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
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 $\left(2k - 1\right)$-spanner of size $n^{1/k} + 1$ ($k \geq 2$)

[Folklore]
Essentially tight if girth conjecture is true [Erdős]
Spanners

Definition

A spanner of stretch $k$ is a subgraph $H$ of $G$ such that, for all pairs of nodes $u$ and $v$, $\text{dist}_H(u, v) \leq k \cdot \text{dist}_G(u, v)$.

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

Essentially tight if girth conjecture is true [Erdős].
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].
**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 $G$

Dynamic algorithm

spanner $H$
Dynamic Problem

undirected $G$

adversary inserts and deletes edges

Dynamic algorithm

spanner $H$
Dynamic Problem

undirected $G$

adversary inserts and deletes edges

Dynamic algorithm

spanner $H$

algorithm adds and removes edges

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

undirected $G$

adversary inserts and deletes edges

Dynamic algorithm

spanner $H$

algorithm adds and removes edges

**Goal:** Maintain edges of spanner $H$ with small update time after edge insertion/deletion in $G$
### Our Results and Related Work

#### Amortized bounds:

<table>
<thead>
<tr>
<th>stretch</th>
<th>size</th>
<th>time</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>(O\left(n^{\frac{1}{2}} + \frac{1}{2}\right))</td>
<td>(O\left(n\right))</td>
<td>[Ausiello et al.]</td>
</tr>
<tr>
<td>(k - 1)</td>
<td>(O\left(n^{\frac{1}{2}} + \frac{1}{k}\right))</td>
<td>(k\log n)</td>
<td>(O\left(k^2 \log 2n\right))</td>
</tr>
<tr>
<td>(k - 1)</td>
<td>(O\left(n^{\frac{1}{2}} + \frac{1}{k}\right))</td>
<td>(\frac{1}{k} - \frac{1}{k})</td>
<td>(O\left(mn^{\frac{1}{2}} + \frac{1}{k}\log\frac{1}{k}\right))</td>
</tr>
</tbody>
</table>

Our result

\[\Rightarrow\]

We give first sublinear worst-case bounds 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 \(O\left(n^{\frac{5}{6}}\right)\)
## Our Results and Related Work

### Amortized bounds:

<table>
<thead>
<tr>
<th>stretch</th>
<th>size</th>
<th>time</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>$O(n^{1+1/2})$</td>
<td>$O(n)$</td>
<td>[Ausiello et al.]</td>
</tr>
<tr>
<td>5</td>
<td>$O(n^{1+1/3})$</td>
<td>$O(n)$</td>
<td>[Ausiello et al.]</td>
</tr>
</tbody>
</table>

⇒ 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 $O\left(\frac{n^5}{9}\right)$.
## Our Results and Related Work

### Amortized bounds:

<table>
<thead>
<tr>
<th>stretch</th>
<th>size</th>
<th>time</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>$O(n^{1+1/2})$</td>
<td>$O(n)$</td>
<td>[Ausiello et al.]</td>
</tr>
<tr>
<td>5</td>
<td>$O(n^{1+1/3})$</td>
<td>$O(n)$</td>
<td>[Ausiello et al.]</td>
</tr>
<tr>
<td>$2k - 1$</td>
<td>$O(n^{1+1/k}k \log n)$</td>
<td>$O(k^2 \log^2 n)$</td>
<td>[Baswana et al.]</td>
</tr>
<tr>
<td>$2k - 1$</td>
<td>$O(n^{1+1/k}k^8 \log^2 n)$</td>
<td>$O(7^k)$</td>
<td>[Baswana et al.]</td>
</tr>
</tbody>
</table>
Our Results and Related Work

Amortized bounds:

<table>
<thead>
<tr>
<th>stretch</th>
<th>size</th>
<th>time</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>$O(n^{1+1/2})$</td>
<td>$O(n)$</td>
<td>[Ausiello et al.]</td>
</tr>
<tr>
<td>5</td>
<td>$O(n^{1+1/3})$</td>
<td>$O(n)$</td>
<td>[Ausiello et al.]</td>
</tr>
<tr>
<td>$2k - 1$</td>
<td>$O(n^{1+1/k} k \log n)$</td>
<td>$O(k^2 \log^2 n)$</td>
<td>[Baswana et al.]</td>
</tr>
<tr>
<td>$2k - 1$</td>
<td>$O(n^{1+1/k} k^8 \log^2 n)$</td>
<td>$O(7^k)$</td>
<td>[Baswana et al.]</td>
</tr>
</tbody>
</table>

Worst-case bounds:
## Our Results and Related Work

### Amortized bounds:

<table>
<thead>
<tr>
<th>stretch</th>
<th>size</th>
<th>time</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>$O(n^{1+1/2})$</td>
<td>$O(n)$</td>
<td>[Ausiello et al.]</td>
</tr>
<tr>
<td>5</td>
<td>$O(n^{1+1/3})$</td>
<td>$O(n)$</td>
<td>[Ausiello et al.]</td>
</tr>
<tr>
<td>$2k - 1$</td>
<td>$O(n^{1+1/k}k \log n)$</td>
<td>$O(k^2 \log^2 n)$</td>
<td>[Baswana et al.]</td>
</tr>
<tr>
<td>$2k - 1$</td>
<td>$O(n^{1+1/k}k^8 \log^2 n)$</td>
<td>$O(7^k)$</td>
<td>[Baswana et al.]</td>
</tr>
</tbody>
</table>

### Worst-case bounds:

<table>
<thead>
<tr>
<th>stretch</th>
<th>size</th>
<th>time</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2k - 1$</td>
<td>$O(n^{1+1/k}k \log^{1-1/k} n)$</td>
<td>$O(mn^{-1/k}\log^{1/k} n)$</td>
<td>[Elkin]</td>
</tr>
</tbody>
</table>
Our Results and Related Work

**Amortized bounds:**

<table>
<thead>
<tr>
<th>stretch</th>
<th>size</th>
<th>time</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>$O(n^{1+1/2})$</td>
<td>$O(n)$</td>
<td>[Ausiello et al.]</td>
</tr>
<tr>
<td>5</td>
<td>$O(n^{1+1/3})$</td>
<td>$O(n)$</td>
<td>[Ausiello et al.]</td>
</tr>
<tr>
<td>$2k - 1$</td>
<td>$O(n^{1+1/k}k \log n)$</td>
<td>$O(k^2 \log^2 n)$</td>
<td>[Baswana et al.]</td>
</tr>
<tr>
<td>$2k - 1$</td>
<td>$O(n^{1+1/k}k^8 \log^2 n)$</td>
<td>$O(7^k)$</td>
<td>[Baswana et al.]</td>
</tr>
</tbody>
</table>

**Worst-case bounds:**

<table>
<thead>
<tr>
<th>stretch</th>
<th>size</th>
<th>time</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2k - 1$</td>
<td>$O(n^{1+1/k}k \log^{1-1/k} n)$</td>
<td>$O(mn^{-1/k} \log^{1/k} n)$</td>
<td>[Elkin]</td>
</tr>
<tr>
<td>3</td>
<td>$O(n^{1+1/2} \log^{1/2} n \log \log n)$</td>
<td>$O(n^{3/4} \log^4 n)$</td>
<td>Our result</td>
</tr>
<tr>
<td>5</td>
<td>$O(n^{1+1/3} \log^{2/3} n \log \log n)$</td>
<td>$O(n^{5/9} \log^4 n)$</td>
<td>Our result</td>
</tr>
</tbody>
</table>
Our Results and Related Work

### Amortized bounds:

<table>
<thead>
<tr>
<th>stretch</th>
<th>size</th>
<th>time</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>$O(n^{1+1/2})$</td>
<td>$O(n)$</td>
<td>[Ausiello et al.]</td>
</tr>
<tr>
<td>5</td>
<td>$O(n^{1+1/3})$</td>
<td>$O(n)$</td>
<td>[Ausiello et al.]</td>
</tr>
<tr>
<td>2$k$ − 1</td>
<td>$O(n^{1+1/k}k \log n)$</td>
<td>$O(k^2 \log^2 n)$</td>
<td>[Baswana et al.]</td>
</tr>
<tr>
<td>2$k$ − 1</td>
<td>$O(n^{1+1/k}k^8 \log^2 n)$</td>
<td>$O(7^k)$</td>
<td>[Baswana et al.]</td>
</tr>
</tbody>
</table>

### Worst-case bounds:

<table>
<thead>
<tr>
<th>stretch</th>
<th>size</th>
<th>time</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>2$k$ − 1</td>
<td>$O(n^{1+1/k}k \log^{1-1/k} n)$</td>
<td>$O(mn^{-1/k} \log^{1/k} n)$</td>
<td>[Elkin]</td>
</tr>
<tr>
<td>3</td>
<td>$O(n^{1+1/2} \log^{1/2} n \log \log n)$</td>
<td>$O(n^{3/4} \log^4 n)$</td>
<td>Our result</td>
</tr>
<tr>
<td>5</td>
<td>$O(n^{1+1/3} \log^{2/3} n \log \log n)$</td>
<td>$O(n^{5/9} \log^4 n)$</td>
<td>Our result</td>
</tr>
</tbody>
</table>

⇒ We give **first** sublinear worst-case bounds
Our Results and Related Work

**Amortized bounds:**

<table>
<thead>
<tr>
<th>stretch</th>
<th>size</th>
<th>time</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>(O(n^{1+1/2}))</td>
<td>(O(n))</td>
<td>[Ausiello et al.]</td>
</tr>
<tr>
<td>5</td>
<td>(O(n^{1+1/3}))</td>
<td>(O(n))</td>
<td>[Ausiello et al.]</td>
</tr>
<tr>
<td>2(k - 1)</td>
<td>(O(n^{1+1/k}k \log n))</td>
<td>(O(k^2 \log^2 n))</td>
<td>[Baswana et al.]</td>
</tr>
<tr>
<td>2(k - 1)</td>
<td>(O(n^{1+1/k}k^8 \log^2 n))</td>
<td>(O(7^k))</td>
<td>[Baswana et al.]</td>
</tr>
</tbody>
</table>

**Worst-case bounds:**

<table>
<thead>
<tr>
<th>stretch</th>
<th>size</th>
<th>time</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>2(k - 1)</td>
<td>(O(n^{1+1/k}k \log^{1-1/k} n))</td>
<td>(O(mn^{-1/k} \log^{1/k} n))</td>
<td>[Elkin]</td>
</tr>
<tr>
<td>3</td>
<td>(O(n^{1+1/2} \log^{1/2} n \log \log n))</td>
<td>(O(n^{3/4} \log^4 n))</td>
<td>Our result</td>
</tr>
<tr>
<td>5</td>
<td>(O(n^{1+1/3} \log^{2/3} n \log \log n))</td>
<td>(O(n^{5/9} \log^4 n))</td>
<td>Our result</td>
</tr>
</tbody>
</table>

\(\Rightarrow\) We give **first** sublinear worst-case bounds

Guarantees with high probability against oblivious adversary
Our Results and Related Work

Amortized bounds:

<table>
<thead>
<tr>
<th>stretch</th>
<th>size</th>
<th>time</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>$O(n^{1+1/2})$</td>
<td>$O(n)$</td>
<td>[Ausiello et al.]</td>
</tr>
<tr>
<td>5</td>
<td>$O(n^{1+1/3})$</td>
<td>$O(n)$</td>
<td>[Ausiello et al.]</td>
</tr>
<tr>
<td>$2k - 1$</td>
<td>$O(n^{1+1/k} k \log n)$</td>
<td>$O(k^2 \log^2 n)$</td>
<td>[Baswana et al.]</td>
</tr>
<tr>
<td>$2k - 1$</td>
<td>$O(n^{1+1/k} k^8 \log^2 n)$</td>
<td>$O(7^k)$</td>
<td>[Baswana et al.]</td>
</tr>
</tbody>
</table>

Worst-case bounds:

<table>
<thead>
<tr>
<th>stretch</th>
<th>size</th>
<th>time</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2k - 1$</td>
<td>$O(n^{1+1/k} k \log^{1-1/k} n)$</td>
<td>$O(mn^{-1/k} \log^{1/k} n)$</td>
<td>[Elkin]</td>
</tr>
<tr>
<td>3</td>
<td>$O(n^{1+1/2} \log^{1/2} n \log \log n)$</td>
<td>$O(n^{3/4} \log^4 n)$</td>
<td>Our result</td>
</tr>
<tr>
<td>5</td>
<td>$O(n^{1+1/3} \log^{2/3} n \log \log n)$</td>
<td>$O(n^{5/9} \log^4 n)$</td>
<td>Our result</td>
</tr>
</tbody>
</table>

⇒ 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:

<table>
<thead>
<tr>
<th>stretch</th>
<th>size</th>
<th>time</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>$O(n^{1+1/2})$</td>
<td>$O(n)$</td>
<td>[Ausiello et al.]</td>
</tr>
<tr>
<td>5</td>
<td>$O(n^{1+1/3})$</td>
<td>$O(n)$</td>
<td>[Ausiello et al.]</td>
</tr>
<tr>
<td>$2k - 1$</td>
<td>$O(n^{1+1/k} k \log n)$</td>
<td>$O(k^2 \log^2 n)$</td>
<td>[Baswana et al.]</td>
</tr>
<tr>
<td>$2k - 1$</td>
<td>$O(n^{1+1/k} k^8 \log^2 n)$</td>
<td>$O(7^k)$</td>
<td>[Baswana et al.]</td>
</tr>
</tbody>
</table>

#### Worst-case bounds:

<table>
<thead>
<tr>
<th>stretch</th>
<th>size</th>
<th>time</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2k - 1$</td>
<td>$O(n^{1+1/k} k \log^{1-1/k} n)$</td>
<td>$O(mn^{-1/k} \log^{1/k} n)$</td>
<td>[Elkin]</td>
</tr>
<tr>
<td>3</td>
<td>$O(n^{1+1/2} \log^{1/2} n \log \log n)$</td>
<td>$O(n^{3/4} \log^4 n)$</td>
<td>Our result</td>
</tr>
<tr>
<td>5</td>
<td>$O(n^{1+1/3} \log^{2/3} n \log \log n)$</td>
<td>$O(n^{5/9} \log^4 n)$</td>
<td>Our result</td>
</tr>
</tbody>
</table>

$\Rightarrow$ 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

1. Pick $O(\sqrt{n \log n})$ centers at random.
2. Form clusters: Connect every node to one of its neighboring centers.⇒ Unclustered nodes have at most $\sqrt{n}$ neighbors with high probability.
3. At any time, spanner consists of the 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 has stretch 3 and size $O(n^{1+1/2} \log n)$ whp (standard proof).
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

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

At any time, spanner consists of the 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 has stretch 3 and size $O(n^{1/2} \log n)$ whp (standard proof)
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

⇒ 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
   ⇒ 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
   ⇒ 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 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
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 $N(v, C')$
⇒ Update time: $O(\text{maxdeg}(G) \log n)$
Maintaining Spanner II

More fine-grained approach:

For every clustered node \( v \) and every other cluster \( C \), maintain set \( \text{In}(v, C) \): incoming edges from cluster \( C \) to \( v \)

Keep one entry of \( \text{In}(v, C) \) 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 \( N(v, j) \)

⇒ Update time: \( O(\max_{\vec{G}}(\text{outdeg}(\vec{G})) \log n) \)
Maintaining Spanner II

More fine-grained approach:

- Orient edges in **arbitrary** way

For every clustered node $v$ and every other cluster $C$, maintain set $\text{In}(v, C)$: incoming edges from cluster $C$ to $v$

Keep one entry of $\text{In}(v, C)$ 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 $\text{N}(v, i)$
  - Add $u$ to $\text{N}(v, j)$

$\Rightarrow$ Update time: $O(\text{maxoutdeg}(\vec{G}) \log n)$
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 \( \text{In}(v, C) \): incoming edges from cluster \( C \) to \( v \)
- Keep **one** entry of \( \text{In}(v, C) \) 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: $O(maxoutdeg(\tilde{G}) \log n)$
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 $\vec{G}$

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: $O(t|H_i|) = O(tn_1 + 1/2 \log n) = O(n_2^{1/2} + 1/2/s)$
Partitioning Trick

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

- Undirected graph $G$
- Orient edges
- Partition into subgraphs $\vec{G}$
- $\vec{G}_1, \vec{G}_2, \ldots, \vec{G}_{t-1}, \vec{G}_t$

Update time: $O(\max\text{outdeg}(\vec{G}_i)) = O(s)$

Size of spanner: $O(t|\vec{H}_i|) = O(tn^{1/2} + 1/2 \log n) = O(n^{1/2} + 1/2/s)$
Partitioning Trick

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

undirected graph $G$

orient edges $\downarrow$

partition into subgraphs $\leftarrow \leftarrow \ldots \rightarrow \rightarrow$

$\tilde{G}_1 \quad \tilde{G}_2 \quad \ldots \quad \tilde{G}_{t-1} \quad \tilde{G}_t$

maintain sub-spanners $\downarrow \downarrow \ldots \downarrow \downarrow$

$\tilde{H}_1 \quad \tilde{H}_2 \quad \ldots \quad \tilde{H}_{t-1} \quad \tilde{H}_t$

Key observation

Each edge update has to be performed in only one subgraph

Update time: $O(\max_{\text{outdeg}}(\tilde{G}_i)) = O(s)$

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

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

- **undirected graph** $G$
- **orient edges** $\vec{G}$
- **partition into subgraphs** $\vec{G}_1 \rightarrow \vec{G}_2 \rightarrow \ldots \rightarrow \vec{G}_{t-1} \rightarrow \vec{G}_t$
- **maintain sub-spanners** $\vec{H}_1 \rightarrow \vec{H}_2 \rightarrow \ldots \rightarrow \vec{H}_{t-1} \rightarrow \vec{H}_t$
- **take union** $\vec{H}$

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

**Update time:** $O(\text{max outdeg}(\vec{G}_i)) = O(s)$

**Size of spanner:** $O(\sum |\vec{H}_i|) = O(n^2 + \frac{1}{2s})$
Partitioning Trick

**Idea:** Partition outgoing edges each node into groups of size \( s \)

undirected graph \( G \)
orient edges \( \rightarrow \)
partition into subgraphs \( \rightarrow \)
\( \vec{G}_1 \rightarrow \vec{G}_2 \rightarrow \ldots \rightarrow \vec{G}_{t-1} \rightarrow \vec{G}_t \)
maintain sub-spanners \( \downarrow \downarrow \downarrow \downarrow \)
\( \vec{H}_1 \rightarrow \vec{H}_2 \rightarrow \ldots \rightarrow \vec{H}_{t-1} \rightarrow \vec{H}_t \)
take union \( \rightarrow \rightarrow \rightarrow \)
spanner \( \vec{H} \)

**Key observation**

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

**Update time:** \( O(\text{maxoutdeg}(\vec{G}_i)) = O(s) \)
Partitioning Trick

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

- undirected graph $G$
- orient edges $\vec{G}$
- partition into subgraphs $\vec{G}_1$, $\vec{G}_2$, ..., $\vec{G}_{t-1}$, $\vec{G}_t$
- maintain sub-spanners $\vec{H}_1$, $\vec{H}_2$, ..., $\vec{H}_{t-1}$, $\vec{H}_t$
- take union $\vec{H}$

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

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

**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

Let $A$ be the set of edges between clustered nodes and $B$ be the set of 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 $H$.

Every node has edges to its first $d$ neighbors in the spanner. When a node becomes unclustered, incident edges already contained.
Smaller Spanner Size

Hierarchical approach:

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

Every node has edges to its first $d$ neighbors in spanner
When node becomes unclustered, incident edges already contained
Smaller Spanner Size

Hierarchical approach:

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

Observation: With every update in $G$, at most 4 edges are added to or removed from $H$. Every node has edges to its first $d$ neighbors in spanner. When node becomes unclustered, incident edges already contained.
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 $H$
Smaller Spanner Size

Hierarchical approach:

- Clustering with \( O((n \log n)/d) \) centers
- \( A \): Edges between clustered nodes
- \( B \): Edges incident to unclustered nodes
- \( |A| \leq O((n^2 \log n)/d) \)
- 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 \( H \)

- 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 
orient edges

\[ G \]

\[ \Downarrow \]

\[ \vec{G} \]
Full Algorithm

- undirected graph \( G \)
- orient edges \( \vec{G} \)
- partition into subgraphs \( \vec{G}_1, \ldots, \vec{G}_t \)

Final spanner: \( H = A_1 \cup \cdots \cup A_t \cup H' \)

Update time: \( O(s + td) = O(s + n^5/d^6) = O(n^{3/(sd^2)} + n + 1) \)

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

undirected graph $G$
orient edges $\vec{G}$
partition into subgraphs $\vec{G}_1 \ldots \vec{G}_t$
maintain partitioned sub-spanners $A_1 \cup \ldots \cup A_t$ $B_1 \ldots B_t$

Union of unclustered parts $B$
maintain spanner $H'$

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

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
Full Algorithm

undirected graph \( G \)
orient edges \( \vec{G} \)
partition into subgraphs
\( \vec{G}_1 \) ... \( \vec{G}_t \)
maintain partitioned sub-spanners
\( A_1 \) ... \( A_t \)
union of unclustered parts
\( B \)
Final spanner:
\[
\text{Final spanner: } H = A_1 \cup \cdots \cup A_t \cup H'
\]
Update time:
\[
O((s + td)) = O((n^5/6 + nd/s)) = O(n^{11/2})
\]
Size of spanner:
\[
O(t \cdot n^2/d + n^{11/2}) = O(n^{11/2})
\]
Full Algorithm

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

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

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

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

undirected graph
orient edges

partition into subgraphs

maintain partitioned sub-spanners

union of unclustered parts

maintain spanner

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

undirected graph $G$
orient edges $\vec{G}$
partition into subgraphs $\vec{G}_1 \ldots \vec{G}_t$
maintain partitioned sub-spanners $A_1 \ldots A_t$
union of unclustered parts $B$
maintain spanner $B$

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

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

undirected graph \( G \)
orient edges \( \rightarrow \)
partition into subgraphs\( \rightarrow \)
maintain partitioned sub-spanners\( \rightarrow \)
union of unclustered parts\( \rightarrow \)
maintain spanner

\[ \text{Final spanner: } H = A_1 \cup \cdots \cup A_t \cup H' \]

\[ \text{Update time: } O(s + td) = O(s + nd/s) = O(n^{5/6}) \]

\[ s = n^{5/6}, \ d = n^{2/3}, \ \text{logarithms omitted} \]
Full Algorithm

undirected graph \( G \)
orient edges \( \vec{G} \)
partition into subgraphs \( \vec{G}_1, \ldots, \vec{G}_t \)
 maintain partitioned sub-spanners \( A_1, B_1, \ldots, A_t, B_t \)
union of unclustered parts \( B \)
maintain spanner \( H' \)

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

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