = \#\text{nodes at level } \ell \)

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

\( N \approx \sqrt{n} \)
Computing Hop Set on Overlay Network

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

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

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

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

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

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

$\Rightarrow$ 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

Open problems:

- $n \rightarrow \log O(1)$
- Be\_ter hop set?
- Improve dependence on $O(n)$ rounds optimal for exact SSSP?
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 $\epsilon$
- $O(n)$ rounds optimal for exact SSSP?
Example: \((n^{1/2+o(1)}, \epsilon)\)-hop set

Case 1: \(\text{dist}(u_0, v) \leq n^{1/2+1/k} / \epsilon\)
Example: \((n^{1/2+o(1)}, \varepsilon)\)-hop set

**Case 2:** \(\text{dist}(u_0, v) > n^{1/2+1/k}/\varepsilon\)
Example: \((n^{1/2+o(1)}, \epsilon)\)-hop set

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

\[ r_0 = n^{1/2} \]
Example: \((n^{1/2+o(1)}, \epsilon)\)-hop set

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

\[ r_0 = n^{1/2} \]

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. \(\text{dist}(u, u') \leq \text{dist}(u, v)\).
Example: \((n^{1/2+o(1)}, \epsilon)-\text{hop set}\)

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

\[
\begin{align*}
    r_0 &= n^{1/2} \\
    r_{i+1} &= \left(1 + \frac{2}{\epsilon}\right) \sum_{0 \leq j \leq i} r_j
\end{align*}
\]

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. \(\text{dist}(u, u') \leq \text{dist}(u, v)\).
Example: \((n^{1/2+o(1)}, \varepsilon)-hop\) set

**Case 2:** \(\text{dist}(u_0, v) > n^{1/2+1/k}/\varepsilon\)

\[ r_0 = n^{1/2} \]
\[ r_{i+1} = \left(1 + \frac{2}{\varepsilon}\right) \sum_{0 \leq j \leq i} r_j \]

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. \(\text{dist}(u, u') \leq \text{dist}(u, v)\).
Example: \((n^{1/2+o(1)}, \varepsilon)\)-hop set

**Case 2:** \(\text{dist}(u_0, v) > n^{1/2+1/k}/\varepsilon\)

\[
\begin{align*}
    r_0 &= n^{1/2} \\
    r_{i+1} &= \left(1 + \frac{2}{\varepsilon}\right) \sum_{0 \leq j \leq i} r_j
\end{align*}
\]

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. \(\text{dist}(u, u') \leq \text{dist}(u, v)\).
Example: \((n^{1/2 + o(1)}, \epsilon)-\text{hop set}\)

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

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. \(\text{dist}(u, u') \leq \text{dist}(u, v)\).

\[
\begin{align*}
    r_0 &= n^{1/2} \\
    r_{i+1} &= \left(1 + \frac{2}{\epsilon}\right) \sum_{0 \leq j \leq i} r_j \\
    &\leq n^{1/2} n^{1/k} \\
    k &= \sqrt{\log n}/ \sqrt{\log 4/\epsilon}
\end{align*}
\]

\[
\text{Weight} \leq (1 + \epsilon) \text{dist}(u_0, v)
\]
Example: \((n^{1/2+o(1)}, \varepsilon)-\text{hop set}\)

**Case 2:** \(\text{dist}(u_0, v) > n^{1/2+1/k}/\varepsilon\)

\[
\begin{align*}
  r_0 &= n^{1/2} \\
  r_{i+1} &= \left(1 + \frac{2}{\varepsilon}\right) \sum_{0 \leq j \leq i} r_j \\
  &\leq n^{1/2} n^{1/k} \\
  k &= \sqrt{\log n/ \sqrt{\log 4/\varepsilon}}
\end{align*}
\]

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. \(\text{dist}(u, u') \leq \text{dist}(u, v)\).

\[
\begin{align*}
  \text{Weight} &\leq (1 + \varepsilon)\text{dist}(u_0, v) \\
  \#\text{Edges} &\leq \frac{k \cdot \text{dist}(u, v)}{n^{1/2}} \leq \frac{k \cdot n}{n^{1/2}} = kn^{1/2}
\end{align*}
\]