Towards Fast and Scalable Graph Pattern Mining

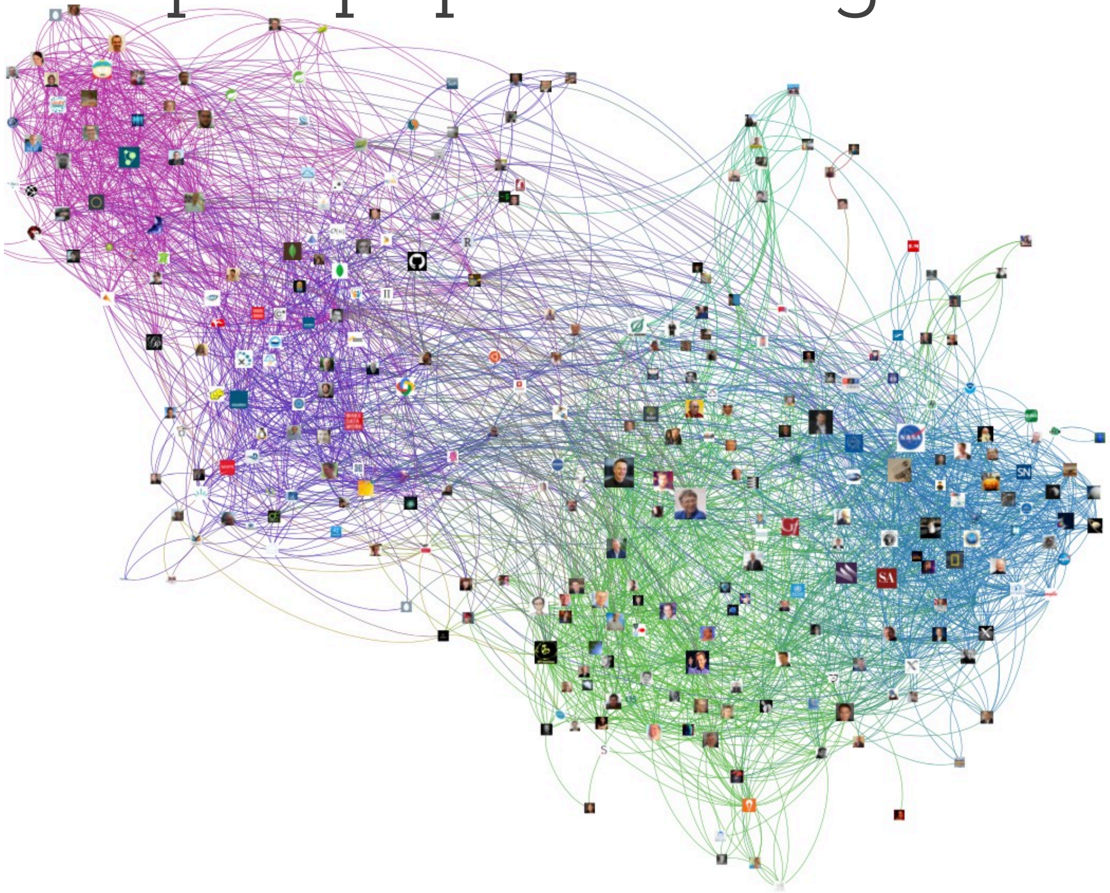
Anand Iyer [★], *Zaoxing Liu* [◆], *Xin Jin* [◆],
Shivaram Venkataraman [▲], *Vladimir Braverman* [◆], *Ion Stoica* [★]

[★] UC Berkeley [◆] Johns Hopkins University [▲] Microsoft Research / University of Wisconsin

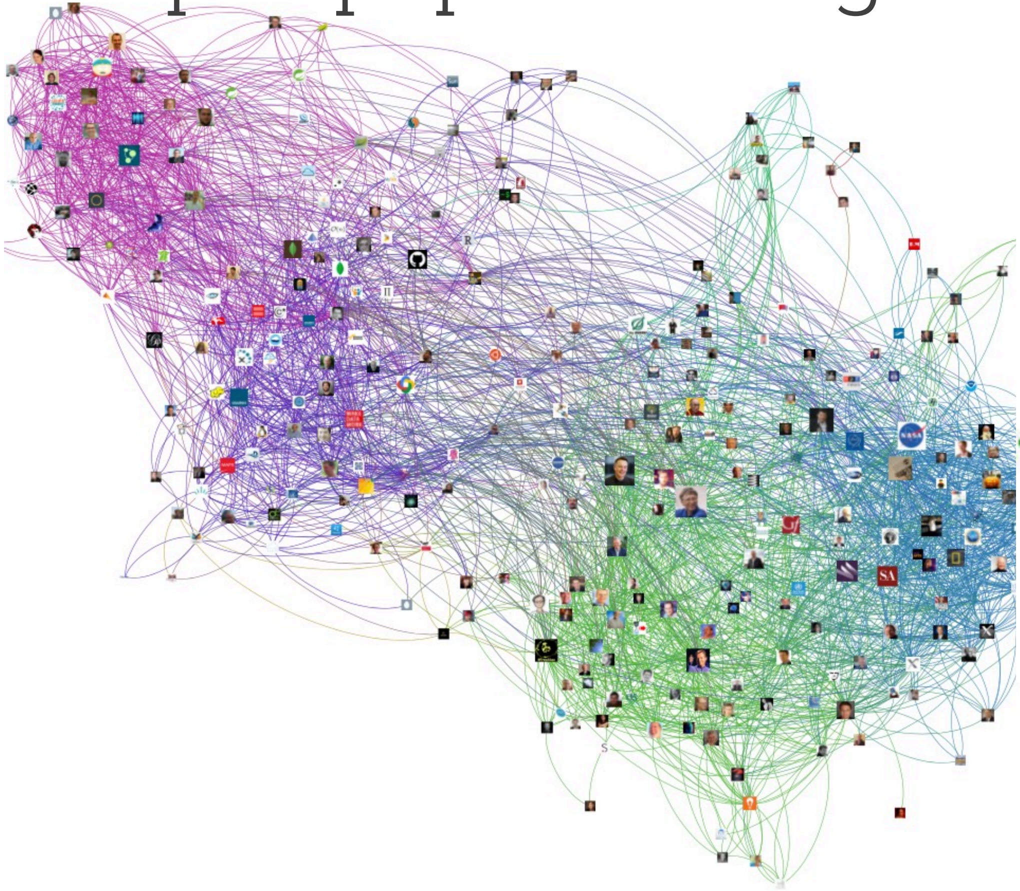
HotCloud, July 09, 2018

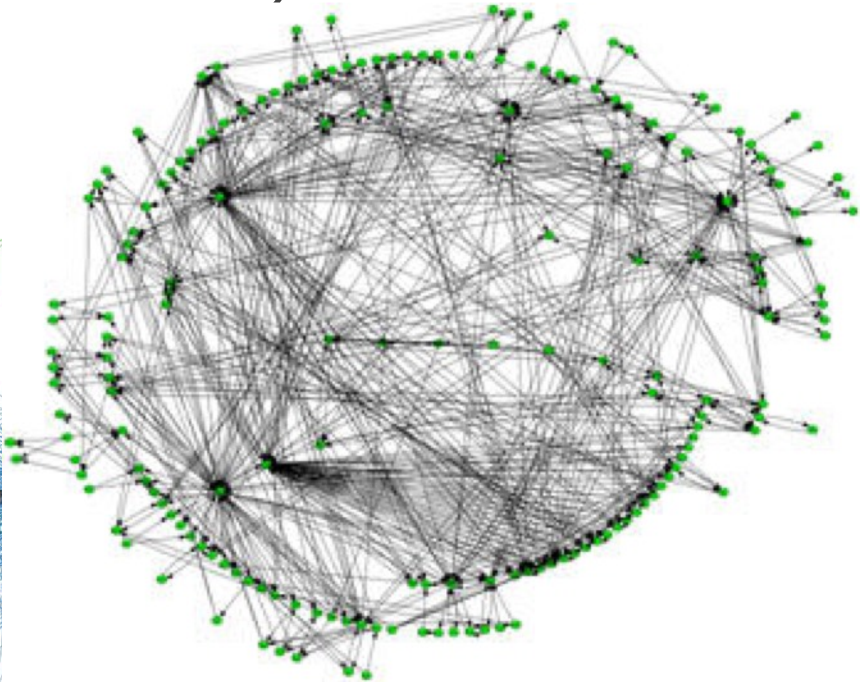
riselab UC Berkeley

Graphs popular in big data analytics

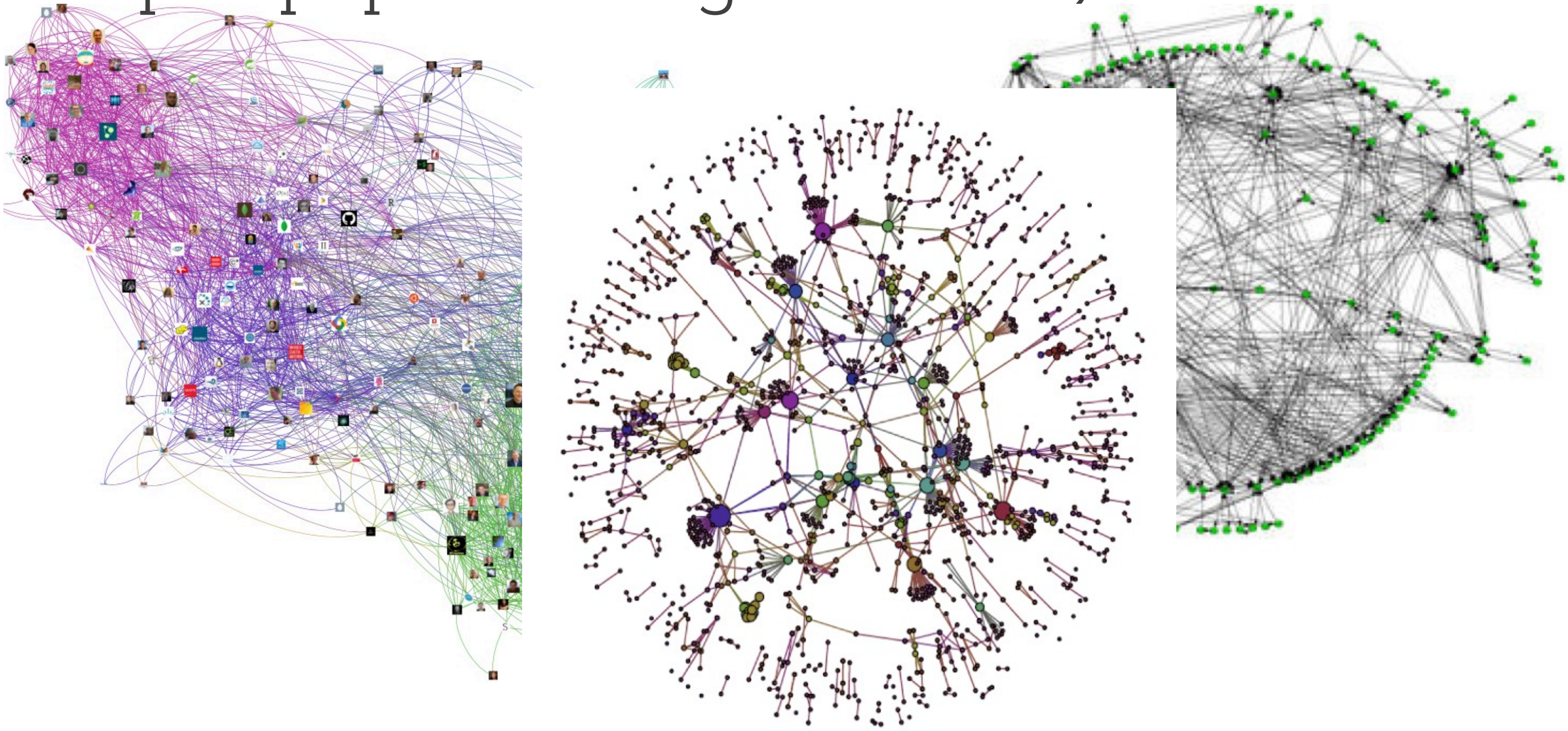


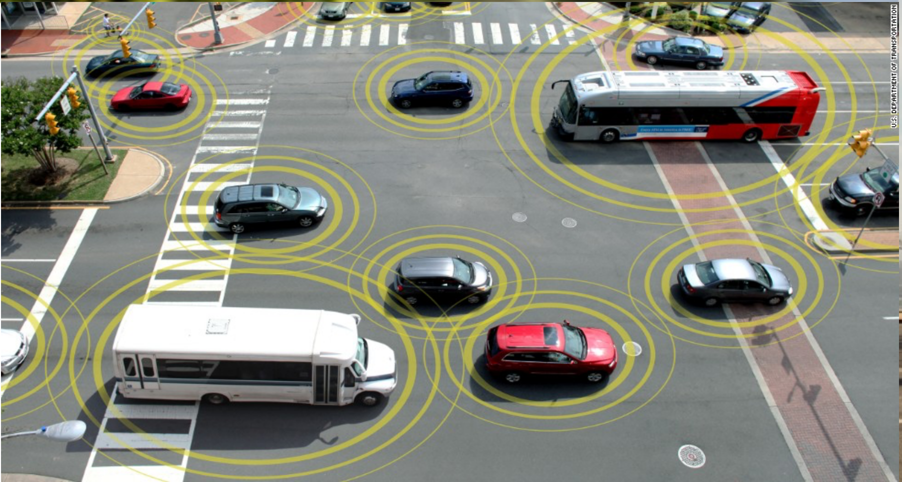
Graphs popular in big data analytics





Graphs popular in big data analytics




U.S. DEPARTMENT OF TRANSPORTATION


Google
self-driving car

Graph Analytics

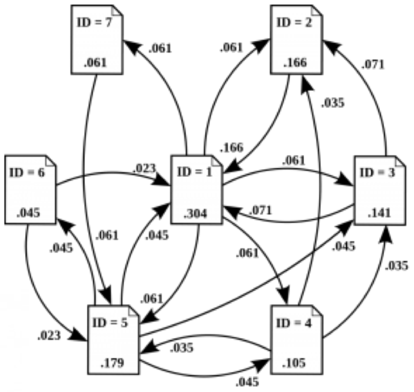
Graph Analytics

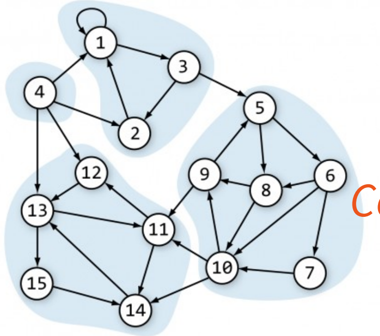
Processing Algorithms

Graph Analytics

Processing Algorithms

PageRank



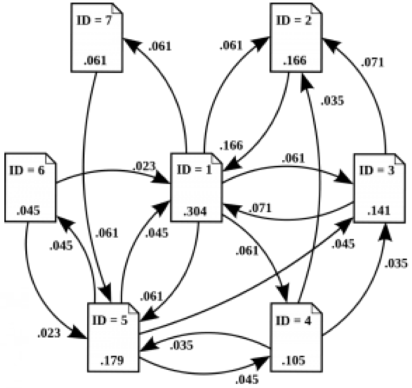


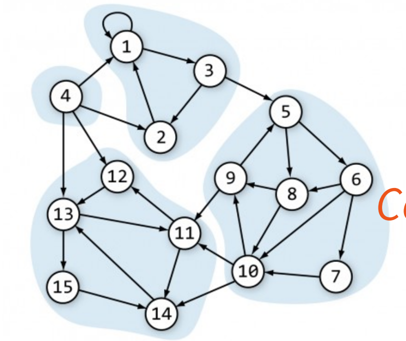
Connected Components

Graph Analytics

Processing Algorithms

PageRank





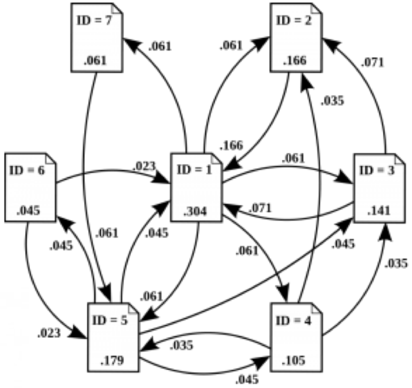
Connected Components

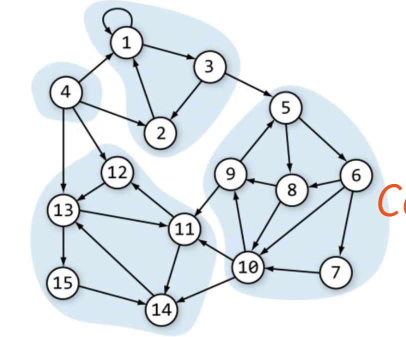
Mining Algorithms

Graph Analytics

Processing Algorithms

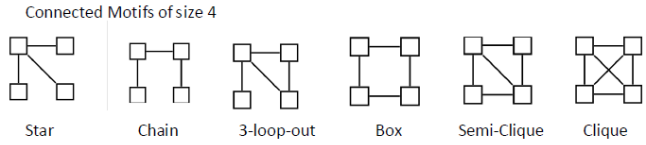
PageRank

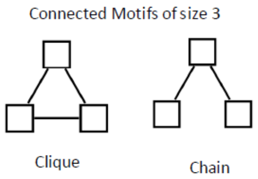




Connected Components

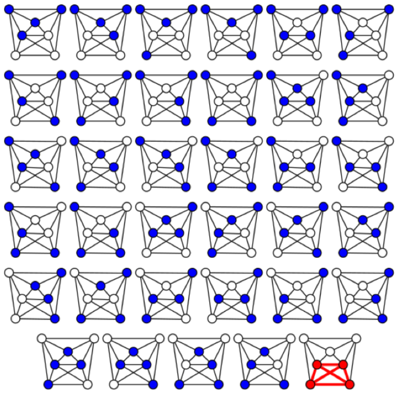
Mining Algorithms





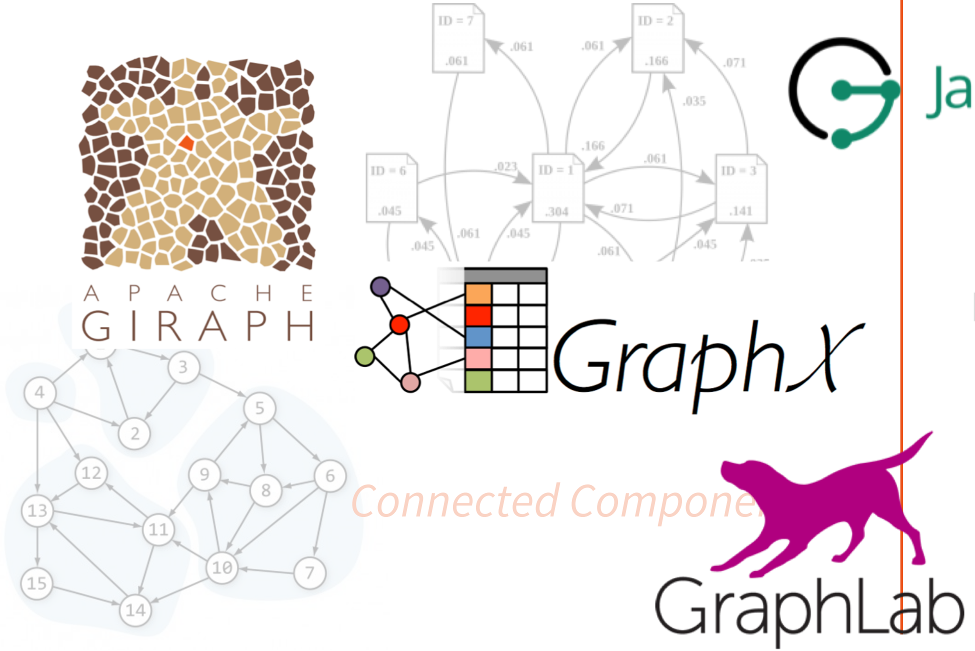
Motifs

Cliques

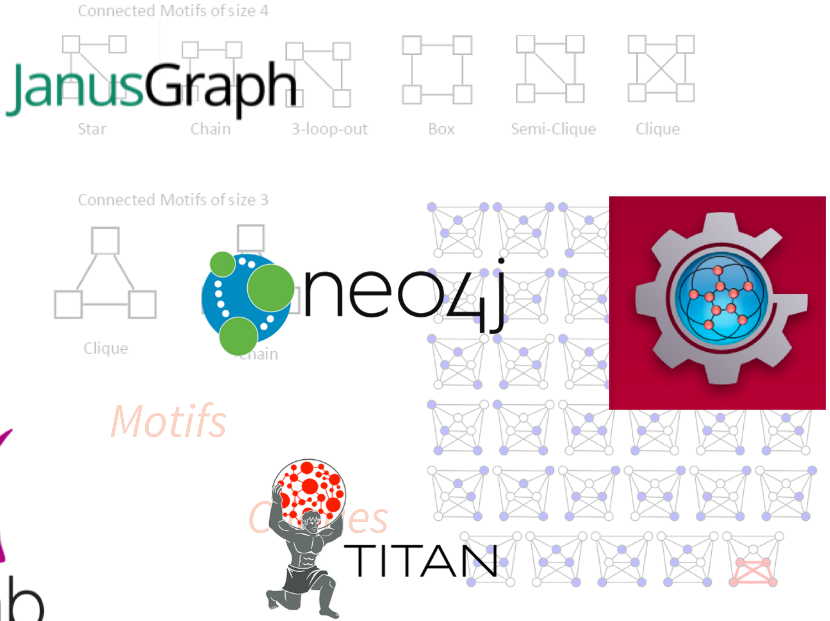


Graph Analytics

Processing Algorithms



Mining Algorithms



Graph Analytics: State-of-the-Art

Processing Algorithms

Computes properties of the underlying graph

Mining Algorithms

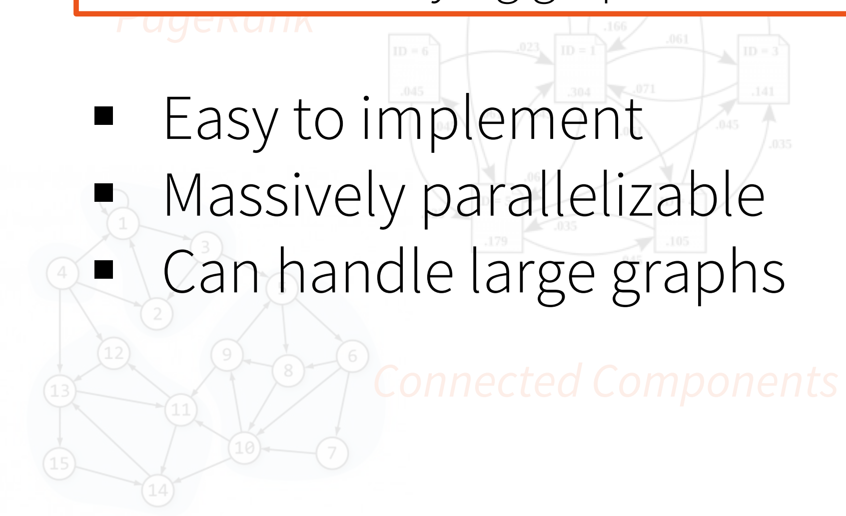
Discovers structural patterns in the underlying graph

Graph Analytics: State-of-the-Art

Processing Algorithms

Computes properties of the underlying graph

- Easy to implement
- Massively parallelizable
- Can handle large graphs



Mining Algorithms

Discovers structural patterns in the underlying graph

Graph Analytics: State-of-the-Art

Processing Algorithms

Computes properties of the underlying graph

- Easy to implement
- Massively parallelizable
- Can handle large graphs

Mining Algorithms

Discovers structural patterns in the underlying graph

- Efficient custom algorithms
- Exponential intermediate data
- Limited to small graphs

Graph Analytics: State-of-the-Art

Processing Algorithms

Computes properties of the underlying graph

- Easy to implement
- Massively parallelizable
- Can handle large graphs

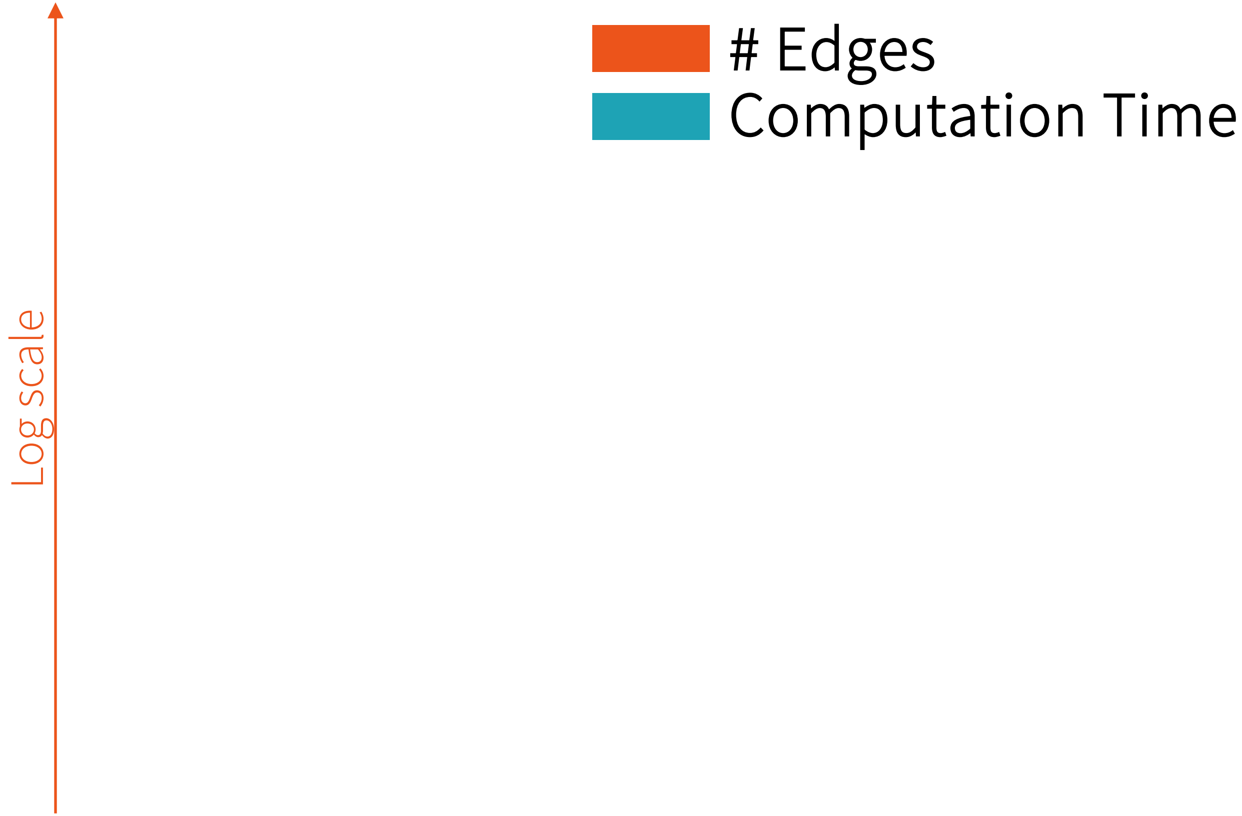
Mining Algorithms

Discovers structural patterns in the underlying graph

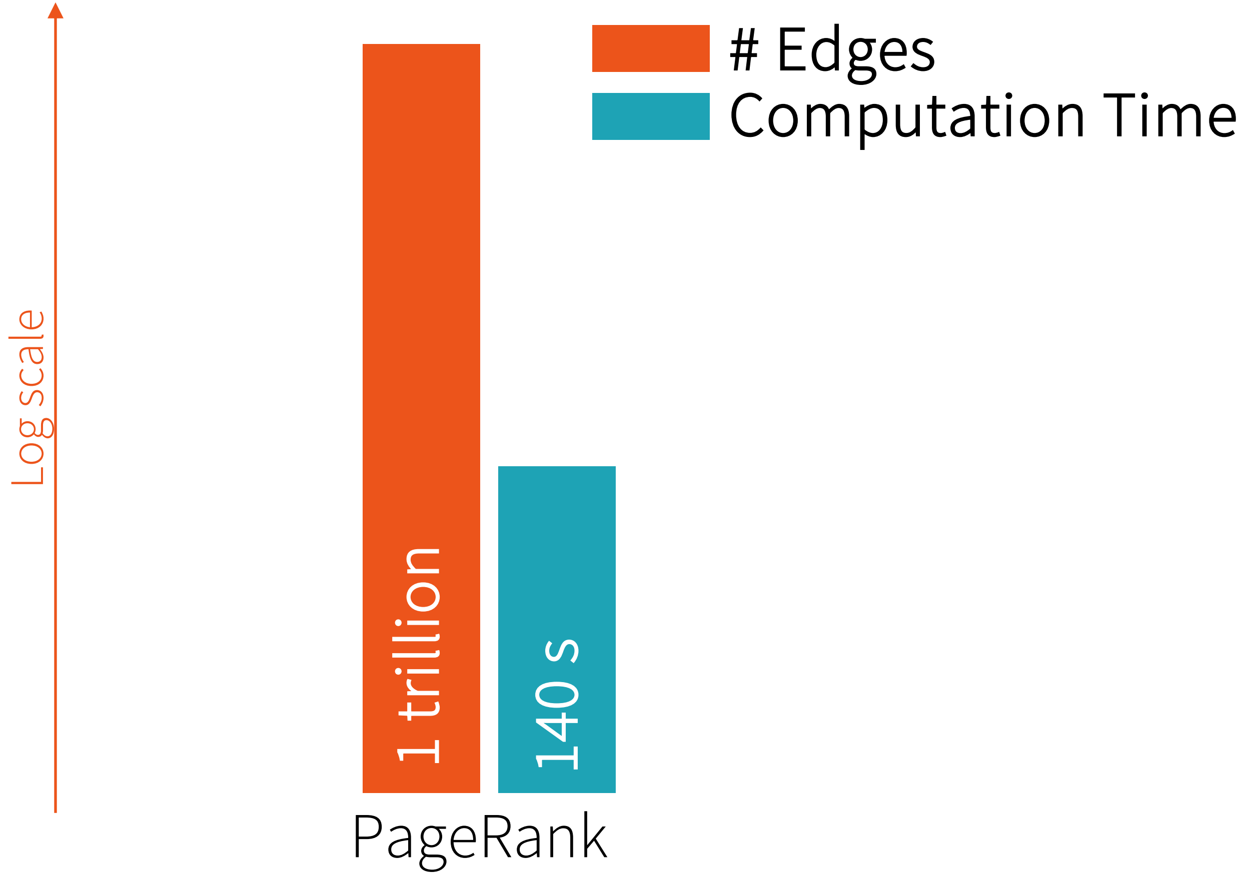
- Efficient custom algorithms
- Exponential intermediate data
- Limited to small graphs

Challenging to mine patterns in large graphs

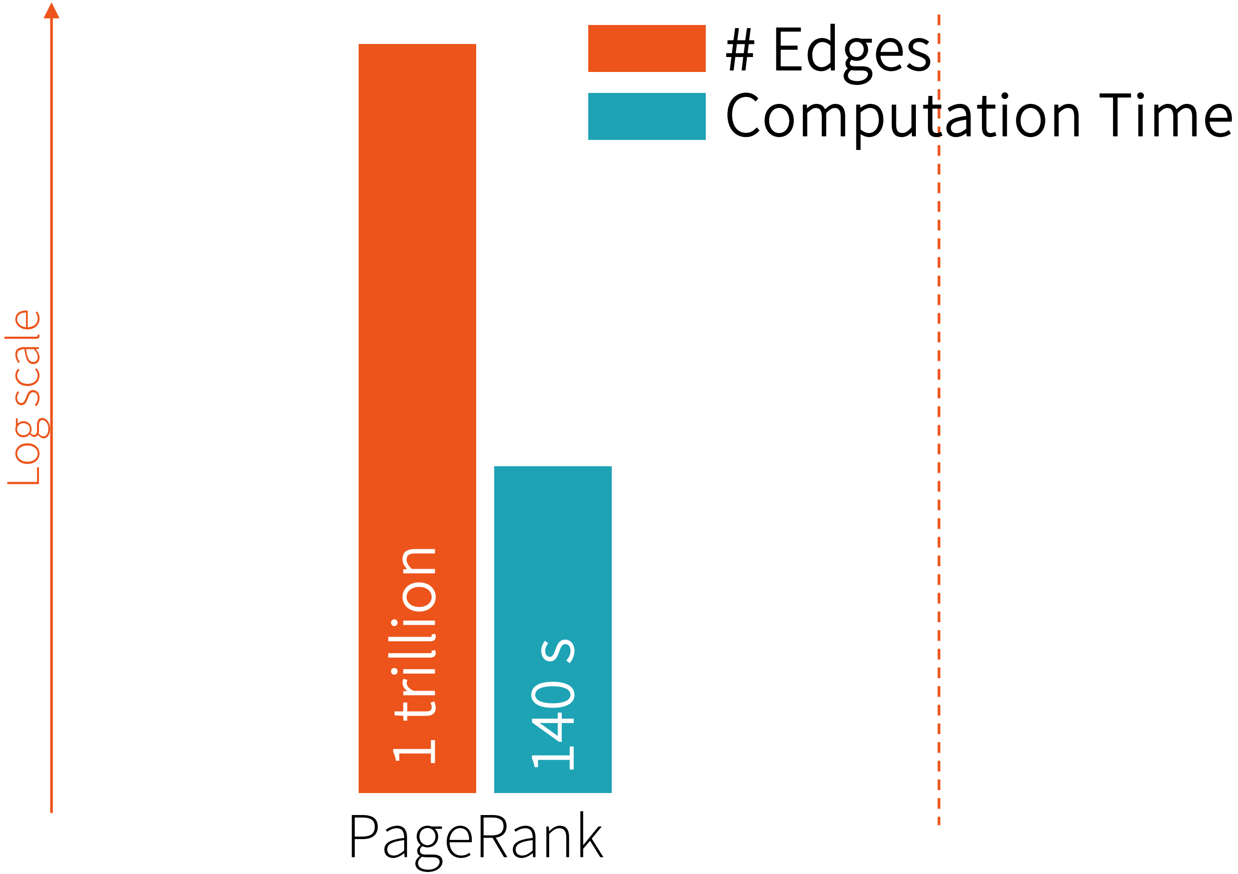
Graph Analytics: Processing vs Mining



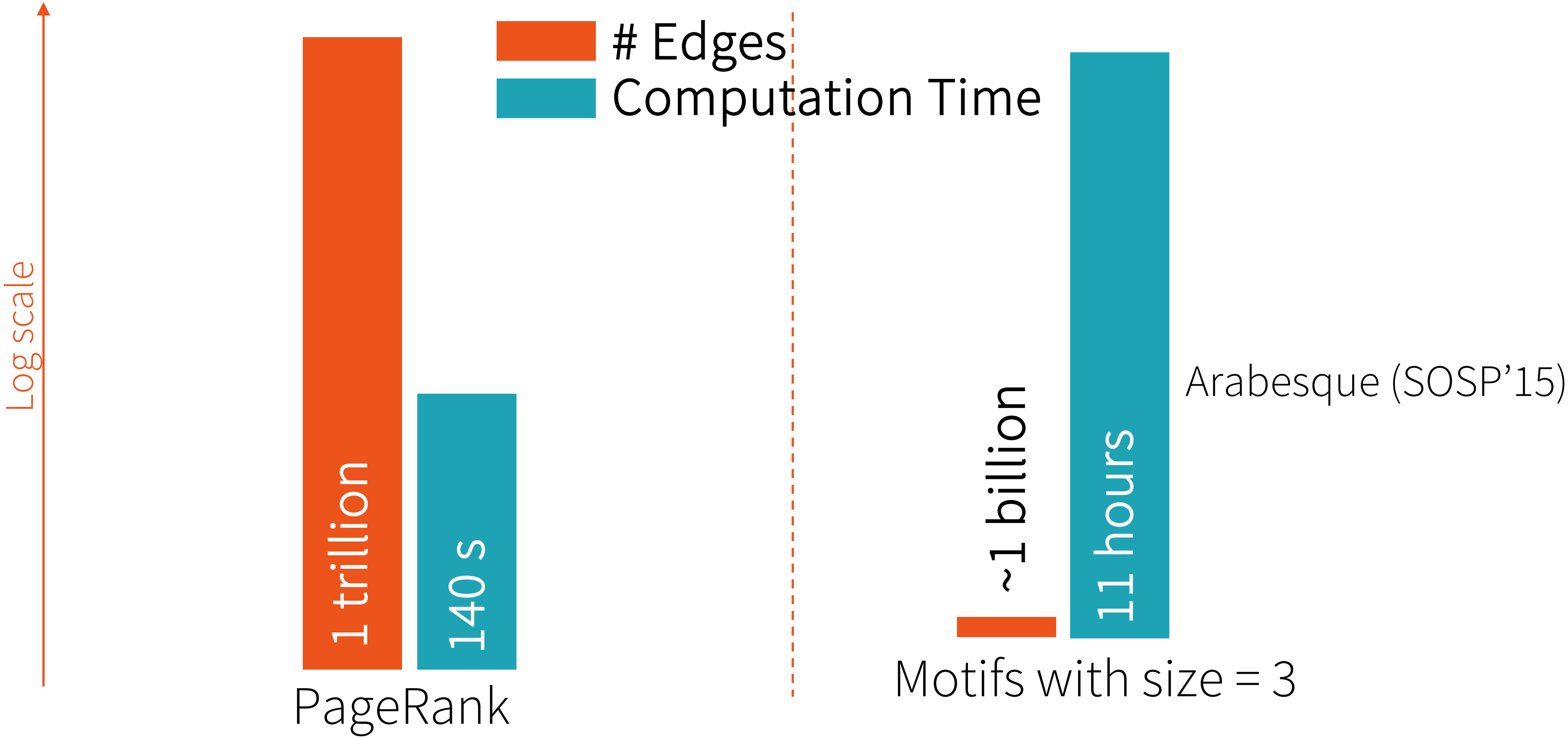
Graph Analytics: Processing vs Mining



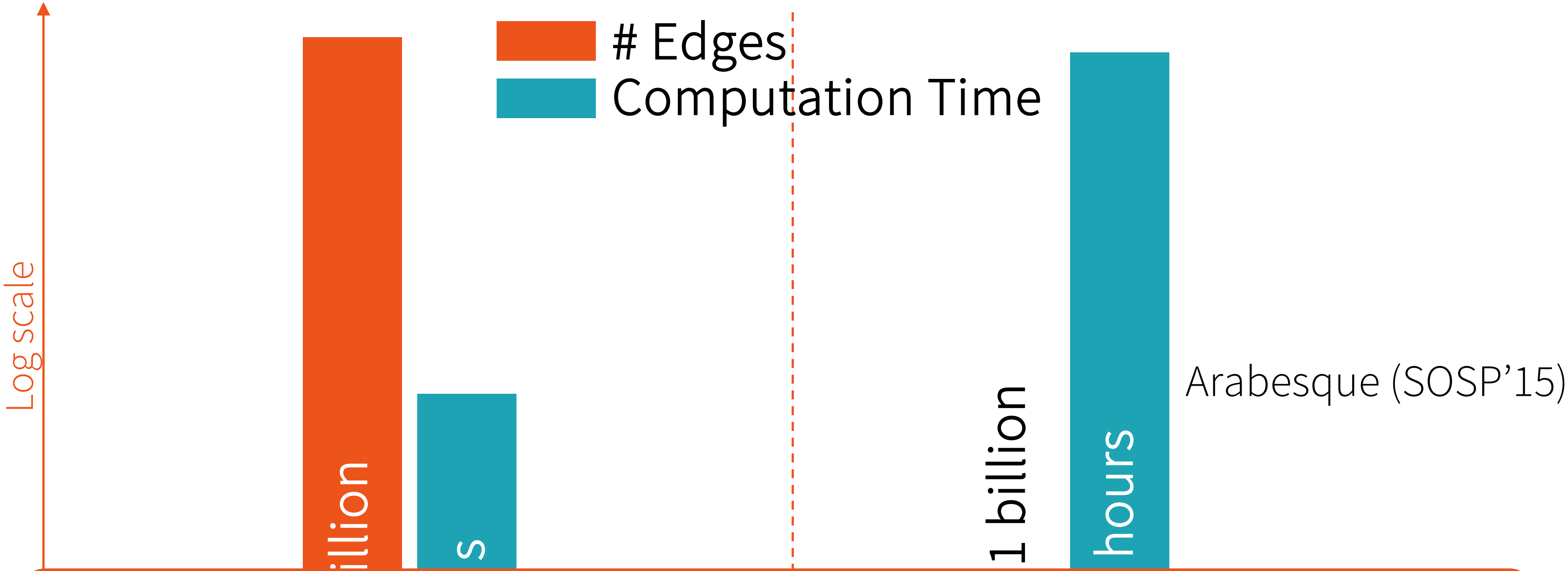
Graph Analytics: Processing vs Mining



Graph Analytics: Processing vs Mining



Graph Analytics: Processing vs Mining



Can graph pattern mining be made both *fast* and *scalable*?

Many mining tasks ask for the number of occurrences and do not need *exact* answers

Many mining tasks ask for the number of occurrences and do not need *exact* answers

Leverage *approximation* for graph pattern mining

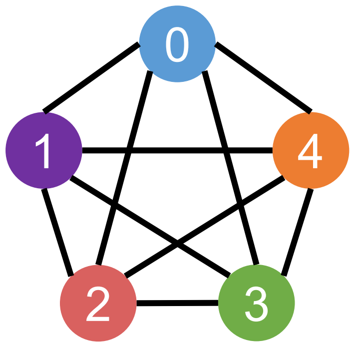
Approximate Analytics

General approach: Apply algorithm on subset(s) (sample) of the input data

Approximate Analytics

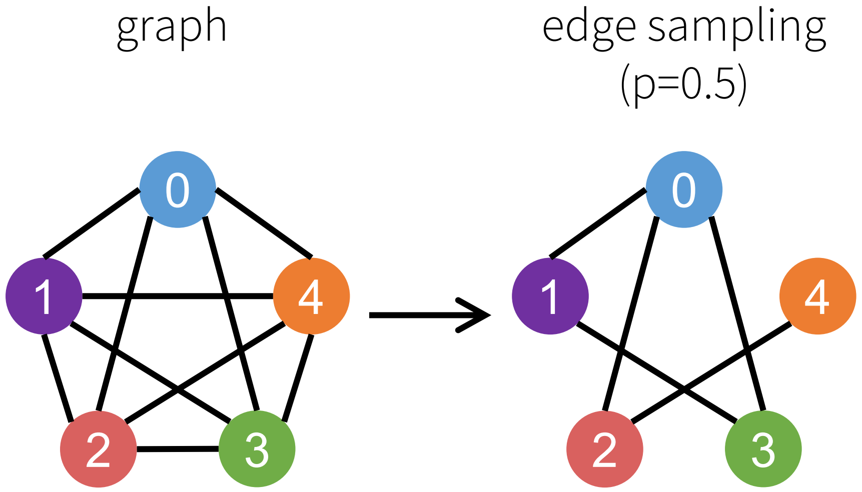
General approach: Apply algorithm on subset(s) (sample) of the input data

graph



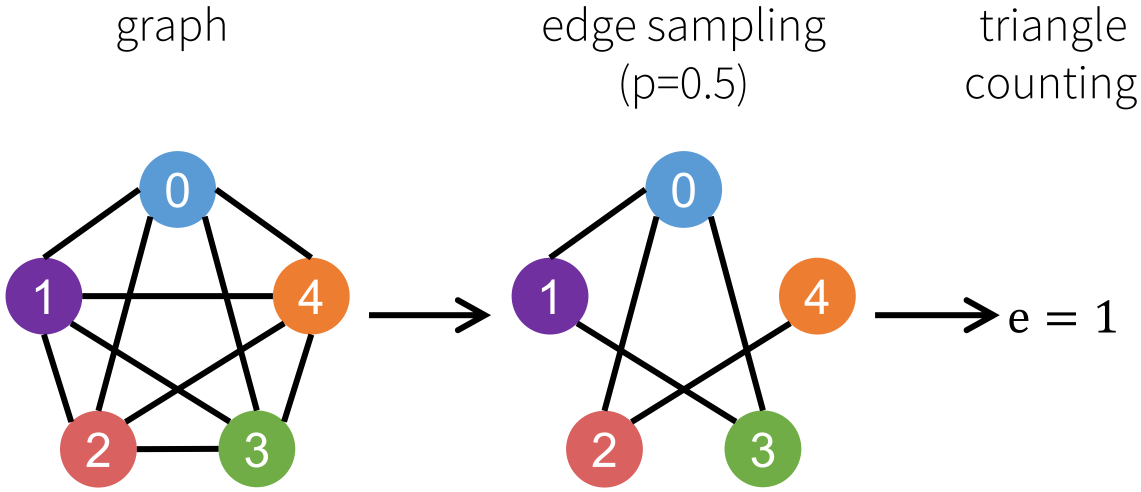
Approximate Analytics

General approach: Apply algorithm on subset(s) (sample) of the input data



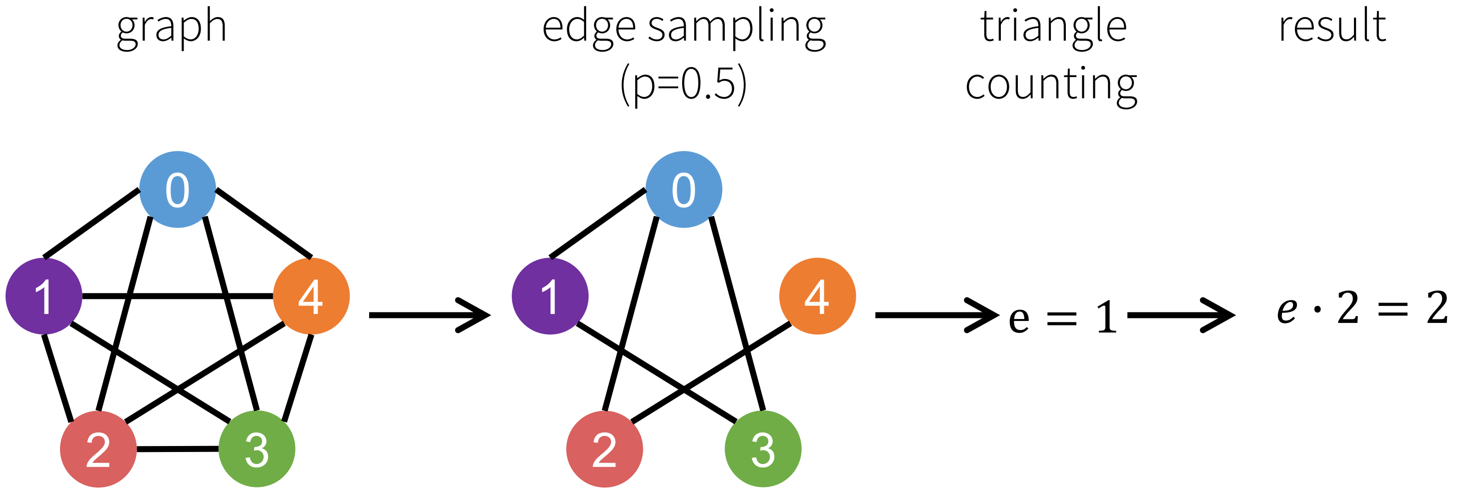
Approximate Analytics

General approach: Apply algorithm on subset(s) (sample) of the input data



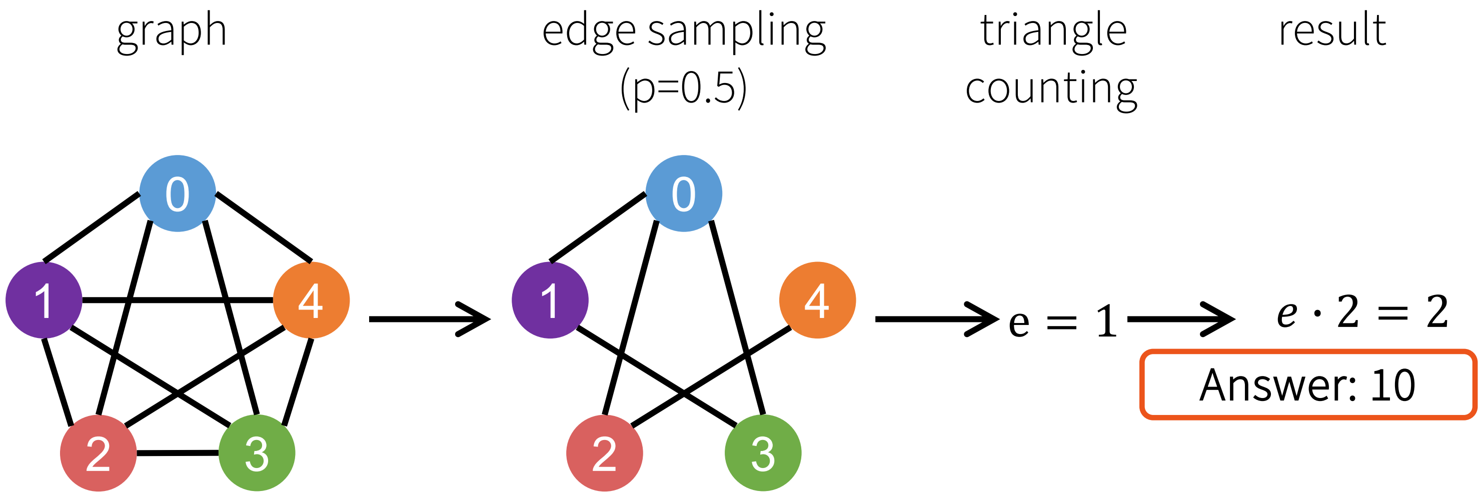
Approximate Analytics

General approach: Apply algorithm on subset(s) (sample) of the input data



Approximate Analytics

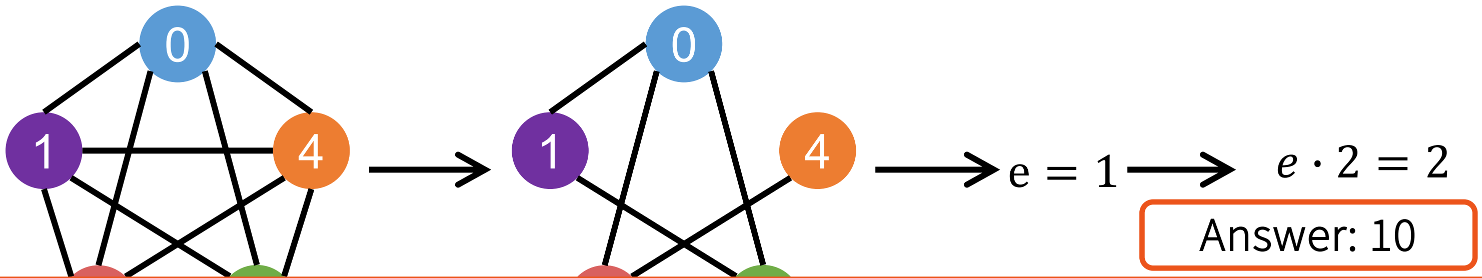
General approach: Apply algorithm on subset(s) (sample) of the input data



Approximate Analytics

General approach: Apply algorithm on subset(s) (sample) of the input data

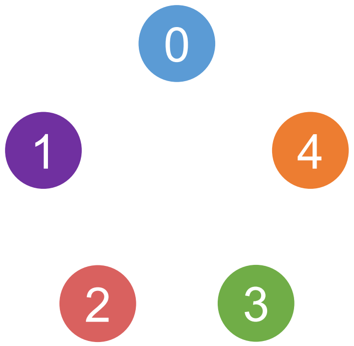
graph | edge sampling (p=0.5) | triangle counting | result



Applying *exact* algorithm on *sampled* graph(s) not the right approach for pattern mining

Approximation by Sampling Patterns

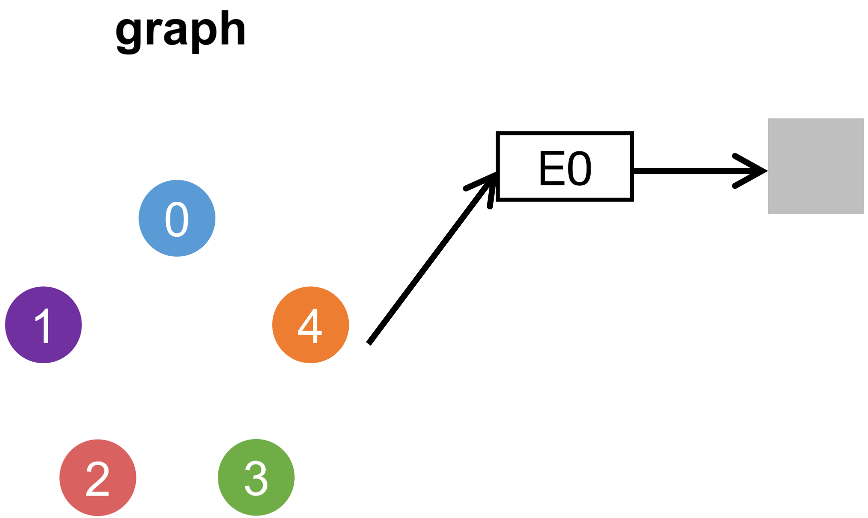
graph



edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Pavan et al. Counting and sampling triangles from a graph stream, VLDB 2013

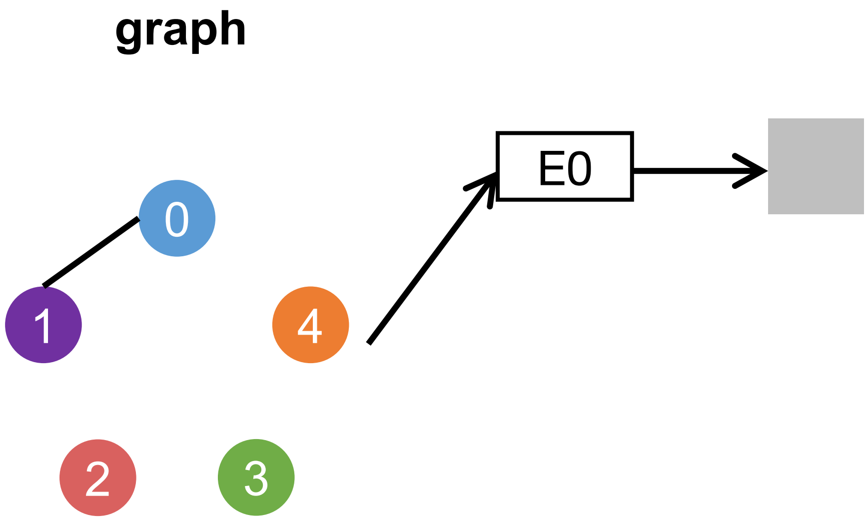
Approximation by Sampling Patterns



edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Pavan et al. Counting and sampling triangles from a graph stream, VLDB 2013

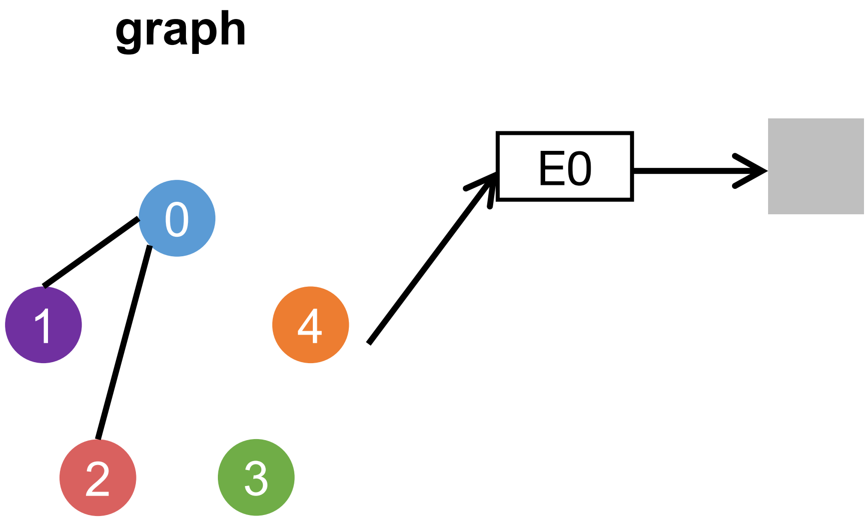
Approximation by Sampling Patterns



edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Pavan et al. Counting and sampling triangles from a graph stream, VLDB 2013

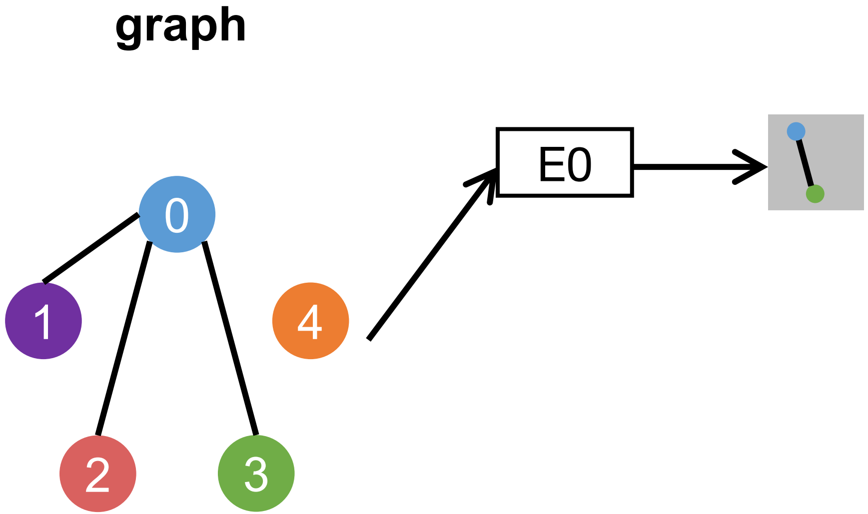
Approximation by Sampling Patterns



edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Pavan et al. Counting and sampling triangles from a graph stream, VLDB 2013

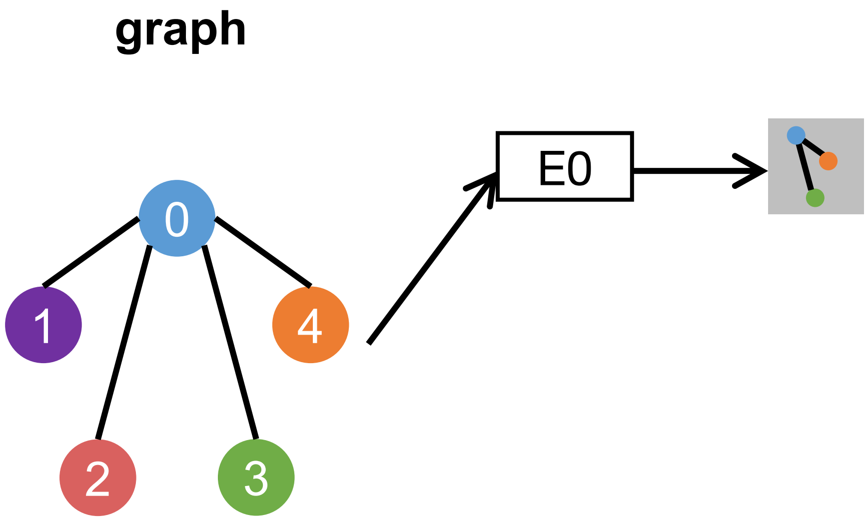
Approximation by Sampling Patterns



edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Pavan et al. Counting and sampling triangles from a graph stream, VLDB 2013

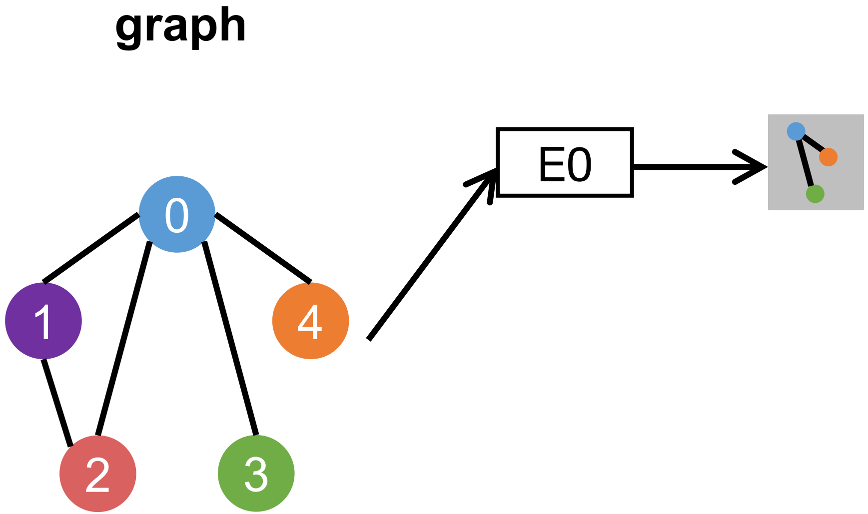
Approximation by Sampling Patterns



edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Pavan et al. Counting and sampling triangles from a graph stream, VLDB 2013

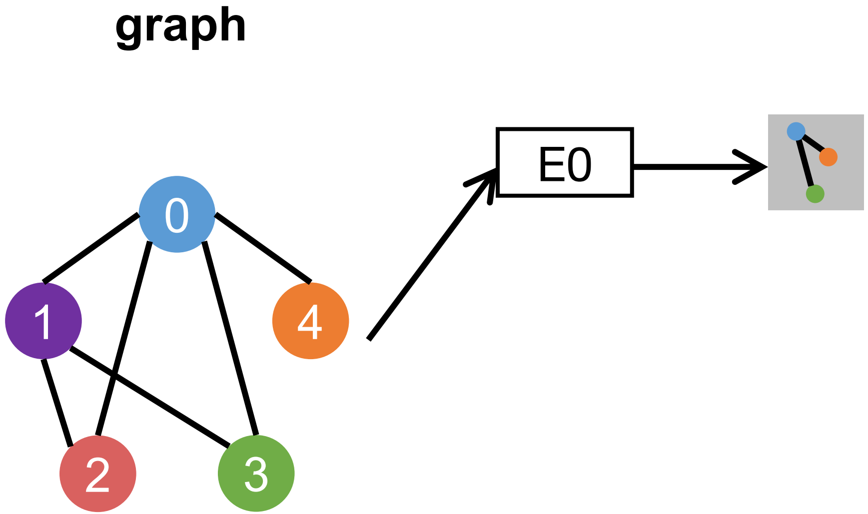
Approximation by Sampling Patterns



edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Pavan et al. Counting and sampling triangles from a graph stream, VLDB 2013

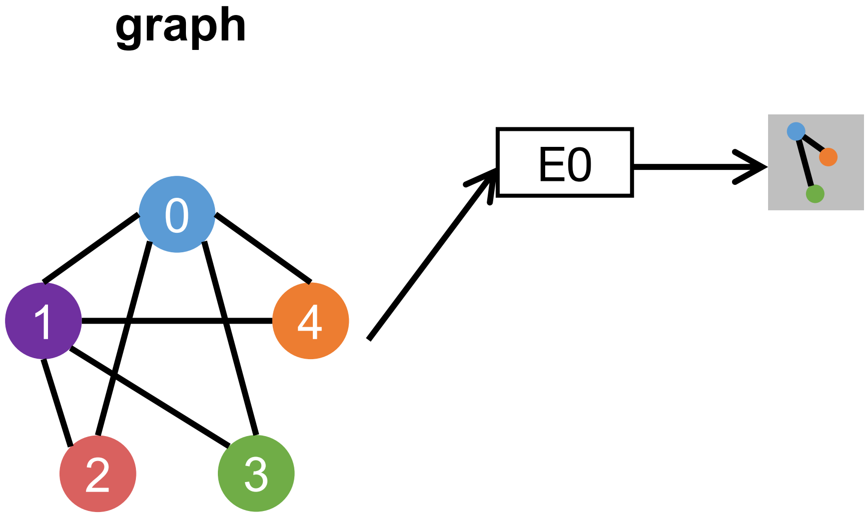
Approximation by Sampling Patterns



edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Pavan et al. Counting and sampling triangles from a graph stream, VLDB 2013

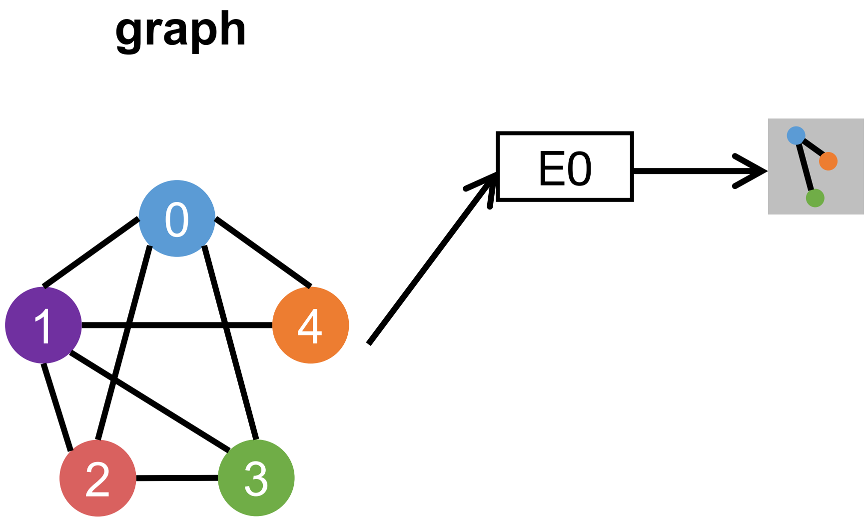
Approximation by Sampling Patterns



edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Pavan et al. Counting and sampling triangles from a graph stream, VLDB 2013

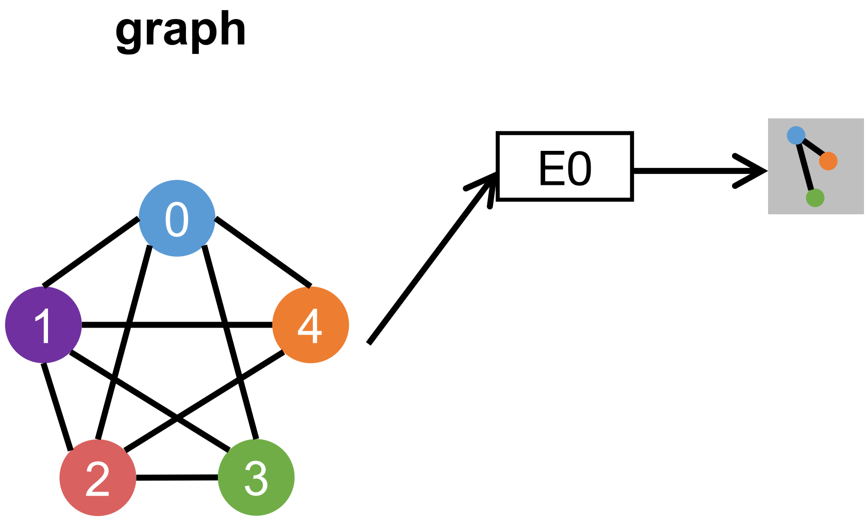
Approximation by Sampling Patterns



edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Pavan et al. Counting and sampling triangles from a graph stream, VLDB 2013

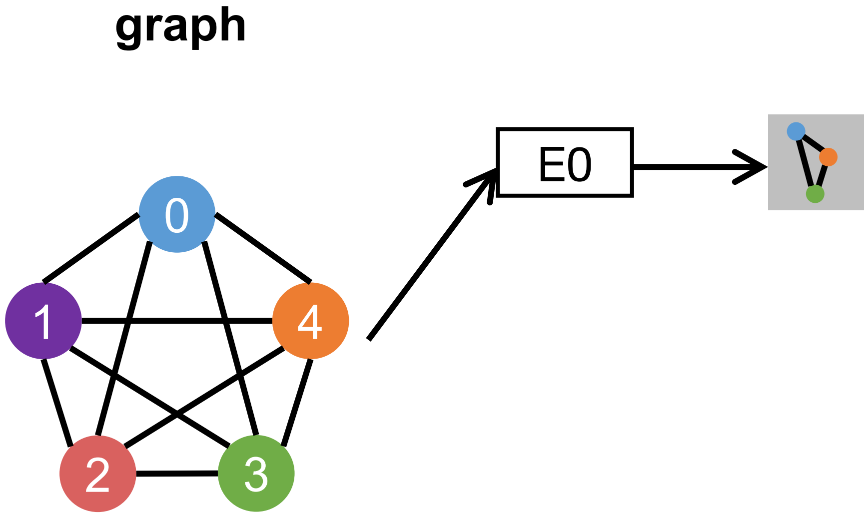
Approximation by Sampling Patterns



edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Pavan et al. Counting and sampling triangles from a graph stream, VLDB 2013

Approximation by Sampling Patterns



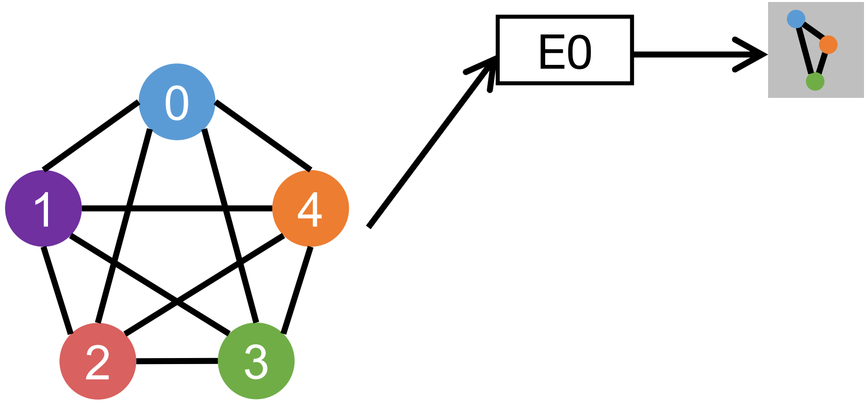
edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Pavan et al. Counting and sampling triangles from a graph stream, VLDB 2013

Approximation by Sampling Patterns

graph

$$p = \frac{1}{10} * \frac{1}{4}$$

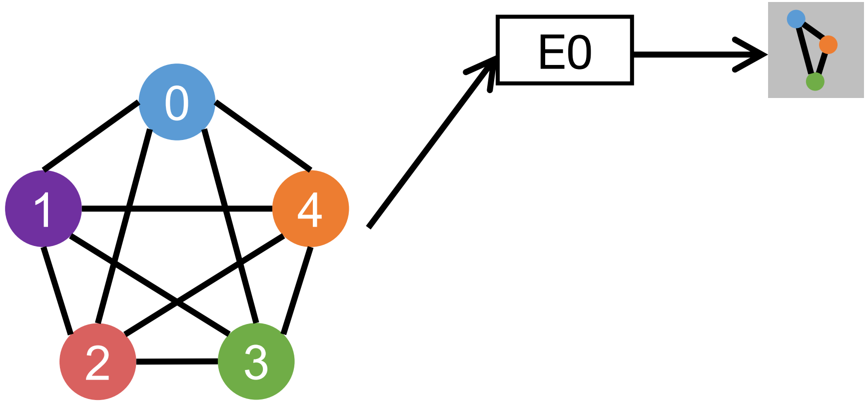


edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Pavan et al. Counting and sampling triangles from a graph stream, VLDB 2013

Approximation by Sampling Patterns

graph



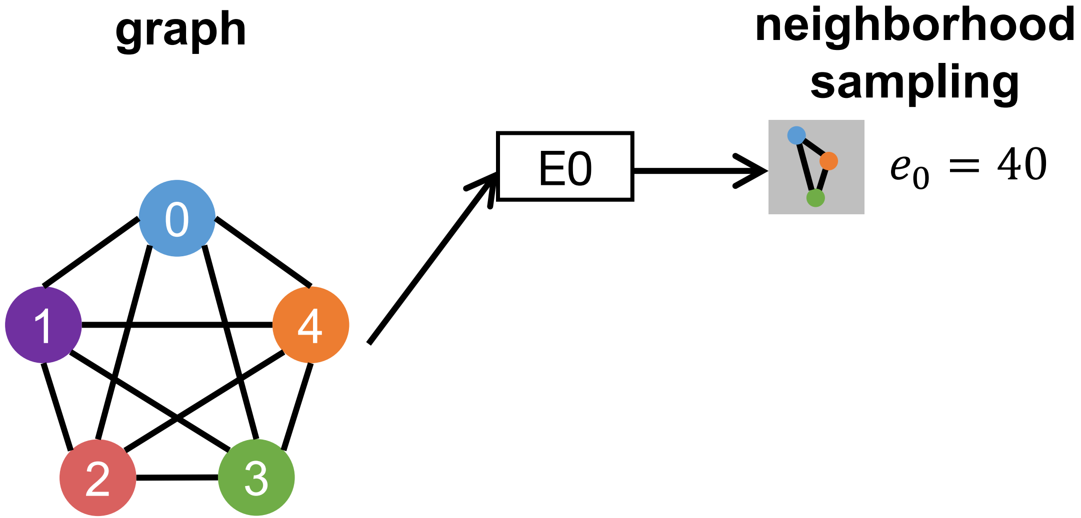
$$p = \frac{1}{10} * \frac{1}{4}$$

$$e_0 = 40$$

edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Pavan et al. Counting and sampling triangles from a graph stream, VLDB 2013

Approximation by Sampling Patterns

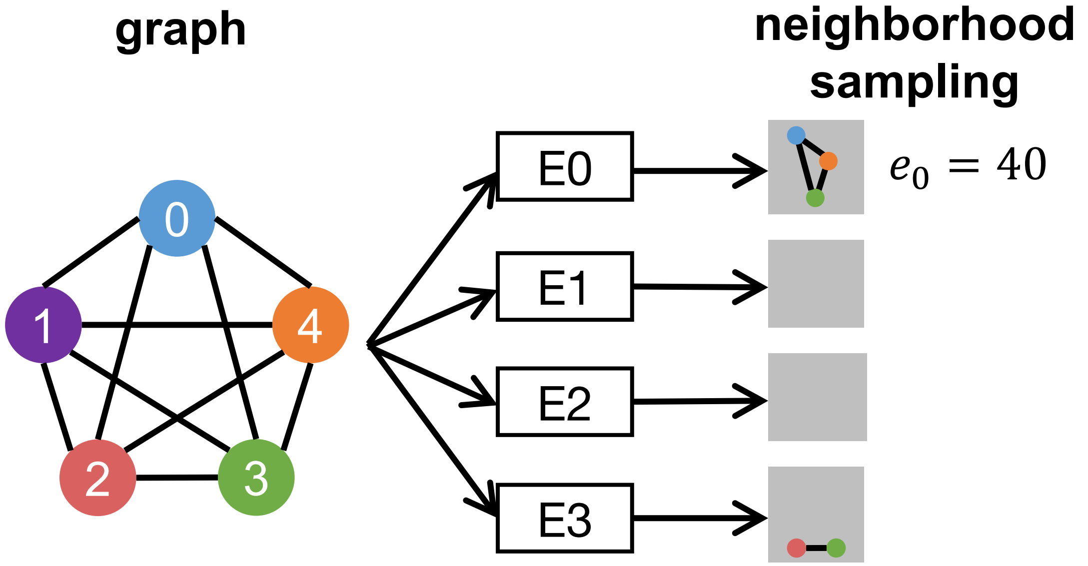


$e_0 = 40$

edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Pavan et al. Counting and sampling triangles from a graph stream, VLDB 2013

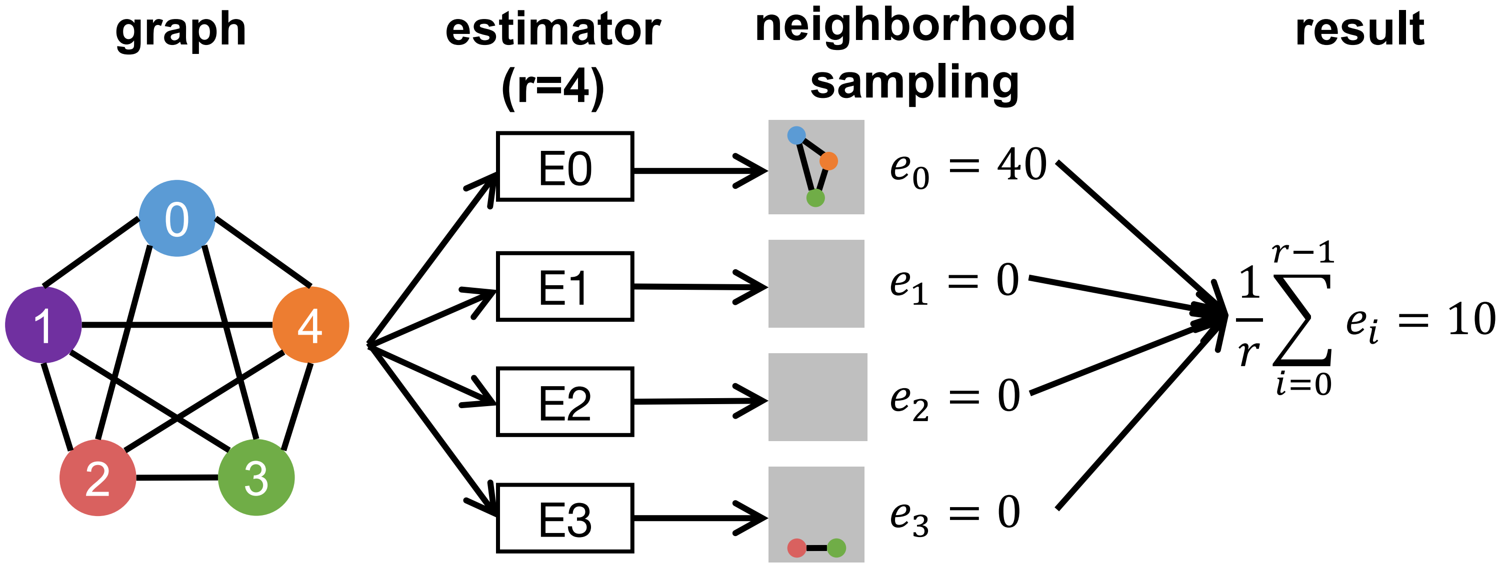
Approximation by Sampling Patterns



edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Pavan et al. Counting and sampling triangles from a graph stream, VLDB 2013

Approximation by Sampling Patterns



edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Pavan et al. Counting and sampling triangles from a graph stream, VLDB 2013

Potential Benefits

- 16 node Apache Spark cluster
- Two graphs: Live Journal (68.9B), Twitter (1.47B)
- Count 3-Motifs (2 patterns: triangle, 3-chain)
- Set error to 5%

Potential Benefits

- 16 node Apache Spark cluster
- Two graphs: Live Journal (68.9B), Twitter (1.47B)
- Count 3-Motifs (2 patterns: triangle, 3-chain)
- Set error to 5%

3-Motif	System	Graph	|V|	|E|	Time
Ours (5%)	16 x 8	LiveJ	4.8M	68.9B	11.5s
Arabesque	16 x 8	LiveJ	41.7M	1.47B	299.2s
Ours (5%)	16 x 8	Twitter	41.7M	1.47B	4m
Arabesque	20x32	Instagram	180M	0.9B	10h45m

Potential Benefits

- 16 node Apache Spark cluster
- Two graphs: Live Journal (68.9B), Twitter (1.47B)
- Count 3-Motifs (2 patterns: triangle, 3-chain)
- Set error to 5%

3-Motif	System	Graph	|V|	|E|	Time
Ours (5%)	16 x 8	LiveJ	4.8M	68.9B	11.5s
Arabesque	16 x 8	LiveJ	41.7M	1.47B	299.2s
Ours (5%)	16 x 8	Twitter	41.7M	1.47B	4m
Arabesque	20x32	Instagram	180M	0.9B	10h45m

Potential Benefits

- 16 node Apache Spark cluster
- Two graphs: Live Journal (68.9B), Twitter (1.47B)
- Count 3-Motifs (2 patterns: triangle, 3-chain)
- Set error to 5%

3-Motif	System	Graph	|V|	|E|	Time
Ours (5%)	16 x 8	LiveJ	4.8M	68.9B	11.5s
Arabesque	16 x 8	LiveJ	41.7M	1.47B	299.2s
Ours (5%)	16 x 8	Twitter	41.7M	1.47B	4m
Arabesque	20x32	Instagram	180M	0.9B	10h45m

Building a General Purpose Approximate Graph Mining System

Building a General Purpose Approximate Graph Mining System

General Patterns

Building a General Purpose Approximate Graph Mining System

General Patterns

Distributed Settings

Building a General Purpose Approximate Graph Mining System

- General Patterns
- Distributed Settings
- Error Estimation

Building a General Purpose Approximate Graph Mining System

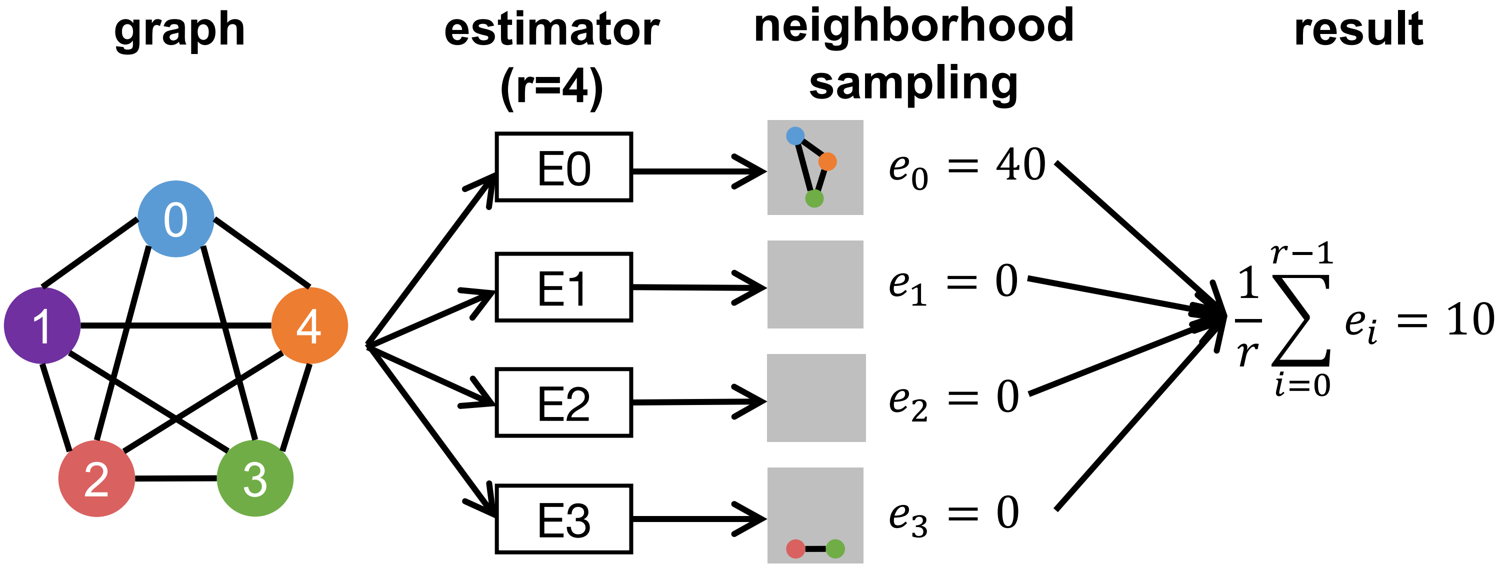
- General Patterns
- Distributed Settings
- Error Estimation
- Handling Updates

Challenge #1: General Patterns

<u>Problem:</u> Neighborhood sampling is for triangle counting

Break down neighborhood sampling into two phases:

- *Sampling* phase
- *Closing* phase

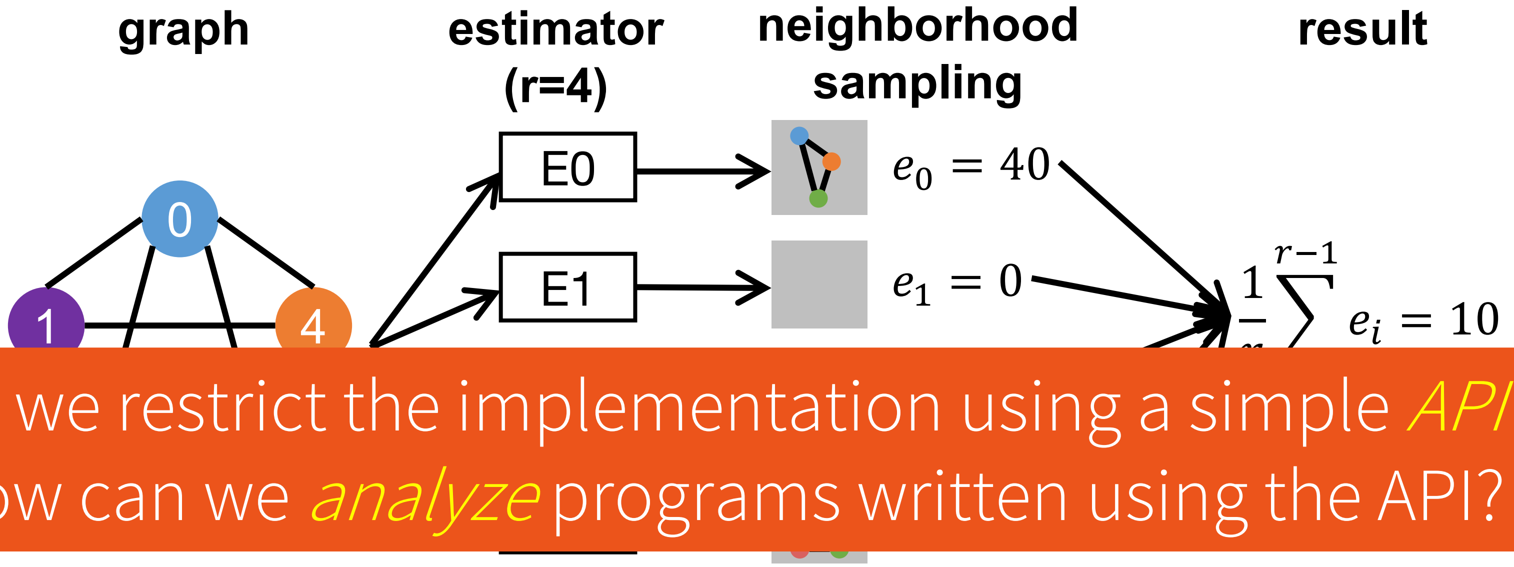


Challenge #1: General Patterns

<u>Problem:</u> Neighborhood sampling is for triangle counting

Break down neighborhood sampling into two phases:

- *Sampling* phase
- *Closing* phase



Can we restrict the implementation using a simple *API*?
How can we *analyze* programs written using the API?

Challenge #2: Distributed Setting

Problem: Neighborhood sampling is for a single machine

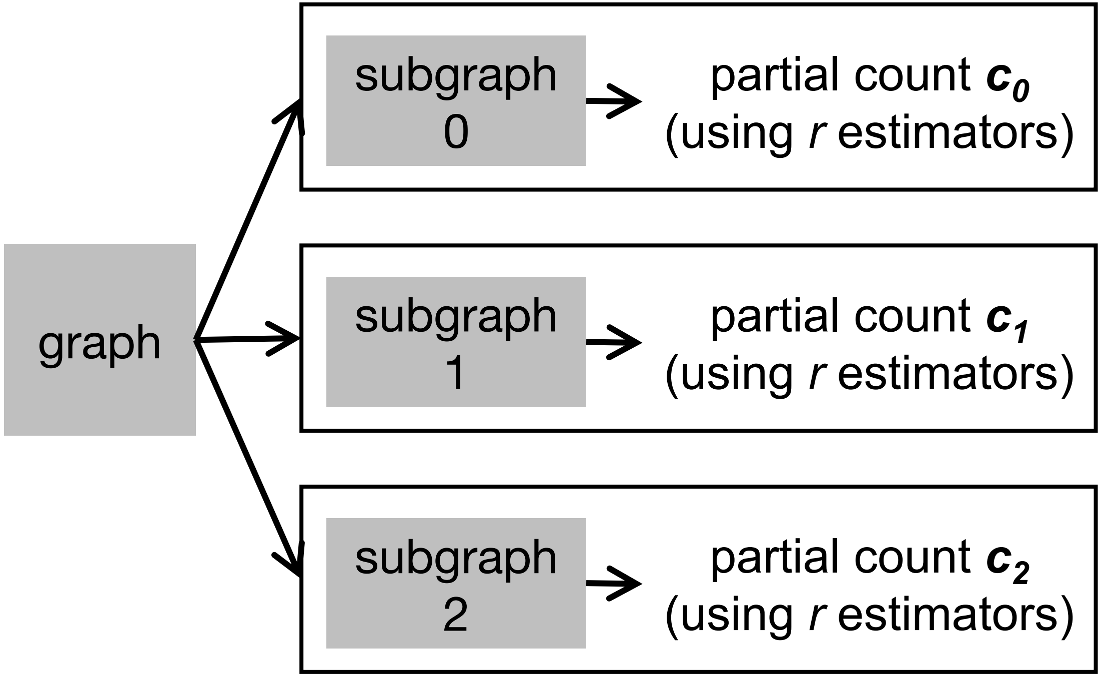
Challenge #2: Distributed Setting

<u>Problem:</u> Neighborhood sampling is for a single machine

graph

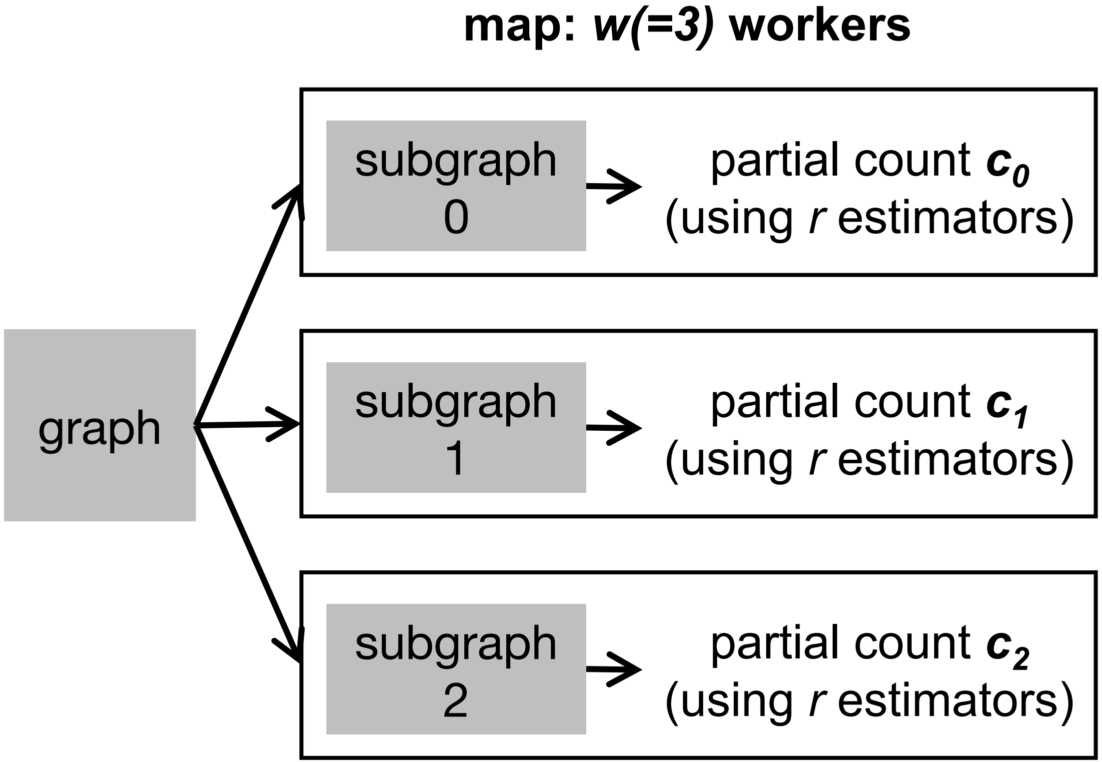
Challenge #2: Distributed Setting

Problem: Neighborhood sampling is for a single machine



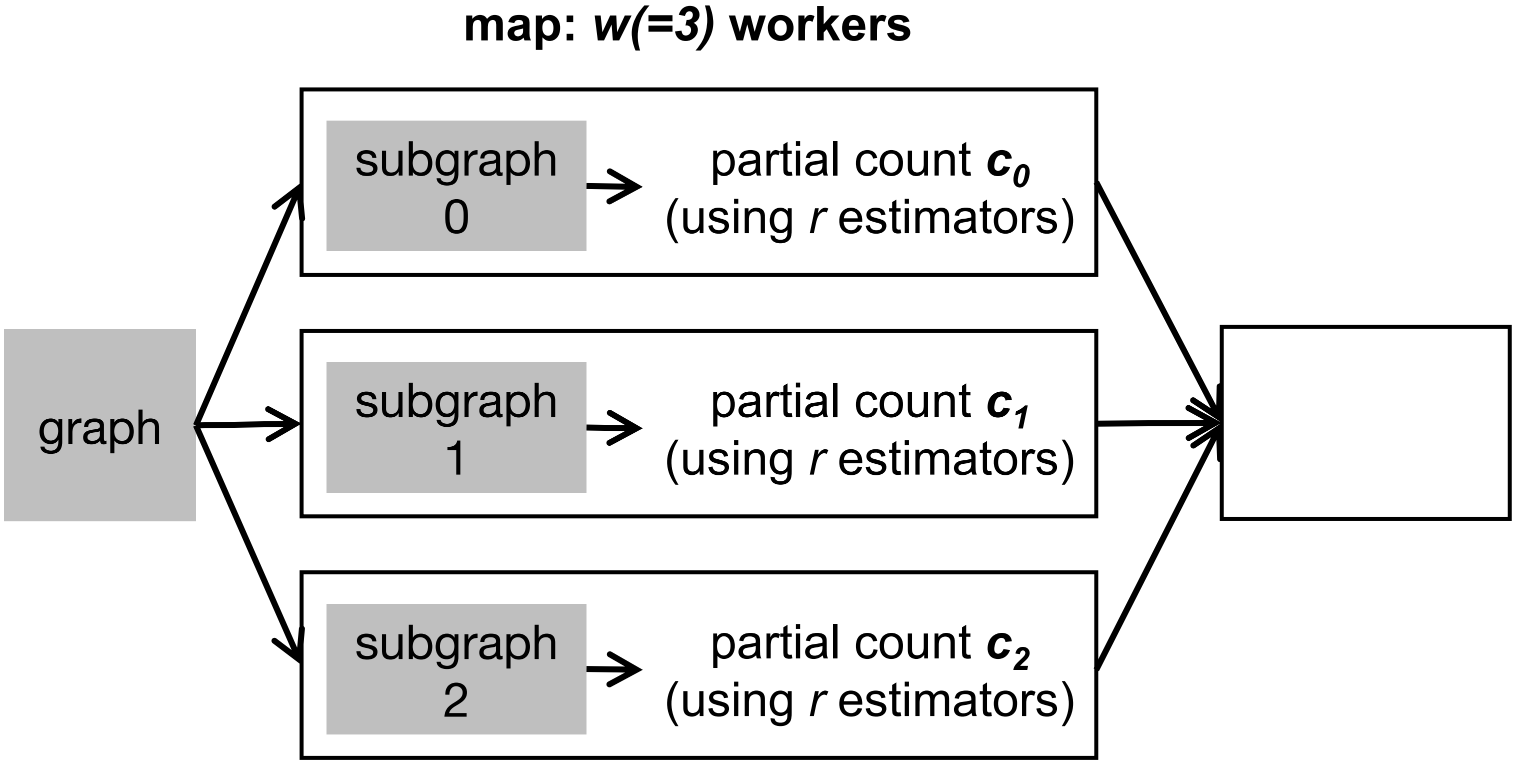
Challenge #2: Distributed Setting

<u>Problem:</u> Neighborhood sampling is for a single machine



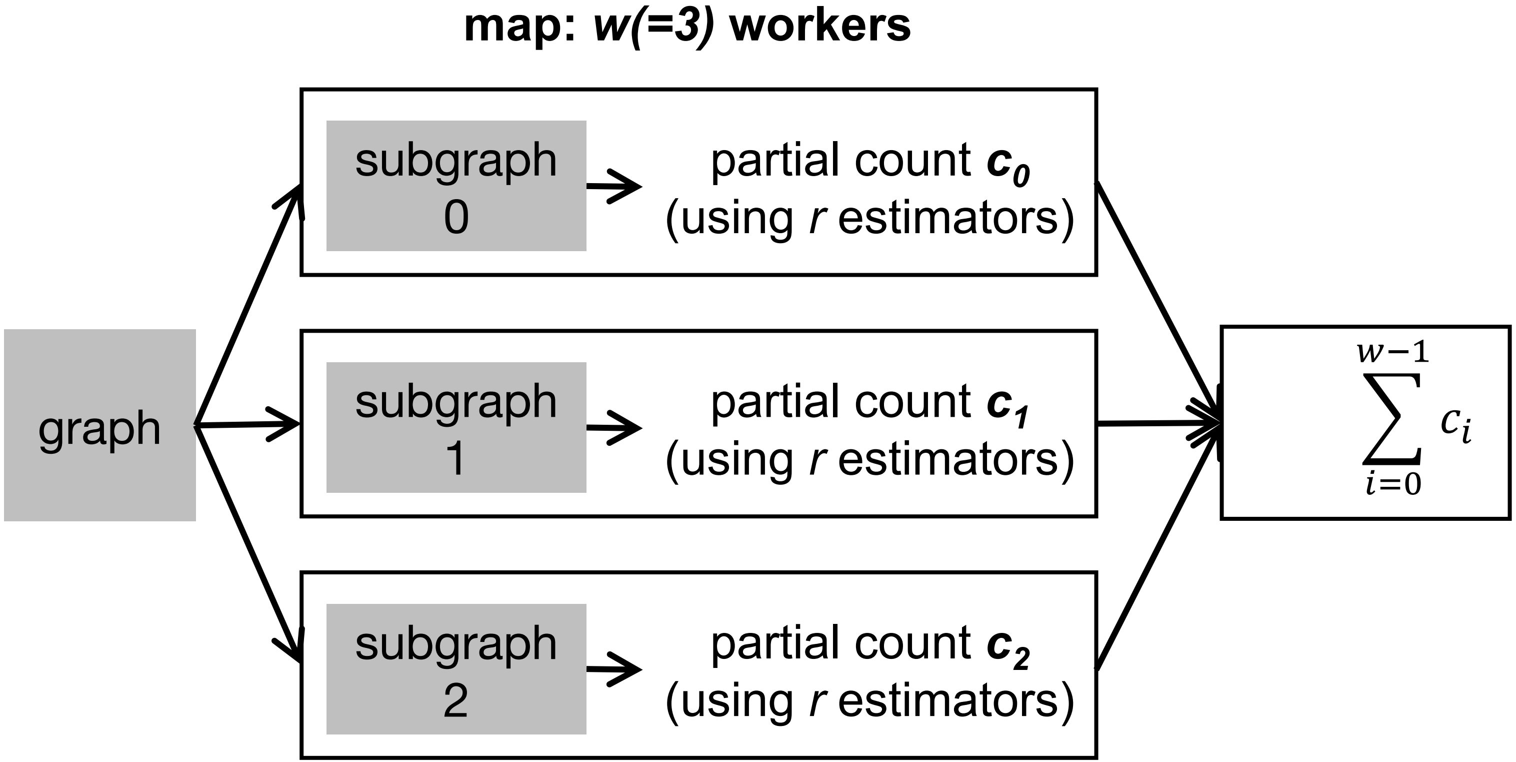
Challenge #2: Distributed Setting

Problem: Neighborhood sampling is for a single machine



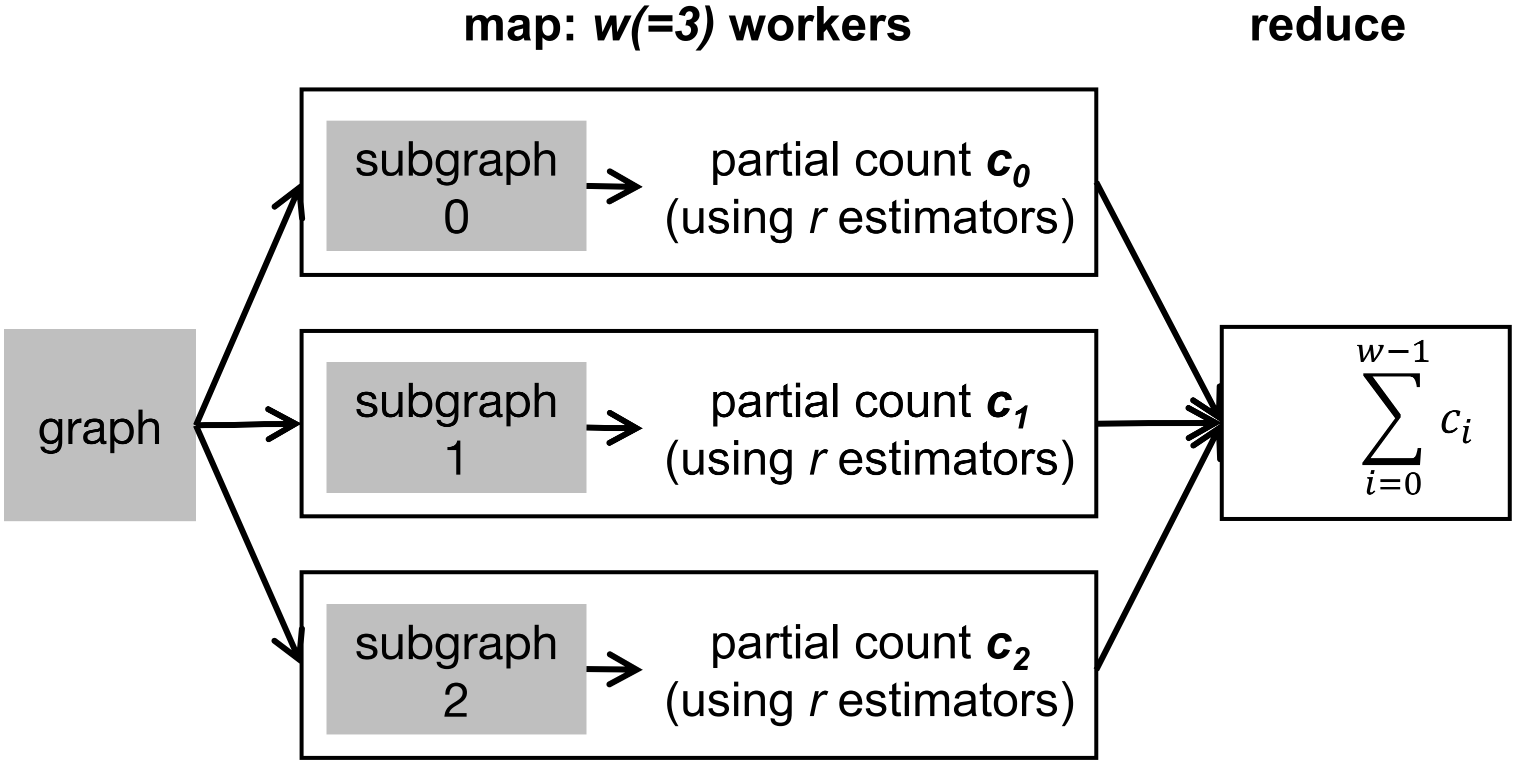
Challenge #2: Distributed Setting

Problem: Neighborhood sampling is for a single machine



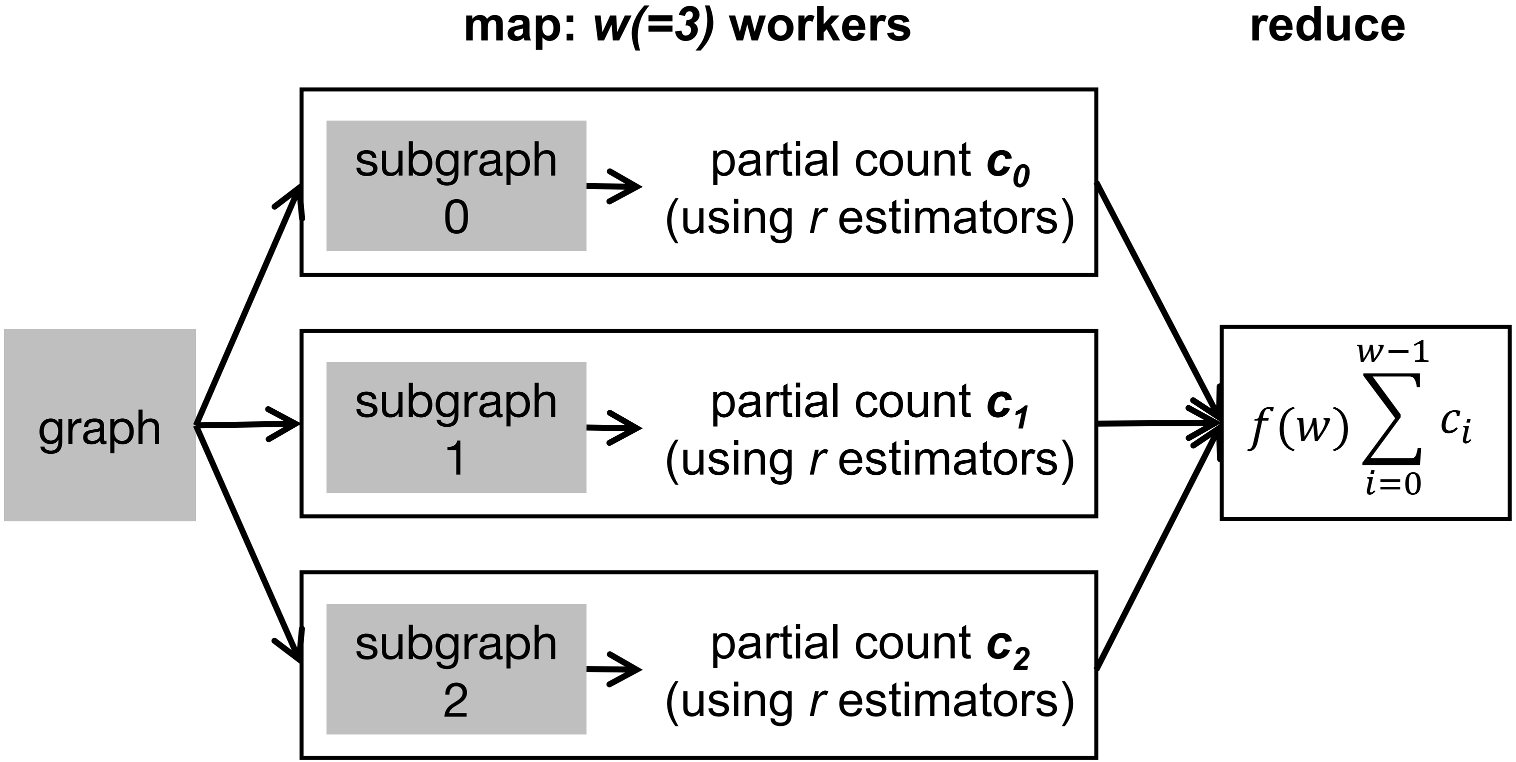
Challenge #2: Distributed Setting

<u>Problem:</u> Neighborhood sampling is for a single machine



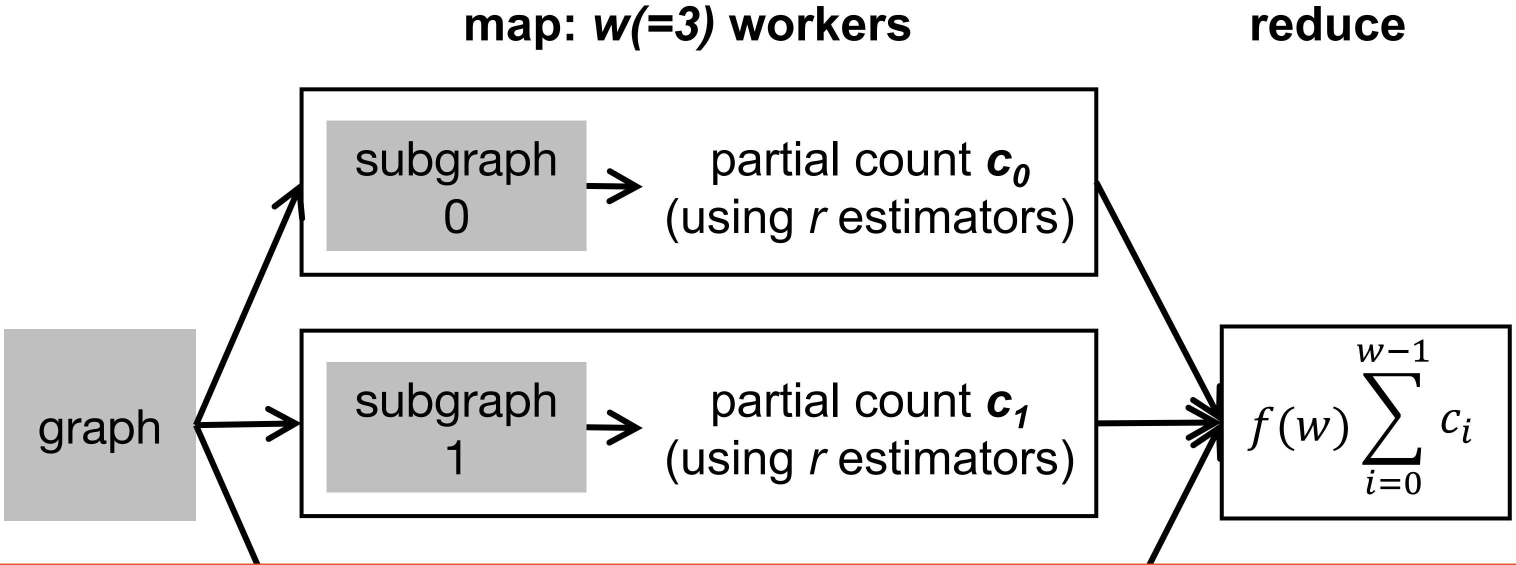
Challenge #2: Distributed Setting

Problem: Neighborhood sampling is for a single machine



Challenge #2: Distributed Setting

Problem: Neighborhood sampling is for a single machine



How do we compute *f(w)* for any pattern?
How does *f(w)* affect error?

Challenge #3: Building Error-Latency Profile

Problem: Given a time / error bound, how many estimators should we use?

Need to build two profiles:

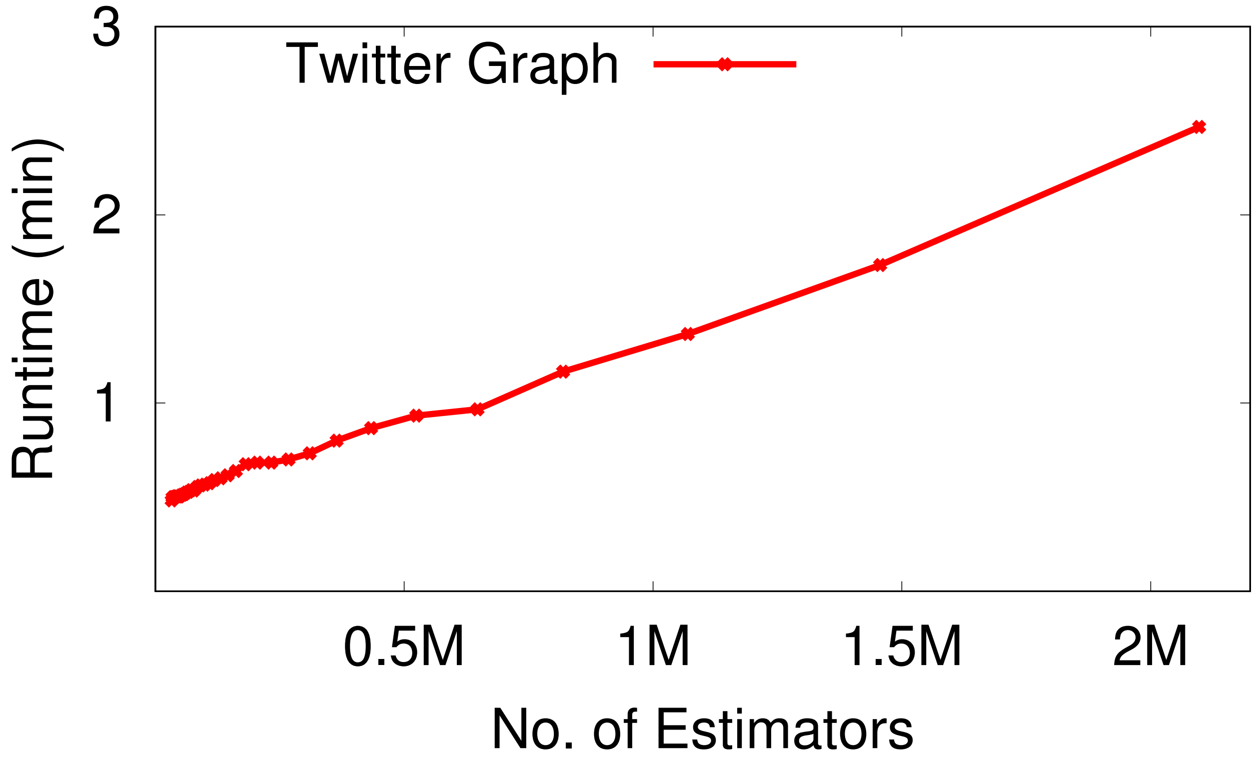
- Time vs #estimators
- Error vs #estimators

Naïve approach:

- Exhaustively run every possible point (infeasible)

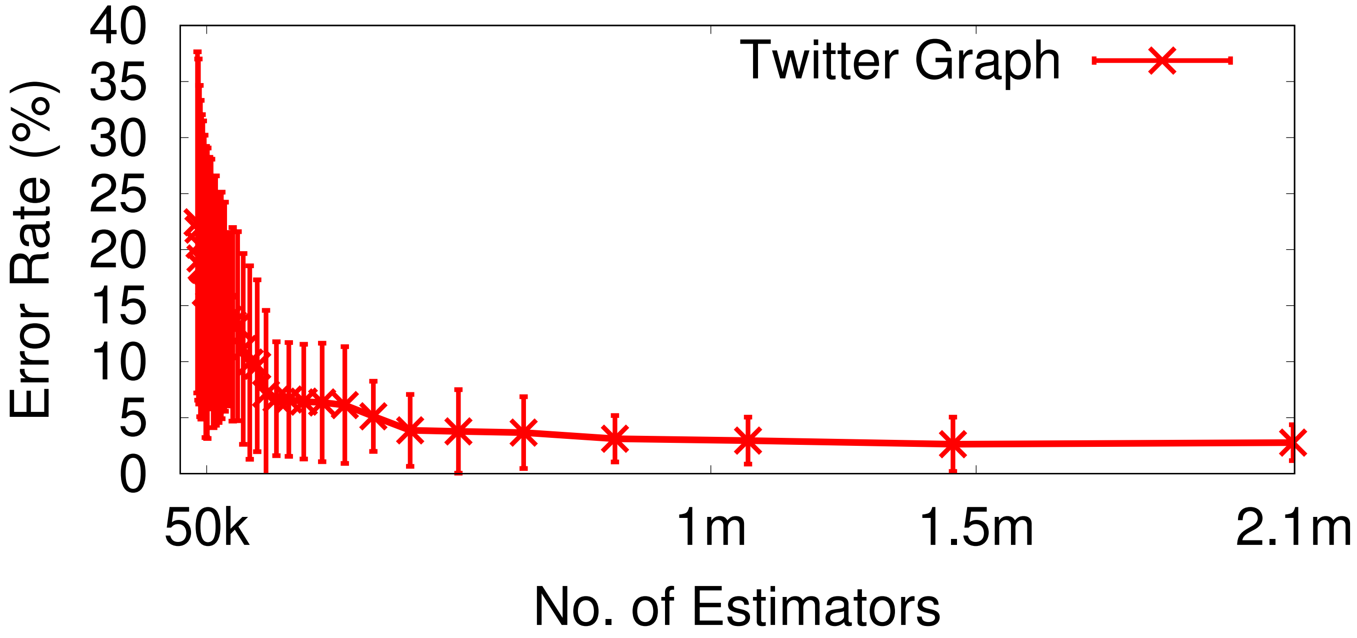
Building Estimators vs Time Profile

Time complexity linear in number of estimators



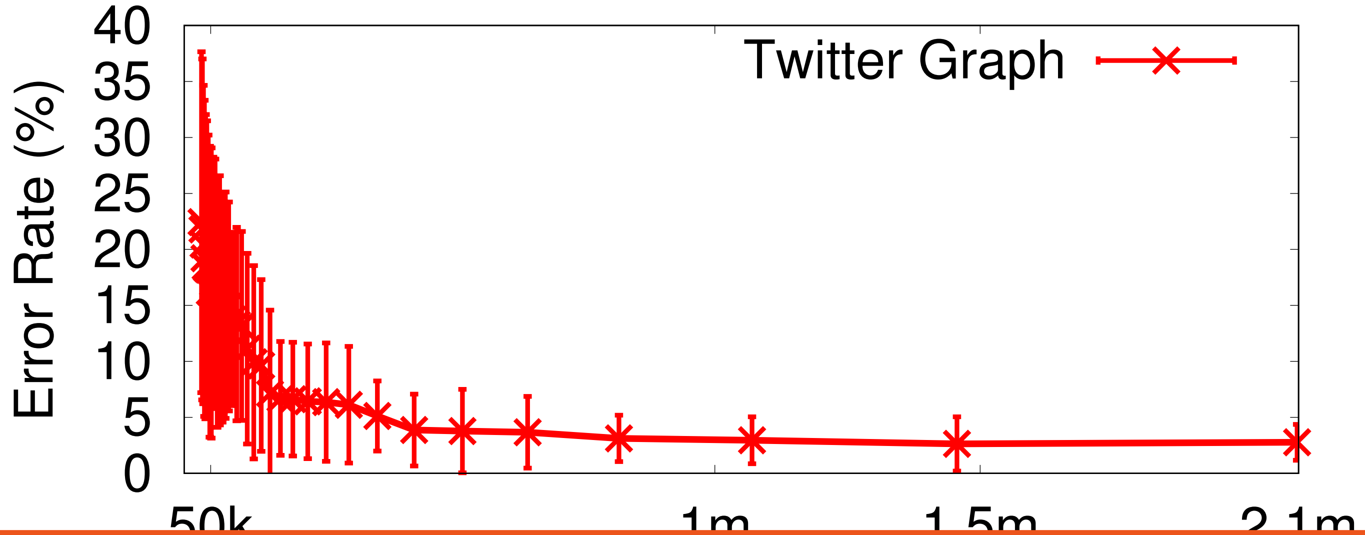
Building Estimators vs Error Profile

Error complexity non-linear in number of estimators



Building Estimators vs Error Profile

Error complexity non-linear in number of estimators



Leverage techniques like *experiment design/Bayesian optimization*?
How do we avoid the need to know the ground truth?

Challenge #4: Updates

Problem: Graphs and queries can be updated/refined

Several systems challenges:

- Incremental pattern mining
  - Can the error-latency profiles be updated?
- Caching
  - Re-use results
  - Pre-computation

Conclusion

- Approximation is a promising solution for pattern mining
  - Significant benefits, and can handle much larger graphs…
  - … but cannot output all instances of the pattern
- Several challenges in realizing it
  - How to extend the technique to general patterns?
  - How to do approximate pattern mining in a distributed setting?
  - How do we estimate the error?
  - How do we handle updates?

http://www.cs.berkeley.edu/~api

api@cs.berkeley.edu