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
Graphs popular in big data analytics
Graphs popular in big data analytics
Graphs popular in big data analytics
Graph Analytics
Graph Analytics

Processing Algorithms

PageRank

Connected Components
Graph Analytics

Processing Algorithms

- PageRank

Mining Algorithms

- Connected Components
Graph Analytics

Processing Algorithms

PageRank

Connected Components

Mining Algorithms

Connected Motifs of size 4
- Star
- Chain
- 3-loop-out
- Box
- Semi-Clique
- Clique

Motifs

Cliques
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

**Minning 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

- # Edges
- Computation Time
Graph Analytics: Processing vs Mining

PageRank

1 trillion

140 s

# Edges
Computation Time
Graph Analytics: Processing vs Mining

- # Edges
- Computation Time

PageRank

1 trillion

140 s
Graph Analytics: Processing vs Mining

- **PageRank**
  - 1 trillion edges
  - 140 seconds computation

- **Motifs with size = 3**
  - ~1 billion edges
  - 11 hours computation

Arabesque (SOSP’15)
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

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

![Diagram showing graph transformation with edge sampling and triangle counting example](image)
Approximate Analytics

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

general approach

edge sampling (p=0.5)

triangle counting

result

general approach

0

1

4

2

3

general approach

edge sampling

result

general approach

$e = 1 \rightarrow e \cdot 2 = 2$
Approximate Analytics

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

edge sampling (p=0.5) triangle counting result

graph

\[
e = 1 \Rightarrow e \cdot 2 = 2
\]

Answer: 10
Approximate Analytics

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

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

Answer: 10
Approximation by Sampling Patterns

draw a diagram here

element 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

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

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

draw a 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

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

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

graph

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

$P = \frac{1}{10} \times \frac{1}{4}$

Pavan et al. Counting and sampling triangles from a graph stream, VLDB 2013
Approximation by Sampling Patterns

graph

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

$e_0 = 40$

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

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
Building a General Purpose Approximate Graph Mining System
Building a General Purpose Approximate Graph Mining System
Building a General Purpose Approximate Graph Mining System

- General Patterns
- Distributed Settings
- Error Estimation
- Handling Updates
Challenge #1: General Patterns

Problem: Neighborhood sampling is for triangle counting
Break down neighborhood sampling into two phases:

- Sampling phase
- Closing phase

![Diagram showing graph, estimator (r=4), neighborhood sampling, and result]

\[
\frac{1}{r} \sum_{i=0}^{r-1} e_i = 10
\]

- \( e_0 = 40 \)
- \( e_1 = 0 \)
- \( e_2 = 0 \)
- \( e_3 = 0 \)
Challenge #1: General Patterns

Problem: 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

Problem: 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

map: $w(=3)$ workers

graph

- subgraph 0 ➔ partial count $c_0$ (using $r$ estimators)
- subgraph 1 ➔ partial count $c_1$ (using $r$ estimators)
- subgraph 2 ➔ partial count $c_2$ (using $r$ estimators)
Challenge #2: Distributed Setting

Problem: Neighborhood sampling is for a single machine

map: \( w(=3) \) workers

graph

- subgraph 0 → partial count \( c_0 \) (using \( r \) estimators)
- subgraph 1 → partial count \( c_1 \) (using \( r \) estimators)
- subgraph 2 → partial count \( c_2 \) (using \( r \) estimators)
Challenge #2: Distributed Setting

Problem: Neighborhood sampling is for a single machine

map: \( w(=3) \) workers

- subgraph 0: partial count \( c_0 \) (using \( r \) estimators)
- subgraph 1: partial count \( c_1 \) (using \( r \) estimators)
- subgraph 2: partial count \( c_2 \) (using \( r \) estimators)

\[
\sum_{i=0}^{w-1} c_i
\]
Challenge #2: Distributed Setting

Problem: Neighborhood sampling is for a single machine

Map: $w(=3)$ workers

Reduce:

$$\sum_{i=0}^{w-1} c_i$$
Challenge #2: Distributed Setting

Problem: Neighborhood sampling is for a single machine

map: $w(=3)$ workers

reduce

$\text{subgraph 0} \rightarrow \text{partial count } c_0$
(assuming $r$ estimators)

$\text{subgraph 1} \rightarrow \text{partial count } c_1$
(assuming $r$ estimators)

$\text{subgraph 2} \rightarrow \text{partial count } c_2$
(assuming $r$ estimators)

\[
f(w) \sum_{i=0}^{w-1} c_i\]
Challenge #2: Distributed Setting

Problem: Neighborhood sampling is for a single machine

map: \( w(=3) \) workers

\[
\text{subgraph 0} \rightarrow \text{partial count } c_0 \\
\text{(using } r \text{ estimators)}
\]

\[
\text{subgraph 1} \rightarrow \text{partial count } c_1 \\
\text{(using } r \text{ estimators)}
\]

\[
f(w) \sum_{i=0}^{w-1} c_i
\]

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

![Twitter Graph](image-url)
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