Fast Dynamic Distance Computation via Dynamic Spanners

Sebastian Forster

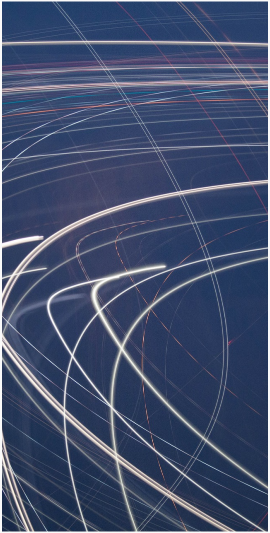
Habilitation Colloquium

University of Salzburg

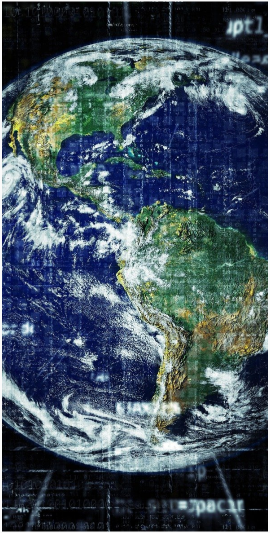
Big Data

The three V's

Volume

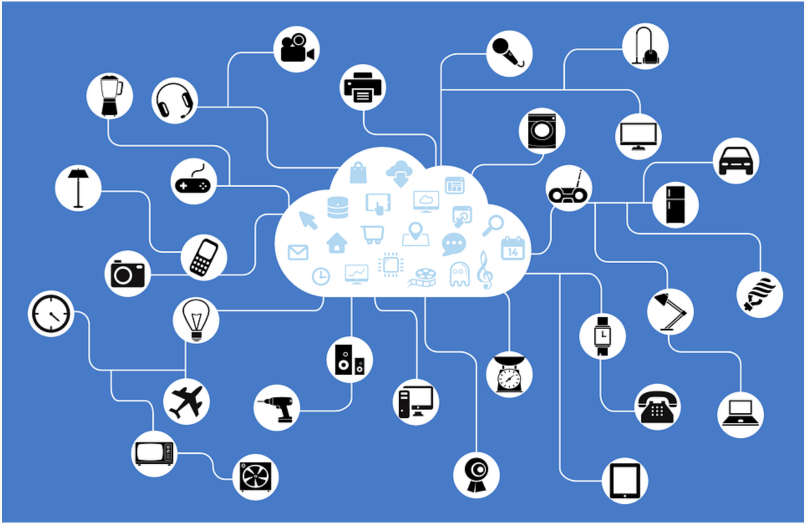


Velocity



Variety

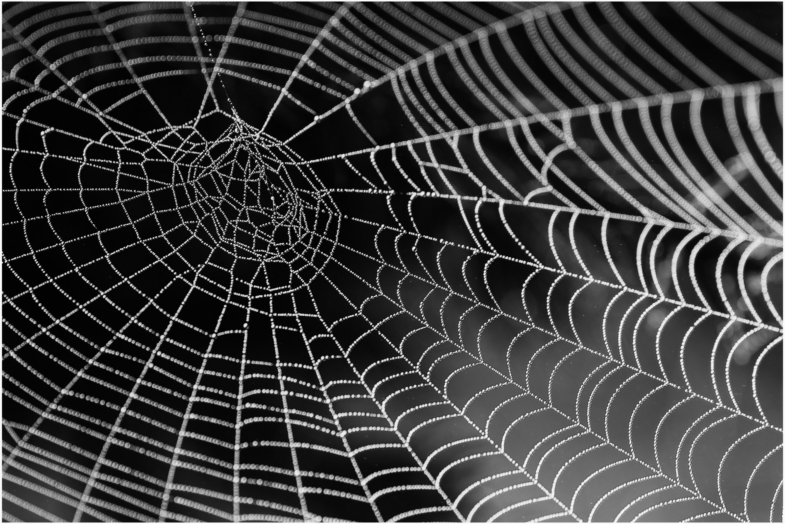
Graphs are Everywhere



Graphs are Everywhere

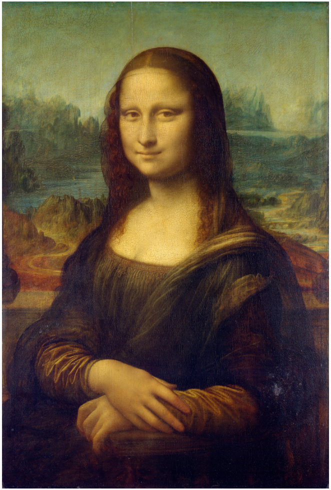
Graphs are Everywhere

Graphs are Everywhere



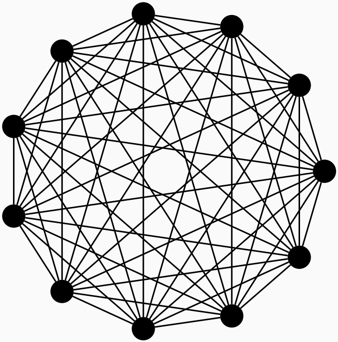
Space Reduction

"Sketching"

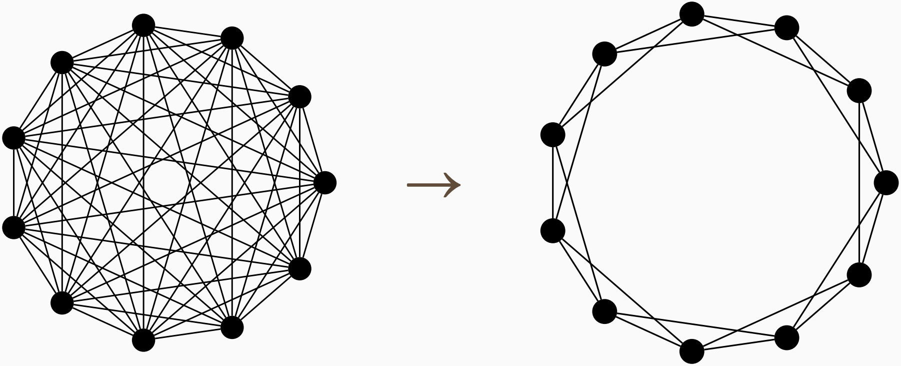


$\approx$

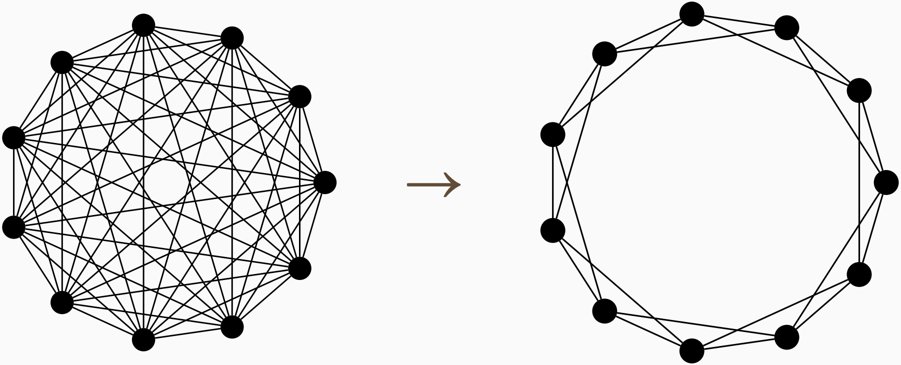
Graph Sparsification



Graph Sparsification

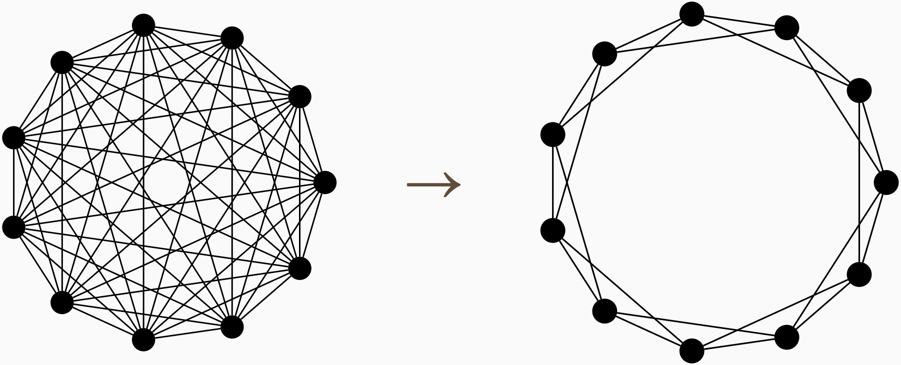


Graph Sparsification



Goal: Reduce number of edges

Graph Sparsification

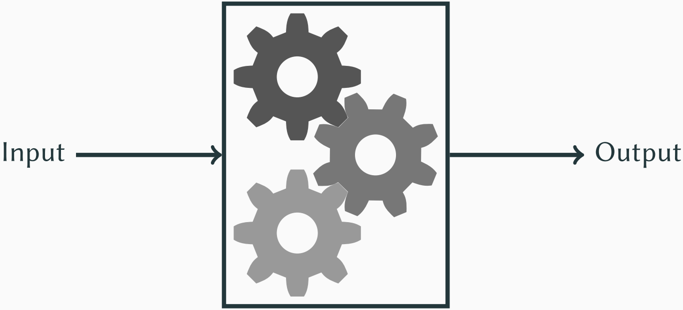


Goal: Reduce number of edges

...at cost of approximation

Dynamic Algorithms

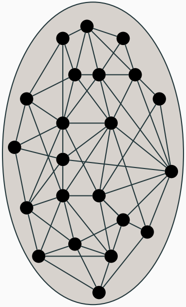
Static Approach



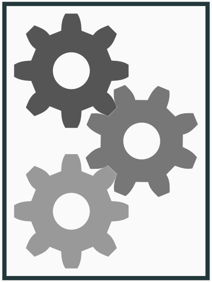
Dynamic Environments

Dynamic Sparsification

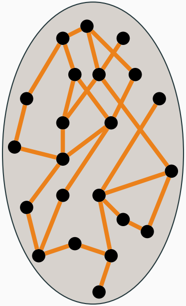
Input graph G



Algorithm

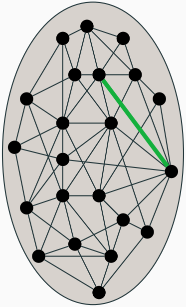


Sparsifier H

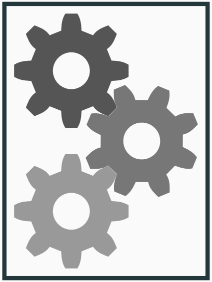


Problem Setting

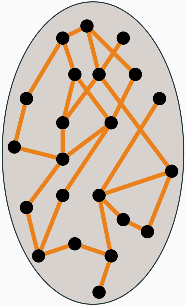
Input graph G



Algorithm



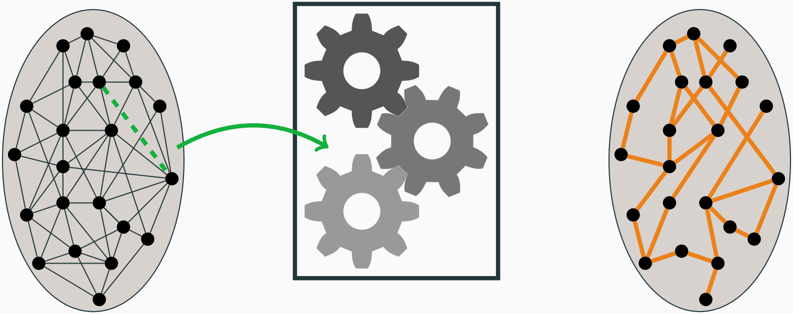
Sparsifier H



Adversary inserts
and deletes edges

Problem Setting

Input graph G | Algorithm | Sparsifier H



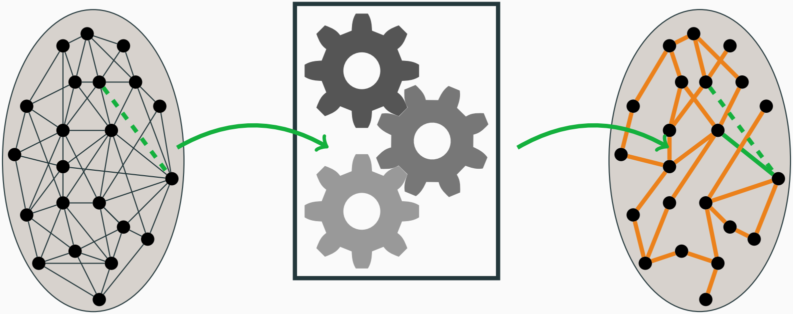
Adversary inserts and deletes edges

Problem Setting

Input graph G

Algorithm

Sparsifier H



Adversary inserts and deletes edges

Algorithm adds and removes edges

Example 1: Distance-Preserving Sparsification

Definition ([Peleg, Schäffer '89])

A **spanner** of **stretch** t of $G = (V, E)$ is a subgraph $H = (V, E')$ such that

$$dist_G(u, v) \leq dist_H(u, v) \leq t \cdot dist_G(u, v)$$

for all pairs of nodes $u, v \in V$.

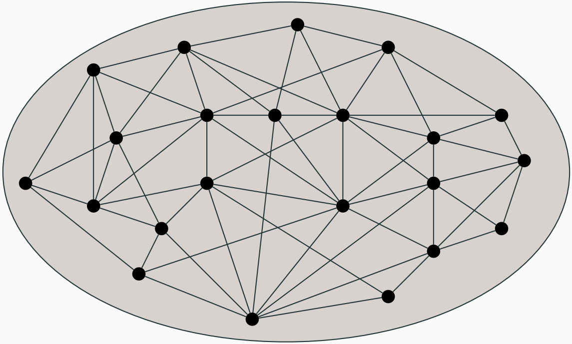
Example 1: Distance-Preserving Sparsification

Definition ([Peleg, Schäffer '89])

A **spanner** of **stretch** t of $G = (V, E)$ is a subgraph $H = (V, E')$ such that

$$dist_G(u, v) \leq dist_H(u, v) \leq t \cdot dist_G(u, v)$$

for all pairs of nodes $u, v \in V$.



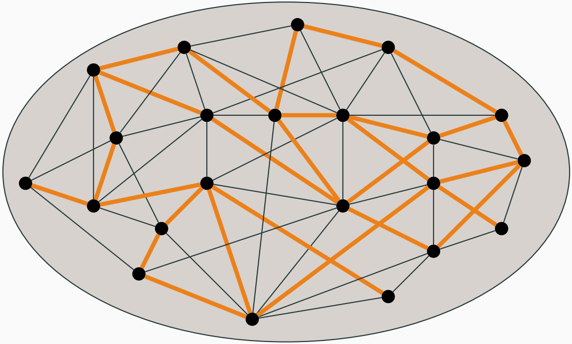
Example 1: Distance-Preserving Sparsification

Definition ([Peleg, Schäffer '89])

A **spanner** of **stretch** t of $G = (V, E)$ is a subgraph $H = (V, E')$ such that

$$dist_G(u, v) \leq dist_H(u, v) \leq t \cdot dist_G(u, v)$$

for all pairs of nodes $u, v \in V$.



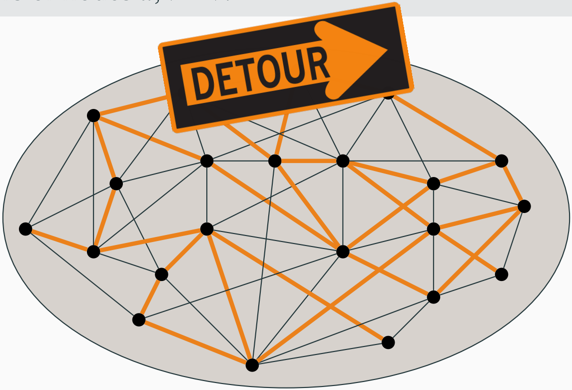
Example 1: Distance-Preserving Sparsification

Definition ([Peleg, Schäffer '89])

A **spanner** of **stretch** t of $G = (V, E)$ is a subgraph $H = (V, E')$ such that

$$dist_G(u, v) \leq dist_H(u, v) \leq t \cdot dist_G(u, v)$$

for all pairs of nodes $u, v \in V$.



Discussion

Theorem

For every integer k, every graph with n nodes admits a spanner of stretch $t = 2k - 1$ with $O(n^{1+1/k})$ edges.

Discussion

Theorem

For every integer k, every graph with n nodes admits a spanner of stretch $t = 2k - 1$ with $O(n^{1+1/k})$ edges.

- $k = 1$: stretch 1, size $O(n^2)$

Discussion

Theorem

For every integer k, every graph with n nodes admits a spanner of stretch $t = 2k - 1$ with $O(n^{1+1/k})$ edges.

- $k = 1$: stretch 1, size $O(n^2) \rightarrow$ input graph

Discussion

Theorem

For every integer k, every graph with n nodes admits a spanner of stretch $t = 2k - 1$ with $O(n^{1+1/k})$ edges.

- $k = 1$: stretch 1, size $O(n^2) \rightarrow$ input graph
- $k = 2$: stretch 3, size $O(n^{3/2})$

Discussion

Theorem

For every integer k, every graph with n nodes admits a spanner of stretch $t = 2k - 1$ with $O(n^{1+1/k})$ edges.

- $k = 1$: stretch 1, size $O(n^2) \rightarrow$ input graph
- $k = 2$: stretch 3, size $O(n^{3/2})$

$\vdots$

- $k = \log n$: stretch $O(\log n)$, size $O(n)$

Discussion

Theorem

For every integer k, every graph with n nodes admits a spanner of stretch $t = 2k - 1$ with $O(n^{1+1/k})$ edges.

- $k = 1$: stretch 1, size $O(n^2) \rightarrow$ input graph
- $k = 2$: stretch 3, size $O(n^{3/2})$

$\vdots$

- $k = \log n$: stretch $O(\log n)$, size $O(n)$

Observation

This stretch/size-tradeoff is tight under the **Girth Conjecture** by Erdős.

Discussion

Theorem

For every integer k, every graph with n nodes admits a spanner of stretch $t = 2k - 1$ with $O(n^{1+1/k})$ edges.

- $k = 1$: stretch 1, size $O(n^2) \to$ input graph
- $k = 2$: stretch 3, size $O(n^{3/2})$

$\vdots$

- $k = \log n$: stretch $O(\log n)$, size $O(n)$

Observation

This stretch/size-tradeoff is tight under the **Girth Conjecture** by Erdős.

Isn't this type of stretch guarantee very weak?

Discussion

Theorem

For every integer k, every graph with n nodes admits a spanner of stretch $t = 2k - 1$ with $O(n^{1+1/k})$ edges.

- $k = 1$: stretch 1, size $O(n^2) \rightarrow$ input graph
- $k = 2$: stretch 3, size $O(n^{3/2})$

$\vdots$

- $k = \log n$: stretch $O(\log n)$, size $O(n)$

Observation

This stretch/size-tradeoff is tight under the **Girth Conjecture** by Erdős.

Isn't this type of stretch guarantee very weak?

Distributed SSSP: **boosting** approach for better approximation

[Becker, **F**, Karrenbauer, Lenzen '17]

Our Spanner Results

Theorem ([Baswana, Khurana, Sarkar '12])

For every k, there is a randomized dynamic algorithm that maintains a spanner of stretch $t = 2k - 1$

- *with* $O(n^{1+1/k} k^8 \log^2 n)$ *and* $O(7^{k/2})$ *amortized update time,*
- *with* $O(n^{1+1/k} k \log n)$ *edges and* $O(k^2 \log^2 n)$ *amortized update time.*

Our Spanner Results

Theorem ([Baswana, Khurana, Sarkar '12])

For every k, there is a randomized dynamic algorithm that maintains a spanner of stretch $t = 2k - 1$

- *with $O(n^{1+1/k} k^8 \log^2 n)$ and $O(7^{k/2})$ amortized update time,*
- *with $O(n^{1+1/k} k \log n)$ edges and $O(k^2 \log^2 n)$ amortized update time.*

Amortized time: Bound holds on average over a sequence of updates

Our Spanner Results

Theorem ([Baswana, Khurana, Sarkar '12])

For every k, there is a randomized dynamic algorithm that maintains a spanner of stretch $t = 2k - 1$

- *with $O(n^{1+1/k} k^8 \log^2 n)$ and $O(7^{k/2})$ amortized update time,*
- *with $O(n^{1+1/k} k \log n)$ edges and $O(k^2 \log^2 n)$ amortized update time.*

Amortized time: Bound holds on average over a sequence of updates

Worst-case time: Hard upper bound for each update

Our Spanner Results

Theorem ([Baswana, Khurana, Sarkar '12])

For every k, there is a randomized dynamic algorithm that maintains a spanner of stretch $t = 2k - 1$

- *with $O(n^{1+1/k} k^8 \log^2 n)$ and $O(7^{k/2})$ amortized update time,*
- *with $O(n^{1+1/k} k \log n)$ edges and $O(k^2 \log^2 n)$ amortized update time.*

Amortized time: Bound holds on average over a sequence of updates

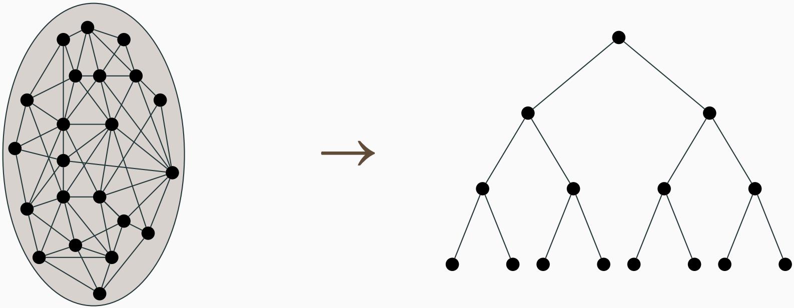
Worst-case time: Hard upper bound for each update

Theorem ([Bernstein, F, Henzinger '19])

For every k, there is a randomized dynamic algorithm that maintains a $(2k - 1)$-spanner with $O(n^{1+1/k} k \log^7 n \log \log n)$ edges and worst-case update time $O(20^{k/2} \log^3 n)$.

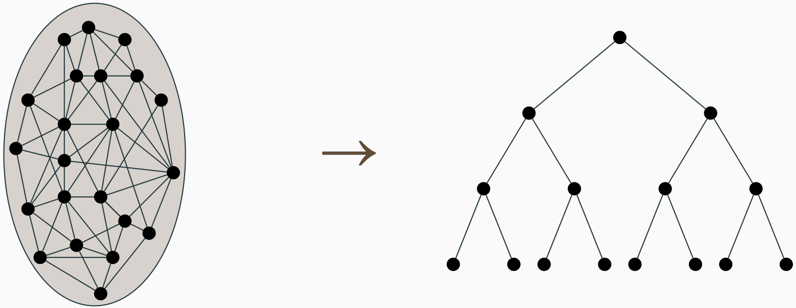
Distance-Preserving Trees

Idea: Embed distance metric into tree metric



Distance-Preserving Trees

Idea: Embed distance metric into tree metric



Results: First dynamic algorithms for tree embeddings:

- Average stretch [**F**, Goranci '19]
  (Recent improvement: [Chechik, Zhang '20])
- Expected stretch [**F**, Goranci, Henzinger '21]
  Applications to distance oracles and buy-at-bulk network design

Example II: Cut-Preserving Sparsification

Definition ([Benczúr/Karger '00])

A $(1 \pm \epsilon)$-cut sparsifier of G is a weighted subgraph H such that, for every cut $(C, V \setminus C)$, the edges $F := E[C, V \setminus C]$ crossing the cut have weight

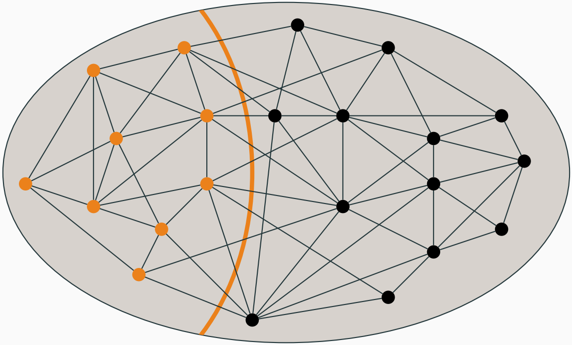
$$(1-\epsilon) \cdot w_G(F) \leq w_H(F) \leq (1+\epsilon) \cdot w_G(F)$$

Example II: Cut-Preserving Sparsification

Definition ([Benczúr/Karger '00])

A $(1 \pm \epsilon)$-cut sparsifier of G is a weighted subgraph H such that, for every cut $(C, V \setminus C)$, the edges $F := E[C, V \setminus C]$ crossing the cut have weight

$$(1-\epsilon) \cdot w_G(F) \leq w_H(F) \leq (1+\epsilon) \cdot w_G(F)$$

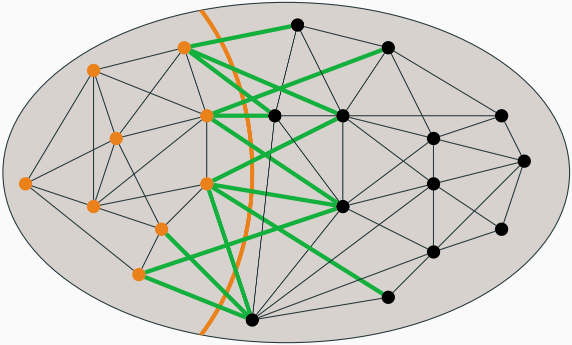


Example II: Cut-Preserving Sparsification

Definition ([Benczúr/Karger '00])

A $(1 \pm \epsilon)$-cut sparsifier of G is a weighted subgraph H such that, for every cut $(C, V \setminus C)$, the edges $F := E[C, V \setminus C]$ crossing the cut have weight

$$(1 - \epsilon) \cdot w_G(F) \leq w_H(F) \leq (1 + \epsilon) \cdot w_G(F)$$

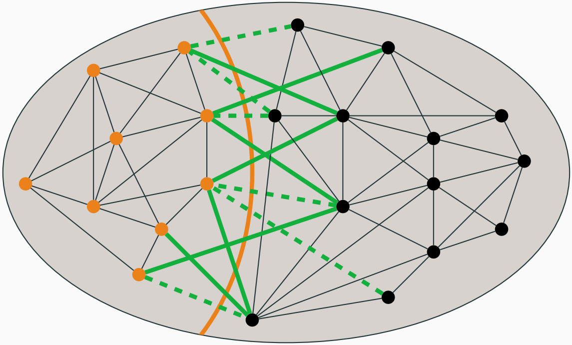


Example II: Cut-Preserving Sparsification

Definition ([Benczúr/Karger '00])

A $(1 \pm \epsilon)$-cut sparsifier of G is a weighted subgraph H such that, for every cut $(C, V \setminus C)$, the edges $F := E[C, V \setminus C]$ crossing the cut have weight

$$(1-\epsilon) \cdot w_G(F) \leq w_H(F) \leq (1+\epsilon) \cdot w_G(F)$$



Our Result

Theorem ([Batson, Spielman, Srivastava '09])

Every graph with n nodes admits a $(1 \pm \epsilon)$-cut sparsifier with $O(n\epsilon^{-2})$ edges.

Our Result

Theorem ([Batson, Spielman, Srivastava '09])

Every graph with n nodes admits a $(1 \pm \epsilon)$-cut sparsifier with $O(n\epsilon^{-2})$ edges.

Connected to solving SDD linear systems! [Spielman, Teng '04]

Our Result

Theorem ([Batson, Spielman, Srivastava '09])

Every graph with n nodes admits a $(1 \pm \epsilon)$-cut sparsifier with $O(n\epsilon^{-2})$ edges.

Connected to solving SDD linear systems! [Spielman, Teng '04]

Theorem ([Abraham, Durfee, Koutis, **K, Peng '16])**

There is a randomized dynamic algorithm for maintaining a $(1 \pm \epsilon)$-cut sparsifier sparsifier with $O(n\epsilon^{-2} \log n)$ edges in worst-case time $O(\epsilon^{-2} \log^7 n)$ per update.

Our Result

Theorem ([Batson, Spielman, Srivastava '09])

Every graph with n nodes admits a $(1 \pm \epsilon)$-cut sparsifier with $O(n\epsilon^{-2})$ edges.

Connected to solving SDD linear systems! [Spielman, Teng '04]

Theorem ([Abraham, Durfee, Koutis, K, Peng '16])

There is a randomized dynamic algorithm for maintaining a $(1 \pm \epsilon)$-cut sparsifier sparsifier with $O(n\epsilon^{-2} \log n)$ edges in worst-case time $O(\epsilon^{-2} \log^7 n)$ per update.

First dynamic algorithm for this problem

Spectral sparsifier with similar guarantees at cost of amortization

Dynamic Distance Approximation

"Gold standard":

- Fully dynamic
- Worst-case update time
- Deterministic
- Meeting an update-time barrier

Towards Assumption-Free Algorithms

“Gold standard”:

- Fully dynamic
- Worst-case update time
- Deterministic
- Meeting an update-time barrier

List of problems with such algorithms is small

Towards Assumption-Free Algorithms

“Gold standard”:

- Fully dynamic
- Worst-case update time
- Deterministic
- Meeting an update-time barrier

List of problems with such algorithms is small

Contribution

We add to this list: $(1+\epsilon)$-approximate distance approximation in unweighted, undirected graphs [van den Brand, **F**, Nazari ’22]

Our Results

Distance approximation in unweighted, undirected graphs:

Approx	Type	Update Time
$1+\epsilon$	single pair	$O(n^{1.407}\epsilon^{-2})$
$1+\epsilon$	single source	$O(n^{1.529}\epsilon^{-2})$
$1+\epsilon$	k sources	$O(n^{1.529}+kn)\cdot O(\epsilon^{-1})^{\sqrt{2\log_{1/\epsilon} n}}$
$1+\epsilon$	all pairs	$O(n^2)\cdot O(\epsilon^{-1})^{\sqrt{2\log_{1/\epsilon} n}}$

Our Results

Distance approximation in unweighted, undirected graphs:

Approx	Type	Update Time
$1+\epsilon$	single pair	$O(n^{1.407}\epsilon^{-2})$
$1+\epsilon$	single source	$O(n^{1.529}\epsilon^{-2})$
$1+\epsilon$	k sources	$O(n^{1.529}+kn)\cdot O(\epsilon^{-1})^{\sqrt{2\log_{1/\epsilon} n}}$
$1+\epsilon$	all pairs	$O(n^2)\cdot O(\epsilon^{-1})^{\sqrt{2\log_{1/\epsilon} n}}$

- Prior work was randomized

  (and had worse update time in case of single pair)

Our Results

Distance approximation in unweighted, undirected graphs:

Approx	Type	Update Time
$1+\epsilon$	single pair	$O(n^{1.407}\epsilon^{-2})$
$1+\epsilon$	single source	$O(n^{1.529}\epsilon^{-2})$
$1+\epsilon$	k sources	$O(n^{1.529}+kn)\cdot O(\epsilon^{-1})^{\sqrt{2\log_{1/\epsilon} n}}$
$1+\epsilon$	all pairs	$O(n^2)\cdot O(\epsilon^{-1})^{\sqrt{2\log_{1/\epsilon} n}}$

- Prior work was randomized
  (and had worse update time in case of single pair)
- Update times match (conditional) lower bounds [van den Brand, Nanongkai, Saranurak '19]

Our Results

Distance approximation in unweighted, undirected graphs:

Approx	Type	Update Time
$1+\epsilon$	single pair	$O(n^{1.407}\epsilon^{-2})$
$1+\epsilon$	single source	$O(n^{1.529}\epsilon^{-2})$
$1+\epsilon$	k sources	$O(n^{1.529}+kn)\cdot O(\epsilon^{-1})^{\sqrt{2\log_{1/\epsilon} n}}$
$1+\epsilon$	all pairs	$O(n^2)\cdot O(\epsilon^{-1})^{\sqrt{2\log_{1/\epsilon} n}}$

- Prior work was randomized
  (and had worse update time in case of single pair)
- Update times match (conditional) lower bounds [van den Brand, Nanongkai, Saranurak '19]

Warm-up: *Randomized* $(1+\epsilon)$-approximate single-source

Our Approach

Idea

Maintain sparsifier and recompute from scratch on sparsifier

Our Approach

Idea

Maintain sparsifier and recompute from scratch on sparsifier

- Maintain hitting set for neighbors of nodes of degree $\geq \sqrt{n}$
- Maintain $\Theta(1/\epsilon)$-bounded distances to all nodes from hitting set nodes and source node s

Our Approach

Idea

Maintain sparsifier and recompute from scratch on sparsifier

- Maintain hitting set for neighbors of nodes of degree $\geq \sqrt{n}$
- Maintain $\Theta(1/\epsilon)$-bounded distances to all nodes from hitting set nodes and source node s
- Additionally, after each update:
    - Obtain $\Theta(1/\epsilon)$-bounded distances $\hat{d}_G(\cdot, \cdot)$
    - Compute $(1 + \epsilon, 2)$-emulator H of size $\tilde{O}(n^{1.5})$

Our Approach

Idea

Maintain sparsifier and recompute from scratch on sparsifier

- Maintain hitting set for neighbors of nodes of degree $\geq \sqrt{n}$
- Maintain $\Theta(1/\epsilon)$-bounded distances to all nodes from hitting set nodes and source node s
- Additionally, after each update:
  - Obtain $\Theta(1/\epsilon)$-bounded distances $\hat{d}_G(\cdot,\cdot)$
  - Compute $(1+\epsilon, 2)$-emulator H of size $\tilde{O}(n^{1.5})$
  - Compute (exact) single-source distances on H
  - Return $\min(\hat{d}_G(s,v), d_H(s,v))$ for every node v

Our Approach

Idea

Maintain sparsifier and recompute from scratch on sparsifier

- Maintain hitting set for neighbors of nodes of degree $\geq \sqrt{n}$
- Maintain $\Theta(1/\epsilon)$-bounded distances to all nodes from hitting set nodes and source node s
- Additionally, after each update:
    - Obtain $\Theta(1/\epsilon)$-bounded distances $\hat{d}_G(\cdot,\cdot)$
    - Compute $(1+\epsilon, 2)$-emulator H of size $\tilde{O}(n^{1.5})$
    - Compute (exact) single-source distances on H
    - Return $\min(\hat{d}_G(s,v), d_H(s,v))$ for every node v

Related work

Randomized algorithm for maintaining $(1+\epsilon, n^{o(1)})$-spanner of size $n^{1+o(1)}$ with update time $O(n^{1.529})$ [Bergamaschi et al. '21]

Hitting Set

Hitting Set

We maintain a set of nodes $S \subseteq V$ of size $\tilde{O}(\sqrt{n})$ such that every heavy node of degree $> \sqrt{n}$ has at least one node of S in its neighborhood.

Hitting Set

Hitting Set

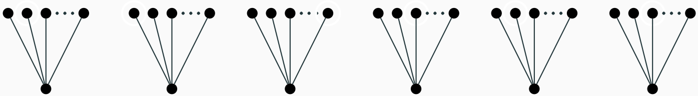
We maintain a set of nodes $S \subseteq V$ of size $\tilde{O}(\sqrt{n})$ such that every heavy node of degree $> \sqrt{n}$ has at least one node of S in its neighborhood.

Randomized approach: Initially, sample a set of size $\tilde{\Theta}(\sqrt{n})$ uniformly at random [Ullman, Yannakakis '90]

Hitting Set

We maintain a set of nodes $S \subseteq V$ of size $\tilde{O}(\sqrt{n})$ such that every heavy node of degree $> \sqrt{n}$ has at least one node of S in its neighborhood.

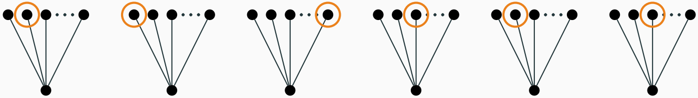
Randomized approach: Initially, sample a set of size $\tilde{\Theta}(\sqrt{n})$ uniformly at random [Ullman, Yannakakis '90]



Hitting Set

We maintain a set of nodes $S \subseteq V$ of size $\tilde{O}(\sqrt{n})$ such that every heavy node of degree $> \sqrt{n}$ has at least one node of S in its neighborhood.

Randomized approach: Initially, sample a set of size $\tilde{\Theta}(\sqrt{n})$ uniformly at random [Ullman, Yannakakis '90]



Emulator Construction

Definition

A $(1+\epsilon, \beta)$-**emulator** of $G = (V, E)$ is a graph $H = (V, E')$ such that

$$dist_G(u, v) \leq dist_H(u, v) \leq (1+\epsilon) \cdot dist_G(u, v) + \beta$$

for all pairs of nodes $u, v \in V$.

Emulator Construction

Definition

A $(1+\epsilon, \beta)$-**emulator** of $G = (V, E)$ is a graph $H = (V, E')$ such that
$$dist_G(u,v) \leq dist_H(u,v) \leq (1+\epsilon) \cdot dist_G(u,v) + \beta$$
for all pairs of nodes $u, v \in V$.

Emulator H has two types of edges:

- For every light node of degree $\leq \sqrt{n}$: edges to all neighbors
- For every node in hitting set: (weighted) edges to all nodes in distance $\leq \lceil 6/\epsilon \rceil$

similar to [Henzinger, **K**, Nanongkai ’13; Dor, Halperin, Zwick ’97]

Emulator Construction

Definition

A $(1+\epsilon, \beta)$-**emulator** of $G = (V, E)$ is a graph $H = (V, E')$ such that

$$dist_G(u,v) \leq dist_H(u,v) \leq (1+\epsilon) \cdot dist_G(u,v) + \beta$$

for all pairs of nodes $u, v \in V$.

Emulator H has two types of edges:

- For every light node of degree $\leq \sqrt{n}$: edges to all neighbors
- For every node in hitting set: (weighted) edges to all nodes in distance $\leq \lceil 6/\epsilon \rceil$

similar to [Henzinger, **K**, Nanongkai '13; Dor, Halperin, Zwick '97]

Lemma

H is a $(1+\frac{\epsilon}{2}, 2)$-emulator of size $\tilde{O}(n^{1.5})$

Emulator Construction

Definition

A $(1+\epsilon, \beta)$-**emulator** of $G = (V, E)$ is a graph $H = (V, E')$ such that

$$dist_G(u,v) \leq dist_H(u,v) \leq (1+\epsilon) \cdot dist_G(u,v) + \beta$$

for all pairs of nodes $u, v \in V$.

Emulator H has two types of edges:

- For every light node of degree $\leq \sqrt{n}$: edges to all neighbors
- For every node in hitting set: (weighted) edges to all nodes in distance $\leq \lceil 6/\epsilon \rceil$

similar to [Henzinger, **K**, Nanongkai '13; Dor, Halperin, Zwick '97]

Lemma

H is a $(1+\frac{\epsilon}{2}, 2)$-emulator of size $\tilde{O}(n^{1.5})$

→ single-source distance on H in time $\tilde{O}(n^{1.5})$

Approximation Guarantee

Subdivide any shortest path into segments of length $\lceil 6/\epsilon \rceil$ (with potentially one segment of smaller length)

Approximation Guarantee

Subdivide any shortest path into segments of length $\lceil 6/\epsilon \rceil$ (with potentially one segment of smaller length)

- Case 1: Segment contains no high-degree node

Subdivide any shortest path into segments of length $\lceil 6/\epsilon \rceil$ (with potentially one segment of smaller length)

- Case 1: Segment contains no high-degree node

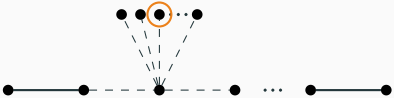
- Case 2: Segment contains high-degree node

Approximation Guarantee

Subdivide any shortest path into segments of length $\lceil 6/\epsilon \rceil$ (with potentially one segment of smaller length)

- Case 1: Segment contains no high-degree node

- Case 2: Segment contains high-degree node

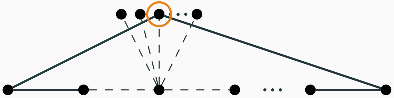


Approximation Guarantee

Subdivide any shortest path into segments of length $\lceil 6/\epsilon \rceil$ (with potentially one segment of smaller length)

- Case 1: Segment contains no high-degree node

- Case 2: Segment contains high-degree node



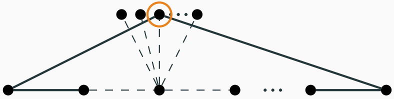
$\rightarrow$ Detour of additive surplus 2

Approximation Guarantee

Subdivide any shortest path into segments of length $\lceil 6/\epsilon \rceil$ (with potentially one segment of smaller length)

- Case 1: Segment contains no high-degree node

- Case 2: Segment contains high-degree node



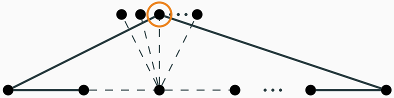
$\rightarrow$ Detour of additive surplus 2

- If segment has length $\lceil 6/\epsilon \rceil$, then multiplicative error of $\leq \frac{\lceil 6/\epsilon \rceil + 2}{\lceil 6/\epsilon \rceil} \leq \frac{6/\epsilon + 3}{6/\epsilon} = 1 + \frac{\epsilon}{2}$

Subdivide any shortest path into segments of length $\lceil 6/\epsilon \rceil$ (with potentially one segment of smaller length)

- Case 1: Segment contains no high-degree node

- Case 2: Segment contains high-degree node



$\rightarrow$ Detour of additive surplus 2

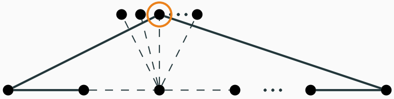
  - If segment has length $\lceil 6/\epsilon \rceil$, then multiplicative error of $\leq \frac{\lceil 6/\epsilon \rceil + 2}{\lceil 6/\epsilon \rceil} \leq \frac{6/\epsilon + 3}{6/\epsilon} = 1 + \frac{\epsilon}{2}$
  - If segment has length $< \lceil 6/\epsilon \rceil$, then additive error of 2

Approximation Guarantee

Subdivide any shortest path into segments of length $\lceil 6/\epsilon \rceil$ (with potentially one segment of smaller length)

- Case 1: Segment contains no high-degree node

- Case 2: Segment contains high-degree node



→ Detour of additive surplus 2

  - If segment has length $\lceil 6/\epsilon \rceil$, then multiplicative error of $\leq \frac{\lceil 6/\epsilon \rceil + 2}{\lceil 6/\epsilon \rceil} \leq \frac{6/\epsilon + 3}{6/\epsilon} = 1 + \frac{\epsilon}{2}$
  - If segment has length $< \lceil 6/\epsilon \rceil$, then additive error of 2

Overall: multiplicative error of $1 + \frac{\epsilon}{2}$, additive error of 2

Algebraic Data Structure

Theorem ([Sankowski '05])

Given any $0 < \delta < 1$ and any sets $A, B \subseteq V$, there is a randomized algorithm for maintaining the $S \times V$ distances up to $\leq \Delta$ with update time $\tilde{O}(\Delta(n^{\omega(1,\delta,1)-\delta} + n^{1+\delta} + |A||B|))$.

Algebraic Data Structure

Theorem ([Sankowski '05])

Given any $0 < \delta < 1$ and any sets $A, B \subseteq V$, there is a randomized algorithm for maintaining the $S \times V$ distances up to $\leq \Delta$ with update time $\tilde{O}(\Delta(n^{\omega(1,\delta,1)-\delta} + n^{1+\delta} + |A||B|))$.

- $O(n^{\omega(1,\delta,1)})$ denotes time needed for multiplying an $n \times n^{\delta}$ matrix with an $n^{\delta} \times n$ matrix

Algebraic Data Structure

Theorem ([Sankowski '05])

Given any $0 < \delta < 1$ and any sets $A, B \subseteq V$, there is a randomized algorithm for maintaining the $S \times V$ distances up to $\leq \Delta$ with update time $\tilde{O}(\Delta(n^{\omega(1,\delta,1)-\delta} + n^{1+\delta} + |A||B|))$.

- $O(n^{\omega(1,\delta,1)})$ denotes time needed for multiplying an $n \times n^{\delta}$ matrix with an $n^{\delta} \times n$ matrix
- With $\delta = 0.528\ldots$, update time is $\tilde{O}(\Delta(n^{1.529} + n^{\alpha+\beta}))$

Algebraic Data Structure

Theorem ([Sankowski '05])

Given any $0 < \delta < 1$ and any sets $A, B \subseteq V$, there is a randomized algorithm for maintaining the $S \times V$ distances up to $\leq \Delta$ with update time $\tilde{O}(\Delta(n^{\omega(1,\delta,1)-\delta} + n^{1+\delta} + |A||B|))$.

- $O(n^{\omega(1,\delta,1)})$ denotes time needed for multiplying an $n \times n^{\delta}$ matrix with an $n^{\delta} \times n$ matrix
- With $\delta = 0.528\ldots$, update time is $\tilde{O}(\Delta(n^{1.529} + n^{\alpha+\beta}))$
- With $A = S \cup \{s\}$, $B = V$ (where $|S| = \tilde{O}(\sqrt{n})$), and $\Delta = O(1/\epsilon)$: update time $O(n^{1.529}/\epsilon)$

Algebraic Data Structure

Theorem ([Sankowski '05])

Given any $0 < \delta < 1$ and any sets $A, B \subseteq V$, there is a randomized algorithm for maintaining the $S \times V$ distances up to $\leq \Delta$ with update time $\tilde{O}(\Delta(n^{\omega(1,\delta,1)-\delta} + n^{1+\delta} + |A||B|))$.

- $O(n^{\omega(1,\delta,1)})$ denotes time needed for multiplying an $n \times n^{\delta}$ matrix with an $n^{\delta} \times n$ matrix
- With $\delta = 0.528\ldots$, update time is $\tilde{O}(\Delta(n^{1.529} + n^{\alpha+\beta}))$
- With $A = S \cup \{s\}$, $B = V$ (where $|S| = \tilde{O}(\sqrt{n})$), and $\Delta = O(1/\epsilon)$: update time $O(n^{1.529}/\epsilon)$

Approximation Guarantee:

- If $d_G(s, v) \leq \lceil 6/\epsilon \rceil$: distance from algebraic data structure

Algebraic Data Structure

Theorem ([Sankowski '05])

Given any $0 < \delta < 1$ and any sets $A, B \subseteq V$, there is a randomized algorithm for maintaining the $S \times V$ distances up to $\leq \Delta$ with update time $\tilde{O}(\Delta(n^{\omega(1,\delta,1)-\delta} + n^{1+\delta} + |A||B|))$.

- $O(n^{\omega(1,\delta,1)})$ denotes time needed for multiplying an $n \times n^{\delta}$ matrix with an $n^{\delta} \times n$ matrix
- With $\delta = 0.528\ldots$, update time is $\tilde{O}(\Delta(n^{1.529} + n^{\alpha+\beta}))$
- With $A = S \cup \{s\}$, $B = V$ (where $|S| = \tilde{O}(\sqrt{n})$), and $\Delta = O(1/\epsilon)$: update time $O(n^{1.529}/\epsilon)$

Approximation Guarantee:

- If $d_G(s, v) \leq \lceil 6/\epsilon \rceil$: distance from algebraic data structure
- If $d_G(s, v) > \lceil 6/\epsilon \rceil$, then approximation from H becomes $(1 + \frac{\epsilon}{2})d_G(s, v) + 2 \leq (1 + \frac{\epsilon}{2})d_G(s, v) + \frac{\epsilon}{3}d_G(s, v) \leq (1 + \epsilon)d_G(s, v)$

Observations:

- Randomization not necessary in algebraic data structure for very small distances

Towards Deterministic Algorithm

Observations:

- Randomization not necessary in algebraic data structure for very small distances
- Hitting set for neighborhoods can be maintained with a lazy approach giving low recourse
  (Each update affects at most two neighborhoods!)

Towards Deterministic Algorithm

Observations:

- Randomization not necessary in algebraic data structure for very small distances
- Hitting set for neighborhoods can be maintained with a lazy approach giving low recourse
  (Each update affects at most two neighborhoods!)
- Algebraic data structure can be extended to slowly changing set of nodes

Conclusion

Questions

- Can we close the “qualitative” gaps between static and dynamic sparsification?

Questions

- Can we close the "qualitative" gaps between static and dynamic sparsification?
- For which problems can we reach the "gold standard"

Questions

- Can we close the “qualitative” gaps between static and dynamic sparsification?
- For which problems can we reach the “gold standard”
- Are there “natural” separations?

Thank you!