
Slim Graph: Practical Lossy Graph Compression for Approximate Graph Processing, Storage, and Analytics
What are the concrete workloads we care about?
Graph Analytics Fundamental Problems and Algorithms [1, many others]

Genome traversals

Graph traversals (e.g., BFS, SSSP)

Iterative schemes (e.g., PageRank)

Connectivity related (e.g., #connected components)

Optimization problems (e.g., MST, colorings, ...)

Graph mining & learning (e.g., clique listing)

We need lots of hardware resources to store them

Running analytics on large graphs gets slow

What does “huge” mean?
Problems!

Huge size

What does “huge” mean?

Lossless compression incurs expensive decompression and it hits fundamental storage lower bounds [1,2]


> 233 TB
271 billion vertices, 12 trillion edges [1]

[1] M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18
Remove some edges and/or vertices (i.e., **sparsification**)

What if we don’t want full precision?

Run graph analytics workloads on these sparsified graphs

Let’s see a curious motivation...

- **High Accuracy**
- **Faster workloads**
- **Less storage**
JPEG compression level: 1%
File size: 823.4 kB
JPEG compression level: 50%
File size: 130.2 kB
**Slim Graph**: A systematic approach for effective lossy graph compression, to enable *storage reductions & speedups of graph analytics*, with a small accuracy tradeoff.

JPG & MP3 target specific things *(pictures & sound)*

Slim Graph targets specific classes of graph workloads / properties
Slim Graph delivers a simple, intuitive, versatile ...

1. **Abstraction & programming model** for easy development and rapid prototyping of lossy graph compression methods.

2. **Compression method** that preserves different graph properties that are important for the practice of graph processing.

3. **Criterion (criteria?)** to assess the accuracy of lossy graph compression methods.

4. **High-performance and extensible system** for implementing and executing lossy graph compression.

Number of ways [1] to sparsify (compress) a graph with $n$ vertices

$$O \left( 2^{n/2} \right)$$

Central concept is **compression kernels**: small code snippets that remove specified local parts of the graph.

Different kernels enable different compression methods.

Let’s see some examples…

Kernels focus on:
- Vertex
- Edge
- Triangle
- Subgraph

Compilation, parallel execution
Slim Graph: Abstraction & Programming Model

Vertex kernels: removing degree-0 vertices

Connected components (other than single vertices) are preserved

Before compression:
Degree-0 vertices will be removed by vertex kernels

During compression:
Executing vertex kernels

After compression:
Connected components (other than single vertices) preserved
kernel(V v) {
    if(v.deg==0)
        atomic SG.del(v);
}
Slim Graph: Abstraction & Programming Model

Input:

- **Edge kernels**: random uniform sampling
- **Sparse/dense neighborhoods preserved w.h.p.**
- **Relative neighborhood sizes provide information about, e.g., vertex importance**

**Before compression:**
- Example high-degree vertex
- Example low-degree vertex

**During compression:**
- Executing edge kernels
- Overlapping kernel instances

**After compression:**
- Relative sizes of neighborhoods are preserved w.h.p.
random_uniform_kernel(E e) {
    double edge_stays = SG.p;
    if (edge_stays < SG.rand(0, 1))
        atomic SG.del(e);
}
Slim Graph: Abstraction & Programming Model

More kernels
Slim Graph: Abstraction & Programming Model

---

```c
1 /********** Single-edge compression kernels (§ 4.2) **********/
2 spectral_sparsify(E e) { //More details in § 4.2.1
3   double Y = SG.connectivity_spectral_parameter();
4   double edge_stays = min(1.0, Y / min(e.u.deg, e.v.deg));
5   if(edge_stays < SG.rand(0,1)) atomic SG.del(e);
6   else e.weight = 1/edge_stays;

723 /********** Single-vertex compression kernel (§ 4.4) **********/
24 low_degree(Y v) {
25   if(v.deg==0) atomic SG.del(v); }

26 /********** Subgraph compression kernels (§ 4.5) **********/
27 derive_spanner(vector<v> subgraph) { //Details in § 4.5.3
28   //Replace "subgraph" with a spanning tree
29   subgraph = derive_spanning_tree(subgraph);
30   //Leave only one edge going to any other subgraph.
31   vector<set<v>> subgraphs(SG.sgr_cnt);
32   foreach(E e: SG.out_edges(subgraph)) {
33     if(subgraphs[e.v.elem_id].empty()) atomic del(e);
34   }
35   derive_summary(vector<v> cluster) { //Details in § 4.5.4
36     //Create a supervertex "sv" out of a current cluster:
37     Y sv = SG.min_id(cluster);
38     SG.summary.insert(sv); //Insert sv into a summary graph
39     //Select edges to preserve) within a current cluster:
40     vector<E> intra = SG.summary_select(cluster, SG.e);
41     SG.corrections_plus.append(intra);
42     //Iterate over all clusters connected to "cluster":
43     foreach(vector<v> cl: SG.out_clusters(out_edges(cluster))) {
44       //E, vector<v>) (se, inter) = SG.superedge(cluster, cl, SG.e);
45       SG.summary.insert(se);
46       SG.corrections_minus.append(inter);
47     }
48     SG.update_convergence();
```

---

---
Slim Graph: Abstraction & Programming Model

Details in the paper

More kernels
How expressive is the compression kernel abstraction?
How expressive is the compression kernel abstraction?

We investigated over 500 papers to distill the key classes of graph sparsification.
How expressive is the compression kernel abstraction?

Compression kernels: an abstraction that enables expressing fundamental classes of sparsification

Random uniform (and other forms of) **sampling**

Spectral sparsifiers (preserve spectra)

Cut sparsifiers (preserve cuts)

Summarizations (preserve neighborhoods)

Spanners (preserve pairwise distances)

Others

We investigated over 500 papers to distill the key classes of graph sparsification
Slim Graph delivers a simple, intuitive, versatile ...

1. **Abstraction & programming model** for easy development and rapid prototyping of lossy graph compression methods.

2. **Compression method** that preserves different graph properties that are important for the practice of graph processing.

3. **Criterion** to assess the accuracy of lossy graph compression methods.

4. **High-performance and extensible system** for implementing and executing lossy graph compression.

Number of ways [1] to sparsify (compress) a graph with \( n \) vertices

\[ O \left( 2^\left(\frac{n}{2}\right) \right) \]

Slim Graph: A Novel Compression Method "Triangle Reduction"

Each triangle, with a certain selected probability $p$, is "reduced" – some of its parts are removed.

Here, we consider **one edge in a triangle**

As we show later, it preserves different graph properties

---

**Before compression:**

- Minimum spanning tree

**During compression:**

- Overlapping kernels
- Bolded edges are parts of detected triangles
- Kernel instances
- Maximum-weight edges in sampled triangles will be removed

**After compression:**

- MST weight is preserved
Slim Graph: A Novel Compression Method "Triangle Reduction"

Each triangle, with a certain selected probability \( p \), is "reduced" – some of its parts are removed.

Here, we consider one edge in a triangle.

As we show later, it preserves different graph properties.

How versatile is Triangle Reduction?

Before compression:

- Minimum spanning tree

During compression:

- Maximum-weight edges in sampled triangles will be
- Overlapping
- Bordered edges are parts of detected triangles
- Kernel instances

After compression:

- MST weight is preserved
The key summary

Triangle Reduction preserves (with high accuracy) representative graph properties associated with fundamental classes of workloads.
The key intuition behind preserving distances by Triangle Reduction

Now we apply Triangle Reduction!

By how much can distances increase?

Distances increase by at most 2x

This is the shortest path between two green vertices in the uncompressed graph. In the worst-case, this will be a new shortest path in the compressed graph.

We provide proofs, derivations, and discussions for many other properties...
### Theoretical Analysis of Slim Graph

| $|V|$ | $|E|$ | Shortest $s$-$t$ path length | Average path length | Diameter | Average degree | Maximum degree | #Triangles | #Connected components | Chromatic number | Max. indep. set size | Max. cardinal. matching size |
|---|---|---|---|---|---|---|---|---|---|---|---|


Different compression methods

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $|V|$ | $|E|$ | Shortest $s$-$t$ path length | Average path length | Diameter | Average degree | Maximum degree | Number of triangles | Number of connected components | Chromatic number | Max. indep. set size | Max. cardinal. matching size |

We analyzed the impact of 6 fundamental lossy graph compression methods (implemented with different compression kernels) on >12 different graph properties.
Theoretical Analysis of Slim Graph

Different graph properties

(...+ some others 😊)

We derive / provide 60+ bounds (it’s actually close to 100 now)

We analyzed the impact of 6 fundamental lossy graph compression methods (implemented with different compression kernels) on >12 different graph properties.

Each field is a separate result
We derive / provide 60+ bounds (it’s actually close to 100 now)

Different compression methods

We analyzed the impact of 6 fundamental lossy graph compression methods (implemented with different compression kernels) on >12 different graph properties

<table>
<thead>
<tr>
<th>Original graph</th>
<th>$n$</th>
<th>$m$</th>
<th>$\mathcal{P}$</th>
<th>$\overline{\mathcal{P}}$</th>
<th>$D$</th>
<th>$\overline{d}$</th>
<th>$d$</th>
<th>$T$</th>
<th>$C$</th>
<th>$C_R$</th>
<th>$I_S$</th>
<th>$M_C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summarization</td>
<td>$n$</td>
<td>$m \pm 2m$</td>
<td>$1, \ldots, \infty$</td>
<td>$1, \ldots, \infty$</td>
<td>$d \pm \epsilon d$</td>
<td>$d \pm \epsilon d$</td>
<td>$T \pm 2m$</td>
<td>$C \pm 2m$</td>
<td>$C_R \pm 2m$</td>
<td>$I_S \pm 2m$</td>
<td>$M_C \pm 2m$</td>
<td></td>
</tr>
<tr>
<td>Edge sampling</td>
<td>$n$</td>
<td>$(1 - p)m$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$(1 - p)d \pm \epsilon d$</td>
<td>$(1 - p)d \pm \epsilon d$</td>
<td>$(1 - p^3)T \pm \epsilon T$</td>
<td>$C \pm pm$</td>
<td>$C_R \pm pm$</td>
<td>$I_S \pm pm$</td>
<td>$M_C \pm pm$</td>
<td></td>
</tr>
<tr>
<td>Spectral sparsifiers</td>
<td>$n$</td>
<td>$\tilde{O}(n/e^2)$</td>
<td>$\leq n$</td>
<td>$\leq n$</td>
<td>$\tilde{O}(1/e^2)$</td>
<td>$\tilde{O}(1/e^2)$</td>
<td>$\tilde{O}(n^{2/3}/e^3)$</td>
<td>$\leq 2(1 + \epsilon)$</td>
<td>$\leq 2(1 + \epsilon)$</td>
<td>$\leq (1 + \epsilon)$</td>
<td>$\geq 0$</td>
<td></td>
</tr>
<tr>
<td>Spanners</td>
<td>$n$</td>
<td>$O(n^{1+1/k})$</td>
<td>$O(kP)$</td>
<td>$O(KP)$</td>
<td>$O(kP)$</td>
<td>$O(kP)$</td>
<td>$O(n^{1/k})$</td>
<td>$O(n^{1/k})$</td>
<td>$O(n^{1/k} \log n)$</td>
<td>$O((n^{1/k} \log n)^2)$</td>
<td>$\geq 0$</td>
<td></td>
</tr>
<tr>
<td>Triangle Reduction</td>
<td>$n$</td>
<td>$\leq m - \frac{pT}{k}$</td>
<td>$\leq \mathcal{P} + pP \leq \mathcal{P} + \frac{pT}{m(n-1)}$</td>
<td>$\leq \mathcal{P} + \frac{pT}{m(n-1)}$</td>
<td>$\leq D + pD \leq (\mathcal{P} + \frac{pT}{m(n-1)}) \leq D + pD$</td>
<td>$\leq (\mathcal{P} + \frac{pT}{m(n-1)})$</td>
<td>$\leq \frac{pT}{m(n-1)}$</td>
<td>$\leq \frac{pT}{m(n-1)}$</td>
<td>$\leq \frac{pT}{m(n-1)}$</td>
<td>$\leq \frac{pT}{m(n-1)}$</td>
<td>$\geq \frac{pT}{m(n-1)}$</td>
<td></td>
</tr>
<tr>
<td>Vertex sampling</td>
<td>$n - k$</td>
<td>$m - k$</td>
<td>$\mathcal{P}$</td>
<td>$\geq \mathcal{P} - \frac{pT}{m(n-1)}$</td>
<td>$\geq \mathcal{P} - \frac{pT}{m(n-1)}$</td>
<td>$\geq D - 2 \geq (1 - \frac{p}{n})T \geq (1 - \frac{p}{n})T$</td>
<td>$\geq \frac{pT}{m(n-1)}$</td>
<td>$\geq \frac{pT}{m(n-1)}$</td>
<td>$\geq \frac{pT}{m(n-1)}$</td>
<td>$\geq \frac{pT}{m(n-1)}$</td>
<td>$\geq \frac{pT}{m(n-1)}$</td>
<td></td>
</tr>
</tbody>
</table>

Each field is a separate result
Theoretical Analysis of Slim Graph

We derive / provide 60+ bounds (it’s actually close to 100 now)

Different graph properties

Different compression methods

We analyzed the impact of 6 fundamental lossy graph compression methods (implemented with different compression kernels) on >12 different graph properties

Looks complex - no worries, we will not go over it here 😄
Slim Graph delivers a simple, intuitive, versatile...

1. **Abstraction & programming model** for easy development and rapid prototyping of lossy graph compression methods

2. **Compression method** that preserves different graph properties that are important for the practice of graph processing

3. **Criterion (criteria?)** to assess the accuracy of lossy graph compression methods

4. **High-performance and extensible system** for implementing and executing lossy graph compression

Number of ways [1] to sparsify (compress) a graph with $n$ vertices

$$O \left( 2^{\frac{n}{2}} \right)$$

Slim Graph: Criteria for Compression Accuracy

Which compression scheme is better (= more accuracy) for which graph property?

One can analyze this in theory. This gives fundamental insights. But... it may be very hard or impossible.

Slim Graph offers different metrics based on the type of the outcome of specific graph processing workloads.
Slim Graph: Criteria for Compression Accuracy

Type of workload output: vertex importance scores

Examples: Degree Centrality, Betweenness Centrality, Katz Centrality

Metric: #reorderings, i.e., “How many pairs of vertices swapped their importance after compression?”

Let’s use degree centrality

Time to compress! 😊

v1 is ⬤
v2 is ⬤
v3 is ⬤
v4 is ⬤
v5 is ⬤
v6 is ⬤
v7 is ⬤
v8 is ⬤

v1 is ⬤
v2 is ⬤
v3 is ⬤
v4 is ⬤
v5 is ⬤
v6 is ⬤
v7 is ⬤
v8 is ⬤

v4 > v1

v5 > v7
Please check the paper for details on the precise statistical formulation 😊

**Metric**: #reorderings, i.e., “How many pairs of vertices swapped their importance after compression?”

**Example**: Degree Centrality, Katz Centrality

**Question**: How do we compress? 😏

**Answer**: We have metrics for other classes of workloads, for example...

**DIVERGENCE**

**P(x)**: Probability distribution before compression

**Q(x)**: Probability distribution after compression
Slim Graph delivers a simple, intuitive, versatile ...

1. **Abstraction & programming model** for easy development and rapid prototyping of lossy graph compression methods.

2. **Compression method** that preserves different graph properties that are important for the practice of graph processing.

3. **Criterion (criteria?)** to assess the accuracy of lossy graph compression methods.

4. **High-performance and extensible system** for implementing and executing lossy graph compression.

---

Number of ways [1] to sparsify (compress) a graph with $n$ vertices:

$$O\left(2^{\binom{n}{2}}\right)$$

Slim Graph: High-Performance Extensible System

Abstraction & Programming Model

```c
// Example kernel:
atomic reduce_triangle(...) {
    // Remove an edge from
    // a triangle with a given
    // probability
}
```

Kernels focus on:
- Edge
- Vertex
- Triangle
- Subgraph

A developer specifies compression kernels that remove selected parts of a graph, constituting different compression methods.

Compilation

Processing Engine

In stage 1, compression kernels are executed in parallel to compress graphs.

Distributed memories or I/O engines can be used for very large graphs.

Analytics Subsystem & Accuracy Metrics

In stage 2, graph algorithms are executed on compressed graphs.

Generation of graphs
Add new compression schemes by modifying current kernels or adding new ones.
Slim Graph: High-Performance Extensible System

Compress graphs (go off node if necessary)

A developer specifies compression kernels that remove selected parts of a graph, constituting different compression methods.

In stage 1, compression kernels are executed in parallel to compress graphs.

In stage 2, graph algorithms are executed on compressed graphs.

Generation of graphs
Slim Graph: High-Performance Extensible System

Compress graphs (go off node if necessary)

Evaluate selected graph algorithms

In stage 1, compression kernels are executed in parallel to compress graphs

In stage 2, graph algorithms are executed on compressed graphs

Generation of graphs

A developer specifies compression kernels that remove selected parts of a graph, consisting of vertices, edges, subgraphs, subgraphs, or entire graphs. Compression kernels are parallelized at runtime. After compression, the algorithms can be evaluated on compressed graphs.
Slim Graph: High-Performance Extensible System

Abstraction & Programming Model

```cpp
// Example kernel:
atomic reduce_triangle(...) {
    // Remove an edge from
    // a triangle with a given
    // probability
}
```

A developer specifies compression kernels that remove selected parts of a graph, constituting different compression methods.

Compilation

Processing Engine

In stage 1, compression kernels are executed in parallel to compress graphs.

Distributed memories or I/O engines can be used for very large graphs.

Use and add new accuracy metrics

Generation of graphs

Analytics Subsystem & Accuracy Metrics
Slim Graph: High-Performance Extensible System

Abstraction & Programming Model

```
// Example kernel:
atomic reduce_triangle(...) {
    Edge
    // Remove an edge from
    // a triangle with a given
    // probability
}
```

A developer specifies compression kernels that remove selected parts of a graph, constituting different compression methods.

Processing Engine

In stage 1, compression kernels are executed in parallel to compress graphs. Distributed memories or I/O engines can be used for very large graphs.

In stage 2, graph algorithms are executed on compressed graphs.

Generation of graphs

Analytics Subsystem & Accuracy Metrics

thread thread thread thread thread
Slim Graph: High-Performance Extensible System

Abstraction & Programming Model

```c
// Example kernel:
atomic reduce_triangle(...) {
  Edge
  Vertex
}
```

Kernels focus on:

A developer specifies compression kernels that remove parts of the graph to constitute compressed graphs.

Analytics

In stage 1, compression kernels are executed in parallel to compress graphs.

Compilation

Distributed memories or I/O engines can be used for very large graphs.

Processing Engine

In stage 2, graph algorithms are executed on compressed graphs.

Generation of graphs

The implementation is publicly available (you can play with lossy compression yourself!)

https://github.com/gersten/SlimGraph
Goal 1: Enable scalable compression of large graphs

CSCS Cray Piz Daint, 64 GB per compute node

Goal 2: Enable comparison of different aspects of lossy graph compression

CSCS Ault server, 768 GB of DRAM
Storage Reductions vs. Speedups vs. Accuracy Loss [vs. Compression Overhead]

Slim Graph enables comparison and analysis of four aspects of lossy graph compression:

- **Storage reductions**
  - Less storage
- **Accuracy loss**
  - Approximation
- **Compression overhead**
  - Faster workloads
- **Speedups of graph algorithms**
  - High accuracy

Various workloads and graphs
Selected insights...

- **Storage** reduced even by > 4x (depends on the structure)
- **Runtime** reduced even by > 50%
- Surprising effects revealed: runtime can increase with fewer edges (synchronization!)
- Use Slim Graph to navigate designing more efficient compression

---

**Triangle Reduction Analysis**
Workload: BFS traversal

- **Colors** indicate the compression ratio: ratio of the number of edges in the compressed graph to the number of edges in the original graph

- **Social network**
  - Storage reduced even by > 4x
  - Runtime reduced even by > 50%

- **Web metadata**
  - Storage reduced even by > 4x
  - Runtime reduced even by > 50%

---

$p$: probability of reducing a triangle

---

Runtime reduction [relative ratio]

- 0.0
- 0.5
- 1.0

0.1 0.3 0.5 0.7 0.9

---

Surprising effects revealed: runtime can increase with fewer edges (synchronization!)
Triangle Reduction Analysis
Workload: BFS traversal

Selected insights...

Accuracy follows theoretical predictions
(good balance in preserving different properties such as distances, etc.)

Storage reduced even by > 4x
(depends on the structure)

Surprising effects revealed: runtime can increase with fewer edges (synchronization!)

Use Slim Graph to navigate designing more efficient compression

Runtime reduced even by > 50%

$p$: probability of reducing a triangle

Social network

Runtime reduction [relative ratio]

0.0 0.5 1.0

0.1 0.3 0.5 0.7 0.9

Social network

Web metadata

Storage compression ratio: ratio of the number of edges in the compressed graph to the number of edges in the original graph (depends on the structure).
What if you want to go extreme in some respect?

**Storage** reduced by
> 10x

**Runtime** reduced by
> 75%

Distances increase at most by 8x, but *proportionally* to the original ones.

Colors indicate the compression ratio: ratio of the number of edges in the compressed graph to the number of edges in the original graph.
## Metrics Analyses

<table>
<thead>
<tr>
<th>Graph</th>
<th>EO 0.8-1-TR</th>
<th>EO 1.0-1-TR</th>
<th>Uniform (p = 0.2)</th>
<th>Uniform (p = 0.5)</th>
<th>Spanner (k = 2)</th>
<th>Spanner (k = 16)</th>
<th>Spanner (k = 128)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s-you</td>
<td>0.0121</td>
<td>0.0167</td>
<td>0.1932</td>
<td>0.6019</td>
<td>0.0054</td>
<td>0.2808</td>
<td>0.2993</td>
</tr>
<tr>
<td>h-hud</td>
<td>0.0187</td>
<td>0.0271</td>
<td>0.0477</td>
<td>0.1633</td>
<td>0.0340</td>
<td>0.2794</td>
<td>0.3247</td>
</tr>
<tr>
<td>il-dbl</td>
<td>0.0459</td>
<td>0.0674</td>
<td>0.0749</td>
<td>0.2929</td>
<td>0.0080</td>
<td>0.1980</td>
<td>0.2005</td>
</tr>
<tr>
<td>v-skt</td>
<td>0.0410</td>
<td>0.0643</td>
<td>0.0674</td>
<td>0.2695</td>
<td>0.0311</td>
<td>0.1101</td>
<td>0.2950</td>
</tr>
<tr>
<td>v-usa</td>
<td>0.0089</td>
<td>0.0100</td>
<td>0.1392</td>
<td>0.5945</td>
<td>0.0000</td>
<td>0.0074</td>
<td>0.0181</td>
</tr>
</tbody>
</table>

- **EO 0.8-1-TR** represents the error rate of the model when the target probability is between 0.8 and 1.0.
- **EO 1.0-1-TR** represents the error rate of the model when the target probability is exactly 1.0.
- **Uniform** measures the performance of the model under uniform distribution.
- **Spanner** measures the performance of the model with different spanner values (k = 2, k = 16, k = 128).

### Graphs

- **s-you**: Original rate 11.38, 0.2-1-TR rate 1.544.
- **s-flx**: Original rate 9.389, 0.2-1-TR rate 0.645.
- **s-flc**: Original rate 1091, 0.2-1-TR rate 6.845.
- **s-cds**: Original rate 3157, 0.2-1-TR rate 18.56.
- **s-lib**: Original rate 938.3, 0.2-1-TR rate 31.51.
- **s-pok**: Original rate 59.82, 0.2-1-TR rate 10.25.
- **h-dbp**: Original rate 6299, 0.2-1-TR rate 1.158.
- **h-hud**: Original rate 14.71, 0.2-1-TR rate 1.832.
- **l-cit**: Original rate 5973, 0.2-1-TR rate 1.994.
- **l-dbl**: Original rate 4557, 0.2-1-TR rate 6.144.
- **v-ewk**: Original rate 235.2, 0.2-1-TR rate 14.13.
- **v-skt**: Original rate 50.88, 0.2-1-TR rate 2.642.
Both **reordering** and **divergence** based accuracy metrics’ values **increase monotonically** (in all the cases) with the scope of compression.

<table>
<thead>
<tr>
<th>Graph</th>
<th>EO 0.8-1-TR</th>
<th>EO 1.0-1-TR</th>
<th>Uniform (p = 0.2)</th>
<th>Uniform (p = 0.5)</th>
<th>Spanner (k = 2)</th>
<th>Spanner (k = 16)</th>
<th>Spanner (k = 128)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s-you</td>
<td>0.0121</td>
<td>0.0167</td>
<td>0.1932</td>
<td>0.6019</td>
<td>0.0054</td>
<td>0.2808</td>
<td>0.2993</td>
</tr>
<tr>
<td>s-flx</td>
<td>0.0027</td>
<td>0.0047</td>
<td>0.0832</td>
<td>0.2984</td>
<td>0.0042</td>
<td>0.1980</td>
<td>0.2005</td>
</tr>
<tr>
<td>s-fle</td>
<td>0.3157</td>
<td>0.3241</td>
<td>0.5481</td>
<td>0.6364</td>
<td>0.0295</td>
<td>0.1410</td>
<td>0.1448</td>
</tr>
<tr>
<td>s-cds</td>
<td>0.9894</td>
<td>0.9894</td>
<td>0.9894</td>
<td>0.9894</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>s-pok</td>
<td>0.5982</td>
<td>0.5982</td>
<td>0.5982</td>
<td>0.5982</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
</tbody>
</table>
SLIM GRAPH ENABLES DISTRIBUTED COMPRESSION FOR LARGE GRAPHS
Distributed Large-Scale Lossy Compression with Slim Graph

5 largest publicly available real-world graphs

Results for the largest graph:

Uncompressed

Fraction of vertices

Outdegree

Loading from disk: 920s

40% edges removed

Loading from disk: 531s

70% edges removed

Loading from disk: 276s

Compression takes 3.8s

Compression takes 3.6s

I/O time is reduced (benefits any computation)

Use Slim Graph to find novel use cases of lossy graph compression

Slim Graph enabled us to discover an interesting effect of “removing the clutter” – (mild) sampling could be used as preprocessing
1. **Abstraction & programming model** for easy development and rapid prototyping of lossy graph compression methods.

2. **Compression method** that preserves different graph properties that are important for the practice of graph processing.

3. **Criterion** to assess the accuracy of lossy graph compression methods.

4. **High-performance and extensible system** for implementing and executing lossy graph compression.

---

Number of ways \([1] \) to sparsify (compress) a graph with \( n \) vertices

\[
O \left(2^{\binom{n}{2}}\right)
\]

---

A COMPRESSION ABSTRACTION & MODEL

Central concept is compression kernels: small code snippets that remove specified local parts of the graph. Different kernels enable different compression methods. Let’s see some examples...

https://github.com/rgersten/SlimGraph

A VERSATILE COMPRESSION SCHEME

Slim Graph: A Novel Compression Method “Triangle Reduction”

- Each triangle, with a certain selected probability, is reduced so some of its parts are removed.
- Here we consider one edge in a triangle.
- As we show later, it preserves different graph properties.

OLAP kernels

Before compression

During compression

After compression

Slim Graph: High-Performance Extensible System

High-Performance System

The implementation is publicly available (you can play with your compression yourself)!
https://github.com/rgersten/SlimGraph

COMPRESSON ACCURACY CRITERIA

SLIM GRAPH OVERVIEW

https://github.com/rgersten/SlimGraph

GUIDELINES

Ongoing work...

THEORETICAL ANALYSIS & EVALUATION

- How expressive is the compression kernel abstraction?
  - Random uniform (and other forms of) sampling
  - Spectral sparsifiers (preserve spectra)
  - Cut sparsifiers (preserve cuts)

- Compression kernels: an abstraction that enables expressing fundamental classes of sparsification
  - Summarizations (preserve neighborhoods)
  - Spans and (preserve pairwise distances)
  - Others

- Theoretical classification results for compression accuracy

- We investigated over 500 papers (theory) to distill the key classes of graph sparsification
Backup Slides and Slides’ Variants
Various real-world graphs are used

Various workloads are considered

Selected insights…
Compressing Largest-Scale Graphs with Slim Graph

The first analysis of the impact of spanners on degree distribution

An interesting “leveling” effect
How large are extreme-scale graphs today?

Labatory for Web Algorithmics datasets [1]

<table>
<thead>
<tr>
<th>Graph</th>
<th>Crawl date</th>
<th>Nodes</th>
<th>Arcs</th>
</tr>
</thead>
<tbody>
<tr>
<td>uk-2014</td>
<td>2014</td>
<td>787,801,471</td>
<td>4,761,452,7250</td>
</tr>
<tr>
<td>eu-2015</td>
<td>2015</td>
<td>1,070,557,254</td>
<td>91,792,261,600</td>
</tr>
<tr>
<td>gsh-2015</td>
<td>2015</td>
<td>988,490,691</td>
<td>33,877,399,152</td>
</tr>
</tbody>
</table>

Web data commons datasets [2]

<table>
<thead>
<tr>
<th>Granularity</th>
<th>#Nodes</th>
<th>#Arcs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page</td>
<td>3,563 million</td>
<td>128,736 million</td>
</tr>
<tr>
<td>Host</td>
<td>101 million</td>
<td>2,043 million</td>
</tr>
</tbody>
</table>

The runs used nearly all memory on compute nodes of TaihuLight!

Sogou 搜狗

> 233 TB

> 271 billion vertices, 12 trillion edges [4]

How about lossless compression?

[Traditional] compression incurs expensive decompression [1,2]

[1] M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18
But... we show [1,2] that ≈20-30% less storage is really as good as you can get due to fundamental storage lower bounds.

Important conclusion: theory won’t let us go (too much) further.

How about lossless compression?

[1] M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18
Substream-Centric Slim Graph

Use a hybrid CPU-FPGA setting!

Process different kernels independently (e.g., remove from the pipeline based on the compression kernel code)

Output a compressed graph

Pipelines – it fits FPGA well!

Use some form of streaming; we can use pipelining efficiently ("streaming ≈ pipelining")

Kernel arguments

DRAM
Slim Graph: Abstraction & Programming Model

Input:

At least two paths (this one is relevant!)

This poor one has 0

Betweenness Centrality [1] relative scores are preserved [2]

Betweenness centrality of a vertex determines the vertex importance (#shortest paths)

[Extreme-Scale] Graphs

Why do we care?

Useful model

Engineering networks

Machine learning

Physics, chemistry

Physics, chemistry

Aminos

Aromatics

Carbohydrates and Organic Acids

Modeling a Philosophical Inquiry: from MySQL to a graph database

The short story of a long refactoring process

Bruno Latour wrote a book about philosophy (an inquiry into modes of existence). He decided that the paper book was no place for the numerous footnotes, documentation or glossary, instead giving access to all this information surrounding the book through a web application which would present itself as a reading companion. He also offered to the community of readers to submit their contributions to his inquiry by writing new documents to be added to the platform. The first version
Slim Graph: Abstraction and Programming Model

**Edge kernels:** implementing spectral sparsification and sampling

```
1 /************ Single-edge compression kernels (§ 4.2) **************/
2 spectral_sparsify(E e) {  // More details in § 4.2.1
3     double Y = SG.connectivity_spectral_parameter();
4     double edge_stays = min(1.0, Y / min(e.u.deg, e.v.deg));
5     if(edge_stays < SG.rand(0,1)) atomic SG.del(e);
6     else e.weight = 1/edge_stays;
7 }
8 random_uniform(E e) {  // More details in § 4.2.2
9     double edge_stays = SG.p;
10    if(edge_stays < SG.rand(0,1)) atomic SG.del(e);
11 }
```
Slim Graph: Abstraction and Programming Model

Subgraph kernels: spanners
Slim Graph: Abstraction & Programming Model

Before compression:
- Example neighborhood
- Find supervertices (clusters of vertices)
- A kernel decides which supervertices are connected with a superedge
- Construct graph $\mathcal{E}$-summary:
  - #edges added to corrections depends on $\mathcal{E}$
  - Vertices in clusters are merged into supervertices
  - Edges between selected clusters are merged into superedges
- Example clusters
- Dashed edges are omitted (selected randomly)
- Bolded edges are merged into superedges

Kernel instances

After compression:
- Dashed edges will be omitted (selected randomly)
- Edges to be inserted when decompressing
- Edges to be removed when decompressing
- "Additional" edges created when merging edges into superedges
- Dashed edges are omitted (lossy summarization)
- Corrections (used to decompress)
- $\mathcal{E}$ determines the scope of lossy compression
- Differences in neighborhoods are determined by $\mathcal{E}$
Single-vertex compression kernel (§ 4.4)

```c
23  /******************************************
24  low_degree(V v) {
25      if(v.deg==0 or v.deg==1) atomic SG.del(v); }
26  /******************************************
```

Subgraph compression kernels (§ 4.5)

```c
27  derive_spanner(vector<V> subgraph) { //Details in § 4.5.3
28      //Replace "subgraph" with a spanning tree
29      subgraph = derive_spanning_tree(subgraph);
30      //Leave only one edge going to any other subgraph.
31      vector<set<V>> subgraphs(SG.sgr_cnt);
32      foreach(E e: SG.out_edges(subgraph)) {
33          if(!subgraphs[e.v.elem_ID].empty()) atomic del(e);
34      }
35  }
```

```c
derive_summary(vector<V> cluster) { //Details in § 4.5.4

tuple_to_summary(sv); //余万元 summary out of a current cluster:
37      SV sv = SG.min_id(cluster);
38      SG.summary.insert(sv); //Insert sv into a summary graph
39      //Select edges (to preserve) within a current cluster:
40      vector<e> intra = SG.summary_select(cluster, SG.e);
41      SG.corrections_plus.append(intra);
42      //Iterate over all clusters connected to "cluster":
43      foreach(vector<V> c: SG.out_clusters(out_edges(cluster))) {
44          [E, vector<E>](se, inter) = SG.superedge(cluster, c1, SG.e);
45          SG.summary.insert(se);
46          SG.corrections_minus.append(inter);
47      }
48      SG.update_convergence();
49  }
```
/* ********** Single-edge compression kernels (§ 4.2) ***********/

1  spectral_sparsify(E e) { // More details in § 4.2.1
2    double Y = SG.connectivity_spectral_parameter();
3    double edge_stays = min(1.0, Y / min(e.u.deg, e.v.deg));
4    if(edge_stays < SG.rand(0,1)) atomic SG.del(e);
5    else e.weight = 1/edge_stays;
6  }
7
8  random_uniform(E e) { // More details in § 4.2.2
9    double edge_stays = SG.p;
10   if(edge_stays < SG.rand(0,1)) atomic SG.del(e);
11  }
12
13  /********** Triangle compression kernels (§ 4.3) ***********/
14  p-1-reduction(vector<E> triangle) {
15    double tr_stays = SG.p;
16    if(tr_stays < SG.rand(0,1))
17      atomic SG.del(rand(triangle));
18  }
19
20  p-1-reduction-EO(vector<E> triangle) {
21    double tr_stays = SG.p;
22    if(tr_stays < SG.rand(0,1)) {
23      E e = rand(triangle);
24      atomic { if(!e.considered) SG.del(e);
25        else e.considered = true; } }
26  }
27
28
The key intuition behind some derivations for Triangle Reduction

This is the shortest path between two green vertices in the uncompressed graph. In the worst-case, this will be a new shortest path in the compressed graph.

Distances increase by at most $2x$

Graph traversals (e.g., BFS, SSSP)

By how much can distances increase?

Distances

By applying Triangle Reduction, distances can increase by at most $2x$.

Can we disconnect a graph?

No 😞

Graph is still connected

A triangle is a 3-cycle!

Connectivity

Connectivity related (e.g., #connected components)

Now we apply Triangle Reduction!

Now we apply Triangle Reduction!

Graph is connected
Challenge 2: Theoretical schemes are complex and hard to code and use – how to simplify?
Challenge 3: What schemes matter in practice?
Theoretical Analysis

<table>
<thead>
<tr>
<th>12 graph properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>Original graph</td>
</tr>
<tr>
<td>Lossy (\epsilon)-summary</td>
</tr>
<tr>
<td>Simple (p)-sampling</td>
</tr>
<tr>
<td>Spectral (\epsilon)-sparsifier</td>
</tr>
<tr>
<td>(O(k))-spanner</td>
</tr>
<tr>
<td>EO (p)-1-Triangle Red. remove (k) deg-1 vertices</td>
</tr>
</tbody>
</table>

**Summarizations are not accurate** (graphs can get arbitrarily disconnected)

Sampling is accurate only in expectation (or w.h.p.) and when not many edges go (all depends on whether a graph gets disconnected)

Spanners and spectral sparsifiers preserve well their associated properties

Some are new and non-trivial, for example we prove constructively a lower bound on the maximum cardinality matching (MCM) size that depends only on the original MCM size.
Triangle Reduction is versatile; it also has properties of 2-spanners (or – w.h.p. – $O(\log n)$ spanners), cut sparsifiers (and is thus a special case of spectral sparsifiers).

Preserves exactly connectivity and the MST weight.

Preserves provably well distances, cuts, and the degree distribution.
A “By Product” of Our Work

The first survey on lossy graph compression

Properties of compression classes

<table>
<thead>
<tr>
<th>Compression scheme</th>
<th>#remaining edges</th>
<th>Work</th>
<th>Serves best...</th>
</tr>
</thead>
<tbody>
<tr>
<td>(§ 4.2.1) Spectral sparsification (*&quot;High-conductance&quot; sampling [16])</td>
<td>(e n^2)</td>
<td>(O(m + n))</td>
<td>Unknown</td>
</tr>
<tr>
<td>(§ 4.2.2) Edge sampling (simple random [16])</td>
<td>(2kn)</td>
<td>(O(m + n))</td>
<td>Unknown</td>
</tr>
<tr>
<td>(§ 4.3) Triangle reduction</td>
<td>(-)</td>
<td>(O(n^2))</td>
<td>[High error rates]</td>
</tr>
<tr>
<td>(§ 4.5.3) Spanners ((O(k))</td>
<td>(O(n \log n e^2))</td>
<td>(O(n \log n))</td>
<td>Count of common neighbors</td>
</tr>
<tr>
<td>(§ 4.5.4) Lossy summarization</td>
<td>(O(C^2 \log n + nm_3))</td>
<td>(O(m + n))</td>
<td>Unknown</td>
</tr>
</tbody>
</table>

Table 2: (§ 4) Considered lossy compression schemes. \(^1\)W.D indicate support for weighted or directed graphs, respectively. Symbols used in Slim Graph schemes (p, k) are explained in corresponding sections. \(^2\)Storage needed to conduct compression. In the SWeG lossy summarization [125], \(e\) controls the approximation ratio while \(I\) is the number of iterations (originally set to 80 [125]). \(^3\)SWeG covers undirected graphs but uses a compression metric for directed graphs. In ApxMd [101], \(e\) controls the approximation ratio, \(C \in O(m)\) is the number of "corrections", \(m_3 \in O(m)\) is the number of "corrected" edges. In lossy linearization [95], \(k \in O(n)\) is a user parameter, \(I\) is the number of iterations of a "re-allocation process" (details in Section V.C.3 in the original work [95]), while \(T\) is a number of iterations for the overall algorithm convergence. In clustered SVD approximation [119, 132], \(n_c \leq n\) is the number of vertices in the largest cluster in low-rank approximation. In cut sparsifiers [15], \(e\) controls the approximation ratio of the cuts.

No time for this – check the paper 😊 (and stay tuned for the full survey paper coming soon!)
Theoretical Analysis

**Triangle Reduction** is versatile; it also has properties of:

- 2-spanners
- $O(\log n)$ spanners (w.h.p.),
- cut sparsifiers

(and is thus a special case of spectral sparsifiers)

Preserves exactly connectivity and the MST weight

Preserves provably well distances, cuts, matchings, the degree distribution, and others..
Various workloads are considered

Various real-world graphs are used

Selected insights...
Various real-world graphs are used

Various workloads are considered

Selected insights...
We use **subgraph kernels** to express and implement **spanners** [1]: a form of lossy compression where the input graph is decomposed into a set of (interconnected) spanning trees, with all other edges removed.

\[ \propto \text{Diameter } D \text{ of subgraph kernels} \]

(higher \( D \) → more edges are removed)

Colors indicate the compression ratio: ratio of the number of edges in the compressed graph to the number of edges in the original graph.

**Storage** reduced even by \( > 10x \) (depends on the structure)

**Runtime** reduced even by \( > 75\% \)

---

Selected insights...

Various real-world graphs are used

Various workloads are considered
## Compressing Largest-Scale Graphs with Slim Graph

The first analysis of the impact of spanners on degree distribution.

An interesting "leveling" effect.
# Accuracy Analysis: Compressing Largest-Scale Graphs with Slim Graph

<table>
<thead>
<tr>
<th>No compression</th>
<th>Sampling (p=0.3)</th>
<th>Sampling (p=0.6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fraction of vertices</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>13</td>
<td>20</td>
</tr>
<tr>
<td>50</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Counts of compute nodes used to compress respective graphs</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Largest-scale graph compression so far**

**5 largest publicly available real-world graphs**

**“removing the clutter” – (mild) sampling could be used as preprocessing?**

**Figure 7: (Accuracy) Impact of random uniform sampling on the degree distribution of large graphs (the largest, h-wdc, has \( \approx 128B \) edges).**
### Kullback-Leibler divergence values between PageRank probability distributions in the original vs. the compressed graph

<table>
<thead>
<tr>
<th>Graph</th>
<th>Triangle Reduction</th>
<th>Edge sampling</th>
<th>Spanners</th>
</tr>
</thead>
<tbody>
<tr>
<td>s-you</td>
<td>0.0121</td>
<td>0.1932</td>
<td>0.0054</td>
</tr>
<tr>
<td>h-hud</td>
<td>0.0187</td>
<td>0.0477</td>
<td>0.0340</td>
</tr>
<tr>
<td>il-dbl</td>
<td>0.0459</td>
<td>0.0749</td>
<td>0.0080</td>
</tr>
<tr>
<td>v-skt</td>
<td>0.0410</td>
<td>0.0674</td>
<td>0.0311</td>
</tr>
<tr>
<td>v-usa</td>
<td>0.0089</td>
<td>0.1392</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

In each category, columns to the right indicate more edges removed. The KL divergence is always larger when more edges are removed.

Various real-world graphs are used.

<table>
<thead>
<tr>
<th>Storage Reductions vs.</th>
<th>vs. Accuracy Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge sampling</td>
<td>Spanners</td>
</tr>
</tbody>
</table>