Approximate Single-Source Shortest Paths: Distributed and Dynamic Algorithms

Sebastian Krinninger

Max Planck Institute for Informatics

joint works with

Ruben Becker
Monika Henzinger
Andreas Karrenbauer
Christoph Lenzen
Danupon Nanongkai
One Problem – Two Results

(1 + ε)-approximate single-source shortest paths (SSSP)
One Problem – Two Results

(1 + \(\varepsilon\))-approximate single-source shortest paths (SSSP)

Distributed algorithm: Deterministically compute approximate shortest paths in \(n^{1/2+o(1)} + Diam^{1+o(1)}\) rounds [HKN ’16]
One Problem – Two Results

**(1 + \(\epsilon\))-approximate single-source shortest paths (SSSP)**

1. Distributed algorithm: Deterministically compute approximate shortest paths in \(n^{1/2 + o(1)} + Diam^{1+o(1)}\) rounds [HKN ’16]

*Similar in spirit:*

   Multipass streaming: \(n^{1+o(1)}\) space with \(n^{o(1)}\) passes [HKN ’16]
One Problem – Two Results

(1 + \(\epsilon\))-approximate single-source shortest paths (SSSP)

1. Distributed algorithm: Deterministically compute approximate shortest paths in \(n^{1/2+o(1)} + Diam^{1+o(1)}\) rounds [HKN '16]

   Similar in spirit:
   Multipass streaming: \(n^{1+o(1)}\) space with \(n^{o(1)}\) passes [HKN '16]

2. Dynamic algorithm: Maintain approximate shortest paths under edge deletions with amortized update time \(n^{o(1)}\) [HKN '14]
One Problem – Two Results

(1 + \(\epsilon\))-approximate single-source shortest paths (SSSP)

1. Distributed algorithm: Deterministically compute approximate shortest paths in \(n^{1/2 + o(1)} + Diam^{1+o(1)}\) rounds [HKN ’16]

   Similar in spirit:
   Multipass streaming: \(n^{1+o(1)}\) space with \(n^{o(1)}\) passes [HKN ’16]

2. Dynamic algorithm: Maintain approximate shortest paths under edge deletions with amortized update time \(n^{o(1)}\) [HKN ’14]

Main technique: Iterative computation of hop set
One Problem – Two Results

(1 + \epsilon)-approximate single-source shortest paths (SSSP)

1. Distributed algorithm: Deterministically compute approximate shortest paths in $n^{1/2+o(1)} + Diam^{1+o(1)}$ rounds \[HKN '16\]

   Similar in spirit:
   Multipass streaming: $n^{1+o(1)}$ space with $n^{o(1)}$ passes \[HKN '16\]

2. Dynamic algorithm: Maintain approximate shortest paths under edge deletions with amortized update time $n^{o(1)}$ \[HKN '14\]

Main technique: Iterative computation of hop set

This talk: constant \(\epsilon\), positive integer edge weights polynomial in \(n\)
Hop Reduction
**Well Known: Spanners**

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>A $k$-spanner is a subgraph $H$ of $G$ such that, for all pairs of nodes $u$ and $v$, $\text{dist}_H(u, v) \leq k \cdot \text{dist}_G(u, v)$.</td>
</tr>
</tbody>
</table>
Well Known: Spanners

**Definition**

A $k$-spanner is a subgraph $H$ of $G$ such that, for all pairs of nodes $u$ and $v$,

$$\text{dist}_H(u, v) \leq k \cdot \text{dist}_G(u, v).$$

**Fact:** Every graph has a $k$-spanner of size $n^{1 + 1/k}$.

**Application:** Running time $T(m, n) \Rightarrow T(n^{1 + 1/k}, n)$.
Well Known: Spanners

**Definition**

A $k$-spanner is a subgraph $H$ of $G$ such that, for all pairs of nodes $u$ and $v$, $\text{dist}_H(u, v) \leq k \cdot \text{dist}_G(u, v)$.
Well Known: Spanners

**Definition**

A \( k \)-spanner is a subgraph \( H \) of \( G \) such that, for all pairs of nodes \( u \) and \( v \),

\[
\text{dist}_H(u, v) \leq k \cdot \text{dist}_G(u, v).
\]

**Fact:** Every graph has a \( k \)-spanner of size \( n^{1+1/k} \) \[Folklore\]

**Application:** Running time \( T(m, n) \Rightarrow T(n^{1+1/k}, n) \)
Less Known: 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 \((1 + \epsilon) \text{dist}(u, v)\).
Less Known: 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 \((1 + \epsilon)dist(u, v)\).

Fact: Every graph has a \((\log O(1), \epsilon/n)\)-hop set of size \(m^{1+o(1)}\) [Cohen '94].
**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 \((1 + \epsilon) \text{dist}(u, v)\).
Less Known: 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 \((1 + \epsilon)\text{dist}(u, v)\).

Fact: Every graph has a \((\log^{O(1)} n, \epsilon)\)-hop set of size \(m^{1+o(1)}\) [Cohen ’94]
Less Known: 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?**

- Dijkstra: SSSP in time \(O(m + n \log n)\)
  
  Not local (global heap), bad for non-centralized models
Less Known: 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 \((1 + \epsilon)\)dist\((u, v)\).

**Application?**
- **Dijkstra:** SSSP in time \(O(m + n \log n)\)
  Not local (global heap), bad for non-centralized models
- **Bellman-Ford:** SSSP in time \(O(mn)\)
  Actually: SSSP up to \(h\) hops in time \(O(mh)\)
  With \(n^{o(1)}, \epsilon\) hop set: \((1 + \epsilon)\)-approximate SSSP in time \(O(m^{1 + o(1)})\)
  Approach used before in parallel setting [Cohen ’94]
Simple Hop Set Based on Balls (following [Thorup/Zwick ’06])

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

\( v \) has **priority** \( i \) if \( v \in A_i \setminus A_{i+1} \)
Simple Hop Set Based on Balls (following [Thorup/Zwick ’06])

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

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

For every node \( u \) of priority \( i \):

\[ Ball(u) = \{ v \in V \mid \text{dist}(u, v) < \text{dist}(u, A_{i+1}) \} \]
Simple Hop Set Based on Balls (following [Thorup/Zwick ’06])

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

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

For every node \( u \) of priority \( i \):

\[ Ball(u) = \{ v \in V \mid \text{dist}(u, v) < \text{dist}(u, A_{i+1}) \} \]
Simple Hop Set Based on Balls (following [Thorup/Zwick ’06])

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

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

For every node \( u \) of priority \( i \):

\[ Ball(u) = \{ v \in V \mid \text{dist}(u,v) < \text{dist}(u,A_{i+1}) \} \]
Simple Hop Set Based on Balls (following [Thorup/Zwick ’06])

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

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

For every node \( u \) of priority \( i \):

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

**Hop set:**

- \((u, v) \in F \) iff \( v \in Ball(u) \)
- \( w(u, v) = \text{dist}_G(u, v) \)
Simple Hop Set Based on Balls (following [Thorup/Zwick ’06])

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

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

For every node \( u \) of priority \( i \):

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

<table>
<thead>
<tr>
<th>priority</th>
<th># nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( n )</td>
</tr>
<tr>
<td>1</td>
<td>( n^{1-1/k} )</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>( k-1 )</td>
<td>( n^{1/k} )</td>
</tr>
</tbody>
</table>

Hop set:

- \((u, v) \in F \) iff \( v \in \text{Ball}(u) \)
- \( w(u, v) = \text{dist}_G(u, v) \)
Simple Hop Set Based on Balls (following [Thorup/Zwick ’06])

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

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

For every node \( u \) of priority \( i \):

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

**Expected size:** \( n^{(i+1)/k} \)

| priority | # nodes | \( |Ball(u)| \) |
|----------|---------|----------------|
| 0        | \( n \) | \( n^{1/k} \) |
| 1        | \( n^{1-1/k} \) | \( n^{2/k} \) |
| \vdots   | \vdots | \vdots |
| \( k-1 \) | \( n^{1/k} \) | \( n \) |

**Hop set:**

- \( (u, v) \in F \) iff \( v \in Ball(u) \)
- \( w(u, v) = \text{dist}_G(u, v) \)
Simple Hop Set Based on Balls (following [Thorup/Zwick ’06])

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

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

For every node \( u \) of priority \( i \):

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

**Expected size:** \( n^{(i+1)/k} \)

| priority | # nodes | \( |Ball(u)| \) | # edges |
|----------|---------|----------------|---------|
| 0        | \( n \) | \( n^{1/k} \)  | \( n^{1+1/k} \) |
| 1        | \( n^{1-1/k} \) | \( n^{2/k} \)  | \( n^{1+1/k} \) |
| \vdots   | \vdots  | \vdots         | \vdots  |
| \( k-1 \) | \( n^{1/k} \) | \( n \)        | \( n^{1+1/k} \) |

**Hop set:**
- \( (u, v) \in F \) iff \( v \in Ball(u) \)
- \( w(u, v) = \text{dist}_G(u, v) \)
Simple Hop Set Based on Balls (following [Thorup/Zwick ’06])

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

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

For every node \( u \) of priority \( i \):

\[ Ball(u) = \{ \nu \in V \mid dist(u, \nu) < dist(u, A_{i+1}) \} \]

Expected size: \( n^{(i+1)/k} \)

| priority | # nodes | \( |Ball(u)| \) | # edges |
|----------|---------|----------------|---------|
| 0        | \( n \)  | \( n^{1/k} \)  | \( n^{1+1/k} \) |
| 1        | \( n^{1-1/k} \) | \( n^{2/k} \)  | \( n^{1+1/k} \) |
| :        | :       | :              | :       |
| \( k - 1 \) | \( n^{1/k} \)  | \( n \)         | \( n^{1+1/k} \) |

\( (u, v) \in F \) iff \( v \in Ball(u) \)

\( w(u, v) = dist_G(u, v) \)
Parameter Choice

\[ k = \frac{\sqrt{\log n}}{\sqrt{\log 4/\epsilon}} \]

\[ \left(\frac{4}{\epsilon}\right)^k = n^{1/k} \]
Parameter Choice

\[ k = \frac{\sqrt{\log n}}{\sqrt{\log 4/\epsilon}} \]

\[ \left( \frac{4}{\epsilon} \right)^k = n^{1/k} = n^{o(1)} \]
Case 1: $\text{dist}(u_0, v) \leq n^{1/2+1/k}/\epsilon$
(\(n^{1/2+o(1)}, \varepsilon\))-hop set

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

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

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

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

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

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

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

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

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

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

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

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

1. Goal: Faster SSSP via hop set
2. Compute hop set by computing balls
3. Computing balls 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 balls
3. Computing balls at least as hard as SSSP
⇒ Back at problem we wanted to solve initially?

No! \((n^{1/2+o(1)}, \epsilon)\)-hop set only requires balls up to \(n^{1/2+o(1)}\) hops
(n^{1/2+o(1)}, \epsilon)-hop set

Iterative computation
In each iteration number of hops is reduced by a factor of n^{1/k}
(\(n^{1/2+o(1)}, \epsilon\))-hop set

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

Algorithm:

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

return \(F = \bigcup_{1 \leq i \leq k} F_i\)
(n^{1/2+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 balls with \( k \) priorities in \( H_i \) up to \( n^{2/k} \) hops
\[
F_i = \{(u, v) \mid v \in Ball(u)\}
\]
end

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

Error amplification: \((1 + \epsilon')^k \leq (1 + \epsilon)\) for \( \epsilon' = 1/(2\epsilon \log n) \)
$(n^{1/2+o(1)}, \epsilon)$-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 balls with $k$ priorities in $H_i$ up to $n^{2/k}$ hops
    $F_i = \{(u, v) \mid v \in Ball(u)\}$
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
Distributed Algorithm
Distributed Algorithm

SSSP in \textbf{CONGEST} model: synchronous rounds, message size $O(\log n)$

Running time = number of rounds

- \textbf{Exact:} $O(n)$ (Bellman-Ford)
- \textbf{(1 + $\epsilon$)-approximation:}
  - $\Omega(n^{1/2}/\log n + Diam)$ [Das Sarma et al. ’11]
  - $O(\epsilon^{-1} \log \epsilon^{-1})$: $O(n^{1/2+\epsilon} + Diam)$ (randomized) [Lenzen, Patt-Shamir ’13]
  - 1 + $\epsilon$: $O(n^{1/2} Diam^{1/4} + Diam)$ (randomized) [Nanongkai ’14]
  - 1 + $\epsilon$: $O(n^{1/2+o(1)} + Diam^{1+o(1)})$ (deterministic) (New)

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
Distributed Algorithm

SSSP in CONGEST model: synchronous rounds, message size $O(\log n)$

Running time = number of rounds

- **Exact**: $O(n)$ (Bellman-Ford)
- **(1 + $\varepsilon$)-approximation:**
  - $\Omega(n^{1/2}/ \log n + Diam)$ [Das Sarma et al. ’11]
  - $O(\varepsilon^{-1} \log \varepsilon^{-1})$: $O(n^{1/2+\varepsilon} + Diam)$ (randomized) [Lenzen, Patt-Shamir ’13]
  - $1 + \varepsilon$: $O(n^{1/2} Diam^{1/4} + Diam)$ (randomized) [Nanongkai ’14]
  - $1 + \varepsilon$: $O(n^{1/2+o(1)} + Diam^{1+o(1)})$ (deterministic) (New)

Our approach:

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

SSSP in **CONGEST** model: synchronous rounds, message size $O(\log n)$

Running time = number of rounds

- **Exact:** $O(n)$ (Bellman-Ford)
- **$(1 + \epsilon)$-approximation:**
  - $\Omega(n^{1/2}/\log n + Diam)$ [Das Sarma et al. ’11]
  - $O(\epsilon^{-1} \log \epsilon^{-1})$: $O(n^{1/2+\epsilon} + Diam)$ (randomized) [Lenzen, Patt-Shamir ’13]
  - $1 + \epsilon$: $O(n^{1/2}Diam^{1/4} + Diam)$ (randomized) [Nanongkai ’14]
  - $1 + \epsilon$: $O(n^{1/2+o(1)} + Diam^{1+o(1)})$ (deterministic) (**New**)

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
Overlay Network

Sample

$N = \tilde{O}\left(\frac{n}{2}\right)$ centers (+ sources)

$\Rightarrow$ Every shortest path with $\geq \frac{n}{2}$ edges contains center whp

Solve SSSP on overlay network using hop set
Sample $N = \tilde{O}(n^{1/2})$ centers (+ source $s$)

$\Rightarrow$ Every shortest path with $\geq n^{1/2}$ edges contains center whp
Sample $N = \tilde{O}(n^{1/2})$ centers (+ source $s$)
$\Rightarrow$ Every shortest path with $\geq n^{1/2}$ edges contains center whp
Solve SSSP on overlay network using hop set
Derandomization of Overlay Network

Randomization: Hit every shortest path with $\geq \sqrt{n}$ edges
Derandomization of Overlay Network

Randomization: Hit every shortest path with $\geq \sqrt{n}$ edges

Deterministic relaxation: Almost hit every path $\geq \sqrt{n}$ edges
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 $s$ \textbf{up to distance} $D$:

\[ \text{Running time: } O(D \cdot \text{Diam} + \sum_{l \leq D} M^l) = O(D \cdot \text{Diam} + N) \]

\[ \Rightarrow \text{Hop Set and approximate SSSP: } O(n^{1/2} + o(1) + \text{Diam} + o(1)) \]
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 $O(Diam + M_{\ell})$ rounds where $M_{\ell} = \#\text{nodes at level } \ell$

Running time: $O(D \cdot Diam + \sum_{\ell \leq D} M_{\ell}) = O(D \cdot Diam + N)$
Computing Hop Set on Overlay Network

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

Broadcast level

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

Running time: $O(D \cdot Diam + \sum_{\ell \leq D} M_\ell) = O(D \cdot Diam + N)$

Computing balls: $\tilde{O}(n^{1/k} \cdot Diam + \sum_v |Ball(v)|) = \tilde{O}(n^{1/k} \cdot Diam + N^{1+1/k})$
Computing Hop Set on Overlay Network

Shortest paths from source $s$ up to distance $D$: $D$ iterations, each $O(Diam + M_\ell)$ rounds where $M_\ell = \#\text{nodes at level } \ell$

Running time: $O(D \cdot Diam + \sum_{\ell \leq D} M_\ell) = O(D \cdot Diam + N)$

Computing balls: $\tilde{O}(n^{1/k} \cdot Diam + \sum_v |Ball(v)|) = \tilde{O}(n^{1/k} \cdot Diam + N^{1+1/k})$

$\Rightarrow$ Hop Set and approximate SSSP: $O(n^{1/2+o(1)} + Diam^{1+o(1)})$
Dynamic Algorithm
Decremental Approximate Shortest Path Problem

$G$ undergoing deletions:

$$\text{dist}_G(s, v) \leq \delta(s, v) \leq \left(1 + \frac{1}{\ln n}\right) \text{dist}_G(s, v)$$

Update time for all updates / Query time per query
Decremental Approximate Shortest Path Problem

$G$ undergoing deletions:

Decremental algorithm

Answer: approximate shortest path of length $\text{dist}_G(s, v) \leq \delta(s, v) \leq (1 + \epsilon^{-1}) \text{dist}_G(s, v)$

Update time for all updates / Query time per query
Decremental Approximate Shortest Path Problem

$G$ undergoing deletions:

Answer: approximate shortest path of length $\text{dist}_{G}(s, v) \leq \delta(s, v) \leq (1 + \epsilon)\text{dist}_{G}(s, v)$

Update time for all updates / Query time per query

18 / 24
Decremental Approximate Shortest Path Problem

\[ \text{dist}_G(s, v) \] would undergoing deletions:

\[ \text{Update} \]

\[ \text{Query} \]

Decremental algorithm
Decremental Approximate Shortest Path Problem

$G$ undergoing deletions:

$\text{dist}_G(s, v)$?

Answer: approximate shortest path of length

$\delta(s, v)$
Decremental Approximate Shortest Path Problem

$G$ undergoing deletions:

$\text{dist}_G(s, v)$?

Answer: approximate shortest path of length

$$\text{dist}_G(s, v) \leq \delta(s, v) \leq (1 + \epsilon)\text{dist}_G(s, v)$$
Decremental Approximate Shortest Path Problem

Given a graph $G$ undergoing deletions:

$G(s, v)$?

Answer: approximate shortest path of length

$$\text{dist}_G(s, v) \leq \delta(s, v) \leq (1 + \epsilon)\text{dist}_G(s, v)$$

Update time for all updates

Query time per query
Overview of Result

New result:

- **Exact:** total update time $O(mn)$ (unweighted) [Even/Shiloach ’81]
  $\Omega(mn)$ [Roditty/Zwick ’04, Henzinger/K/Nanongkai/Saranurak ’15]
- **$(1 + \epsilon)$-approx.:** $O(n^{2+o(1)})$ (unweighted) [Bernstein/Roditty ’11]
- **New:** $O(m^{1+o(1)})$ (weighted) [Henzinger/K/Nanongkai ’14]
Overview of Result

New result:

- Exact: total update time $O(mn)$ (unweighted) [Even/Shiloach ’81] \(\Omega(mn)\) [Roditty/Zwick ’04, Henzinger/K/Nanongkai/Saranurak ’15]
- \((1 + \epsilon)\)-approx.: $O(n^{2+o(1)})$ (unweighted) [Bernstein/Roditty ’11]
- **New**: $O(m^{1+o(1)})$ (weighted) [Henzinger/K/Nanongkai ’14]

Techniques for maintaining balls:
Overview of Result

New result:

- Exact: total update time $O(mn)$ (unweighted) [Even/Shiloach ’81]
  $\Omega(mn)$ [Roditty/Zwick ’04, Henzinger/K/Nanongkai/Saranurak ’15]
- $(1 + \epsilon)$-approx.: $O(n^{2+o(1)})$ (unweighted) [Bernstein/Roditty ’11]
- **New**: $O(m^{1+o(1)})$ (weighted) [Henzinger/K/Nanongkai ’14]

Techniques for maintaining balls:

- Even-Shiloach: $O(mD)$ for SSSP up to depth $D$
Overview of Result

New result:

- **Exact**: total update time $O(mn)$ (unweighted) [Even/Shiloach ’81]
  - $\Omega(mn)$ [Roditty/Zwick ’04, Henzinger/K/Nanongkai/Saranurak ’15]
- $(1 + \epsilon)$-approx.: $O(n^{2+o(1)})$ (unweighted) [Bernstein/Roditty ’11]
- **New**: $O(m^{1+o(1)})$ (weighted) [Henzinger/K/Nanongkai ’14]

Techniques for maintaining balls:
- Even-Shiloach: $O(mD)$ for SSSP up to depth $D$
- Restart when distance to next priority changes
Overview of Result

New result:

- Exact: total update time $O(mn)$ (unweighted) [Even/Shiloach ’81]
  $\Omega(mn)$ [Roditty/Zwick ’04, Henzinger/K/Nanongkai/Saranurak ’15]
- $(1 + \epsilon)$-approx.: $O(n^{2+o(1)})$ (unweighted) [Bernstein/Roditty ’11]
- New: $O(m^{1+o(1)})$ (weighted) [Henzinger/K/Nanongkai ’14]

Techniques for maintaining balls:

- Even-Shiloach: $O(mD)$ for SSSP up to depth $D$
- Restart when distance to next priority changes
- Bounding number of nodes in balls not enough
  All edges incident to balls go into running time
  $\Rightarrow$ Sample edges instead of nodes
Overview of Result

New result:

- **Exact:** total update time $O(mn)$ (unweighted) [Even/Shiloach ’81]
  $\Omega(mn)$ [Roditty/Zwick ’04, Henzinger/K/Nanongkai/Saranurak ’15]
- $(1 + \varepsilon)$-approx.: $O(n^{2+o(1)})$ (unweighted) [Bernstein/Roditty ’11]
- **New:** $O(m^{1+o(1)})$ (weighted) [Henzinger/K/Nanongkai ’14]

Techniques for maintaining balls:

- **Even-Shiloach:** $O(mD)$ for SSSP up to depth $D$
- Restart when distance to next priority changes
- Bounding number of nodes in balls not enough
  - All edges incident to balls go into running time
  $\Rightarrow$ Sample edges instead of nodes
- Deletions-only problem, but edges might be added to hop set
  - Monotone ES-tree framework [Henzinger/K/Nanongkai ’13]
New Approach
Theorem ([Becker/Karrenbauer/K/Lenzen arXiv’16])

There is a deterministic algorithm for computing \((1 + \epsilon)\) approximate SSSP in \(\tilde{O}(\sqrt{n} + Diam)\) rounds.
New Distributed Algorithm

Theorem ([Becker/Karrenbauer/K/Lenzen arXiv’16])

There is a deterministic algorithm for computing \((1 + \epsilon)\) approximate SSSP in \(\widetilde{O}(\sqrt{n} + Diam)\) rounds.

Key insight: Solve more general problem

Shortest Transshipment Problem

Find the cheapest route for sending units of a single good from sources to sinks along the edges of a graph as specified by demands on nodes.
New Distributed Algorithm

Theorem ([Becker/Karrenbauer/K/Lenzen arXiv’16])

There is a deterministic algorithm for computing $(1 + \varepsilon)$ approximate SSSP in $\tilde{O}(\sqrt{n} + \text{Diam})$ rounds.

Key insight: Solve more general problem

Shortest Transshipment Problem

Find the cheapest route for sending units of a single good from sources to sinks along the edges of a graph as specified by demands on nodes.

“Uncapacitated minimum-cost flow”
Theorem ([Becker/Karrenbauer/K/Lenzen arXiv’16])

There is a deterministic algorithm for computing \((1 + \varepsilon)\) approximate SSSP in \(\widetilde{O}(\sqrt{n} + \text{Diam})\) rounds.

Key insight: Solve more general problem

**Shortest Transshipment Problem**

Find the cheapest route for sending units of a single good from sources to sinks along the edges of a graph as specified by demands on nodes.

“Uncapacitated minimum-cost flow”

**SSSP**: source has demand \(-(n - 1)\), other nodes have demand 1
Shortest Transshipment Problem

Shortest transshipment as linear program:

\[
\text{minimize } \| Wx \|_1 \quad \text{s.t. } Ax = b
\]
Shortest Transshipment Problem

Shortest transshipment as linear program:

\[
\text{minimize } \| Wx \|_1 \quad \text{s.t. } Ax = b
\]

Dual program:

\[
\text{maximize } b^T y \quad \text{s.t. } \| W^{-1} A^T y \|_\infty \leq 1
\]
Shortest Transshipment Problem

Shortest transshipment as linear program:

\[
\text{minimize } \|Wx\|_1 \quad \text{s.t. } Ax = b
\]

Dual program:

\[
\text{maximize } b^T y \quad \text{s.t. } \|W^{-1}A^T y\|_\infty \leq 1
\]

Equivalent:

\[
\text{minimize } \|W^{-1}A^T y\|_\infty \quad \text{s.t. } b^T \pi = 1
\]
Shortest Transshipment Problem

Shortest transshipment as linear program:

\[
\text{minimize } \|Wx\|_1 \quad \text{s.t. } Ax = b
\]

Dual program:

\[
\text{maximize } b^T y \quad \text{s.t. } \|W^{-1}A^T y\|_\infty \leq 1
\]

Equivalent:

\[
\text{minimize } \|W^{-1}A^T y\|_\infty \quad \text{s.t. } b^T \pi = 1
\]

We approximate \(\|\cdot\|_\infty\) by soft-max:

\[
lse_\beta(x) := \frac{1}{\beta} \ln \left( \sum_{i \in [d]} (e^{\beta x_i} + e^{-\beta x_i}) \right)
\]
Gradient Descent

Algorithm at a glance:

1. Soft-max is differentiable → apply gradient descent
Gradient Descent

Algorithm at a glance:

1. Soft-max is differentiable → apply gradient descent
2. Each iteration: solve transshipment problem with different demand vector $b'$ depending on current gradient
Gradient Descent

Algorithm at a glance:

1. Soft-max is differentiable → apply gradient descent
2. Each iteration: solve transshipment problem with different demand vector $b'$ depending on current gradient
3. Key observation: For $b'$, bad approximation is sufficient
Gradient Descent

Algorithm at a glance:

1. Soft-max is differentiable → apply gradient descent
2. Each iteration: solve transshipment problem with different demand vector $b'$ depending on current gradient
3. Key observation: For $b'$, bad approximation is sufficient
4. Compute spanner on overlay network and solving transshipment on overlay spanner

Spanner has stretch $O(\log n)$ and size $\tilde{O}(n)$
Gradient Descent

Algorithm at a glance:

1. Soft-max is differentiable → apply gradient descent
2. Each iteration: solve transshipment problem with different demand vector $b'$ depending on current gradient
3. Key observation: For $b'$, bad approximation is sufficient
4. Compute spanner on overlay network and solving transshipment on overlay spanner  
   \[\text{Spanner has stretch } O(\log n) \text{ and size } \tilde{O}(n)\]
5. Overall: Polylog iterations, each solving $O(\log n)$-approximate transshipment on graph of $\tilde{O}(n)$ edges
Conclusion

Main contributions:

- Two almost tight algorithms
- Combinatorial and algebraic tools

Open problems:

- Parallel: improve Cohen's \( m_1 + o(1) \) work with polylog depth?
- Be/titer hop set? \( n^{o(1)} \rightarrow \log O(1/n) \)

Deterministic dynamic SSSP algorithm

Vision: Dynamic algorithms as data structures inside other algorithms

Is \( O(n) \) rounds for exact distributed SSSP optimal?
Conclusion

Main contributions:

- Two almost tight algorithms
- Combinatorial and algebraic tools

Open problems:

- Parallel: improve Cohen’s $m^{1+o(1)}$ work with polylog depth?
- Better hop set? $n^{o(1)} \rightarrow \log^{O(1)} n$
- Deterministic dynamic SSSP algorithm
  Vision: Dynamic algorithms as data structures inside other algorithms
- Is $O(n)$ rounds for exact distributed SSSP optimal?