Parallel Approximate Undirected Shortest Paths Via Low Hop Emulators

Alexandr Andoni, Clifford Stein, Peilin Zhong

Columbia University

April 20, 2020

Motivation for Parallel Algorithms

Assess the efficiency of parallel algorithms by analyzing:

- Depth, $D_P(n)$, is length of the longest series of operations
- Work, $W_P(n)$, is runtime in a sequential setting

We seek algorithms that are:

- Polylog(arithmic) depth , $D_P(n) = \mathcal{O}\big(\text{poly}(\log n)\big)$
- Nearly work-efficient, $W_P(n) = \mathcal{O}\big(T_S(n) \cdot \text{poly}(\log n)\big)$

Parallel algorithms for shortest paths problem

Given a (non–negative) weighted graph $G = (V, E)$ shortest paths problem finds the shortest path from $s \to t$ in G.

The single source shortest paths problem is the shortest path from $s \in S$ to every vertex in V.

Let $n = |V|$ and $m = |E|$. What algorithms are there?

Parallel algorithms for shortest paths problem

Given a (non–negative) weighted graph $G = (V, E)$ shortest paths problem finds the shortest path from $s \to t$ in G.

The single source shortest paths problem is the shortest path from $s \in S$ to every vertex in V.

Let $n = |V|$ and $m = |E|$. What algorithms are there?

- Dijkstra's algorithm requires $\mathcal{O}(m + n \cdot \log(n))$ work/depth
- Bellman Ford requires $\mathcal{O}(nm)$ work and $\mathcal{O}(n)$ depth
- Floyd–Warshall requires $\mathcal{O}(n^3)$ work and $\mathcal{O}(\log n)$ depth

Parallel algorithms for shortest paths problem

Given a (non–negative) weighted graph $G = (V, E)$ shortest paths problem finds the shortest path from $s \to t$ in G.

The single source shortest paths problem is the shortest path from $s \in S$ to every vertex in V.

Let $n = |V|$ and $m = |E|$. What algorithms are there?

- Dijkstra's algorithm requires $\mathcal{O}(m + n \cdot \log(n))$ work/depth
- Bellman Ford requires $\mathcal{O}(nm)$ work and $\mathcal{O}(n)$ depth
- Floyd–Warshall requires $\mathcal{O}(n^3)$ work and $\mathcal{O}(\log n)$ depth

Question: Does there exist a $(1 + \varepsilon)$–shortest paths algorithm with $\mathcal{O}\big(m \cdot \text{poly}(\log n)\big)$ work and $\mathcal{O}(\text{poly}(\log n))$ depth?

Previous and New Results

Previous Results

1. Use hopset [Coh94] to solve $(1+\varepsilon)$–shortest paths in polylog depth and $\Omega(n^{2.1})$ work
2. Any $(1+\varepsilon)$–shortest paths algorithm with polylog depth using only hopsets requires $\Omega(n^{1+\varepsilon'})$ work [ABP18]

Previous and New Results

Previous Results

1. Use hopset [Coh94] to solve $(1+\varepsilon)$–shortest paths in polylog depth and $\Omega(n^{2.1})$ work

2. Any $(1+\varepsilon)$–shortest paths algorithm with polylog depth using only hopsets requires $\Omega(n^{1+\varepsilon'})$ work [ABP18]

This Paper

1. Construct a novel data structure, a low hop emulator, to solve approximate single source shortest paths

2. Compute $(1+\varepsilon)$–approximate shortest path in polylog depth and nearly linear work via [She17] and low hop emulators

Previous and New Results

Previous Results

1. Use hopset [Coh94] to solve $(1+\varepsilon)$–shortest paths in polylog depth and $\Omega(n^{2.1})$ work
2. Any $(1+\varepsilon)$–shortest paths algorithm with polylog depth using only hopsets requires $\Omega(n^{1+\varepsilon'})$ work [ABP18]

This Paper

1. Construct a novel data structure, a low hop emulator, to solve approximate single source shortest paths
2. Compute $(1+\varepsilon)$–approximate shortest path in polylog depth and nearly linear work via [She17] and low hop emulators

Given a graph $G = (V, E)$, a low hop emulator is a weighted graph $H = (V, F)$ where any shortest $(s-t)$–path with $\mathcal{O}(\log\log n)$ edge traversals and $|F| = \mathcal{O}\big(m \cdot \mathrm{poly}(\log n)\big)$.

Overview

1. Constructing a Low Hop Emulator
   - Constructing a subemulator
   - Recursive subemulators
   - Collapsing into a low–hop emulator
   - Constructing a low–hop emulator (in parallel)

2. $(1+\varepsilon)$ shortest paths in polylog depth and nearly linear work
   - Solving shortest paths via optimization and Sherman's framework

Constructing a low hop emulator

To construct a low hop emulator,

1. Construct a subemulator
2. Recursive subemulators
3. Collapse all subemulators down to a single graph $\rightarrow$ low–hop emulator

Constructing a low hop emulator

To construct a low hop emulator,

1. Construct a subemulator
2. Recursive subemulators
3. Collapse all subemulators down to a single graph $\rightarrow$ low–hop emulator

A subemulator is a graph $H = (S, F')$ where $S \subset V$ and F' is a weighted edge set that approximates distances well.

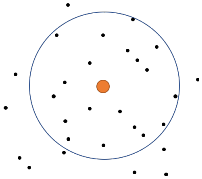
1. Every vertex $v \in V$ is close to a vertex in S
2. Distances in H approximate distances in G

Constructing a subemulator

A subemulator is a graph $H = (S, F')$ where $S \subset V$ and F' is a weighted edge set that approximates distances well.

1. Select vertices first by sampling

Our analysis depends on the ball $B_{G,b}(v)$, which is the closest b vertices (graph distance) to v in G,

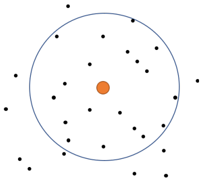


Constructing a subemulator

A subemulator is a graph $H = (S, F')$ where $S \subset V$ and F' is a weighted edge set that approximates distances well.

1. Select vertices first by sampling
2. Add more vertices to ensure vertex in V is close to a vertex in S

Our analysis depends on the ball $B_{G,b}(v)$, which is the closest b vertices (graph distance) to v in G,

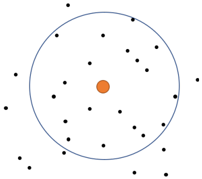


Constructing a subemulator

A subemulator is a graph $H = (S, F')$ where $S \subset V$ and F' is a weighted edge set that approximates distances well.

1. Select vertices first by sampling
2. Add more vertices to ensure vertex in V is close to a vertex in S
3. Add edges and assign weights so that local distances are well–approximated

Our analysis depends on the ball $B_{G,b}(v)$, which is the closest b vertices (graph distance) to v in G,



Selecting vertices

Fix a ball size b (to be defined later).

1. Construct S by sampling every vertex with probability $p = \min(50\frac{\log n}{b}, \frac{1}{2})$
2. If $v \in V$ is not near any vertex in S, add v to S
3. Store the leader $q(v) \leftarrow$ closest vertex $u \in S$ to $v \in V$

Output: A sparse vertex set S and mapping $q : V \to S$

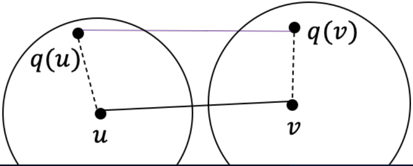
Adding edges

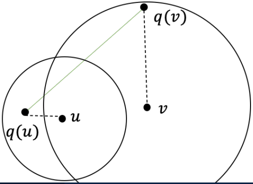
Adding edges:

1. For all $(u, v) \in E$, add edge $(q(u), q(v))$
2. Fix $v \in V$. $\forall u \in B(v)$, add edge $((q(u), q(v))$

When adding an edge $e = (q(u), q(v))$ to F, update weight

$$w(e) = \min \begin{cases} w(e), \text{(initialize to } \infty) \\ d_G(q(u), u) + d_G(u, v) + d_G(v, q(v)) \end{cases}.$$





Properties of subemulators

Given a subemulator $H = (S, F')$, Size

- $\mathbb{E}\big[|S|\big] < n$
- $|F'| \leq m + nb$

Distance approximation

- For any $u, v \in S$,

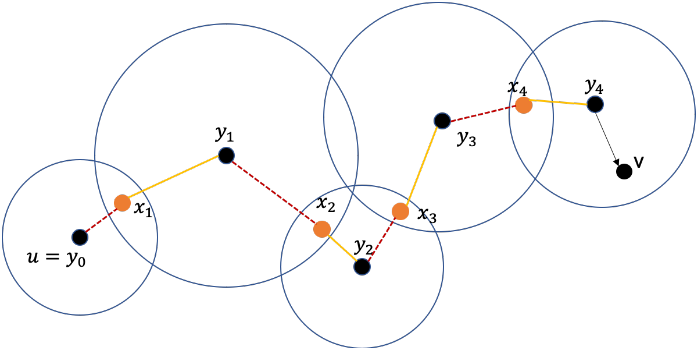
$$d_G(u, v) \leq d_H(u, v) \leq 8 \cdot d_G(u, v).$$

- For any $u, v \in V$,

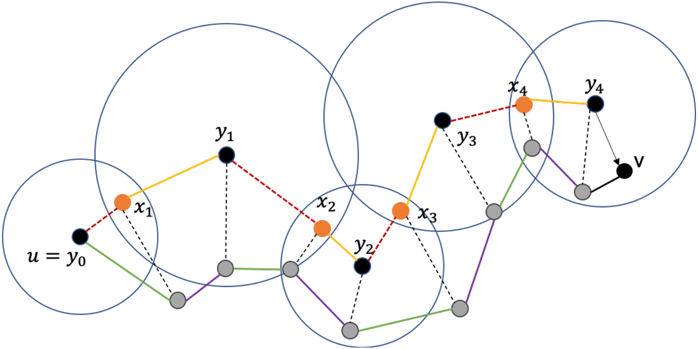
$$d_H\big(q(u), q(v)\big) \leq d_G\big(u, q(u)\big) + 22 \cdot d_G(u, v) + d_G\big(v, q(v)\big).$$

8-approximation for shortest (u, v)–path

Consider arbitrary $u, v \in S$ and its shortest path in G.



8-approximation for shortest (u, v)–path



$$
\begin{aligned}
d_H(u, v) &\leq \sum_{i=1}^{t} {\color{green} w_H\big(q(y_{i-1}), q(x_i)\big)} + {\color{purple} w_H\big(q(x_i), q(y_i)\big)} + w_H(y_t, v) \\
&\leq 8 \cdot \Big(\sum_{i=1}^{t} {\color{red} w_G(y_{i-1}, x_i)} + {\color{orange} w_G(x_i, y_i)} + w_G(y_t, v) \Big) \\
&= 8 \cdot d_G(u, v).
\end{aligned}
$$

Constructing a Low Hop Emulator

High level approach:

1. Construct a subemulator
2. Recursive subemulators
3. Collapse all subemulators down to a single graph $\rightarrow$ low–hop emulator

Recursive subemulators

Set $b_0 \ll n$.

Let the first subemulator H_0 be the original graph G.

While $|V(H_i)| \geq b_i$:

1. $\forall v \in V(H_i)$, save $B_{H_i,b_i}(v)$
2. $H_{i+1} \leftarrow \text{SUBEMULATOR}(H_i, b_i)$
3. Update $b_{i+1} \leftarrow b_i^{1.25}$ and $i \leftarrow i+1$

Output: Set of subemulators H_i and set of balls $B(v'), \forall v' \in H_i$

Recursive subemulators

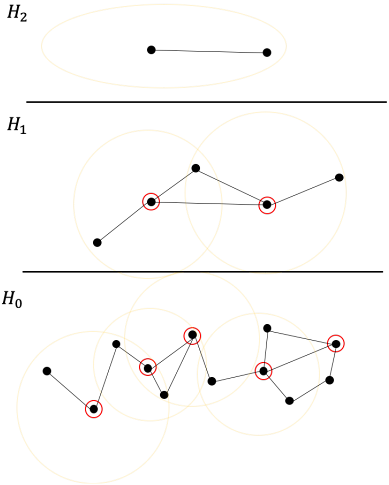


Figure: Recursive subemulators

Calculating distances from multiple subemulators

Let u, v be arbitrary vertices whose distance we want to compute.

$d = 0,\ i = 0$
$u_0 = u,\ v_0 = v$
While u_i, v_i are not in the same ball:

1. $d \leftarrow d + d_{H_i}\big(q(u_i), u_i)\big) + d_{H_i}\big(v, q(v_i)\big)$
2. $u_{i+1} \leftarrow q(u_i),\ v_{i+1} \leftarrow q(v_i)$
3. $i \leftarrow i + 1$

Return $d + d_{H_i}(u_i, v_i)$.

Calculating distances from multiple subemulators

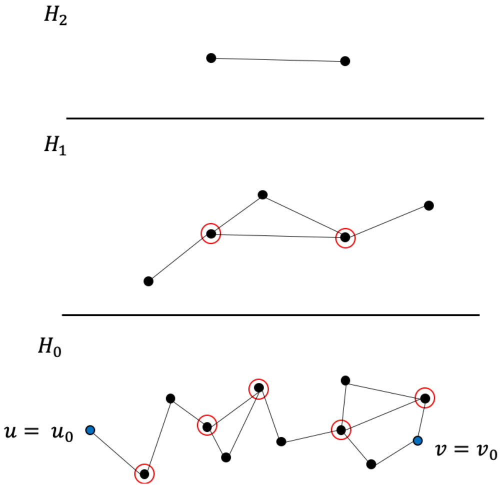


Figure: Traversing H_0

Calculating distances from multiple subemulators

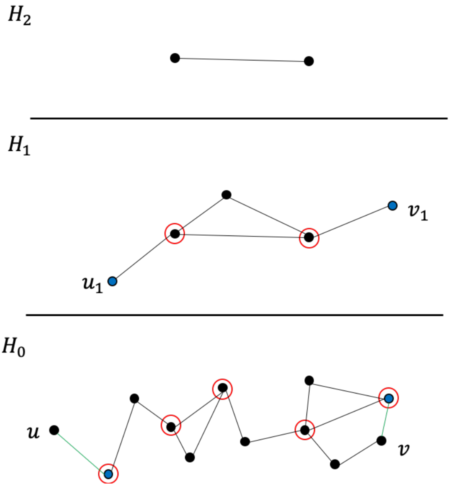


Figure: Traversing H_1

Calculating distances from multiple subemulators

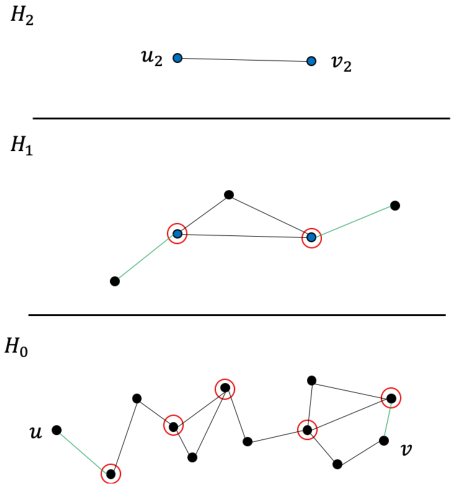


Figure: Traversing H_2

Calculating distances from multiple subemulators

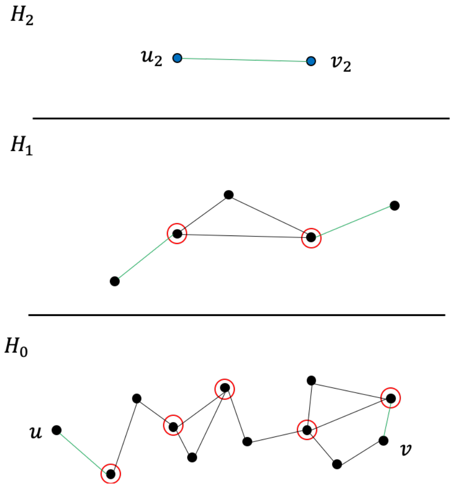


Figure: Near vertices exact distance

Properties of a distance oracle

1. Number of subemulators is $\mathcal{O}(\log\log n)$
2. Traversing up–down subemulators to get (u, v)–shortest path is $26^{\mathcal{O}(\log\log n)} = \text{poly}(\log n)$–approximation
3. Store $\mathcal{O}(\log\log n)$ subemulators and each vertex's ball

Constructing a Low Hop Emulator

High level approach:

1. Construct a subemulator
2. Recursive subemulators
3. Collapse all subemulators down to a single graph $\rightarrow$ low–hop emulator

Collapsing towards a low–hop emulator

1. $V' \leftarrow \big(V(H_0), 0\big) \bigcup \ldots \big(V(H_i), i\big) \ldots \bigcup \big(V(H_t), t\big)$
2. Add the edge from a vertex up to its leaders in next level
3. Keep all edges within each subemulator
4. Add the edge between close vertices that are within same subemulator

Let t be number of levels. Set

$$w_F\big((u,i),(v,j)\big) = 27^{t-\max(i,j)} \cdot d_{H_i}(u,v).$$

Collapsing towards a low–hop emulator

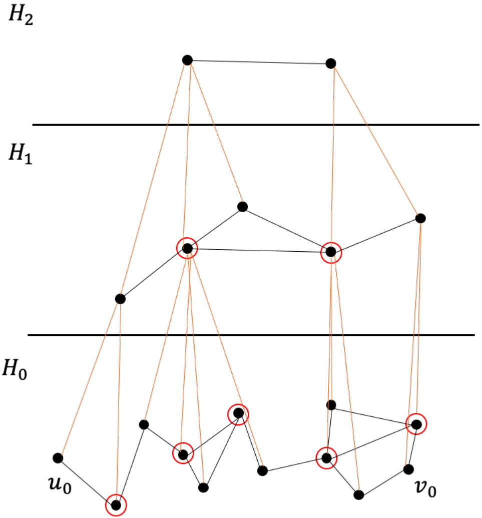


Figure: Edges between levels

Collapsing towards a low–hop emulator

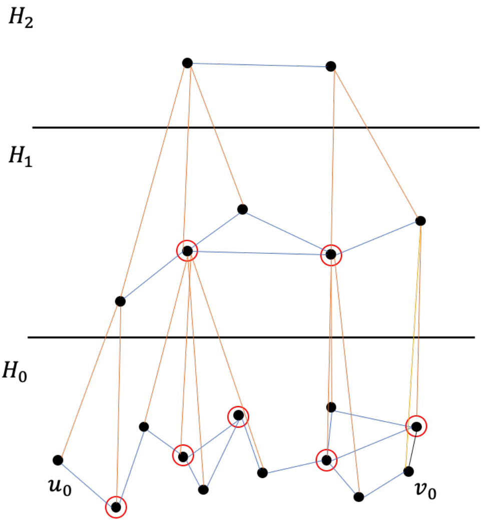


Figure: Edges within a level

Collapsing towards a low–hop emulator

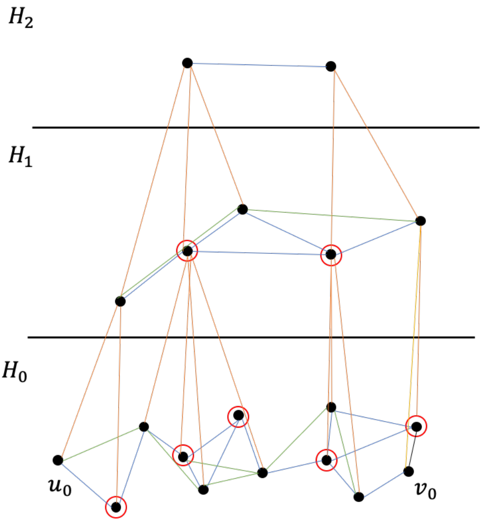


Figure: Edges between nearby vertices

Collapsing towards a low–hop emulator

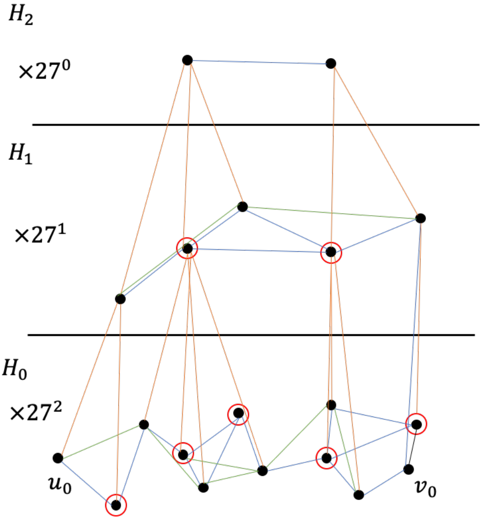


Figure: Scaling edges

Collapsing towards a low–hop emulator

1. Can remove copies of a vertex v, $(v, 0), (v, 1), \ldots$, and preserve edges by squishing all copies of vertex to bottom level

   OK since have orange edges between (v, i) and $(v, i + 1)$ are weight zero.

2. There always exists a shortest $(u - v)$–path with minimal edge traversals without using blue edges

   Close edges will be covered by green edges and far edges will never be used.

Collapsing towards a low–hop emulator

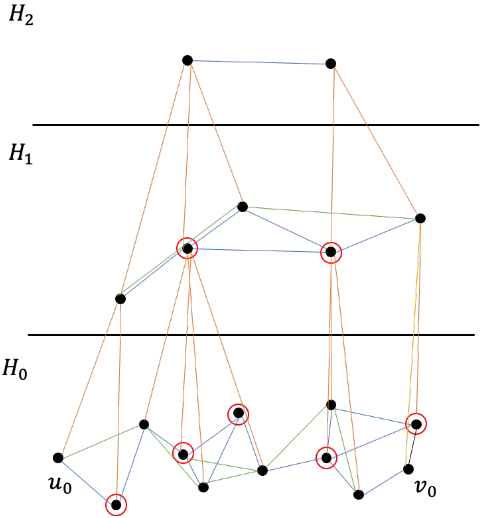


Figure: Almost low–hop emualtor

Collapsing towards a low–hop emulator

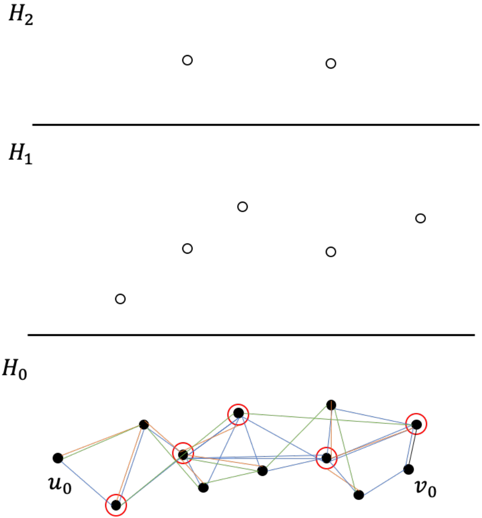


Figure: Removing redundant vertices

Collapsing towards a low–hop emulator

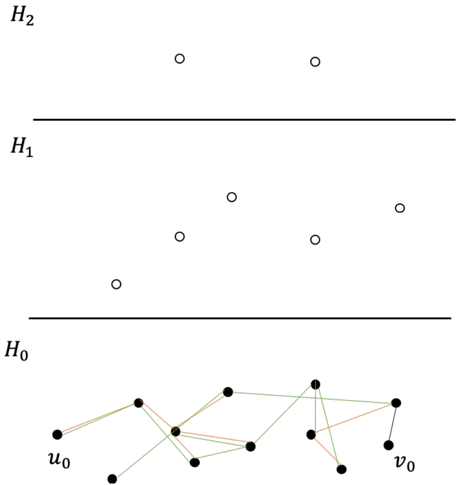


Figure: Remove blue edges

Recap

High level approach:

1. Construct a subemulator
2. Recursive subemulators
3. Collapse all subemulators down to a single graph $\rightarrow$ low–hop emulator

Recap

High level approach:

1. Construct a subemulator
2. Recursive subemulators
3. Collapse all subemulators down to a single graph $\rightarrow$ low–hop emulator

Our constructed graph $H = (V', F)$:

1. Shortest path requires $\mathcal{O}(\log\log n)$ traversals
2. Distance is $27^{\mathcal{O}(\log\log n)} = \text{poly}(\log n)$–approximated
3. $V' = V$
4. $\mathbb{E}[|F|] = \mathcal{O}\big(n \cdot \text{poly}(\log n)\big)$

Corollary: H is a low–hop emulator.

Constructing a low–hop emulator (in parallel)

Assume that we have computer that allows concurrent reads and concurrent writes (CRCW).

1. Create the ball centered at v
2. Selecting vertices
3. Adding edges
4. Recursive subemulators
5. Collapsing towards a low–hop emulator

Creating the ball centered at v (in parallel)

Uses path–doubling idea as done in Floyd–Warshall.

$L_0(v) \leftarrow$ closest b neighbors of v (including v itself)
For $u \in L_0(v)$, compute $\text{dist}^{(0)}(v, u)$

For $i = 1, \ldots, t = \lceil \log n \rceil$:

1. $\forall v, u \in V$, set $\text{dist}^{(i)}(v, u) \leftarrow \infty$
2. Fix $v \in V$. Consider (v, x, u) such that $x \in L_{i-1}(v)$ and $u \in L_{i-1}(x)$. Update $\text{dist}^{(i)}(v, u)$ if $\text{dist}^{(i-1)}(v, x) + \text{dist}^{(i-1)}(x, u)$ is smaller
3. For every $v \in V$, set $L_i(v) \leftarrow$ closest b vertices

Output: $B(v) \leftarrow L_t$ and exact distance $d^{(t)}(v, \cdot)$

Creating the ball centered at v (in parallel)

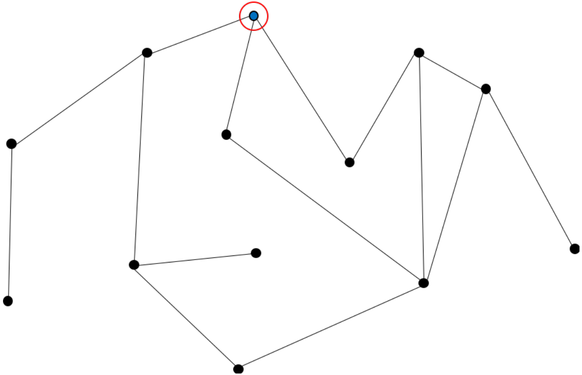


Figure: Expanding ball with $b = 5$. Reach of 0

Creating the ball centered at v (in parallel)

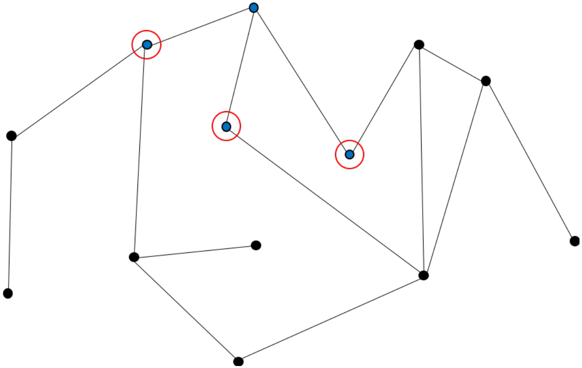


Figure: Expanding ball with $b = 5$. Reach of 1

Creating the ball centered at v (in parallel)

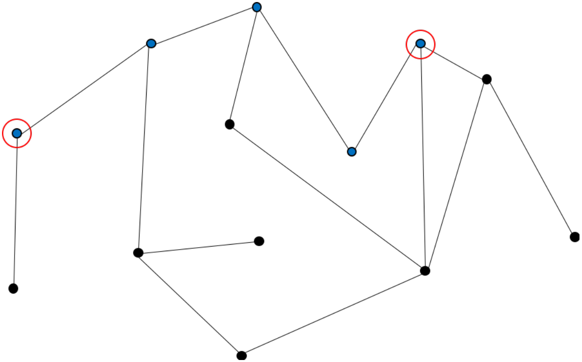


Figure: Expanding ball with $b = 5$. Reach of 2

Creating the ball centered at v (in parallel)

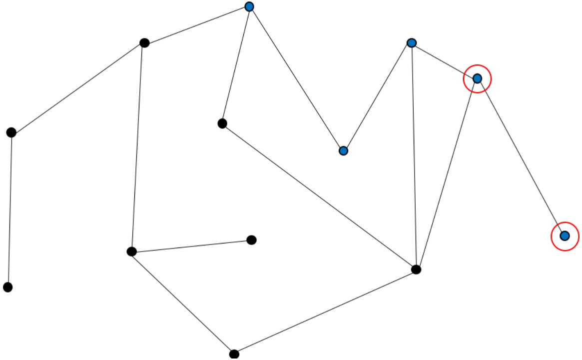


Figure: Expanding ball with $b = 5$. Reach of 4

Creating the ball centered at v (in parallel)

$L_0(v) \leftarrow$ closest b neighbors of v. Compute $\mathrm{dist}^{(0)}(v, u)$

- Use polylog depth and nearly work-efficient parallel sorting algorithm $\Rightarrow \mathcal{O}(\log^2 n)$ depth and $\mathcal{O}(m \log^2 n)$ work
- Initialize distance in $\mathcal{O}(1)$ depth and $\mathcal{O}(m)$ work

Creating the ball centered at v (in parallel)

For $i = 1, \ldots, t = \lceil \log n \rceil$:

1. $\forall v, u \in V$, set $\text{dist}^{(i)}(v, u) \leftarrow \infty$
2. Fix $v \in V$. Consider (v, x, u) such that $x \in L_{i-1}(v)$ and $u \in L_{i-1}(x)$. Update $\text{dist}^{(i)}(v, u)$ if ...
3. For every $v \in V$, set $L_i(v) \leftarrow$ closest b vertices

We perform ball computations on each vertex in G or a subemulator H_i. Repeat $\log n$ times:

1. $\leq b^2$ vertex pairs per vertex $v \in V$.
2. $\mathbb{E}[|V(H_i)|] = \mathcal{O}(n \cdot \text{poly}(\log n)) / b_i^2 \Rightarrow \mathcal{O}(n \cdot \text{poly}(\log n))$ work and $\mathcal{O}(1)$ depth per iteration
3. Use sorting algorithm

Selecting vertices (in parallel)

Fix a ball size b (to be defined later).

1. Construct S by sampling every vertex
2. If $v \in V$ is not near any vertex in S, add v to S
3. Store the leader $q(v) \leftarrow$ closest vertex $u \in S$ to $v \in V$

Selecting vertices (in parallel)

Fix a ball size b (to be defined later).

1. Construct S by sampling every vertex
2. If $v \in V$ is not near any vertex in S, add v to S
3. Store the leader $q(v) \leftarrow$ closest vertex $u \in S$ to $v \in V$

1. $\mathcal{O}(1)$ depth and $\mathcal{O}(n)$ work
2. Construct a ball. Check requires $\mathcal{O}(1)$ depth and $\mathcal{O}(n)$ work
3. Ditto

Adding edges (in parallel)

Adding edges:

1. For all $(u, v) \in E$, add edge $\big(q(u), q(v)\big)$
2. Fix $v \in V$. $\forall u \in B(v)$, add edge $\big((q(u), q(v)\big)$

When adding an edge $e = \big(q(u), q(v)\big)$ to F, update weight

$$w(e) = \min \begin{cases} w(e), (\text{initialize to } \infty) \\ d_G(q(u), u) + d_G(u, v) + d_G(v, q(v)) \end{cases}.$$

Adding edges (in parallel)

Adding edges:

1. For all $(u, v) \in E$, add edge $(q(u), q(v))$
2. Fix $v \in V$. $\forall u \in B(v)$, add edge $((q(u), q(v))$

When adding an edge $e = (q(u), q(v))$ to F, update weight

$$w(e) = \min \begin{cases} w(e), (\text{initialize to } \infty) \\ d_G(q(u), u) + d_G(u, v) + d_G(v, q(v)) \end{cases} .$$

1. Uses ball to compute exact distances. $\mathcal{O}(1)$ depth and $\mathcal{O}(m)$ work
2. Ditto
3. Ditto

Recursive subemulators (in parallel)

Set $b_0 \ll n$.

Let the first subemulator H_0 be the original graph G.

While $|V(H_i)| \geq b_i$:

1. $\forall v \in V(H_i)$, save $B_{H_i, b_i}(v)$
2. $H_{i+1} \leftarrow \text{SUBEMULATOR}(H_i, b_i)$
3. Update $b_{i+1} \leftarrow b_i^{1.25}$ and $i \leftarrow i + 1$

Recursive subemulators (in parallel)

Set $b_0 \ll n$.

Let the first subemulator H_0 be the original graph G.

While $|V(H_i)| \geq b_i$:

1. $\forall v \in V(H_i)$, save $B_{H_i, b_i}(v)$
2. $H_{i+1} \leftarrow \text{SUBEMULATOR}(H_i, b_i)$
3. Update $b_{i+1} \leftarrow b_i^{1.25}$ and $i \leftarrow i + 1$

Repeat $\mathcal{O}(\log \log n)$ times:

1. Ball construction
2. Proved is polylog depth and nearly linear
3. $\mathcal{O}(1)$

Collapsing towards a low–hop emulator (in parallel)

For every subemulator $H_0, \ldots, H_{\mathcal{O}(\log logn)}$:

1. Add the edge from a vertex up to its leaders in next level
2. Add the edge between close vertices that are within same subemulator

Let t be number of levels. Set

$$w_F(u, v) = 27^{t - \max(i,j)} \cdot d_{H_i}(u, v).$$

Collapsing towards a low–hop emulator (in parallel)

For every subemulator $H_0, \ldots, H_{\mathcal{O}(\log\, log n)}$:

1. Add the edge from a vertex up to its leaders in next level
2. Add the edge between close vertices that are within same subemulator

Let t be number of levels. Set

$$w_F(u, v) = 27^{t-\max(i,j)} \cdot d_{H_i}(u, v).$$

Repeat $\mathcal{O}(\log \log n)$ for each subemulator

1. $\mathcal{O}(1)$ depth and $\mathcal{O}\big(n \cdot \mathrm{poly}(\log n)\big)$ work
2. Ditto
3. Weights: Ditto

Constructing a low–hop emulator (in parallel)

Recap. Assuming we have a concurrent read, concurrent write computational model, we can solve the following problems in polylog depth and near linear work:

1. Create the ball centered at v
2. Selecting vertices
3. Adding edges
4. Recursive subemulators
5. Collapsing towards a low–hop emulator

Constructing a low–hop emulator can be done in same depth and work.

Applications

Compute the following problems with polylog approx, polylog depth, and nearly linear work:

1. SSSP via Bellman-Ford
2. Bourgain's embedding
3. Low diameter decomposition
4. Metric tree embedding

Uncapacitated min–cost flow (transshipment) problem

Let $W \in \mathbb{R}^{m \times m}$ be a diagonal matrix of weights. Let $A \in \mathbb{R}^{n \times m}$ be the incidence matrix,

$$A_{iu} = \begin{cases} 1 \ : \ \exists \text{ edge } u = (i,j) \\ -1 : \ \exists \text{ edge } u = (j,i) \\ 0 \ : \ \text{otherwise} \end{cases} .$$

Find a vector $f \in \mathbb{R}^m$ such that

$$\min_{f \in \mathbb{R}^m} \|Wf\|_1$$
$$\text{s.t. } Af = b,$$

where $b \in \mathbb{R}^n$ is the demand vector, where we require $\sum_i b_i = 0$.

If $b(s) = 1, \ b(t) = -1$, then solves (s,t)–shortest path length.

Uncapacitated min–cost flow problem

An equivalent problem:

Let $x = Wf$. Find the optimal x^* such that

$$x^* = \min_{x\mathbb{R}^m} \|x\|_1$$
$$\text{s.t. } AW^{-1}x = b.$$

Uncapacitated min–cost flow problem

An equivalent problem:

Let $x = Wf$. Find the optimal x^* such that

$$x^* = \min_{x\mathbb{R}^m} \|x\|_1$$
$$\text{s.t. } AW^{-1}x = b.$$

Lemma: There exists $(1+\varepsilon)$–approximation algorithm to the optimization problem above in that runs in polylog depth if there exists a matrix P such that

$$\|x^*\|_1 \leq \|Pb\|_1 \leq \mathcal{O}(\operatorname{poly}\log n) \cdot \|x^*\|_1 .$$

Earth Movers Distance Problem

The Earth Mover's Distance (EMD) problem is

$$\min_{\pi: V\times V \to \mathbb{R}_{\geq 0}} \sum_{(u,v)\in V\times V} \pi(u,v)\cdot \|\phi(u)-\phi(v)\|_1$$

$$\text{s.t. } \forall u \in V, \sum_{v\in V}\pi(u,v) - \sum_{v\in V}\pi(v,u) = b_u.$$

Find a vector $f \in \mathbb{R}^m$ such that

$$\min_{f\in\mathbb{R}^m} \|Wf\|_1$$

$$\text{s.t. } Af = b,$$

Bourgain's Embedding

Theorem (Bourgain's Embedding)

Every metric space (V, d_V) can be embedded in ℓ_p with distortion $\mathcal{O}(\log n)$.

Given a graph $G = (V, E)$ and distance $d : V \times V \to \mathbb{R}^+$, there exists a mapping $\phi : V \to [\Delta]^{\mathcal{O}(\log^2 n)}$ such that

$$d(u, v) \leq \|\phi(u) - \phi(v)\|_1 \leq \mathcal{O}(\log n) d(u, v),$$

where $\Delta \leq \sum_{e \in E} w_e$.

Using Bourgain's embedding via low–hop emulators, we can find a mapping $\phi : V \to [\Delta]^\eta$ such that

$$\mathrm{OPT}_{EMD}(b) \leq \mathcal{O}(\mathrm{poly}(\log n))\mathrm{OPT}_{transshipment}(b).$$

Creating a preconditioner using grids

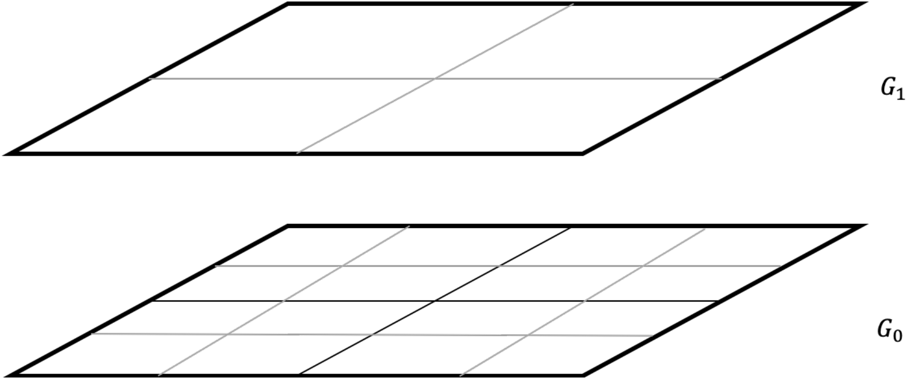


Figure: Grids

Creating a preconditioner

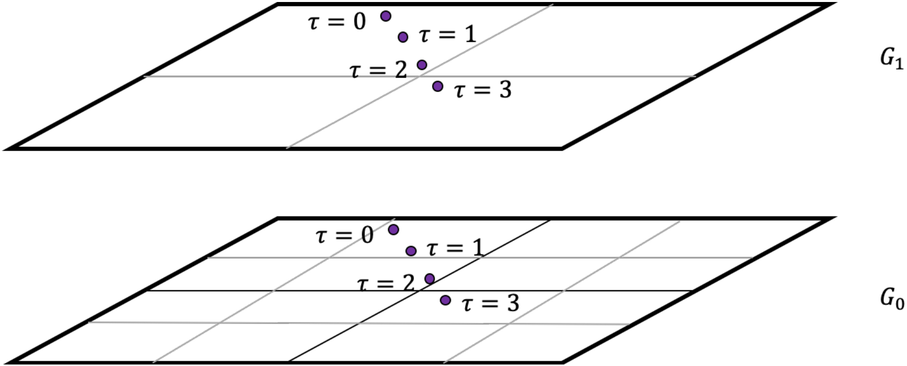


Figure: Grids with points

Creating a preconditioner

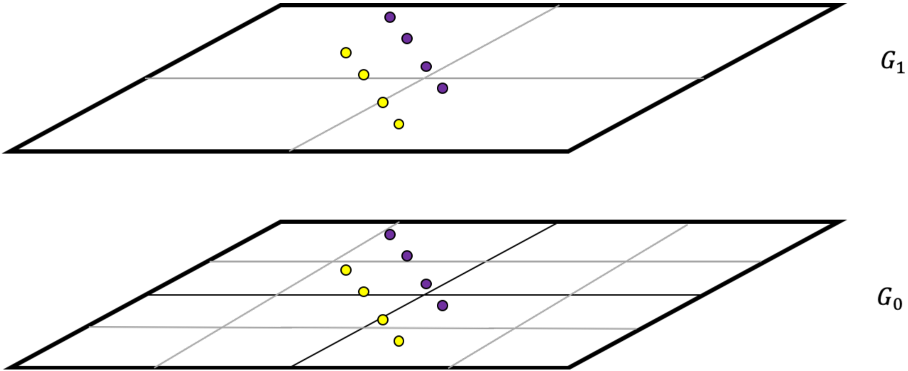


Figure: Points in same cell vs. random shift τ

Creating a preconditioner

Construct a series of $L = 1 + \log \Delta$ grids G_i as above.

Let $\tau \in [\Delta]$ be a uniform random variable.

Lemma: Define the vector

$$h_{(i,C)} = \eta \cdot 2^i \sum_{v \in V : \phi(v) + \tau \cdot \mathbf{1}_\eta \in C} b_v,$$

where i is the level of the grid (ie. G_i) and $C \in G_i$ is a cell. Then,

1. $\mathbb{E}_\tau[\|h\|_1] \leq 2L\eta \cdot \mathrm{OPT}_{EMD}(b)$
2. $\|h\|_1 \geq \mathrm{OPT}_{EMD}(b)$,

where $\eta = \mathcal{O}(\log^2 n)$.

Creating a preconditioner

Construct a vector h' such that $\mathbb{E}\left[\,\|h\|_1\,\right] = \|h'\|_1$,

$$h'_{(i,C,\tau)} = \eta \cdot \sum_{v \in V : \phi(v) + \tau \cdot \mathbf{1}_\eta \in C} b_v .$$

Creating a preconditioner

Construct a vector h' such that $\mathbb{E}\left[\,\|h\|_1\,\right] = \|h'\|_1$,

$$h'_{(i,C,\tau)} = \eta \cdot \sum_{v \in V : \phi(v) + \tau \cdot \mathbf{1}_\eta \in C} b_v .$$

Prescribes a matrix P' where

$$P'_{(i,C,\tau),v} = \begin{cases} \eta \ : \ \phi(v) + \tau \cdot 1_\eta \in C \\ 0 \ : \ \text{otherwise} \end{cases} .$$

Then $h' = P'b$, and

$$\|x^*\|_1 \leq \left\|h'\right\|_1 = \left\|P'b\right\|_1 \leq (\text{poly}(\log n)) \, \|x^*\|_1$$

Creating a preconditioner

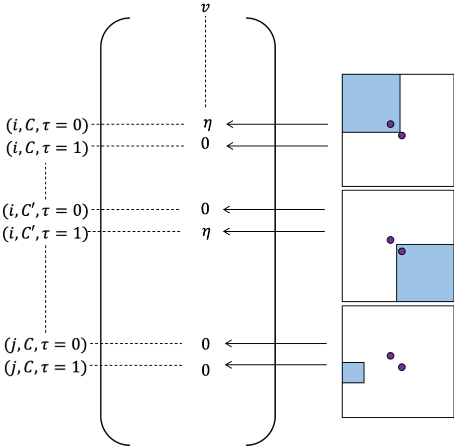


Figure: Preconditioner vs. Grids

Uncapacitated min–cost flow problem

We want to solve the following problem:

Let $x = Wf$. Find the optimal x^* such that

$$x^* = \min_{x\mathbb{R}^m} \|x\|_1$$
$$\text{s.t. } AW^{-1}x = b.$$

Uncapacitated min–cost flow problem

We want to solve the following problem:

Let $x = Wf$. Find the optimal x^* such that

$$x^* = \min_{x\mathbb{R}^m} \|x\|_1$$
$$\text{s.t. } AW^{-1}x = b.$$

Lemma: There exists $(1+\varepsilon)$–approximation algorithm to the optimization problem above in that runs in polylog depth using a matrix P' where

$$\|x^*\|_1 \leq \|P'b\|_1 \leq \mathcal{O}(\text{poly}\log n) \cdot \|x^*\|_1 .$$

Summary

1. Constructing a Low Hop Emulator
   - Constructing a subemulator
   - Recursive subemulators
   - Collapsing into a low–hop emulator
   - Constructing a low–hop emulator (in parallel)

2. $(1+\varepsilon)$ shortest paths in polylog depth and nearly linear work
   - Solving shortest paths via optimization and Sherman's framework

Thanks. Questions?