Towards Optimal Dynamic Graph Compression

Sebastian Krinninger

Universität Salzburg

Austrian Computer Science Day 2018
Graphs are Everywhere
Graphs are Everywhere
Graphs are Everywhere
Graphs are Everywhere
Graph Compression
Graph Compression
Graph Compression

Goal: Semantic Compression

Subgraph for algorithmic applications
Graph Compression

Goal: Semantic Compression
Graph Compression

Goal: Semantic Compression

Subgraph for algorithmic applications
Too Good to be True?

“There ain't no such thing as a free lunch. . . except for ACSD 2018.

Thanks Christoph!
Too Good to be True?

“There ain’t no such thing as a free lunch.”
Too Good to be True?

“There ain’t no such thing as a free lunch.”

…except for ACSD 2018.
Too Good to be True?

“There ain’t no such thing as a free lunch.”

…except for ACSD 2018.
Thanks Christoph!
Lossy Compression

When are two graphs approximately the same?

→ Problem-specific measures
Lossy Compression

→ Compression at cost of approximation

When are two graphs approximately the same?

→ Problem-specific measures
Lossy Compression

→ Compression at cost of approximation

When are two graphs approximately the same?

→ Problem-specific measures
Lossy Compression

When are two graphs approximately the same?

→ Problem-specific measures

→ Compression at cost of approximation
Lossy Compression

Cannot reconstruct original graph after compression

→ Compression at cost of approximation
Lossy Compression

Cannot reconstruct original graph after compression
→ Compression at cost of approximation

When are two graphs approximately the same?
→ Problem-specific measures
Our World is not Static
Our World is not Static
Our World is not Static
Our World is not Static
Our World is not Static

**Goal:** Fast recomputation of solution after each insertion/deletion of an edge
Dynamic Graph Compression

Input graph $G$

Algorithm

Compressed graph $H$
Dynamic Graph Compression

Input graph $G$

Algorithm

Compressed graph $H$

adversary inserts and deletes edges
Dynamic Graph Compression

Input graph $G$ 

Algorithm

Compressed graph $H$

adversary inserts and deletes edges
Dynamic Graph Compression

Input graph $G$

Algorithm

Compressed graph $H$

adversary inserts and deletes edges

algorithm adds and removes edges
Let’s take a look under the hood!
Example 1: Distance-Preserving Compression

**Definition**

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

\[
\text{dist}_G(u, v) \leq \text{dist}_H(u, v) \leq t \cdot \text{dist}_G(u, v)
\]

for all pairs of nodes \( u, v \in V \).
Example 1: Distance-Preserving Compression

**Definition**

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

$$\text{dist}_G(u, v) \leq \text{dist}_H(u, v) \leq t \cdot \text{dist}_G(u, v)$$

for all pairs of nodes $u, v \in V$. 
Example 1: Distance-Preserving Compression

**Definition**

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

$$\text{dist}_G(u, v) \leq \text{dist}_H(u, v) \leq t \cdot \text{dist}_G(u, v)$$

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

---

In the diagram, the red edges represent the spanner $H$ with stretch $t$, where $H$ is a subgraph of $G$.
Example 1: Distance-Preserving Compression

**Definition**

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

$$\text{dist}_G(u, v) \leq \text{dist}_H(u, v) \leq t \cdot \text{dist}_G(u, v)$$

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 $O(n^2)$

Isn't this stretch guarantee very weak? In many applications: boosting approach for be/t_ter approximation.
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

Isn't this stretch guarantee very weak? In many applications: boosting approach for better approximation.
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 $O(n^{3/2})$

**Lemma**

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

Isn't this stretch guarantee very weak? In many applications: boosting approach for better approximation
**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 \( O(n^2) \) → input graph
- \( k = 2 \): stretch 3, size \( O(n^{3/2}) \)
- \( \vdots \)
- \( k = \log n \): stretch \( O(\log n) \), size \( O(n) \)

Isn't this stretch guarantee very weak? In many applications: boosting approach for better approximation.
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 $O(n)$

Lemma

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

**Lemma**

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

Isn’t this 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 $O(n^2)$ → input graph
- $k = 2$: stretch 3, size $O(n^{3/2})$
- ...
- $k = \log n$: stretch $O(\log n)$, size $O(n)$

Lemma

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

Isn’t this stretch guarantee very weak?

In many applications: boosting approach for better approximation
Our Spanner Results

Theorem ([Baswana, Sarkar ’08])

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

- with $O(n^{1+1/k}k^8 \log^2 n)$ edges in amortized time $O(7^{k/2})$ per update,
- with $O(n^{1+1/k}k \log n)$ edges in amortized time $O(k^2 \log^2 n)$ per update.
Our Spanner Results

Theorem ([Baswana, Sarkar ’08])

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

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

Amortized time: Time bound holds on average over a sequence of updates.
Our Spanner Results

**Theorem ([Baswana, Sarkar ’08])**

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

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

**Amortized time:** Time bound holds on average over a sequence of updates

**Worst-case time:** Hard upper bound for each update
Our Spanner Results

Theorem ([Baswana, Sarkar ’08])

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

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

Amortized time: Time bound holds on average over a sequence of updates
Worst-case time: Hard upper bound for each update

Theorem ([Bernstein, Henzinger, K submitted])

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

Theorem ([Baswana, Sarkar ’08])

For every $k$, there is a dynamic algorithm that maintains a spanner of stretch $t = 2k - 1$
- with $O(n^{1+1/k}k^8 \log^2 n)$ edges in amortized time $O(7^{k/2})$ per update,
- with $O(n^{1+1/k}k \log n)$ edges in amortized time $O(k^2 \log^2 n)$ per update.

Amortized time: Time bound holds on average over a sequence of updates
Worst-case time: Hard upper bound for each update

Theorem ([Bernstein, Henzinger, K submitted])

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

Theorem ([Goranci, K submitted])

For every $k$, there is a dynamic algorithm that maintains a $(2k - 1)$-spanner with $O(n^{1+1/k} \log n)$ edges in amortized time $O(k \log^2 n)$ per update.
More Succinct Compression

**Question:** How much compression is possible?

Need to preserve connectivity: spanning tree is the limit

Number of edges: \( n - 1 \)

Drawback: Cannot have “hard” stretch guarantee anymore, only average

Theorem ([Goranci, submitted])

There is a dynamic algorithm that maintains a spanning tree of average stretch 
\[ t = n o(1) \]
with amortized time \( O(n^{1/2} + o(1)) \) per update.

Matches stretch of seminal static construction! [Alon/Karp/Peleg/West]
More Succinct Compression

**Question:** How much compression is possible?
Need to preserve connectivity: spanning tree is the limit
More Succinct Compression

**Question**: How much compression is possible?
Need to preserve connectivity: spanning tree is the limit
More Succinct Compression

**Question:** How much compression is possible?

Need to preserve connectivity: spanning tree is the limit

Number of edges: $n - 1$
More Succinct Compression

**Question:** How much compression is possible?
Need to preserve connectivity: spanning tree is the limit

Number of edges: $n - 1$

**Drawback:** Cannot have “hard” stretch guarantee anymore, only average

---

Theorem ([Goranci, K. submitted])

There is a dynamic algorithm that maintains a spanning tree of average stretch $t = n^{o(1)}$ with amortized time $O(n^{1/2} + o(1))$ per update.

Matches stretch of seminal static construction! ([Alon/Karp/Peleg/West])
**More Succinct Compression**

**Question:** How much compression is possible?
Need to preserve connectivity: spanning tree is the limit

Number of edges: $n - 1$

**Drawback:** Cannot have “hard” stretch guarantee anymore, only average

**Theorem ([Goranci, K submitted])**

There is a dynamic algorithm that maintains a spanning tree of average stretch $t = n^{o(1)}$ with amortized time $O(n^{1/2+o(1)})$ per update.
More Succinct Compression

**Question**: How much compression is possible?
Need to preserve connectivity: spanning tree is the limit

Number of edges: \( n - 1 \)

**Drawback**: Cannot have “hard” stretch guarantee anymore, only average

**Theorem ([Goranci, K submitted])**

There is a dynamic algorithm that maintains a spanning tree of average stretch \( t = n^{o(1)} \) with amortized time \( O(n^{1/2+o(1)}) \) per update.

Matches stretch of seminal static construction! [Alon/Karp/Peleg/West]
Example II: Cut-Preserving Compression

**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 \(E[C, V \setminus C]\) crossing the cut have weight

\[
(1 - \epsilon) \cdot w_G(E[C, V \setminus C]) \leq w_H(E[C, V \setminus C]) \leq (1 + \epsilon) \cdot w_G(E[C, V \setminus C])
\]
Example II: Cut-Preserving Compression

Definition ([Benczúr/Karger ’00])

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

\[
(1 - \varepsilon) \cdot w_G(E[C, V \setminus C]) \leq w_H(E[C, V \setminus C]) \leq (1 + \varepsilon) \cdot w_G(E[C, V \setminus C])
\]
Example II: Cut-Preserving Compression

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 \(E[C, V \setminus C]\) crossing the cut have weight

\[
(1 - \epsilon) \cdot w_G(E[C, V \setminus C]) \leq w_H(E[C, V \setminus C]) \leq (1 + \epsilon) \cdot w_G(E[C, V \setminus C])
\]
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 \(E[C, V \setminus C]\) crossing the cut have weight

\[
(1 - \epsilon) \cdot w_G(E[C, V \setminus C]) \leq w_H(E[C, V \setminus C]) \leq (1 + \epsilon) \cdot w_G(E[C, V \setminus C])
\]
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.

Deep Connection 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.

Deep Connection to solving SDD linear systems! [Spielman/Teng ’04]

**Theorem (Abraham, Durfee, Koutis, K, Peng ’16)**

There is a dynamic algorithm for maintaining a spectral 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.

Deep Connection to solving SDD linear systems! [Spielman/Teng ’04]

Theorem (Abraham, Durfee, Koutis, K, Peng ’16)

There is a dynamic algorithm for maintaining a spectral 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
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.*

Deep Connection to solving SDD linear systems! [Spielman/Teng ’04]

**Theorem (Abraham, Durfee, Koutis, K, Peng ’16)**

*There is a dynamic algorithm for maintaining a spectral 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

**Internally uses dynamic spanner with stretch** \( O(\log n) \)
Conclusion

Graph compression

- Mathematically clean framework
Conclusion

Graph compression
- Mathematically clean framework
- Powerful tool in modern algorithm design
Conclusion

Graph compression

- Mathematically clean framework
- Powerful tool in modern algorithm design

My goals:

- Rebuild graph compression results in the dynamic world
- Tighten connection between dynamic graph algorithms and combinatorial/continuous optimization
Conclusion

Graph compression
- Mathematically clean framework
- Powerful tool in modern algorithm design

My goals:
- Rebuild graph compression results in the dynamic world
- Tighten connection between dynamic graph algorithms and combinatorial/continuous optimization

Thank you!
Closing Words