Advanced Topics in Distributed Systems

Mar 25, 2020

TABLE OF CONTENTS

INTRODUCTION

FREQUENT SUBGRAPH MINING

Goal of Frequent Subgraph Mining (FSM) is to extract frequent subgraphs from graphs.

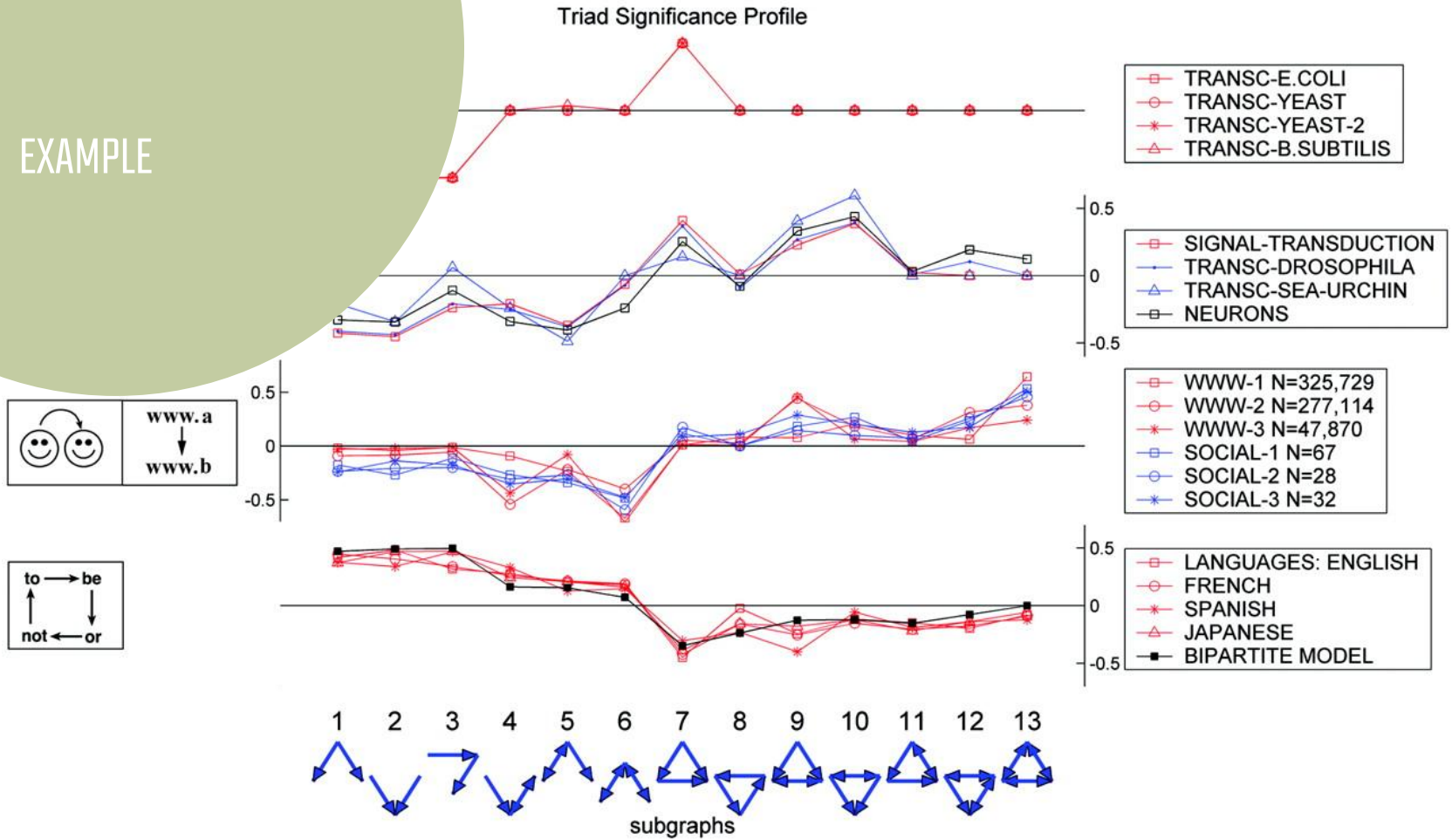
- **Graph.** $G = (V, E)$
- **Subgraph.** $G' = (V', E')$

$$V' \subseteq V \text{ and } E' \subseteq E$$

(Graph Isomorphism is in **NP**.) Subgraph Isomorphism is in **NP-Complete**.

Frequent Subgraphs are used in **chemistry**, **analysis of social networks**, etc.

EXAMPLE



Ron Milo et al. *Superfamilies of Evolved and Designed Networks.* In *Science*. 2004.

MapReduce

Large-Scale Frequent Subgraph Mining in MapReduce

INTRODUCTION

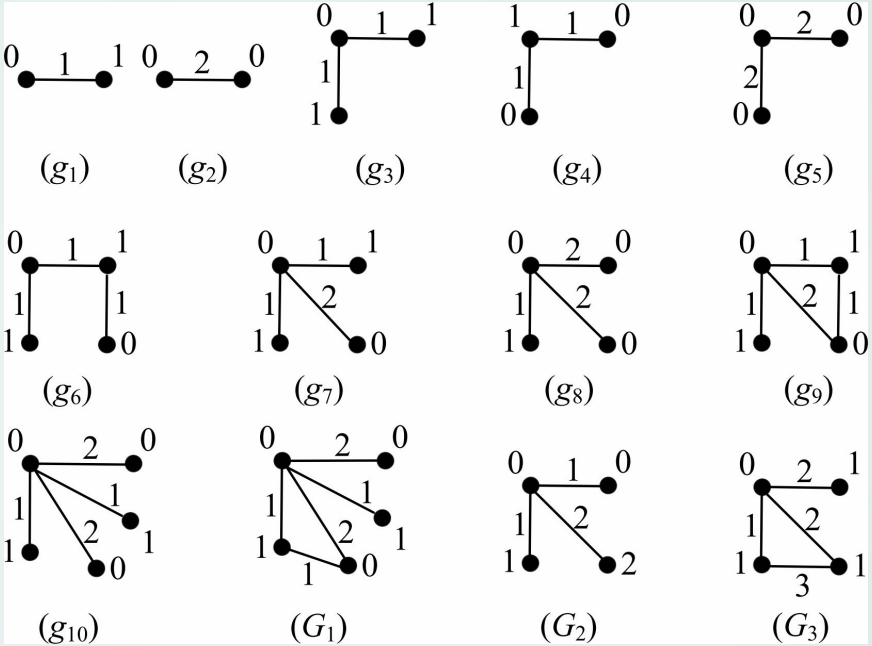
- **Transaction.** We usually mine 1 large graph, but in this case we will mine *N* small graphs.

It is worth remembering that vertices and edges are **labeled**, in this case.

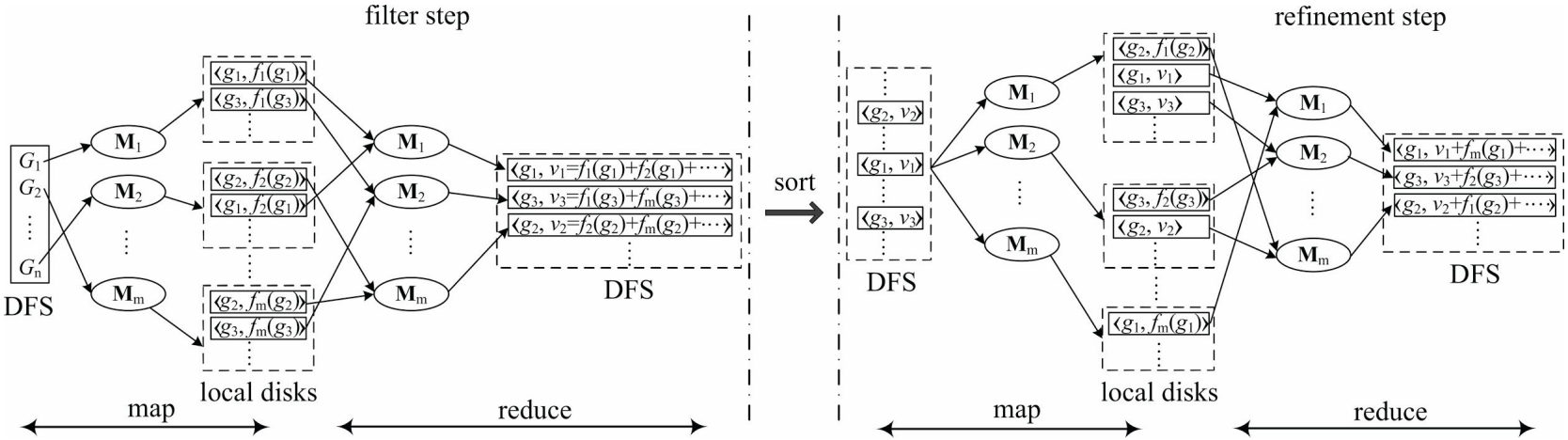
Like [4], MapReduce Frequent Subgraph Mining (MRFSM), which is based on MapReduce, is able to find subgraphs whose support is greater than θ, where θ is the threshold.

EXAMPLE

- $f(g_1) = |\{G_1, G_2, G_3\}| = 3$
- $f(g_{10}) = |\{G_1\}| = 1$



MRFSM


filter step
refinement step
DFS
local disks
map
reduce
sort

FILTER

- **Map.** After dividing G into $G_1, G_2, \ldots, G_m$, we assign G_i to M_i, which signals g_i if $f(g_i)/n_i \geq \theta$, where $n_i = |G_i|$.
- **Reduce.** Write the union of the sets of the candidates to DFS.

Unfortunately, there is a risk of a large number of **false positives**, because it is possible for subgraphs to be frequent at the level of G_i but infrequent at the level of G.

FILTER

- **Map.** We distribute the graphs in G among the machines, randomly. M_i returns frequent subgraphs as well as infrequent subgraphs that are selected as detailed below.
- **Reduce.** If the upper bound of $f(g_i)$, *i.e.*, $f^{\top}(g_i)$, is greater than $\theta \cdot n$, then we will promote g_i to candidate.

$$f^{\top}(g_i) = \sum f_i^{\top}(g_i)$$

If $f_i^{\top}(g_i)$ is unknown, then we will use $\lceil n_i \cdot \theta \rceil - 1$ as a proxy.

FILTER

It is possible for subgraphs to be infrequent at the level of G_i but frequent at the level of G.

$G \setminus G_i$ is unknown. On M_f it is necessary to compute the **probability** of a subgraph that is infrequent at the level of G_i being globally frequent. In theory, frequency follows a **binomial distribution**.

If the upper bound of the probability is greater than ρ, then we will approve a locally infrequent subgraph.

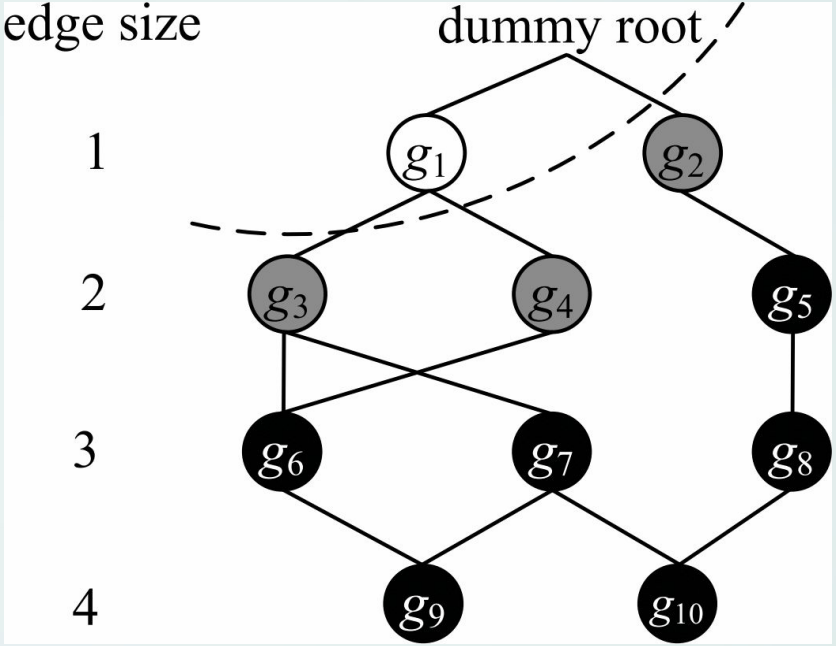
BINOMIAL DISTRIBUTION

$p = f_i(G') / n_j$

$$Pr\left\{\exists j \neq i, f_j\left(G'\right) \geq \lceil \theta \cdot n_j \rceil\right\}$$
$$\leq \sum_{j \neq i} Pr\left\{ f_i\left(G'\right) \geq \lceil \theta \cdot \rceil\right\}$$
$$= \sum_{j \neq i} \sum_{k=\lceil \theta \cdot n_j \rceil}^{n_j} \binom{n_j}{k} p^k \left(1-p\right)^{n_j - k}$$

LATTICE

It is possible to construct a lattice by adding an edge at a time.



REFINEMENT

From **Sorting**, M_i receives S — in other words, pairs of
- **keys**, which are subgraphs; and
- **values**, which are frequencies.

$A^i(g)$ is the set of supergraphs of g, in case of M_i,

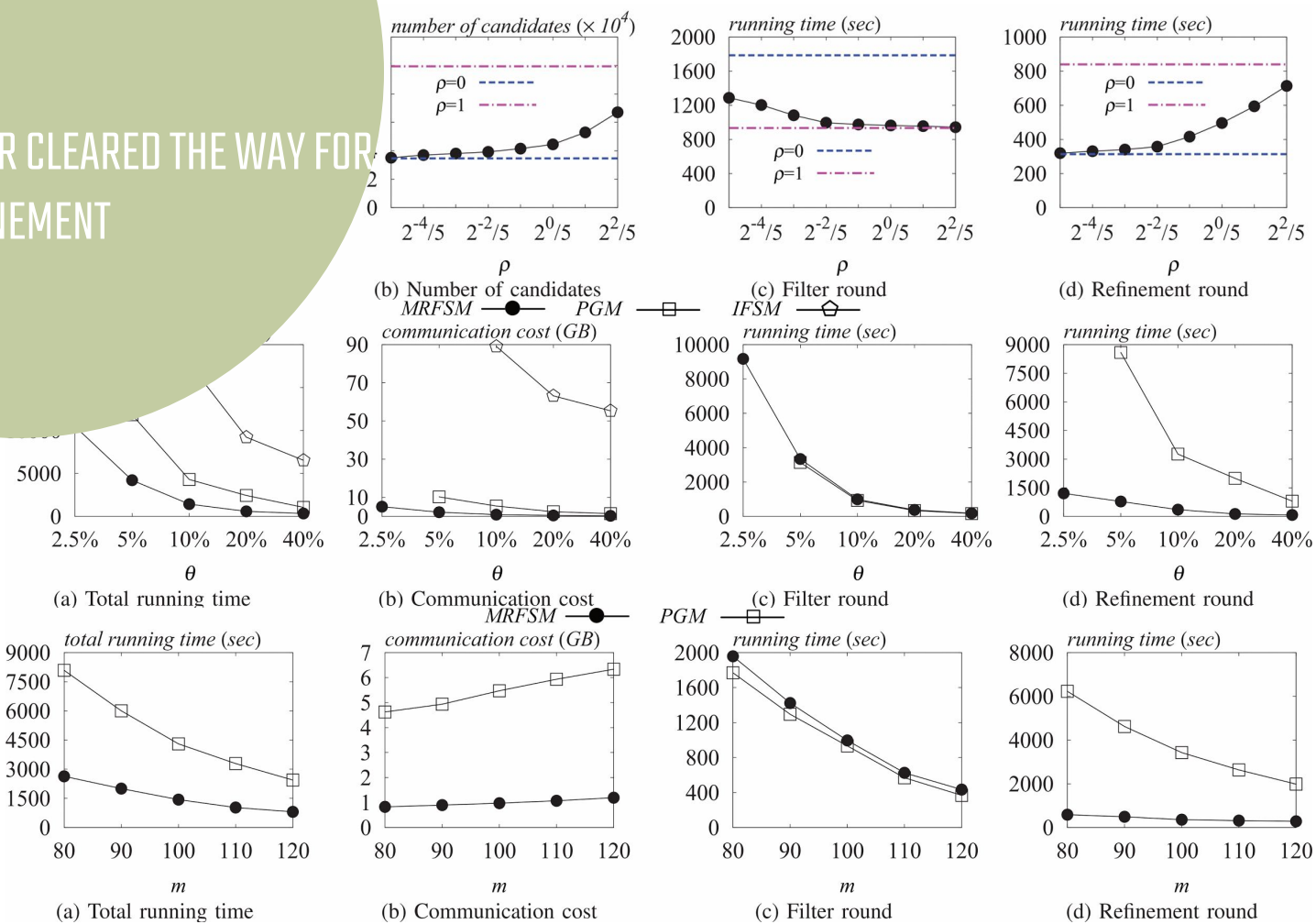
- **Map.**
  For example, g_7 and g_8 are subgraphs of g_{10}.
  - **Top-Down.**
    - **First.** Pairs are in ascending order of size. $ub(g_{10}) = A^i(g_7) \cap A^i(g_8)$
  - **Bottom-Up.**
    - **First.** Pairs are in ascending order of size. $ub(g_{10}) = ub(g_7) \cap ub(g_8)$.
    - **Second.** Pairs are in descending order of size. In case of g_8, it is necessary to mine $ub(g_8) \setminus lb(g_8)$, which is $ub(g_8) \setminus A^i(g_{10})$.
- **Reduce.** Compute the frequencies of the frequent subgraphs, by summing.

COMPRESSION

We need to reduce our overheads, via **compression of canonical labeling**.

Like [5], $G = (V, E, L_V, l_V, L_e)$, which is (de)compressed. Also, spanning trees, plus $E \setminus E'$, where E is the set of edges of I and E' is the set of edges of ST, are encoded. Indeed, there is some evidence that frequent subgraphs are commonly spanning trees.

FILTER CLEARED THE WAY FOR REFINEMENT



Pregel

Frequent Subgraph Mining
Based on Pregel

Pregel

MapReduce is not specialized for graphs, but there is a variety of graph processing distributed systems such as **Pregel**.

With every superstep, we

- **receive** messages;
- do something; and
- **send** messages.

Bulk Synchronous Parallel (BPS) is used for Pregel. Indeed, there is a **barrier** between superstep and superstep.

Apache Giraph is the open-source equivalent of Pregel.

INTRODUCTION

Unfortunately, Pregel is not good with tasks that imply structure. See [6]. (😐)
For this reason, we need a number of **aggregators**.

Zhao et al. propose Pregel-Based Frequent Subgraph Mining, a.k.a. **pegi**, which returns a set of frequent subgraphs, starting from a (labeled) graph.

N.B. If *g* is a subgraph of *G*, then we will call *f*(*g*) an **embedding** of *g* in *G*.

1 MACHINE

Algorithm 1: Baseline(G, τ)

Input : G is a graph; τ is a support threshold.
Output: P is a set of frequent subgraphs, initialized to $\emptyset$.

```
1  P ← P^1 ← find frequent single edges in G;
2  foreach edge e ∈ P^1 do DFSMine (G, e) ;

   Function DFSMine(G, p)
3      enumerate 1-edge extension of p and embeddings;
4      foreach enumerated edge e for p do
5          p' ← p ∪ {e};
6          if e is a forward edge with target vertex v
           then
7              if φ_{p'}(v) < τ then continue;
8          if φ(p') < τ ∨ p' ∈ P then continue;
9          P ← P ∪ {p'} ;        /* find an answer */
10         DFSMine (G, p');
```

M-MACHINES

Algorithm 2: master. compute()

```
1  switch phase do
2      case VERTEX:
3          V_f ← GetAggregatedValue(frq_v);
4          V_f ← get frequent vertices with images;
5          V_t ← retrieve images of first vertex in V_f;
6          Aggregate(nxt_v,V_t);
7      case GROW:
8          e ← GrowPattern(E_c, Φ);
9          Aggregate(nxt_e,e);
10     case UPDATE:
11         V_t ← GetAggregatedValue(nxt_v);
12         update global embedding tree by adding edges
           incident on V_t;
```

MASTER

WORKER

Algorithm 3: vertex. compute()

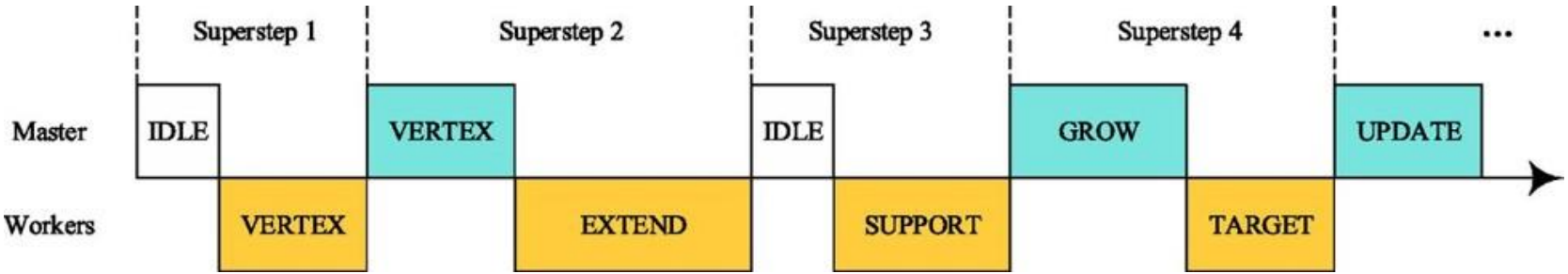
```
1  switch phase do
2      case VERTEX: Aggregate(frq_v, this.l_v) ;
3      case EXTEND: ExploreEdge(V_t) ;
4      case SUPPORT:
5          foreach distinct message m do
6              Aggregate(m.e, 1);
7      case TARGET:
8          if this is backtrack then update local
           embedding tree by removing last updated
           edges;
9          e ← GetAggregatedValue(nxt_e);
10         Aggregate(nxt_v, V_t(e));
```

N MACHINES

pattern growth and **embedding discovery**

- **Worker.** Send labels of vertices.
- **Master.** Produce set of frequent vertices, that is, frequent subgraphs of size 1. Share target vertex.
- **Worker.** Starting from target vertex, send messages to neighbors.
- **Worker.** Upon receive, compute (local) support, which we share with aggregator.
- **Master.** Select edge to be added.
- **Worker.** Send embeddings.
- **Master.** Update embedding trees.
- etc.

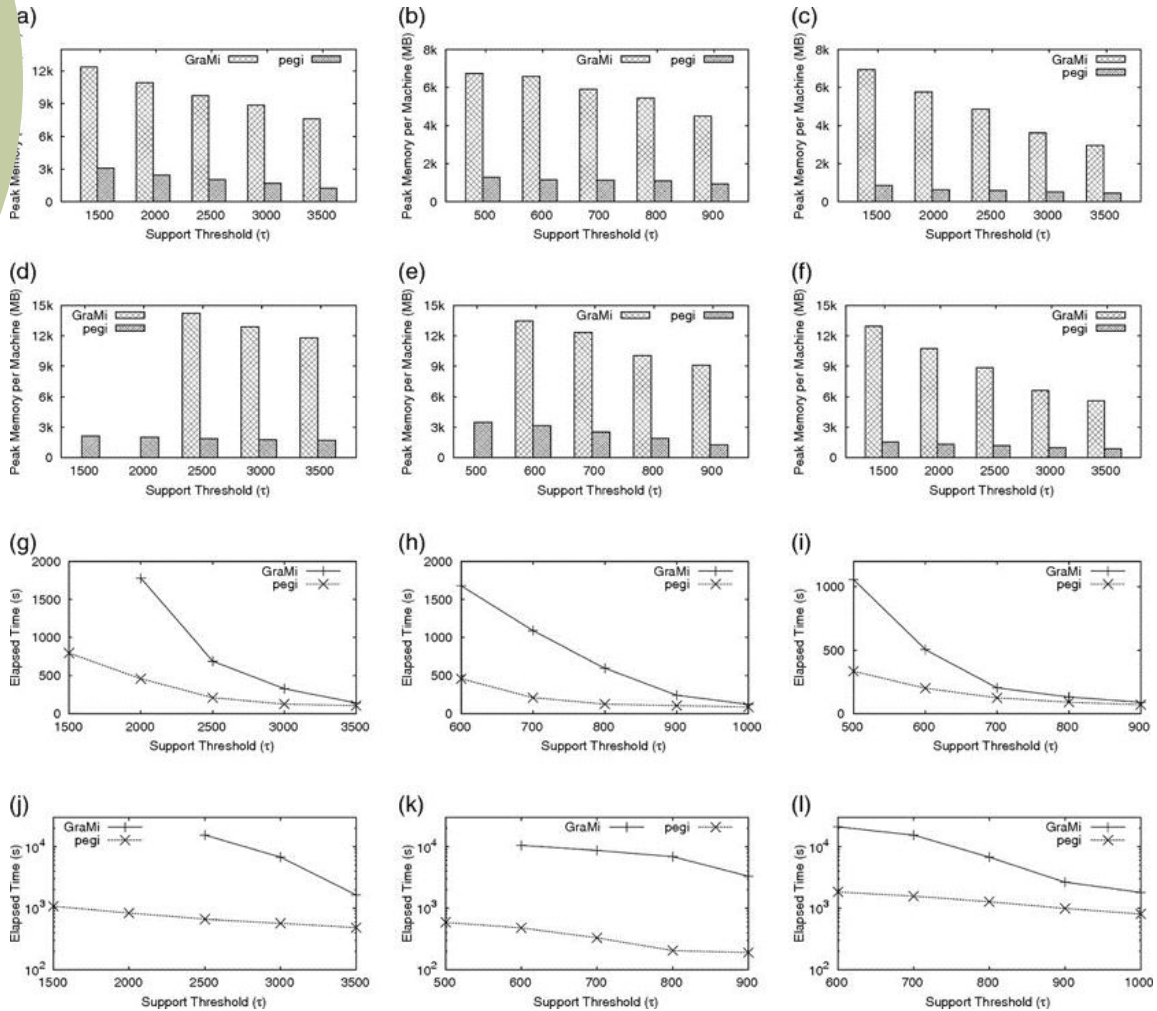
FLOW


Superstep 1
Superstep 2
Superstep 3
Superstep 4
...
Master
IDLE
VERTEX
IDLE
GROW
UPDATE
Workers
VERTEX
EXTEND
SUPPORT
TARGET

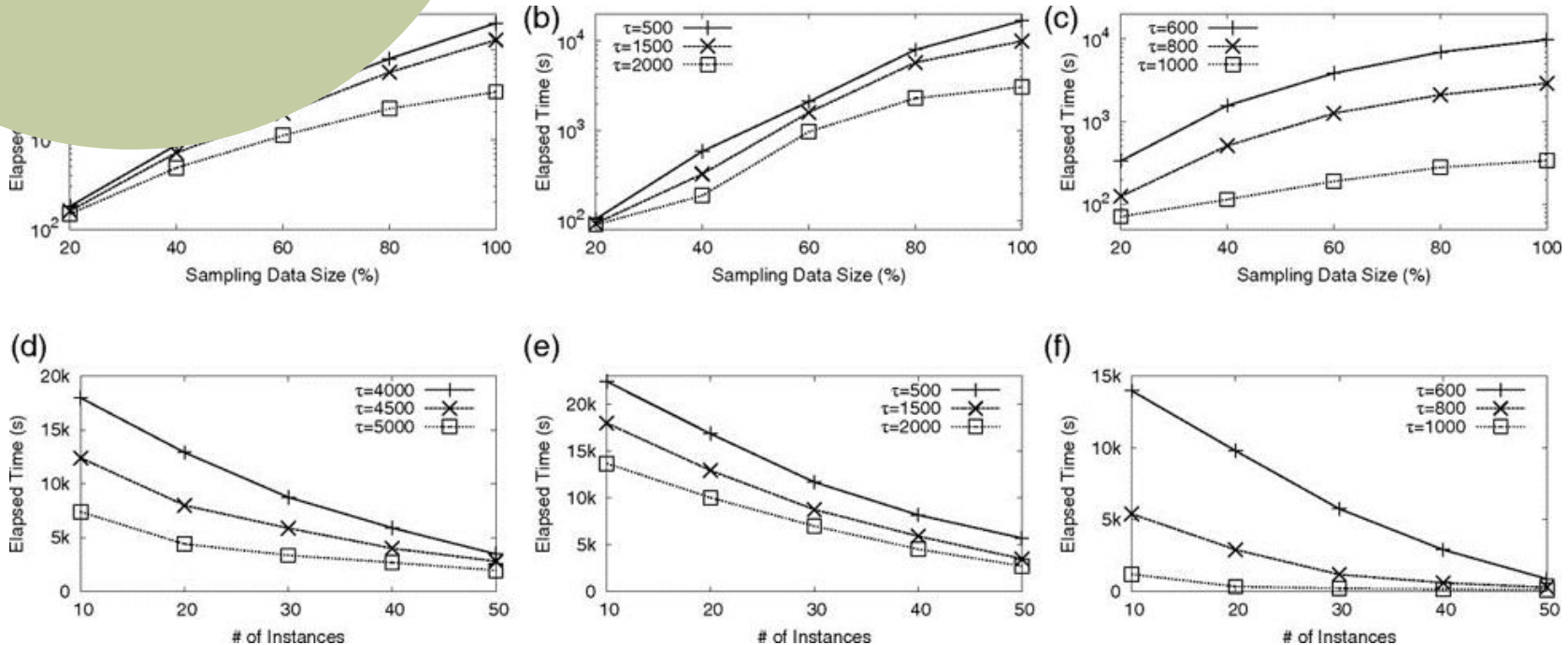
OPTIMIZATIONS

- **+Filter.** If the upper bound of the estimation of the support of *u* is less than the threshold, then we will skip *u*.
- **+Backward.** There is a large number of superstes and barriers. So, we do pattern grow until a new vertex is added.

GraphMi VERSUS pegi



SCALABILITY



GRAPHLETS

Graphlets are, by definition, **induced** subgraphs.

Graphlets enable the analysis of **higher-order** networks, which highlight connections that are in the background. For example, they are used to detect not only **communities** but also **roles**.

In case of graphlet counting, it is necessary

- to **enumerate**; and
- to **count**.

For example, $k = 4$. There are two to the power of sixteen (2^{2k}) subgraphs and only six graphlets in this case.

LEGO

It is possible to view a graph as the union of graphlets, instead of vertices and edges.

PARALLEL

Leveraging Multiple GPUs and CPUs for Graphlet Counting in Large Networks

INTRODUCTION

Rossi and Zhou propose an algorithm that is used to count the number of **connected** and **disconnected** graphlets of size $k = \{2, 3, 4\}$ at the graph level and at the edge level.

In comparison with CPUs, GPUs have the advantage of

- higher **performance** in terms of FLOPS, and
- higher **memory bandwidth** in terms of B/s.

Thus, Rossi and Zhou propose a hybrid algorithm, which is able to leverage multiple GPUs and CPUs. Authors present **load balancing** and **work stealing**, too.

VERTEX VERSUS EDGE

Unlike MapReduce and Pregel, which are vertexcentric, Rossi and Zhou propose an algorithm that counts the number of graphlets per edge.

z_{ij} is the number of graphlets H_j per vertex v_i, x_{ij} is the number of graphlets H_j per edge e_i,

$$Z_j = \sum z_{ij} \text{ and } X_j = \sum x_{ij}$$

Of course,

$$Z_j = X_j$$

Since the number of vertices is much less than the number of edges, *i.e.*,

$$N \ll M$$

, the average number of graphlets H_j per edge is less than the average number of graphlets H_j per vertex, *i.e.*,

$$X_j / M < Z_j / N$$

Overall, an edgecentric algorithm is better at **load balancing** than a vertexcentric algorithm.

GPUs AND CPUs

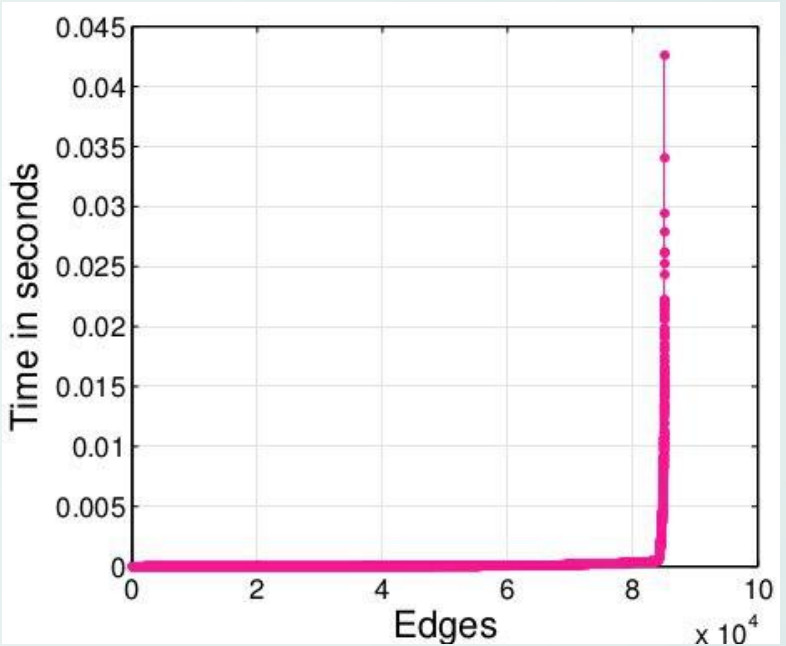
CPUs and GPUs have both their **pros and cons**.

On the one hand, GPUs are, by definition, parallel, and as a result they are able to run a large number of tasks simultaneously. It is worth remembering that they are good at homogeneous tasks.
On the other hand, CPUs are able to run a wide range of tasks.

So, it makes sense to assign edges with a large number of neighbors to CPUs and thus to assign edges with a small number of neighbors to GPUs.

POWER LAW

Networks are known to show a **degree distribution** that is a power law.



QUEUE

$$\Pi = \{\underbrace{e_1, \ldots, e_k}_{\Pi_{\text{CPUs}}}, \overbrace{e_{k+1}, \ldots, e_{j-1}}^{\text{unprocessed } (j-k)-1}, \underbrace{e_j, e_{j+1}, \ldots, e_M}_{\Pi_{\text{GPUs}}}\}$$

DETAILS

Edges are in descending order of **difficulty**, which is, by definition, proportional to number of neighbors.

$$\Gamma(e) = \Gamma(u, v) = \{\Gamma(u) \cup \Gamma(v) \setminus \{u, v\}\}$$

The whole idea of graphlet counting is to make processes be independent. That it, we are able to mine neighborhood in parallel.

- **CPUs** are responsible for the first b_{CPUs} in the queue.
- **GPUs** are responsible for the last b_{GPUs} in the queue. Tasks are assigned in a **round robin**.

$$b_{CPUs} < b_{GPUs}$$

Thanks to b_{CPUs} and b_{GPUs}, it is possible to switch from **single GPU**, to **multi-GPU**, and to **hybrid multi-core CPU-GPU**.

TRIANGLES, CLIQUES, AND CYCLES

Rossi and Zhou propose a series of algorithms for CPUs and a series of algorithms for GPUs. The whole idea of the article is to make the most of their strength and weaknesses. For example, hash algorithms are used for CPUs and search algorithms are used for GPUs.

- T is the set of vertices that are in **triangles** with e_i
- S_u is the set of vertices that are in **two-stars**, whose center is u, with e_i

It is possible to compute the number of graphlets of size $k = \{2, 3, 4\}$ at the edge level and at the graph level, starting from the number of **triangles**, *i.e.*, $\mathbf{X}_{k,3}$, the number of **cliques**, *i.e.*, $\mathbf{X}_{k,7}$, and the number of **cycles**, *i.e.*, $\mathbf{X}_{k,10}$, plus the number of vertices, i.e., N, and the number of edges, i.e., M.

$$\mathbf{X}_{k,3} = |T|$$

$\mathbf{X}_{k,7}$ and $\mathbf{X}_{k,10}$, are derived, starting with T and S_u, too.

$k=3$

$$C_3 = \sum_{e_k=(v,u)\in E} \mathbf{X}_{k,3} = \sum_{e_k=(v,u)\in E} |T|$$

$$C_4 = \sum_{e_k=(v,u)\in E} |S_v| + |S_u|$$

$$C_5 = \sum_{e_k=(v,u)\in E} N - (|S_v| + |S_u| + |T|) - 2$$

GRAPH

N and *M* are known.

EDGE

It is worth remembering that *T* and S_u and S_v are known.

$$X_3 = \frac{1}{3} \cdot C_3$$

$$X_4 = \frac{1}{2} \cdot C_4$$

$$X_5 = C_5$$

$$X_6 = \binom{N}{3} - (X_3 + X_4 + X_5)$$

$k=2$

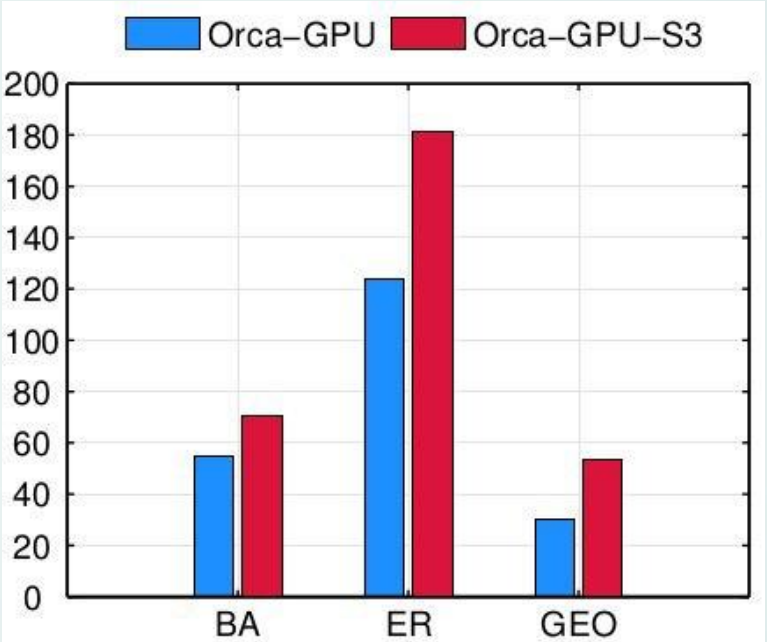
- the number of connected graphlets is equal to the number of edges
- the number of disconnected graphlets is equal to the complementary of the number of edges

$$X_1 = M$$

$$X_2 = \binom{N}{2} - M$$

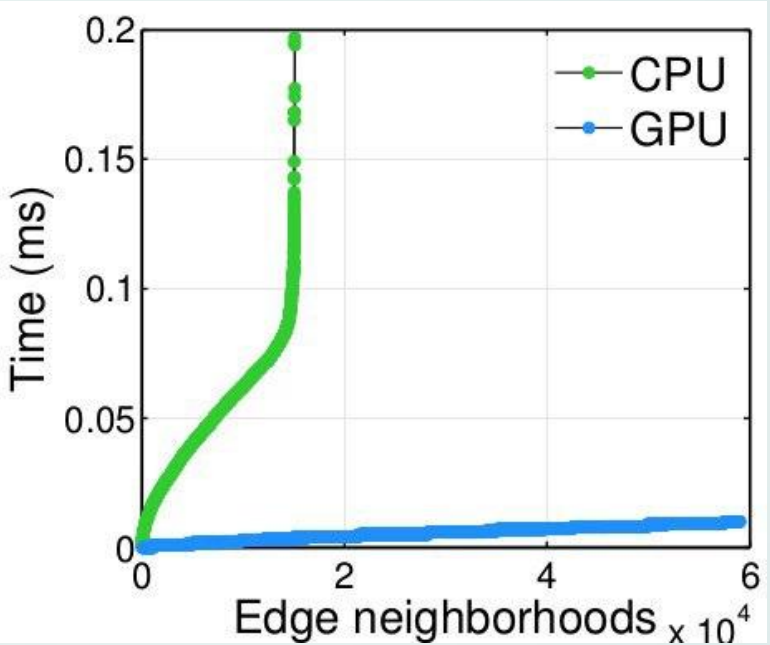
TIME

Compared to [7], which is designed in the single GPU setting, Rossi and Zhou have improved their times by 102%, on average.



DEGREE

In practice, degree provides an accurate estimation of the difficulty.



CONCLUSION

CONCLUSION

01

MapReduce

Based on MapReduce, it resolves issues that are caused by having a partial knowledge, via Probability.

02

Pregel

Unfortunately, Pregel is not good with structure-related tasks. Aggregators are used for coordination.

03

PARALLEL

Both GPUs and CPUs are used for graphlet counting. Also, it moves from vertexcentrism to edgecentrism.

References

[1] Wenqing Lin, Xiaokui Xiao, and Gabriel Ghinita. *Large-Scale Frequent Subgraph Mining in MapReduce.* In *2014 IEEE 30th International Conference on Data Engineering*. 2014.

[2] Xiang Zhao et al. *Frequent Subgraph Mining Based on Pregel. The Computer Journal*. 2016.

[3] Ryan A. Rossi and Rong Zhou. *Leveraging Multiple GPUs and CPUs for Graphlet Counting in Large Networks.* In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. 2016.

[4] Rakesh Agrawal and Ramakrishnan Srikant. *Fast Algorithms for Mining Association Rules.* In *Proceedings of the 20th International Conference on Very Large Data Bases*. 1994.

[5] Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. *An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data.* In *European Conference on Principles of Data Mining and Knowledge Discovery*. 2000.

[6] Jun Gao et al. *Continuous Pattern Detection over Billion-Edge Graph Using Distributed Framework.* In *2014 IEEE 30th International Conference on Data Engineering*. 2014.

[7] Aleksandar Milinković, Stevan Milinković, and Ljubomir Lazić. *A contribution to acceleration of graphlet counting.* In *Infoteh Jahorina Symposium*. 2015.

THANKS

Do you have any questions?

Susanna Pozzoli
spozzoli@kth.se