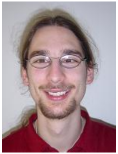
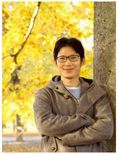
Single-Source Shortest Paths: Towards Optimality

Sebastian Forster

Department of Computer Sciences
University of Salzburg, Austria
Previously known as S. Krinninger

ADGA 2018

joint works with

				
Ruben Becker	Monika Henzinger	Andreas Karrenbauer	Christoph Lenzen	Danupon Nanongkai

Problem Definition

Problem Definition

Context

Goal: Compute shortest paths from a source node s to all other nodes

Context

Goal: Compute shortest paths from a source node s to all other nodes

How can this be an open problem??

Context

Goal: Compute shortest paths from a source node s to all other nodes

How can this be an open problem??

- (Nearly) optimal solutions known in RAM model

Context

Goal: Compute shortest paths from a source node s to all other nodes

How can this be an open problem??

- (Nearly) optimal solutions known in RAM model
- Not fully understood in CONGEST model

Context

Goal: Compute shortest paths from a source node s to all other nodes

How can this be an open problem??

- (Nearly) optimal solutions known in RAM model
- Not fully understood in CONGEST model
- Not fully understood in PRAM model

Context

Goal: Compute shortest paths from a source node s to all other nodes

How can this be an open problem??

- (Nearly) optimal solutions known in RAM model
- Not fully understood in CONGEST model
- Not fully understood in PRAM model
- To be fair: non-negative weights also not fully understood in RAM model

CONGEST Model

Idea: Measure amount of communication for network to compute result

Running time = #communication rounds

CONGEST Model

Idea: Measure amount of communication for network to compute result
Running time = #communication rounds

Model definition:

- Processors with unique IDs modeled as nodes
- Synchronous rounds (global clock)
- In each round, every node sends (at most) one message to each neighbor
- Message size $O(\log n)$
- Unlimited internal computation between rounds

CONGEST Model

Idea: Measure amount of communication for network to compute result
Running time = #communication rounds

Model definition:

- Processors with unique IDs modeled as nodes
- Synchronous rounds (global clock)
- In each round, every node sends (at most) one message to each neighbor
- Message size $O(\log n)$
- Unlimited internal computation between rounds
- Communication network: unweighted undirected graph of diameter D
- Edges are "annotated" with (non-negative) weights and directions
- Weights represent costs (not time)
- This talk: integer edge weights bounded by $n^{O(1)}$

CONGEST Model

Idea: Measure amount of communication for network to compute result
Running time = #communication rounds

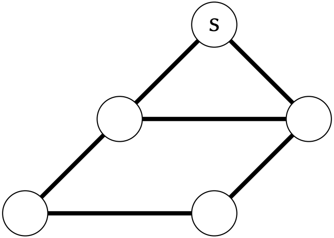
Model definition:

- Processors with unique IDs modeled as nodes
- Synchronous rounds (global clock)
- In each round, every node sends (at most) one message to each neighbor
- Message size $O(\log n)$
- Unlimited internal computation between rounds
- Communication network: unweighted undirected graph of diameter D
- Edges are "annotated" with (non-negative) weights and directions
- Weights represent costs (not time)
- This talk: integer edge weights bounded by $n^{O(1)}$

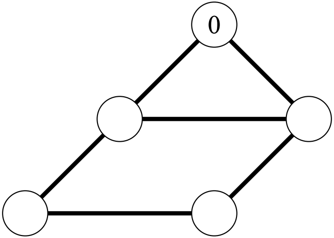
Distributed problem statement:

- Initial knowledge: incident edges, source
- Terminal knowledge: distance to the source, parent on shortest path tree

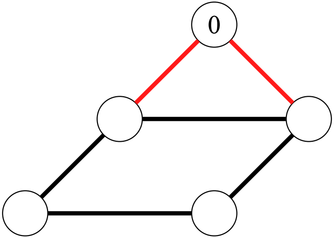
Unweighted Graphs: BFS



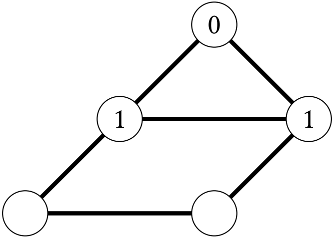
Unweighted Graphs: BFS



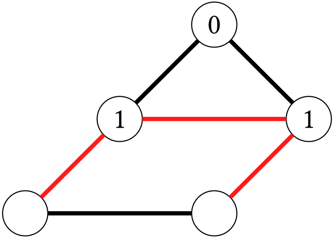
Unweighted Graphs: BFS



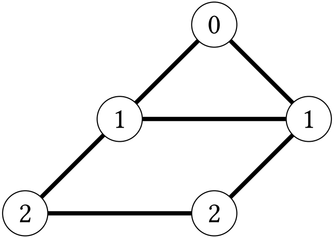
Unweighted Graphs: BFS



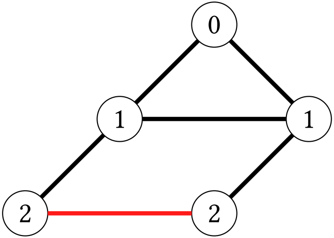
Unweighted Graphs: BFS



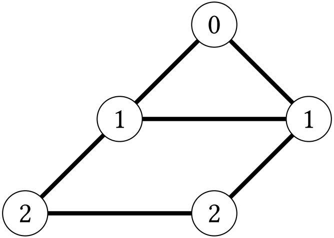
Unweighted Graphs: BFS



Unweighted Graphs: BFS

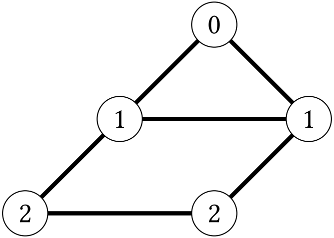


Unweighted Graphs: BFS



Breadth-first search tree can be computed in $O(D)$ rounds.

Unweighted Graphs: BFS



Breadth-first search tree can be computed in $O(D)$ rounds.

Our goal: efficient algorithms for weighted graphs

Known Results

Exact SSSP:

$O(n)$	Bellman-Ford
$\tilde{O}(n^{2/3}D^{1/3} + n^{5/6})$	[Elkin '17]

[1] $\epsilon \geq 1/\log^{O(1)} n$

Known Results

Exact SSSP:

$O(n)$	Bellman-Ford
$\tilde{O}(n^{2/3}D^{1/3} + n^{5/6})$	[Elkin '17]
$\tilde{O}(n^{3/4}D^{1/4})$	[Ghaffari/Li '18]
$\tilde{O}(n^{3/4+o(1)} + \min\{n^{3/4}D^{1/6}, n^{6/7}\} + D)$	[Ghaffari/Li '18]

[1] $\epsilon \geq 1/\log^{O(1)} n$

Known Results

Exact SSSP:

$O(n)$	Bellman-Ford
$\tilde{O}(n^{2/3}D^{1/3} + n^{5/6})$	[Elkin '17]
$\tilde{O}(n^{3/4}D^{1/4})$	[Ghaffari/Li '18]
$\tilde{O}(n^{3/4+o(1)} + \min\{n^{3/4}D^{1/6}, n^{6/7}\} + D)$	[Ghaffari/Li '18]
$\tilde{O}(\sqrt{nD})$	[**F**/Nanongkai]
$\tilde{O}(\sqrt{n}D^{1/4} + n^{3/5} + D)$	[**F**/Nanongkai]

[1] $\epsilon \geq 1/\log^{O(1)} n$

Known Results

Exact SSSP:

$O(n)$	Bellman-Ford
$\tilde{O}(n^{2/3}D^{1/3} + n^{5/6})$	[Elkin '17]
$\tilde{O}(n^{3/4}D^{1/4})$	[Ghaffari/Li '18]
$\tilde{O}(n^{3/4+o(1)} + \min\{n^{3/4}D^{1/6}, n^{6/7}\} + D)$	[Ghaffari/Li '18]
$\tilde{O}(\sqrt{nD})$	[**F**/Nanongkai]
$\tilde{O}(\sqrt{n}D^{1/4} + n^{3/5} + D)$	[**F**/Nanongkai]

$(1+\epsilon)$**-approximate SSSP:**

$\tilde{O}((\sqrt{n}D^{1/4} + D)/\epsilon^{O(1)})$	[Nanongkai '14]
$\tilde{O}((\sqrt{n} + D)n^{o(1)})$ [1]	[Henzinger/**K**/Nanongkai '16]
$\tilde{O}((\sqrt{n} + D)/\epsilon^{O(1)})$	[Becker/Karrenbauer/**K**/Lenzen '17]

[1] $\epsilon \geq 1/\log^{O(1)} n$

Known Results

Exact SSSP:

$O(n)$	Bellman-Ford
$\tilde{O}(n^{2/3}D^{1/3} + n^{5/6})$	[Elkin '17]
$\tilde{O}(n^{3/4}D^{1/4})$	[Ghaffari/Li '18]
$\tilde{O}(n^{3/4+o(1)} + \min\{n^{3/4}D^{1/6}, n^{6/7}\} + D)$	[Ghaffari/Li '18]
$\tilde{O}(\sqrt{nD})$	[**F**/Nanongkai]
$\tilde{O}(\sqrt{n}D^{1/4} + n^{3/5} + D)$	[**F**/Nanongkai]

$(1+\epsilon)$-approximate SSSP:

$\tilde{O}((\sqrt{n}D^{1/4} + D)/\epsilon^{O(1)})$	[Nanongkai '14]
$\tilde{O}((\sqrt{n} + D)n^{o(1)})$ [1]	[Henzinger/**K**/Nanongkai '16]
$\tilde{O}((\sqrt{n} + D)/\epsilon^{O(1)})$	[Becker/Karrenbauer/**K**/Lenzen '17]

Common Lower Bound:

$\tilde{\Omega}(\sqrt{n} + D)$ — [Peleg/Rubinovich '99] [Elkin '04] [Das Sarma et al. '11]

[1] $\epsilon \geq 1/\log^{O(1)} n$

More Related Work

Approximation Algorithms:

- [Lenzen/Patt-Shamir ’13]
- [Lenzen/Patt-Shamir ’15]

More Related Work

Approximation Algorithms:

- [Lenzen/Patt-Shamir '13]
- [Lenzen/Patt-Shamir '15]

All-Pairs Shortest Paths and k-Source Shortest Paths:

- [Holzer/Wattenhofer '12]
- [Elkin/Neiman '16]
- [Huang/Nanongkai/Saranurak '17]
- [Agarwal/Ramachandran/King/Pontecorvi '18]
- [Agarwal/Ramachandran '18]

More Related Work

Approximation Algorithms:

- [Lenzen/Patt-Shamir ’13]
- [Lenzen/Patt-Shamir ’15]

All-Pairs Shortest Paths and k-Source Shortest Paths:

- [Holzer/Wattenhofer ’12]
- [Elkin/Neiman ’16]
- [Huang/Nanongkai/Saranurak ’17]
- [Agarwal/Ramachandran/King/Pontecorvi ’18]
- [Agarwal/Ramachandran ’18]

Congested Clique:

- [Censor-Hillel et al. ’15]
- [Holzer/Pinsker ’15]

Basic Tools

Broadcasting

Lemma

Suppose k pieces of information (of size $O(\log n)$ each) are distributed among the nodes of the network. All this information can be made known to all nodes in $O(k + D)$ rounds.

Need to respect bounded message size!

Broadcasting

Lemma

Suppose k pieces of information (of size $O(\log n)$ each) are distributed among the nodes of the network. All this information can be made known to all nodes in $O(k + D)$ rounds.

Need to respect bounded message size!

Algorithm:

1. Compute BFS tree (from arbitrary root)
2. Aggregate information at root bottom up
   Queue of outgoing messages at each node
3. Distribute information from root top down
   Send one piece at a time

"Pipelining"

Broadcasting

Lemma

Suppose k pieces of information (of size $O(\log n)$ each) are distributed among the nodes of the network. All this information can be made known to all nodes in $O(k + D)$ rounds.

Need to respect bounded message size!

Algorithm:

1. Compute BFS tree (from arbitrary root)
   Time: $O(D)$
2. Aggregate information at root bottom up
   Queue of outgoing messages at each node
   Time: $O(k + D)$
3. Distribute information from root top down
   Send one piece at a time
   Time: $O(k + D)$

“Pipelining”

Bellman-Ford

Algorithm:

1. Initialize $\delta_0(s) = 0$ and $\delta(v) \neq$ for $v \neq s$
2. In round i, set $\delta_i(v) = \min_{(u,v)\in E}(\delta_{i-1}(u) + w(u,v))$

Bellman-Ford

Algorithm:

1. Initialize $\delta_0(s) = 0$ and $\delta(v) \neq$ for $v \neq s$
2. In round i, set $\delta_i(v) = \min_{(u,v)\in E}(\delta_{i-1}(u) + w(u, v))$

Lemma

Can compute shortest paths from given source in $O(n)$ rounds

Bellman-Ford

Algorithm:

1. Initialize $\delta_0(s) = 0$ and $\delta(v) \neq$ for $v \neq s$
2. In round i, set $\delta_i(v) = \min_{(u,v)\in E}(\delta_{i-1}(u) + w(u,v))$

Lemma

Can compute shortest paths from given source in $O(n)$ rounds

Fine-grained analysis: After h rounds, algorithm has computed shortest h-hop paths (shortest among all paths with a "budget" of h edges)

Bellman-Ford

Algorithm:

1. Initialize $\delta_0(s) = 0$ and $\delta(v) \neq$ for $v \neq s$
2. In round i, set $\delta_i(v) = \min_{(u,v)\in E}(\delta_{i-1}(u) + w(u,v))$

Lemma

Can compute shortest paths from given source in $O(n)$ rounds

Fine-grained analysis: After h rounds, algorithm has computed shortest h-hop paths (shortest among all paths with a "budget" of h edges)

Lemma

Can compute h-hop shortest paths from given source in $O(h)$ rounds

Bellman-Ford

Algorithm:

1. Initialize $\delta_0(s) = 0$ and $\delta(v) \neq$ for $v \neq s$
2. In round i, set $\delta_i(v) = \min_{(u,v)\in E}(\delta_{i-1}(u) + w(u,v))$

Lemma

Can compute shortest paths from given source in $O(n)$ rounds

Fine-grained analysis: After h rounds, algorithm has computed shortest h-hop paths (shortest among all paths with a "budget" of h edges)

Lemma

Can compute h-hop shortest paths from given source in $O(h)$ rounds

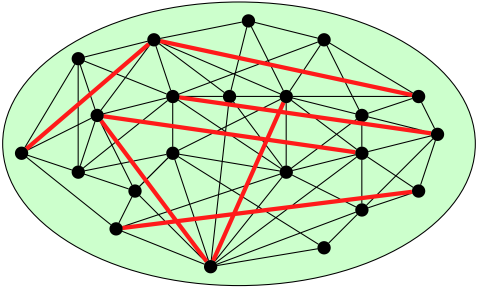
Intuition

SSSP is easy if shortest path has only few edges (hops)!

Hopsets

Definition ([Cohen '00])

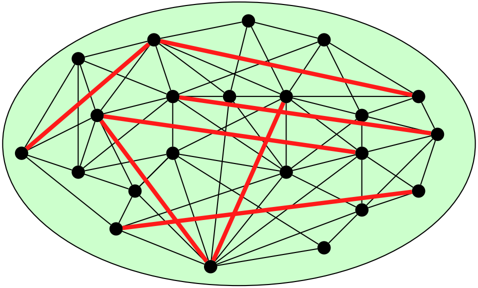
An (h, ϵ)-hopset is a set of weighted edges F such that, for every pair of nodes u and v, there is a path from u to v with *at most h edges* of weight at most $(1+\epsilon)\,\mathrm{dist}_G(u,v)$ in $G \cup F$.



Hopsets

Definition ([Cohen '00])

An (h, ϵ)-hopset is a set of weighted edges F such that, for every pair of nodes u and v, there is a path from u to v with *at most h edges* of weight at most $(1 + \epsilon)\,\text{dist}_G(u, v)$ in $G \cup F$.



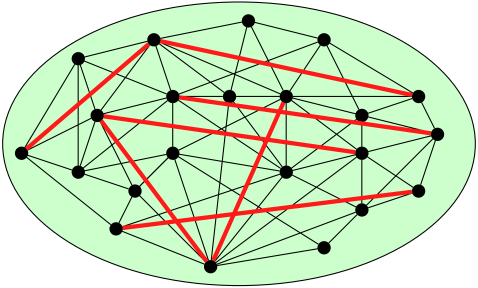
Observation

Given (h, ϵ)-hopset, h-hop shortest paths provide $(1 + \epsilon)$-approximation

Hopsets

Definition ([Cohen '00])

An (h, ϵ)-hopset is a set of weighted edges F such that, for every pair of nodes u and v, there is a path from u to v with *at most h edges* of weight at most $(1 + \epsilon)\,\mathrm{dist}_G(u, v)$ in $G \cup F$.

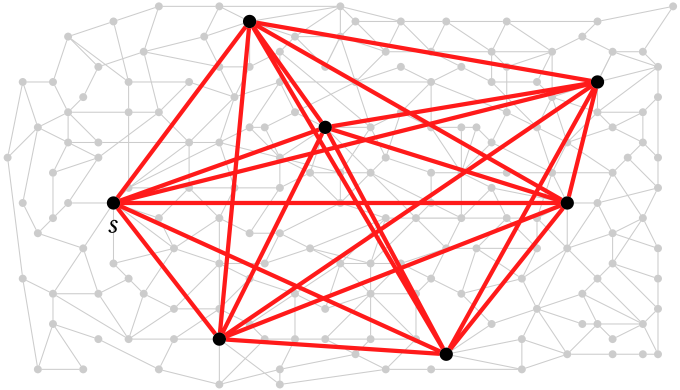


Observation

Given (h, ϵ)-hopset, h-hop shortest paths provide $(1 + \epsilon)$-approximation

Attention: Hopset edges cannot literally be "added" to network!

Skeleton Graph: Intuition



Skeleton Graph

Randomized skeleton H**:**

1. Sample $\tilde{O}(n/h)$ *skeleton nodes* uniformly at random (+ source s)
2. Set $w_H(x,y) = \text{dist}_G^h(x,y)$ (h-hop distance)

Skeleton Graph

Randomized skeleton H:

1. Sample $\tilde{O}(n/h)$ *skeleton nodes* uniformly at random (+ source s)
2. Set $w_H(x,y) = \text{dist}_G^h(x,y)$ (h-hop distance)

Lemma ([Klein/Subramanian '97])

Skeleton is an exact $(\tilde{O}(n/h+h), 0)$-hopset with high probability.

Skeleton Graph

Randomized skeleton H:

1. Sample $\tilde{O}(n/h)$ *skeleton nodes* uniformly at random (+ source s)
2. Set $w_H(x, y) = \text{dist}_G^h(x, y)$ (h-hop distance)

Lemma ([Klein/Subramanian '97])

Skeleton is an exact $(\tilde{O}(n/h + h), 0)$-hopset with high probability.

Lemma ([Ullman/Yannakakis '90])

Every shortest path with $h/2$ edges contains skeleton with high probability.

Skeleton Graph

Randomized skeleton H:

1. Sample $\tilde{O}(n/h)$ *skeleton nodes* uniformly at random (+ source s)
2. Set $w_H(x, y) = \mathrm{dist}^h_G(x, y)$ (h-hop distance)

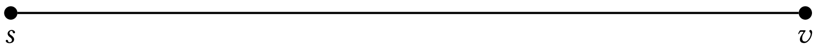
Lemma ([Klein/Subramanian '97])

Skeleton is an exact $(\tilde{O}(n/h + h), 0)$-hopset with high probability.

Lemma ([Ullman/Yannakakis '90])

Every shortest path with $h/2$ edges contains skeleton with high probability.

Proof of hopset property:



Skeleton Graph

Randomized skeleton H:

1. Sample $\tilde{O}(n/h)$ *skeleton nodes* uniformly at random (+ source s)
2. Set $w_H(x,y) = \mathrm{dist}_G^h(x,y)$ (h-hop distance)

Lemma ([Klein/Subramanian '97])

Skeleton is an exact $(\tilde{O}(n/h+h), 0)$-hopset with high probability.

Lemma ([Ullman/Yannakakis '90])

Every shortest path with $h/2$ edges contains skeleton with high probability.

Proof of hopset property:

Skeleton Graph

Randomized skeleton H:

1. Sample $\tilde{O}(n/h)$ *skeleton nodes* uniformly at random (+ source s)
2. Set $w_H(x, y) = \text{dist}_G^h(x, y)$ (h-hop distance)

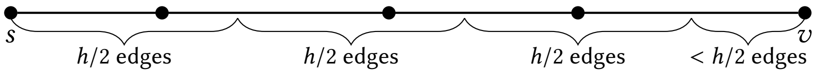
Lemma ([Klein/Subramanian '97])

Skeleton is an exact $(\tilde{O}(n/h + h), 0)$-hopset with high probability.

Lemma ([Ullman/Yannakakis '90])

Every shortest path with $h/2$ edges contains skeleton with high probability.

Proof of hopset property:



Skeleton Graph

Randomized skeleton H:

1. Sample $\tilde{O}(n/h)$ *skeleton nodes* uniformly at random (+ source s)
2. Set $w_H(x, y) = \text{dist}_G^h(x, y)$ (h-hop distance)

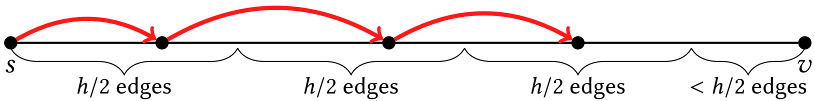
Lemma ([Klein/Subramanian '97])

Skeleton is an exact $(\tilde{O}(n/h + h), 0)$-hopset with high probability.

Lemma ([Ullman/Yannakakis '90])

Every shortest path with $h/2$ edges contains skeleton with high probability.

Proof of hopset property:



Skeleton Graph

Randomized skeleton H:

1. Sample $\tilde{O}(n/h)$ *skeleton nodes* uniformly at random (+ source s)
2. Set $w_H(x, y) = \text{dist}_G^h(x, y)$ (h-hop distance)

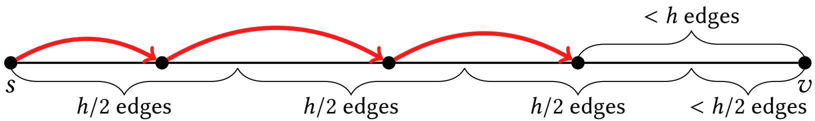
Lemma ([Klein/Subramanian '97])

Skeleton is an exact $(\tilde{O}(n/h + h), 0)$-hopset with high probability.

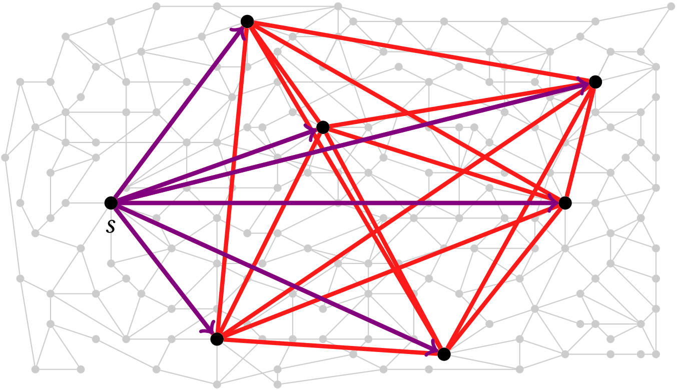
Lemma ([Ullman/Yannakakis '90])

Every shortest path with $h/2$ edges contains skeleton with high probability.

Proof of hopset property:



Skeleton Shortcuts: Intuition



Skeleton Shortcuts

1. Suppose we could compute SSSP on skeleton H
2. Shortcut edges F from s to skeleton nodes: $w_F(s, x) = \text{dist}_H(s, x)$

Skeleton Shortcuts

1. Suppose we could compute SSSP on skeleton H
2. Shortcut edges F from s to skeleton nodes: $w_F(s, x) = \mathrm{dist}_H(s, x)$

Observation

Shortcuts F are an exact *source-wise* $(h, 0)$-hopset with high probability.

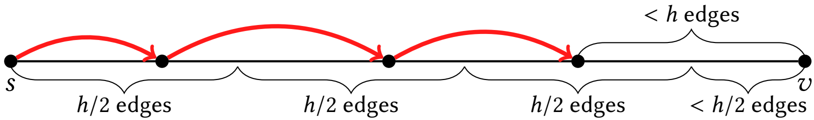
Skeleton Shortcuts

1. Suppose we could compute SSSP on skeleton H
2. Shortcut edges F from s to skeleton nodes: $w_F(s,x) = \text{dist}_H(s,x)$

Observation

Shortcuts F are an exact *source-wise* $(h,0)$-hopset with high probability.

Recall proof:



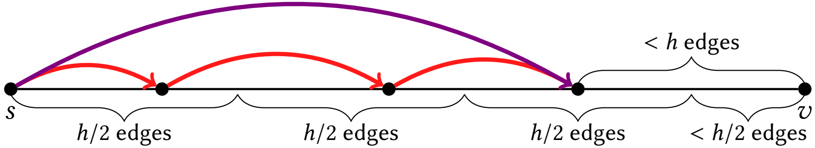
Skeleton Shortcuts

1. Suppose we could compute SSSP on skeleton H
2. Shortcut edges F from s to skeleton nodes: $w_F(s,x) = \text{dist}_H(s,x)$

Observation

Shortcuts F are an exact *source-wise* $(h, 0)$-hopset with high probability.

Recall proof:



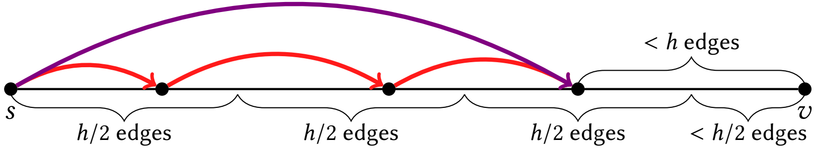
Skeleton Shortcuts

1. Suppose we could compute SSSP on skeleton H
2. Shortcut edges F from s to skeleton nodes: $w_F(s,x) = \text{dist}_H(s,x)$

Observation

Shortcuts F are an exact *source-wise* $(h,0)$-hopset with high probability.

Recall proof:



Good news:

- Cannot literally "add" shortcuts to network, but can run Bellman-Ford on $G \cup F$

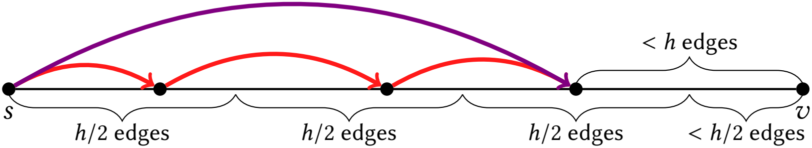
Skeleton Shortcuts

1. Suppose we could compute SSSP on skeleton H
2. Shortcut edges F from s to skeleton nodes: $w_F(s,x) = \mathrm{dist}_H(s,x)$

Observation

Shortcuts F are an exact *source-wise* $(h,0)$-hopset with high probability.

Recall proof:



Good news:

- Cannot literally “add” shortcuts to network, but can run Bellman-Ford on $G \cup F$
- Only first iteration uses shortcut edges of F
- If each skeleton node knows shortcut to s, simulate first iteration in $O(D)$ rounds

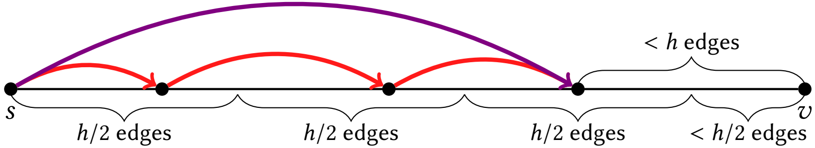
Skeleton Shortcuts

1. Suppose we could compute SSSP on skeleton H
2. Shortcut edges F from s to skeleton nodes: $w_F(s, x) = \text{dist}_H(s, x)$

Observation

Shortcuts F are an exact *source-wise* $(h, 0)$-hopset with high probability.

Recall proof:



Good news:

- Cannot literally "add" shortcuts to network, but can run Bellman-Ford on $G \cup F$
- Only first iteration uses shortcut edges of F
- If each skeleton node knows shortcut to s, simulate first iteration in $O(D)$ rounds $\rightarrow O(h + D)$ rounds

A First Idea

Algorithm 1:

1. Determine skeleton nodes: random sample of $\tilde{O}(n/h)$ nodes + s (repeat sampling if too large)
2. Compute h-hop distances from all skeleton nodes (such that $\text{dist}_G^h(x, v)$ is known to v)
3. Make skeleton known to every node
4. Determine set of shortcut edges F (Internally compute SSSP on skeleton H for every node)
5. Compute h-hop distances from s in $G \cup F$ (h Bellman-Ford iterations)

A First Idea

Algorithm 1:

1. Determine skeleton nodes: random sample of $\tilde{O}(n/h)$ nodes + s
   (repeat sampling if too large)
   Time: $O(D)$
2. Compute h-hop distances from all skeleton nodes
   (such that $\text{dist}_G^h(x, v)$ is known to v)
3. Make skeleton known to every node
   Time: $O(n^2/h^2 + D)$
4. Determine set of shortcut edges F
   (Internally compute SSSP on skeleton H for every node)
   Time: 0
5. Compute h-hop distances from s in $G \cup F$
   (h Bellman-Ford iterations)
   Time: $O(h)$

A First Idea

Algorithm 1:

1. Determine skeleton nodes: random sample of $\tilde{O}(n/h)$ nodes + s
   (repeat sampling if too large)
   Time: $O(D)$
2. Compute h-hop distances from all skeleton nodes
   (such that $\mathrm{dist}_G^h(x, v)$ is known to v)
   Time: $\tilde{O}(h \cdot n/h) = \tilde{O}(n)$ (sequential)
3. Make skeleton known to every node
   Time: $O(n^2/h^2 + D)$
4. Determine set of shortcut edges F
   (Internally compute SSSP on skeleton H for every node)
   Time: 0
5. Compute h-hop distances from s in $G \cup F$
   (h Bellman-Ford iterations)
   Time: $O(h)$

Multiple Bounded-Hop Distances

Goal: Run $\tilde{O}(n/h)$ instances of Bellman-Ford (h iterations) "in parallel"

Multiple Bounded-Hop Distances

Goal: Run $\tilde{O}(n/h)$ instances of Bellman-Ford (h iterations) "in parallel"

Obstacle:

- In each instance, every node sends to all its neighbors
- One iteration in all instances: up to $\tilde{O}(n/h)$ messages over each edge

Multiple Bounded-Hop Distances

Goal: Run $\tilde{O}(n/h)$ instances of Bellman-Ford (h iterations) "in parallel"

Obstacle:

- In each instance, every node sends to all its neighbors
- One iteration in all instances: up to $\tilde{O}(n/h)$ messages over each edge
- Bandwidth only allows one message
- Could simulate sending of $\tilde{O}(n/h)$ messages in $\tilde{O}(n/h)$ rounds

Multiple Bounded-Hop Distances

Goal: Run $\tilde{O}(n/h)$ instances of Bellman-Ford (h iterations) "in parallel"

Obstacle:

- In each instance, every node sends to all its neighbors
- One iteration in all instances: up to $\tilde{O}(n/h)$ messages over each edge
- Bandwidth only allows one message
- Could simulate sending of $\tilde{O}(n/h)$ messages in $\tilde{O}(n/h)$ rounds

Alternative to Bellman-Ford: **"Weighted BFS"**

- Replace each weighted edge e by path of $w(e)$ unweighted edges

Multiple Bounded-Hop Distances

Goal: Run $\tilde{O}(n/h)$ instances of Bellman-Ford (h iterations) "in parallel"

Obstacle:

- In each instance, every node sends to all its neighbors
- One iteration in all instances: up to $\tilde{O}(n/h)$ messages over each edge
- Bandwidth only allows one message
- Could simulate sending of $\tilde{O}(n/h)$ messages in $\tilde{O}(n/h)$ rounds

Alternative to Bellman-Ford: **"Weighted BFS"**

- Replace each weighted edge e by path of $w(e)$ unweighted edges
- Replacement can be simulated in BFS computation
- Can compute shortest paths of weight $\leq L$ in time $O(L)$

Multiple Bounded-Hop Distances

Goal: Run $\tilde{O}(n/h)$ instances of Bellman-Ford (h iterations) "in parallel"

Obstacle:

- In each instance, every node sends to all its neighbors
- One iteration in all instances: up to $\tilde{O}(n/h)$ messages over each edge
- Bandwidth only allows one message
- Could simulate sending of $\tilde{O}(n/h)$ messages in $\tilde{O}(n/h)$ rounds

Alternative to Bellman-Ford: **"Weighted BFS"**

- Replace each weighted edge e by path of $w(e)$ unweighted edges
- Replacement can be simulated in BFS computation
- Can compute shortest paths of weight $\leq L$ in time $O(L)$
- Bandwidth-friendly: at most one message per node
- Pseudopolynomial: h-hop shortest paths in time $O(hW_{\max})$

Approximate Bounded-Hop Distances

Weight rounding technique: [Klein/Subramanian '97]

- Round up weights to multiples of φ

Approximate Bounded-Hop Distances

Weight rounding technique: [Klein/Subramanian '97]

- Round up weights to multiples of φ
- Scale down rounded weights to integers
- Speed-up: shortest paths of weight $\leq L$ in time $O(L/\varphi)$

Approximate Bounded-Hop Distances

Weight rounding technique: [Klein/Subramanian '97]

- Round up weights to multiples of φ
- Scale down rounded weights to integers
- Speed-up: shortest paths of weight $\leq L$ in time $O(L/\varphi)$
- But: Each edge traversal gives additive error of φ

Approximate Bounded-Hop Distances

Weight rounding technique: [Klein/Subramanian '97]

- Round up weights to multiples of φ
- Scale down rounded weights to integers
- Speed-up: shortest paths of weight $\leq L$ in time $O(L/\varphi)$
- But: Each edge traversal gives additive error of φ
- Choice of $\varphi_i = \epsilon 2^i / h$ deals with range $2^i \leq \text{dist}^h(s, v) \leq 2^{i+1}$

Lemma ([Nanongkai '14])

Can compute $(1+\epsilon)$-approximate h-hop shortest paths from given source in $\tilde{O}(h/\epsilon)$ rounds such that each node sends $\tilde{O}(1/\epsilon)$ messages

Multiple Approximate Bounded-Hop Distances

Efficient parallelization: **Random start delays** [Leighton/Maggs/Rao '94]

- For each skeleton node: random integer delay from 0 to $\tilde{O}(n/h)$

Multiple Approximate Bounded-Hop Distances

Efficient parallelization: **Random start delays** [Leighton/Maggs/Rao '94]

- For each skeleton node: random integer delay from 0 to $\tilde{O}(n/h)$
- Results in $O(\log n)$ simultaneous messages over each edge whp
- Simulate each such round by $O(\log n)$ rounds

Multiple Approximate Bounded-Hop Distances

Efficient parallelization: **Random start delays** [Leighton/Maggs/Rao '94]

- For each skeleton node: random integer delay from 0 to $\tilde{O}(n/h)$
- Results in $O(\log n)$ simultaneous messages over each edge whp
- Simulate each such round by $O(\log n)$ rounds

Lemma ([Nanongkai '14])

Can compute $(1+\epsilon)$*-approximate skeleton of* $\tilde{O}(n/h)$ *nodes in time* $\tilde{O}(h/\epsilon + n/h)$

Multiple Approximate Bounded-Hop Distances

Efficient parallelization: **Random start delays** [Leighton/Maggs/Rao '94]

- For each skeleton node: random integer delay from 0 to $\tilde{O}(n/h)$
- Results in $O(\log n)$ simultaneous messages over each edge whp
- Simulate each such round by $O(\log n)$ rounds

Lemma ([Nanongkai '14])

Can compute $(1+\epsilon)$-approximate skeleton of $\tilde{O}(n/h)$ nodes in time $\tilde{O}(h/\epsilon + n/h)$

Remarks:

- Alternative: Weight rounding + source detection [Lenzen/Peleg '13]

Multiple Approximate Bounded-Hop Distances

Efficient parallelization: **Random start delays** [Leighton/Maggs/Rao '94]

- For each skeleton node: random integer delay from 0 to $\tilde{O}(n/h)$
- Results in $O(\log n)$ simultaneous messages over each edge whp
- Simulate each such round by $O(\log n)$ rounds

Lemma ([Nanongkai '14])

Can compute $(1+\epsilon)$*-approximate skeleton of* $\tilde{O}(n/h)$ *nodes in time* $\tilde{O}(h/\epsilon + n/h)$

Remarks:

- Alternative: Weight rounding + source detection [Lenzen/Peleg '13]
- Approximate skeleton is $(\tilde{O}(n/h + h), \epsilon)$ hopset

Refined Algorithm

Algorithm 2:

1. Determine skeleton nodes: random sample of $\tilde{O}(n/h)$ nodes + s
   (repeat sampling if too large)

2. Compute $(1+\epsilon)$-approximate h-hop distances from all skeleton nodes
   (such that $\mathrm{dist}_G^h(x, v)$ is known to v)

3. Make skeleton known to every node

4. Determine set of shortcut edges F
   (Internally compute SSSP on skeleton H for every node)

5. Compute h-hop distances from s in $G \cup F$

Refined Algorithm

Algorithm 2:

1. Determine skeleton nodes: random sample of $\tilde{O}(n/h)$ nodes + s
   (repeat sampling if too large)
   Time: $O(D)$
2. Compute $(1+\epsilon)$-approximate h-hop distances from all skeleton nodes
   (such that $\text{dist}_G^h(x,v)$ is known to v)
   Time: $\tilde{O}(h/\epsilon + n/h)$
3. Make skeleton known to every node
   Time: $O(n^2/h^2 + D)$
4. Determine set of shortcut edges F
   (Internally compute SSSP on skeleton H for every node)
   Time: 0
5. Compute h-hop distances from s in $G \cup F$
   Time: $O(h)$

Refined Algorithm

Algorithm 2:

1. Determine skeleton nodes: random sample of $\tilde{O}(n/h)$ nodes + s (repeat sampling if too large)
   Time: $O(D)$
2. Compute $(1+\epsilon)$-approximate h-hop distances from all skeleton nodes (such that $\mathrm{dist}_G^h(x, v)$ is known to v)
   Time: $\tilde{O}(h/\epsilon + n/h)$
3. Make skeleton known to every node
   Time: $O(n^2/h^2 + D)$
4. Determine set of shortcut edges F
   (Internally compute SSSP on skeleton H for every node)
   Time: 0
5. Compute h-hop distances from s in $G \cup F$
   Time: $O(h)$

Theorem

Can compute $(1+\epsilon)$*-approximate SSSP in time* $\tilde{O}(n^{2/3}/\epsilon + D)$ *with* $h = n^{2/3}$

Computing on Skeleton via Broadcast

Goal: Recurse on skeleton to improve efficiency

Computing on Skeleton via Broadcast

Goal: Recurse on skeleton to improve efficiency

Obstacle:

- Edges between skeleton nodes do not exists in communication network!
- How to run algorithm “on” skeleton?

Computing on Skeleton via Broadcast

Goal: Recurse on skeleton to improve efficiency

Obstacle:

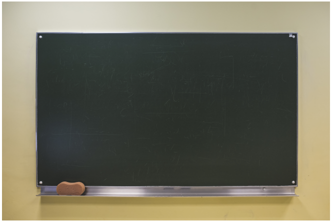
- Edges between skeleton nodes do not exists in communication network!
- How to run algorithm “on” skeleton?

Idea: Simulate a round with total of k messages on skeleton by making all messages global knowledge in time $O(k + D)$

Reduction to Blackboard model

Blackboard model:

- Communication in synchronized rounds
- Write messages on "blackboard" to make them global knowledge
- No congestion constraint, only total size of messages is relevant

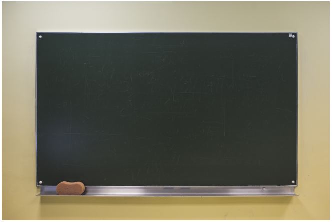


Reduction to Blackboard model

Blackboard model:

- Communication in synchronized rounds
- Write messages on “blackboard” to make them global knowledge
- No congestion constraint, only total size of messages is relevant

(Shared-memory clique??)

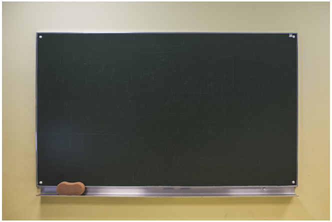


Reduction to Blackboard model

Blackboard model:

- Communication in synchronized rounds
- Write messages on "blackboard" to make them global knowledge
- No congestion constraint, only total size of messages is relevant

(Shared-memory clique??)



Lemma ([Nanongkai '14])

Any algorithm with $R(k)$ rounds and messages of total size $M(k)$ in blackboard model, can be simulated on skeleton of k nodes in $\tilde{O}(M(k) + R(k)D)$ rounds in the CONGEST model.

Back to Our Algorithm

Algorithm 3:

1. Determine skeleton nodes: random sample of $\tilde{O}(n/h)$ nodes + s
2. Compute $(1+\epsilon)$-approximate h-hop distances from all skeleton nodes
3. Compute $(1+\epsilon)$-approximate shortest paths from s on skeleton
   Simulate Algorithm 2 with $R(k) = \tilde{O}(h'/\epsilon)$ and $M(k) = k^2/(h\epsilon)$ where $k = \tilde{O}(n/h)$.
4. Determine set of shortcut edges F
5. Compute h-hop distances from s in $G \cup F$

Back to Our Algorithm

Algorithm 3:

1. Determine skeleton nodes: random sample of $\tilde{O}(n/h)$ nodes + s
   Time: $O(D)$
2. Compute $(1+\epsilon)$-approximate h-hop distances from all skeleton nodes
   Time: $\tilde{O}(h/\epsilon + n/h)$
3. Compute $(1+\epsilon)$-approximate shortest paths from s on skeleton
   Simulate Algorithm 2 with $R(k) = \tilde{O}(h'/\epsilon)$ and $M(k) = k^2/(h\epsilon)$ where $k = \tilde{O}(n/h)$.
   Time: $O(n^2/(\epsilon h^2 h') + Dh'/\epsilon)$
4. Determine set of shortcut edges F
   Time: 0
5. Compute h-hop distances from s in $G \cup F$
   Time: $O(h)$

Back to Our Algorithm

Algorithm 3:

1. Determine skeleton nodes: random sample of $\tilde{O}(n/h)$ nodes + s
   Time: $O(D)$
2. Compute $(1+\epsilon)$-approximate h-hop distances from all skeleton nodes
   Time: $\tilde{O}(h/\epsilon + n/h)$
3. Compute $(1+\epsilon)$-approximate shortest paths from s on skeleton
   Simulate Algorithm 2 with $R(k) = \tilde{O}(h'/\epsilon)$ and $M(k) = k^2/(h\epsilon)$ where $k = \tilde{O}(n/h)$.
   Time: $O(n^2/(\epsilon h^2 h') + Dh'/\epsilon)$
4. Determine set of shortcut edges F
   Time: 0
5. Compute h-hop distances from s in $G \cup F$
   Time: $O(h)$

Theorem ([F/Nanongkai '18])

Can compute $(1+\epsilon)$*-approximate SSSP in time* $\tilde{O}((\sqrt{n}D^{1/4} + D)/\epsilon)$ *with* $h = \sqrt{n}D^{1/4}$ *and* $h' = \sqrt{n}/D^{3/4}$

Exact SSSP

Scaling Approach

Two scaling techniques [Gabow '85]:

1. **Bitwise scaling:** In each iteration read next bit of weights
2. **Recursive scaling:** Reduce maximum distance by potential transformation with approximate distances

Scaling Approach

Two scaling techniques [Gabow '85]:

1. **Bitwise scaling:** In each iteration read next bit of weights
2. **Recursive scaling:** Reduce maximum distance by potential transformation with approximate distances

We follow recursive scaling:

- Similar to [Klein/Subramanian '97] in PRAM model

Scaling Approach

Two scaling techniques [Gabow '85]:

1. **Bitwise scaling:** In each iteration read next bit of weights
2. **Recursive scaling:** Reduce maximum distance by potential transformation with approximate distances

We follow recursive scaling:

- Similar to [Klein/Subramanian '97] in PRAM model
- Compute approximate distances: $\frac{1}{2} \cdot \mathrm{dist}_G(s, v) \leq \hat{d}(s, v) \leq \mathrm{dist}_G(s, v)$

Scaling Approach

Two scaling techniques [Gabow '85]:

1. **Bitwise scaling:** In each iteration read next bit of weights
2. **Recursive scaling:** Reduce maximum distance by potential transformation with approximate distances

We follow recursive scaling:

- Similar to [Klein/Subramanian '97] in PRAM model
- Compute approximate distances: $\frac{1}{2} \cdot \mathrm{dist}_G(s, v) \leq \hat{d}(s, v) \leq \mathrm{dist}_G(s, v)$
- Potential transformation: $w'(u, v) = w_G(u, v) + \hat{d}(s, u) - \hat{d}(s, v)$
  Does not change shortest paths

Scaling Approach

Two scaling techniques [Gabow '85]:

1. **Bitwise scaling:** In each iteration read next bit of weights
2. **Recursive scaling:** Reduce maximum distance by potential transformation with approximate distances

We follow recursive scaling:

- Similar to [Klein/Subramanian '97] in PRAM model
- Compute approximate distances: $\frac{1}{2} \cdot \mathrm{dist}_G(s, v) \leq \hat{d}(s, v) \leq \mathrm{dist}_G(s, v)$
- Potential transformation: $w'(u, v) = w_G(u, v) + \hat{d}(s, u) - \hat{d}(s, v)$
  Does not change shortest paths
- Solve recursively with weights w': Maximum distance has halved!

Scaling Approach

Two scaling techniques [Gabow '85]:

1. **Bitwise scaling:** In each iteration read next bit of weights
2. **Recursive scaling:** Reduce maximum distance by potential transformation with approximate distances

We follow recursive scaling:

- Similar to [Klein/Subramanian '97] in PRAM model
- Compute approximate distances: $\frac{1}{2} \cdot \mathrm{dist}_G(s, v) \leq \hat{d}(s, v) \leq \mathrm{dist}_G(s, v)$
- Potential transformation: $w'(u, v) = w_G(u, v) + \hat{d}(s, u) - \hat{d}(s, v)$
  Does not change shortest paths
- Solve recursively with weights w': Maximum distance has halved!
- **But:** Want to keep edge weights non-negative

Scaling Approach

Two scaling techniques [Gabow '85]:

1. **Bitwise scaling:** In each iteration read next bit of weights
2. **Recursive scaling:** Reduce maximum distance by potential transformation with approximate distances

We follow recursive scaling:

- Similar to [Klein/Subramanian '97] in PRAM model
- Compute approximate distances: $\frac{1}{2} \cdot \mathrm{dist}_G(s, v) \leq \hat{d}(s, v) \leq \mathrm{dist}_G(s, v)$
- Potential transformation: $w'(u, v) = w_G(u, v) + \hat{d}(s, u) - \hat{d}(s, v)$
  Does not change shortest paths
- Solve recursively with weights w': Maximum distance has halved!
- **But:** Want to keep edge weights non-negative
- Additional constraint: $\hat{d}(s, v) \leq \hat{d}(s, u) + w_G(u, v)$

Reduction

Theorem ([Klein/Subramanian '97])

Suppose auxiliary algorithm computes distance estimate $\hat{d}(s, \cdot)$ such that

- *For every node v:* $\frac{1}{2} \cdot \mathrm{dist}_G(s, v) \leq \hat{d}(s, v) \leq \mathrm{dist}_G(s, v)$ ***(approximation)***
- *For every edge (u, v):* $\hat{d}(s, v) \leq \hat{d}(s, u) + w_G(u, v)$ ***(domination)***

Then exact SSSP can be computed by calling auxiliary algorithm $O(\log(nW_{\max}))$ times (+ bookkeeping work).

Reduction

Theorem ([Klein/Subramanian '97])

Suppose auxiliary algorithm computes distance estimate $\hat{d}(s, \cdot)$ *such that*

- *For every node* v*:* $\frac{1}{2} \cdot \mathrm{dist}_G(s, v) \leq \hat{d}(s, v) \leq \mathrm{dist}_G(s, v)$ ***(approximation)***
- *For every edge* (u, v)*:* $\hat{d}(s, v) \leq \hat{d}(s, u) + w_G(u, v)$ ***(domination)***

Then exact SSSP can be computed by calling auxiliary algorithm $O(\log(nW_{\max}))$ *times (+ bookkeeping work).*

Our contribution: Design suitable auxiliary algorithm

Reduction

Theorem ([Klein/Subramanian '97])

Suppose auxiliary algorithm computes distance estimate $\hat{d}(s, \cdot)$ *such that*

- *For every node* v*:* $\frac{1}{2} \cdot \mathrm{dist}_G(s, v) \leq \hat{d}(s, v) \leq \mathrm{dist}_G(s, v)$ ***(approximation)***
- *For every edge* (u, v)*:* $\hat{d}(s, v) \leq \hat{d}(s, u) + w_G(u, v)$ ***(domination)***

Then exact SSSP can be computed by calling auxiliary algorithm $O(\log(nW_{\max}))$ *times (+ bookkeeping work).*

Our contribution: Design suitable auxiliary algorithm

- Leverage techniques from *approximate* SSSP

Reduction

Theorem ([Klein/Subramanian '97])

Suppose auxiliary algorithm computes distance estimate $\hat{d}(s, \cdot)$ *such that*

- *For every node* v*:* $\frac{1}{2} \cdot \mathrm{dist}_G(s, v) \leq \hat{d}(s, v) \leq \mathrm{dist}_G(s, v)$ ***(approximation)***
- *For every edge* (u, v)*:* $\hat{d}(s, v) \leq \hat{d}(s, u) + w_G(u, v)$ ***(domination)***

Then exact SSSP can be computed by calling auxiliary algorithm $O(\log(nW_{\max}))$ *times (+ bookkeeping work).*

Our contribution: Design suitable auxiliary algorithm

- Leverage techniques from *approximate* SSSP
- Careful design to satisfy domination constraint

Reduction

Theorem ([Klein/Subramanian '97])

Suppose auxiliary algorithm computes distance estimate $\hat{d}(s, \cdot)$ *such that*

- *For every node* v: $\frac{1}{2} \cdot \mathrm{dist}_G(s, v) \leq \hat{d}(s, v) \leq \mathrm{dist}_G(s, v)$ ***(approximation)***
- *For every edge* (u, v): $\hat{d}(s, v) \leq \hat{d}(s, u) + w_G(u, v)$ ***(domination)***

Then exact SSSP can be computed by calling auxiliary algorithm $O(\log(nW_{\max}))$ *times (+ bookkeeping work).*

Our contribution: Design suitable auxiliary algorithm

- Leverage techniques from *approximate* SSSP
- Careful design to satisfy domination constraint

Fine print:

- Inherent dependence on $\log(W_{\max})$ to bound maximum distance

Reduction

Theorem ([Klein/Subramanian '97])

Suppose auxiliary algorithm computes distance estimate $\hat{d}(s, \cdot)$ *such that*

- *For every node* v*:* $\frac{1}{2} \cdot \mathrm{dist}_G(s, v) \leq \hat{d}(s, v) \leq \mathrm{dist}_G(s, v)$ ***(approximation)***
- *For every edge* (u, v)*:* $\hat{d}(s, v) \leq \hat{d}(s, u) + w_G(u, v)$ ***(domination)***

Then exact SSSP can be computed by calling auxiliary algorithm $O(\log(nW_{\max}))$ *times (+ bookkeeping work).*

Our contribution: Design suitable auxiliary algorithm

- Leverage techniques from *approximate* SSSP
- Careful design to satisfy domination constraint

Fine print:

- Inherent dependence on $\log(W_{\max})$ to bound maximum distance
- Must solve directed problem

Reduction

Theorem ([Klein/Subramanian '97])

Suppose auxiliary algorithm computes distance estimate $\hat{d}(s,\cdot)$ *such that*

- *For every node* v*:* $\frac{1}{2}\cdot \mathrm{dist}_G(s,v) \le \hat{d}(s,v) \le \mathrm{dist}_G(s,v)$ ***(approximation)***
- *For every edge* (u,v)*:* $\hat{d}(s,v) \le \hat{d}(s,u) + w_G(u,v)$ ***(domination)***

Then exact SSSP can be computed by calling auxiliary algorithm $O(\log(nW_{\max}))$ *times (+ bookkeeping work).*

Our contribution: Design suitable auxiliary algorithm

- Leverage techniques from *approximate* SSSP
- Careful design to satisfy domination constraint

Fine print:

- Inherent dependence on $\log(W_{\max})$ to bound maximum distance
- Must solve directed problem
- Must accept 0-weight edges

Reduction

Theorem ([Klein/Subramanian '97])

Suppose auxiliary algorithm computes distance estimate $\hat{d}(s, \cdot)$ *such that*

- *For every node* v: $\frac{1}{2} \cdot \mathrm{dist}_G(s, v) \leq \hat{d}(s, v) \leq \mathrm{dist}_G(s, v)$ ***(approximation)***
- *For every edge* (u, v): $\hat{d}(s, v) \leq \hat{d}(s, u) + w_G(u, v)$ ***(domination)***

Then exact SSSP can be computed by calling auxiliary algorithm $O(\log(nW_{\max}))$ *times (+ bookkeeping work).*

Our contribution: Design suitable auxiliary algorithm

- Leverage techniques from *approximate* SSSP
- Careful design to satisfy domination constraint

Fine print:

- Inherent dependence on $\log(W_{\max})$ to bound maximum distance
- Must solve directed problem
- Must accept 0-weight edges
  $\rightarrow$ Reduction to positive edge weights

Auxiliary Algorithm

1. Determine skeleton nodes: random sample of $\tilde{O}(n/h)$ nodes + s
2. Compute $\frac{1}{2}$-approximate h-hop distances from all skeleton nodes (Compute 2-approximation and scale down)
3. Compute *exact* SSSP on skeleton
4. Determine set of shortcut edges F
5. Compute h-hop distances from s in $G \cup F$

Auxiliary Algorithm

1. Determine skeleton nodes: random sample of $\tilde{O}(n/h)$ nodes + s
   Time: $O(D)$
2. Compute $\frac{1}{2}$-approximate h-hop distances from all skeleton nodes
   (Compute 2-approximation and scale down)
   Time: $\tilde{O}(h/\epsilon + n/h)$
3. Compute *exact* SSSP on skeleton
   Time: ???
4. Determine set of shortcut edges F
   Time: 0
5. Compute h-hop distances from s in $G \cup F$
   Time: $O(h)$

Auxiliary Algorithm

1. Determine skeleton nodes: random sample of $\tilde{O}(n/h)$ nodes + s
   Time: $O(D)$
2. Compute $\frac{1}{2}$-approximate h-hop distances from all skeleton nodes (Compute 2-approximation and scale down)
   Time: $\tilde{O}(h/\epsilon + n/h)$
3. Compute *exact* SSSP on skeleton
   Time: ???
4. Determine set of shortcut edges F
   Time: 0
5. Compute h-hop distances from s in $G \cup F$
   Time: $O(h)$

Theorem

- *For every node v:* $\frac{1}{2} \cdot \mathrm{dist}_G(s,v) \leq \hat{d}(s,v) \leq \mathrm{dist}_G(s,v)$ *(approximation)*
- *For every edge (u,v):* $\hat{d}(s,v) \leq \hat{d}(s,u) + w_G(u,v)$ *(domination)*

Proof of Domination

- Need to show: $\mathrm{dist}^h_{G \cup F}(s, v) \le \mathrm{dist}^h_{G \cup F}(s, u) + w_G(u, v)$

Proof of Domination

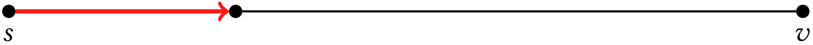
- Need to show: $\text{dist}^h_{G\cup F}(s, v) \leq \text{dist}^h_{G\cup F}(s, u) + w_G(u, v)$
- We show that $\text{dist}^h_{G\cup F}(s, v) = \text{dist}_{G\cup F}(s, v)$

Proof of Domination

- Need to show: $\text{dist}^h_{G \cup F}(s, v) \leq \text{dist}^h_{G \cup F}(s, u) + w_G(u, v)$
- We show that $\text{dist}^h_{G \cup F}(s, v) = \text{dist}_{G \cup F}(s, v)$
- Then domination follows from triangle inequality

Proof of Domination

- Need to show: $\mathrm{dist}^h_{G \cup F}(s, v) \leq \mathrm{dist}^h_{G \cup F}(s, u) + w_G(u, v)$
- We show that $\mathrm{dist}^h_{G \cup F}(s, v) = \mathrm{dist}_{G \cup F}(s, v)$
- Then domination follows from triangle inequality



Proof idea:

- Shortest path in $G \cup F$ has the following structure: at most one shortcut edge to skeleton node followed by a shortest path π in G

Proof of Domination

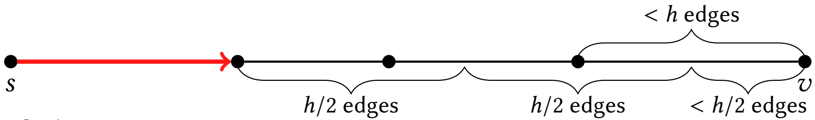
- Need to show: $\mathrm{dist}^h_{G\cup F}(s,v) \leq \mathrm{dist}^h_{G\cup F}(s,u) + w_G(u,v)$
- We show that $\mathrm{dist}^h_{G\cup F}(s,v) = \mathrm{dist}_{G\cup F}(s,v)$
- Then domination follows from triangle inequality

Proof idea:

- Shortest path in $G \cup F$ has the following structure: at most one shortcut edge to skeleton node followed by a shortest path π in G
- Subdivide π into subsequent chunks of $h/2$ edges

Proof of Domination

- Need to show: $\mathrm{dist}^h_{G \cup F}(s, v) \leq \mathrm{dist}^h_{G \cup F}(s, u) + w_G(u, v)$
- We show that $\mathrm{dist}^h_{G \cup F}(s, v) = \mathrm{dist}_{G \cup F}(s, v)$
- Then domination follows from triangle inequality

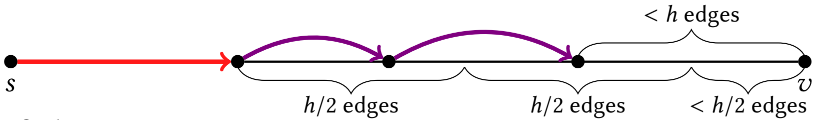


Proof idea:

- Shortest path in $G \cup F$ has the following structure: at most one shortcut edge to skeleton node followed by a shortest path π in G
- Subdivide π into subsequent chunks of $h/2$ edges
- With high probability, each chunk contains a skeleton node

Proof of Domination

- Need to show: $\text{dist}^h_{G\cup F}(s,v) \le \text{dist}^h_{G\cup F}(s,u) + w_G(u,v)$
- We show that $\text{dist}^h_{G\cup F}(s,v) = \text{dist}_{G\cup F}(s,v)$
- Then domination follows from triangle inequality

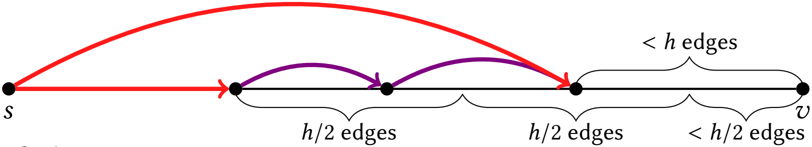


Proof idea:

- Shortest path in $G \cup F$ has the following structure: at most one shortcut edge to skeleton node followed by a shortest path π in G
- Subdivide π into subsequent chunks of $h/2$ edges
- With high probability, each chunk contains a skeleton node
- Following skeleton nodes with skeleton edges would be at least as cheap as following π (underestimated approximation!)

Proof of Domination

- Need to show: $\mathrm{dist}^h_{G \cup F}(s, v) \leq \mathrm{dist}^h_{G \cup F}(s, u) + w_G(u, v)$
- We show that $\mathrm{dist}^h_{G \cup F}(s, v) = \mathrm{dist}_{G \cup F}(s, v)$
- Then domination follows from triangle inequality

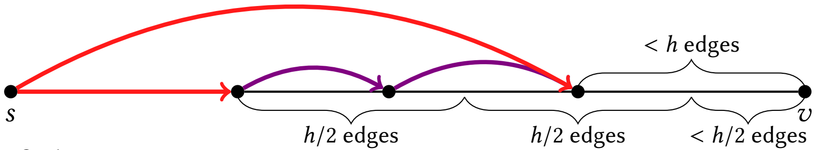


Proof idea:

- Shortest path in $G \cup F$ has the following structure: at most one shortcut edge to skeleton node followed by a shortest path π in G
- Subdivide π into subsequent chunks of $h/2$ edges
- With high probability, each chunk contains a skeleton node
- Following skeleton nodes with skeleton edges would be at least as cheap as following π (underestimated approximation!)
- Shortcut edge in $G \cup F$ to last skeleton node is as least as cheap

Proof of Domination

- Need to show: $\text{dist}^h_{G\cup F}(s,v) \leq \text{dist}^h_{G\cup F}(s,u) + w_G(u,v)$
- We show that $\text{dist}^h_{G\cup F}(s,v) = \text{dist}_{G\cup F}(s,v)$
- Then domination follows from triangle inequality

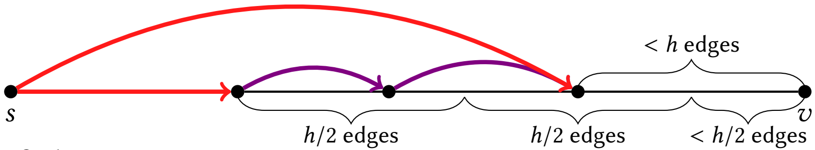


Proof idea:

- Shortest path in $G \cup F$ has the following structure: at most one shortcut edge to skeleton node followed by a shortest path π in G
- Subdivide π into subsequent chunks of $h/2$ edges
- With high probability, each chunk contains a skeleton node
- Following skeleton nodes with skeleton edges would be at least as cheap as following π (underestimated approximation!)
- Shortcut edge in $G \cup F$ to last skeleton node is as least as cheap
- Reason: Triangle inequality for exact distances!

Proof of Domination

- Need to show: $\mathrm{dist}^h_{G \cup F}(s, v) \leq \mathrm{dist}^h_{G \cup F}(s, u) + w_G(u, v)$
- We show that $\mathrm{dist}^h_{G \cup F}(s, v) = \mathrm{dist}_{G \cup F}(s, v)$
- Then domination follows from triangle inequality



Proof idea:

- Shortest path in $G \cup F$ has the following structure: at most one shortcut edge to skeleton node followed by a shortest path π in G
- Subdivide π into subsequent chunks of $h/2$ edges
- With high probability, each chunk contains a skeleton node
- Following skeleton nodes with skeleton edges would be at least as cheap as following π (underestimated approximation!)
- Shortcut edge in $G \cup F$ to last skeleton node is as least as cheap
- Reason: Triangle inequality for exact distances!
- Now: remainder of π has $< h$ edges

How to Solve on Skeleton

Recall: We need *exact* SSSP on skeleton to compute shortcuts

How to Solve on Skeleton

Recall: We need *exact* SSSP on skeleton to compute shortcuts

Two Variants:

1. Dijkstra's algorithm on skeleton
2. Recurse on skeleton using our new algorithm

How to Solve on Skeleton

Recall: We need *exact* SSSP on skeleton to compute shortcuts

Two Variants:

1. Dijkstra's algorithm on skeleton
   - $\tilde{O}(n/h)$ iterations
   - Time $O(D)$ per iteration

   Total running time: $\tilde{O}(\sqrt{nD})$
2. Recurse on skeleton using our new algorithm

How to Solve on Skeleton

Recall: We need *exact* SSSP on skeleton to compute shortcuts

Two Variants:

1. Dijkstra's algorithm on skeleton
   - $\tilde{O}(n/h)$ iterations
   - Time $O(D)$ per iteration

   Total running time: $\tilde{O}(\sqrt{nD})$

2. Recurse on skeleton using our new algorithm
   Blackboard model:
   - $R(k) = \tilde{O}(h)$ rounds
   - $M(k) = \tilde{O}(nh + n^2/h)$ messages

   Total running time: $\tilde{O}(\sqrt{n}D^{1/4} + n^{3/5} + D)$

Discussion: Implementation of Klein/Subramanian?

We borrow many ideas from PRAM algorithm of Klein and Subramanian

Discussion: Implementation of Klein/Subramanian?

We borrow many ideas from PRAM algorithm of Klein and Subramanian

Main difference:

- Klein and Subramanian: Skeleton as hopset
- Our approach: Shortcuts from skeleton

Discussion: Implementation of Klein/Subramanian?

We borrow many ideas from PRAM algorithm of Klein and Subramanian

Main difference:

- Klein and Subramanian: Skeleton as hopset
- Our approach: Shortcuts from skeleton

New trade-off for directed graphs in PRAM model:

- Klein and Subramanian: work $\tilde{O}(m\sqrt{n})$ and depth $\tilde{O}(\sqrt{n})$
- Our approach: work $\tilde{O}((n^3/h^3 + mh + mn/h))$ and depth $\tilde{O}(h)$

≈

Faster Approximation

Broadcast Congested Clique

Model:

- Network topology is a clique
- In each round, every node sends **one** message to all its neighbors

Broadcast Congested Clique

Model:

- Network topology is a clique
- In each round, every node sends **one** message to all its neighbors

Lemma

Any broadcast congested clique algorithm with $R(k)$ rounds can be simulated on skeleton of k nodes in $O((k + D)R(k))$ rounds in the CONGEST model.

Broadcast Congested Clique

Model:

- Network topology is a clique
- In each round, every node sends **one** message to all its neighbors

Lemma

Any broadcast congested clique algorithm with $R(k)$ rounds can be simulated on skeleton of k nodes in $O((k + D)R(k))$ rounds in the CONGEST model.

Theorem ([Nanongkai '14])

In directed graphs, can compute $(1 + \epsilon)$-approximate skeleton with $k = \tilde{O}(\sqrt{n})$ nodes in $\tilde{O}(\sqrt{n})$ rounds. The algorithm is correct with high probability.

Broadcast Congested Clique

Model:

- Network topology is a clique
- In each round, every node sends **one** message to all its neighbors

Lemma

Any broadcast congested clique algorithm with $R(k)$ rounds can be simulated on skeleton of k nodes in $O((k + D)R(k))$ rounds in the CONGEST model.

Theorem ([Nanongkai '14])

In directed graphs, can compute $(1 + \epsilon)$-approximate skeleton with $k = \tilde{O}(\sqrt{n})$ nodes in $\tilde{O}(\sqrt{n})$ rounds. The algorithm is correct with high probability.

Theorem ([Henzinger/K/Nanongkai '16])

In undirected graphs, can compute $(1 + \epsilon)$-approximate skeleton with $k = \tilde{O}(\sqrt{n})$ nodes deterministically in $\tilde{O}(\sqrt{n})$ rounds.

Fast Hopset Construction

Theorem ([Henzinger/K/Nanongkai '16])

Can compute $(1+\epsilon)$-approximate SSSP on undirected Broadcast Congested Clique deterministically in $n^{o(1)}$ rounds for any given $\epsilon \geq 1/\log^{O(1)}$.

Fast Hopset Construction

Theorem ([Henzinger/K/Nanongkai '16])

Can compute $(1+\epsilon)$-approximate SSSP on undirected Broadcast Congested Clique deterministically in $n^{o(1)}$ rounds for any given $\epsilon \geq 1/\log^{O(1)}$.

Recall: Given (h, ϵ)-hopset, $(1+\epsilon)$-approximate SSSP can be computed in $O(h)$ rounds.

Fast Hopset Construction

Theorem ([Henzinger/K/Nanongkai '16])

Can compute $(1+\epsilon)$-approximate SSSP on undirected Broadcast Congested Clique deterministically in $n^{o(1)}$ rounds for any given $\epsilon \geq 1/\log^{O(1)}$.

Recall: Given (h, ϵ)-hopset, $(1+\epsilon)$-approximate SSSP can be computed in $O(h)$ rounds.

Ideas:

- Observation: distance oracle of [Thorup/Zwick '05] gives $(n^{o(1)}, \epsilon)$ hopset in undirected graphs [Bernstein '09]

Fast Hopset Construction

Theorem ([Henzinger/K/Nanongkai '16])

Can compute $(1+\epsilon)$-approximate SSSP on undirected Broadcast Congested Clique deterministically in $n^{o(1)}$ rounds for any given $\epsilon \geq 1/\log^{O(1)}$.

Recall: Given (h, ϵ)-hopset, $(1+\epsilon)$-approximate SSSP can be computed in $O(h)$ rounds.

Ideas:

- Observation: distance oracle of [Thorup/Zwick '05] gives $(n^{o(1)}, \epsilon)$ hopset in undirected graphs [Bernstein '09]
- Vanilla Thorup/Zwick already requires SSSP computation
- Iterative Approach: Bounded-hop SSSP allows hop reduction

Fast Hopset Construction

Theorem ([Henzinger/K/Nanongkai '16])

Can compute $(1+\epsilon)$-approximate SSSP on undirected Broadcast Congested Clique deterministically in $n^{o(1)}$ rounds for any given $\epsilon \geq 1/\log^{O(1)}$.

Recall: Given (h, ϵ)-hopset, $(1+\epsilon)$-approximate SSSP can be computed in $O(h)$ rounds.

Ideas:

- Observation: distance oracle of [Thorup/Zwick '05] gives $(n^{o(1)}, \epsilon)$ hopset in undirected graphs [Bernstein '09]
- Vanilla Thorup/Zwick already requires SSSP computation
- Iterative Approach: Bounded-hop SSSP allows hop reduction
- Hopset is obtained after sufficiently many hop reductions

Fast Hopset Construction

Theorem ([Henzinger/K/Nanongkai '16])

Can compute $(1+\epsilon)$-approximate SSSP on undirected Broadcast Congested Clique deterministically in $n^{o(1)}$ rounds for any given $\epsilon \geq 1/\log^{O(1)}$.

Recall: Given (h, ϵ)-hopset, $(1+\epsilon)$-approximate SSSP can be computed in $O(h)$ rounds.

Ideas:

- Observation: distance oracle of [Thorup/Zwick '05] gives $(n^{o(1)}, \epsilon)$ hopset in undirected graphs [Bernstein '09]
- Vanilla Thorup/Zwick already requires SSSP computation
- Iterative Approach: Bounded-hop SSSP allows hop reduction
- Hopset is obtained after sufficiently many hop reductions

Remarks:

- Hopset lower bound indicates $n^{o(1)}$ barrier [Abboud/Bodwin/Pettie '17]
- Tight hopsets exist [Huang/Pettie '17] [Elkin/Neiman '17]

Gradient Descent Approach

Theorem ([Becker/Karrenbauer/K/Lenzen '17])

Can compute $(1+\epsilon)$-approximate SSSP on undirected Broadcast Congested Clique in $\log^{O(1)} n/\epsilon^{O(1)}$ rounds with high probability

Gradient Descent Approach

Theorem ([Becker/Karrenbauer/**K**/Lenzen '17])

Can compute $(1+\epsilon)$-approximate SSSP on undirected Broadcast Congested Clique in $\log^{O(1)} n/\epsilon^{O(1)}$ rounds with high probability

Linear Programming Formulation

Primal:	minimize $\|Wx\|_1$	s.t. $Ax = b$
Dual:	maximize $b^T y$	s.t. $\left\|W^{-1}A^T y\right\|_\infty \leq 1$

Gradient Descent Approach

Theorem ([Becker/Karrenbauer/K/Lenzen '17])

Can compute $(1+\epsilon)$*-approximate SSSP on undirected Broadcast Congested Clique in* $\log^{O(1)} n/\epsilon^{O(1)}$ *rounds with high probability*

Linear Programming Formulation

Primal: minimize $\|Wx\|_1$ s.t. $Ax = b$

Dual: maximize $b^T y$ s.t. $\left\|W^{-1}A^T y\right\|_\infty \leq 1$

- More general problem: Uncapacitated minimum-cost flow

Gradient Descent Approach

Theorem ([Becker/Karrenbauer/K/Lenzen '17])

Can compute $(1+\epsilon)$*-approximate SSSP on undirected Broadcast Congested Clique in* $\log^{O(1)} n/\epsilon^{O(1)}$ *rounds with high probability*

Linear Programming Formulation

Primal:	minimize $\|Wx\|_1$	s.t. $Ax = b$
Dual:	maximize $b^T y$	s.t. $\left\|W^{-1}A^T y\right\|_\infty \leq 1$

- More general problem: Uncapacitated minimum-cost flow
- Gradient descent algorithm for finding dual solution

Gradient Descent Approach

Theorem ([Becker/Karrenbauer/K/Lenzen '17])

Can compute $(1+\epsilon)$*-approximate SSSP on undirected Broadcast Congested Clique in* $\log^{O(1)} n/\epsilon^{O(1)}$ *rounds with high probability*

Linear Programming Formulation

Primal:	minimize $\|Wx\|_1$	s.t. $Ax = b$
Dual:	maximize $b^T y$	s.t. $\left\|W^{-1}A^T y\right\|_\infty \le 1$

- More general problem: Uncapacitated minimum-cost flow
- Gradient descent algorithm for finding dual solution
- Smooth approximation of infinity norm

Gradient Descent Approach

Theorem ([Becker/Karrenbauer/K/Lenzen '17])

Can compute $(1+\epsilon)$-approximate SSSP on undirected Broadcast Congested Clique in $\log^{O(1)} n/\epsilon^{O(1)}$ rounds with high probability

Linear Programming Formulation

Primal: minimize $\|Wx\|_1$ s.t. $Ax = b$

Dual: maximize $b^T y$ s.t. $\left\|W^{-1}A^T y\right\|_\infty \leq 1$

- More general problem: Uncapacitated minimum-cost flow
- Gradient descent algorithm for finding dual solution
- Smooth approximation of infinity norm
- Find good update step by routing gradient via a spanner

Gradient Descent Approach

Theorem ([Becker/Karrenbauer/K/Lenzen '17])

Can compute $(1+\epsilon)$*-approximate SSSP on undirected Broadcast Congested Clique in* $\log^{O(1)} n/\epsilon^{O(1)}$ *rounds with high probability*

Linear Programming Formulation

Primal: minimize $\|Wx\|_1$ s.t. $Ax = b$

Dual: maximize $b^T y$ s.t. $\left\|W^{-1}A^T y\right\|_\infty \leq 1$

- More general problem: Uncapacitated minimum-cost flow
- Gradient descent algorithm for finding dual solution
- Smooth approximation of infinity norm
- Find good update step by routing gradient via a spanner
- Crux: Another transshipment instance on sparser graph

Gradient Descent Approach

Theorem ([Becker/Karrenbauer/K/Lenzen '17])

Can compute $(1+\epsilon)$-approximate SSSP on undirected Broadcast Congested Clique in $\log^{O(1)} n/\epsilon^{O(1)}$ rounds with high probability

Linear Programming Formulation

Primal: minimize $\|Wx\|_1$ s.t. $Ax = b$

Dual: maximize $b^T y$ s.t. $\left\|W^{-1}A^T y\right\|_\infty \leq 1$

- More general problem: Uncapacitated minimum-cost flow
- Gradient descent algorithm for finding dual solution
- Smooth approximation of infinity norm
- Find good update step by routing gradient via a spanner
- Crux: Another transshipment instance on sparser graph
- Randomized rounding approach for primal tree solution

Gradient Descent Approach

Theorem ([Becker/Karrenbauer/K/Lenzen '17])

Can compute $(1+\epsilon)$*-approximate SSSP on undirected Broadcast Congested Clique in* $\log^{O(1)} n/\epsilon^{O(1)}$ *rounds with high probability*

Linear Programming Formulation

Primal:	minimize $\|Wx\|_1$	s.t. $Ax = b$
Dual:	maximize $b^T y$	s.t. $\left\|W^{-1}A^T y\right\|_\infty \leq 1$

- More general problem: Uncapacitated minimum-cost flow
- Gradient descent algorithm for finding dual solution
- Smooth approximation of infinity norm
- Find good update step by routing gradient via a spanner
- Crux: Another transshipment instance on sparser graph
- Randomized rounding approach for primal tree solution
- Due to approximation error: tree solution only bounds sum of distances (*on average* guarantee)

Gradient Descent Approach

Theorem ([Becker/Karrenbauer/K/Lenzen '17])

Can compute $(1+\epsilon)$*-approximate SSSP on undirected Broadcast Congested Clique in* $\log^{O(1)} n/\epsilon^{O(1)}$ *rounds with high probability*

Linear Programming Formulation

Primal: minimize $\|Wx\|_1$ s.t. $Ax = b$

Dual: maximize $b^T y$ s.t. $\left\|W^{-1}A^T y\right\|_\infty \leq 1$

- More general problem: Uncapacitated minimum-cost flow
- Gradient descent algorithm for finding dual solution
- Smooth approximation of infinity norm
- Find good update step by routing gradient via a spanner
- Crux: Another transshipment instance on sparser graph
- Randomized rounding approach for primal tree solution
- Due to approximation error: tree solution only bounds sum of distances (*on average* guarantee)
- Markov-style argument for finding approximate distances

Conclusion

Take-home message:

- Wide array of techniques
- Approximate SSSP with nearly tight running time
- Exact SSSP seems in reach

Conclusion

Take-home message:

- Wide array of techniques
- Approximate SSSP with nearly tight running time
- Exact SSSP seems in reach

Open problems:

1. Match single-source reachability barrier

Conclusion

Take-home message:

- Wide array of techniques
- Approximate SSSP with nearly tight running time
- Exact SSSP seems in reach

Open problems:

1. Match single-source reachability barrier
   - Reachability: $\tilde{O}(\sqrt{n}D^{1/4} + D)$ rounds [Ghaffari/Udwani '15]

Conclusion

Take-home message:

- Wide array of techniques
- Approximate SSSP with nearly tight running time
- Exact SSSP seems in reach

Open problems:

1. Match single-source reachability barrier
    - Reachability: $\tilde{O}(\sqrt{n}D^{1/4} + D)$ rounds [Ghaffari/Udwani '15]
    - Bottleneck: $R(k) = \tilde{O}(h)$ rounds and $M(k) = \tilde{O}(nh + n^2/h)$ messages in blackboard model

Conclusion

Take-home message:

- Wide array of techniques
- Approximate SSSP with nearly tight running time
- Exact SSSP seems in reach

Open problems:

1. Match single-source reachability barrier
   - Reachability: $\tilde{O}(\sqrt{n}D^{1/4} + D)$ rounds [Ghaffari/Udwani '15]
   - Bottleneck: $R(k) = \tilde{O}(h)$ rounds and $M(k) = \tilde{O}(nh + n^2/h)$ messages in blackboard model
   - Also open in PRAM model

Conclusion

Take-home message:

- Wide array of techniques
- Approximate SSSP with nearly tight running time
- Exact SSSP seems in reach

Open problems:

1. Match single-source reachability barrier
   - Reachability: $\tilde{O}(\sqrt{n}D^{1/4} + D)$ rounds [Ghaffari/Udwani '15]
   - Bottleneck: $R(k) = \tilde{O}(h)$ rounds and $M(k) = \tilde{O}(nh + n^2/h)$ messages in blackboard model
   - Also open in PRAM model
2. Find deterministic sublinear exact algorithm

Conclusion

Take-home message:

- Wide array of techniques
- Approximate SSSP with nearly tight running time
- Exact SSSP seems in reach

Open problems:

1. Match single-source reachability barrier
    - Reachability: $\tilde{O}(\sqrt{n}D^{1/4} + D)$ rounds [Ghaffari/Udwani '15]
    - Bottleneck: $R(k) = \tilde{O}(h)$ rounds and $M(k) = \tilde{O}(nh + n^2/h)$ messages in blackboard model
    - Also open in PRAM model
2. Find deterministic sublinear exact algorithm
3. Is $\tilde{O}(\sqrt{n})$ rounds tight on *Broadcast* Congested Clique?

Thank you!

slides: https://www.cosy.sbg.ac.at/~forster/