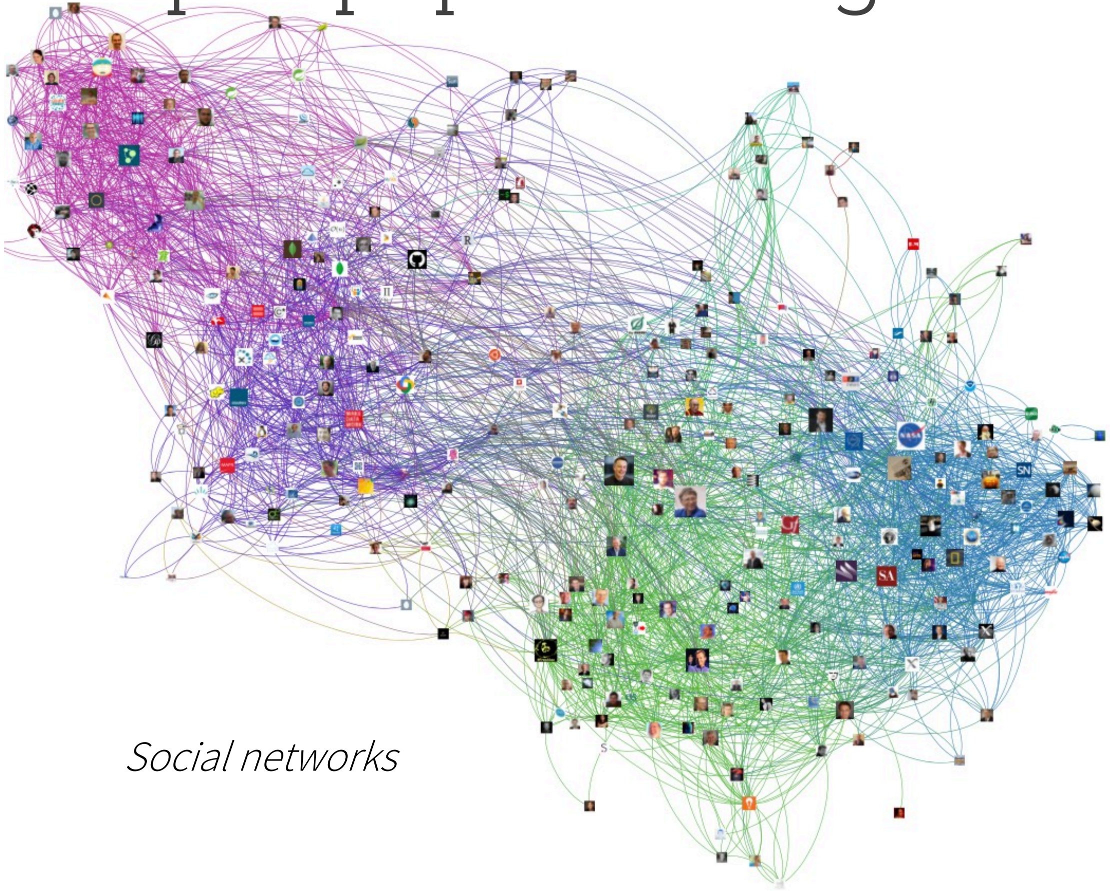
ASAP: Fast, Approximate Graph Pattern Mining at Scale

Anand Iyer[*], *Zaoxing Liu*[◆], *Xin Jin*[◆],
Shivaram Venkataraman[+], *Vladimir Braverman*[◆], *Ion Stoica*[*]
[*]UC Berkeley [◆]Johns Hopkins University [+]University of Wisconsin & Microsoft

OSDI, October 10, 2018

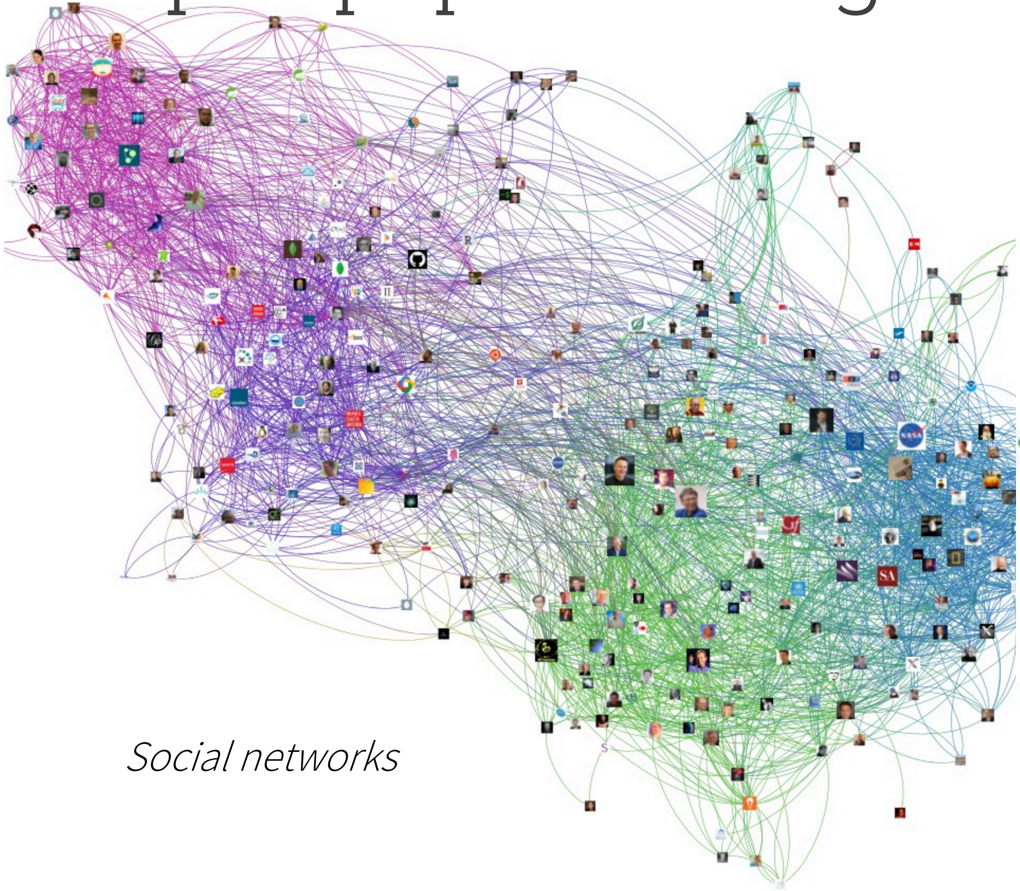
risel ab
UC Berkeley

Graphs popular in big data analytics

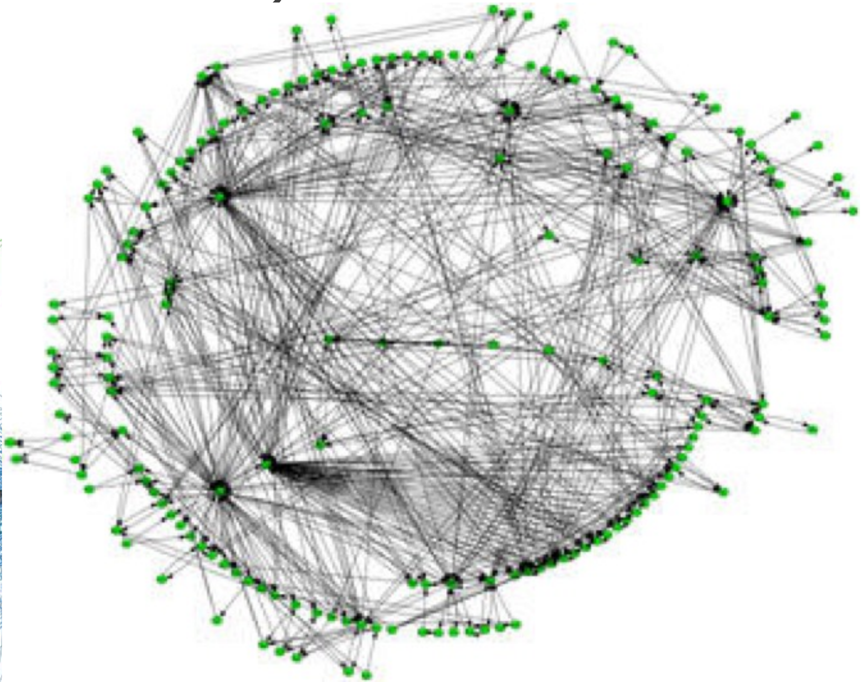


Social networks

Graphs popular in big data analytics

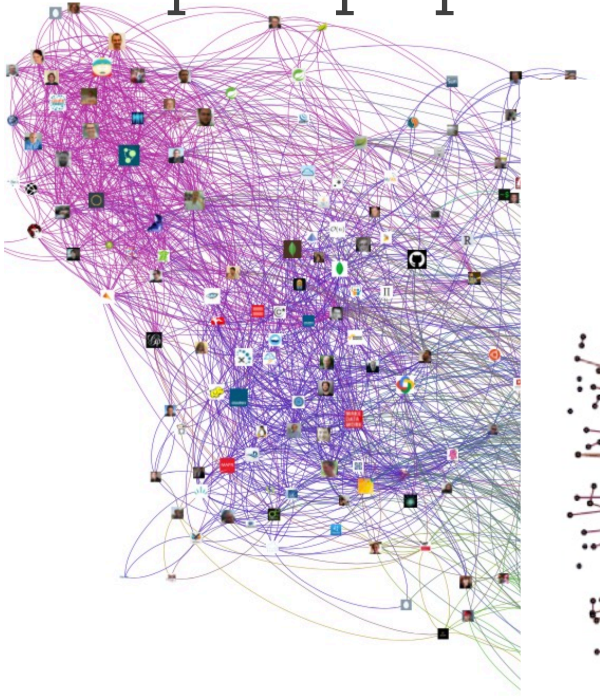


Social networks

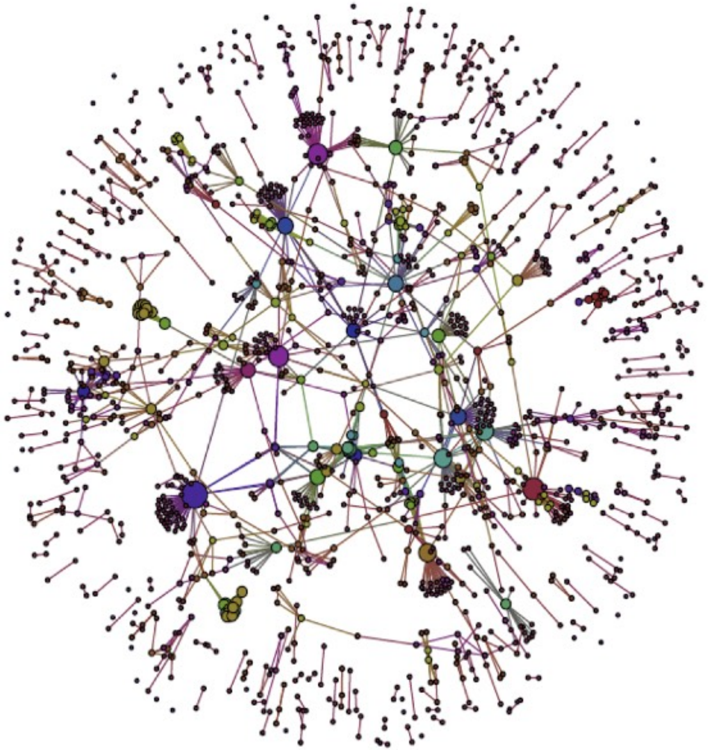


Metabolic network of a single cell organism

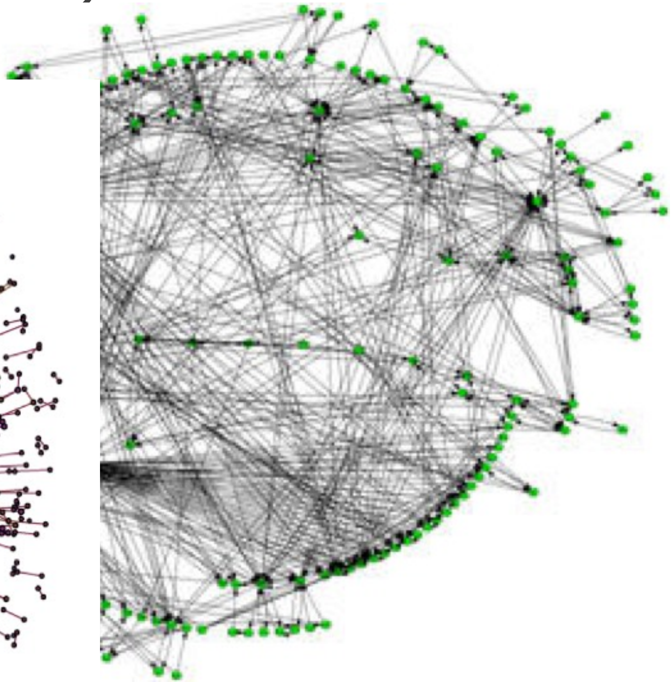
Graphs popular in big data analytics



Social networks



Tuberculosis



etwork of a single cell organism

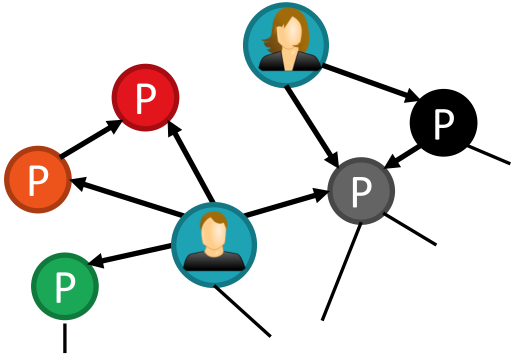
Graphs popular in big data analytics

Also popular in traditional enterprises*

*"The Ubiquity of Large Graphs and Surprising Challenges of Graph Processing" ,Sahu et. al, VLDB 2018 (best paper)

Graphs popular in big data analytics

Also popular in traditional enterprises*

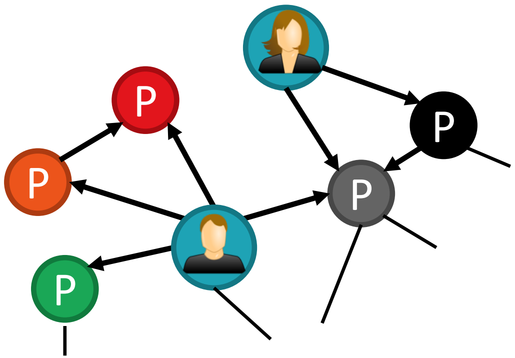


Products and customers

*"The Ubiquity of Large Graphs and Surprising Challenges of Graph Processing" ,Sahu et. al, VLDB 2018 (best paper)

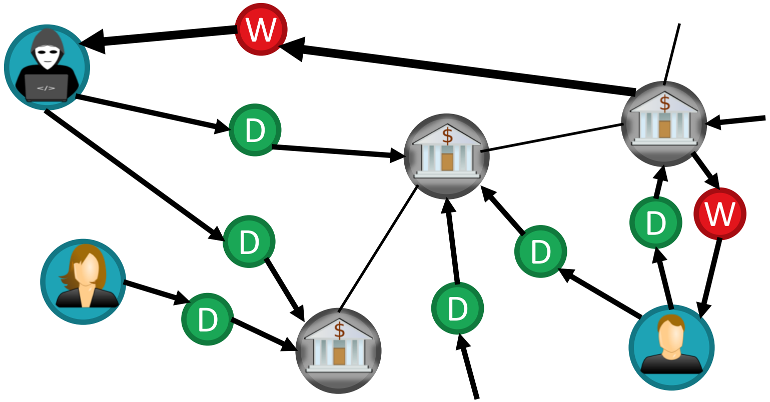
Graphs popular in big data analytics

Also popular in traditional enterprises*



Products and customers

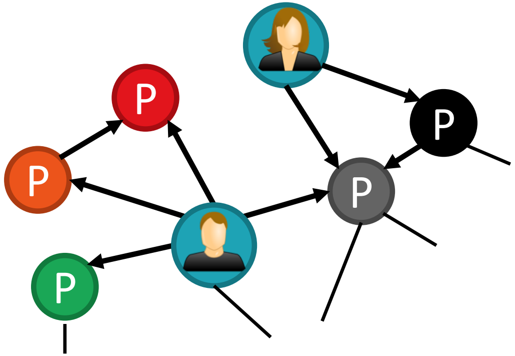
Transactions and involved entities



*"The Ubiquity of Large Graphs and Surprising Challenges of Graph Processing" ,Sahu et. al, VLDB 2018 (best paper)

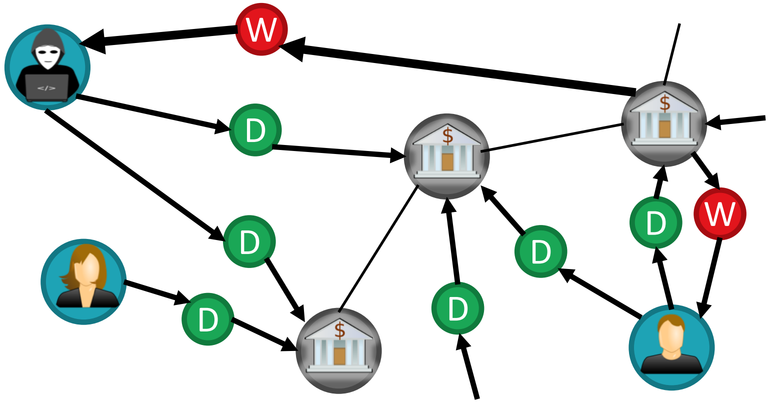
Graphs popular in big data analytics

Also popular in traditional enterprises*



Which (classes of) products are frequently bought together?

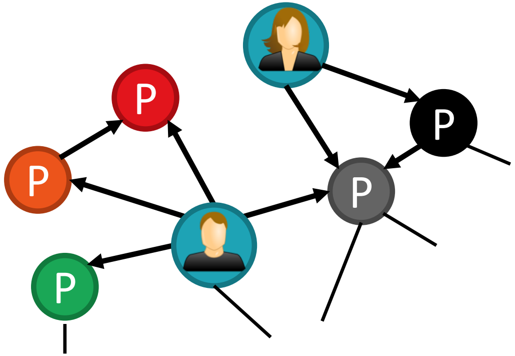
Transactions and involved entities



*"The Ubiquity of Large Graphs and Surprising Challenges of Graph Processing" ,Sahu et. al, VLDB 2018 (best paper)

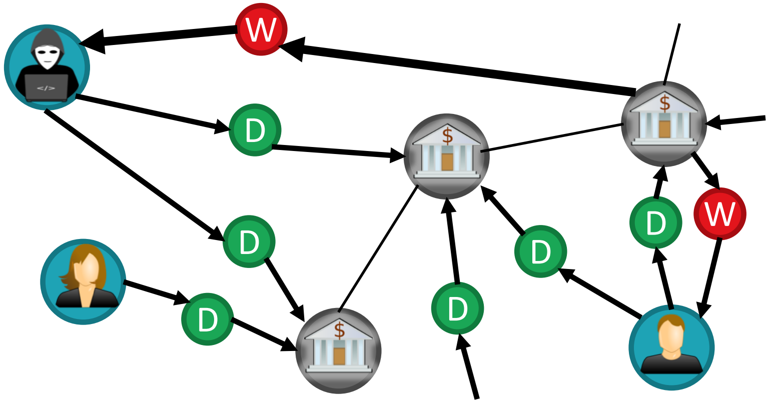
Graphs popular in big data analytics

Also popular in traditional enterprises*



Which (classes of) products are frequently bought together?

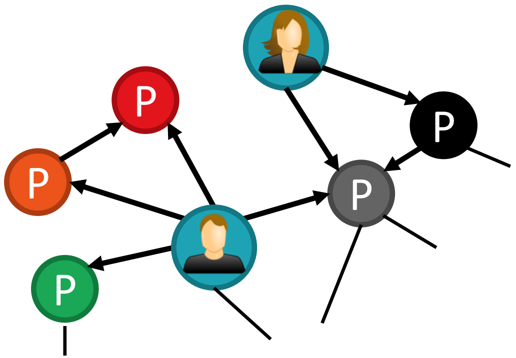
Small deposits followed by large withdrawal



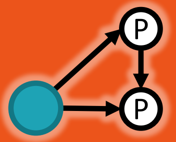
*"The Ubiquity of Large Graphs and Surprising Challenges of Graph Processing" ,Sahu et. al, VLDB 2018 (best paper)

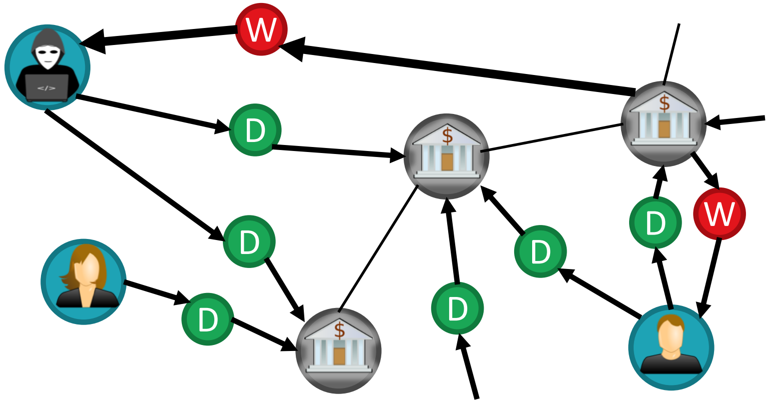
Graphs popular in big data analytics

Also popular in traditional enterprises*

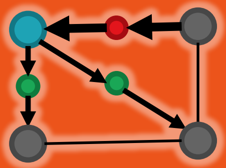


Which (classes of) products are frequently bought together?





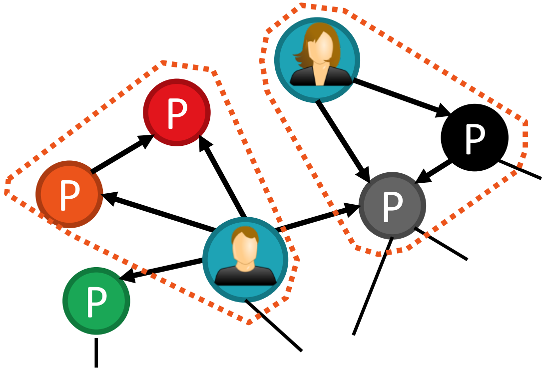
Small deposits followed by large withdrawal



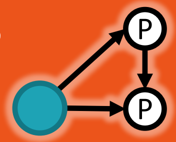
*"The Ubiquity of Large Graphs and Surprising Challenges of Graph Processing" ,Sahu et. al, VLDB 2018 (best paper)

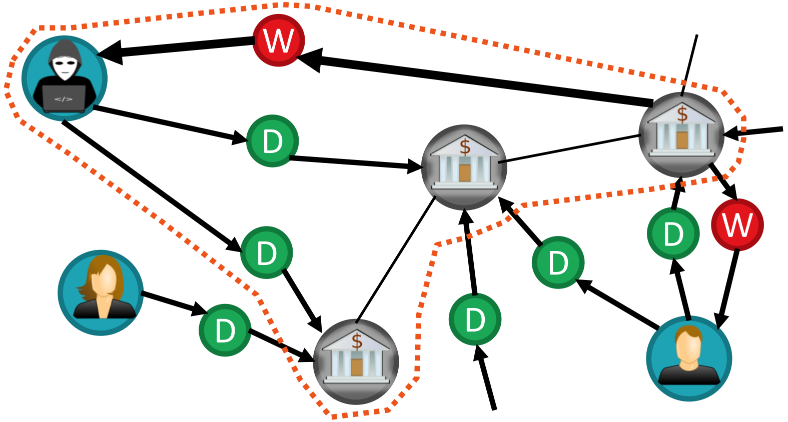
Graphs popular in big data analytics

Also popular in traditional enterprises*

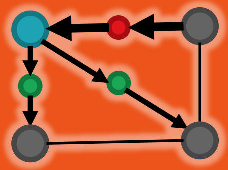


Which (classes of) products are frequently bought together?





Small deposits followed by large withdrawal



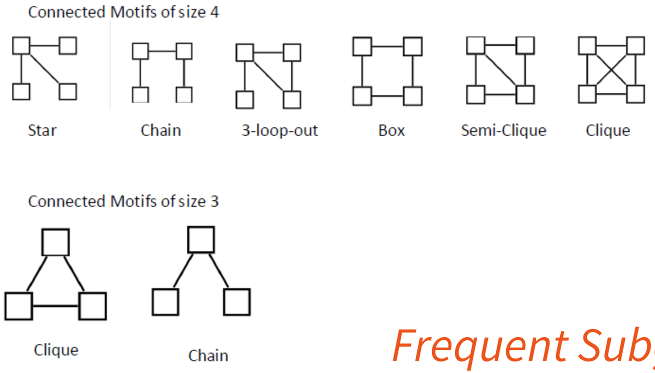
*"The Ubiquity of Large Graphs and Surprising Challenges of Graph Processing" ,Sahu et. al, VLDB 2018 (best paper)

Graph Pattern Mining

Discover structural patterns in the underlying graph

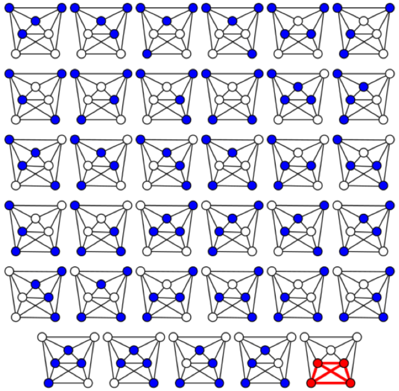
Graph Pattern Mining

Discover structural patterns in the underlying graph



Motifs

Frequent Subgraphs



Cliques

Graph Pattern Mining

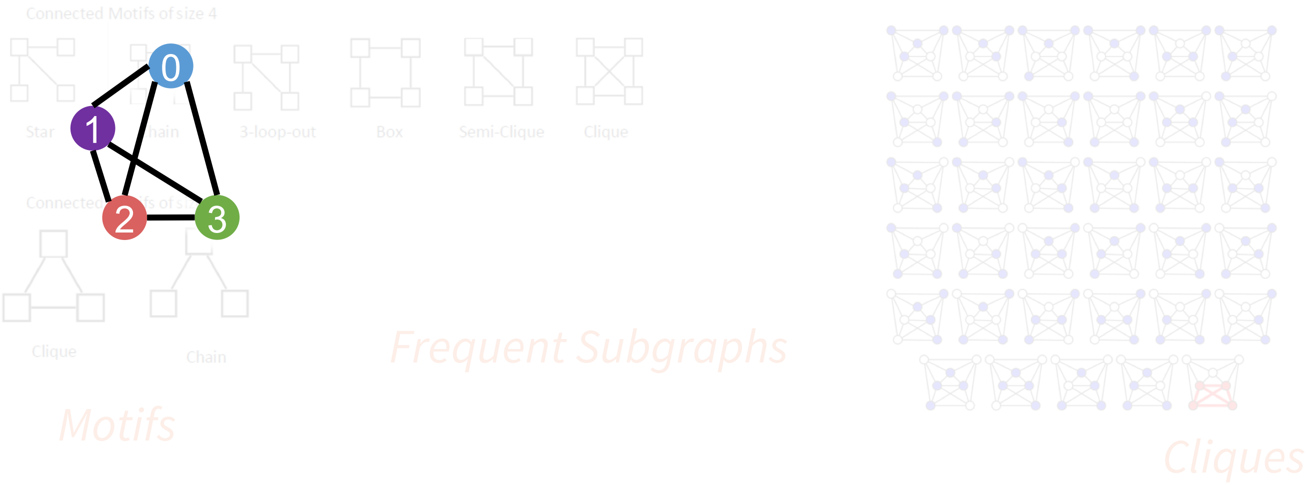
Discover structural patterns in the underlying graph

Standard approach: Iterative expansion

Graph Pattern Mining

Discover structural patterns in the underlying graph

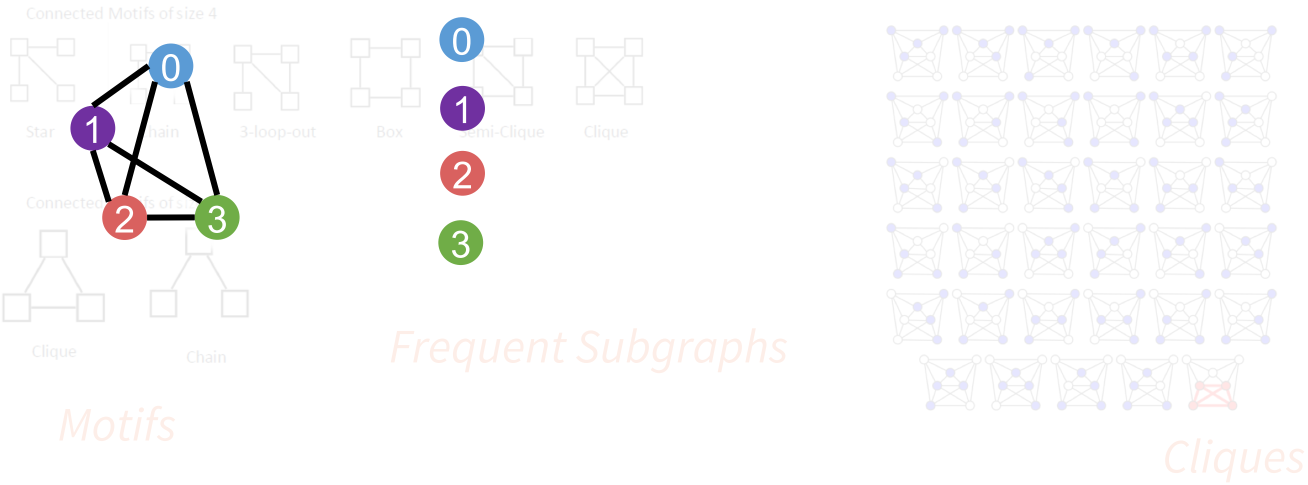
Standard approach: Iterative expansion



Graph Pattern Mining

Discover structural patterns in the underlying graph

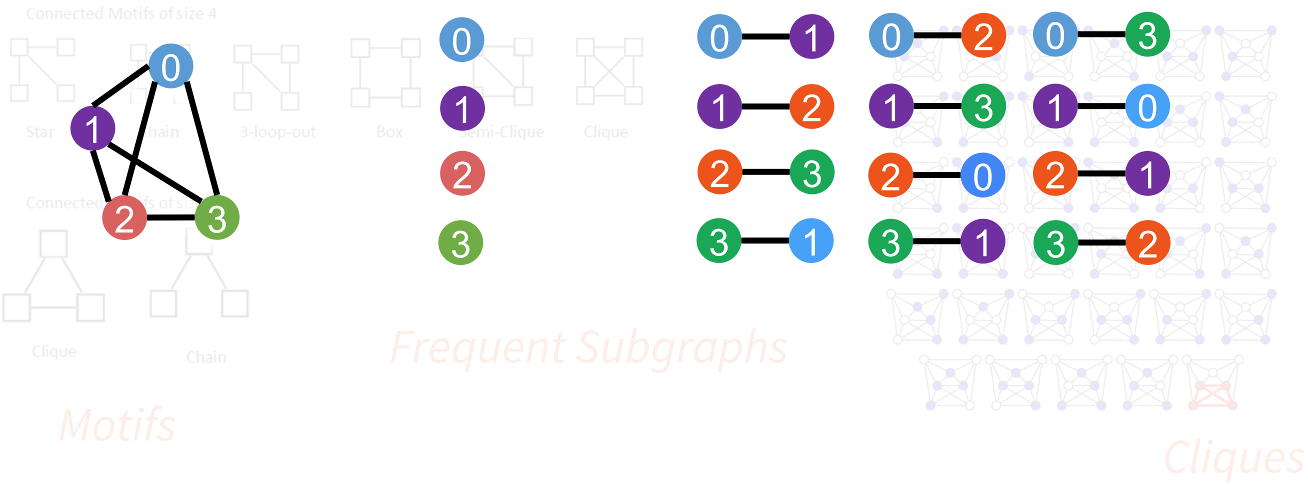
Standard approach: Iterative expansion



Graph Pattern Mining

Discover structural patterns in the underlying graph

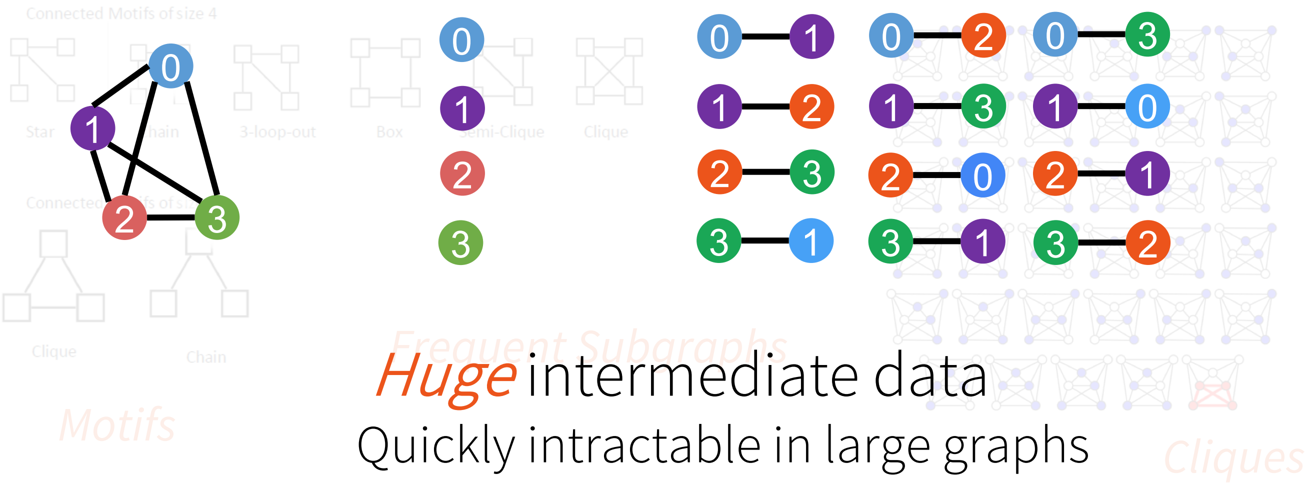
Standard approach: Iterative expansion



Graph Pattern Mining

Discover structural patterns in the underlying graph

Standard approach: Iterative expansion



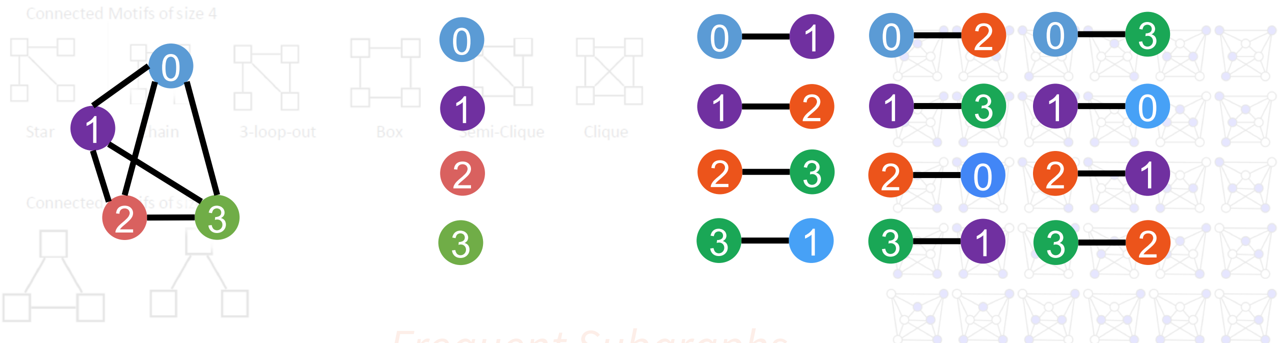
Huge intermediate data

Quickly intractable in large graphs

Graph Pattern Mining

Discover structural patterns in the underlying graph

Standard approach: Iterative expansion

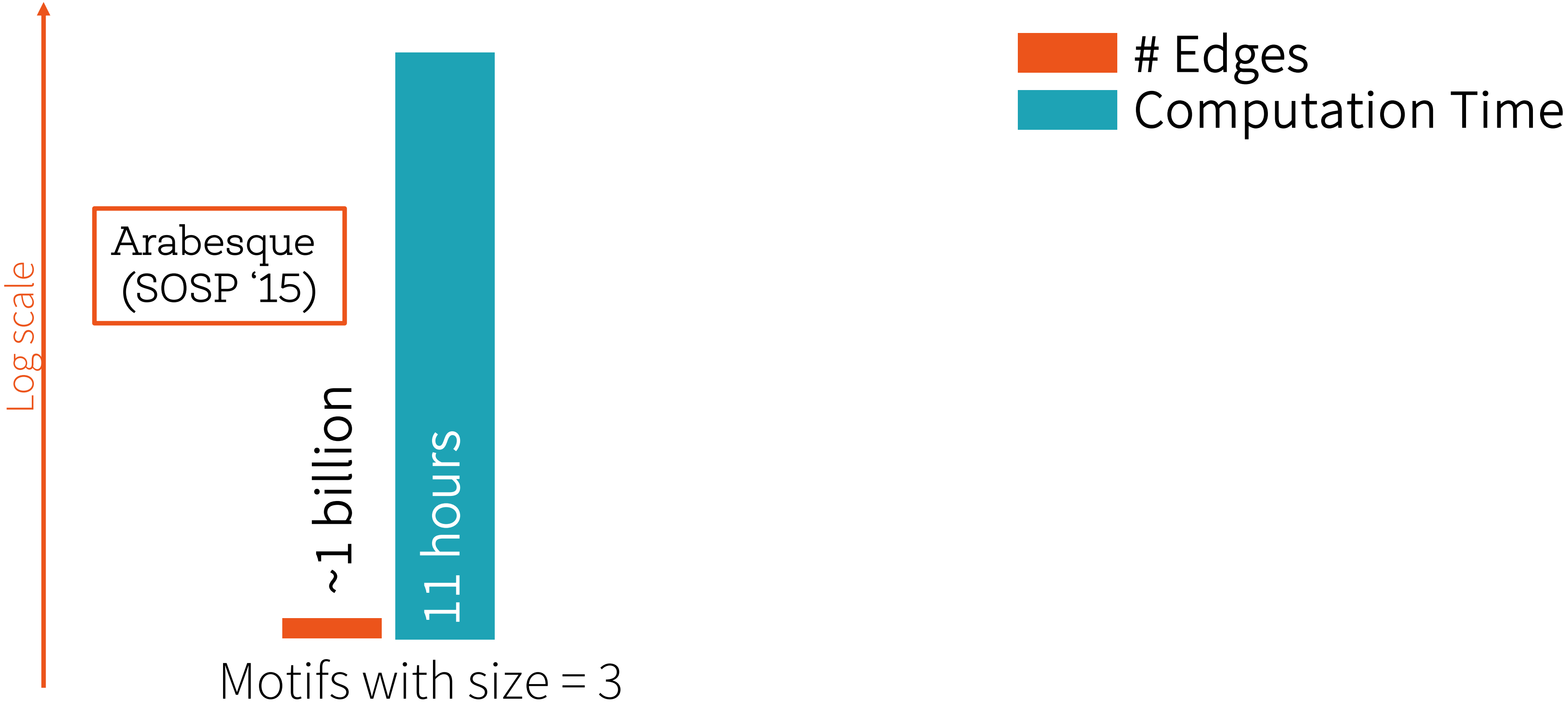


Challenging to mine patterns in large graphs

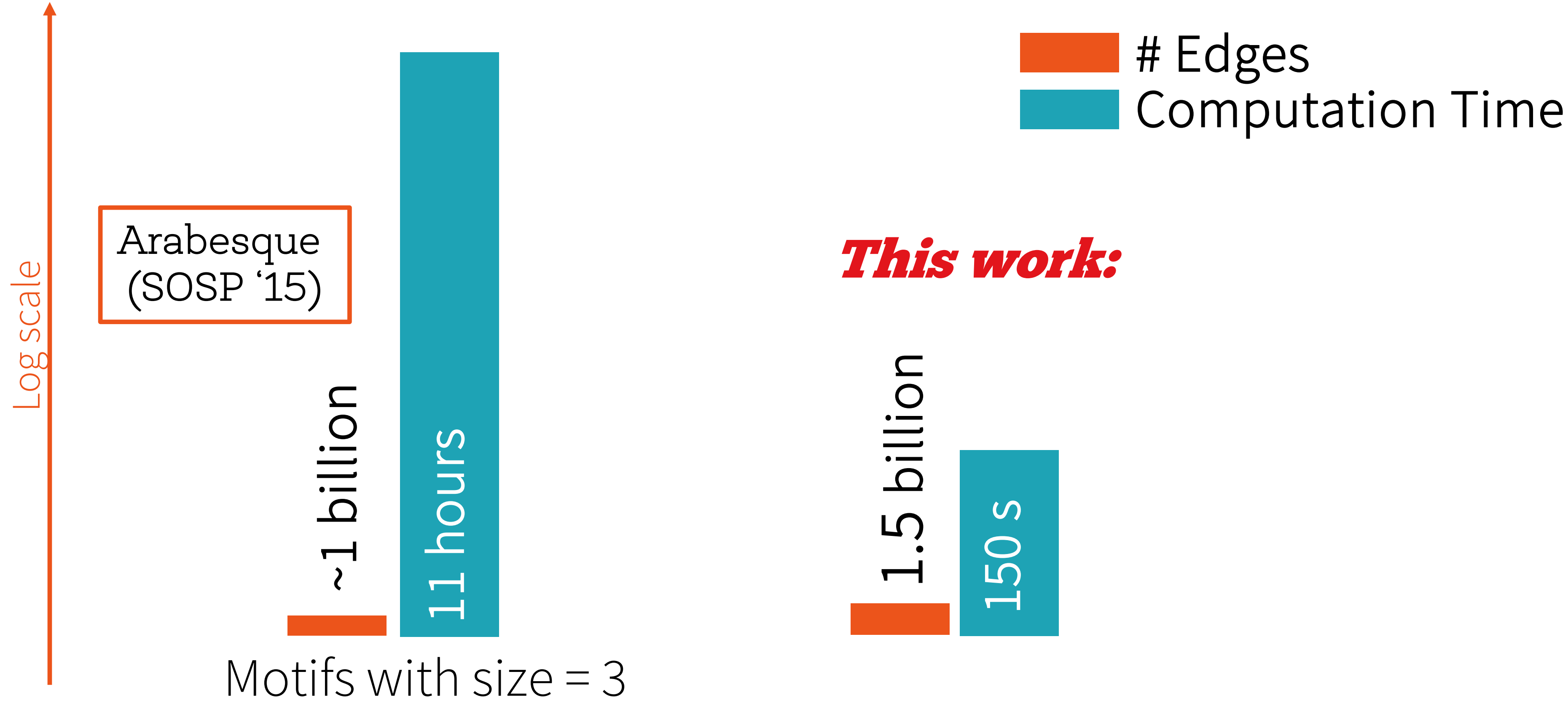
Graph Pattern Mining

Graph Pattern Mining

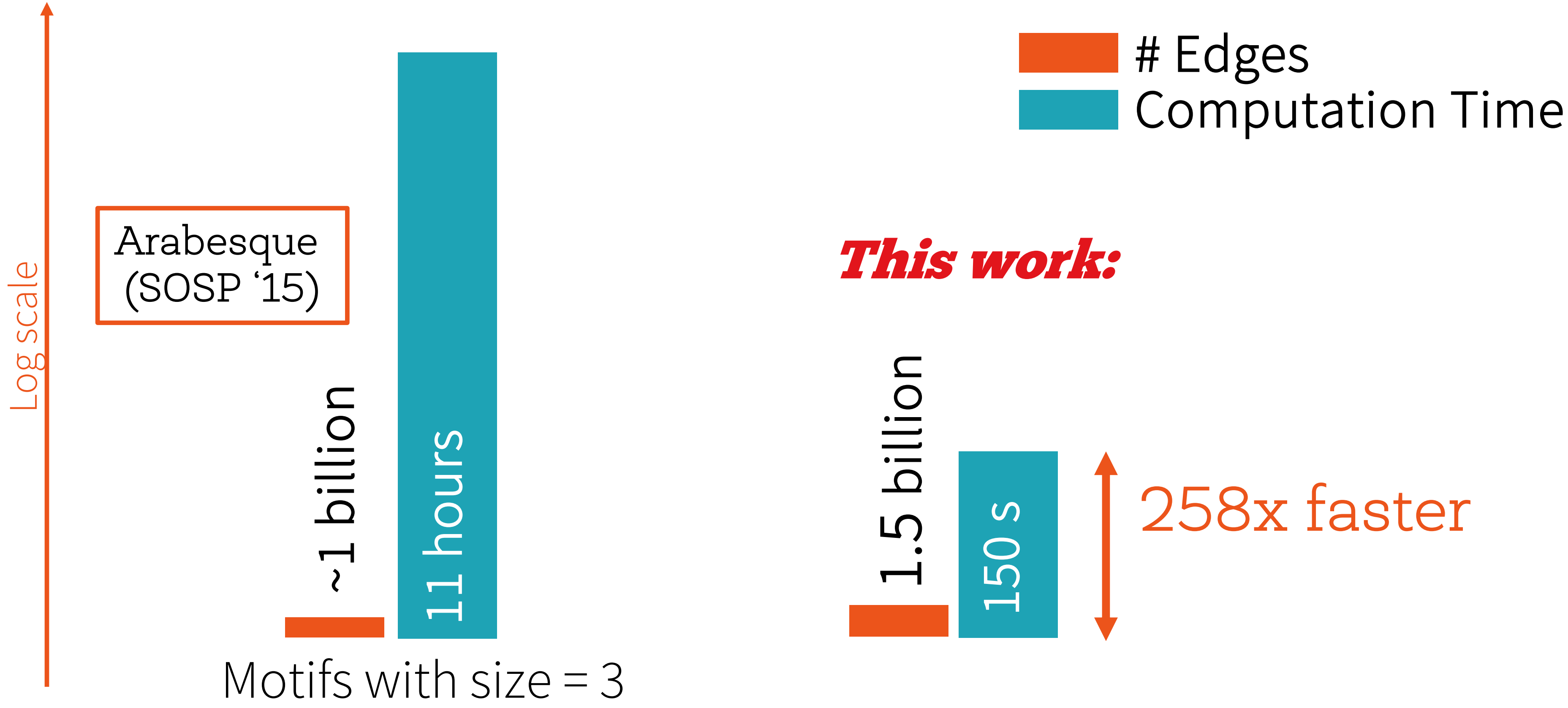
Graph Pattern Mining



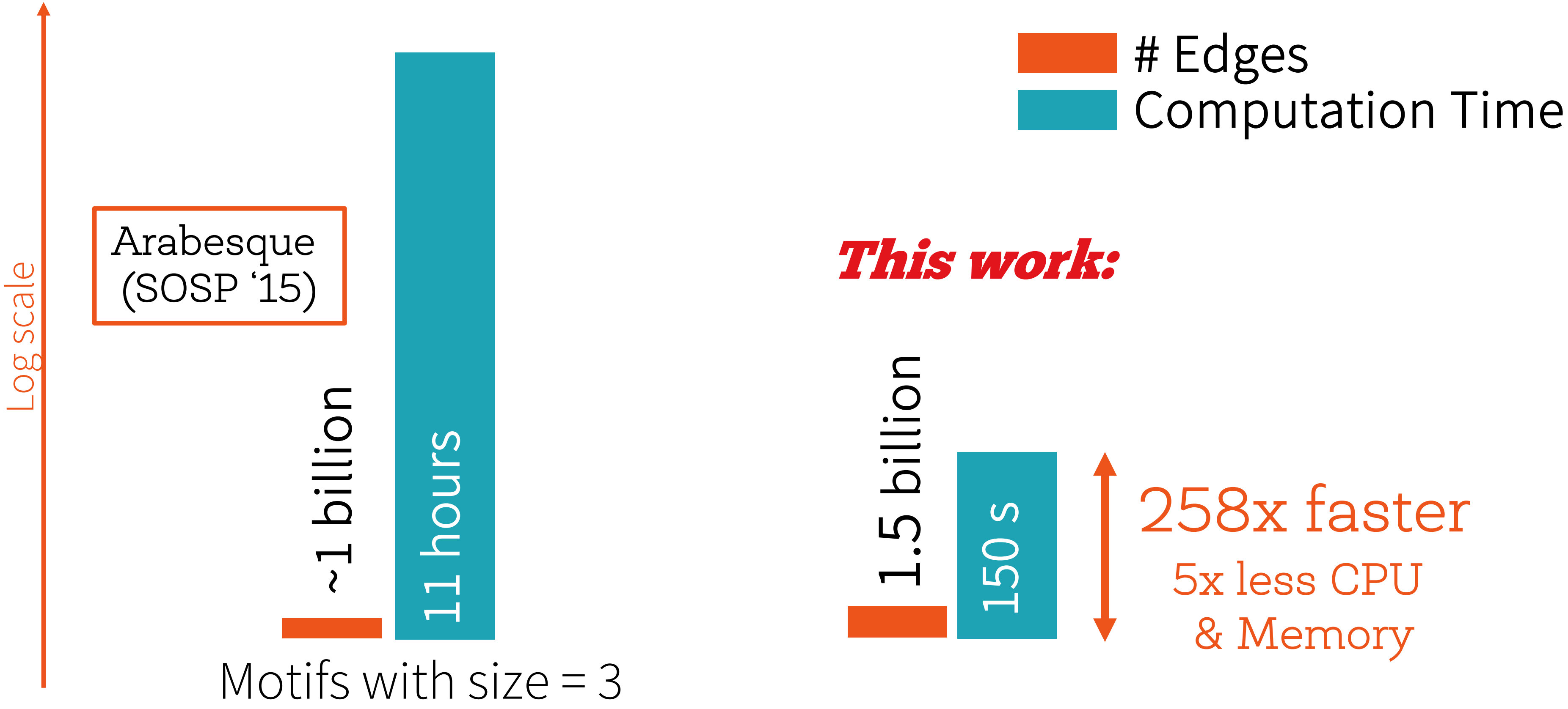
Graph Pattern Mining



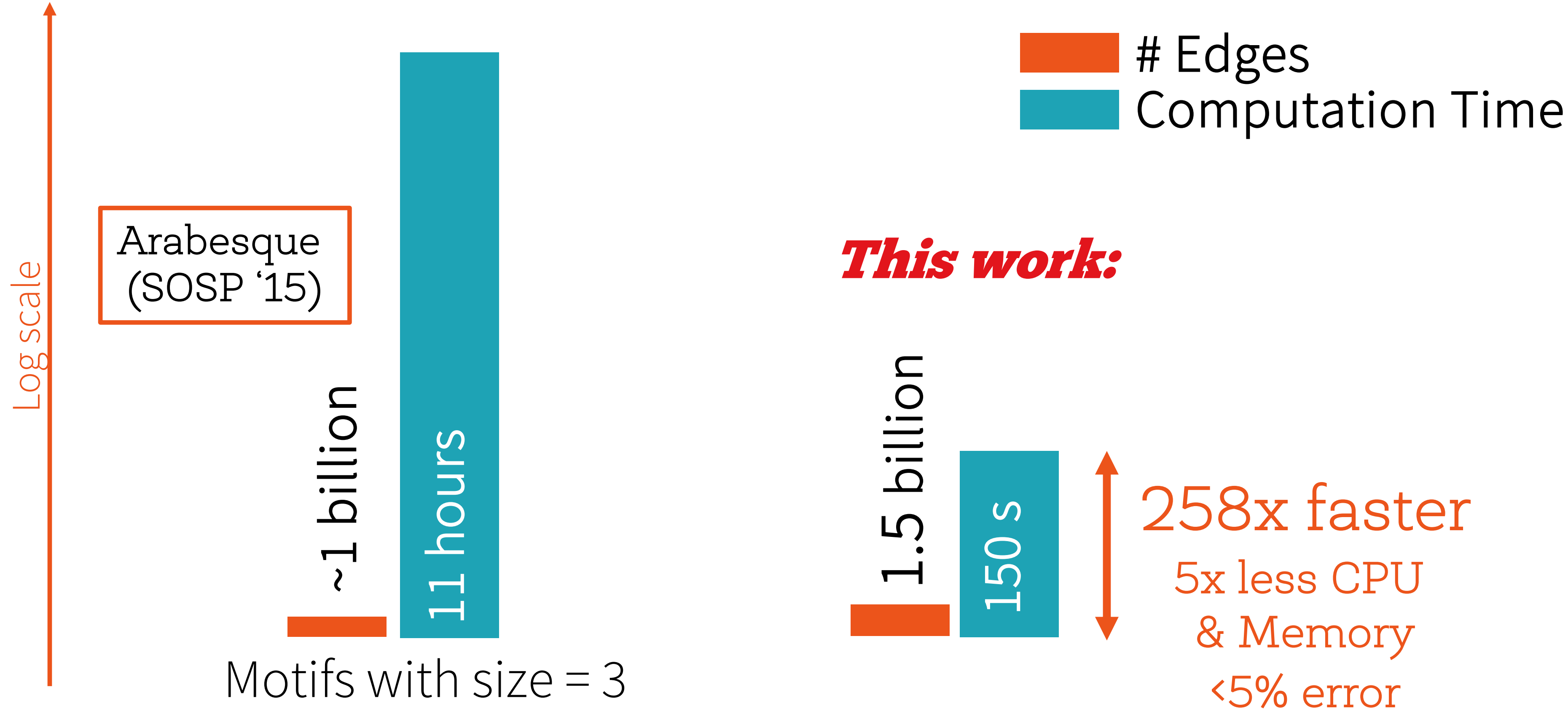
Graph Pattern Mining



Graph Pattern Mining



Graph Pattern Mining



Many mining tasks do not need *exact* answers

Many mining tasks do not need *exact* answers

Leverage *approximation* for pattern mining

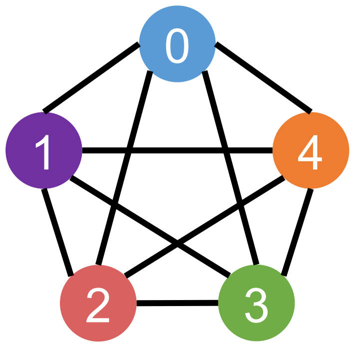
Approximate Analytics

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

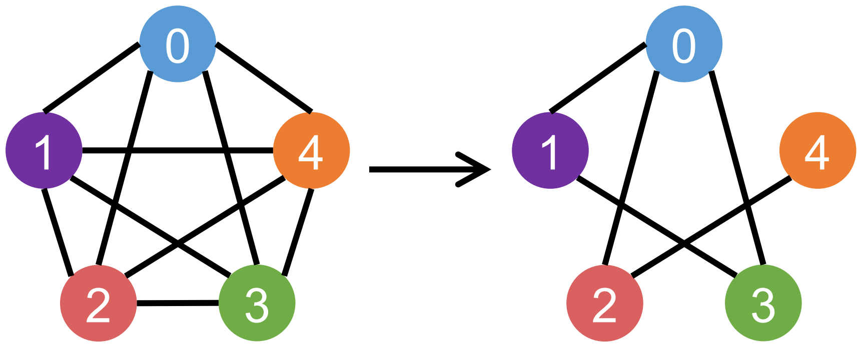


Approximate Analytics

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

graph

edge sampling
(p=0.5)



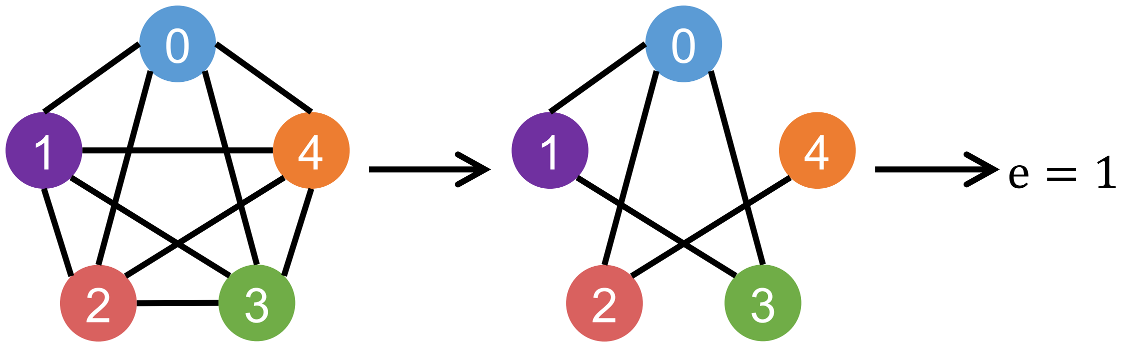
Approximate Analytics

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

graph

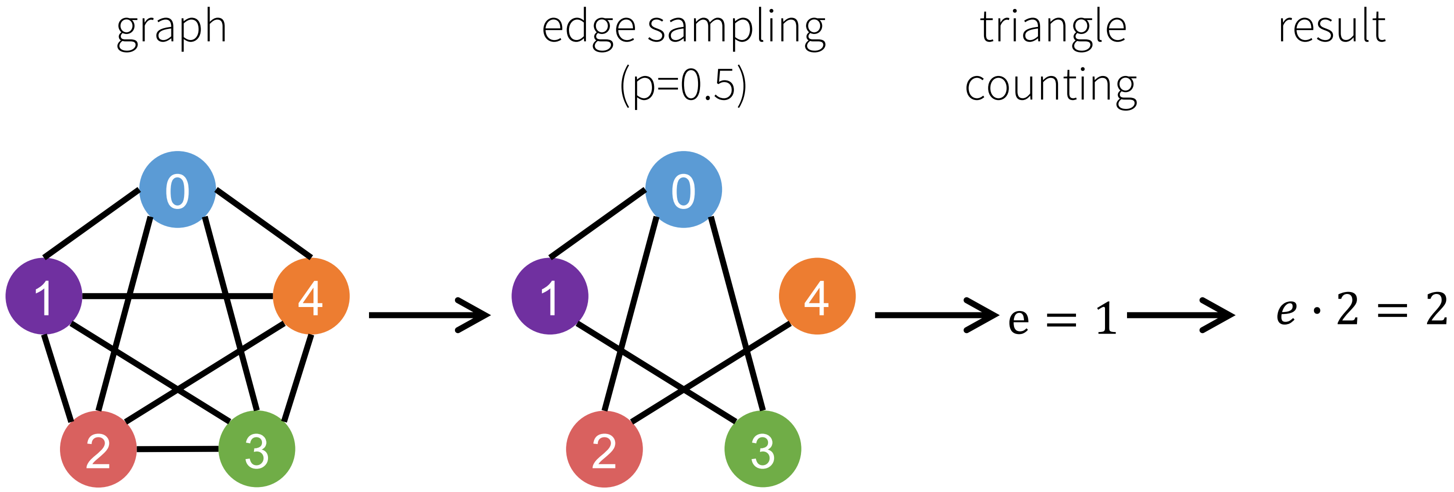
edge sampling (p=0.5)

triangle counting



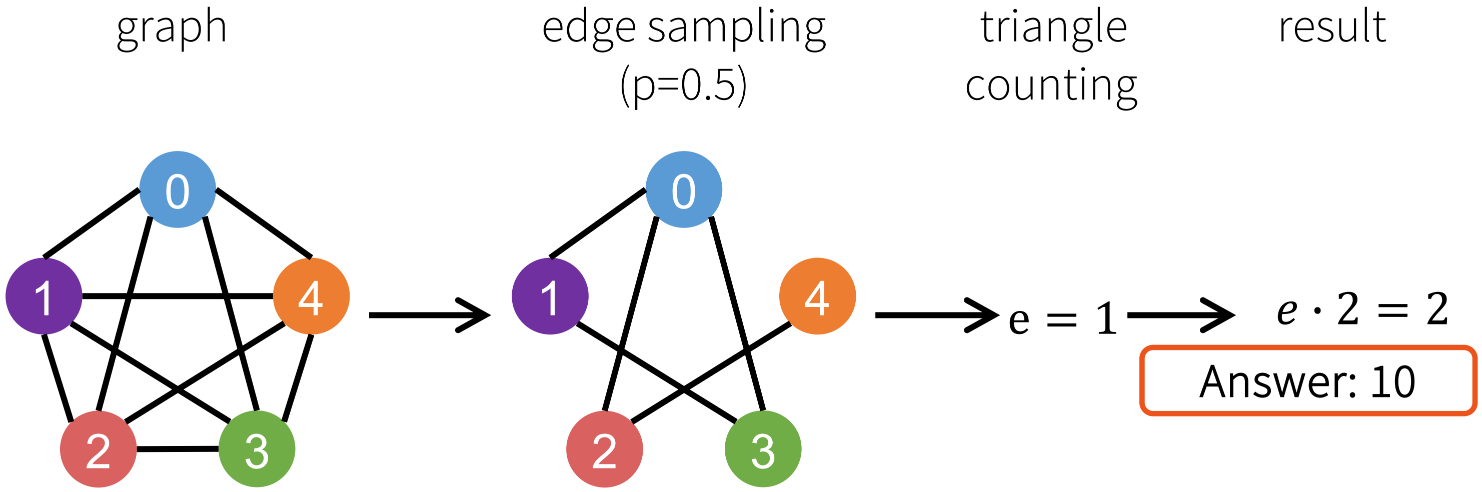
Approximate Analytics

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



Approximate Analytics

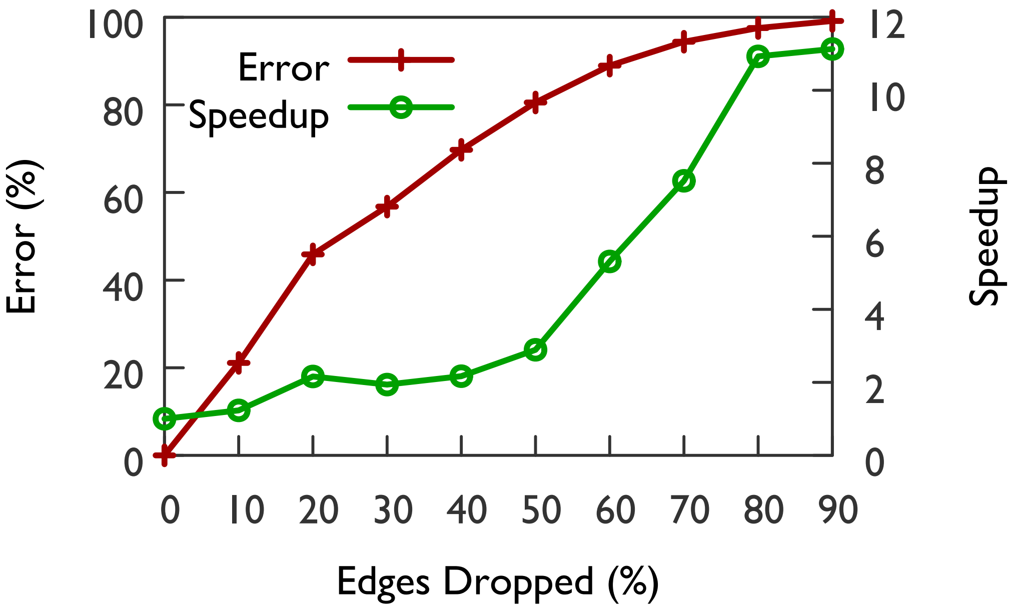
General approach: Apply algorithm on subset(s) (sample) of the input data



Approximate Analytics

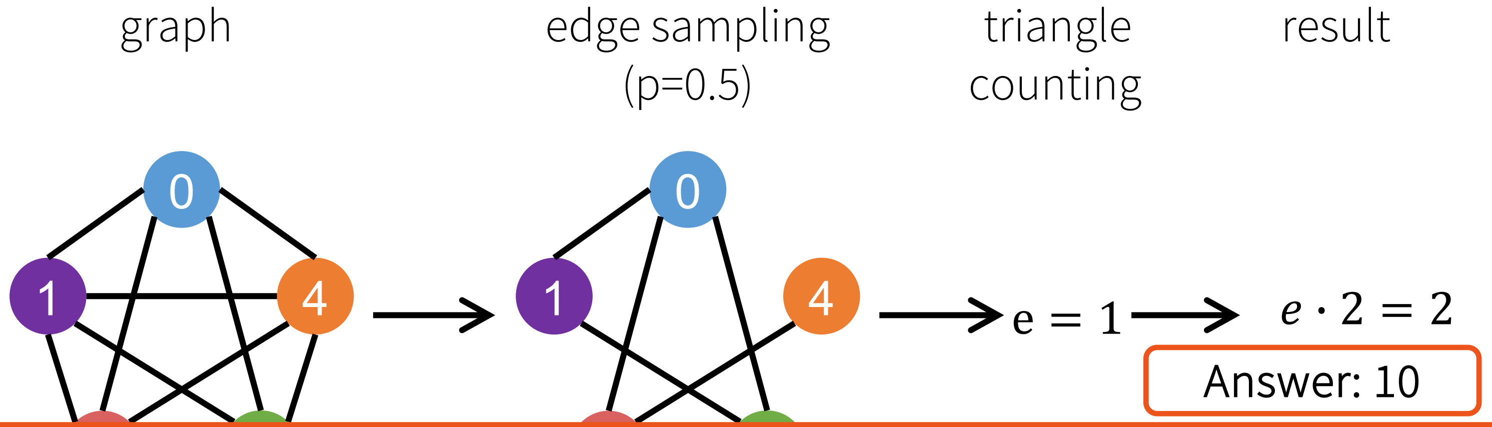
General approach: Apply algorithm on subset(s) (sample) of the input data

graph



Approximate Analytics

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



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

ASAP *leverages* existing work in graph approximation theory and makes it *practical*

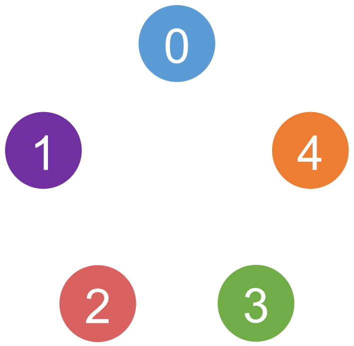
Graph Pattern Mining Theory

Sample instances of the pattern from the graph stream

Graph Pattern Mining Theory

Sample instances of the pattern from the graph stream

graph

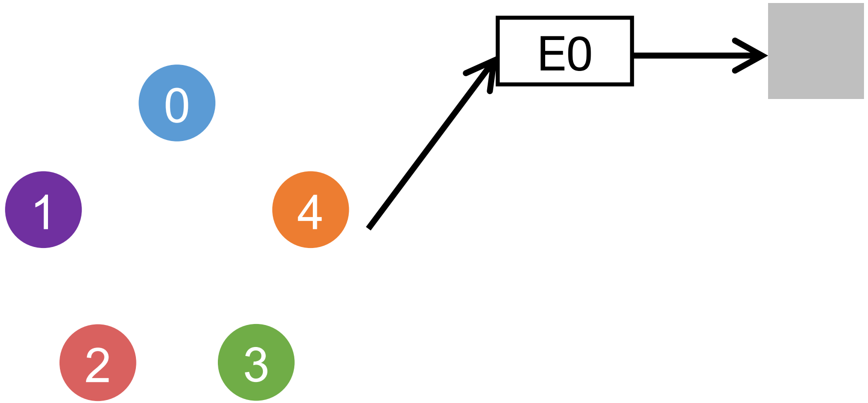


edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Graph Pattern Mining Theory

Sample instances of the pattern from the graph stream

graph

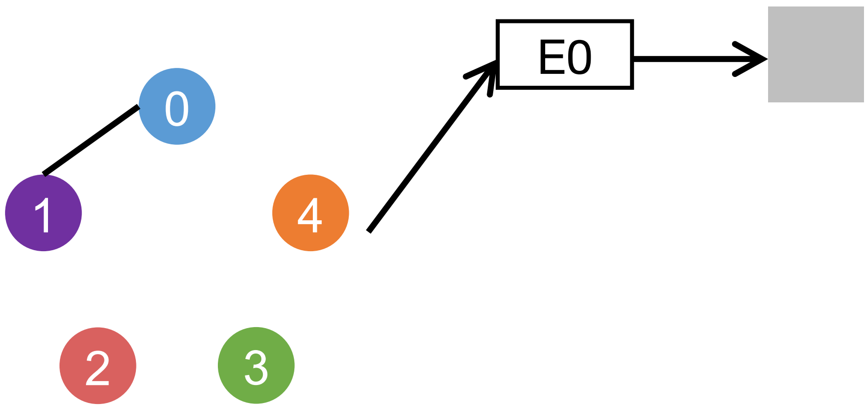


edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Graph Pattern Mining Theory

Sample instances of the pattern from the graph stream

graph

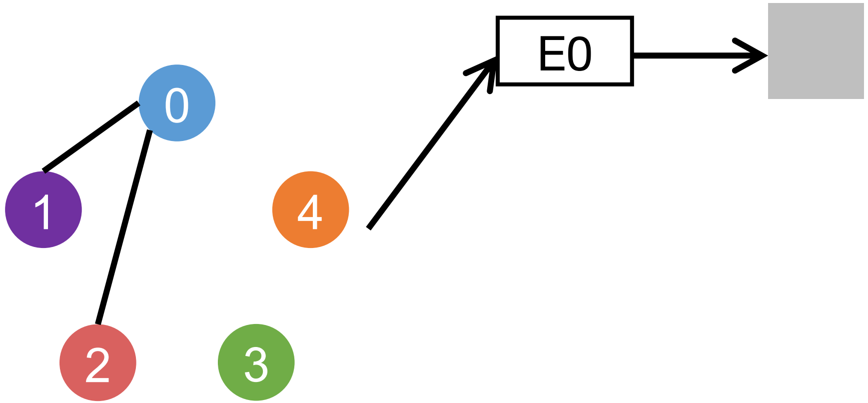


edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Graph Pattern Mining Theory

Sample instances of the pattern from the graph stream

graph

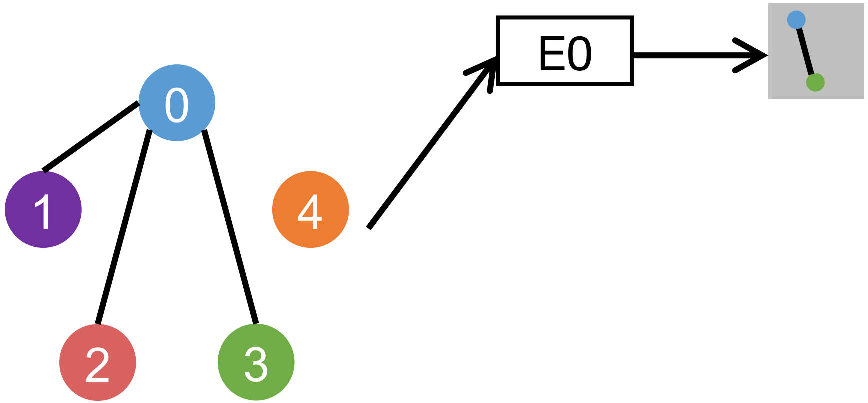


edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Graph Pattern Mining Theory

Sample instances of the pattern from the graph stream

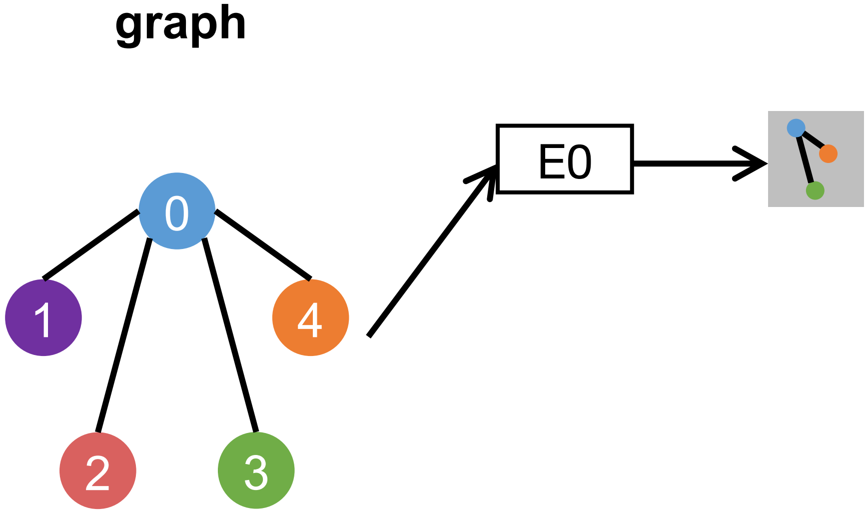
graph



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

Graph Pattern Mining Theory

Sample instances of the pattern from the graph stream

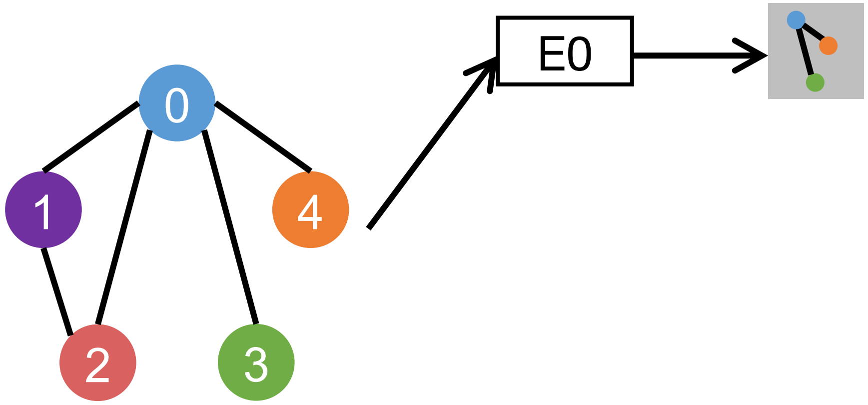


edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Graph Pattern Mining Theory

Sample instances of the pattern from the graph stream

graph

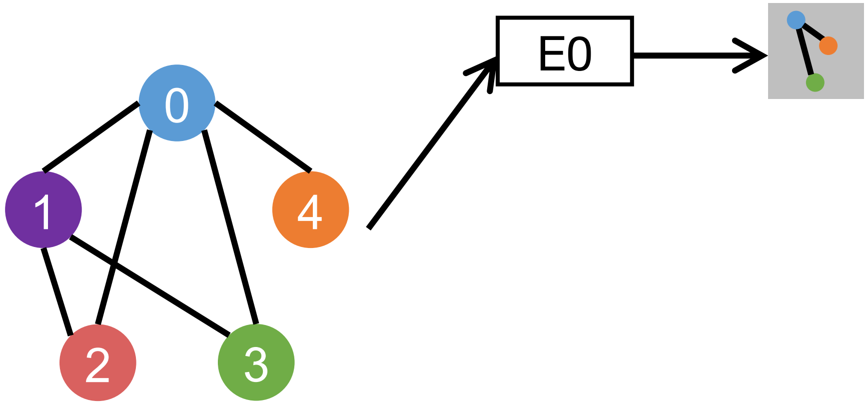


edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Graph Pattern Mining Theory

Sample instances of the pattern from the graph stream

graph

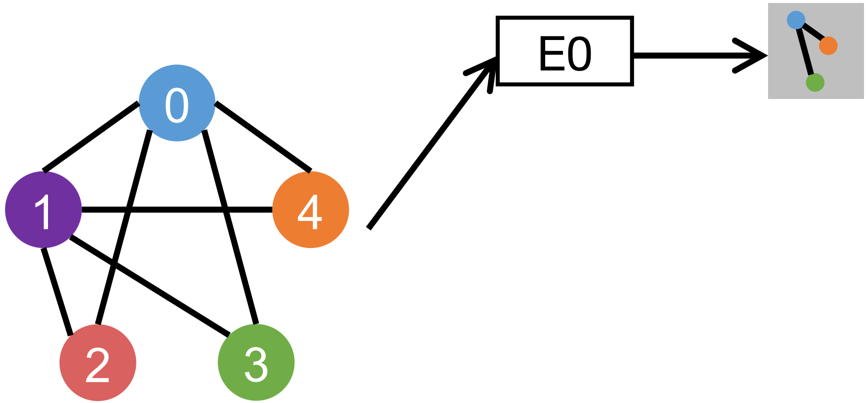


edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Graph Pattern Mining Theory

Sample instances of the pattern from the graph stream

graph

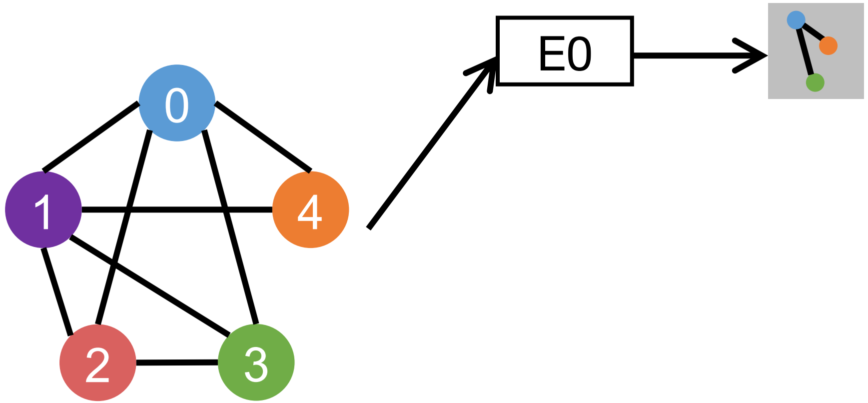


edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Graph Pattern Mining Theory

Sample instances of the pattern from the graph stream

graph

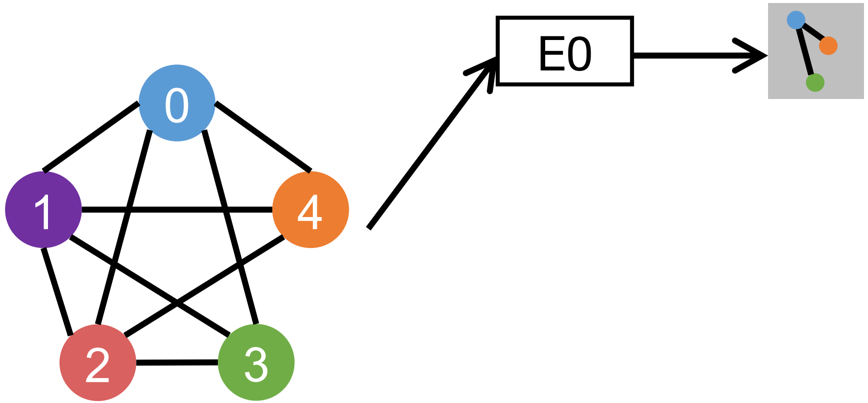


edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Graph Pattern Mining Theory

Sample instances of the pattern from the graph stream

graph

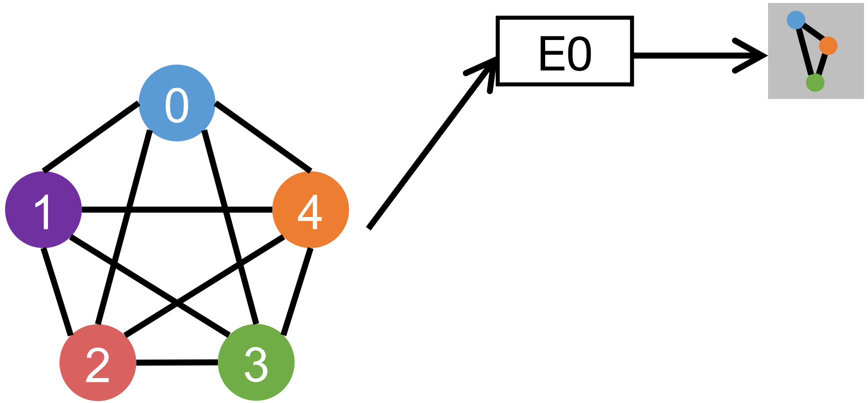


edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Graph Pattern Mining Theory

Sample instances of the pattern from the graph stream

graph



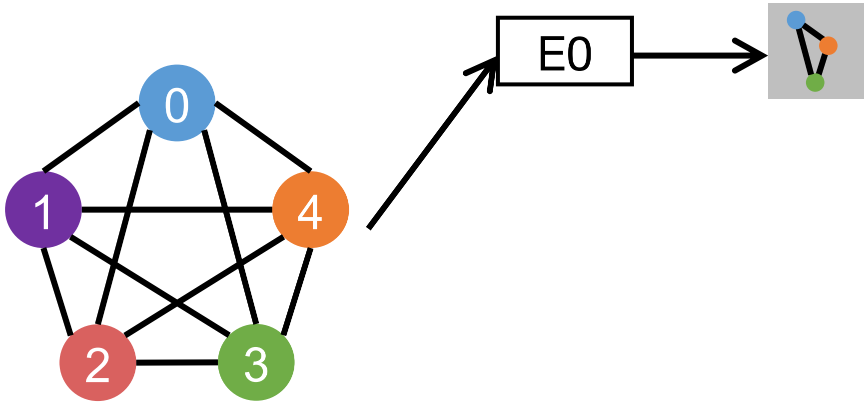
edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Graph Pattern Mining Theory

Sample instances of the pattern from the graph stream

graph

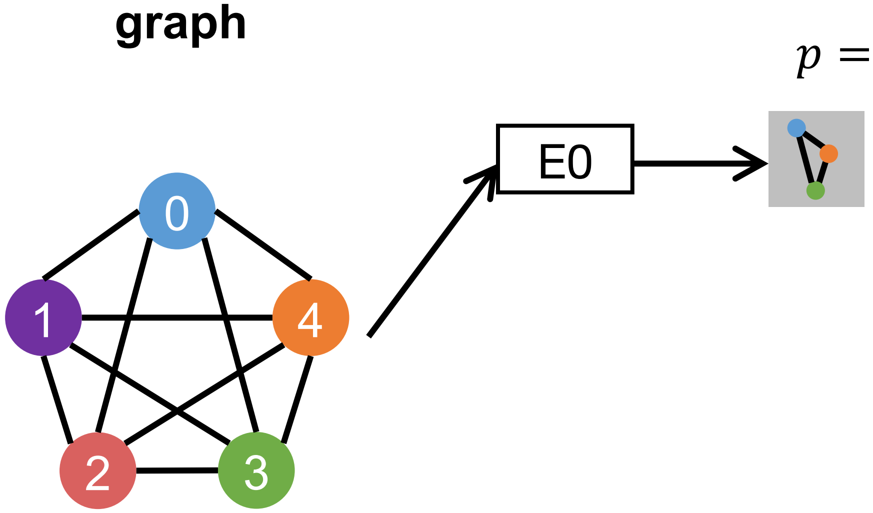
$$p = \frac{1}{10} * \frac{1}{4}$$



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

Graph Pattern Mining Theory

Sample instances of the pattern from the graph stream



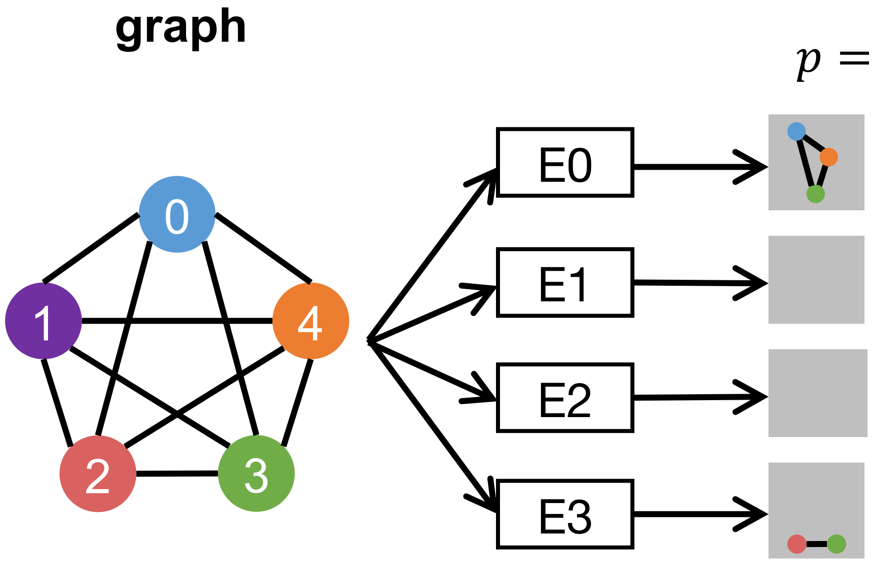
$$p = \frac{1}{10} * \frac{1}{4}$$

$$e_0 = 40$$

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

Graph Pattern Mining Theory

Sample instances of the pattern from the graph stream



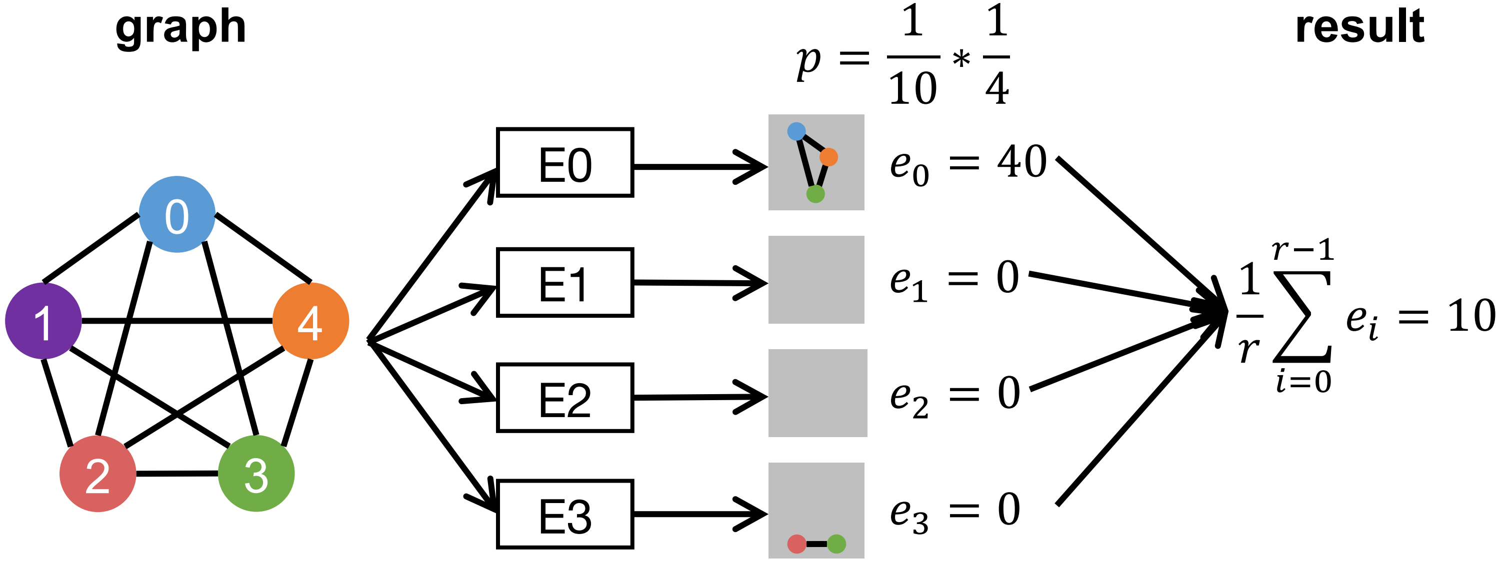
$$p = \frac{1}{10} * \frac{1}{4}$$

$$e_0 = 40$$

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

Graph Pattern Mining Theory

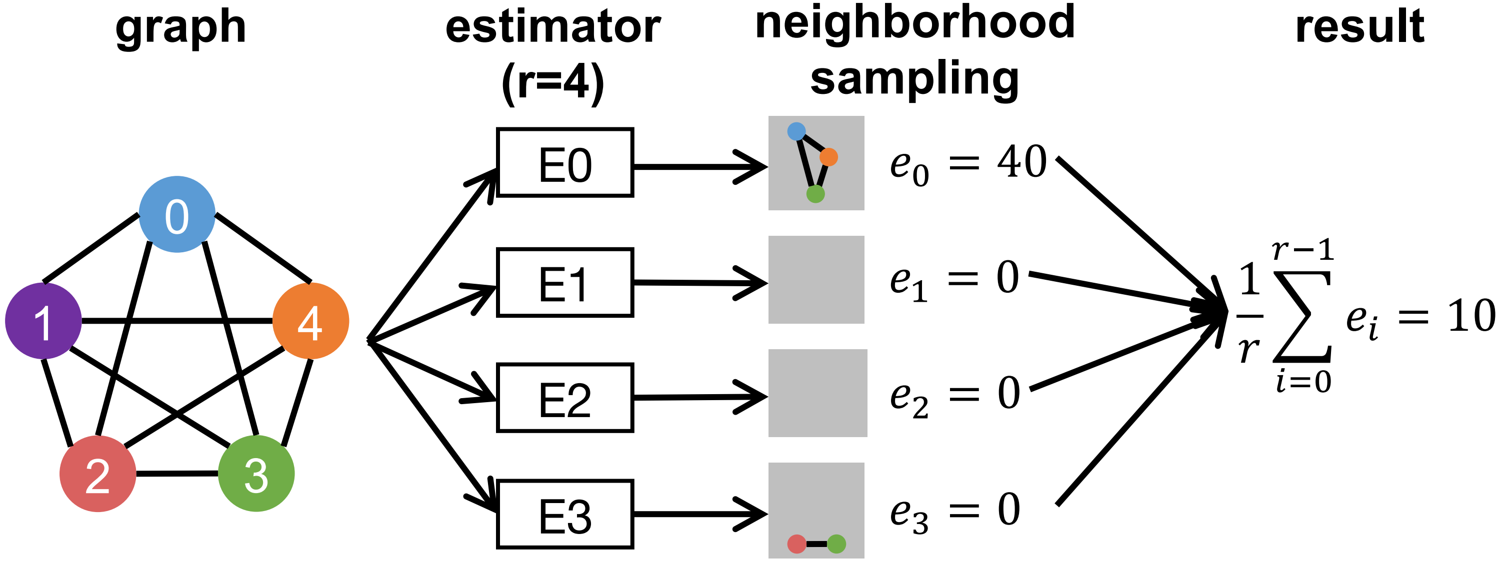
Sample instances of the pattern from the graph stream



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

Graph Pattern Mining Theory

Sample instances of the pattern from the graph stream



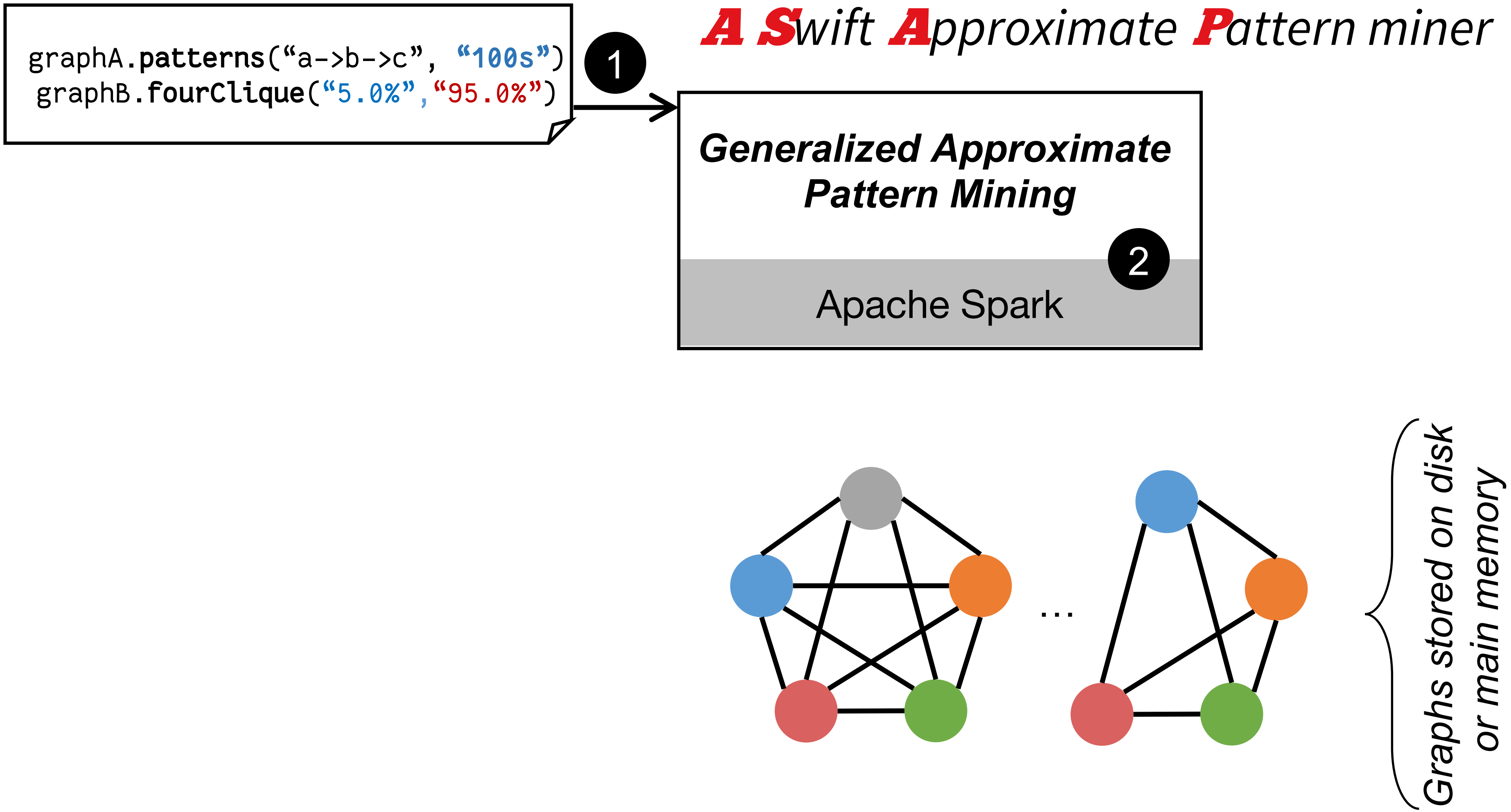
edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

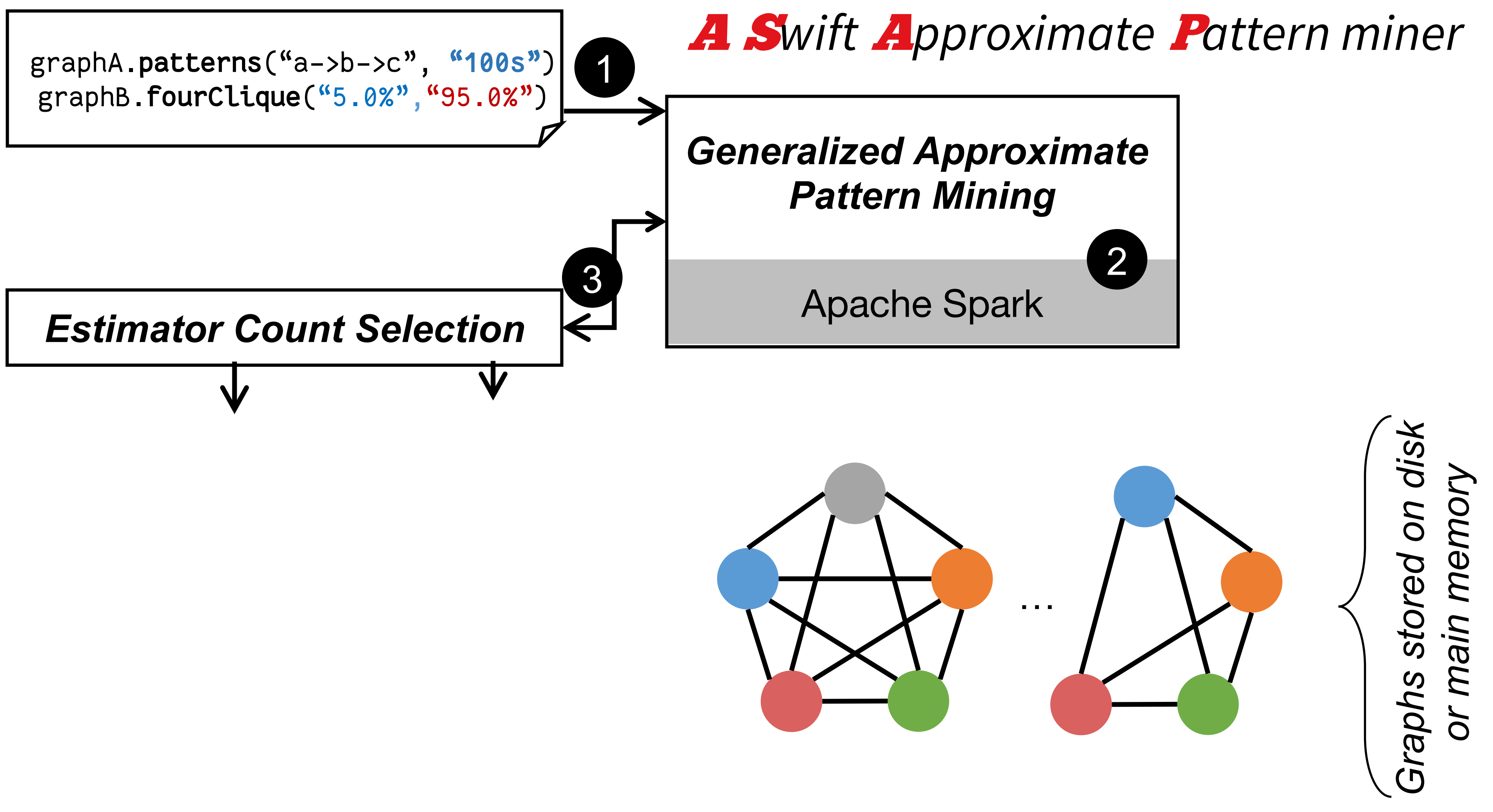
Pavan et al. Counting and sampling triangles from a graph stream, VLDB 2013

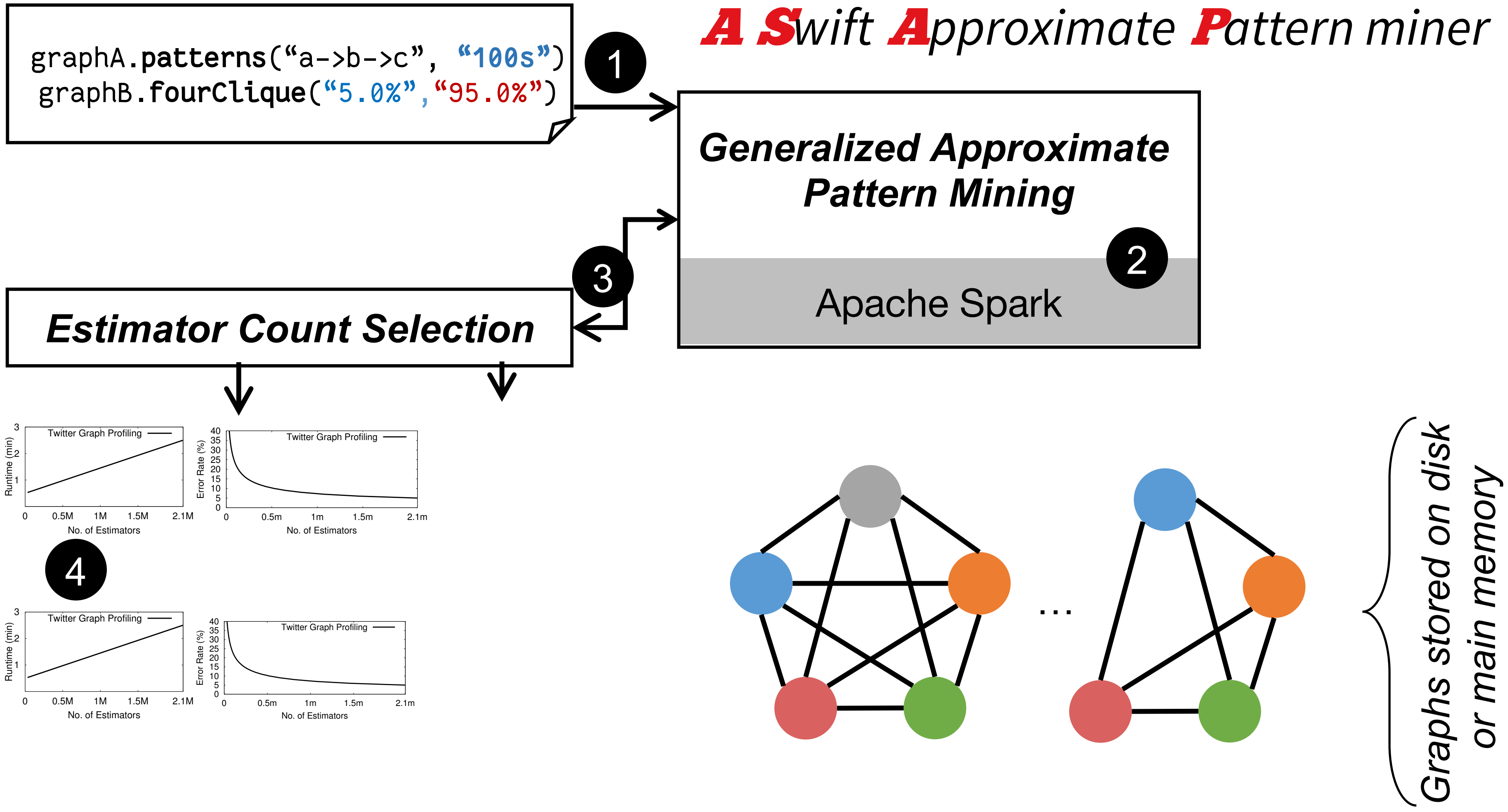
A Swift Approximate Pattern miner

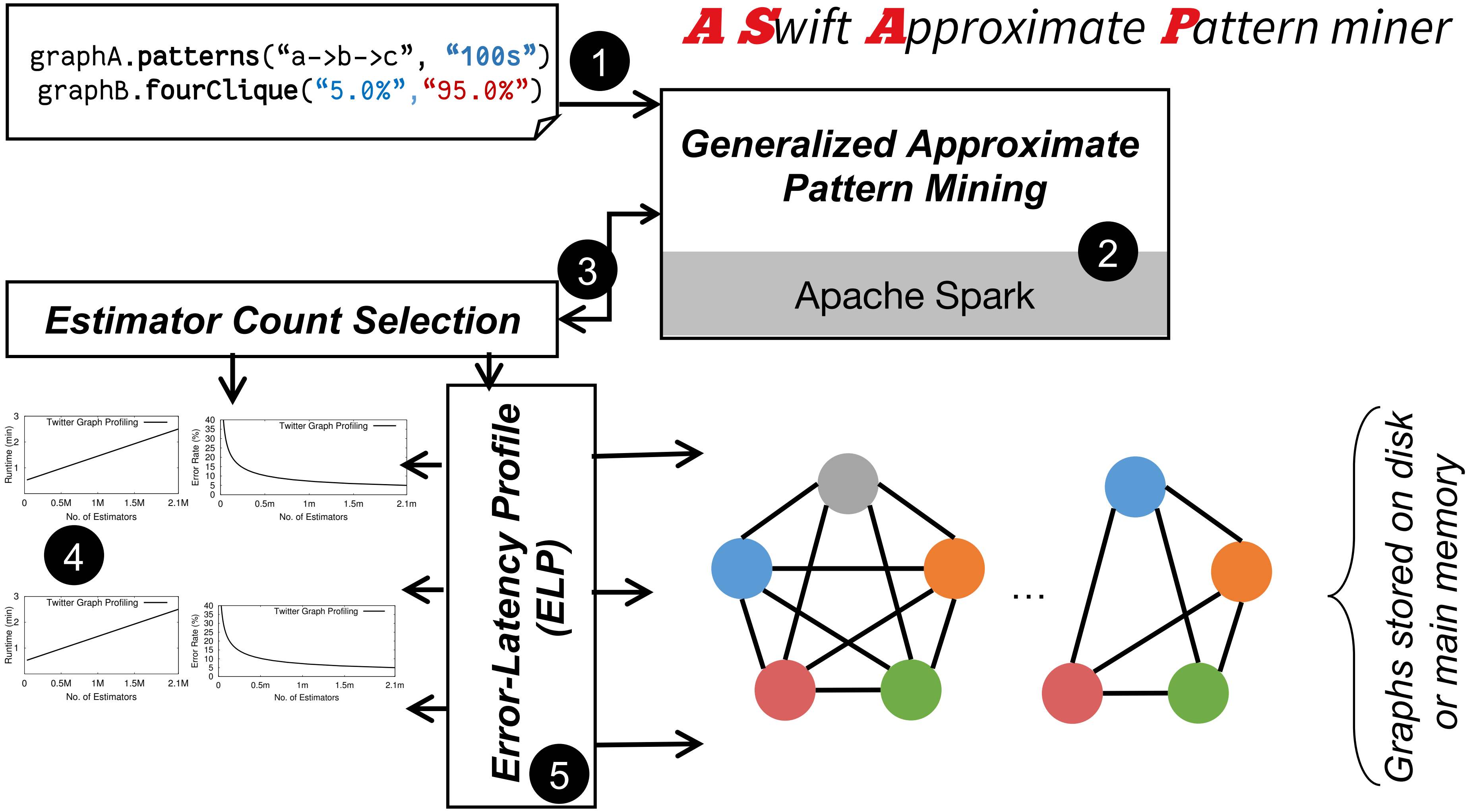
```
graphA.patterns("a->b->c", "100s")
graphB.fourClique("5.0%","95.0%")
```

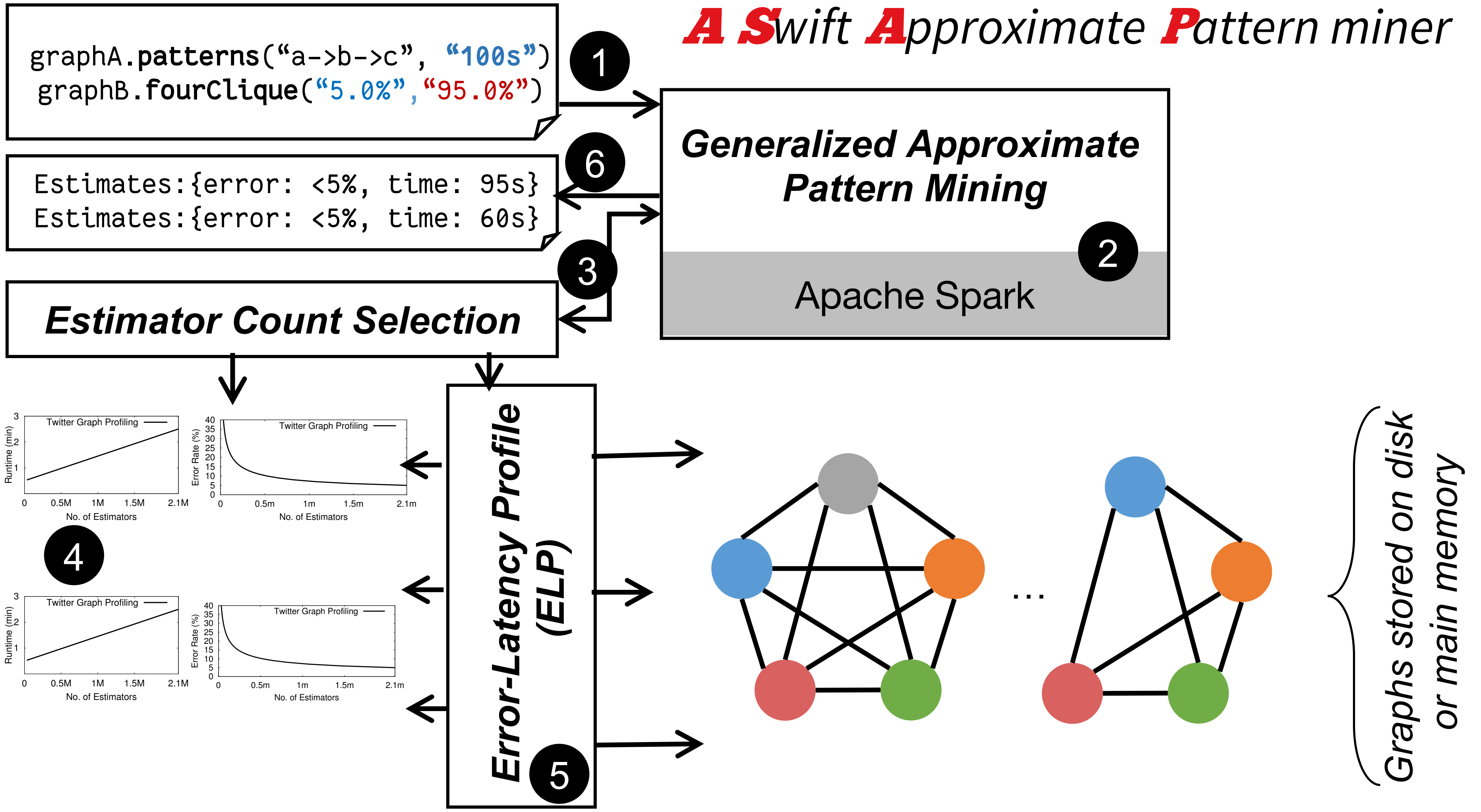
1


A Swift Approximate Pattern miner
graphA.patterns("a->b->c", "100s")
graphB.fourClique("5.0%","95.0%")
1
Generalized Approximate Pattern Mining
2
Apache Spark
...
Graphs stored on disk or main memory

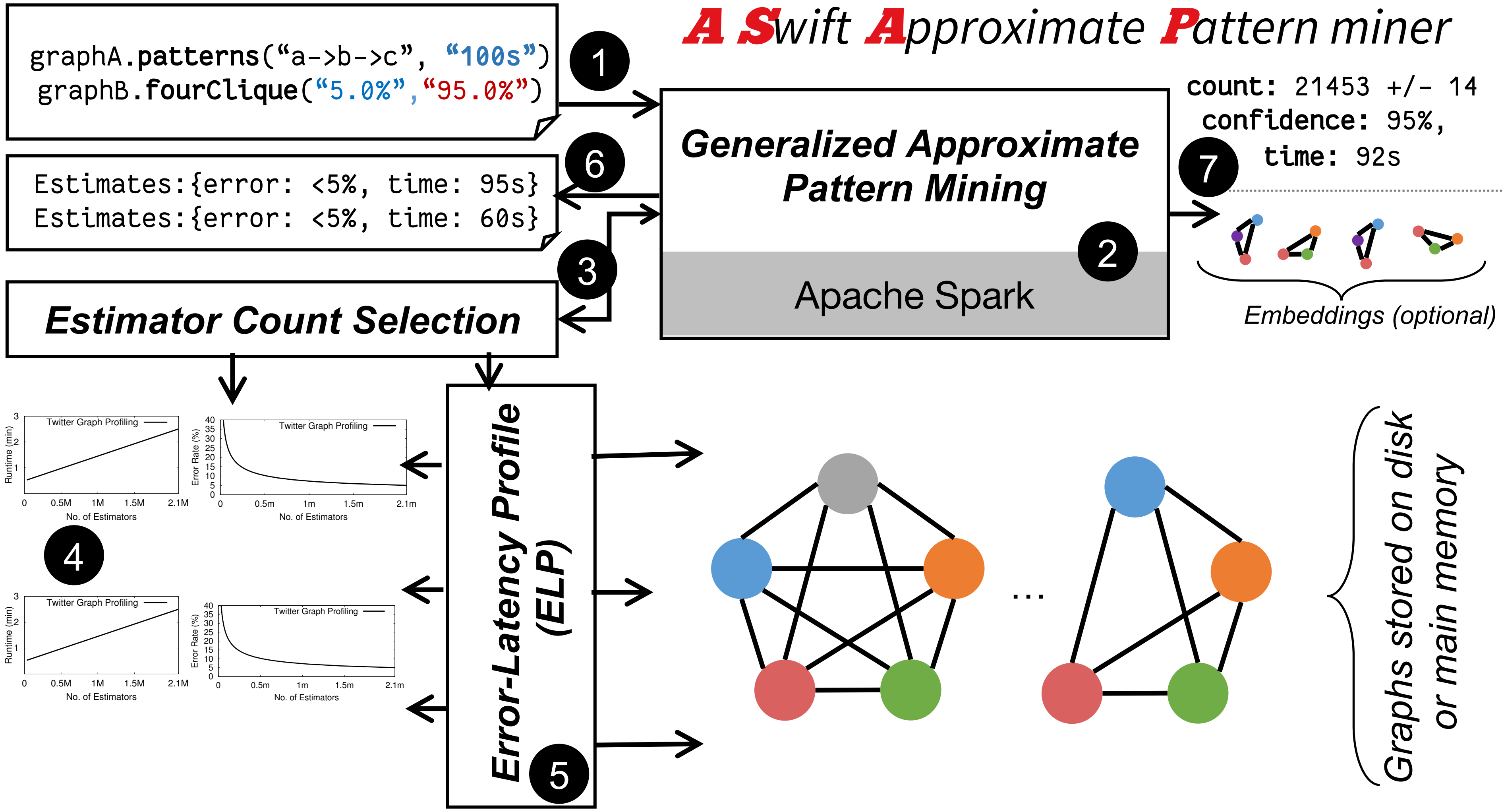

A Swift Approximate Pattern miner
graphA.patterns("a->b->c", "100s")
graphB.fourClique("5.0%","95.0%")
1
Generalized Approximate Pattern Mining
2
Apache Spark
3
Estimator Count Selection
...
Graphs stored on disk or main memory


A Swift Approximate Pattern miner
graphA.patterns("a->b->c", "100s")
graphB.fourClique("5.0%","95.0%")
1
Generalized Approximate Pattern Mining
2
Apache Spark
3
Estimator Count Selection
4
Twitter Graph Profiling
Runtime (min)
No. of Estimators
Error Rate (%)
...
Graphs stored on disk or main memory

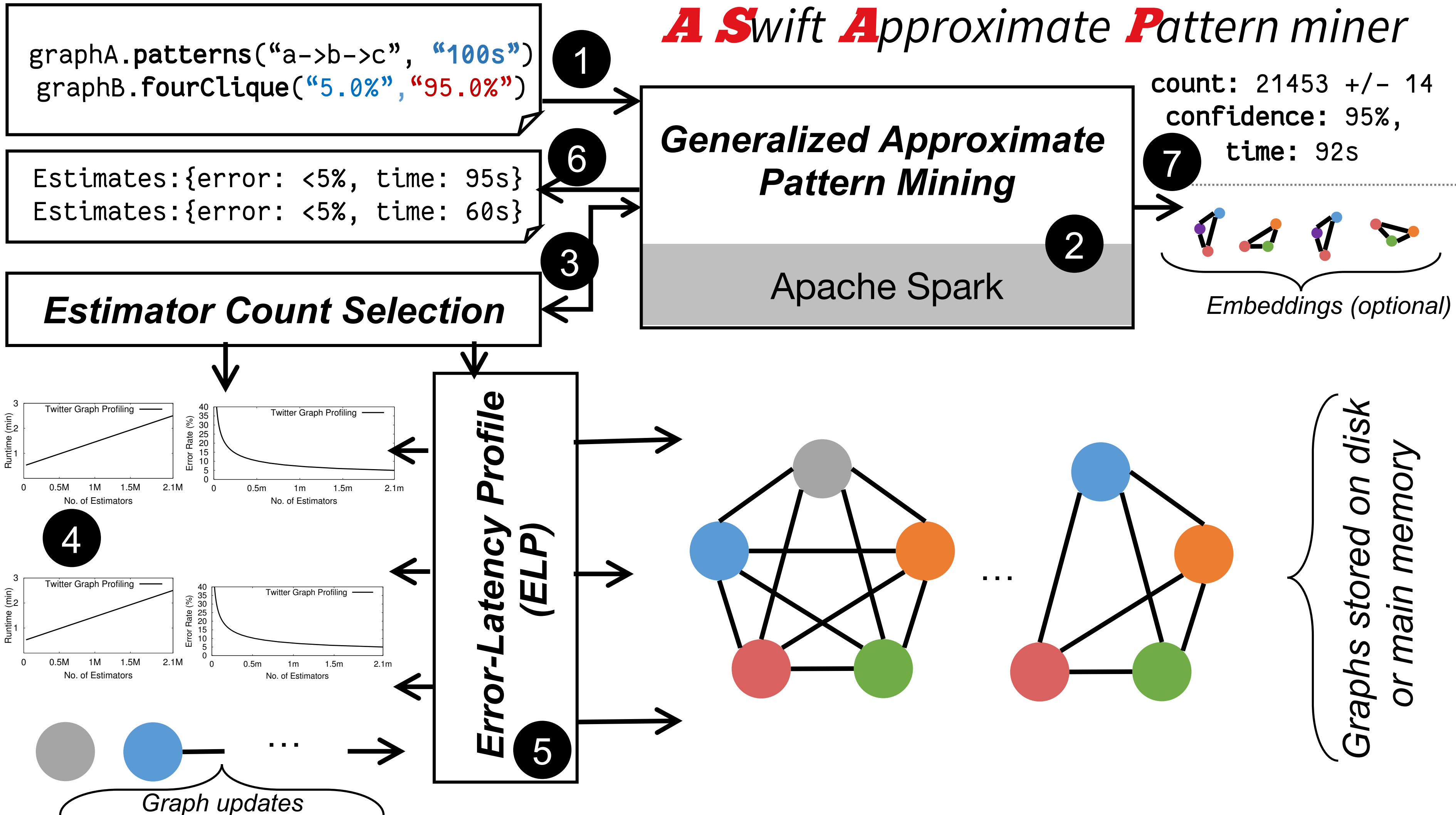

A Swift Approximate Pattern miner
graphA.patterns("a->b->c", "100s")
graphB.fourClique("5.0%","95.0%")
1
Generalized Approximate Pattern Mining
2
Apache Spark
3
Estimator Count Selection
Twitter Graph Profiling
Runtime (min)
No. of Estimators
Error Rate (%)
4
Error-Latency Profile (ELP)
5
...
Graphs stored on disk or main memory

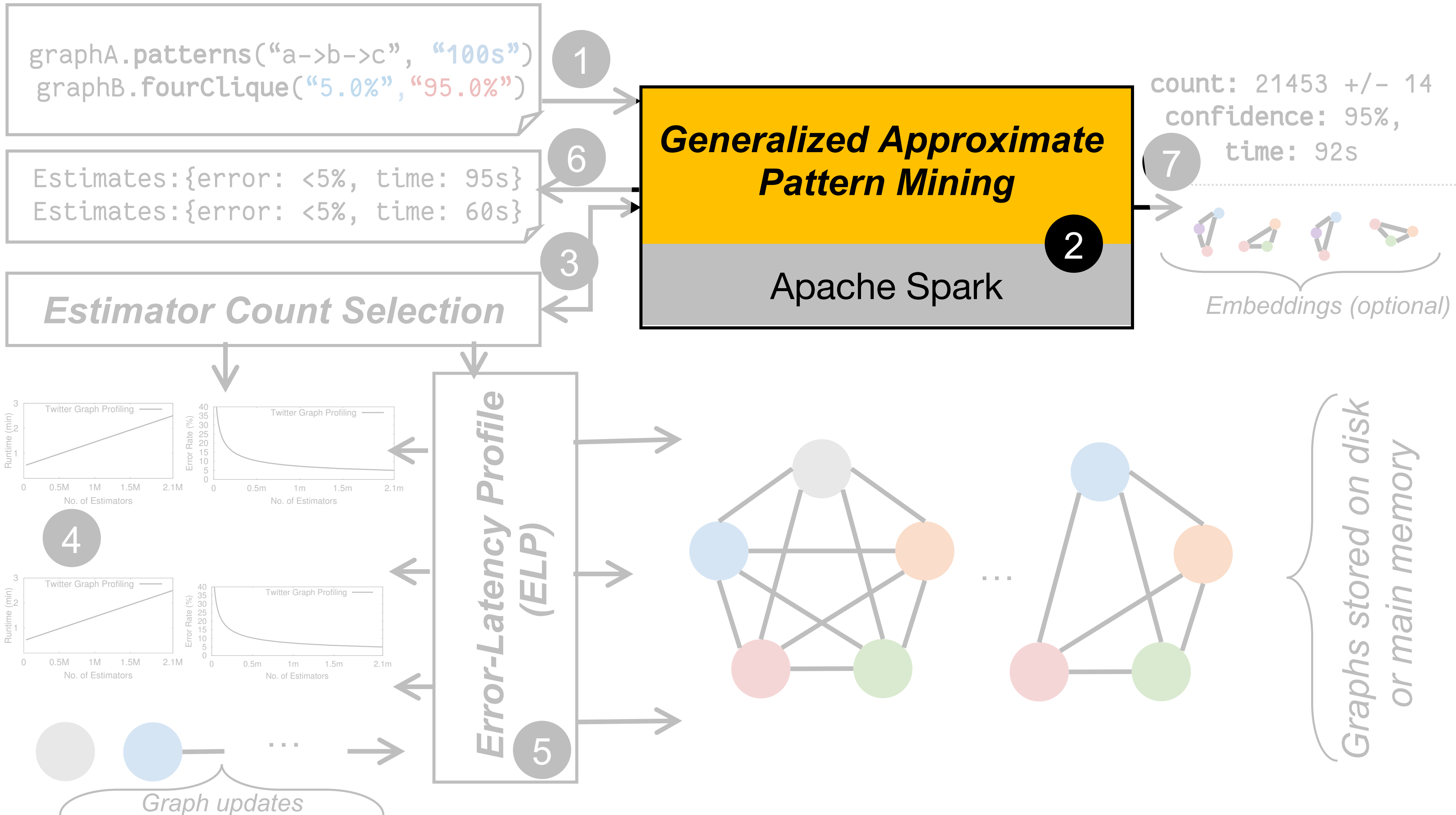


A Swift Approximate Pattern miner
graphA.patterns("a->b->c", "100s")
graphB.fourClique("5.0%","95.0%")
1
Generalized Approximate Pattern Mining
2
Apache Spark
6
Estimates:{error: <5%, time: 95s}
Estimates:{error: <5%, time: 60s}
3
Estimator Count Selection
Twitter Graph Profiling
Runtime (min)
No. of Estimators
Error Rate (%)
4
Error-Latency Profile (ELP)
5
Graphs stored on disk or main memory

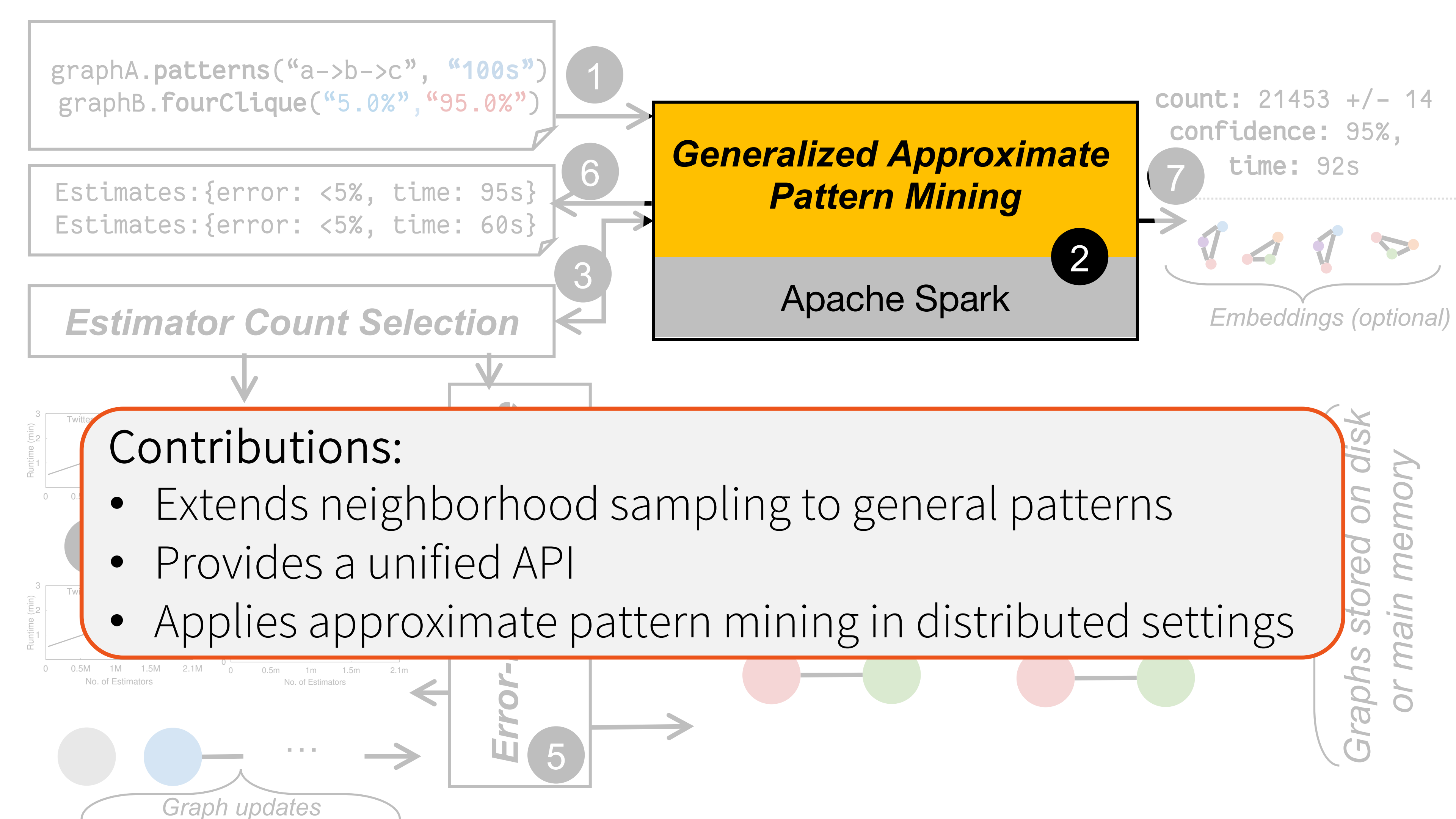


A Swift Approximate Pattern miner
graphA.patterns("a->b->c", "100s")
graphB.fourClique("5.0%","95.0%")
1
Generalized Approximate Pattern Mining
Apache Spark
2
count: 21453 +/- 14
confidence: 95%,
time: 92s
7
Embeddings (optional)
Estimates:{error: <5%, time: 95s}
Estimates:{error: <5%, time: 60s}
6
3
Estimator Count Selection
Twitter Graph Profiling
Runtime (min)
No. of Estimators
Error Rate (%)
4
Error-Latency Profile (ELP)
5
Graphs stored on disk or main memory


A Swift Approximate Pattern miner
graphA.patterns("a->b->c", "100s")
graphB.fourClique("5.0%","95.0%")
1
Generalized Approximate Pattern Mining
Apache Spark
2
count: 21453 +/- 14
confidence: 95%,
time: 92s
7
Embeddings (optional)
Estimates:{error: <5%, time: 95s}
Estimates:{error: <5%, time: 60s}
6
3
Estimator Count Selection
Twitter Graph Profiling
Runtime (min)
No. of Estimators
Twitter Graph Profiling
Error Rate (%)
No. of Estimators
4
Error-Latency Profile (ELP)
5
Graph updates
Graphs stored on disk or main memory



graphA.patterns("a->b->c", "100s")
graphB.fourClique("5.0%","95.0%")
1
Generalized Approximate Pattern Mining
2
Apache Spark
count: 21453 +/- 14
confidence: 95%,
time: 92s
7
Embeddings (optional)
Estimates:{error: <5%, time: 95s}
Estimates:{error: <5%, time: 60s}
6
3
Estimator Count Selection
Twitter Graph Profiling
Runtime (min)
No. of Estimators
Error Rate (%)
4
Error-Latency Profile (ELP)
5
Graphs stored on disk or main memory
Graph updates



graphA.patterns("a->b->c", "100s")
graphB.fourClique("5.0%","95.0%")
1
Generalized Approximate Pattern Mining
2
Apache Spark
count: 21453 +/- 14
confidence: 95%,
time: 92s
7
Embeddings (optional)
6
Estimates:{error: <5%, time: 95s}
Estimates:{error: <5%, time: 60s}
3
Estimator Count Selection
Contributions:
• Extends neighborhood sampling to general patterns
• Provides a unified API
• Applies approximate pattern mining in distributed settings
No. of Estimators
5
Graph updates
Graphs stored on disk or main memory

Generalized Approximate Pattern Mining

Developers write a single estimator using ASAP's API

API	**Description**
`sampleVertex: ()`$\rightarrow(v, p)$	Uniformly sample one vertex from the graph.
`SampleEdge: ()`$\rightarrow(e, p)$	Uniformly sample one edge from the graph.
`ConditionalSampleVertex: (subgraph)`$\rightarrow(v, p)$	Uniformly sample a vertex that appears after a sampled subgraph.
`ConditionalSampleEdge: (subgraph)`$\rightarrow(e, p)$	Uniformly sample an edge that is adjacent to the given subgraph and comes after the subgraph in the order.
`ConditionalClose: (subgraph, subgraph)`$\rightarrow boolean$	Given a sampled subgraph, check if another subgraph that appears later in the order can be formed.

Generalized Approximate Pattern Mining

Developers write a single estimator using ASAP's API

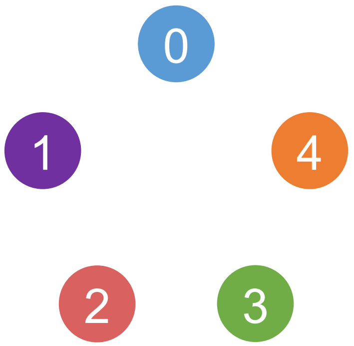
API	**Description**
`sampleVertex`: `()` $\rightarrow (v, p)$	Uniformly sample one vertex from the graph.
`SampleEdge`: `()` $\rightarrow (e, p)$	Uniformly sample one edge from the graph.
`ConditionalSampleVertex`: `(subgraph)` $\rightarrow (v, p)$	Uniformly sample a vertex that appears after a sampled subgraph.
`ConditionalSampleEdge`: `(subgraph)` $\rightarrow (e, p)$	Uniformly sample an edge that is adjacent to the given subgraph and comes after the subgraph in the order.
`ConditionalClose`: `(subgraph, subgraph)` $\rightarrow$ *boolean*	Given a sampled subgraph, check if another subgraph that appears later in the order can be formed.

Generalized Approximate Pattern Mining

Developers write a single estimator using ASAP's API

API	**Description**
`sampleVertex`: () $\rightarrow (v, p)$	Uniformly sample one vertex from the graph.
`SampleEdge`: () $\rightarrow (e, p)$	Uniformly sample one edge from the graph.
`ConditionalSampleVertex`: (`subgraph`) $\rightarrow (v, p)$	Uniformly sample a vertex that appears after a sampled subgraph.
`ConditionalSampleEdge`: (`subgraph`) $\rightarrow (e, p)$	Uniformly sample an edge that is adjacent to the given subgraph and comes after the subgraph in the order.
`ConditionalClose`: (`subgraph`, `subgraph`) $\rightarrow boolean$	Given a sampled subgraph, check if another subgraph that appears later in the order can be formed.

Using ASAP's API

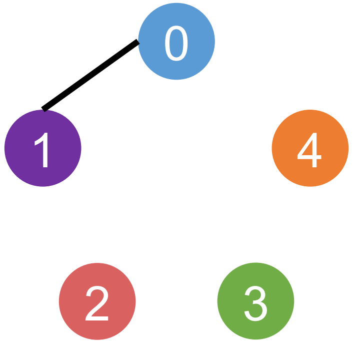


SampleTriangle

```
(e1, p1) = sampleEdge()
(e2, p2) = conditionalSampleEdge(Subgraph(e1))
if (!e2) return 0
subgraph1 = Subgraph(e1, e2)
subgraph2 = Triangle(e1, e2)-subgraph1
if conditionalClose(subgraph1, subgraph2)
  return 1/(p1.p2)
else return 0
```

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

Using ASAP's API

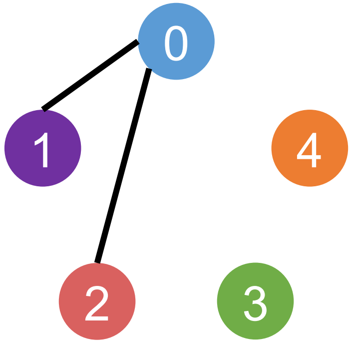


SampleTriangle

```
(e1, p1) = sampleEdge()
(e2, p2) = conditionalSampleEdge(Subgraph(e1))
if (!e2) return 0
subgraph1 = Subgraph(e1, e2)
subgraph2 = Triangle(e1, e2)-subgraph1
if conditionalClose(subgraph1, subgraph2)
  return 1/(p1.p2)
else return 0
```

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

Using ASAP's API

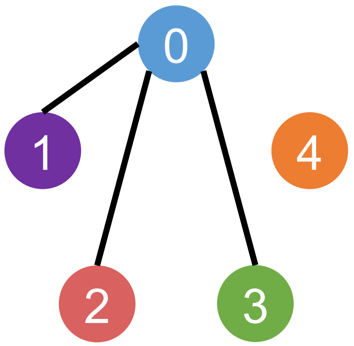


SampleTriangle

```
(e1, p1) = sampleEdge()
(e2, p2) = conditionalSampleEdge(Subgraph(e1))
if (!e2) return 0
subgraph1 = Subgraph(e1, e2)
subgraph2 = Triangle(e1, e2)-subgraph1
if conditionalClose(subgraph1, subgraph2)
  return 1/(p1.p2)
else return 0
```

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

Using ASAP's API

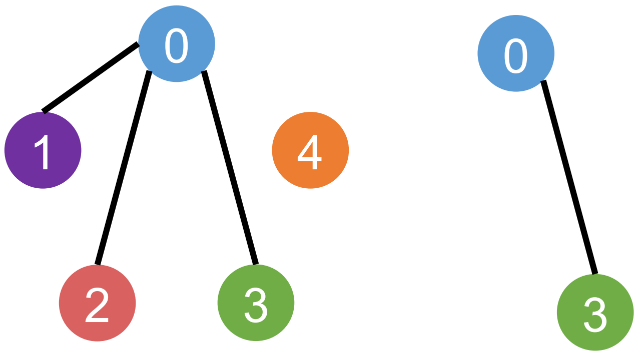


SampleTriangle

```
(e1, p1) = sampleEdge()
(e2, p2) = conditionalSampleEdge(Subgraph(e1))
if (!e2) return 0
subgraph1 = Subgraph(e1, e2)
subgraph2 = Triangle(e1, e2)-subgraph1
if conditionalClose(subgraph1, subgraph2)
  return 1/(p1.p2)
else return 0
```

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

Using ASAP's API

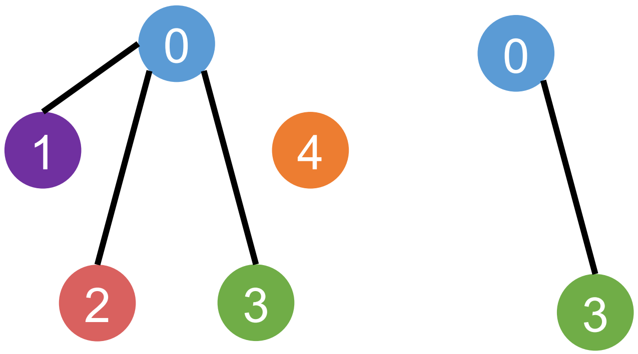


SampleTriangle

```
(e1, p1) = sampleEdge()
(e2, p2) = conditionalSampleEdge(Subgraph(e1))
if (!e2) return 0
subgraph1 = Subgraph(e1, e2)
subgraph2 = Triangle(e1, e2)-subgraph1
if conditionalClose(subgraph1, subgraph2)
  return 1/(p1.p2)
else return 0
```

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

Using ASAP's API

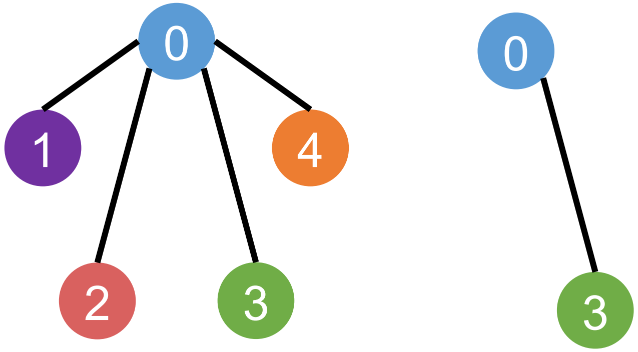


SampleTriangle

```
(e1, p1) = sampleEdge()
(e2, p2) = conditionalSampleEdge(Subgraph(e1))
if (!e2) return 0
subgraph1 = Subgraph(e1, e2)
subgraph2 = Triangle(e1, e2)-subgraph1
if conditionalClose(subgraph1, subgraph2)
  return 1/(p1.p2)
else return 0
```

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

Using ASAP's API

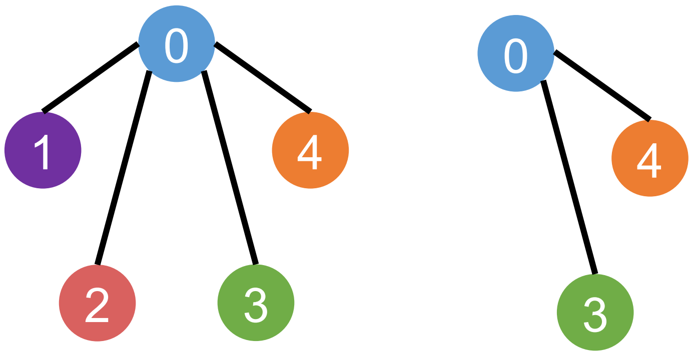


SampleTriangle

```
(e1, p1) = sampleEdge()
(e2, p2) = conditionalSampleEdge(Subgraph(e1))
if (!e2) return 0
subgraph1 = Subgraph(e1, e2)
subgraph2 = Triangle(e1, e2)-subgraph1
if conditionalClose(subgraph1, subgraph2)
  return 1/(p1.p2)
else return 0
```

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

Using ASAP's API

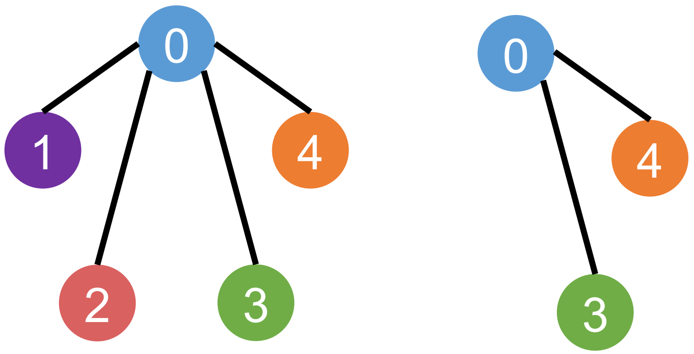


SampleTriangle

```
(e1, p1) = sampleEdge()
(e2, p2) = conditionalSampleEdge(Subgraph(e1))
if (!e2) return 0
subgraph1 = Subgraph(e1, e2)
subgraph2 = Triangle(e1, e2)-subgraph1
if conditionalClose(subgraph1, subgraph2)
  return 1/(p1.p2)
else return 0
```

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

Using ASAP's API

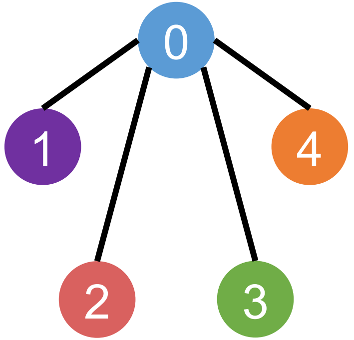


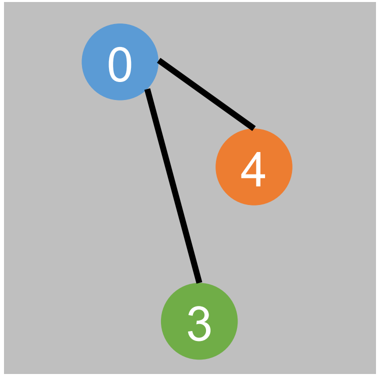
SampleTriangle

```
(e1, p1) = sampleEdge()
(e2, p2) = conditionalSampleEdge(Subgraph(e1))
if (!e2) return 0
subgraph1 = Subgraph(e1, e2)
subgraph2 = Triangle(e1, e2)-subgraph1
if conditionalClose(subgraph1, subgraph2)
  return 1/(p1.p2)
else return 0
```

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

Using ASAP's API



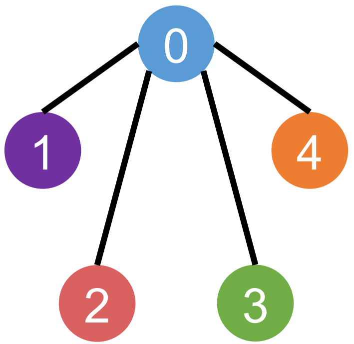


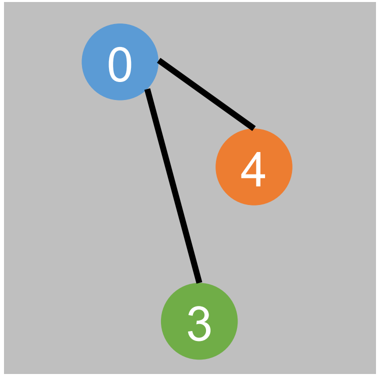
SampleTriangle

```
(e1, p1) = sampleEdge()
(e2, p2) = conditionalSampleEdge(Subgraph(e1))
if (!e2) return 0
subgraph1 = Subgraph(e1, e2)
subgraph2 = Triangle(e1, e2)-subgraph1
if conditionalClose(subgraph1, subgraph2)
  return 1/(p1.p2)
else return 0
```

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

Using ASAP's API



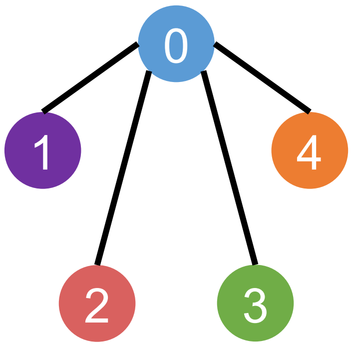


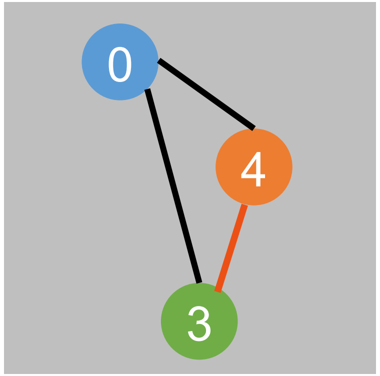
SampleTriangle

```
(e1, p1) = sampleEdge()
(e2, p2) = conditionalSampleEdge(Subgraph(e1))
if (!e2) return 0
subgraph1 = Subgraph(e1, e2)
subgraph2 = Triangle(e1, e2)-subgraph1
if conditionalClose(subgraph1, subgraph2)
  return 1/(p1.p2)
else return 0
```

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

Using ASAP's API



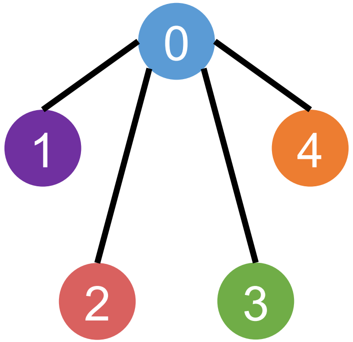


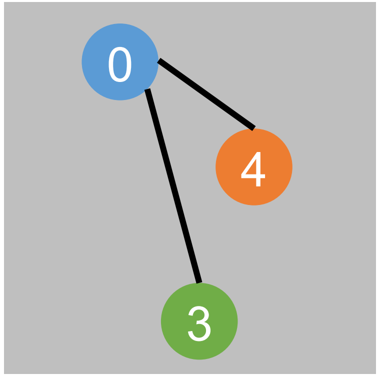
SampleTriangle

```
(e1, p1) = sampleEdge()
(e2, p2) = conditionalSampleEdge(Subgraph(e1))
if (!e2) return 0
subgraph1 = Subgraph(e1, e2)
subgraph2 = Triangle(e1, e2)-subgraph1
if conditionalClose(subgraph1, subgraph2)
  return 1/(p1.p2)
else return 0
```

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

Using ASAP's API



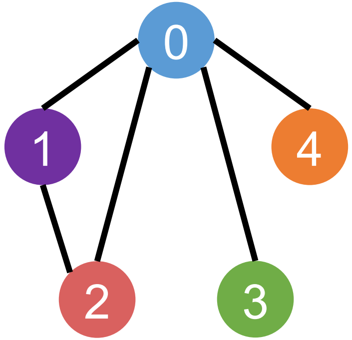


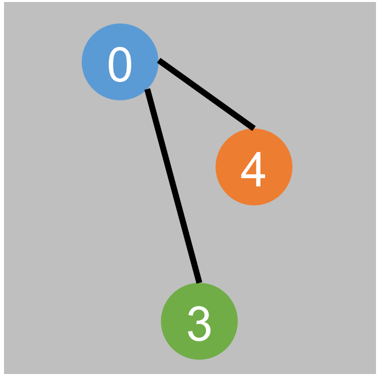
SampleTriangle

```
(e1, p1) = sampleEdge()
(e2, p2) = conditionalSampleEdge(Subgraph(e1))
if (!e2) return 0
subgraph1 = Subgraph(e1, e2)
subgraph2 = Triangle(e1, e2)-subgraph1
if conditionalClose(subgraph1, subgraph2)
  return 1/(p1.p2)
else return 0
```

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

Using ASAP’s API



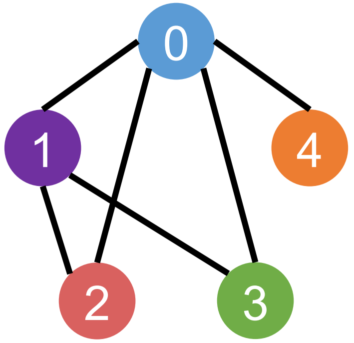


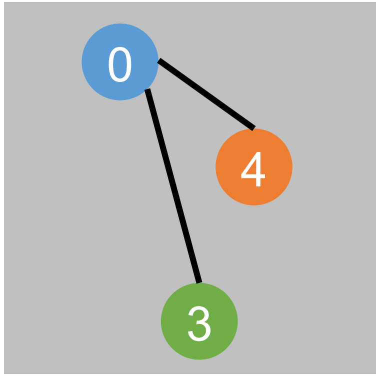
SampleTriangle

```
(e1, p1) = sampleEdge()
(e2, p2) = conditionalSampleEdge(Subgraph(e1))
if (!e2) return 0
subgraph1 = Subgraph(e1, e2)
subgraph2 = Triangle(e1, e2)-subgraph1
if conditionalClose(subgraph1, subgraph2)
  return 1/(p1.p2)
else return 0
```

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

Using ASAP's API



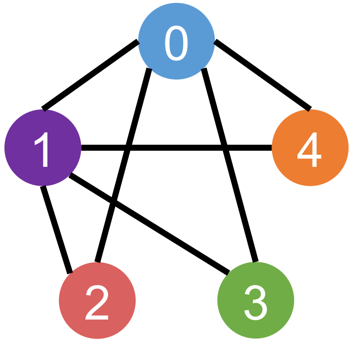


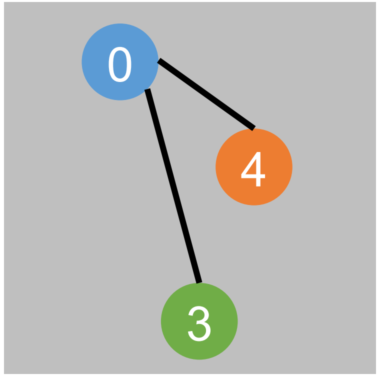
SampleTriangle

```
(e1, p1) = sampleEdge()
(e2, p2) = conditionalSampleEdge(Subgraph(e1))
if (!e2) return 0
subgraph1 = Subgraph(e1, e2)
subgraph2 = Triangle(e1, e2)-subgraph1
if conditionalClose(subgraph1, subgraph2)
  return 1/(p1.p2)
else return 0
```

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

Using ASAP's API



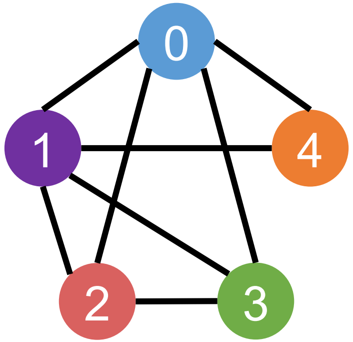


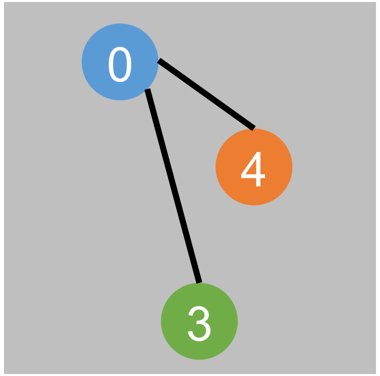
SampleTriangle

```
(e1, p1) = sampleEdge()
(e2, p2) = conditionalSampleEdge(Subgraph(e1))
if (!e2) return 0
subgraph1 = Subgraph(e1, e2)
subgraph2 = Triangle(e1, e2)-subgraph1
if conditionalClose(subgraph1, subgraph2)
  return 1/(p1.p2)
else return 0
```

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

Using ASAP's API



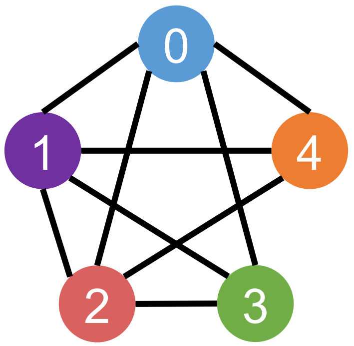


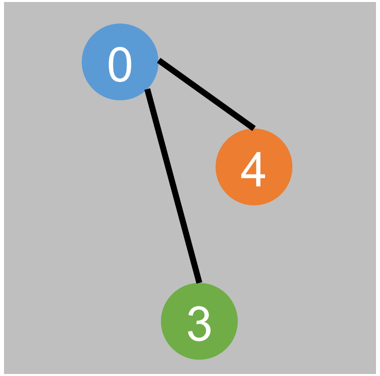
SampleTriangle

```
(e1, p1) = sampleEdge()
(e2, p2) = conditionalSampleEdge(Subgraph(e1))
if (!e2) return 0
subgraph1 = Subgraph(e1, e2)
subgraph2 = Triangle(e1, e2)-subgraph1
if conditionalClose(subgraph1, subgraph2)
  return 1/(p1.p2)
else return 0
```

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

Using ASAP's API



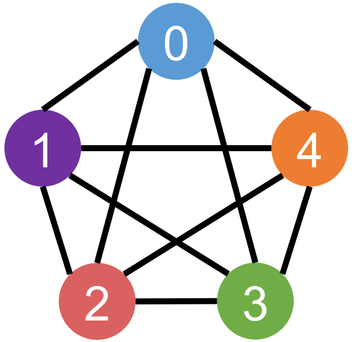


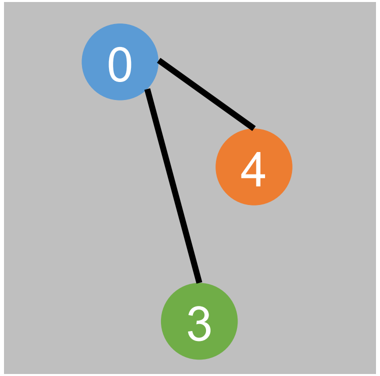
SampleTriangle

```
(e1, p1) = sampleEdge()
(e2, p2) = conditionalSampleEdge(Subgraph(e1))
if (!e2) return 0
subgraph1 = Subgraph(e1, e2)
subgraph2 = Triangle(e1, e2)-subgraph1
if conditionalClose(subgraph1, subgraph2)
  return 1/(p1.p2)
else return 0
```

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

Using ASAP's API



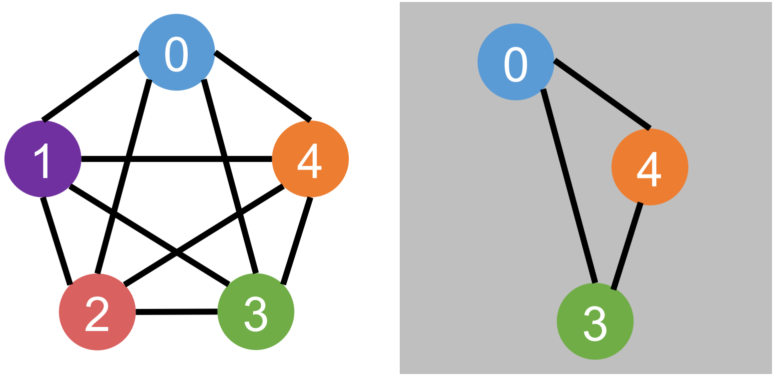


SampleTriangle

```
(e1, p1) = sampleEdge()
(e2, p2) = conditionalSampleEdge(Subgraph(e1))
if (!e2) return 0
subgraph1 = Subgraph(e1, e2)
subgraph2 = Triangle(e1, e2)-subgraph1
if conditionalClose(subgraph1, subgraph2)
  return 1/(p1.p2)
else return 0
```

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

Using ASAP's API

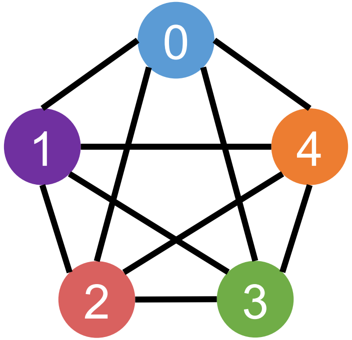


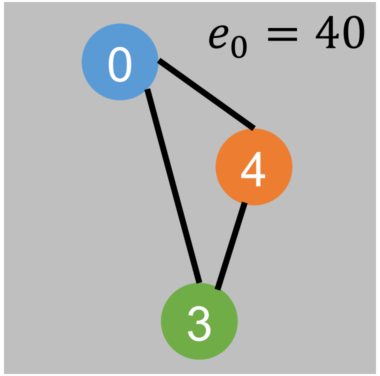
SampleTriangle

```
(e1, p1) = sampleEdge()
(e2, p2) = conditionalSampleEdge(Subgraph(e1))
if (!e2) return 0
subgraph1 = Subgraph(e1, e2)
subgraph2 = Triangle(e1, e2)-subgraph1
if conditionalClose(subgraph1, subgraph2)
  return 1/(p1.p2)
else return 0
```

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

Using ASAP's API



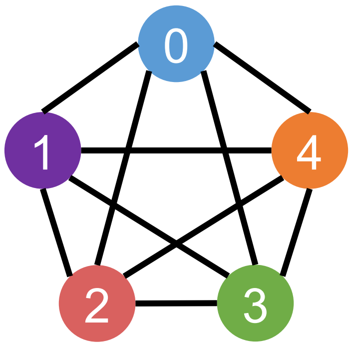


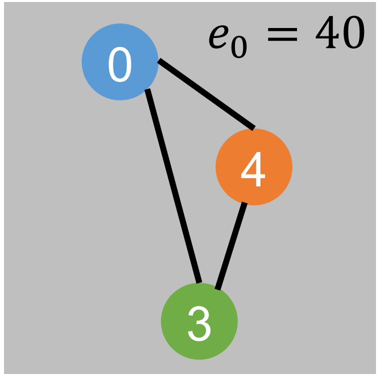
SampleTriangle

```
(e1, p1) = sampleEdge()
(e2, p2) = conditionalSampleEdge(Subgraph(e1))
if (!e2) return 0
subgraph1 = Subgraph(e1, e2)
subgraph2 = Triangle(e1, e2)-subgraph1
if conditionalClose(subgraph1, subgraph2)
  return 1/(p1.p2)
else return 0
```

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

Using ASAP's API





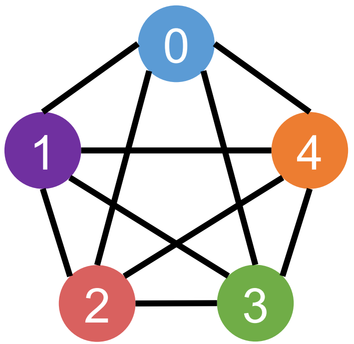
SampleTriangle

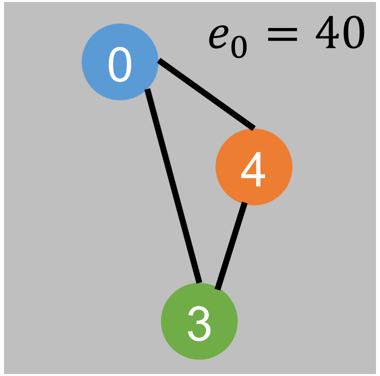
```
(e1, p1) = sampleEdge()
(e2, p2) = conditionalSampleEdge(Subgraph(e1))
if (!e2) return 0
subgraph1 = Subgraph(e1, e2)
subgraph2 = Triangle(e1, e2)-subgraph1
if conditionalClose(subgraph1, subgraph2)
  return 1/(p1.p2)
else return 0
```

Sampling phase *fixes the vertices* for a particular instance of a pattern and closing phase *waits for remaining edges*

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

Using ASAP's API





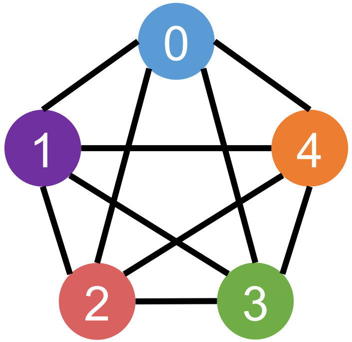
SampleTriangle

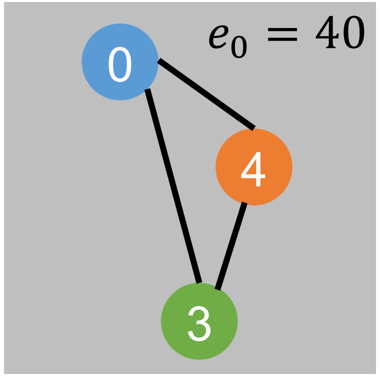
```
(e1, p1) = sampleEdge()
(e2, p2) = conditionalSampleEdge(Subgraph(e1))
if (!e2) return 0
subgraph1 = Subgraph(e1, e2)
subgraph2 = Triangle(e1, e2)-subgraph1
if conditionalClose(subgraph1, subgraph2)
  return 1/(p1.p2)
else return 0
```

ASAP computes the right expectations, runs many instances of the estimator and aggregates results

Using ASAP's API

See paper for more examples & proof



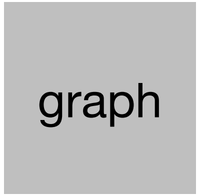


SampleTriangle

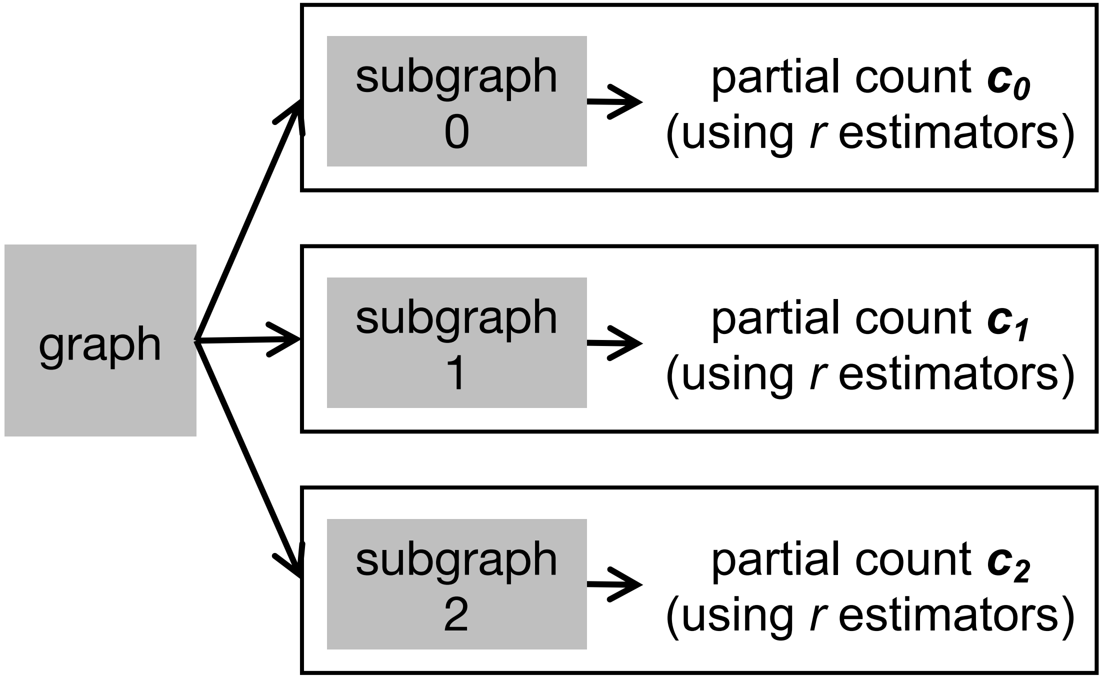
```
(e1, p1) = sampleEdge()
(e2, p2) = conditionalSampleEdge(Subgraph(e1))
if (!e2) return 0
subgraph1 = Subgraph(e1, e2)
subgraph2 = Triangle(e1, e2)-subgraph1
if conditionalClose(subgraph1, subgraph2)
  return 1/(p1.p2)
else return 0
```

ASAP computes the right expectations, runs many instances of the estimator and aggregates results

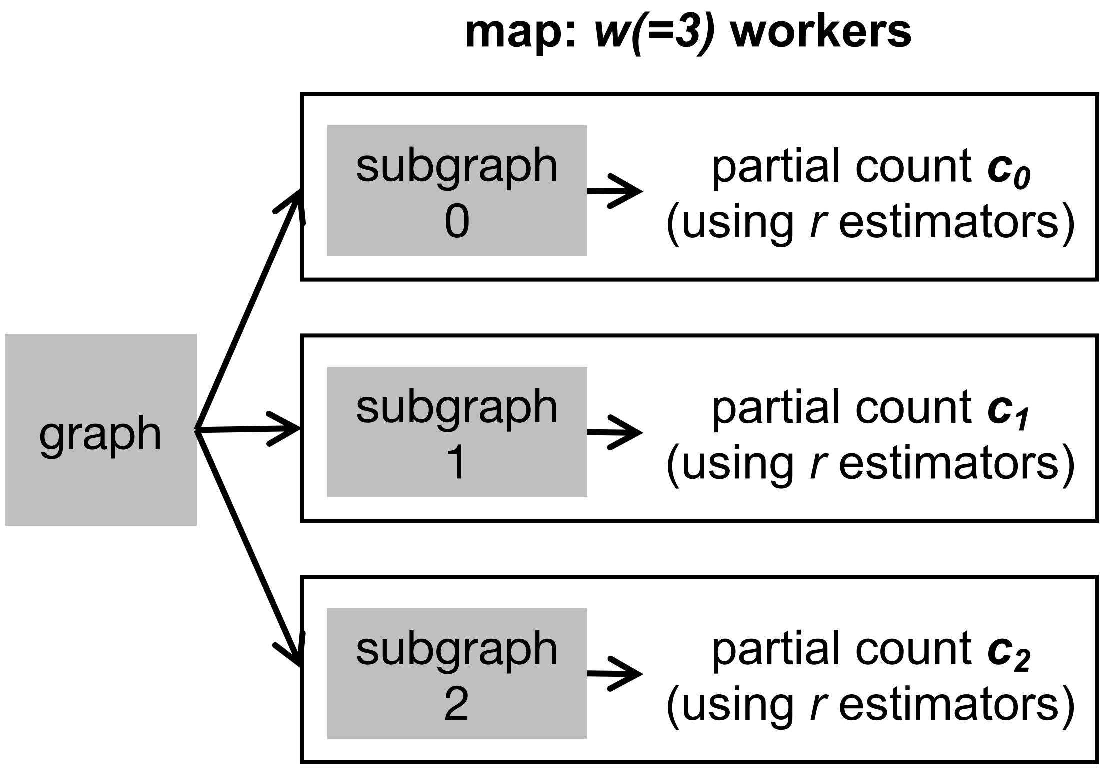
Applying to Distributed Settings



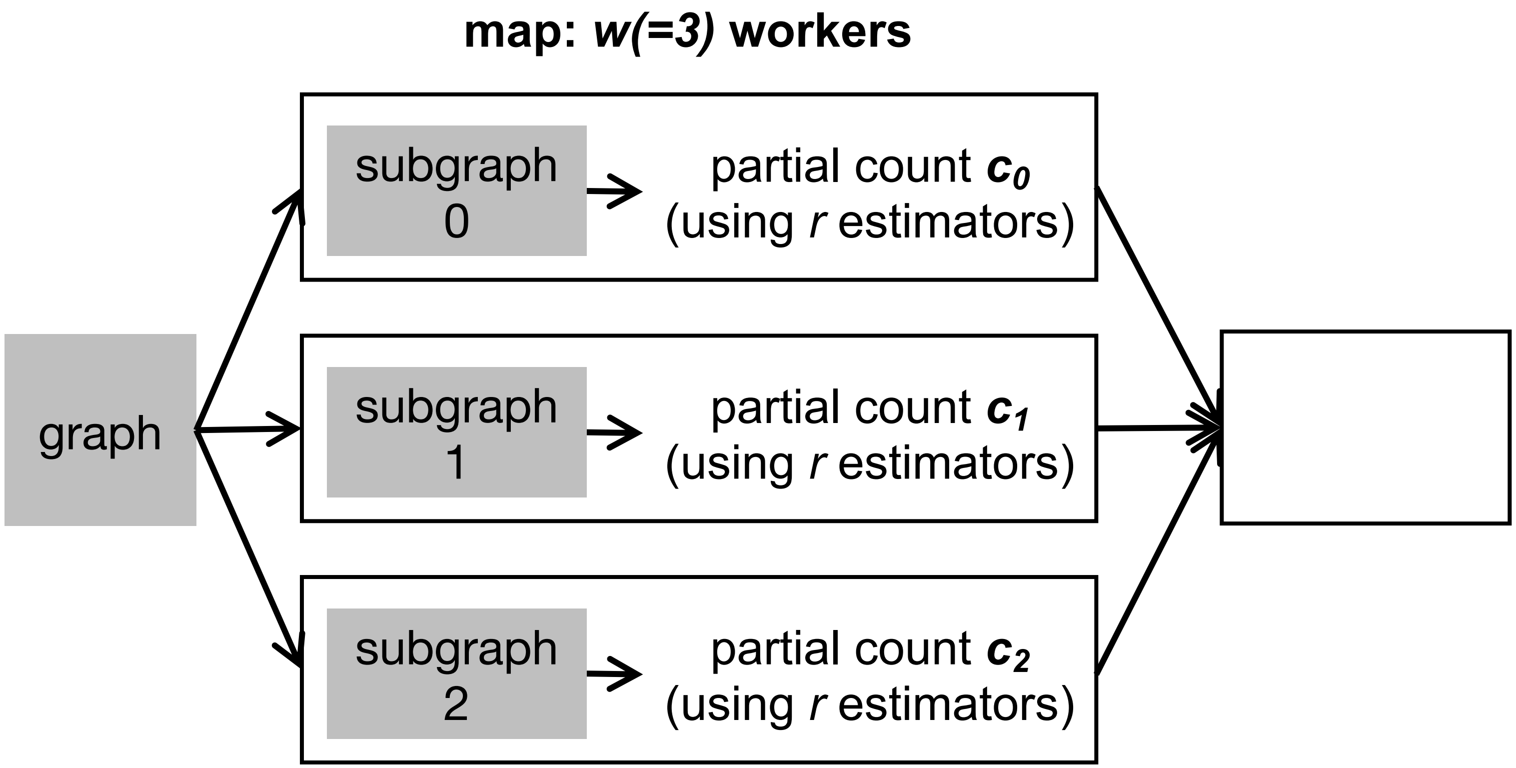
Applying to Distributed Settings



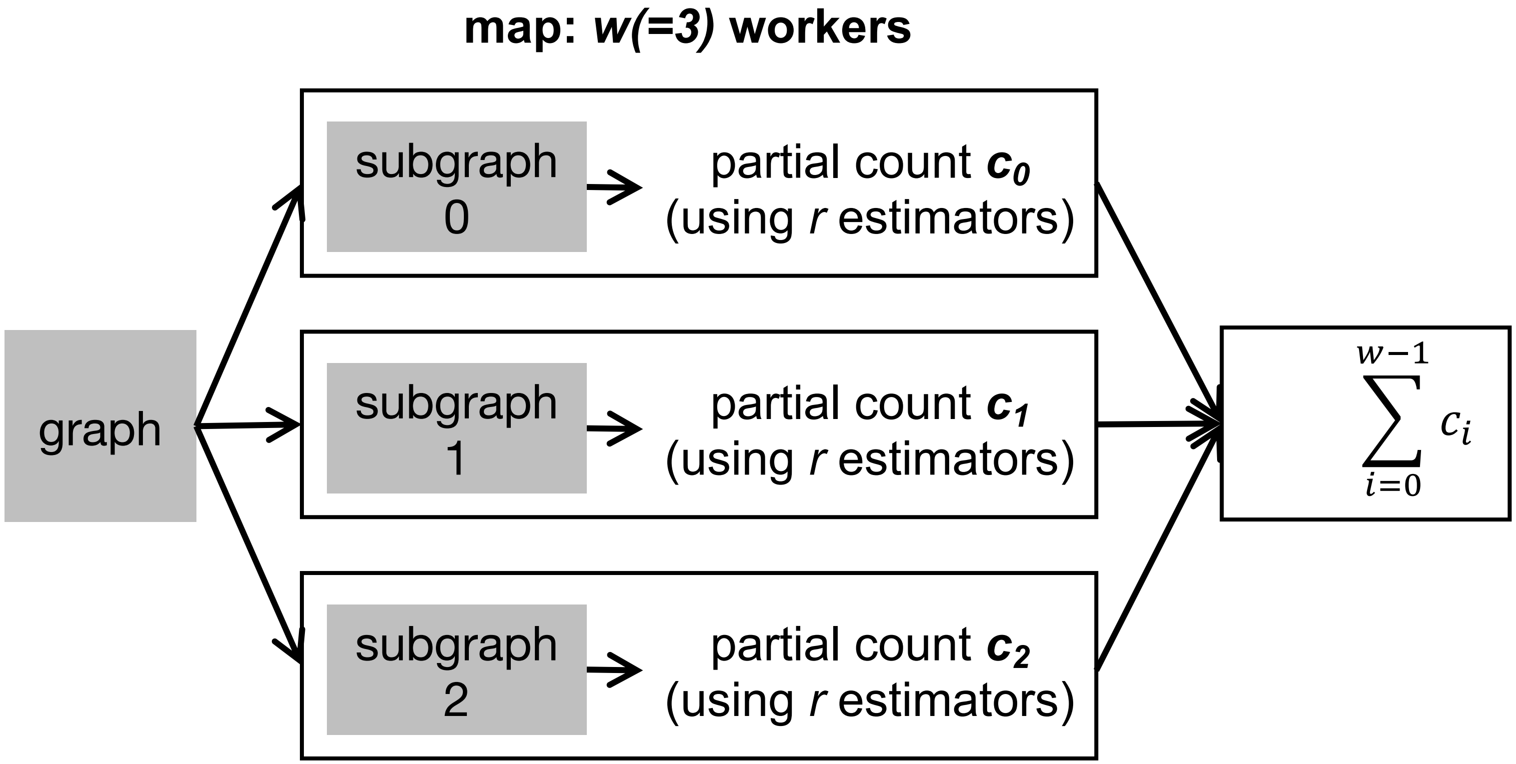
Applying to Distributed Settings



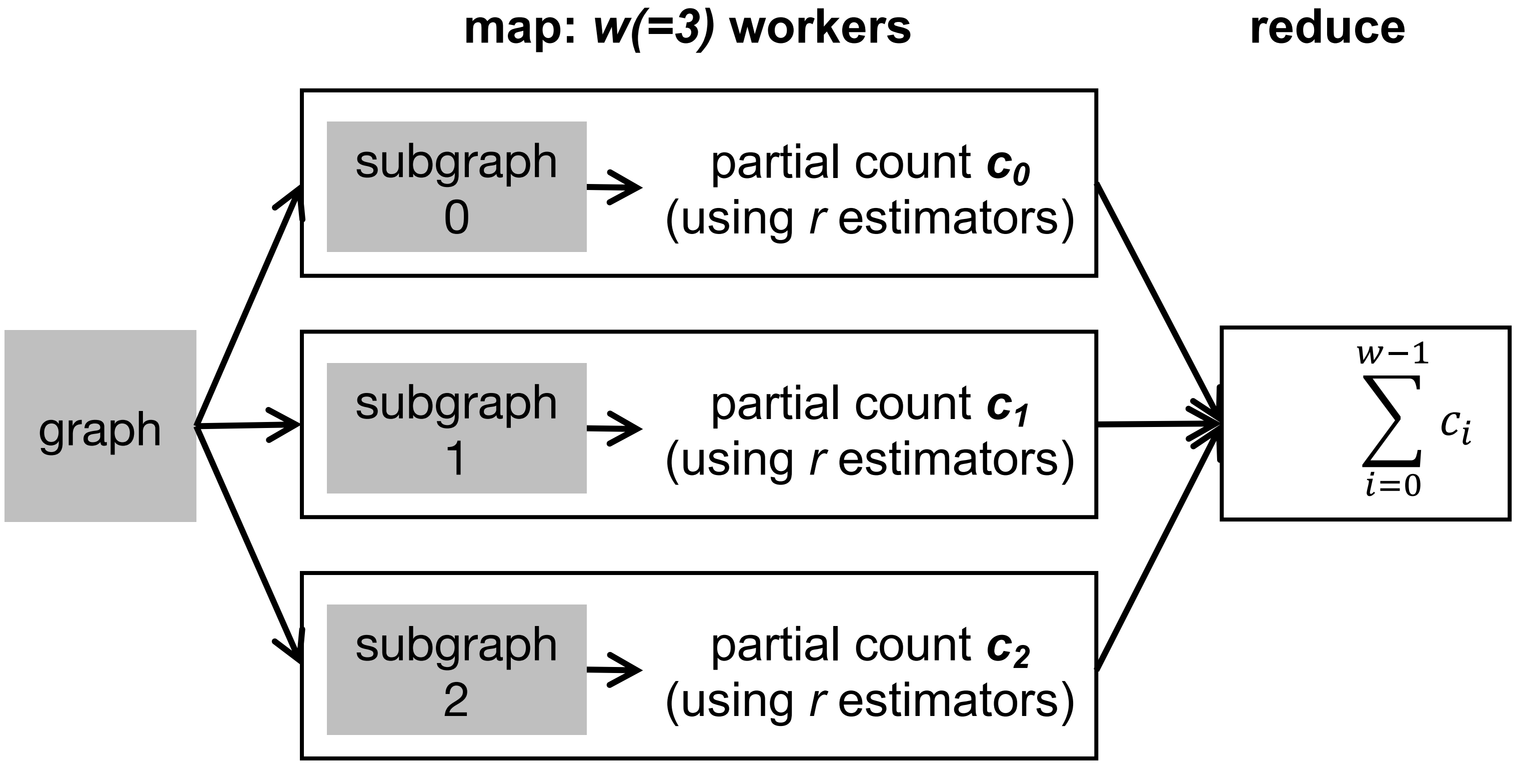
Applying to Distributed Settings



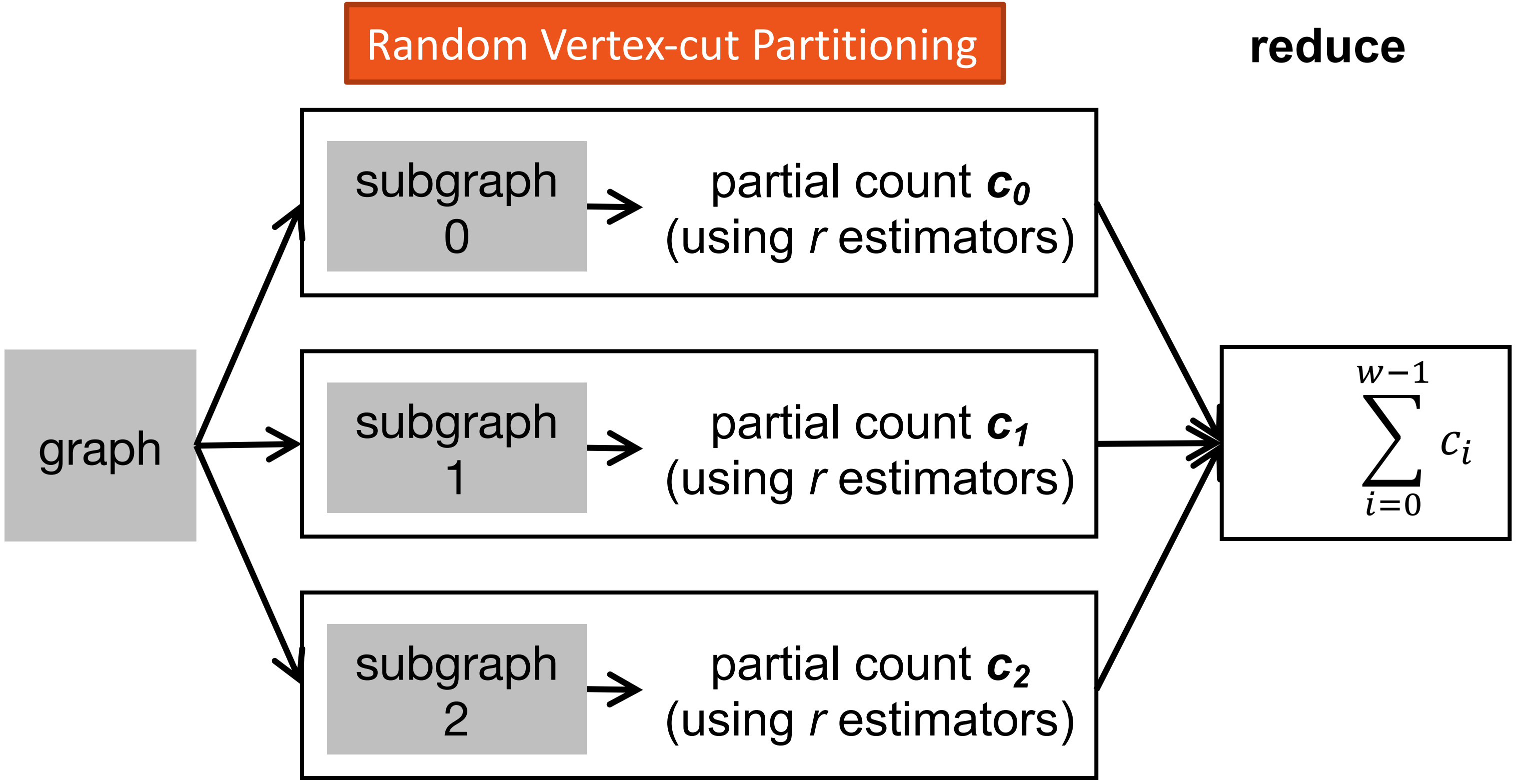
Applying to Distributed Settings



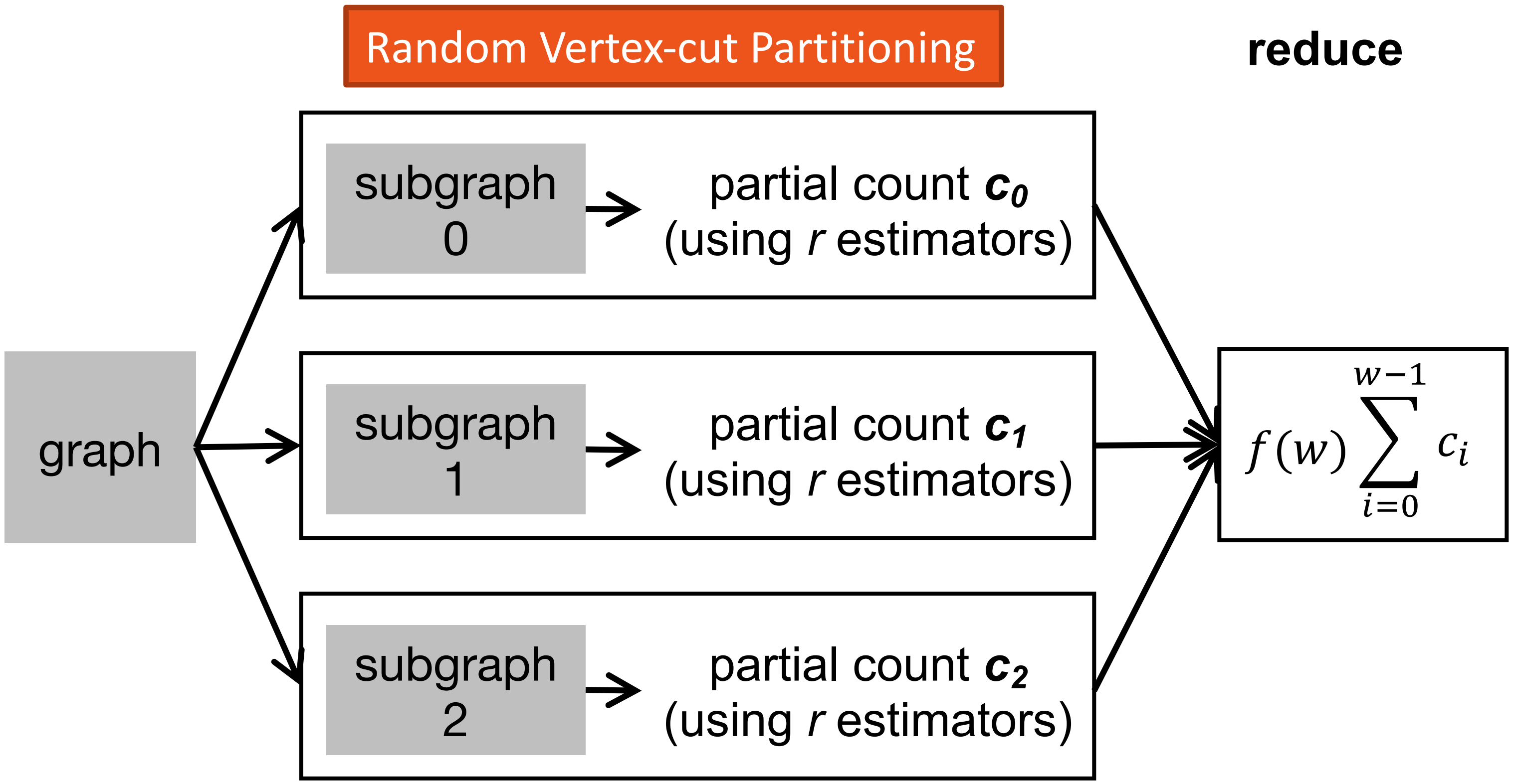
Applying to Distributed Settings



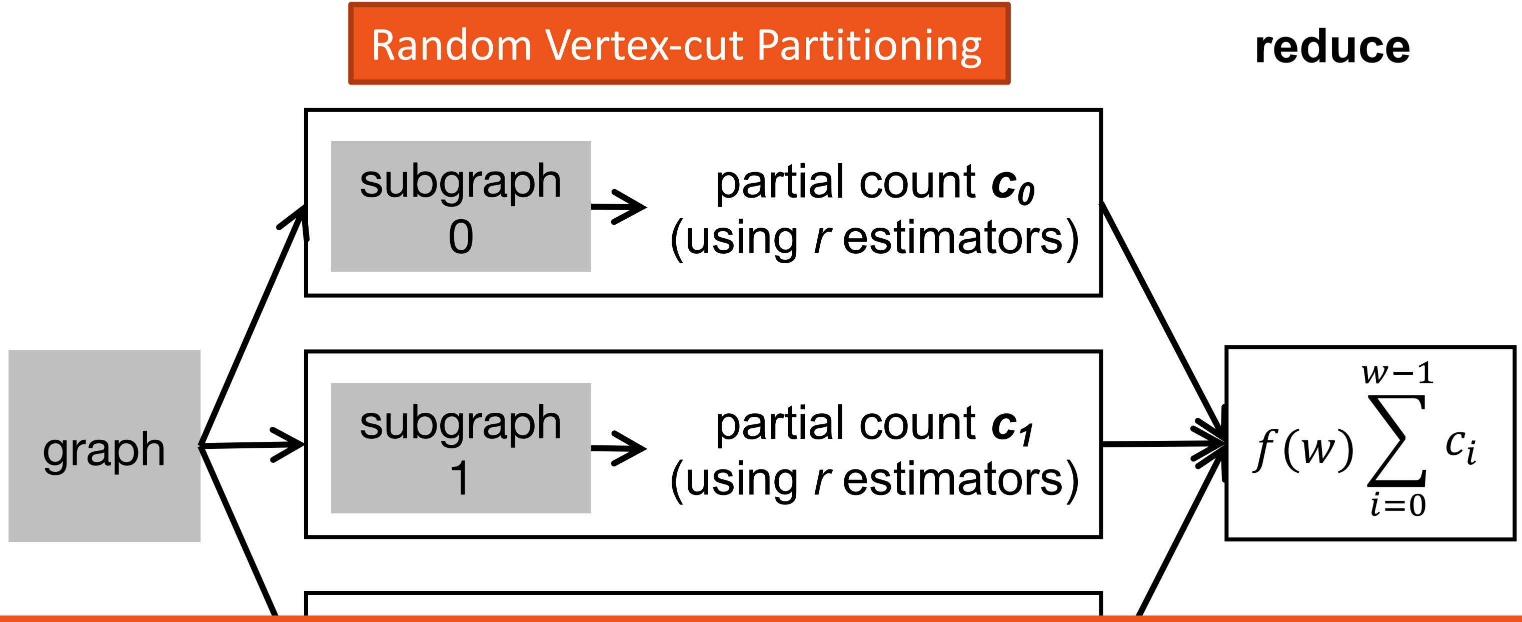
Applying to Distributed Settings



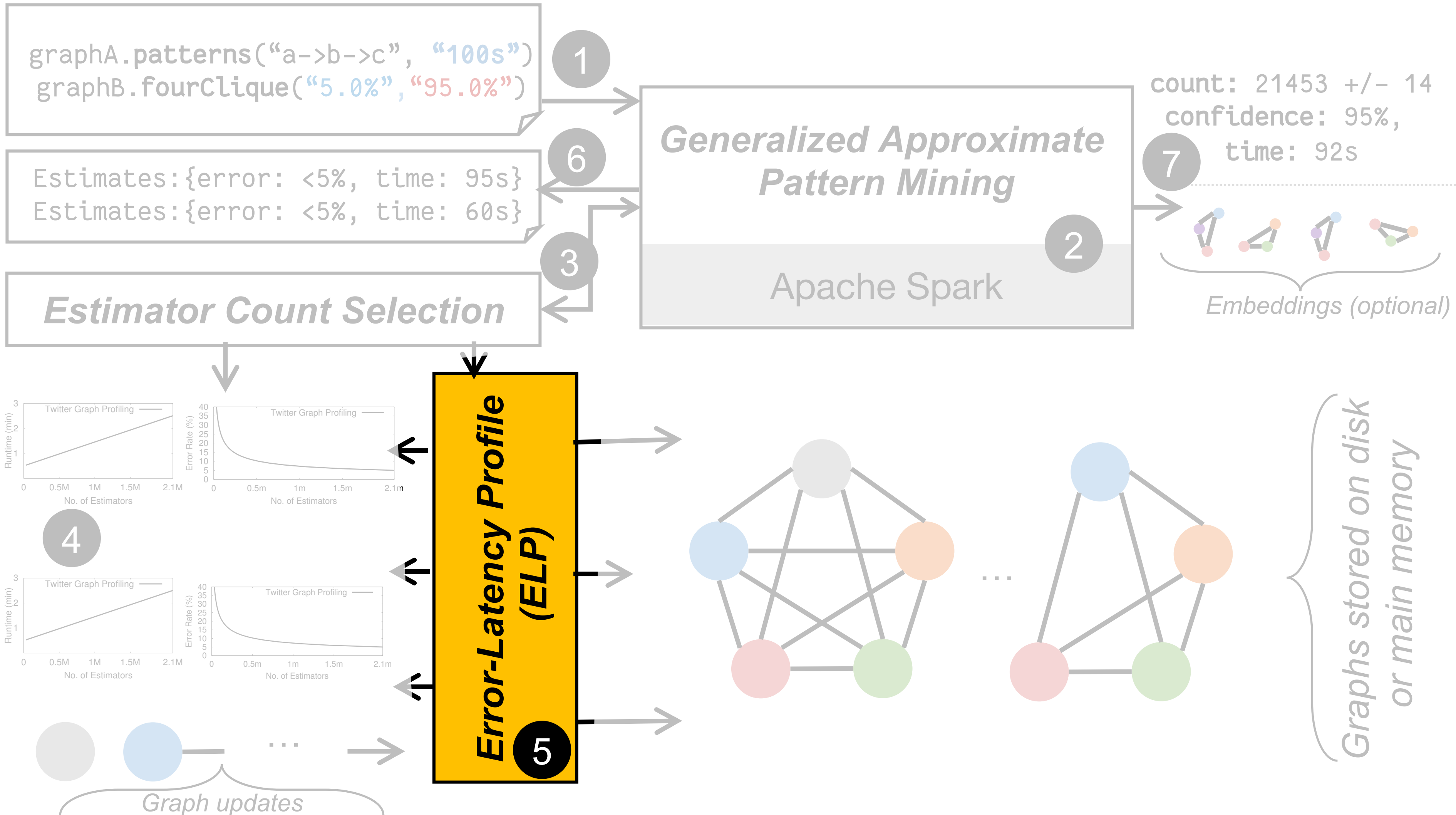
Applying to Distributed Settings



Applying to Distributed Settings



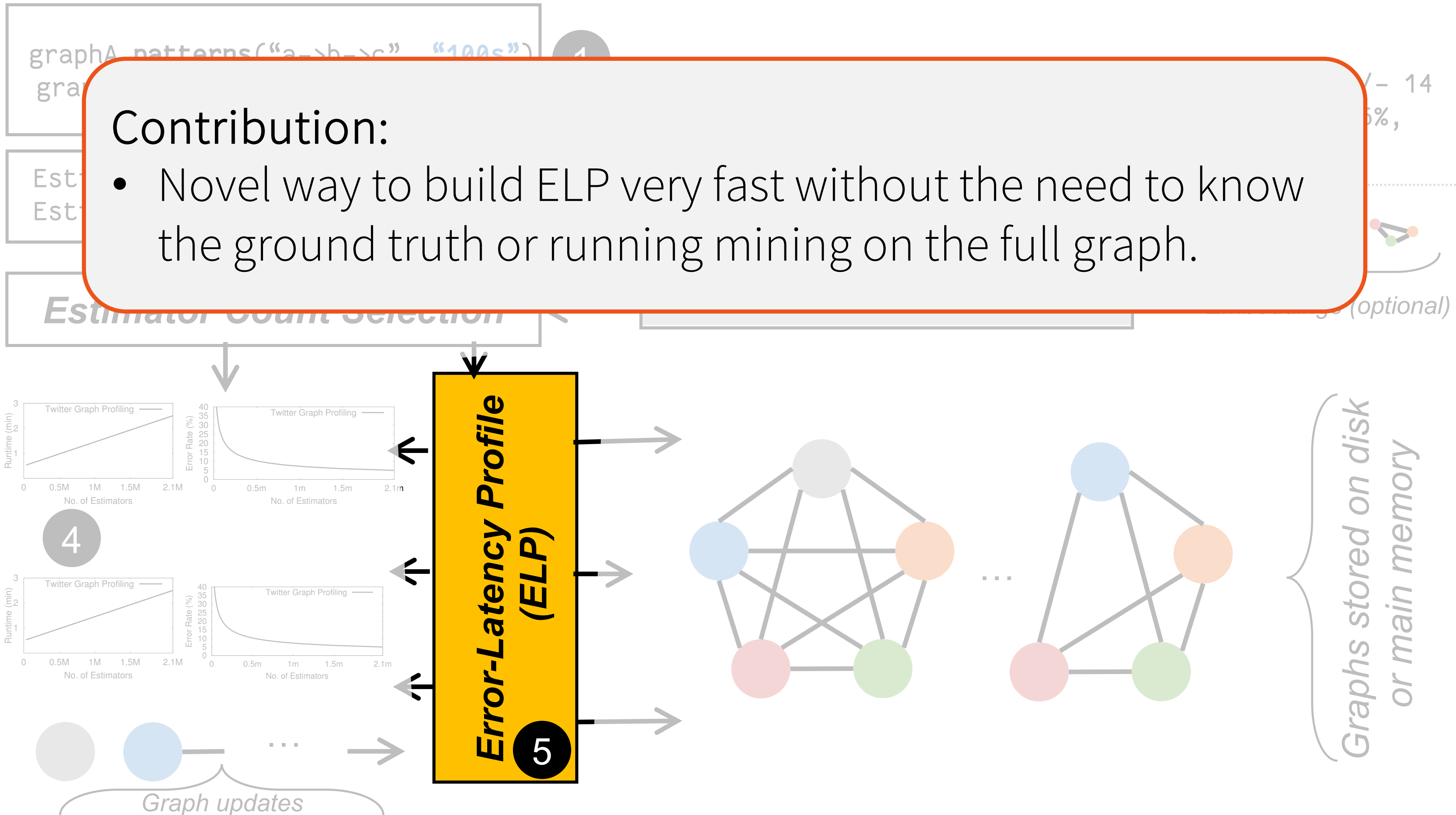
Upper bounds on f(w) can be proved using Hajnal-Szemerédi theorem



graphA.patterns("a->b->c", "100s")
graphB.fourClique("5.0%","95.0%")
1
Generalized Approximate Pattern Mining
Apache Spark
2
count: 21453 +/- 14
confidence: 95%,
time: 92s
7
Embeddings (optional)
6
Estimates:{error: <5%, time: 95s}
Estimates:{error: <5%, time: 60s}
3
Estimator Count Selection
Twitter Graph Profiling
Runtime (min)
No. of Estimators
Error Rate (%)
4
Error-Latency Profile (ELP)
5
Graphs stored on disk or main memory
Graph updates

Contribution:

- Novel way to build ELP very fast without the need to know the ground truth or running mining on the full graph.



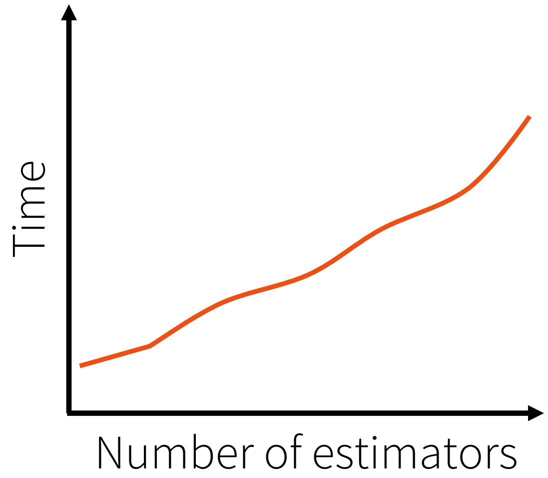
Building Error-Latency Profile

Given a time / error bound, how many estimators should ASAP use?

Building Error-Latency Profile

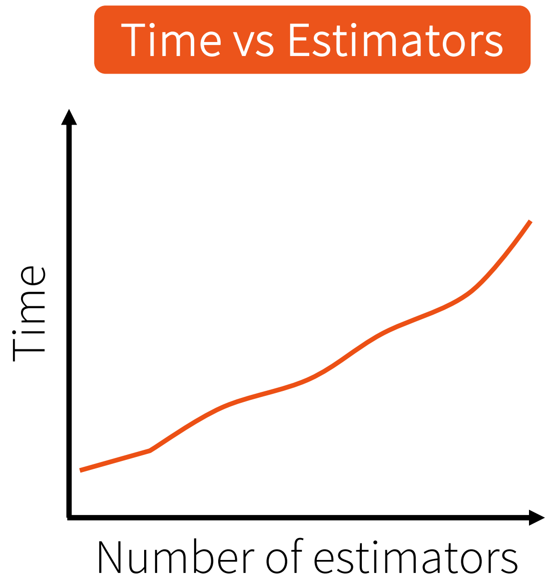
Given a time / error bound, how many estimators should ASAP use?

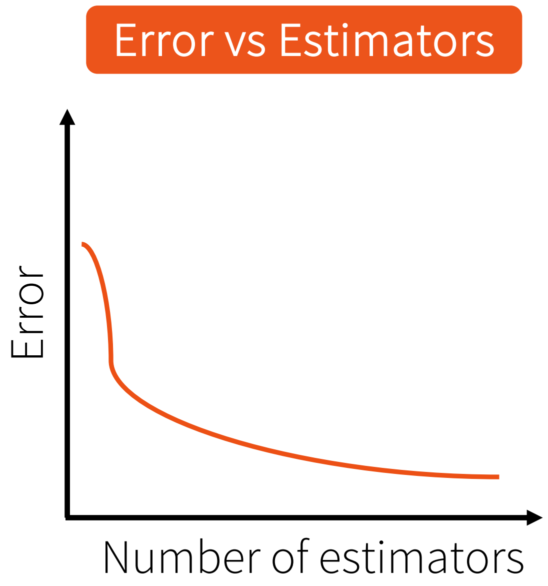
Time vs Estimators



Building Error-Latency Profile

Given a time / error bound, how many estimators should ASAP use?



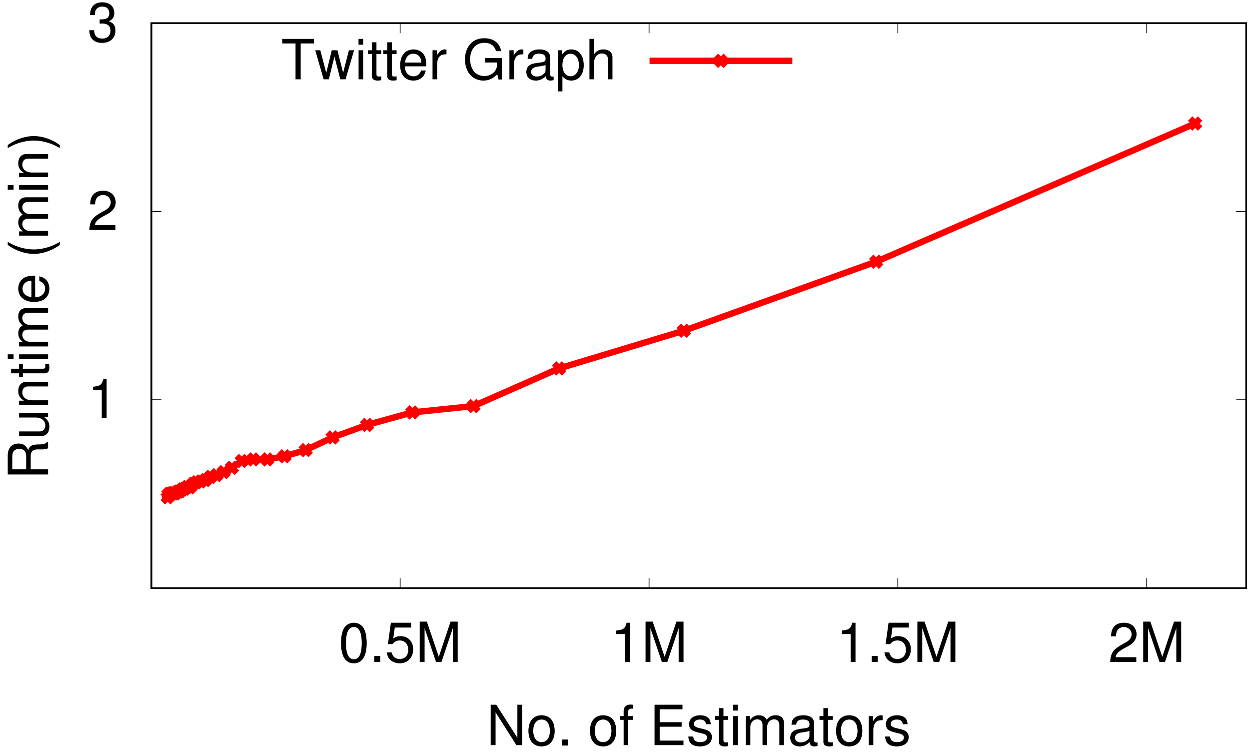


Building Estimators vs Time Profile

- Time complexity linear in number of estimators

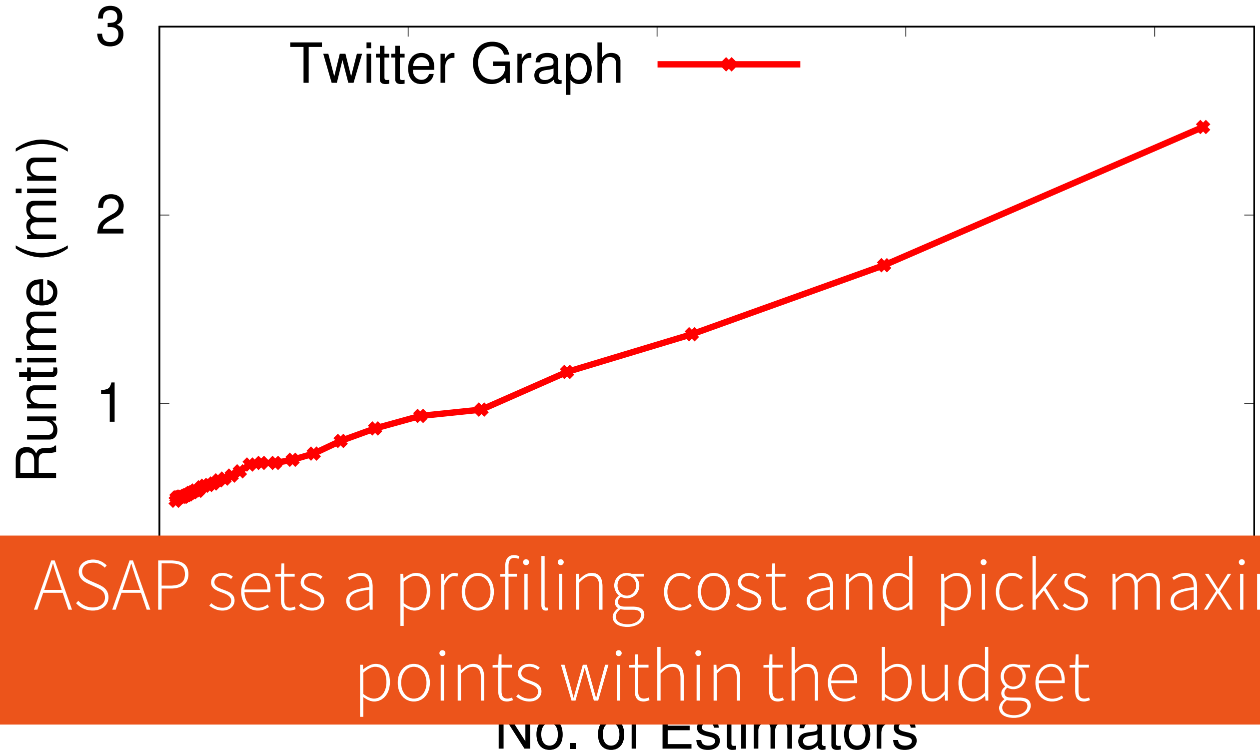
Building Estimators vs Time Profile

Time complexity linear in number of estimators



Building Estimators vs Time Profile

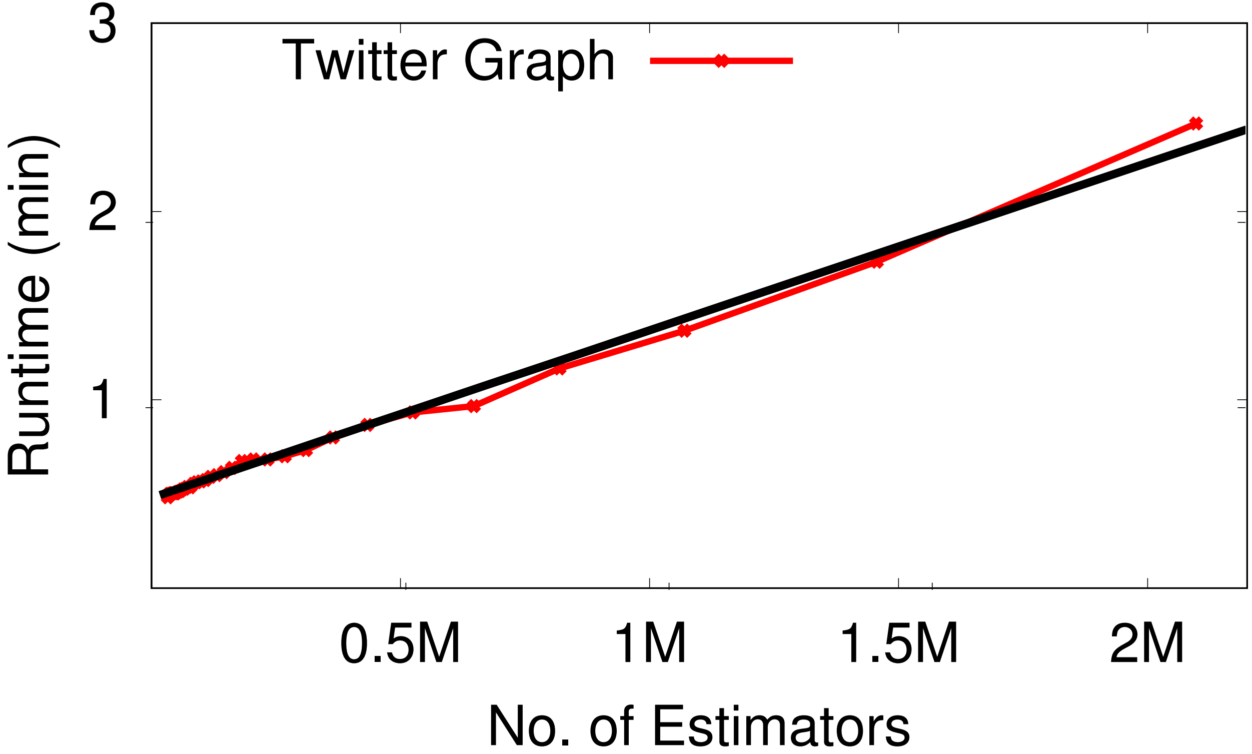
Time complexity linear in number of estimators



ASAP sets a profiling cost and picks maximum points within the budget

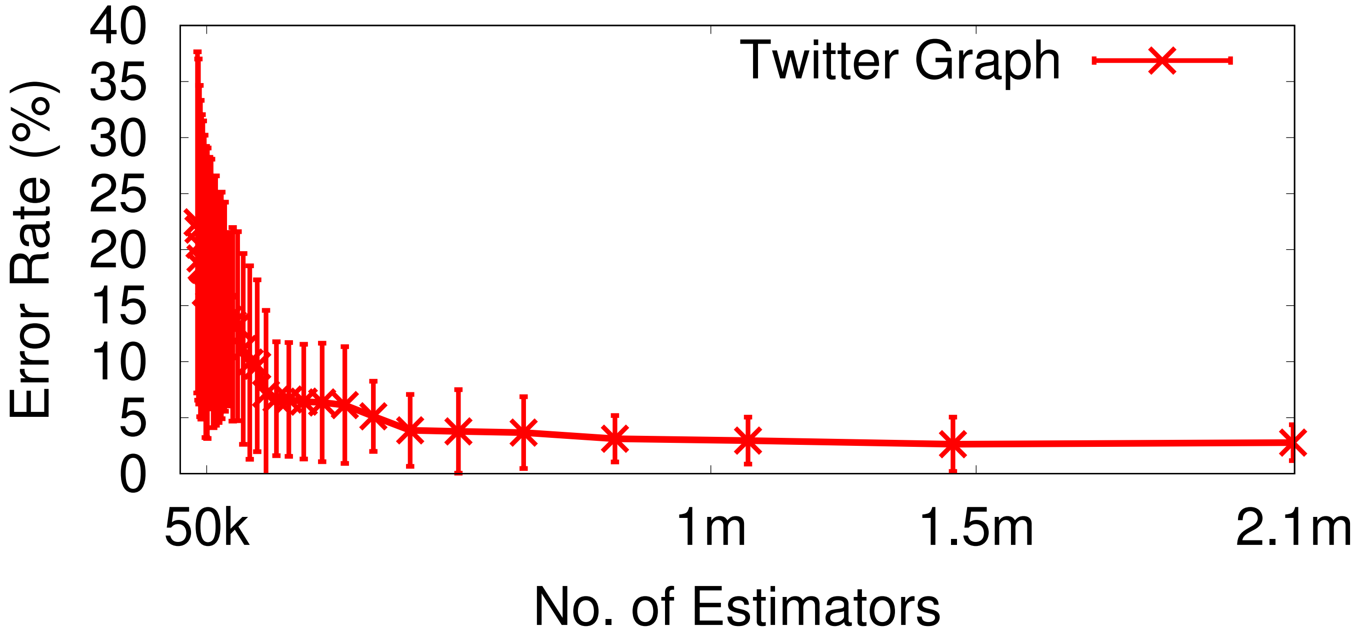
Building Estimators vs Time Profile

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

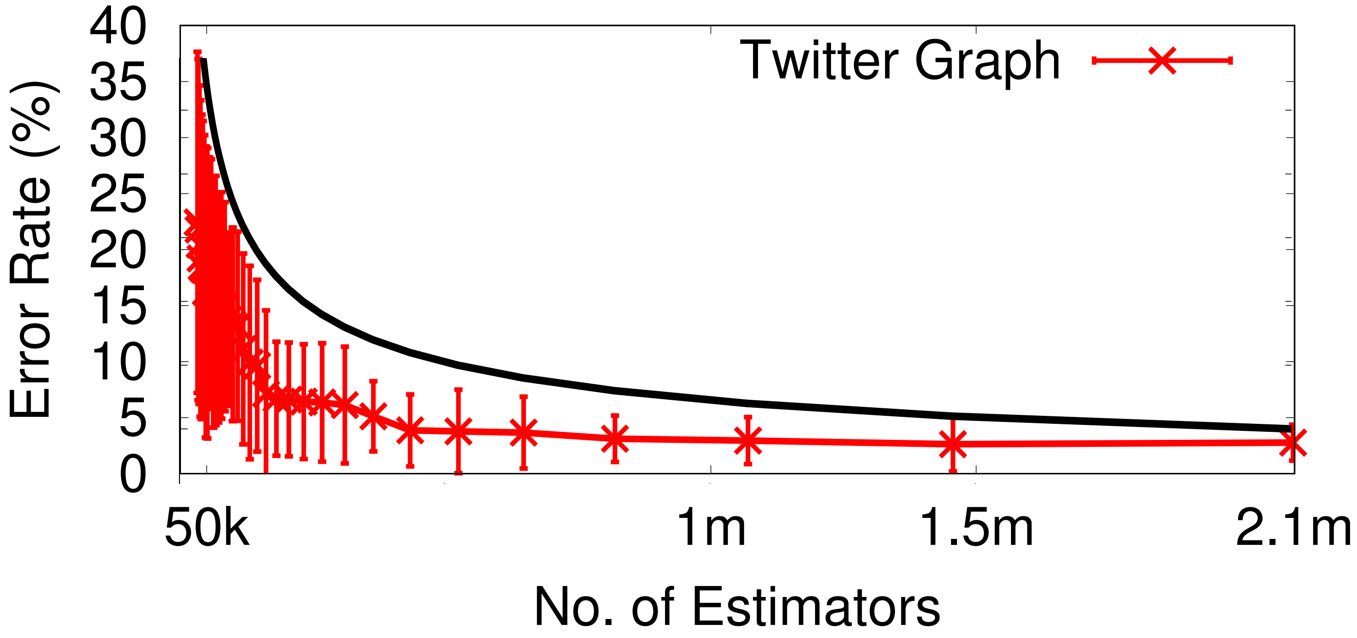
Error complexity non-linear in number of estimators

Key idea: Use a very small sample of the graph to build the ELP

- Chernoff analysis provides a *loose upper bound* on the number of estimators.
- In small graphs, a *large number of estimators* can get us very *close to ground truth*.

Building Estimators vs Error Profile

Error complexity non-linear in number of estimators



Advanced Mining

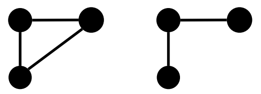
More details in the paper

Predicate Matching

- Find patterns where vertices are of type "electronics"
- ASAP allows simple edge and vertex predicates

Motif Mining

- Some patterns are building blocks for other patterns
- ASAP caches state of the estimators and reuses them



Accuracy Refinement

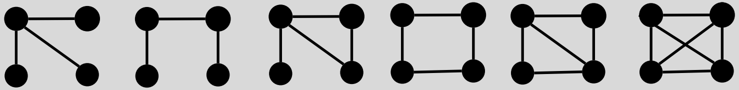
- Users may require more accurate answer later
- ASAP can checkpoint and reuse estimators

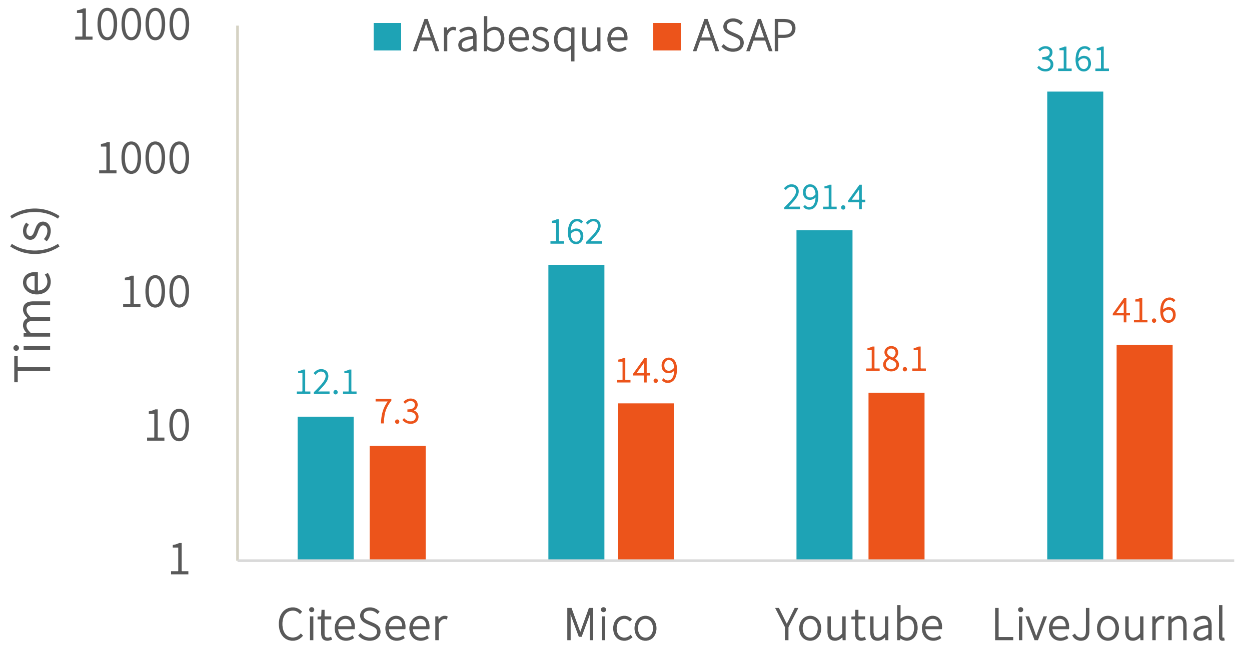
Implementation & Evaluation

- Implemented on Apache Spark
  - Not limited to it, only relies on simple dataflow operators
- Evaluated in a 16 node cluster
  - **Twitter:** 1.47B edges
  - **Friendster:** 1.8B edges
  - **UK:** 3.73B edges
- Comparison using representative patterns:
  - 3 (2 patterns), 4 (6 patterns) and 5 motifs (21 patterns)

Performance on Small Graphs

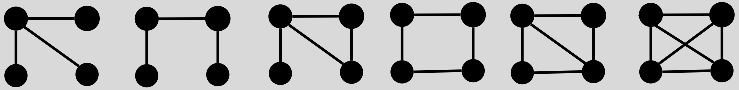
4-Motifs (6 patterns)

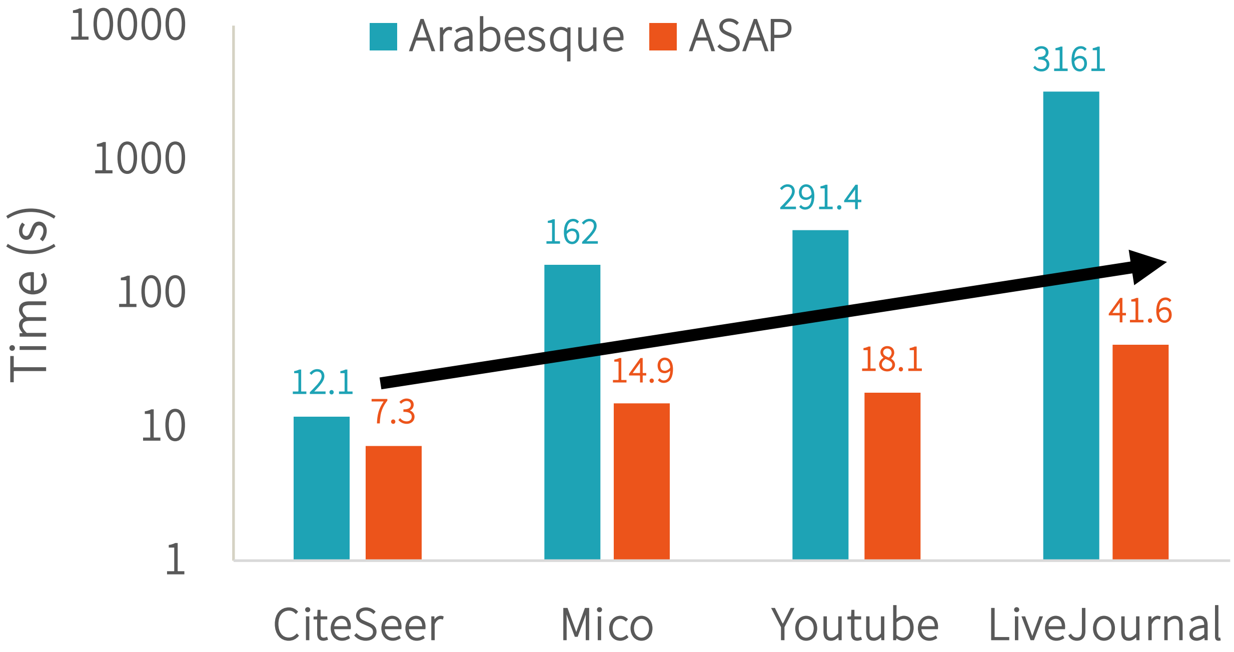




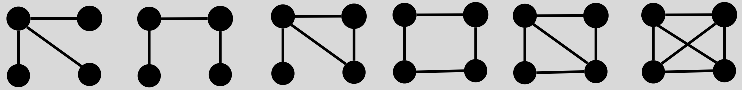
Performance on Small Graphs

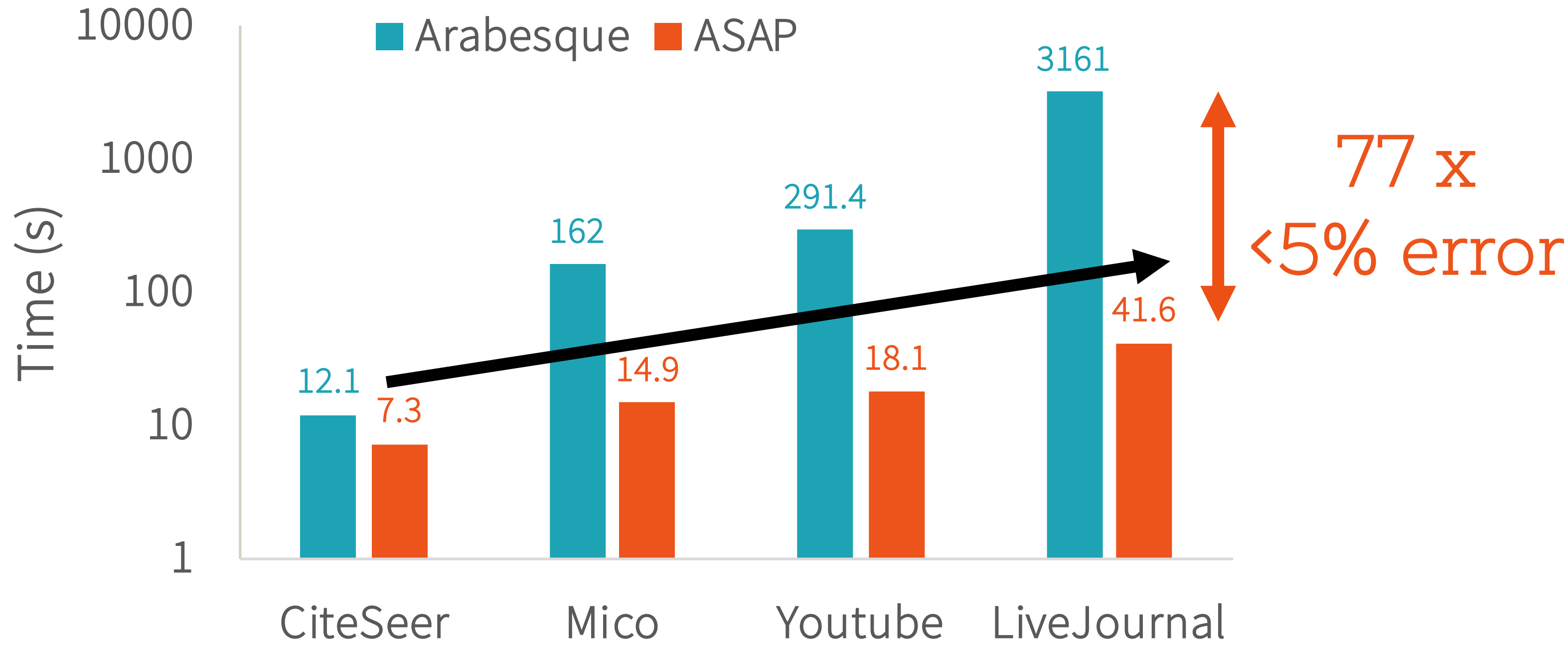
4-Motifs (6 patterns)



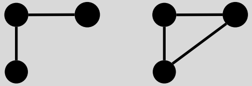


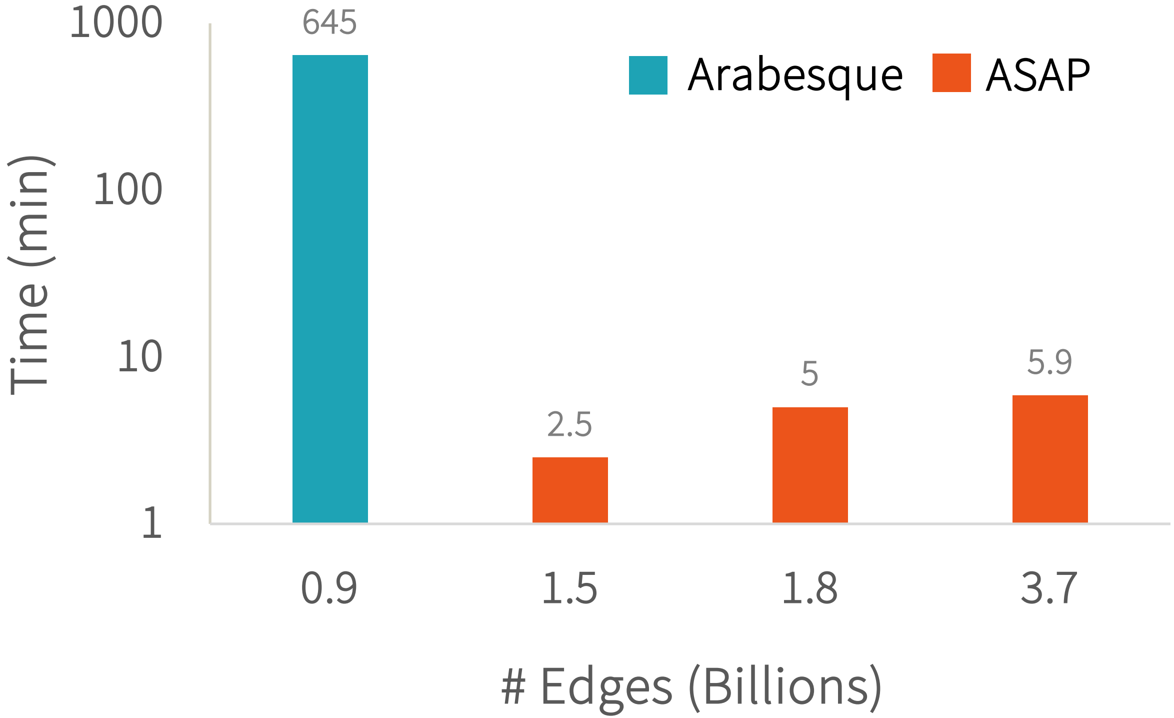
Performance on Small Graphs

4-Motifs (6 patterns) 



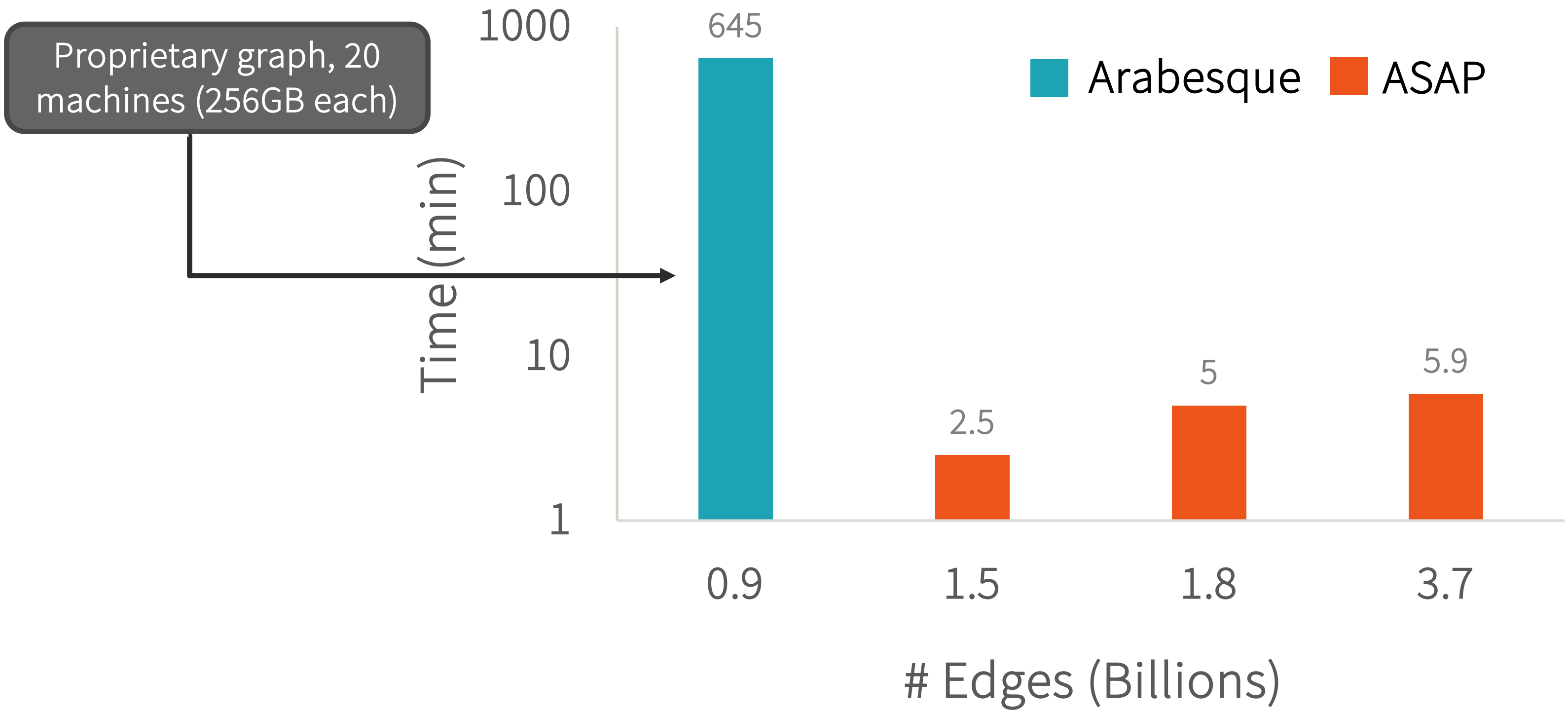
Large Graphs & Simple Patterns

3-Motifs (2 patterns) 



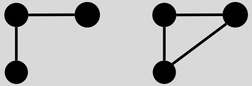
Large Graphs & Simple Patterns

3-Motifs (2 patterns)

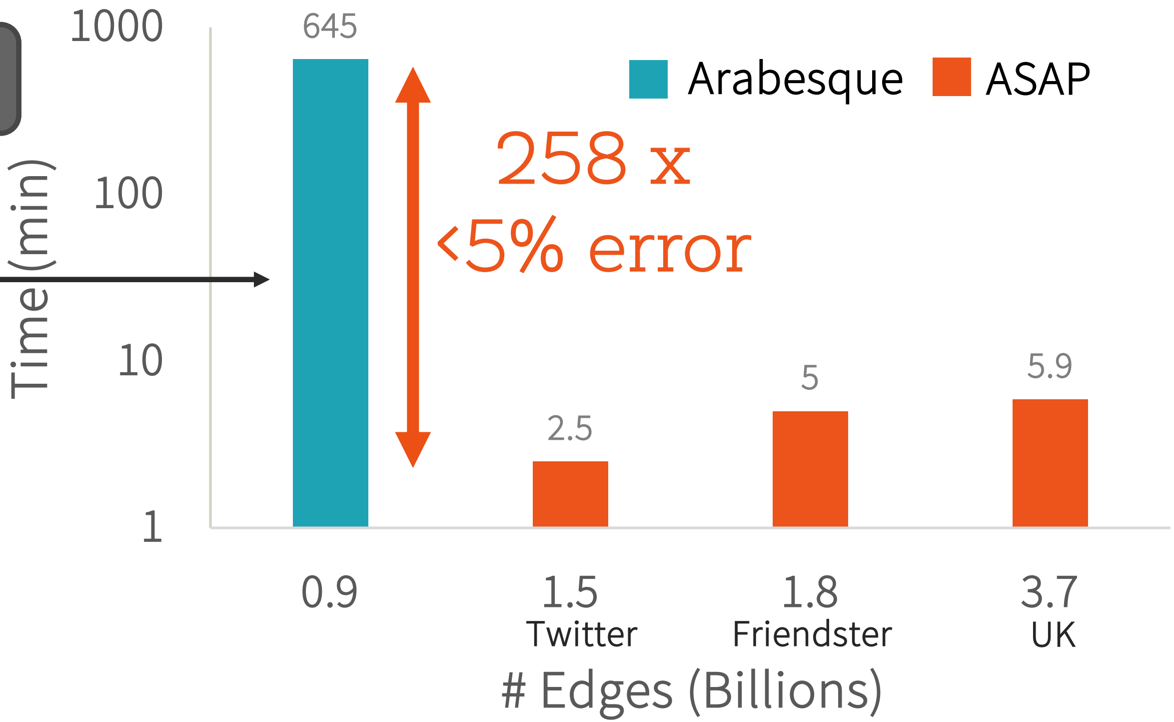


Large Graphs & Simple Patterns

3-Motifs (2 patterns)

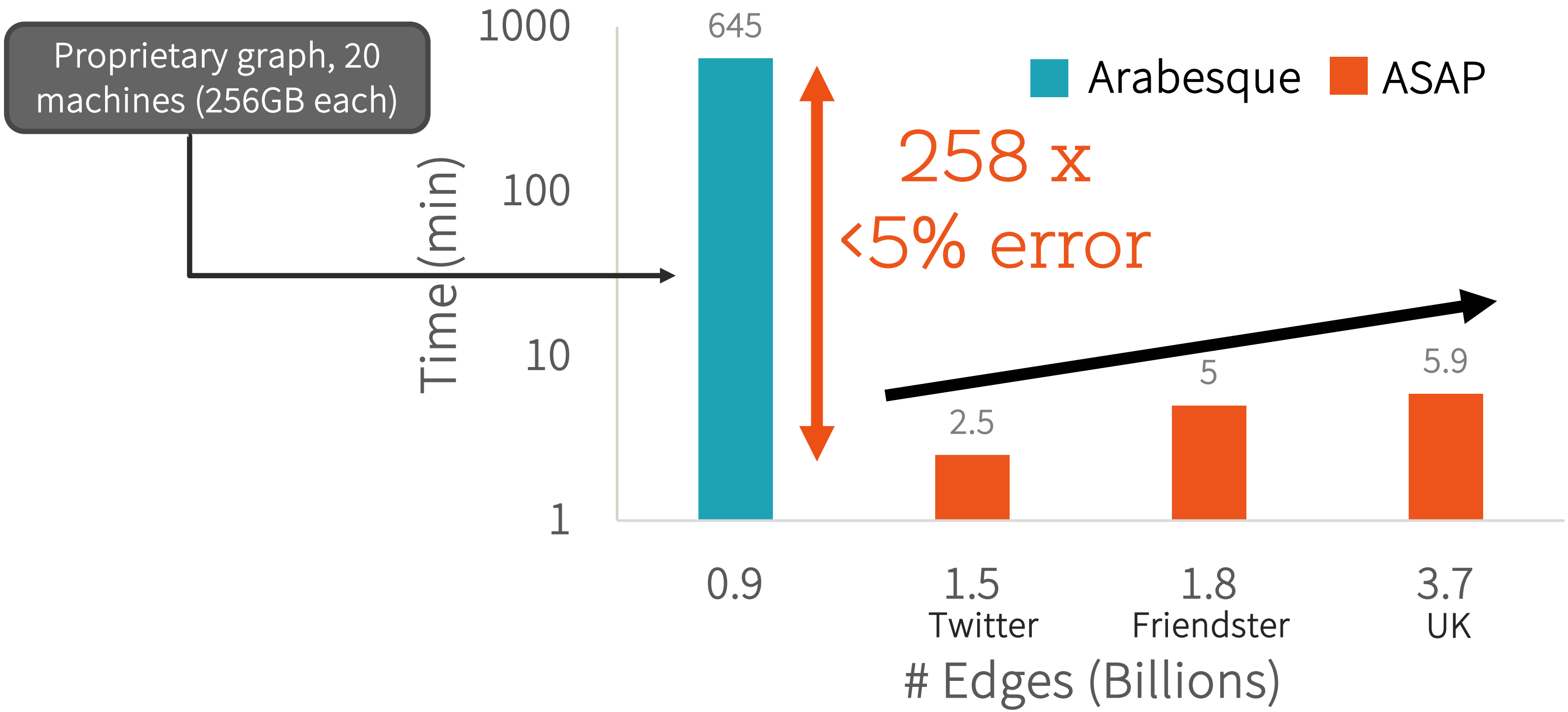


Proprietary graph, 20 machines (256GB each)



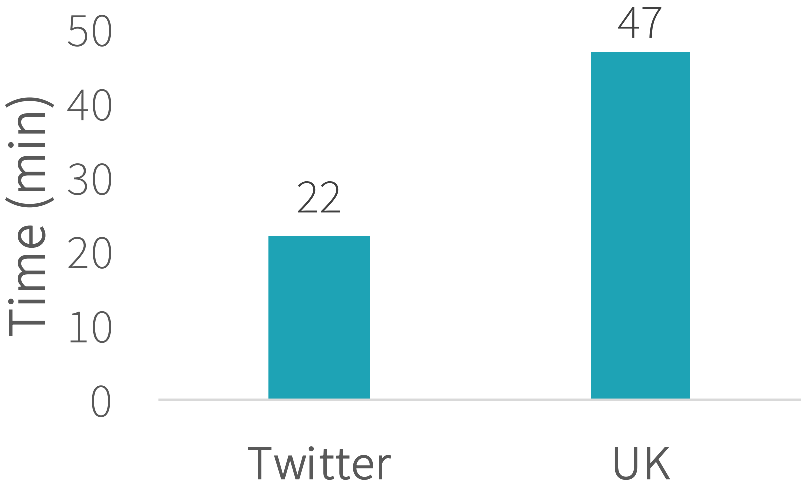
Large Graphs & Simple Patterns

3-Motifs (2 patterns)

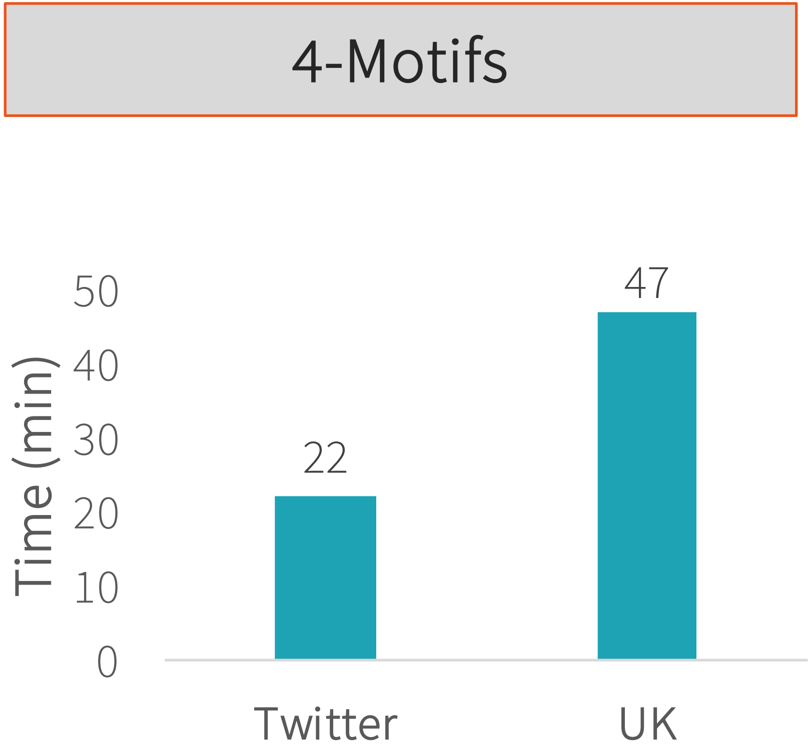


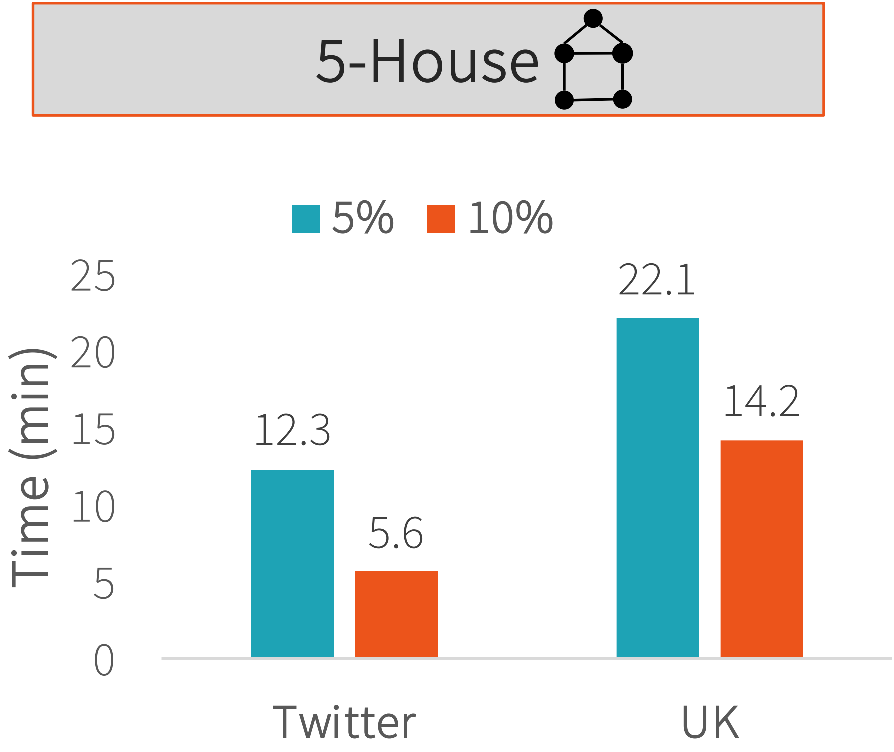
Large Graphs & Complex Patterns

4-Motifs



Large Graphs & Complex Patterns





Summary

- Pattern mining important & challenging problem
  - Applications in many domains
- ASAP uses approximation for fast pattern mining
  - Leverages graph mining theory & makes it practical
  - Simple API for developers
- ASAP outperforms existing solutions
  - Can handle much larger graphs with fewer resources

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