Low Hop Emulators and Uncapacitated Min-Cost Flow[1]

Caleb Ju and Tim Baer

Parallel Algorithms Reading Group
University of Illinois at Urbana-Champaign

October 15, 2020

[1] Based on results from *Parallel Approximate Undirected Shortest Paths Via Low Hop Emulators* by Andoni, Stein, and Zhong (published in STOC 2020).

Overview

1. Low Hop Emulators and Applications
2. Constructing a Low Hop Emulator
3. Uncapacitated Min-Cost Flow in Sherman's Framework
4. Constructing a Good Preconditioner

Recent Results

Theorem (Andoni et. al SSSP)

polylog(n)-*approximate single source shortest path with* polylog(n) *depth and* $m \cdot$ polylog(n) *work via low hop emulators.*

Theorem (Andoni et. al (s, t)-SP)

$(1 + \epsilon)$-*approximate* $(s - t)$-*shortest path with* polylog(n) *depth and* $m \cdot$ polylog(n) *work via reduction to uncapacitated min-cost flow.*

[2] *Faster Parallel Algorithm for Approximate Shortest Path* by Li (published in STOC 2020).

Recent Results

Theorem (Andoni et. al SSSP)

polylog(n)*-approximate single source shortest path with* polylog(n) *depth and* $m \cdot$ polylog(n) *work via low hop emulators.*

Theorem (Andoni et. al (s, t)-SP)

$(1+\epsilon)$*-approximate* $(s-t)$*-shortest path with* polylog(n) *depth and* $m \cdot$ polylog(n) *work via reduction to uncapacitated min-cost flow.*

Theorem (Li SSSP[2])

$(1+\epsilon)$*-approximate single source shortest path with* polylog(n) *depth and* $m \cdot$ polylog(n) *work via reduction to uncapacitated min-cost flow.*

---

[2] *Faster Parallel Algorithm for Approximate Shortest Path* by Li (published in STOC 2020).

What is a Low Hop Emulator?

What is a low hop emulator?

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

[3] Uses hierarchy of discretizations to solve a linear system.

[4] $d(u, v) \leq \tilde{d}(u, v) \leq f(n)d(u, v)$, where $f(n)$ is the approximation factor, and d and $\tilde{d}$ are the exact and approximate distances, respectively.

What is a Low Hop Emulator?

What is a low hop emulator?

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

- Is multigrid inspired[3]
- Provable approximation/distortion factors[4]
- Can be constructed in parallel

[3] Uses hierarchy of discretizations to solve a linear system.

[4] $d(u,v) \leq \tilde{d}(u,v) \leq f(n)d(u,v)$, where $f(n)$ is the approximation factor, and d and $\tilde{d}$ are the exact and approximate distances, respectively.

Applications

1. Uncapacitated min–cost flow: find cheapest way to *route* flow from supply vertices to demand vertices
2. Bourgain's embedding: embed *any* metric space into ℓ_p with distortion $\mathcal{O}(\log n)$
3. Low diameter decomposition: decompose a graph into subsets such that *far* vertices are *unlikely* to belong to the same subset

Constructing the Low Hop Emulator

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

How can we build a low hop emulator?

Constructing the Low Hop Emulator

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

How can we build a low hop emulator?

1. Build a sparser graph using a subemulator
2. Recursively apply subemulator $\mathcal{O}\big(\log \log n\big)$ times

Constructing the Low Hop Emulator

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

How can we build a low hop emulator?

1. Build a sparser graph using a subemulator
2. Recursively apply subemulator $\mathcal{O}\big(\log \log n\big)$ times

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

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. Initially construct S by sampling vertices
2. Add more vertices to ensure every vertex is close to a vertex in S
3. Add weighted edges so that local distances are well–approximated

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. Initially construct S by sampling vertices
2. Add more vertices to ensure every vertex is close to a vertex in S
3. Add weighted edges so that local distances are well–approximated

The ball, $B_{G,b}(v)$: closest b vertices (w.r.t graph distance) to v.

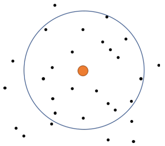


Figure: Ball

Selecting Vertices

1. Initially construct S by sampling vertices
2. Add vertices to S so that every $v \in V$ is close to a vertex in S
3. Add weighted edges so that local distances are well–approximated

Given ball size b (typically $b = \mathcal{O}(\log n)$).

1. For every vertex $v \in V$, construct its ball $B(v)$
2. Construct S by sampling every vertex with probability $p = \min(50\frac{\log n}{b}, \frac{1}{2})$
3. For any $v \in V \setminus S$ whose ball does not contain any vertex in S, then add v to S
4. Store the leader$(v) \leftarrow$ closest vertex $u \in S$ to $v \in V$

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

Selecting vertices

Can we only compute the balls for the sampled vertices?

For any $v \in V \setminus S$ whose ball does not contain any vertex in S, then add v to S

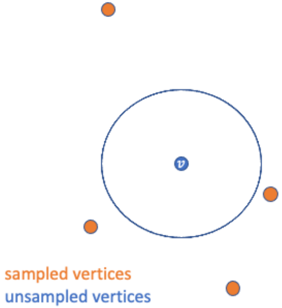


Figure: Line 4 of selecting vertices

For any $v \in V \setminus S$ that is not contained in the ball of any vertex in S, then add v to S

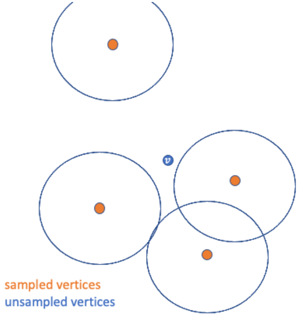


Figure: Only compute balls for sampled vertices

Adding Edges

1. Initially construct S by sampling vertices
2. Add more vertices to ensure vertex in V is close to a vertex in S
3. Add weighted edges so that local distances are well–approximated

Adding Edges

1. Initially construct S by sampling vertices
2. Add more vertices to ensure vertex in V is close to a vertex in S
3. Add weighted edges so that local distances are well–approximated

Denote $q(v) = \text{leader}(v) \in S$

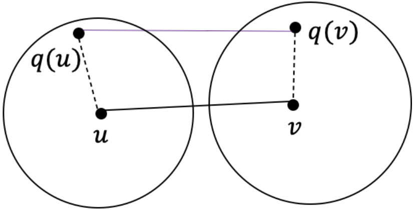


Figure: For every edge from G, add edge between the leaders

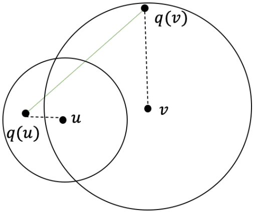


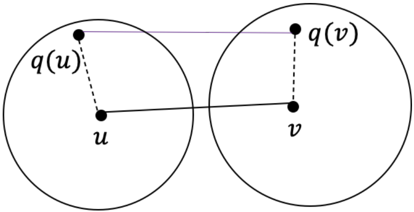
Figure: For every pair of vertices that are "close", add an edge between their leaders

Adding Edges

Denote $q(v) = \text{leader}(v) \in S$.

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

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



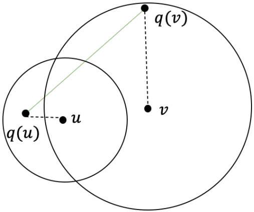


Figure:

Properties of Subemulators

Let $H = (S, F')$ be a subemulator of G.

- $\mathbb{E}\left[|S|\right] \leq \frac{3}{4}n$ and $|F'| \leq m + nb$
- For any $u, v \in S$, their distance in H is distorted by a constant factor[5]
- For any $u, v \in V$, the distance between their leaders in H is distorted by a constant factor[6]

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

[5] For any $u, v \in S$, $d_G(u, v) \leq d_H(u, v) \leq 8 \cdot d_G(u, v)$.

[6] For any $u, v \in V$, $d_H\big(q(u), q(v)\big) \leq d_G\big(u, q(u)\big) + 22 \cdot d_G(u, v) + d_G\big(v, q(v)\big)$.

Constructing the Low Hop Emulator

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

How can we build a low hop emulator?

1. Define a restriction operator via subemulator
2. Recursively apply subemulator $\mathcal{O}\big(\log \log n\big)$ times

Combining Nested Subemulators as H (Distance Oracle)

1. Form $t = \mathcal{O}(\log \log n)$ recursive subemulators
2. Add an edge from a vertex to its leaders in the next level
3. Keep edges within each subemulator
4. Add edges between "close" vertices in the same level

Scale: $w_H(u, v) = 27^{t - \max(\mathrm{lvl}(u), \mathrm{lvl}(v))} \cdot d_{H_i}(u, v)$.

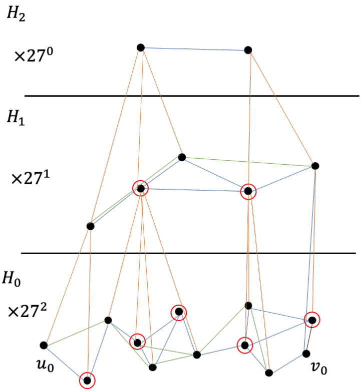


Figure: Scaling edges

Properties of a Distance Oracle

Lemma

Let H be distance oracle.

- $|E(H)| = \mathcal{O}\big(m \cdot \mathrm{poly}(\log n)\big)$
- *For every $u, v \in V$ and corresponding $u^{(0)}, v^{(0)} \in H_0$ (in the bottom level of H), then $d_H(u^{(0)}, v^{(0)})$ is distorted by at most* $\mathrm{polylog}(n)$[7]

[7] Precisely, $d_G(u, v) \leq d_H(u^{(0)}, v^{(0)}) \leq 26^{\mathcal{O}(\log\log n)} \cdot d_G(u, v)$

Properties of a Distance Oracle

Lemma

Let H be distance oracle.

- $|E(H)| = \mathcal{O}\big(m \cdot \mathrm{poly}(\log n)\big)$
- *For every $u, v \in V$ and corresponding $u^{(0)}, v^{(0)} \in H_0$ (in the bottom level of H), then $d_H(u^{(0)}, v^{(0)})$ is distorted by at most* $\mathrm{polylog}(n)$[7]

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

[7] Precisely, $d_G(u, v) \le d_H(u^{(0)}, v^{(0)}) \le 26^{\mathcal{O}(\log \log n)} \cdot d_G(u, v)$

Creating the Ball Centered at v (in parallel)

$B_{G,b}(v) = \{\text{closest } b \text{ vertices to } v\}$.
Goal: Build all n balls in $\log n$ depth and nearly linear work.

Creating the Ball Centered at v (in parallel)

$B_{G,b}(v) = \{\text{closest } b \text{ vertices to } v\}$.
Goal: Build all n balls in $\log n$ depth and nearly linear work.

Idea 1: Compute all pairs shortest path (APSP) using $\log(n)$ *path doublings*. Then save b closest vertices.

Cost: $\log(n)$ iterations with $\mathcal{O}\left(n^3 \cdot \log(n)\right)$ work.

Creating the Ball Centered at v (in parallel)

$B_{G,b}(v) = \{\text{closest } b \text{ vertices to } v\}$.
Goal: Build all n balls in $\log n$ depth and nearly linear work.

Idea 1: Compute all pairs shortest path (APSP) using $\log(n)$ *path doublings*. Then save b closest vertices.

Cost: $\log(n)$ iterations with $\mathcal{O}\left(n^3 \cdot \log(n)\right)$ work.

Idea 2: Compute *partial* APSP using $\log(b)$ *truncated* path doublings. After each path doubling, keep b closest vertices.

Cost: $\log(b)$ iterations with $\mathcal{O}\left(n \cdot b^2 \cdot \log(n)\right)$ work. Set $b = \mathcal{O}(\log n)$.

Creating the Ball Centered at v (in parallel)

Let $\boldsymbol{B}^{(i)} \in \mathbb{R}^{n\times n}$ on the $(\min, +)$ semiring contain the tenative distances to the b closest vertices to vertex $u, \forall u \in V$.

$$\boldsymbol{B}^{(i+1)} = \mathsf{F}[\boldsymbol{B}^{(i)}\boldsymbol{B}^{(i)}]$$

where F extracts the b closest neighbors to $u, \forall u \in V$.

Creating the Ball Centered at v (in parallel)

Let $\boldsymbol{B}^{(i)} \in \mathbb{R}^{n \times n}$ on the $(\min, +)$ semiring contain the tenative distances to the b closest vertices to vertex $u, \forall u \in V$.

$$\boldsymbol{B}^{(i+1)} = \mathsf{F}[\boldsymbol{B}^{(i)}\boldsymbol{B}^{(i)}]$$

where F extracts the b closest neighbors to $u, \forall u \in V$.

1. Requires $\mathcal{O}(\log b)$ iterations
2. Parallelizable using existing matrix-matrix product techniques
3. Can be sparsified by specifying a row filter

Creating the Ball Centered at v (in parallel)

Alternatively, let $B_u^{(i)}$ be a sorted b-vector containing tentative distances to the b closest vertices to vertex u.

$$B_u^{(i+1)} = B_u^{(i)} \oplus \Big(\bigoplus_{(u,v)\in E} \{w(u,v) + B_v^{(i)}\} \Big)$$

where the reduction operator $x \oplus y$ merges x and y and returns the first b distances.

Creating the Ball Centered at v (in parallel)

Alternatively, let $B_u^{(i)}$ be a sorted b-vector containing tentative distances to the b closest vertices to vertex u.

$$B_u^{(i+1)} = B_u^{(i)} \oplus \Big(\bigoplus_{(u,v)\in E} \{w(u,v) + B_v^{(i)}\}\Big)$$

where the reduction operator $x \oplus y$ merges x and y and returns the first b distances.

1. Requires $\mathcal{O}(b)$ iterations
2. Can be embedded into a generalized matrix-vector product with the adjacency matrix and parallelized with existing techniques
3. Can be sparsified by specifying a row filter

Tuning Parameters

1. approximation constant ϵ
2. size of the ball b
3. number of subemulators
4. sampling probability of vertices
5. scaling factor for combining nested subemulators

Uncapacitated Min–Cost Flow (Transshipment)

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

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

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

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

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 (Transshipment)

An equivalent problem:

Let $x = \boldsymbol{W} f$. Find the optimal x^* such that

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

Uncapacitated Min–Cost Flow (Transshipment)

An equivalent problem:

Let $x = \boldsymbol{W} f$. Find the optimal x^* such that

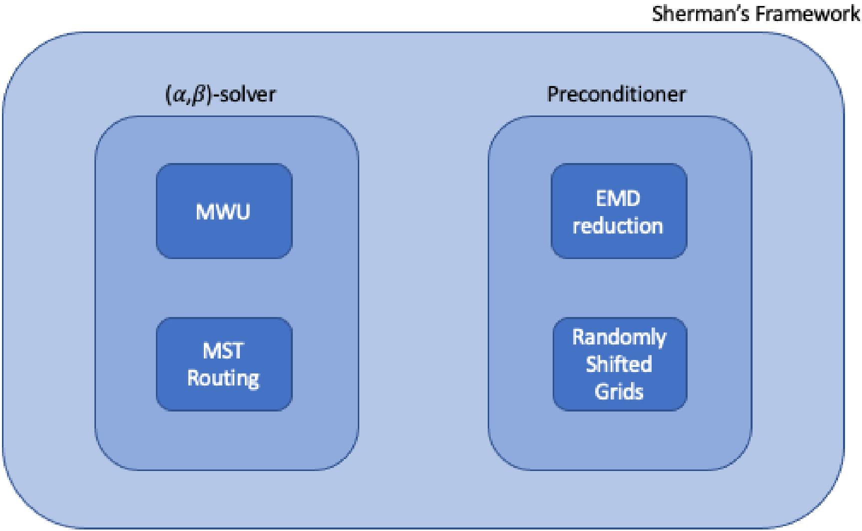
$$x^* = \min_{x \in \mathbb{R}^m} \|x\|_1$$
$$\text{s.t. } \boldsymbol{A}\boldsymbol{W}^{-1} x = b.$$

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

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

Sherman's Framework

We use Sherman's framework to solve uncapacitated min-cost flow.



Sherman's Framework[8]

Let $\mathcal{X}, \mathcal{Y}$ be finite dimensional vector spaces, where $\mathcal{X}$ is also a Banach space, and let $\boldsymbol{A} \in Lin(\mathcal{X}, \mathcal{Y})$ be fixed. Consider the problem:

$$\min_{x \in \mathcal{X}} \|x\|_{\mathcal{X}}$$
$$\text{s.t. } \boldsymbol{A}x = b,$$

[8] Based on results from *Generalized Preconditioning and Network Flow Problems* by Sherman (published in SODA 2017).

Sherman's Framework[8]

Let $\mathcal{X}, \mathcal{Y}$ be finite dimensional vector spaces, where $\mathcal{X}$ is also a Banach space, and let $\boldsymbol{A} \in Lin(\mathcal{X}, \mathcal{Y})$ be fixed. Consider the problem:

$$\min_{x \in \mathcal{X}} \|x\|_{\mathcal{X}}$$
$$\text{s.t. } \boldsymbol{A}x = b,$$

x is an (α, β)-solution to the above problem if

1. $\frac{\|x\|}{\|x_{opt}\|} \leq \alpha$, and
2. $\frac{\|\boldsymbol{A}x - b\|}{\|\boldsymbol{A}\|\|x_{opt}\|} \leq \beta$

[8]Based on results from *Generalized Preconditioning and Network Flow Problems* by Sherman (published in SODA 2017).

Sherman's Framework

x is an (α, β)-solution to the above problem if

1. $\frac{\|x\|}{\|x_{opt}\|} \leq \alpha$, and
2. $\frac{\|\boldsymbol{A}x - b\|}{\|\boldsymbol{A}\|\|x_{opt}\|} \leq \beta$

The non-linear condition number of $\boldsymbol{A} : \mathcal{X} \to \mathcal{Y}$ is

$$\kappa_{\mathcal{X}\to\mathcal{Y}}(\boldsymbol{A}) = \min\{\frac{\|A\|_{\mathcal{X}\to\mathcal{Y}} \|x\|_{\mathcal{X}}}{\|\boldsymbol{A}x\|_{\mathcal{Y}}} : \boldsymbol{A}x \neq 0\}$$

Sherman's Framework

x is an (α, β)-solution to the above problem if

1. $\frac{\|x\|}{\|x_{opt}\|} \leq \alpha$, and
2. $\frac{\|\boldsymbol{A}x - b\|}{\|\boldsymbol{A}\|\|x_{opt}\|} \leq \beta$

The non-linear condition number of $\boldsymbol{A} : \mathcal{X} \to \mathcal{Y}$ is

$$\kappa_{\mathcal{X}\to\mathcal{Y}}(\boldsymbol{A}) = \min\{\frac{\|A\|_{\mathcal{X}\to\mathcal{Y}} \|x\|_{\mathcal{X}}}{\|\boldsymbol{A}x\|_{\mathcal{Y}}} : \boldsymbol{A}x \neq 0\}$$

Theorem (Composition of solvers)

Let F_i be a $(\alpha_i, \beta_i/\kappa)$-solver for $\boldsymbol{A} : \mathcal{X} \to \mathcal{Y}$, where $\boldsymbol{A}$ has non-linear condition number κ. Then, the composition $F_2 \circ F_1$ is an $(\alpha_1 + \alpha_2\beta_1, \beta_1\beta_2/\kappa)$-solver for the same problem.

Multiplicative Weights Update

We can reduce the problem

$$\min_{x \in \mathcal{X}} \|x\|_{\mathcal{X}}$$
$$\text{s.t. } \boldsymbol{A}x = b,$$

to a feasibility problem, which we can approximately solve with the multiplicative weights update method.

Multiplicative Weights Update

We can reduce the problem

$$\min_{x \in \mathcal{X}} \|x\|_{\mathcal{X}}$$
$$\text{s.t. } \boldsymbol{A}x = b,$$

to a feasibility problem, which we can approximately solve with the multiplicative weights update method.

> Consider an arbitrary decision and a panel of n experts. Over a series of rounds, the multiplicative weights update method rewards experts whose predictions are good and punishes experts whose predictions are poor.

As an iterative method, we can use the composition of solvers theorem.

Earth Mover's Distance Problem

Theorem (Bourgain's Embedding)

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

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

Earth Mover's Distance Problem

Theorem (Bourgain's Embedding)

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

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

We can reduce approximating the uncapacitated min-cost flow on G to approximating the cost of the Earth Mover's Distance problem.

The Earth Mover's Distance (EMD) problem is

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

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

Randomly Shifted Grids

We use randomly shifted grids to obtain a β-approximation to OPT_{EMD}.

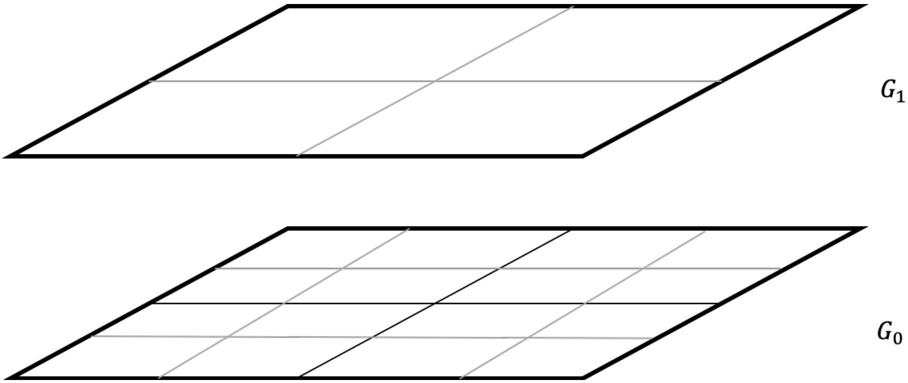


Figure: Grids

Randomly Shifted Grids

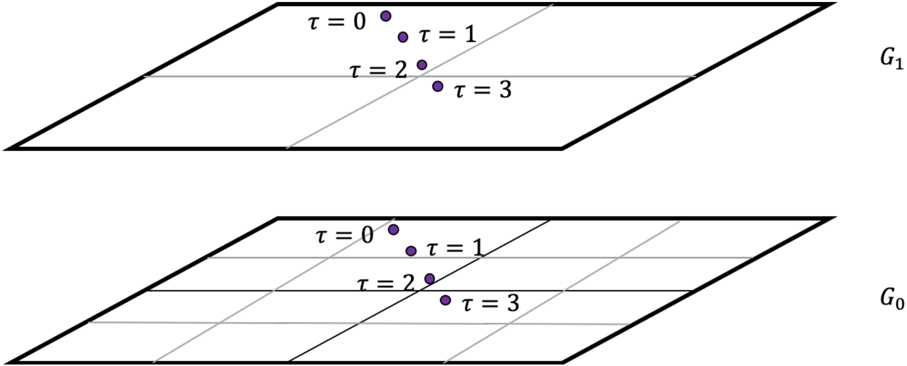


Figure: Grids with points

Preconditioner

Construct a sequence of $L = 1 + \log \Delta$ grids G_i and let τ be a random variable over $[\Delta]$.

For each level $i \in \{0, 1, ..., L-1\}$, each cell $C \in G_i$, and each shift value $\tau \in [2^i]$, we set $h' \in \mathbb{R}^{\sum_{i=0}^{L-1} 2^i |G_i|}$ to

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

Then,

1. $OPT_{EMD}(b) \leq \|h'\|_1 \leq 2Ld \cdot OPT_{EMD}(b)$
2. $\kappa(P'AW^{-1}) \leq 2Ld\alpha$

Preconditioner

Construct a sequence of $L = 1 + \log \Delta$ grids G_i and let τ be a random variable over $[\Delta]$.

For each level $i \in \{0, 1, \ldots, L-1\}$, each cell $C \in G_i$, and each shift value $\tau \in [2^i]$, we set $h' \in \mathbb{R}^{\sum_{i=0}^{L-1} 2^i |G_i|}$ to

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

Then,

1. $OPT_{EMD}(b) \leq \|h'\|_1 \leq 2Ld \cdot OPT_{EMD}(b)$
2. $\kappa(P'AW^{-1}) \leq 2Ld\alpha$

Observe that h' can be written as a linear map $h' = Pb$ where

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

Preconditioner

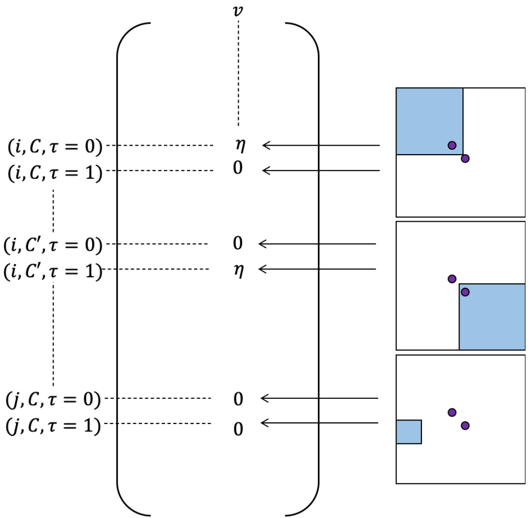
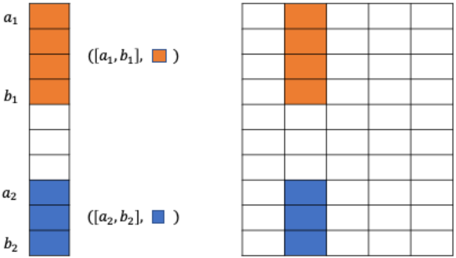


Figure: Preconditioner

Compressed Representation

Given a vector $x \in \mathbb{R}^r$, a compressed representation of x is a set of tuples $\mathrm{I} = \{([a_1, b_1], c_1), ([a_2, b_2], c_2), \ldots, ([a_s, b_s], c_s)\}$ where $c_i \in \mathbb{R}, [a_i, b_i] \subseteq [1, r]$ such that

1. $\forall i \neq j \in [s], [a_i, b_i] \cap [a_j, c_j] = \varnothing$
2. $\forall j \in [a_i, b_i], x_j = c_i$
3. $\forall j \in [1, r] \setminus \bigcup_{i \in [s]} [a_i, b_i], x_j = 0$



Implicit Preconditioner

How can we construct our implicit preconditioner?

$$P'(i, C, \tau), v = \begin{cases} d & \phi(v) + \tau \cdot 1_d \in C \\ 0 & \text{otherwise} \end{cases}$$

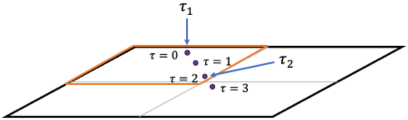


Figure: Grid with shifted points

Fix a vertex $v \in V$, a level $l \in L$, and a cell C_k. If $\exists \tau \in [2^l]$ such that $\phi(v) + \tau \cdot 1_d \in C_k$,

1. $\tau_1 = \underset{\tau \in [2^l] : \phi(v) + \tau \cdot 1_d \in C_k}{min} \tau$
2. $\tau_2 = \underset{\tau \in [2^l] : \phi(v) + \tau \cdot 1_d \in C_k}{max} \tau$
3. $a \leftarrow (k-1)2^l + \sum_{j=0}^{l-1} 2^j |C_j|$
4. $\mathrm{I}_v \leftarrow \mathrm{I}_v \cup \{([a + \tau_1, a + \tau_2], d)\}$

Implicit Preconditioner

Let $x = \boldsymbol{W} f$. Find the optimal x^* such that

$$x^* = \min_{x \in \mathbb{R}^m} \|x\|_1$$

$$\text{s.t. } \boldsymbol{AW}^{-1} x = b.$$

Implicit Preconditioner

Let $x = \boldsymbol{W}f$. Find the optimal x^* such that

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

Theorem

Given an undirected graph $G = (V, E, w)$ and a mapping $\phi(v) : V \to [\Delta]^d$ such that

$$\forall u, v \in V, d_G(u, v) \leq \|\phi(u) - \phi(v)\|_1 \leq \alpha \cdot d_G(u, v),$$

we can efficiently compute a compressed representation $\mathrm{I} = (\mathrm{I}_1, \mathrm{I}_2, \ldots, \mathrm{I}_n)$ of a matrix P with full column rank and

1. $\kappa(PAW^{-1}) \leq \mathcal{O}(\alpha L d)$
2. *each I_i of size at most $(d+1)L$*

Fast Operations

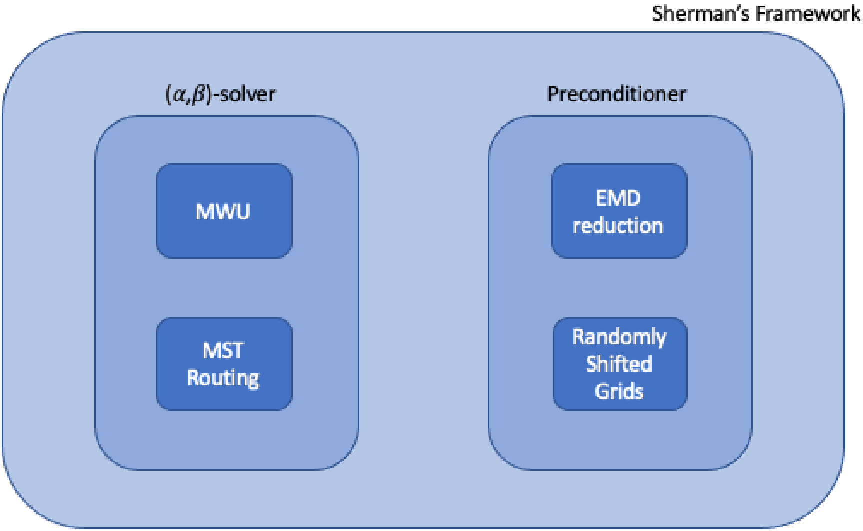
How can we do matrix-vector products with our compressed representation?

$MatVec(\mathrm{I} = (\mathrm{I}_1, \mathrm{I}_2, \ldots, \mathrm{I}_n), g \in \mathbb{R}^n)$

1. $S \leftarrow \varnothing, \widehat{\mathrm{I}} \leftarrow \varnothing$
2. For $i \in [n] : g_i \neq 0$
   1. for each $([a, b], c) \in \mathrm{I}_i$, $S \leftarrow S \cup \{(a, cg_i), (b+1, -cg_i)\}$
3. Sort $S = \{(q_1, z_1), (q_2, z_2), \ldots, (q_k, z_k)\}$ such that $q_1 \leq q_2 \leq \ldots \leq q_k$
4. For each $j \in \{2, 3, \ldots, k\} : q_j > q_{j-1}, \widehat{\mathrm{I}} \leftarrow \widehat{\mathrm{I}} \cup \{([q_{j-1}, q_j - 1], \sum_{t:q_t<q_j} z_t)\}$

Sherman's Framework

We use Sherman's framework to solve uncapacitated min-cost flow.



Summary

1. Low Hop Emulators and Applications
2. Constructing a Low Hop Emulator
3. Uncapacitated Min-Cost Flow in Sherman's Framework
4. Constructing a Good Preconditioner

Fin

Thanks. Questions?