ProbGraph: High-Performance and High-Accuracy Graph Mining with Probabilistic Set Representations

Graph Mining
Graph Mining

A huge & complex graph dataset
Graph Mining

Pattern counting (triangles, higher-order cliques, dense subgraphs, ...)

A huge & complex graph dataset
Graph Mining

A huge & complex graph dataset

Pattern counting (triangles, higher-order cliques, dense subgraphs, ...)

Clustering, Link Prediction, Vertex Similarity, ...
Graph Mining: Do We Care?
Graph Mining: Do We Care?

Social sciences
Graph Mining: Do We Care?

Social sciences

Engineering
Graph Mining: Do We Care?

Social sciences
Biology
Chemistry
Engineering
Communication
Web graph analysis
Medicine
Cybersecurity

...even philosophy 😊

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 to place for the numerous footnotes, documentation or glossary. Instead, giving success to all the information surrounding the book through a web application which would present itself as a reading companion. He also offered the community of philosophical readers the possibility to comment, discuss and even design their own readings.
Graph Mining: Do We Care?

Challenges

Modeling a Philosophical Inquiry: from MySQL to a graph database
The short story of a long refactoring process
Graph Mining & Graph Datasets: Challenges
Graph Mining & Graph Datasets: Challenges

Huge

Graph Mining & Graph Datasets: Challenges

Huge

Irregular

Graph Mining & Graph Datasets: Challenges

- Huge
- Irregular
- Communication-heavy
- Synchronization-heavy

Graph Mining & Graph Datasets: Challenges

Huge
Communication-heavy
Synchronization-heavy
Irregular

Graph Mining & Graph Datasets: Challenges

- Huge
- Irregular
- Communication-heavy
- Synchronization-heavy
- Power-hungry

Time complexities often $O(n^k)$ for $k \geq 2$

Goal: Making Graph Mining Radically Faster
Goal: Making Graph Mining Radically Faster

Do we need 100% accurate results in all cases?
Goal: Making Graph Mining Radically Faster

Do we need 100% accurate results in all cases?

Let’s say we can choose between...

- Find all the patterns (e.g., cliques) in 1 day
- Find $\geq 90\%$ of all the patterns in 30 minutes
Goal: Making Graph Mining Radically Faster

Do we need 100% accurate results in all cases?

Let’s say we can choose between...

- Find all the patterns (e.g., cliques) in 1 day
- Find $\geq 90\%$ of all the patterns in 30 minutes
Approximate Graph Processing: State & Challenges
Approximate Graph Processing: State & Challenges

We analyzed > 500 works and identified three classes of schemes...
Approximate Graph Processing: State & Challenges

We analyzed > 500 works and identified three classes of schemes...

- **Approximable algorithms (APX, etc.)**
- **Heuristics**
- **Lossy graph compression**
Approximate Graph Processing: State & Challenges

We analyzed > 500 works and identified three classes of schemes...

- Approximable algorithms (APX, etc.)
- Lossy graph compression
- Heuristics

...they all have problems
Approximate Graph Processing: State & Challenges

We analyzed > 500 works and identified three classes of schemes...

Approximable algorithms (APX, etc.)
- Specific
- Slow

Little parallelism

Low accuracy

Heuristics

...they all have problems

Lossy graph compression
Approximate Graph Processing: State & Challenges

We analyzed > 500 works and identified three classes of schemes...

- Approximable algorithms (APX, etc.)
  - Little parallelism

- Heuristics
  - Specific
  - No/loose accuracy guarantees
  - Specific
  - Slow
  - Low accuracy

...they all have problems

- Lossy graph compression
Approximate Graph Processing: State & Challenges

We analyzed > 500 works and identified three classes of schemes...

- Approximable algorithms (APX, etc.)
  - Little parallelism
  - Specific
  - Slow

- Heuristics
  - Specific
  - No/loose accuracy guarantees
  - Slow

- Lossy graph compression
  - Large memory overheads
  - No/loose accuracy guarantees
  - Slow
Approximate Graph Processing: Current Issues & Our Objectives
Approximate Graph Processing: Current Issues & Our Objectives

- Little parallelism
- Specific
- No/loose accuracy guarantees
- Slow
- Low accuracy
- Large memory overheads
Approximate Graph Processing: Current Issues & Our Objectives

- Little parallelism
- Rich parallelism
- Specific
- No/loose accuracy guarantees
- Slow
- Low accuracy
- Large memory overheads
Approximate Graph Processing: Current Issues & Our Objectives

- Little parallelism → Rich parallelism
- Specific → Wide applicability
- No/loose accuracy guarantees
- Slow
- Low accuracy
- Large memory overheads
Approximate Graph Processing: Current Issues & Our Objectives

- Little parallelism → Rich parallelism
- Specific → Wide applicability
- No/loose accuracy guarantees → Strong accuracy guarantees
- Slow
- Low accuracy
- Large memory overheads
Approximate Graph Processing: Current Issues & Our Objectives

- Little parallelism → Rich parallelism
- Specific → Wide applicability
- No/loose accuracy guarantees → Strong accuracy guarantees
- Slow → High performance
- Low accuracy
- Large memory overheads
Approximate Graph Processing: Current Issues & Our Objectives

- Little parallelism → Rich parallelism
- Specific → Wide applicability
- No/loose accuracy guarantees → Strong accuracy guarantees
- Slow → High performance
- Low accuracy → High accuracy
- Large memory overheads
Approximate Graph Processing: Current Issues & Our Objectives

- Little parallelism → Rich parallelism
- Specific → Wide applicability
- No/loose accuracy guarantees → Strong accuracy guarantees
- Slow → High performance
- Low accuracy → High accuracy
- Large memory overheads → Low & controllable memory overheads
How to achieve all these objectives in a single design?

- No/loose accuracy guarantees
- Strong accuracy guarantees
- Slow
- High performance
- Low accuracy
- High accuracy
- Large memory overheads
- Low & controllable memory overheads

Approximate Graph Processing: Current Issues & Our Objectives
Approximate Graph Processing: Current Issues & Our Objectives

How to achieve all these objectives in a single design?

We develop **ProbGraph**: a graph representation that uses probabilistic set representations (aka sketches)
High-Level Approach Taken in ProbGraph
High-Level Approach Taken in ProbGraph

Keep the original graph
High-Level Approach Taken in ProbGraph

- Keep the original graph
- Maintain a very small “sketch” of a graph
High-Level Approach Taken in ProbGraph

Keep the original graph

Maintain a very small “sketch” of a graph

Use the sketch to answer performance critical queries
High-Level Approach Taken in ProbGraph

Keep the original graph

Maintain a very small “sketch” of a graph

What design to use for the sketch, to satisfy all the goals?

Use the sketch to answer performance critical queries
ProbGraph key idea: Use probabilistic set representations (set sketches)
**ProbGraph key idea**: Use probabilistic set representations (set sketches)

A set = \{A, B, C\}
**ProbGraph key idea:** Use probabilistic set representations (set sketches)

A set = \{A, B, C\}
ProbGraph key idea: Use probabilistic set representations (set sketches)

A set = \{A, B, C\}
**ProbGraph key idea:** Use probabilistic set representations (set sketches)

A set = \{A, B, C\}

[Bloom filters (BF) [1]](1)
[MinHash [2]](2)
[K Minimum Values [3]](3)

**ProbGraph key idea:** Use probabilistic set representations (set sketches)

A set = \{A, B, C\}

Bloom filters (BF) [1]  
MinHash [2]  
K Minimum Values [3]

ProbGraph key idea: Use probabilistic set representations (set sketches)

A set = \{A, B, C\}

Bloom filters (BF) [1]  
MinHash [2]  
K Minimum Values [3]

Bloom Filters for Graph Mining

A set = \{A, B, C\}
Bloom Filters for Graph Mining

A set = \{A, B, C\}

Bloom filter $\mathcal{B}_X$ of X

Bitvector of size $B_X$ [bits]
Bloom Filters for Graph Mining

A set = \{A, B, C\}

Bloom filter $B_X$ of X

Bitvector of size $B_X$ [bits] $B_X = 12$
Bloom Filters for Graph Mining

A set = \{A, B, C\}

Bloom filter \( \mathcal{B}_X \) of \( X \)

Bitvector of size \( B_X \) [bits] \( B_X = 12 \)

Hash functions \( h_1, \ldots, h_b \)

\( h_i : X \rightarrow \{1, \ldots, B_X\} \)
Bloom Filters for Graph Mining

A set = \{A, B, C\}

Bloom filter $\mathcal{B}_X$ of $X$

- Bitvector of size $B_X$ [bits] $B_X = 12$
  - Hash functions $h_1, \ldots, h_b$
  - $h_i : X \rightarrow \{1, \ldots, B_X\}$
  - $b = 2$
  - $h_2, h_1 : X \rightarrow \{1, \ldots, 12\}$
Bloom Filters for Graph Mining

A set = \{A, B, C\}

Bloom filter $\mathcal{B}_X$ of $X$

- Bitvector of size $B_X$ [bits]: $B_X = 12$

- Hash functions $h_1, \ldots, h_b$
  - $h_i : X \rightarrow \{1, \ldots, B_X\}$
  - $b = 2$
  - $h_2, h_1 : X \rightarrow \{1, \ldots, 12\}$

  - $h_1(A) = 3$
  - $h_2(A) = 5$
Bloom Filters for Graph Mining

A set = \{A, B, C\}

Bloom filter \( \mathcal{B}_X \) of \( X \)

Bitvector of size \( B_X \) [bits] \( B_X = 12 \)

Hash functions \( h_1, \ldots, h_b \)
\( h_i : X \to \{1, \ldots, B_X\} \)

\( b = 2 \)
\( h_2, h_1 : X \to \{1, \ldots, 12\} \)

\( h_1(\text{A}) = 3 \)
\( h_2(\text{A}) = 5 \)
**Bloom Filters for Graph Mining**

A set = \{A, B, C\}

**Bloom filter** $\mathcal{B}_X$ of $X$

Bitvector of size $B_X$ [bits]  \[ B_X = 12 \]

Hash functions $h_1, \ldots, h_b$

$h_i : X \rightarrow \{1, \ldots, B_X\}$

$b = 2$

$h_2, h_1 : X \rightarrow \{1, \ldots, 12\}$

- $h_1(\text{A}) = 3$  \[ h_1(\text{B}) = 1 \]
- $h_2(\text{A}) = 5$  \[ h_2(\text{B}) = 8 \]
Bloom Filters for Graph Mining

A set = \{A, B, C\}

Bloom filter $B_X$ of $X$

Bitvector of size $B_X$ [bits] $B_X = 12$

Hash functions $h_1, \ldots, h_b$
$h_i : X \rightarrow \{1, \ldots, B_X\}$

\[
\begin{align*}
B_X &= 12 \\
\text{b} &= 2 \\
h_2, h_1 : X &\rightarrow \{1, \ldots, 12\} \\

h_1(A) &= 3 & h_1(B) &= 1 \\
h_2(A) &= 5 & h_2(B) &= 8
\end{align*}
\]
Bloom Filters for Graph Mining

A set = \{A, B, C\}

Bloom filter \( \mathcal{B}_X \) of X

Bitvector of size \( B_X \) [bits]

\[
\begin{array}{cccccccccccc}
\hline
& & & & & & & & & & & & \\
\hline
\end{array}
\]

\( B_X = 12 \)

Hash functions \( h_1, \ldots, h_b \)

\( h_i : X \rightarrow \{1, \ldots, B_X\} \)

\( b = 2 \)

\( h_2, h_1 : X \rightarrow \{1, \ldots, 12\} \)

\begin{align*}
& h_1(\text{A}) = 3 & h_1(\text{B}) = 1 & h_1(\text{C}) = 4 \\
& h_2(\text{A}) = 5 & h_2(\text{B}) = 8 & h_2(\text{C}) = 11
\end{align*}
Bloom Filters for Graph Mining

A set = \{A, B, C\}

Bloom filter $B_X$ of $X$

Bitvector of size $B_X$ [bits] $B_X = 12$

Hash functions $h_1, ..., h_b$

$h_i : X \rightarrow \{1, ..., B_X\}$

$b = 2$

$h_2, h_1 : X \rightarrow \{1, ..., 12\}$

$h_1(\text{A}) = 3$ \hspace{1cm} $h_1(\text{B}) = 1$ \hspace{1cm} $h_1(\text{C}) = 4$

$h_2(\text{A}) = 5$ \hspace{1cm} $h_2(\text{B}) = 8$ \hspace{1cm} $h_2(\text{C}) = 11$
Bloom Filters for Graph Mining
Bloom Filters for Graph Mining

Each neighborhood $N_u$ is a set of vertices
Each neighborhood \( N_u \) is a set of vertices
Bloom Filters for Graph Mining

Each neighborhood $N_u$ is a set of vertices

“Sketch” each $N_u$ with a Bloom filter
Each neighborhood $N_u$ is a set of vertices

“Sketch” each $N_u$ with a Bloom filter
Each neighborhood $N_u$ is a set of vertices

„Sketch” each $N_u$ with a Bloom filter
Bloom Filters for Graph Mining

Each neighborhood $N_u$ is a set of vertices

“Sketch” each $N_u$ with a Bloom filter
Bloom Filters for Graph Mining

Each neighborhood \( N_u \) is a set of vertices

„Sketch” each \( N_u \) with a Bloom filter
ProbGraph: Summary of Design
ProbGraph: Summary of Design

Input graph $G$
ProbGraph: Summary of Design

Input graph G

Standard graph representation (e.g., CSR)
ProbGraph: Summary of Design

Input graph G

Standard graph representation (e.g., CSR)
ProbGraph: Summary of Design

Input graph G

ProbGraph representation

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>
ProbGraph: Summary of Design

Input graph G

ProbGraph representation

Bloom filters

1 → 2 → 3
2 → 1 → 3 → 4
3 → 1 → 2 → 4 → 5 → 6
4 → 2 → 3 → 5
5 → 3 → 4 → 6 → 7 → 8
6 → 3 → 5 → 7
7 → 5 → 6 → 8
8 → 5 → 7
ProbGraph: Summary of Design

Input graph G

ProbGraph representation

Bloom filters

Larger $B_x$: more accuracy & more storage required. Lower $B_x$: vice versa.
ProbGraph: Summary of Design

ProbGraph representation

Bloom filters

Larger $B_x$: more accuracy & more storage required. Lower $B_x$: vice versa.

$B_x$ is often small $\rightarrow$ little storage
ProbGraph: Summary of Design

Input graph G

1 3 6
2 4 7
5

ProbGraph representation

B_x is often small \rightarrow little storage
BFs have the same size \rightarrow great load balancing

Bloom filters

Larger B_x : more accuracy & more storage required. Lower B_x : vice versa.
How does our idea compare to other Bloom filter use cases?
How does our idea compare to other Bloom filter use cases?

Traditional BF use case: presence tracking
How does our idea compare to other Bloom filter use cases?

Traditional BF use case: presence tracking

Data stored somewhere
How does our idea compare to other Bloom filter use cases?

Traditional BF use case: presence tracking

A BF cache tracking the presence of data

Data stored somewhere
How does our idea compare to other Bloom filter use cases?

Traditional BF use case: presence tracking

A BF cache tracking the presence of data

Data stored somewhere
How does our idea compare to other Bloom filter use cases?

Traditional BF use case: presence tracking

Traditional BF use case: presence tracking

A BF cache tracking the presence of data

Data stored somewhere

Insert an element
How does our idea compare to other Bloom filter use cases?

Traditional BF use case: presence tracking

- Insert an element
- Set the appropriate BF bits

A BF cache tracking the presence of data

Data stored somewhere
How does our idea compare to other Bloom filter use cases?

Traditional BF use case: presence tracking

1. Insert an element
2. Set the appropriate BF bits
3. A BF cache tracking the presence of data
4. Data stored somewhere
How does our idea compare to other Bloom filter use cases?

Traditional BF use case: presence tracking
How does our idea compare to other Bloom filter use cases?

Traditional BF use case: presence tracking

Is the data in question over there?

Yes

A BF cache tracking the presence of data

Data stored somewhere
How does our idea compare to other Bloom filter use cases?

Traditional BF use case: presence tracking

Is the data in question over there?

Yes

Fetch the data
How does our idea compare to other Bloom filter use cases?

Traditional BF use case: presence tracking

Is the data in question over there?

Yes

Fetch the data

This is usually a slow operation

A BF cache tracking the presence of data

Data stored somewhere
How does our idea compare to other Bloom filter use cases?

Traditional BF use case: presence tracking

Is the data in question over there?

Yes

Fetch the data

A BF cache tracking the presence of data

Data stored somewhere

This is a very fast operation

This is usually a slow operation

Yes
The novelty of ProbGraph

Data stored somewhere

A BF cache tracking the presence of data

Is the data in question over there?

Fetch the data

This is usually a slow operation

This is a very fast operation

Yes

Traditional BF use case: presence tracking

How does our idea compare to other Bloom filter use cases?
We use BFs as a sketch of the actual dataset.
The novelty of ProbGraph

We use BFs as a sketch of the actual dataset

This is usually a slow operation

Data stored somewhere

Traditional BF use case: presence tracking

How does our idea compare to other Bloom filter use cases?

Fetch the data

This is a very fast operation

Yes

BF cache tracking the presence of data

Is the data in question over there?
The novelty of ProbGraph

We use BFs as a sketch of the actual dataset

How do we exactly use these sketches to benefit graph mining?
Observation: Set Intersection Cardinality is Prevalent in Graph Mining
Observation: Set Intersection Cardinality is Prevalent in Graph Mining
Observation: Set Intersection Cardinality is Prevalent in Graph Mining

$|X \cap Y|$
Observation: Set Intersection Cardinality is Prevalent in Graph Mining

Graph Mining

- A huge & complex graph dataset
- Clustering, Link Prediction, Vertex Similarity, ...
- Pattern counting (triangles, higher-order cliques, dense subgraphs, ...)

$|X \cap Y|$
Observation: Set Intersection Cardinality is Prevalent in Graph Mining

\[ |X \cap Y| \]

We greatly accelerate \(|X \cap Y|\) with BFs

A huge & complex graph dataset

Pattern counting (triangles, higher-order cliques, dense subgraphs, ...)

Clustering, Link Prediction, Vertex Similarity, ...
ProbGraph key idea, continued
ProbGraph key idea, continued
ProbGraph key idea, continued
ProbGraph key idea, continued
ProbGraph key idea, continued
ProbGraph key idea, continued

$\mathbb{N}_u$ $u$ $\mathbb{N}_v$

$\mathbb{N}_u$ $\rightarrow$ [Diagram of nodes and edges]

$\mathbb{N}_v$ $\rightarrow$ [Diagram of nodes and edges]
ProbGraph key idea, continued
ProbGraph key idea, continued
Approximate Graph Processing: Our Objectives

- Little parallelism → Rich parallelism
- Specific → Wide applicability
- No/loose accuracy guarantees → Strong accuracy guarantees
- Slow → High performance
- Low accuracy → High accuracy
- Large memory overheads → Low & controllable memory overheads
Approximate Graph Processing: Our Objectives

- Little parallelism
  - Specific
  - No/loose accuracy guarantees
  - Slow
  - Low accuracy
  - Large memory overheads

- Rich parallelism
  - Wide applicability
  - Strong accuracy guarantees
  - High performance
  - High accuracy
  - Low & controllable memory overheads
ProbGraph: Fast & Parallel Execution

Probabilistic Graphical Models (ProbGraphs) enable efficient and parallel execution of queries. The diagram illustrates how sets $N_u$ and $N_v$ can be processed bitwise to compute the set intersection $N_u \cap N_v$. The bitwise AND operation efficiently computes this intersection, leading to faster and parallel execution of queries in ProbGraphs.
ProbGraph: Fast & Parallel Execution

\[ u \quad N_u \quad N_v \quad v \]

(different colors indicate different workers)

Bitwise AND

\[ |N_u \cap N_v| \]
ProbGraph: Fast & Parallel Execution

Embarrassingly parallel, O(1) depth

(different colors indicate different workers)
Approximate Graph Processing: Our Objectives

- Little parallelism → Rich parallelism
  - Specific → Wide applicability
  - No/loose accuracy guarantees → Strong accuracy guarantees
  - Slow → High performance
  - Low accuracy → High accuracy
  - Large memory overheads → Low & controllable memory overheads
Approximate Graph Processing: Our Objectives

- Little parallelism → Rich parallelism

- Specific → Wide applicability
- No/loose accuracy guarantees → Strong accuracy guarantees
- Slow → High performance
- Low accuracy → High accuracy
- Large memory overheads → Low & controllable memory overheads
Approximate Graph Processing: Our Objectives

- Little parallelism → Rich parallelism
- Specific → Wide applicability
- No/loose accuracy guarantees → Strong accuracy guarantees
- Slow → High performance
- Low accuracy → High accuracy
- Large memory overheads → Low & controllable memory overheads

High performance
Rich parallelism
Wide applicability
Strong accuracy guarantees
Approximate Graph Processing: Our Objectives

- Little parallelism → Rich parallelism
- Specific → Wide applicability
- No/loose accuracy guarantees → Strong accuracy guarantees
- Slow → High performance
- Low accuracy → High performance
- Large memory overheads → Low memory overheads

Let’s see 4 example use cases...
Use Case 1: Link Prediction

Which links will appear? Which links are missing?
Use Case 1: Link Prediction

Which links will appear?  
Which links are missing?
Use Case 1: Link Prediction

Which links will appear?

Which links are missing?
Use Case 1: Link Prediction

Which links will appear?  Which links are missing?

Predict future data

Fixing missing data
Use Case 1: Link Prediction

Which links will appear?
Which links are missing?

Fixing missing data

Predict future data
Use Case 2: Clique Counting
Use Case 2: Clique Counting
Use Case 2: Clique Counting
Use Case 2: Clique Counting

Learning over higher-order networks
Use Case 3: Clustering

# Clusters?

Structure of clusters?
Use Case 3: Clustering

# Clusters? Structure of clusters?
Use Case 3: Clustering

# Clusters?  Structure of clusters?

Minibatch selection in Graph Neural Networks
Use Case 4: Vertex Similarity
Use Case 4: Vertex Similarity
Use Case 4: Vertex Similarity

Enhancing graph embedding construction
Use Case 4: Vertex Similarity

Enhancing graph embedding construction
Use Case 4: Vertex Similarity

Enhancing graph embedding construction
Approximate Graph Processing: Our Objectives

- Little parallelism → Rich parallelism
- Specific → Wide applicability
- No/loose accuracy guarantees → Strong accuracy guarantees
- Slow → High performance
- Low accuracy → High accuracy
- Large memory overheads → Low & controllable memory overheads
Approximate Graph Processing: Our Objectives

- Little parallelism \rightarrow \text{Rich parallelism} with 
- Specific \rightarrow \text{Wide applicability}

- No/loose accuracy guarantees \rightarrow \text{Strong accuracy guarantees}
- Slow \rightarrow \text{High performance}
- Low accuracy \rightarrow \text{High accuracy}
- Large memory overheads \rightarrow \text{Low & controllable memory overheads}
Approximate Graph Processing: Our Objectives

- Little parallelism → Rich parallelism
- Specific → Wide applicability
- No/loose accuracy guarantees → Strong accuracy guarantees
- Slow → High performance
- Low accuracy → High accuracy
- Large memory overheads → Low & controllable memory overheads
ProbGraph: Summary of Theoretical Results
ProbGraph: Summary of Theoretical Results

We want guarantees for

$|\text{ProbGraphEstimate} - \text{exactResult}|$
ProbGraph: Summary of Theoretical Results

We want guarantees for

\[|\text{ProbGraphEstimate} - \text{exactResult}|\]

We incorporate statistical theory of estimators.
ProbGraph is asymptotically unbiased
ProbGraph is asymptotically unbiased

ProbGraph sketch size (storage needed)

Computation result
ProbGraph is asymptotically unbiased

Computation result

probGraph sketch size (storage needed)

exactResult
ProbGraph is asymptotically unbiased

\[ E[\text{ProbGraphEstimate}] \]

exactResult

ProbGraph sketch size (storage needed)

Computation result
ProbGraph is asymptotically unbiased

\[ E[ProbGraph\text{Estimate}] \]

The difference goes to zero

Computation result

ProbGraph sketch size (storage needed)
ProbGraph is asymptotically unbiased

Zero average error at some point… but the variance can still go wild

The difference goes to zero

$E[ProbGraphEstimate]$ 

exactResult

ProbGraph sketch size (storage needed)
ProbGraph is consistent

Computation result

ProbGraph sketch size (storage needed)

exactResult
ProbGraph is consistent

One can always find a ProbGraph sketch that delivers a required accuracy.

**exactResult**

ProbGraph sketch size (storage needed)

Computation result
ProbGraph is consistent

One can always find a ProbGraph sketch that delivers a required accuracy.
ProbGraph is **consistent**

The variance also converges to zero with the increasing sketch size.

One can **always** find a ProbGraph sketch that delivers a **required** accuracy.

---

**ProbGraphEstimate**

- Computation result
- exactResult

**ProbGraph sketch size (storage needed)**
ProbGraph is asymptotically efficient

![Graph](image-url)

- **ProbGraph sketch size (storage needed)**
- **Computation result**

**exactResult**
ProbGraph is asymptotically efficient

ProbGraph sketch size (storage needed)

Computation result

Other estimators [1-8]

exactResult

ProbGraph is asymptotically efficient

\[
\begin{align*}
\text{ProbGraphEstimate} & \quad \text{Other estimators} \\
\text{exactResult} & \quad \text{Computation result}
\end{align*}
\]

ProbGraph is asymptotically efficient

No other consistent estimator has lower MSE / variance

ProbGraph sketch size (storage needed)

Computation result

ProbGraphEstimate

Other estimators

exactResult

ProbGraph has strong concentration bounds
ProbGraph has **strong concentration bounds**

Deviation $t$ from the real value
ProbGraph has strong concentration bounds

\[ P(|\text{ProbGraphEstimate|} \leq \alpha \text{ from the real value} \right) \]
ProbGraph has **strong concentration bounds**

\[ P(|\text{ProbGraphEstimate}|) \]

This probability decreases **exponentially fast**

Deviation \( t \) from the real value
ProbGraph has **strong concentration bounds**

\[ P(|\text{ProbGraphEstimate}| \leq \Delta) \]

- This probability decreases exponentially fast.
- ProbGraph is unlikely to deviate much from the true values.
Approximate Graph Processing: Our Objectives

- Little parallelism → Rich parallelism
- Specific → Wide applicability
- No/loose accuracy guarantees → Strong accuracy guarantees
- Slow → High performance
- Low accuracy → High accuracy
- Large memory overheads → Low & controllable memory overheads
Approximate Graph Processing: Our Objectives

- Little parallelism vs. Rich parallelism
- Specific vs. Wide applicability
- No/loose accuracy guarantees vs. Strong accuracy guarantees
- Slow vs. High performance
- Low accuracy vs. High accuracy
- Large memory overheads vs. Low & controllable memory overheads
Approximate Graph Processing: Our Objectives

- Little parallelism → Rich parallelism
- Specific → Wide applicability
- No/loose accuracy guarantees → Strong accuracy guarantees
- Slow → High performance
- Low accuracy → High accuracy
- Large memory overheads → Low & controllable memory overheads
Evaluation: Used Machines & Objectives
Evaluation: Used Machines & Objectives

CSCS Cray Piz Daint,
64 GB per compute node
Evaluation: Used Machines & Objectives

CSCS Cray Piz Daint, 64 GB per compute node
Goal: One design with...

- **large** speedups +
- **small & controlled** accuracy loss +
- **small & controlled** memory requirements

CSCS Cray Piz Daint,
64 GB per compute node
Considered Graph Datasets
Considered Graph Datasets

67 graph datasets,
15 areas,
5 major graph
dataset repositories
Considered Graph Datasets

67 graph datasets,
15 areas,
5 major graph dataset repositories

Real-world graphs

Social networks

Purchases

Gene functions

Communication

Citation graphs

Compute graphs

Economic nets

Synthetic graphs

Kronecker [1]
Erdös-Rényi [2]

67 graph datasets, 15 areas, 5 major graph dataset repositories

Real-world graphs

Social networks

Purchases

Gene functions

Communication

Citation graphs

Compute graphs

Economic nets

Synthetic graphs

Kronecker [1]
Erdös-Rényi [2]

Considered Graph Datasets

- 67 graph datasets
- 15 areas
- 5 major graph dataset repositories

**Real-world graphs**
- Purchases
- Gene functions
- Brain structure
- Communication
- Citation graphs
- Compute graphs
- Economic nets
- Mathematics
- Medicine
- Chemistry
- Purchases
- Gene functions
- Brain structure
- Communication
- Citation graphs
- Compute graphs
- Economic nets
- Mathematics
- Medicine
- Chemistry

**Synthetic graphs**
- Kronecker [1]
- Erdös-Rényi [2]

**Highly irregular data**
**Lots of load imbalance**

Triangle Counting
Cores/threads: 32
Max memory overhead: 20%
Triangle Counting

Cores/threads: 32
Max memory overhead: 20%
Triangle Counting

Cores/threads: 32
Max memory overhead: 20%

\[ \text{Speedup over the exact tuned baseline} \]
Triangle Counting

Cores/threads: 32
Max memory overhead: 20%

Accuracy: Relative count of a given pattern

Speedup over the exact tuned baseline
Triangle Counting

Cores/threads: 32
Max memory overhead: 20%

Each data point: the execution of a given scheme for a specific graph dataset
Triangle Counting
Cores/threads: 32
Max memory
everhead: 20%

Each data point: the execution of a given scheme for a specific graph dataset

Triangle Counting

Cores/threads: 32
Max memory overhead: 20%

Each data point: the execution of a given scheme for a specific graph dataset

ProbGraph

Exact baseline [1]

Triangle Counting

Cores/threads: 32
Max memory overhead: 20%

Each data point: the execution of a given scheme for a specific graph dataset

- **ProbGraph**
- **Exact baseline [1]**
- **Heuristics, no formal guarantees [2]**

Triangle Counting
Cores/threads: 32
Max memory overhead: 20%

Each data point: the execution of a given scheme for a specific graph dataset

- ProbGraph
- Exact baseline [1]
- Heuristics, no formal guarantees [2]
- Heuristics, formal guarantees [3-4]

---

1. S. Beamer et al., „The GAP Benchmark Suite“. 2015
2. S. Singh et al., “Scalable and performant graph processing on GPUs using approximate computing”. IEEE TMSCS. 2018
Triangle Counting

Cores/threads: 32
Max memory overhead: 20%

Each data point: the execution of a given scheme for a specific graph dataset

- **ProbGraph**
- Exact baseline [1]
- Heuristics, no formal guarantees [2]
- Heuristics, formal guarantees [3-4]
- Lossy graph compression [5-6]

---

Triangle Counting

Cores/threads: 32
Max memory overhead: 20%

Each data point: the execution of a given scheme for a specific graph dataset

For most graphs, we have...
...very high speedups

Triangle Counting

Cores/threads: 32
Max memory overhead: 20%

Each data point: the execution of a given scheme for a specific graph dataset

For most graphs, we have...
- very high speedups
- very good accuracy

ProbGraph

Exact baseline [1]

Heuristics, no formal guarantees [2]

Heuristics, formal guarantees [3-4]

Lossy graph compression [5-6]

1. S. Beamer et al., „The GAP Benchmark Suite“. 2015
2. S. Singh et al., “Scalable and performant graph processing on GPUs using approximate computing”. IEEE TMSCS. 2018
Triangle Counting

Cores/threads: 32
Max memory overhead: 20%

For most graphs, we have...
...very high speedups
...very good accuracy
...mild memory requirements

Each data point: the execution of a given scheme for a specific graph dataset

- ProbGraph
- Exact baseline [1]
- Heuristics, no formal guarantees [2]
- Heuristics, formal guarantees [3-4]
- Lossy graph compression [5-6]

80% accuracy

4-Clique Counting

Cores/threads: 32
Max memory overhead: 20%

Each data point: the execution of a given scheme for a specific graph dataset

For most graphs, we have...
...very high speedups
...very good accuracy
...mild memory requirements

[1] based on S. Beamer et al., „The GAP Benchmark Suite“. 2015
4-Clique Counting

Cores/threads: 32
Max memory overhead: 20%

Each data point: the execution of a given scheme for a specific graph dataset

For most graphs, we have...

...very high speedups
...very good accuracy
...mild memory requirements

[1] based on S. Beamer et al., „The GAP Benchmark Suite“. 2015
Clustering (Scaling)

Max memory overhead: 20%

Clustering (Scaling)

Max memory overhead: 20%

Clustering (Scaling)

Max memory overhead: 20%

Clustering (Scaling)

Max memory overhead: 20%

Clustering (Scaling)

Max memory overhead: 20%

Clustering (Scaling)

Max memory overhead: 20%

Clustering (Scaling)

Max memory overhead: 20%

Why do we scale so well?

---

Clustering (Scaling)

Max memory overhead: 20%

Why do we scale so well?

Great load balancing properties

---

ProbGraph representation

Bloom filters
ProbGraph representation

Bloom filters

∪

1 2 4 5 6
3 4 6 7 8

∪

1 2 3
2 3 4
5 6 7 8

∪

1 2 3
2 3 4
5 6 7 8

∩

5 6 8
3 4 5 7
5 6 8

∩

2 3 4 5 6
...Many more data & a lot of strong theory results!
Approximate Graph Processing: Our Objectives

- Little parallelism → Rich parallelism
- Specific → Wide applicability
- No/loose accuracy guarantees → Strong accuracy guarantees
- Slow → High performance
- Low accuracy → High accuracy
- Large memory overheads → Low & controllable memory overheads
Approximate Graph Processing: Our Objectives

- Little parallelism → Rich parallelism
- Specific → Wide applicability
- No/loose accuracy guarantees → Strong accuracy guarantees
- Slow → High performance
- Low accuracy → High accuracy
- Large memory overheads → Low & controllable memory overheads
Conclusion: ProbGraph Enables Approximate Graph Mining with...

- Rich parallelism
- Wide applicability
- Strong accuracy guarantees
- High performance
- High accuracy
- Low & controllable memory overheads
Conclusion: ProbGraph Enables Approximate Graph Mining with...

- Rich parallelism
- Wide applicability
- Strong accuracy guarantees
- High performance
- High accuracy
- Low & controllable memory overheads
Conclusion: ProbGraph Enables Approximate Graph Mining with...

- Rich parallelism
- Wide applicability
- Strong accuracy guarantees
- High performance
- High accuracy
- Low & controllable memory overheads

Thank you
Conclusion: ProbGraph Enables Approximate Graph Mining with...

- Rich parallelism
- Wide applicability
- Strong accuracy guarantees
- High performance
- High accuracy
- Low & controllable memory overheads

Thank you

Want to know more?

- youtube.com/@spcl
- twitter.com/spcl_eth
- spcl.inf.ethz.ch
- github.com/spcl
Conclusion: ProbGraph Enables Approximate Graph Mining with...

- Rich parallelism
- Wide applicability
- Strong accuracy guarantees
- High performance
- High accuracy
- Low & controllable memory overheads

Backup slides

Want to know more?
- youtube.com/@spcl
- twitter.com/spcl_eth
- spcl.inf.ethz.ch
- github.com/spcl