ASAP: Fast, Approximate Graph Pattern Mining at Scale

Anand Iyer et al. @ OSDI 2018

Presenter: Yunang Chen

ASAP Design Overview

A *S*wift *A*pproximate *P*attern-miner

Navigates tradeoff between result accuracy and latency

Runs on general-purpose distributed dataflow platform

Supports for generalized graph pattern mining algorithms

Graph Pattern Mining

Standard approach: Iterative expansion

Lack of scalability

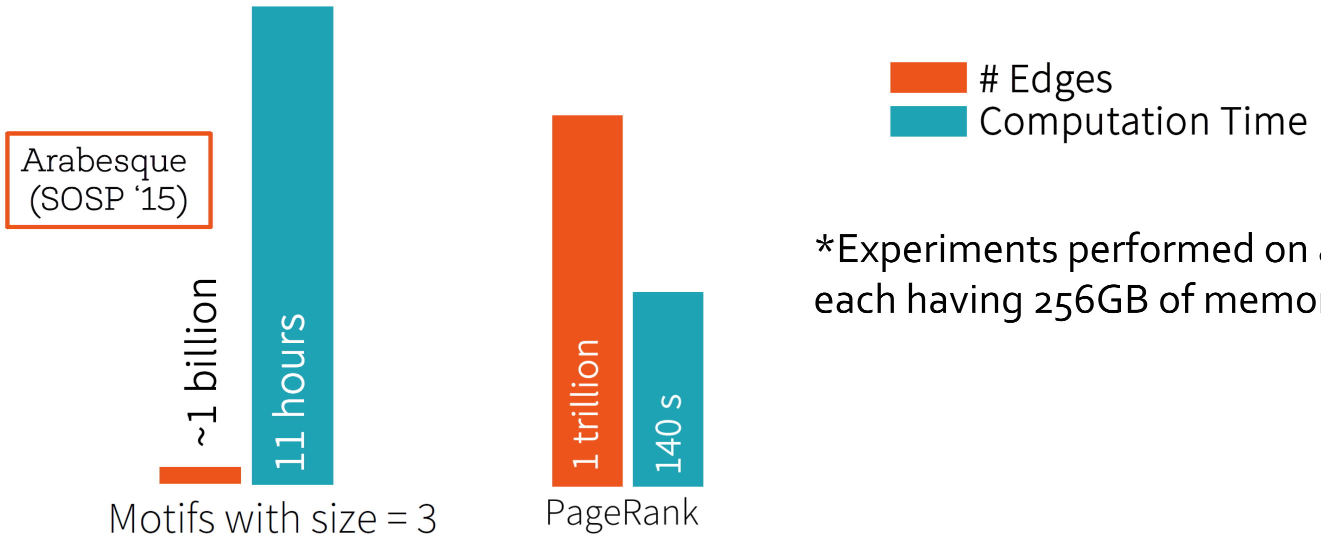
- Generate exponentially large intermediate candidate sets
- Need to store + exchange them in distributed environment

Graph Pattern Mining

Standard approach: Iterative expansion

Lack of scalability

- Generate exponentially large intermediate candidate sets
- Need to store + exchange them in distributed environment



*Experiments performed on a cluster of 20 machines, each having 256GB of memory.

Graph Pattern Mining

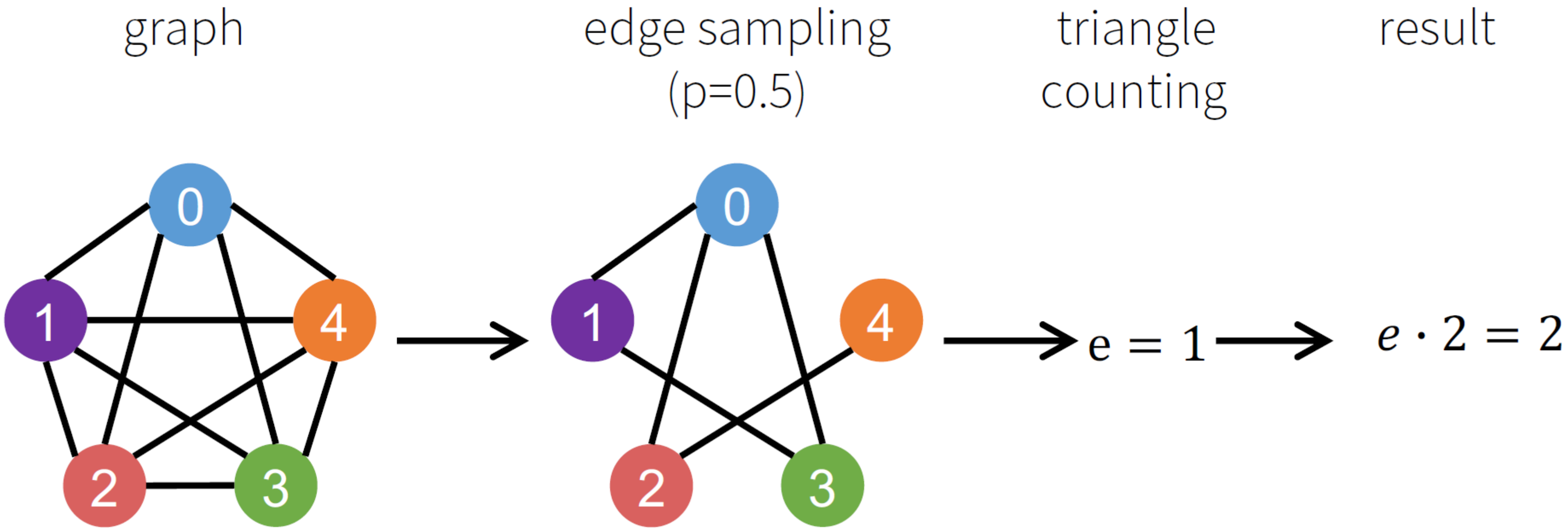
Many pattern mining tasks do not need exact answers.

- Frequent sub-graph mining (FSM) finds the frequency of subgraphs but with an end-goal of ordering them by occurrences.

Leverage approximation for pattern mining

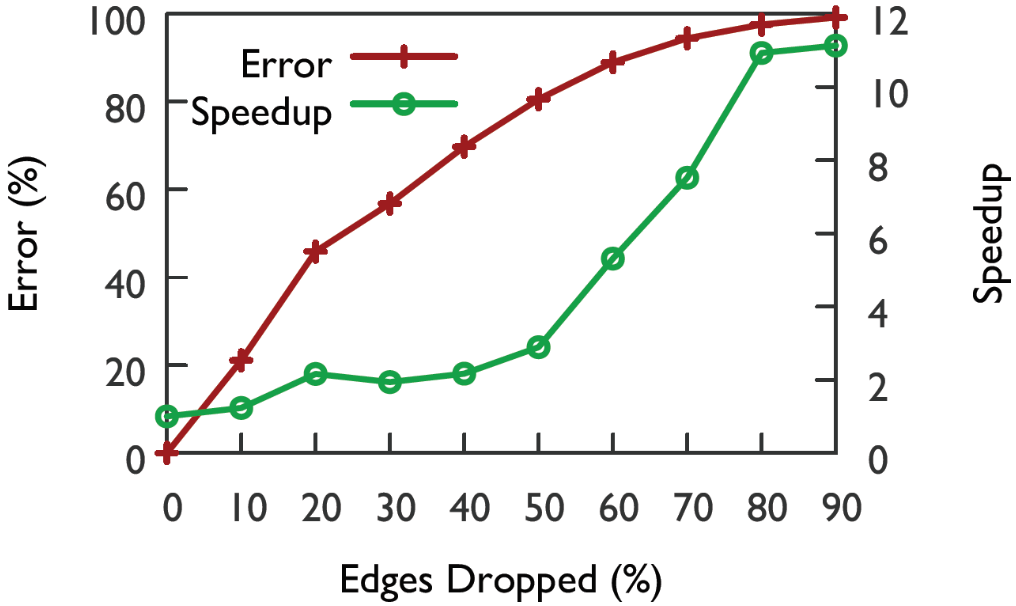
Approximate Pattern Mining

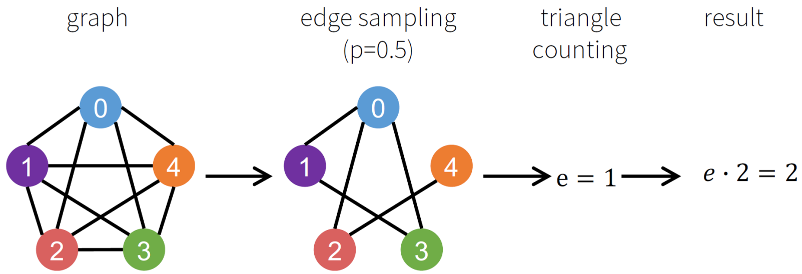
Previous approach: Apply the exact same algorithm on subsets of the input data, then use the statistical properties of these subsets to estimate final results.



Approximate Pattern Mining

Previous approach: Apply the exact same algorithm on subsets of the input data, then use the statistical properties of these subsets to estimate final results.





- No significant speedup
- Large error rate

Approximate Pattern Mining

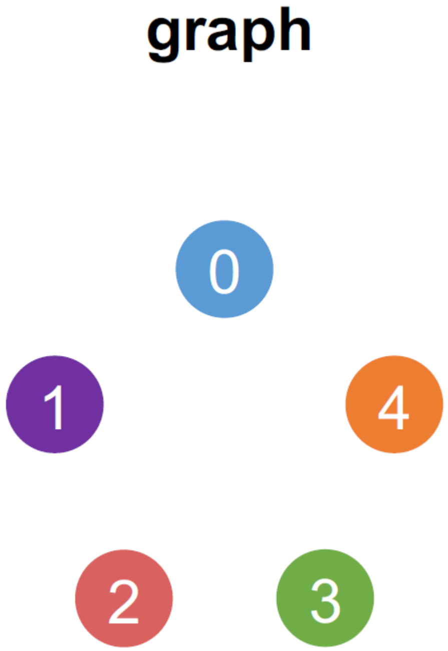
Neighborhood sampling:

1. Model the edges in the graph as a stream
2. Sample one edge, e_1
3. Gradually add more adjacent edges, $e_2, \ldots, e_k$
4. Stop when the edges form the pattern or becomes impossible to do so
5. Use the probability of sampling to bound the total number of occurrences of the pattern:
   $P(e_1, \ldots, e_k) = P(e_1) \times P(e_2 \mid e_1) \times \ldots \times P(e_k \mid e_1, \ldots, e_{k-1})$
6. Repeat Step 1-5 multiple times

Approximate Pattern Mining

Neighborhood sampling: Triangle Counting

1. Model the edges in the graph as a stream

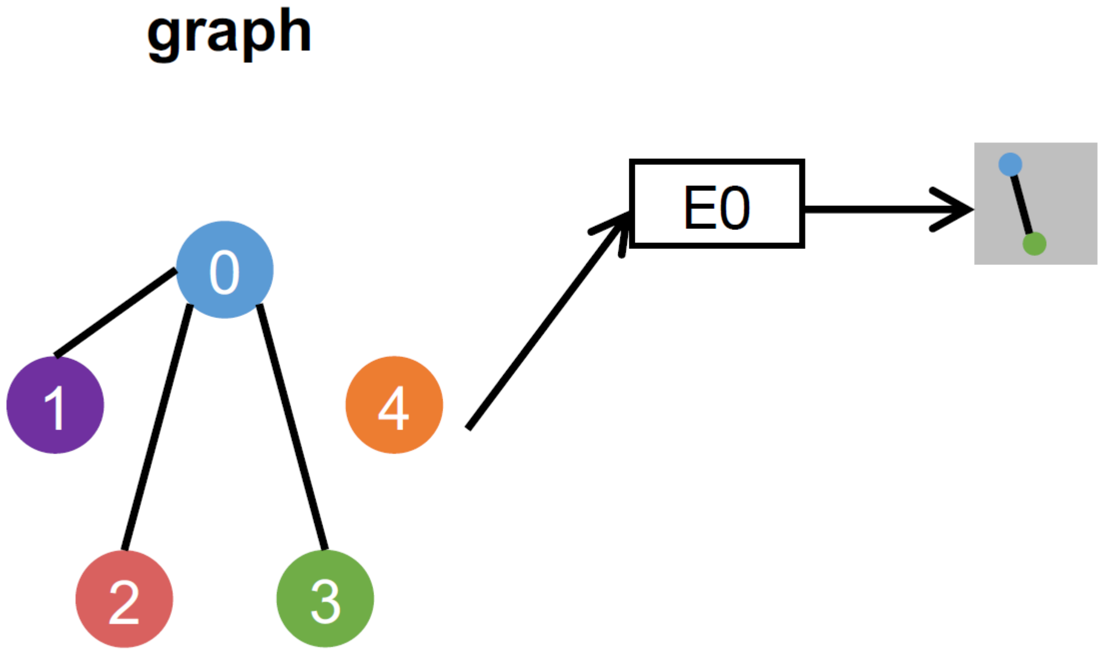


edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Approximate Pattern Mining

Neighborhood sampling: Triangle Counting

2. Sample one edge

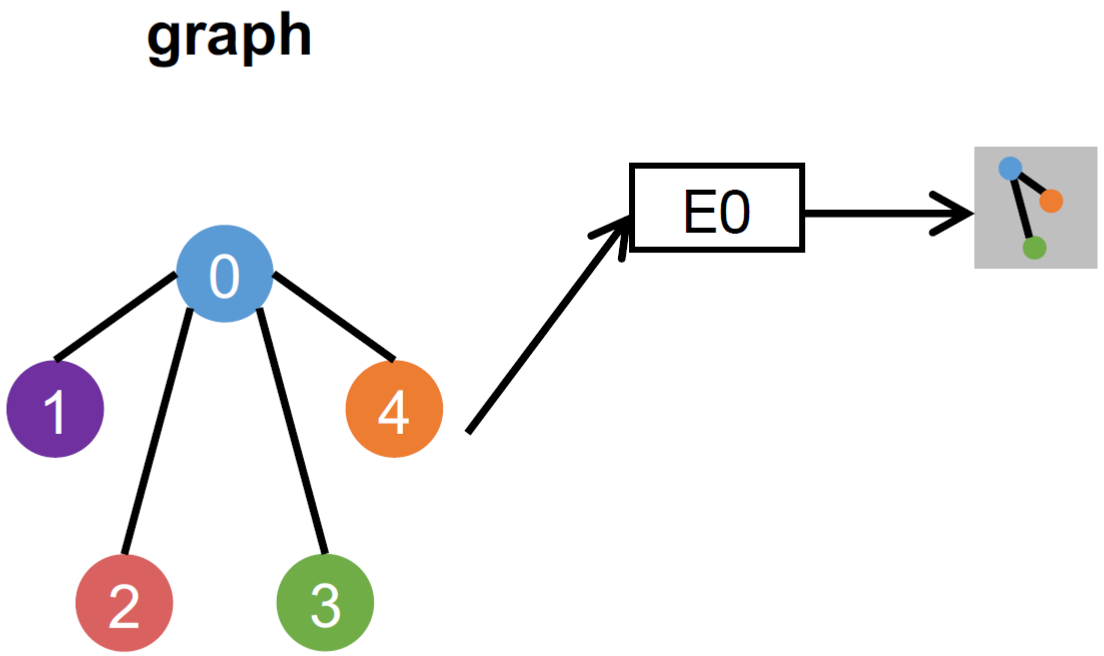


edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Approximate Pattern Mining

Neighborhood sampling: Triangle Counting

3. Gradually add more adjacent edges

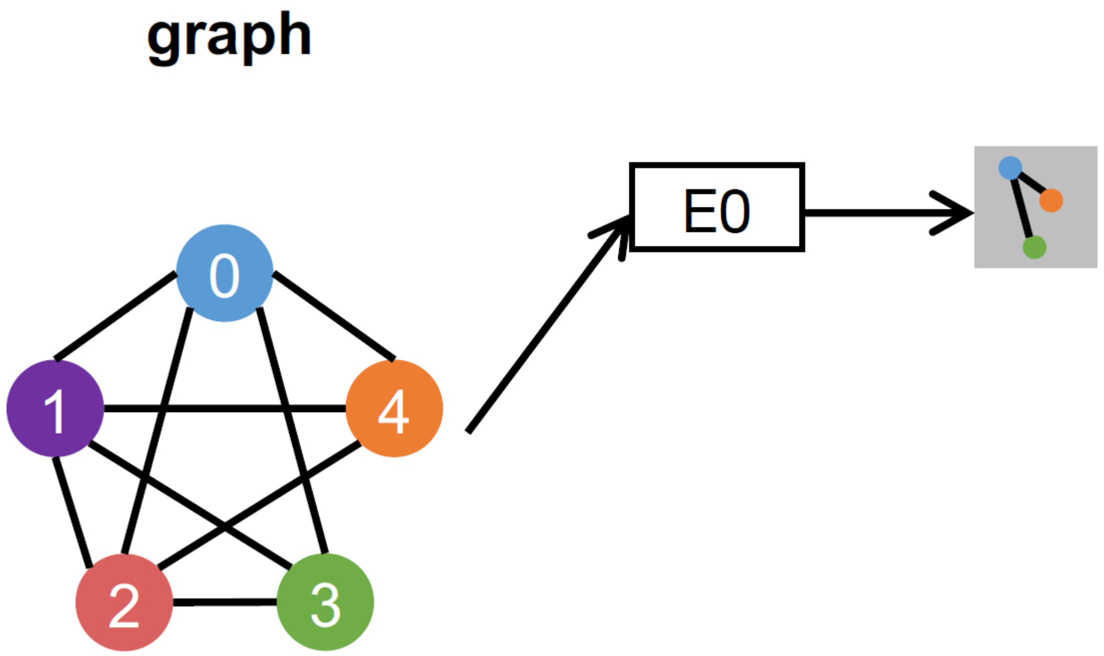


edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Approximate Pattern Mining

Neighborhood sampling: Triangle Counting

4. Stop when the edges form the pattern or becomes impossible to do so

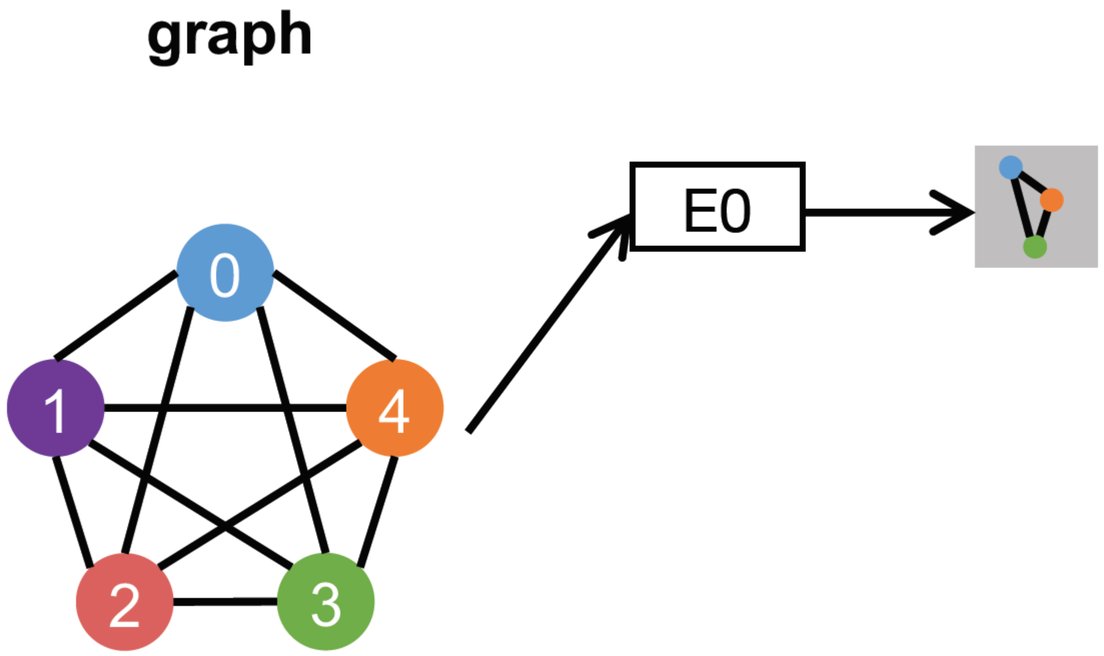


edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Approximate Pattern Mining

Neighborhood sampling: Triangle Counting

4. Stop when the edges form the pattern or becomes impossible to do so

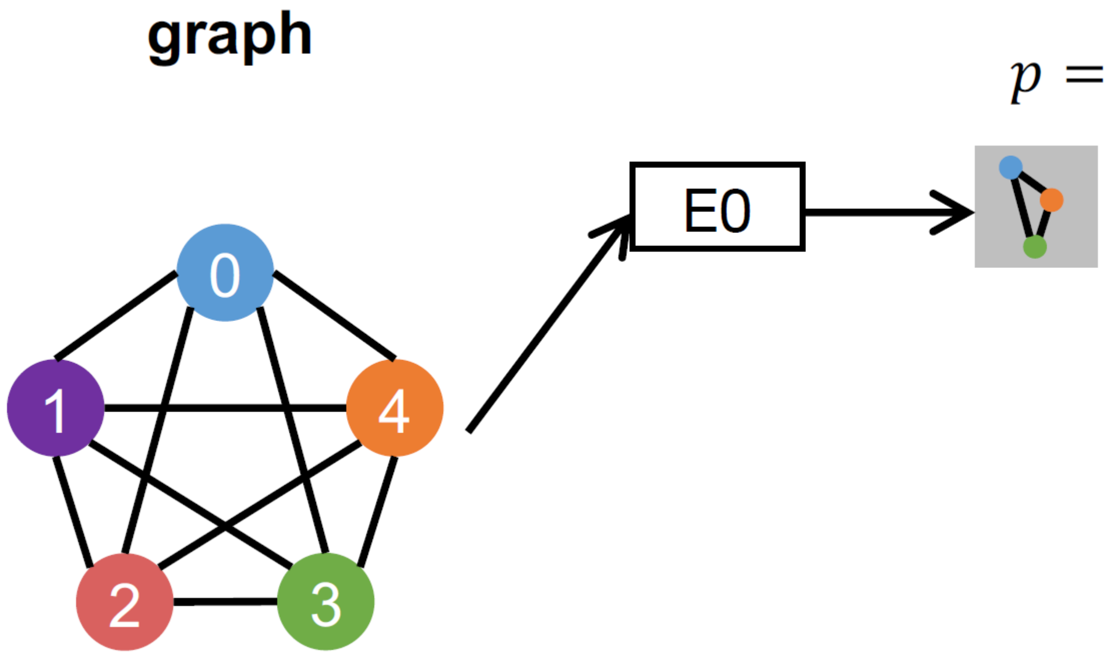


edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Approximate Pattern Mining

Neighborhood sampling: Triangle Counting

5. Use the probability of sampling to bound the total number of occurrences



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

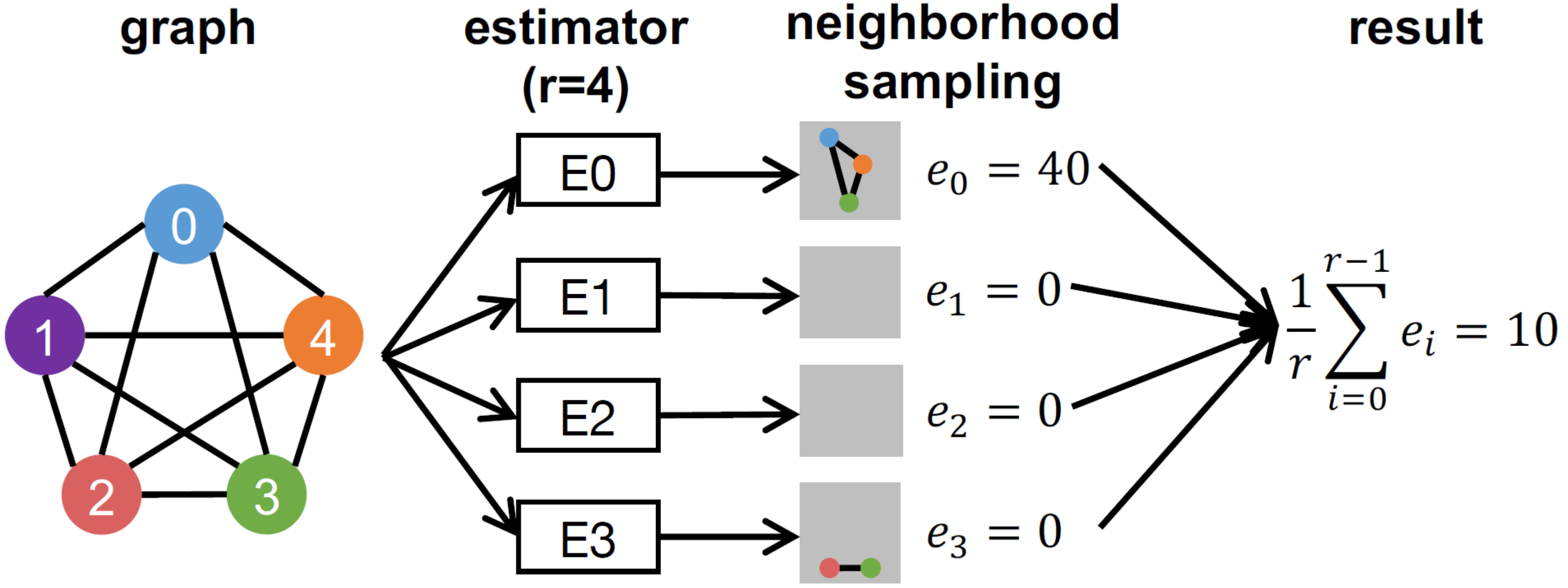
$$e_0 = 40$$

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

Approximate Pattern Mining

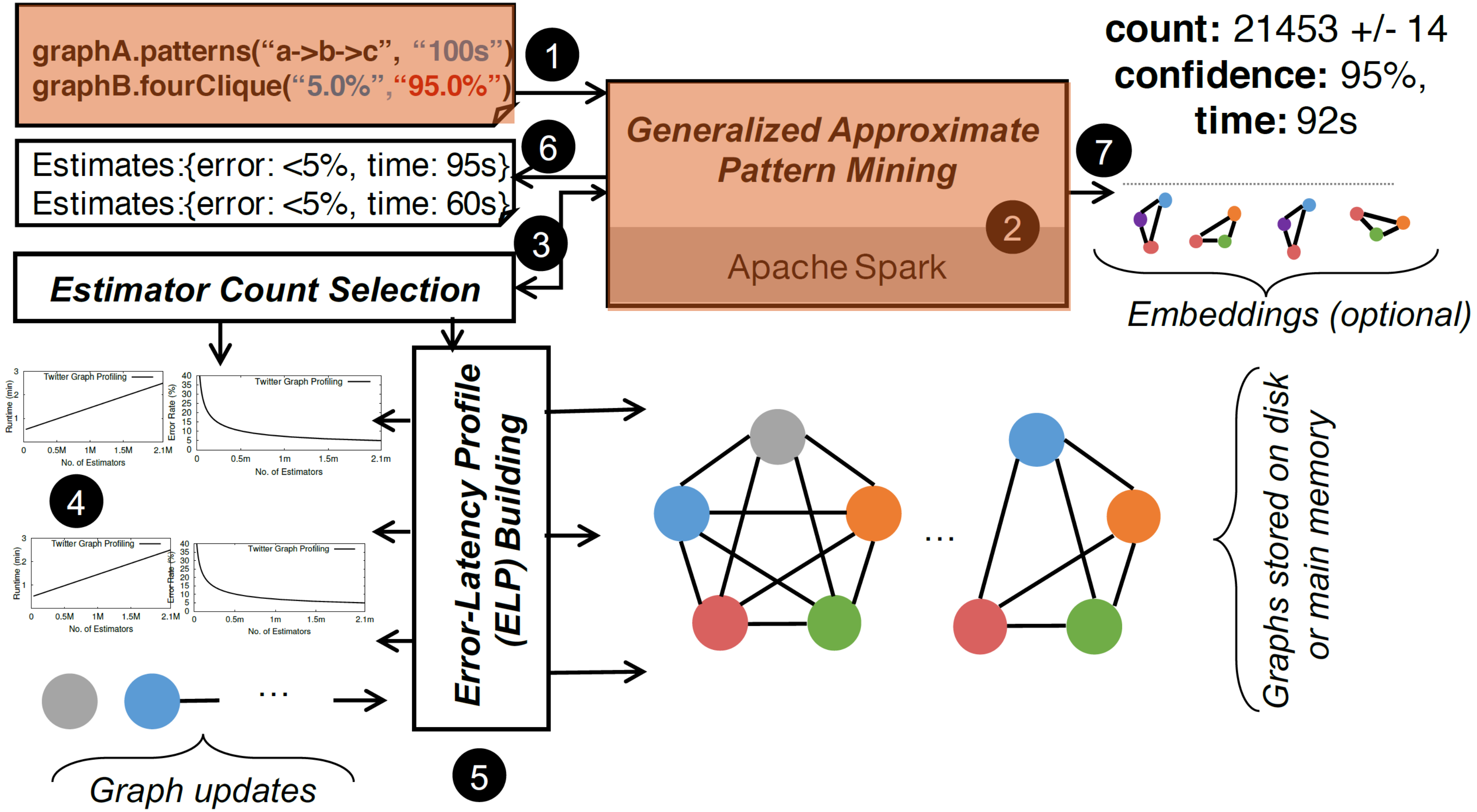
Neighborhood sampling: Triangle Counting

6. Repeat Step 1-5 multiple times



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

ASAP Architecture



Programming API

Neighborhood sampling:

1. Model the edges in the graph as a stream
2. Sample one edge, e_1
3. Gradually add more adjacent edges, e_2, …, e_k
4. Stop when the edges form the pattern or becomes impossible to do so
5. Use the probability of sampling to bound the total number of occurrences of the pattern: $P(e_1, \ldots, e_k) = P(e_1) \times P(e_2 \mid e_1) \times \ldots \times P(e_k \mid e_1, \ldots, e_{k-1})$
6. Repeat Step 1-5 multiple times

API
`sampleVertex`: $() \rightarrow (v, p)$
`SampleEdge`: $() \rightarrow (e, p)$
`ConditionalSampleVertex`: `(subgraph)` $\rightarrow (v, p)$
`ConditionalSampleEdge`: `(subgraph)` $\rightarrow (e, p)$
`ConditionalClose`: `(subgraph, subgraph)` $\rightarrow boolean$

Programming API

```
(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
```

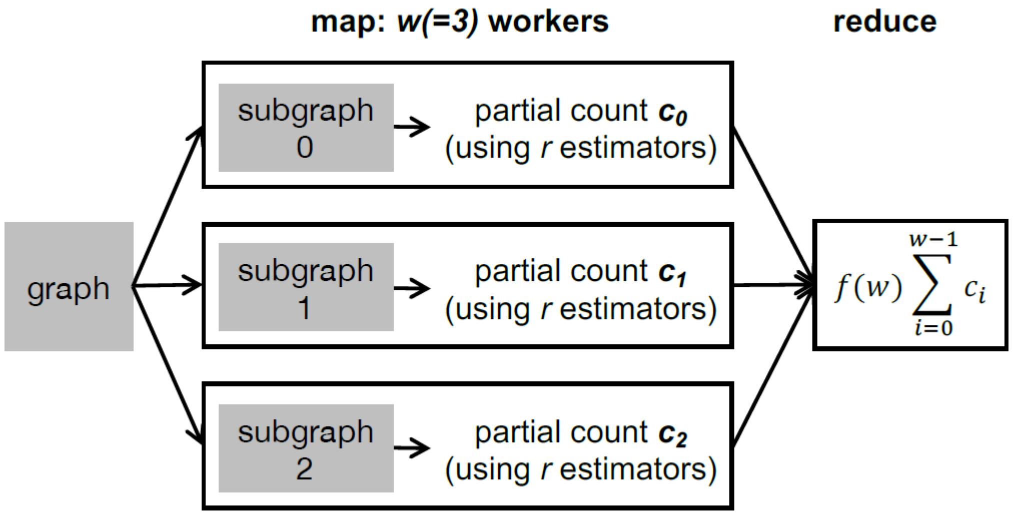
Sampling Phase: fix the vertices for a pattern

Closing Phase: waiting for remaining edges to complete the pattern

Distributed Execution

Rely on map and reduce operations

1. Partition the vertices across w workers
2. Apply estimator task on each subgraph to produce a partial count
3. Sum up partial counts
4. Adjust for underestimation by multiplying $f(w)$
   e.g. for triangle count, $f(w)=1/w^2$

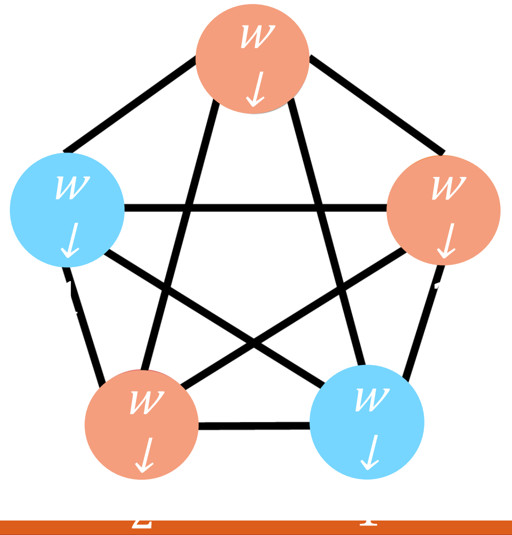


Distributed Execution

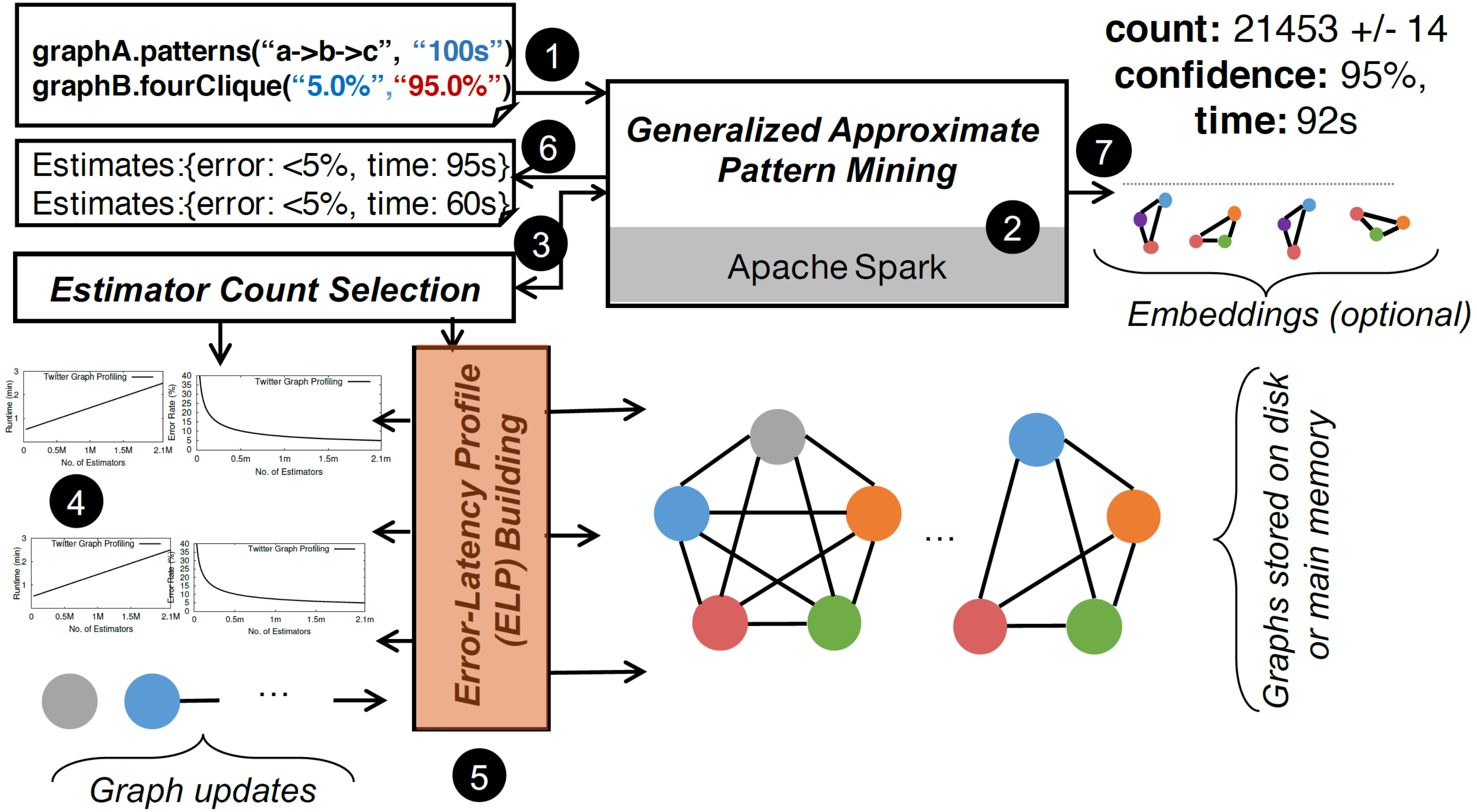
Rely on map and reduce operations

1. Partition the vertices across w workers
2. Apply estimator task on each subgraph to produce a partial count
3. Sum up partial counts
4. Adjust for underestimation by multiplying $f(w)$
   e.g. for triangle count, $f(w) = w \uparrow 2$

- Patterns across partitions are ignored
- Total occurrence is reduced by $1/f(w)$



ASAP Architecture

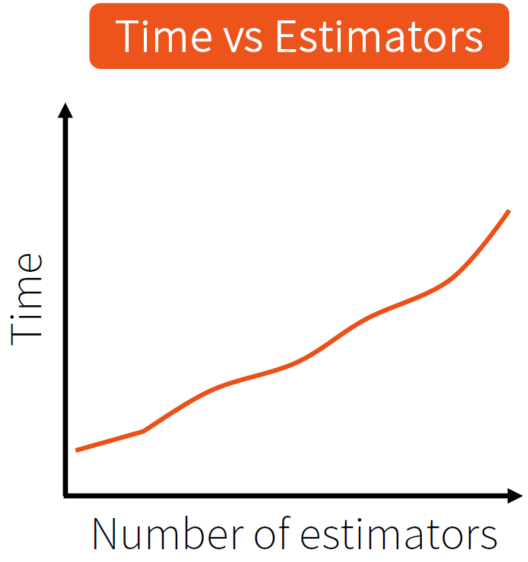


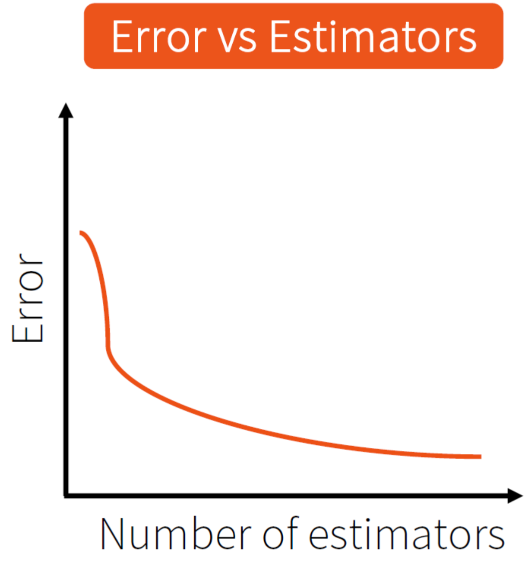
Error-Latency Profile (ELP)

ASAP can perform tasks in two modes:

- Time budget T
- Error budget ϵ

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

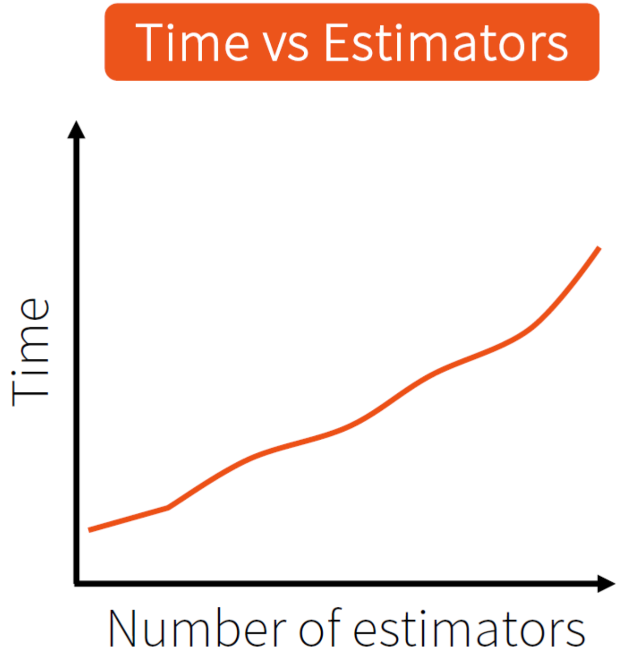




Error-Latency Profile (ELP)

Running time scales linearly with number of estimators

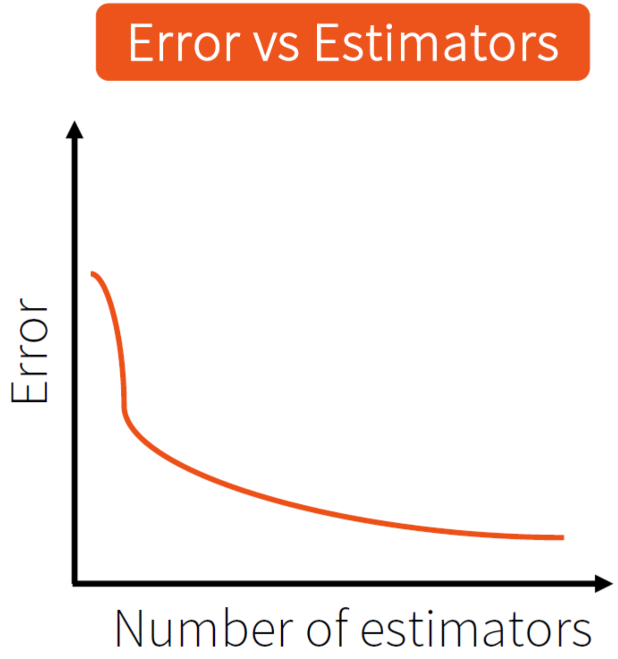
Test exponentially spaced points + extrapolation to build a linear model



Error-Latency Profile (ELP)

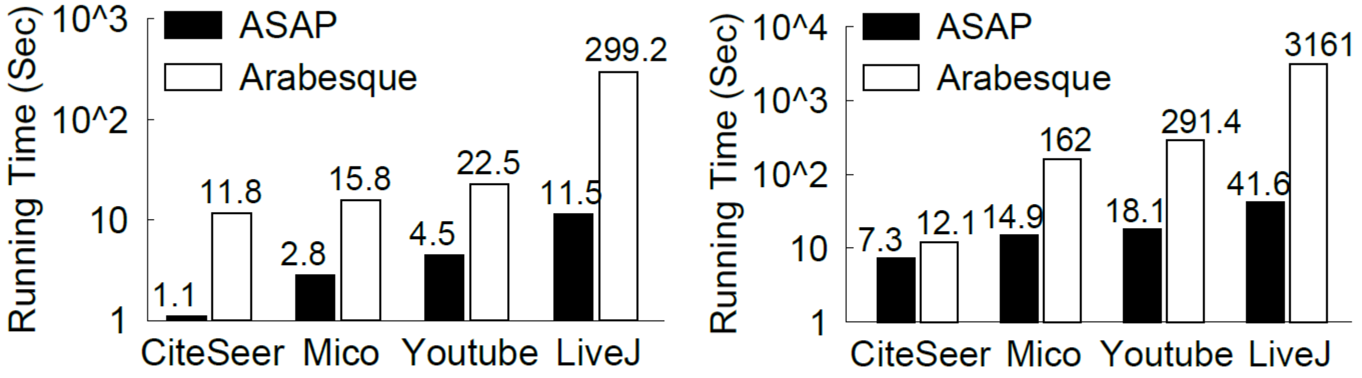
Chernoff bound for triangle counting: $N_e > K \times m \times \Delta / \epsilon^2 P$

Estimate ground truth P_s on a small sample of the graph + scale to P



Evaluation

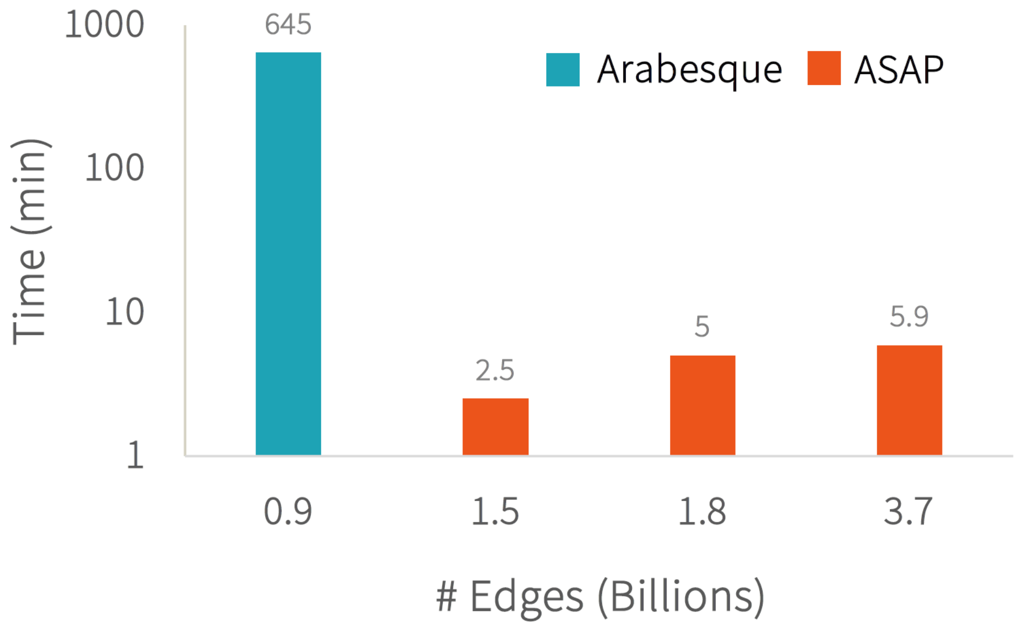
77x speedup with under 5% loss of accuracy for smaller graphs (0.01-30 million edges)



(a) 3-Motif Counting **(b)** 4-Motif Counting

Evaluation

258x speedup with under 5% loss of accuracy for larger graphs



Conclusion

ASAP is the first system that does fast, scalable approximate graph pattern mining on large graphs.

ASAP outperforms Arabesque by more than a magnitude faster with a sacrifice of 5% accuracy.

ASAP scales to larger graphs whereas Arabesque fails to complete execution.

Reference

- https://www.usenix.org/sites/default/files/conference/protected-files/osdi18_slides_iyer.pdf
- Iyer, Anand Padmanabha, et al. "ASAP: fast, approximate graph pattern mining at scale." *Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation*. USENIX Association, 2018.
- Iyer, Anand Padmanabha, et al. "Towards fast and scalable graph pattern mining." 10th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 18). USENIX} Association}, 2018.