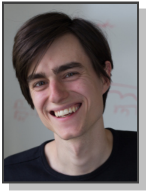
Fully Dynamic Spanners with Worst-Case Update Time Guarantees

Greg Bodwin
Stanford University



Sebastian Krinninger
Max Planck Institute for Informatics

European Symposium on Algorithms 2016

Motivation

Computing on Sparser Graphs

- **Idea:** Sparsify graph while (approximately) preserving relevant properties

Motivation

Computing on Sparser Graphs

- **Idea:** Sparsify graph while (approximately) preserving relevant properties
- **Goal:** Graph with $m' \ll n^2$ edges (where n is number of nodes)

Motivation

Computing on Sparser Graphs

- **Idea:** Sparsify graph while (approximately) preserving relevant properties
- **Goal:** Graph with $m' \ll n^2$ edges (where n is number of nodes)
- Improves running time / space requirements of algorithms

Motivation

Computing on Sparser Graphs

- **Idea:** Sparsify graph while (approximately) preserving relevant properties
- **Goal:** Graph with $m' \ll n^2$ edges (where n is number of nodes)
- Improves running time / space requirements of algorithms
- Sparsification was key to recent progress in dynamic algorithms
- Study sparsification as dynamic problem on its own

Motivation

Computing on Sparser Graphs

- **Idea:** Sparsify graph while (approximately) preserving relevant properties
- **Goal:** Graph with $m' \ll n^2$ edges (where n is number of nodes)
- Improves running time / space requirements of algorithms
- Sparsification was key to recent progress in dynamic algorithms
- Study sparsification as dynamic problem on its own

Amortized vs. **Worst-Case** Bounds

Motivation

Computing on Sparser Graphs

- **Idea:** Sparsify graph while (approximately) preserving relevant properties
- **Goal:** Graph with $m' \ll n^2$ edges (where n is number of nodes)
- Improves running time / space requirements of algorithms
- Sparsification was key to recent progress in dynamic algorithms
- Study sparsification as dynamic problem on its own

Amortized vs. **Worst-Case** Bounds

- Many dynamic algorithms amortize running time over sequence of updates

Motivation

Computing on Sparser Graphs

- **Idea:** Sparsify graph while (approximately) preserving relevant properties
- **Goal:** Graph with $m' \ll n^2$ edges (where n is number of nodes)
- Improves running time / space requirements of algorithms
- Sparsification was key to recent progress in dynamic algorithms
- Study sparsification as dynamic problem on its own

Amortized vs. **Worst-Case** Bounds

- Many dynamic algorithms amortize running time over sequence of updates
- Not suitable for real-time systems: Hard guarantees needed

Spanners

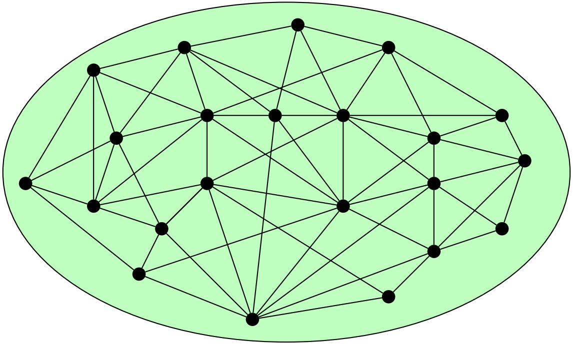
Definition

A spanner of **stretch** k is a subgraph H of G such that, for all pairs of nodes u and v, $dist_H(u, v) \leq k \cdot dist_G(u, v)$.

Spanners

Definition

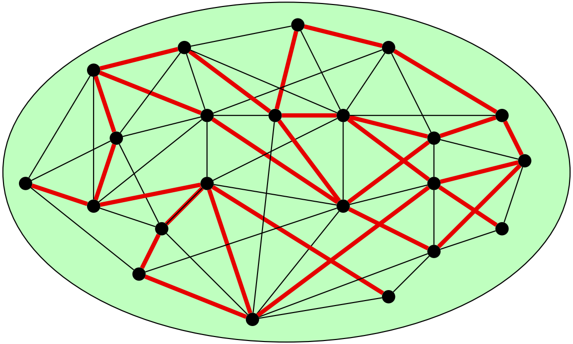
A spanner of **stretch** k is a subgraph H of G such that, for all pairs of nodes u and v, $dist_H(u,v) \leq k \cdot dist_G(u,v)$.



Spanners

Definition

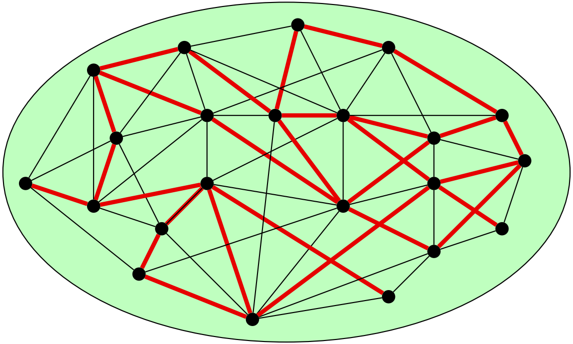
A spanner of **stretch** k is a subgraph H of G such that, for all pairs of nodes u and v, $dist_H(u,v) \leq k \cdot dist_G(u,v)$.



Spanners

Definition

A spanner of **stretch** k is a subgraph H of G such that, for all pairs of nodes u and v, $dist_H(u, v) \leq k \cdot dist_G(u, v)$.

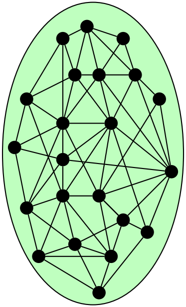


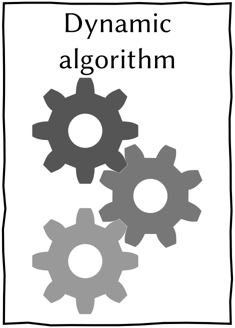
Fact: Every graph has a $(2k-1)$-spanner of size $n^{1+1/k}$ $(k \geq 2)$ [Folklore]

Essentially tight if **girth conjecture** is true [Erdős]

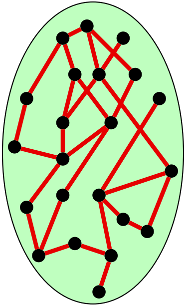
Dynamic Problem

undirected G

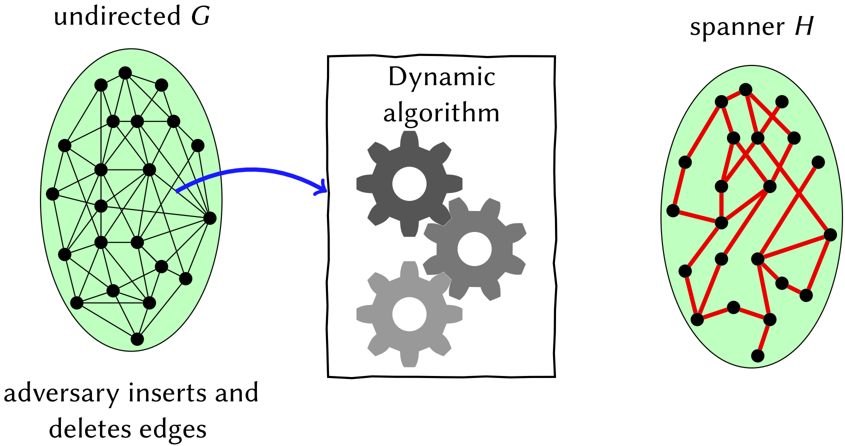




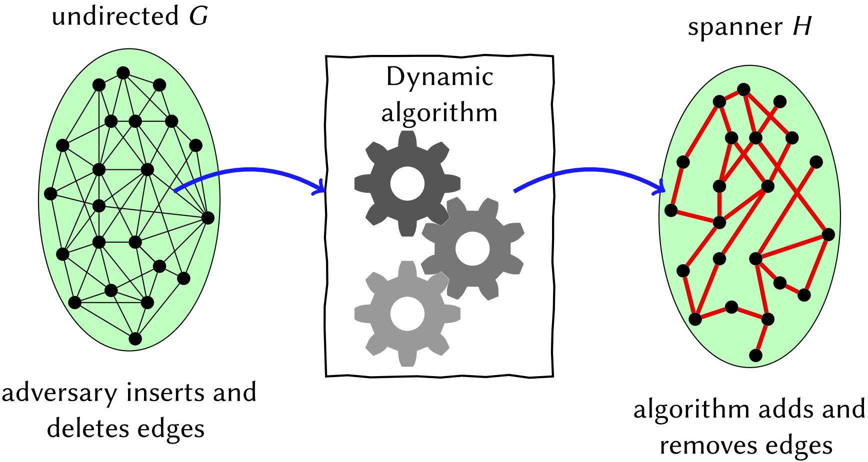
spanner H



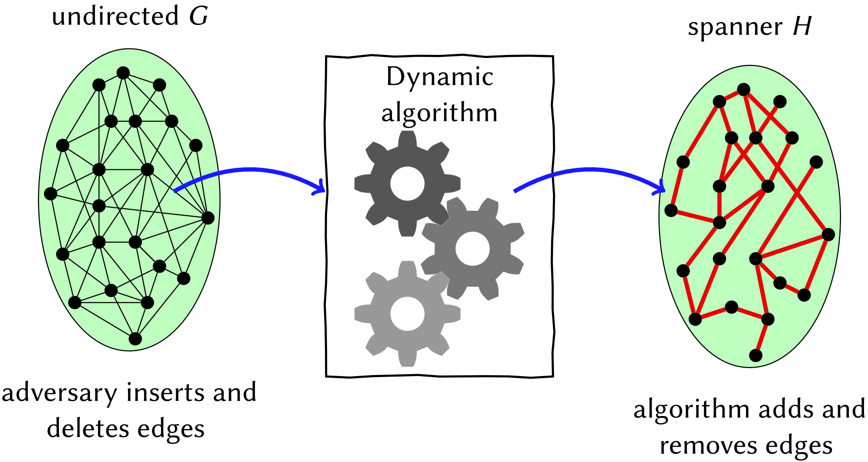
Dynamic Problem



Dynamic Problem



Dynamic Problem



Goal: Maintain edges of spanner H with small update time after edge insertion/deletion in G

Our Results and Related Work

Amortized bounds:

stretch	*size*	*time*	*reference*

Our Results and Related Work

Amortized bounds:

stretch	*size*	*time*	*reference*
3	$O(n^{1+1/2})$	$O(n)$	[Ausiello et al.]
5	$O(n^{1+1/3})$	$O(n)$	[Ausiello et al.]

Our Results and Related Work

Amortized bounds:

stretch	*size*	*time*	*reference*
3	$O(n^{1+1/2})$	$O(n)$	[Ausiello et al.]
5	$O(n^{1+1/3})$	$O(n)$	[Ausiello et al.]
$2k-1$	$O(n^{1+1/k} k \log n)$	$O(k^2 \log^2 n)$	[Baswana et al.]
$2k-1$	$O(n^{1+1/k} k^8 \log^2 n)$	$O(7^k)$	[Baswana et al.]

Our Results and Related Work

Amortized bounds:

stretch	*size*	*time*	*reference*
3	$O(n^{1+1/2})$	$O(n)$	[Ausiello et al.]
5	$O(n^{1+1/3})$	$O(n)$	[Ausiello et al.]
$2k-1$	$O(n^{1+1/k} k \log n)$	$O(k^2 \log^2 n)$	[Baswana et al.]
$2k-1$	$O(n^{1+1/k} k^8 \log^2 n)$	$O(7^k)$	[Baswana et al.]

Worst-case bounds:

Our Results and Related Work

Amortized bounds:

stretch	*size*	*time*	*reference*
3	$O(n^{1+1/2})$	$O(n)$	[Ausiello et al.]
5	$O(n^{1+1/3})$	$O(n)$	[Ausiello et al.]
$2k-1$	$O(n^{1+1/k} k \log n)$	$O(k^2 \log^2 n)$	[Baswana et al.]
$2k-1$	$O(n^{1+1/k} k^8 \log^2 n)$	$O(7^k)$	[Baswana et al.]

Worst-case bounds:

stretch	*size*	*time*	*reference*
$2k-1$	$O(n^{1+1/k} k \log^{1-1/k} n)$	$O(mn^{-1/k} \log^{1/k} n)$	[Elkin]

Our Results and Related Work

Amortized bounds:

stretch	*size*	*time*	*reference*
3	$O(n^{1+1/2})$	$O(n)$	[Ausiello et al.]
5	$O(n^{1+1/3})$	$O(n)$	[Ausiello et al.]
$2k-1$	$O(n^{1+1/k} k \log n)$	$O(k^2 \log^2 n)$	[Baswana et al.]
$2k-1$	$O(n^{1+1/k} k^8 \log^2 n)$	$O(7^k)$	[Baswana et al.]

Worst-case bounds:

stretch	*size*	*time*	*reference*
$2k-1$	$O(n^{1+1/k} k \log^{1-1/k} n)$	$O(mn^{-1/k} \log^{1/k} n)$	[Elkin]
3	$O(n^{1+1/2} \log^{1/2} n \log \log n)$	$O(n^{3/4} \log^4 n)$	**Our result**
5	$O(n^{1+1/3} \log^{2/3} n \log \log n)$	$O(n^{5/9} \log^4 n)$	**Our result**

Our Results and Related Work

Amortized bounds:

stretch	*size*	*time*	*reference*
3	$O(n^{1+1/2})$	$O(n)$	[Ausiello et al.]
5	$O(n^{1+1/3})$	$O(n)$	[Ausiello et al.]
$2k-1$	$O(n^{1+1/k} k \log n)$	$O(k^2 \log^2 n)$	[Baswana et al.]
$2k-1$	$O(n^{1+1/k} k^8 \log^2 n)$	$O(7^k)$	[Baswana et al.]

Worst-case bounds:

stretch	*size*	*time*	*reference*
$2k-1$	$O(n^{1+1/k} k \log^{1-1/k} n)$	$O(mn^{-1/k} \log^{1/k} n)$	[Elkin]
3	$O(n^{1+1/2} \log^{1/2} n \log \log n)$	$O(n^{3/4} \log^4 n)$	**Our result**
5	$O(n^{1+1/3} \log^{2/3} n \log \log n)$	$O(n^{5/9} \log^4 n)$	**Our result**

$\Rightarrow$ We give **first** sublinear worst-case bounds

Our Results and Related Work

Amortized bounds:

stretch	*size*	*time*	*reference*
3	$O(n^{1+1/2})$	$O(n)$	[Ausiello et al.]
5	$O(n^{1+1/3})$	$O(n)$	[Ausiello et al.]
$2k-1$	$O(n^{1+1/k} k \log n)$	$O(k^2 \log^2 n)$	[Baswana et al.]
$2k-1$	$O(n^{1+1/k} k^8 \log^2 n)$	$O(7^k)$	[Baswana et al.]

Worst-case bounds:

stretch	*size*	*time*	*reference*
$2k-1$	$O(n^{1+1/k} k \log^{1-1/k} n)$	$O(mn^{-1/k} \log^{1/k} n)$	[Elkin]
3	$O(n^{1+1/2} \log^{1/2} n \log\log n)$	$O(n^{3/4} \log^4 n)$	**Our result**
5	$O(n^{1+1/3} \log^{2/3} n \log\log n)$	$O(n^{5/9} \log^4 n)$	**Our result**

$\Rightarrow$ We give **first** sublinear worst-case bounds

Guarantees with high probability against oblivious adversary

Our Results and Related Work

Amortized bounds:

stretch	*size*	*time*	*reference*
3	$O(n^{1+1/2})$	$O(n)$	[Ausiello et al.]
5	$O(n^{1+1/3})$	$O(n)$	[Ausiello et al.]
$2k-1$	$O(n^{1+1/k} k \log n)$	$O(k^2 \log^2 n)$	[Baswana et al.]
$2k-1$	$O(n^{1+1/k} k^8 \log^2 n)$	$O(7^k)$	[Baswana et al.]

Worst-case bounds:

stretch	*size*	*time*	*reference*
$2k-1$	$O(n^{1+1/k} k \log^{1-1/k} n)$	$O(mn^{-1/k} \log^{1/k} n)$	[Elkin]
3	$O(n^{1+1/2} \log^{1/2} n \log \log n)$	$O(n^{3/4} \log^4 n)$	**Our result**
5	$O(n^{1+1/3} \log^{2/3} n \log \log n)$	$O(n^{5/9} \log^4 n)$	**Our result**

$\Rightarrow$ We give **first** sublinear worst-case bounds

Guarantees with high probability against oblivious adversary

This talk: *Sparsification of paper* (reduces time until BBQ)

Our Results and Related Work

Amortized bounds:

stretch	*size*	*time*	*reference*
3	$O(n^{1+1/2})$	$O(n)$	[Ausiello et al.]
5	$O(n^{1+1/3})$	$O(n)$	[Ausiello et al.]
$2k-1$	$O(n^{1+1/k} k \log n)$	$O(k^2 \log^2 n)$	[Baswana et al.]
$2k-1$	$O(n^{1+1/k} k^8 \log^2 n)$	$O(7^k)$	[Baswana et al.]

Worst-case bounds:

stretch	*size*	*time*	*reference*
$2k-1$	$O(n^{1+1/k} k \log^{1-1/k} n)$	$O(mn^{-1/k} \log^{1/k} n)$	[Elkin]
3	$O(n^{1+1/2} \log^{1/2} n \log\log n)$	$O(n^{3/4} \log^4 n)$	**Our result**
5	$O(n^{1+1/3} \log^{2/3} n \log\log n)$	$O(n^{5/9} \log^4 n)$	**Our result**

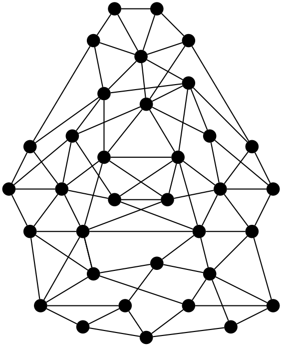
⇒ We give **first** sublinear worst-case bounds

Guarantees with high probability against oblivious adversary

This talk: *Sparsification of paper* (reduces time until BBQ)

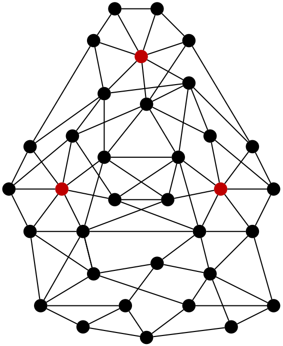
Will only show: stretch 3 in worst-case update time $O(n^{5/6})$

Spanner by Randomized Clustering



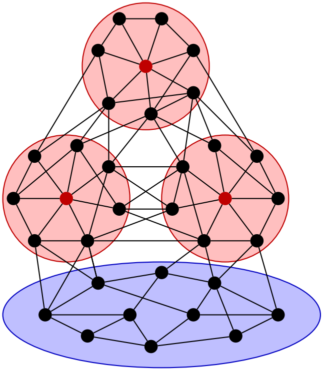
Spanner by Randomized Clustering

1. Pick $O(\sqrt{n}\log n)$ centers at random



Spanner by Randomized Clustering

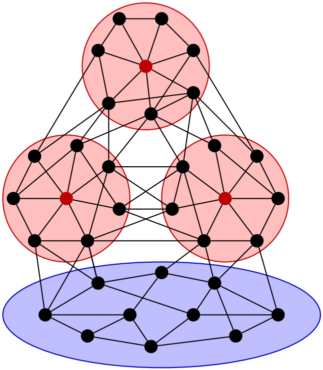
1. Pick $O(\sqrt{n} \log n)$ centers at random
2. Form clusters: Connect every node to one of its neighboring centers



Spanner by Randomized Clustering

1. Pick $O(\sqrt{n} \log n)$ centers at random
2. Form clusters: Connect every node to one of its neighboring centers

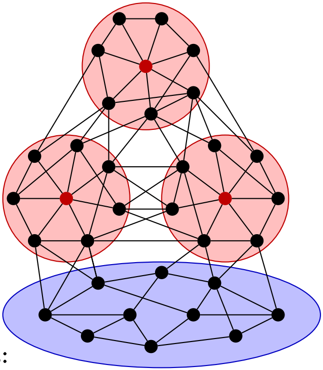
$\Rightarrow$ Unclustered nodes have at most $\sqrt{n}$ neighbors with high probability



Spanner by Randomized Clustering

1. Pick $O(\sqrt{n} \log n)$ centers at random
2. Form clusters: Connect every node to one of its neighboring centers

$\Rightarrow$ Unclustered nodes have at most $\sqrt{n}$ neighbors with high probability

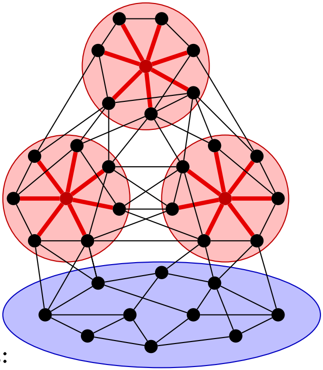


At any time, spanner consists of following edges:

Spanner by Randomized Clustering

1. Pick $O(\sqrt{n}\log n)$ centers at random
2. Form clusters: Connect every node to one of its neighboring centers

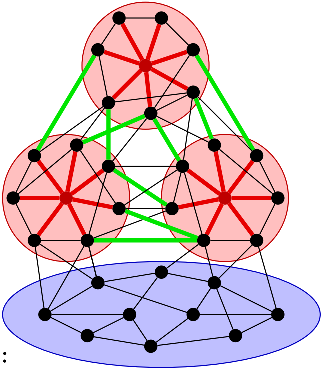
$\Rightarrow$ Unclustered nodes have at most $\sqrt{n}$ neighbors with high probability



At any time, spanner consists of following edges:

1. For every clustered node, edge to cluster center

Spanner by Randomized Clustering



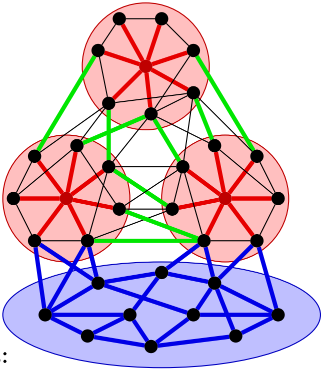
1. Pick $O(\sqrt{n}\log n)$ centers at random
2. Form clusters: Connect every node to one of its neighboring centers

$\Rightarrow$ Unclustered nodes have at most $\sqrt{n}$ neighbors with high probability

At any time, spanner consists of following edges:

1. For every clustered node, edge to cluster center
2. For every clustered node v and every other cluster, **one** edge from v to other cluster

Spanner by Randomized Clustering



1. Pick $O(\sqrt{n}\log n)$ centers at random
2. Form clusters: Connect every node to one of its neighboring centers

$\Rightarrow$ Unclustered nodes have at most $\sqrt{n}$ neighbors with high probability

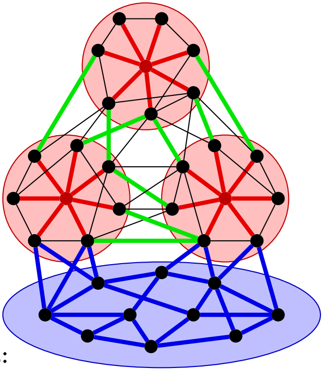
At any time, spanner consists of following edges:

1. For every clustered node, edge to cluster center
2. For every clustered node v and every other cluster, **one** edge from v to other cluster
3. For every node, edge to its **first** $\sqrt{n}$ neighbors

Spanner by Randomized Clustering

1. Pick $O(\sqrt{n}\log n)$ centers at random
2. Form clusters: Connect every node to one of its neighboring centers

$\Rightarrow$ Unclustered nodes have at most $\sqrt{n}$ neighbors with high probability



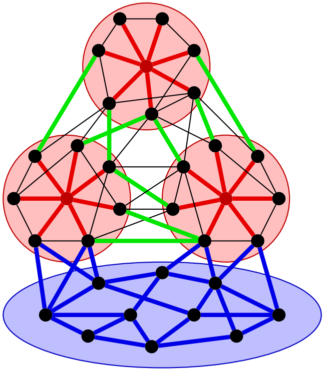
At any time, spanner consists of following edges:

1. For every clustered node, edge to cluster center
2. For every clustered node v and every other cluster, **one** edge from v to other cluster
3. For every node, edge to its **first** $\sqrt{n}$ neighbors

$\Rightarrow$ Spanner has stretch 3 and size $O(n^{1+1/2}\log n)$ whp (standard proof)

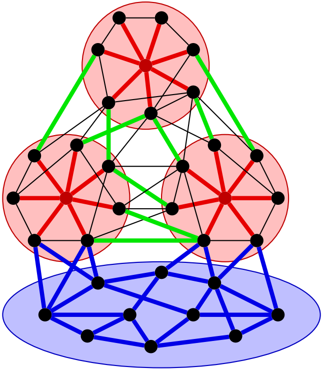
Maintaining Spanner I

- Random choice of centers at initialization



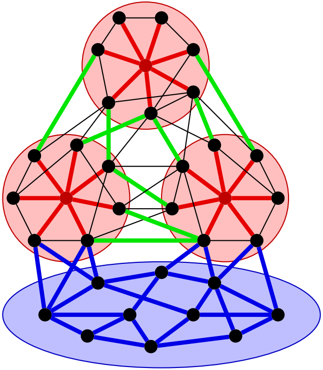
Maintaining Spanner I

- Random choice of centers at initialization
- Nodes might join or leave clusters after update in G



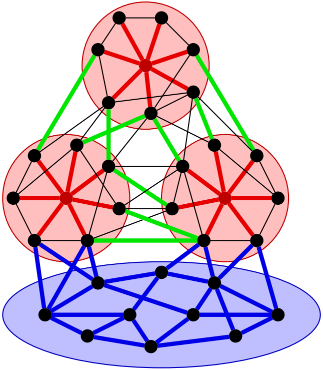
Maintaining Spanner I

- Random choice of centers at initialization
- Nodes might join or leave clusters after update in G
- For every clustered node v and every other cluster C, maintain set $N(v, C)$: edges between v and C



Maintaining Spanner I

- Random choice of centers at initialization
- Nodes might join or leave clusters after update in G
- For every clustered node v and every other cluster C, maintain set $N(v, C)$: edges between v and C
- Keep **one** entry of $N(v, C)$ in spanner

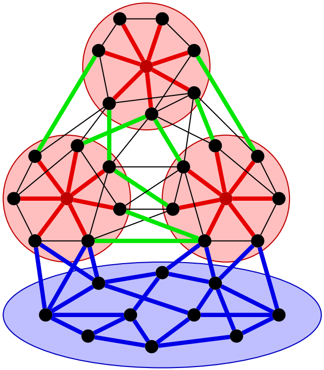


Maintaining Spanner I

- Random choice of centers at initialization
- Nodes might join or leave clusters after update in G
- For every clustered node v and every other cluster C, maintain set $N(v, C)$: edges between v and C
- Keep **one** entry of $N(v, C)$ in spanner
- Whenever node u changes from cluster C to cluster C':
  For every incident edge (u, v)

  Remove (u, v) from $N(v, C)$

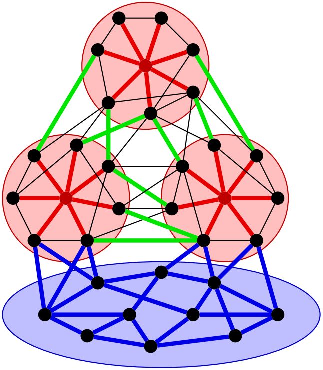
  Add (u, v) to $N(v, C')$



Maintaining Spanner I

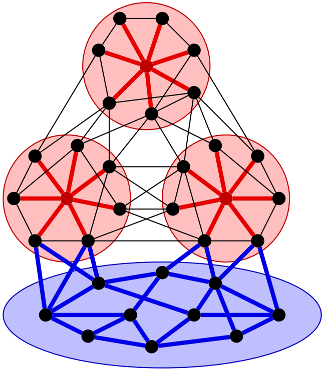
- Random choice of centers at initialization
- Nodes might join or leave clusters after update in G
- For every clustered node v and every other cluster C, maintain set $N(v, C)$: edges between v and C
- Keep **one** entry of $N(v, C)$ in spanner
- Whenever node u changes from cluster C to cluster C':
  For every incident edge (u, v)
  Remove (u, v) from $N(v, C)$
  Add (u, v) to $N(v, C')$

$\Rightarrow$ Update time: $O(maxdeg(G) \log n)$



Maintaining Spanner II

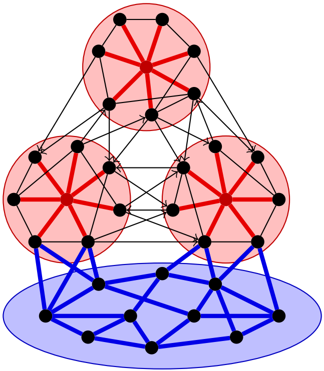
More fine-grained approach:



Maintaining Spanner II

More fine-grained approach:

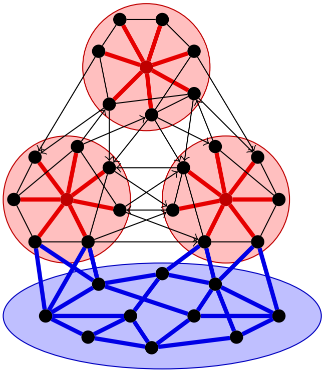
- Orient edges in **arbitrary** way



Maintaining Spanner II

More fine-grained approach:

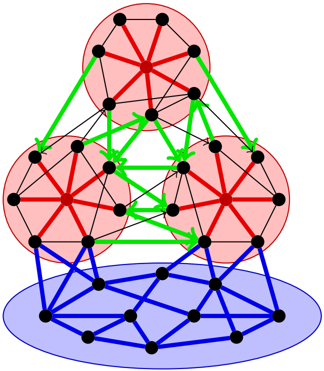
- Orient edges in **arbitrary** way
- For every clustered node v and every other cluster C, maintain set $In(v, C)$: incoming edges from cluster C to v



Maintaining Spanner II

More fine-grained approach:

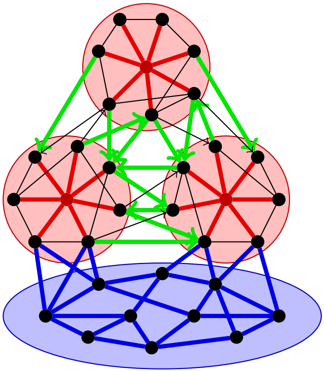
- Orient edges in **arbitrary** way
- For every clustered node v and every other cluster C, maintain set $In(v, C)$: incoming edges from cluster C to v
- Keep **one** entry of $In(v, C)$ in spanner



Maintaining Spanner II

More fine-grained approach:

- Orient edges in **arbitrary** way
- For every clustered node v and every other cluster C, maintain set $In(v, C)$: incoming edges from cluster C to v
- Keep **one** entry of $In(v, C)$ in spanner
- No connection between clusters lost! For inter-cluster edge, one endpoint responsible to connect clusters



Maintaining Spanner II

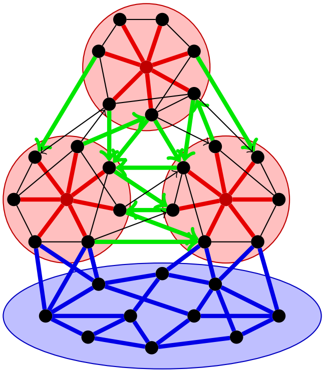
More fine-grained approach:

- Orient edges in **arbitrary** way
- For every clustered node v and every other cluster C, maintain set $In(v, C)$: incoming edges from cluster C to v
- Keep **one** entry of $In(v, C)$ in spanner
- No connection between clusters lost! For inter-cluster edge, one endpoint responsible to connect clusters
- Whenever node u changes from cluster C to cluster C':

  For every outgoing edge (u, v) of v

  Remove u from $N(v, i)$

  Add u to $N(v, j)$



Maintaining Spanner II

More fine-grained approach:

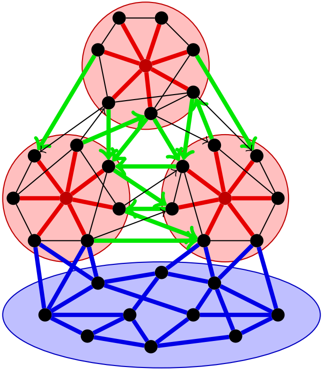
- Orient edges in **arbitrary** way
- For every clustered node v and every other cluster C, maintain set $In(v, C)$: incoming edges from cluster C to v
- Keep **one** entry of $In(v, C)$ in spanner
- No connection between clusters lost! For inter-cluster edge, one endpoint responsible to connect clusters
- Whenever node u changes from cluster C to cluster C':

  For every outgoing edge (u, v) of v

  Remove u from $N(v, i)$

  Add u to $N(v, j)$

$\Rightarrow$ Update time: $O(maxoutdeg(\vec{G}) \log n)$



Partitioning Trick

Idea: Partition outgoing edges each node into groups of size s

Partitioning Trick

Idea: Partition outgoing edges each node into groups of size s

undirected graph G

Partitioning Trick

Idea: Partition outgoing edges each node into groups of size s

undirected graph	G
orient edges	$\downarrow$
	$\vec{G}$

Partitioning Trick

Idea: Partition outgoing edges each node into groups of size s

$$
\begin{array}{llllll}
\text{undirected graph} & & & G & & \\
\text{orient edges} & & & \downarrow & & \\
 & & & \vec{G} & & \\
\text{partition into subgraphs} & \swarrow & \swarrow & \dots & \searrow & \searrow \\
 & \vec{G}_1 & \vec{G}_2 & \dots & \vec{G}_{t-1} & \vec{G}_t
\end{array}
$$

Partitioning Trick

Idea: Partition outgoing edges each node into groups of size s

$$
\begin{array}{llllll}
\text{undirected graph} & & & G & & \\
\text{orient edges} & & & \downarrow & & \\
 & & & \vec{G} & & \\
\text{partition into subgraphs} & \swarrow & \swarrow & \dots & \searrow & \searrow \\
 & \vec{G}_1 & \vec{G}_2 & \dots & \vec{G}_{t-1} & \vec{G}_t \\
\text{maintain sub-spanners} & \downarrow & \downarrow & \dots & \downarrow & \downarrow \\
 & \vec{H}_1 & \vec{H}_2 & \dots & \vec{H}_{t-1} & \vec{H}_t
\end{array}
$$

Partitioning Trick

Idea: Partition outgoing edges each node into groups of size s

$$\begin{array}{lccccc}
\text{undirected graph} & & & G & & \\
\text{orient edges} & & & \downarrow & & \\
 & & & \vec{G} & & \\
\text{partition into subgraphs} & \swarrow & \swarrow & \dots & \searrow & \searrow \\
 & \vec{G}_1 & \vec{G}_2 & \dots & \vec{G}_{t-1} & \vec{G}_t \\
\text{maintain sub-spanners} & \downarrow & \downarrow & \dots & \downarrow & \downarrow \\
 & \vec{H}_1 & \vec{H}_2 & \dots & \vec{H}_{t-1} & \vec{H}_t \\
\text{take union} & \searrow & \searrow & \dots & \swarrow & \swarrow \\
\text{spanner} & & & H & &
\end{array}$$

Partitioning Trick

Idea: Partition outgoing edges each node into groups of size s

undirected graph			G		
orient edges			$\downarrow$		
			$\vec{G}$		
partition into subgraphs	$\swarrow$	$\swarrow$	$\dots$	$\searrow$	$\searrow$
	$\vec{G}_1$	$\vec{G}_2$	$\dots$	$\vec{G}_{t-1}$	$\vec{G}_t$
maintain sub-spanners	$\downarrow$	$\downarrow$	$\dots$	$\downarrow$	$\downarrow$
	$\vec{H}_1$	$\vec{H}_2$	$\dots$	$\vec{H}_{t-1}$	$\vec{H}_t$
take union	$\searrow$	$\searrow$	$\dots$	$\swarrow$	$\swarrow$
spanner			H		

Key observation

Each edge update has to be performed in only **one** subgraph

Update time: $O(maxoutdeg(\vec{G}_i)) = O(s)$

Partitioning Trick

Idea: Partition outgoing edges each node into groups of size s

$$
\begin{array}{llllll}
\text{undirected graph} & & & G & & \\
\text{orient edges} & & & \downarrow & & \\
 & & & \vec{G} & & \\
\text{partition into subgraphs} & \swarrow & \swarrow & \dots & \searrow & \searrow \\
 & \vec{G}_1 & \vec{G}_2 & \dots & \vec{G}_{t-1} & \vec{G}_t \\
\text{maintain sub-spanners} & \downarrow & \downarrow & \dots & \downarrow & \downarrow \\
 & \vec{H}_1 & \vec{H}_2 & \dots & \vec{H}_{t-1} & \vec{H}_t \\
\text{take union} & \searrow & \searrow & \dots & \swarrow & \swarrow \\
\text{spanner} & & & H & &
\end{array}
$$

Key observation

Each edge update has to be performed in only **one** subgraph

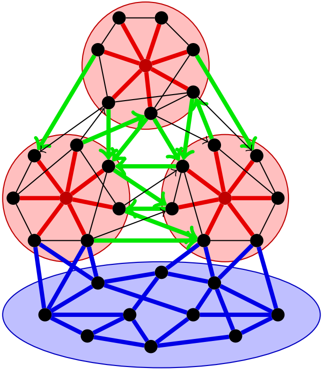
Update time: $O(maxoutdeg(\vec{G}_i)) = O(s)$

Size of spanner: $O(t|H_i|) = O(tn^{1+1/2} \log n) = O(n^{2+1/2}/s)$

Smaller Spanner Size

Hierarchical approach:

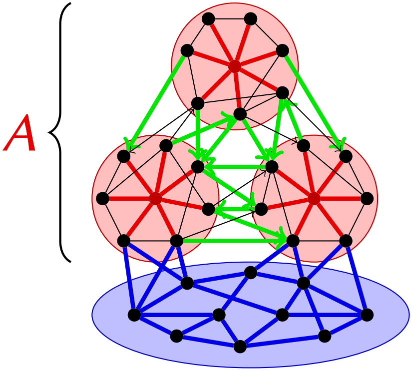
- Clustering with $O((n \log n)/d)$ centers



Smaller Spanner Size

Hierarchical approach:

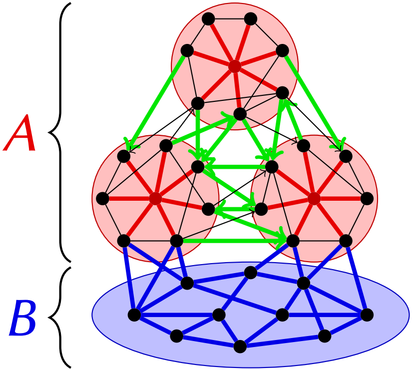
- Clustering with $O((n \log n)/d)$ centers
- A: Edges between clustered nodes



Smaller Spanner Size

Hierarchical approach:

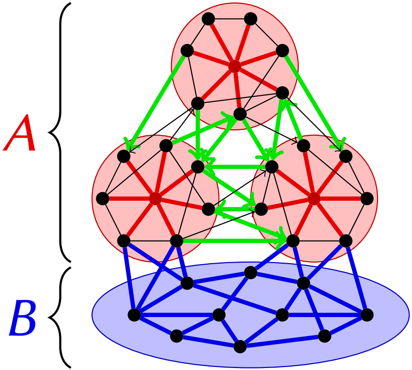
- Clustering with $O((n \log n)/d)$ centers
- A: Edges between clustered nodes
- B: Edges incident to unclustered nodes



Smaller Spanner Size

Hierarchical approach:

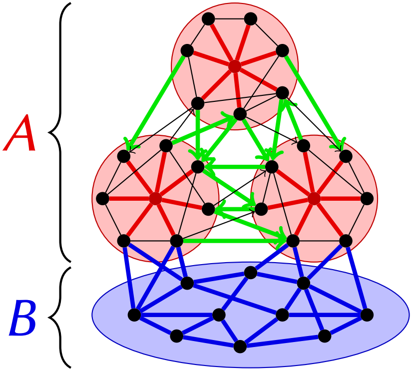
- Clustering with $O((n \log n)/d)$ centers
- A: Edges between clustered nodes
- B: Edges incident to unclustered nodes
- $|A| \leq O((n^2 \log n)/d)$
- Every node in B has degree $\leq d$



Smaller Spanner Size

Hierarchical approach:

- Clustering with $O((n \log n)/d)$ centers
- A: Edges between clustered nodes
- B: Edges incident to unclustered nodes
- $|A| \leq O((n^2 \log n)/d)$
- Every node in B has degree $\leq d$
- Apply spanner algorithm on B
  Update Time: $O(d \log n)$

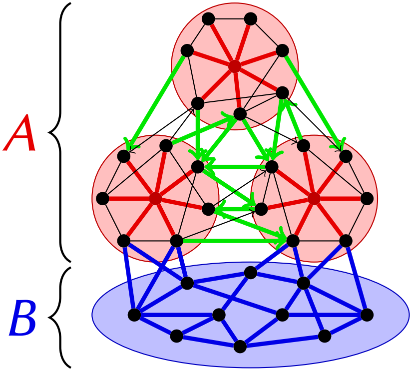


Observation: With every update in G, at most 4 edges are added to or removed from in H

Smaller Spanner Size

Hierarchical approach:

- Clustering with $O((n \log n)/d)$ centers
- A: Edges between clustered nodes
- B: Edges incident to unclustered nodes
- $|A| \le O((n^2 \log n)/d)$
- Every node in B has degree $\le d$
- Apply spanner algorithm on B
  Update Time: $O(d \log n)$



Observation: With every update in G, at most 4 edges are added to or removed from in H

- **Every** node has edges to its **first** d neighbors in spanner
- When node becomes **unclustered**, incident edges already contained

Full Algorithm

undirected graph G

Full Algorithm

undirected graph $\quad G$

orient edges $\quad \downarrow$

$\vec{G}$

Full Algorithm

undirected graph	G
orient edges	$\downarrow$
	$\vec{G}$
partition into subgraphs	$\swarrow \quad \dots \quad \searrow$
	$\vec{G}_1 \qquad \dots \qquad \vec{G}_t$

Full Algorithm

$$\begin{array}{llllllll}
\text{undirected graph} & & & & G & & & \\
\text{orient edges} & & & & \downarrow & & & \\
 & & & & \vec{G} & & & \\
\text{partition into subgraphs} & & & \swarrow & \dots & \searrow & & \\
 & & \vec{G}_1 & & \dots & & \vec{G}_t & \\
\text{maintain partitioned sub-spanners} & \swarrow & & \searrow & \dots & \swarrow & & \searrow \\
 & A_1 & & B_1 & \dots & A_t & & B_t
\end{array}$$

Full Algorithm

undirected graph	G
orient edges	$\downarrow$
	$\vec{G}$
partition into subgraphs	$\swarrow \quad \dots \quad \searrow$
	$\vec{G}_1 \qquad \dots \qquad \vec{G}_t$
maintain partitioned sub-spanners	$\swarrow \qquad \searrow \quad \dots \quad \swarrow \qquad \searrow$
	$A_1 \qquad B_1 \quad \dots \quad A_t \qquad B_t$
union of unclustered parts	$\searrow \quad \dots \quad \swarrow$
	B

Full Algorithm

undirected graph	G
orient edges	$\downarrow$
	$\vec{G}$
partition into subgraphs	$\swarrow \quad \dots \quad \searrow$
	$\vec{G}_1 \quad \dots \quad \vec{G}_t$
maintain partitioned sub-spanners	$\swarrow \quad \searrow \quad \dots \quad \swarrow \quad \searrow$
	$A_1 \quad B_1 \quad \dots \quad A_t \quad B_t$
union of unclustered parts	$\searrow \quad \dots \quad \swarrow$
	B
maintain spanner	$\downarrow$
	H'

Full Algorithm

undirected graph				G			
orient edges				$\downarrow$			
				$\vec{G}$			
partition into subgraphs			$\swarrow$	$\dots$	$\searrow$		
		$\vec{G}_1$		$\dots$		$\vec{G}_t$	
maintain partitioned sub-spanners	$\swarrow$		$\searrow$	$\dots$	$\swarrow$		$\searrow$
	A_1		B_1	$\dots$	A_t		B_t
union of unclustered parts			$\searrow$	$\dots$		$\swarrow$	
				B			
maintain spanner				$\downarrow$			
				H'			

Final spanner: $H = A_1 \cup \cdots \cup A_t \cup H'$

Full Algorithm

$$\begin{array}{lccccccc}
\text{undirected graph} & & & & G & & & \\
\text{orient edges} & & & & \downarrow & & & \\
 & & & & \vec{G} & & & \\
\text{partition into subgraphs} & & & \swarrow & \dots & \searrow & & \\
 & & \vec{G}_1 & & \dots & & \vec{G}_t & \\
\text{maintain partitioned sub-spanners} & \swarrow & & \searrow & \dots & \swarrow & & \searrow \\
 & A_1 & & B_1 & \dots & A_t & & B_t \\
\text{union of unclustered parts} & & & \searrow & \dots & & \swarrow & \\
 & & & & B & & & \\
\text{maintain spanner} & & & & \downarrow & & & \\
 & & & & H' & & &
\end{array}$$

Final spanner: $H = A_1 \cup \cdots \cup A_t \cup H'$

$s = n^{5/6}$, $d = n^{2/3}$, logarithms omitted

Full Algorithm

$$\begin{array}{llllllll}
\text{undirected graph} & & & & G & & & \\
\text{orient edges} & & & & \downarrow & & & \\
 & & & & \vec{G} & & & \\
\text{partition into subgraphs} & & & \swarrow & \dots & \searrow & & \\
 & & \vec{G}_1 & & \dots & & \vec{G}_t & \\
\text{maintain partitioned sub-spanners} & \swarrow & & \searrow & \dots & \swarrow & & \searrow \\
 & A_1 & & B_1 & \dots & A_t & & B_t \\
\text{union of unclustered parts} & & & \searrow & \dots & & \swarrow & \\
 & & & & B & & & \\
\text{maintain spanner} & & & & \downarrow & & & \\
 & & & & H' & & &
\end{array}$$

Final spanner: $H = A_1 \cup \cdots \cup A_t \cup H'$

Update time: $O(s + td) = O(s + nd/s) = O(n^{5/6})$

$s = n^{5/6}$, $d = n^{2/3}$, logarithms omitted

Full Algorithm

$$\begin{array}{lccccccc}
\text{undirected graph} & & & & G & & & \\
\text{orient edges} & & & & \downarrow & & & \\
 & & & & \vec{G} & & & \\
\text{partition into subgraphs} & & & \swarrow & \dots & \searrow & & \\
 & & \vec{G}_1 & & \dots & & \vec{G}_t & \\
\text{maintain partitioned sub-spanners} & \swarrow & & \searrow & \dots & \swarrow & & \searrow \\
 & A_1 & & B_1 & \dots & A_t & & B_t \\
\text{union of unclustered parts} & & & \searrow & \dots & & \swarrow & \\
 & & & & B & & & \\
\text{maintain spanner} & & & & \downarrow & & & \\
 & & & & H' & & &
\end{array}$$

Final spanner: $H = A_1 \cup \cdots \cup A_t \cup H'$

Update time: $O(s + td) = O(s + nd/s) = O(n^{5/6})$

Size of spanner: $O(t \cdot n^2/d + n^{1+1/2}) = O(n^3/(sd) + n^{1+1/2}) = O(n^{1+1/2})$

$s = n^{5/6}$, $d = n^{2/3}$, logarithms omitted

Conclusion

Summary:

- Main idea: Orienting and partitioning edges
- Careful hierarchy unleashes full potential

Conclusion

Summary:

- Main idea: Orienting and partitioning edges
- Careful hierarchy unleashes full potential
- 3-spanner: $O(n^{3/4} \log^4 n)$ update time
- 5-spanner: $O(n^{5/9} \log^4 n)$ update time

Conclusion

Summary:

- Main idea: Orienting and partitioning edges
- Careful hierarchy unleashes full potential
- 3-spanner: $O(n^{3/4} \log^4 n)$ update time
- 5-spanner: $O(n^{5/9} \log^4 n)$ update time

Open Problems:

- Emerging barrier of $\sqrt{n}$: lower bound?
- Worst-case update time for larger stretches
- Sublinear deterministic algorithms

Questions?