ScaleMine: Scalable Parallel Frequent Subgraph Mining in a Single Large Graph

EHAB ABDELHAMID, IBRAHIM ABDELAZIZ, PANOS KALNIS, ZUHAIR KHAYYAT, FUAD JAMOUR

Presented by Hyun Ryong (Ryan) Lee

Background and Motivation

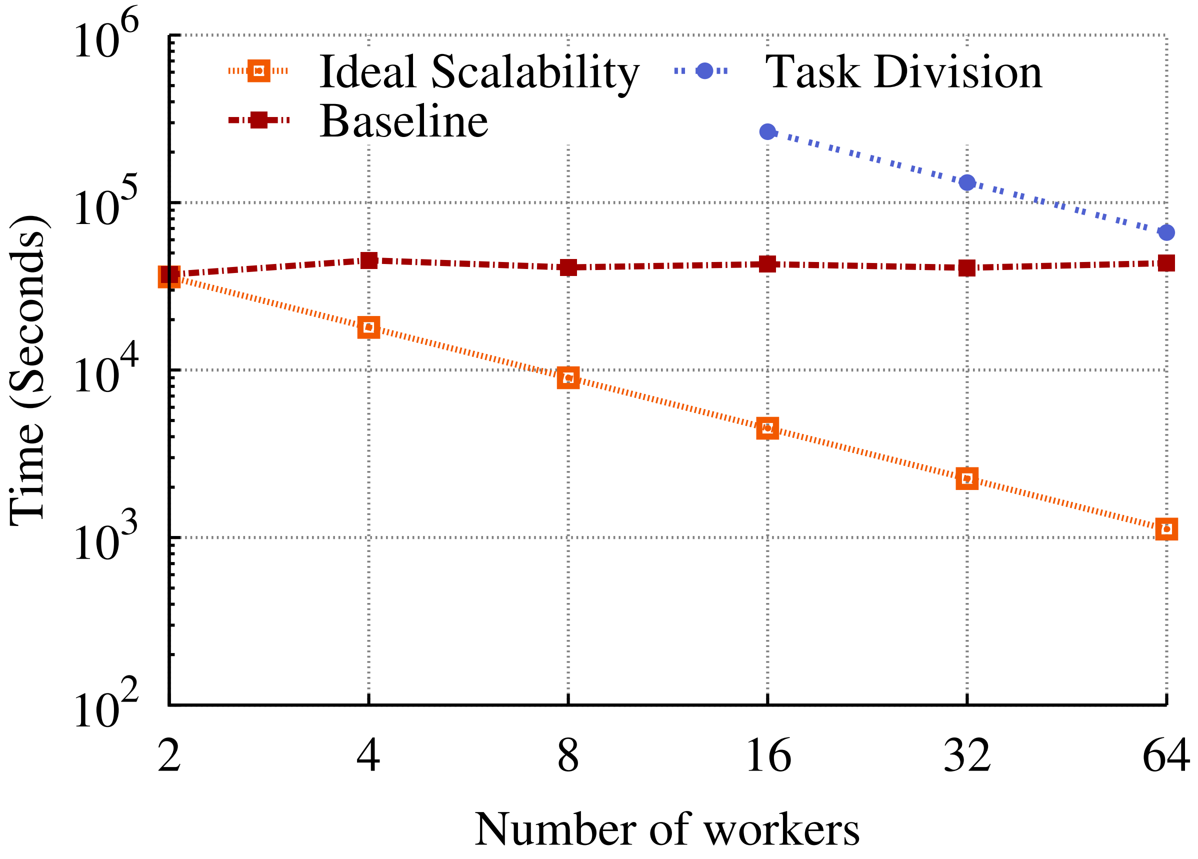
Problem: Frequent Subgraph Mining (FSM)

- Finding all subgraphs with frequency larger than a threshold.
- Essential for clustering, image processing, ...

Prior work scale poorly due to load imbalance and communication overheads

- ”Tree” of subgraphs is highly irregular -> imbalance
- Dividng up subgraph determination task has high communication and synchronization overheads.

Background and Motivation



Scalemine Solves Imbalance

Idea: Divide into two phases

- 1st Phase: approximately determine likely frequent subgraphs.
  - *Identify set of subgraphs with high probability*
  - *Collect statistics*
  - *Predict execution time for each subgraph calculation*

- 2nd Phase: Exact FSM algorithm
  - *Use candidate tasks from the 1st phase when the task pool runs low*

What is Subgraph Mining?

Given a graph G(V,E,L) with V nodes, E edges and L labels...

- S(V’,E’,L) is a subgraph of G if there is an *isomorphism relationship*
  - » *All vertices match in labels*
  - » *All edges match in labels and connectivity*

Frequent Subgraph Mining (FSM) finds subgraphs with number of matches (*support*) > τ

- This work deals with *unique* vertex matchings (called MNI metric).

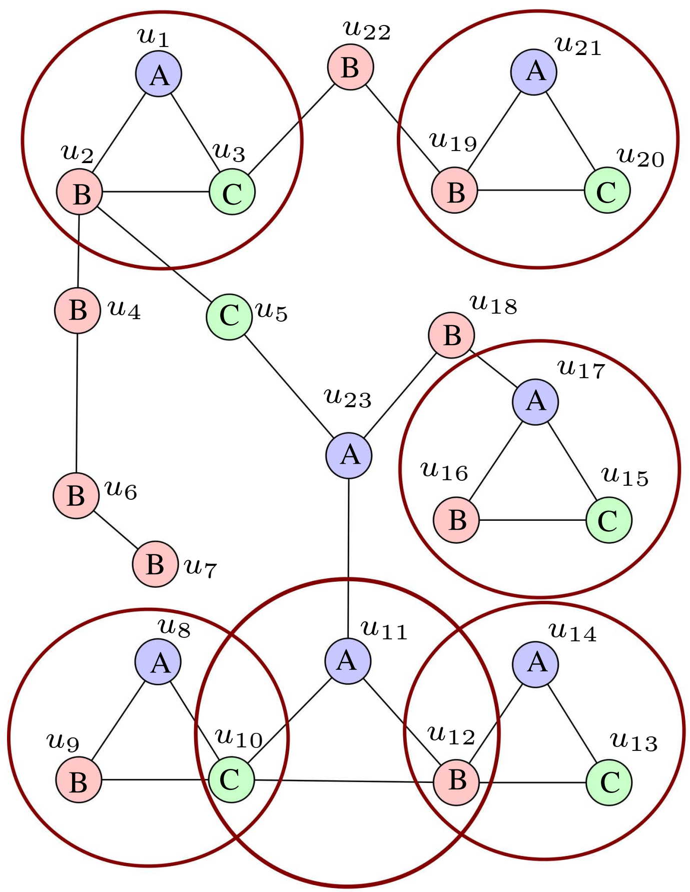
MNI Metric

Find number of distinct matches for each vertex v_i

- Create an MNI_{table} , where each column (MNI_{col}) consists of matches for the vertex (called *valid nodes*)
- The number of entries in *all* columns > τ -> valid subgraph

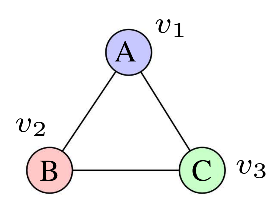
MNI Metric

Input Graph G



(a)

Subgraph S



(b)

MNI Table

v_1	v_2	v_3
u_1	u_2	u_3
u_{21}	u_{19}	u_{20}
u_{17}	u_{16}	u_{15}
u_{14}	u_{12}	u_{13}
u_{11}	u_9	u_{10}
u_8		

(c)

Approximation Phase

Goals

- Representative
- Efficient
- Informative

Approach: Use *sampling* to construct a set of subgraphs with high probability of being frequent

Approximation Phase

Given probability of success p_i, and number of nodes N_i...

- $\mathbf{MNI_{col}(v_i) = N_i p_i}$
- But we don't know p_i!

Use the Central Limit Theorem to estimate p_i

- Distribution of means of a large number of i.i.d. random variables is approximately **normal**, regardless of underlying distribution

$$\hat{\mu} = n\hat{p} \qquad \hat{\sigma} = \frac{\sigma}{\sqrt{n}}$$

Approximation Phase

Define a **vague area** for inconclusive estimates

$$low = \hat{\mu} - (z\hat{\sigma})$$

$$high = \hat{\mu} + (z\hat{\sigma})$$

Approximation Phase

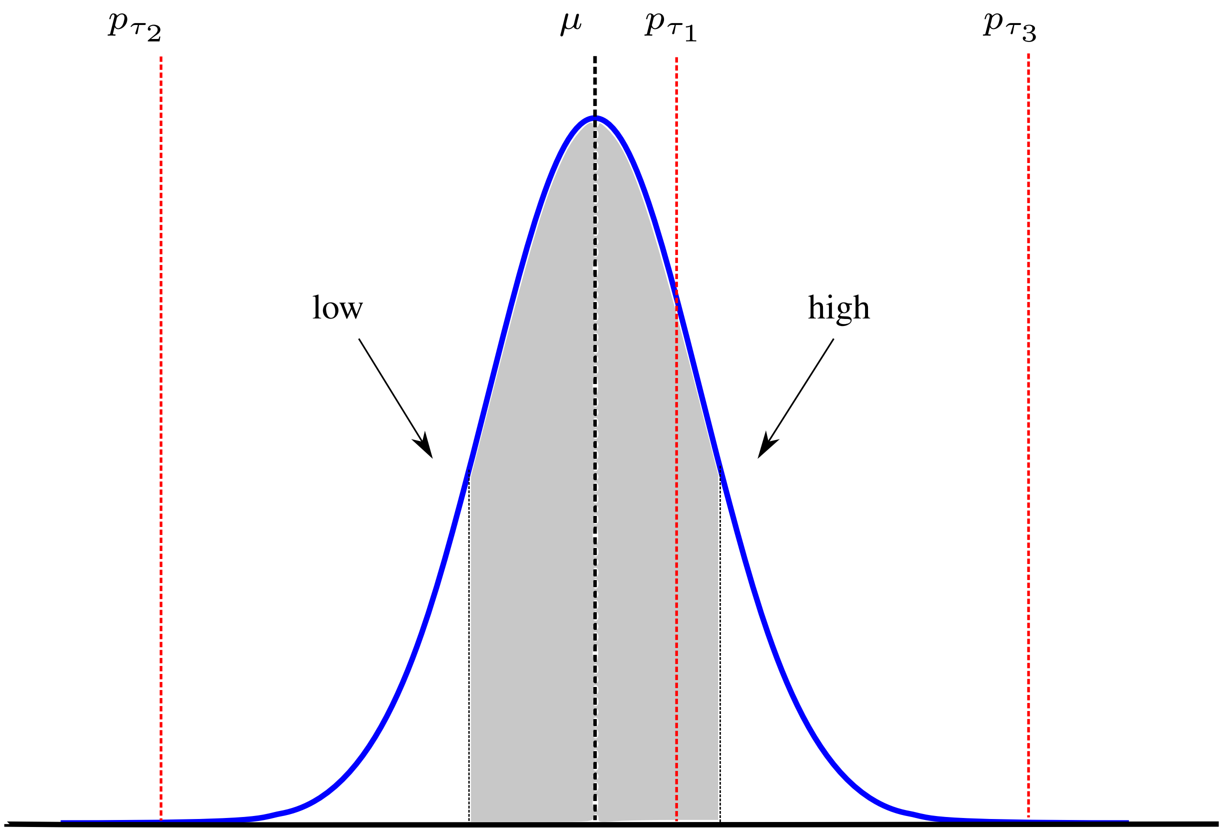


Fig. 3. The distribution of means of the samples

Approximation Phase

Input: G the input graph, τ support threshold, S Candidate Subgraph, $maxS$ Maximum number of samples, $minS$ Minimum number of samples, $bSize$ sample size

Output: r the estimated support

```
D ← CreateDomains(G, S)
r ← 0
foreach D_i ∈ D do
    nValids ← 0; totalValids ← 0; nInvalids ← 0
    counter ← 0
    P_τ ← τ/|D_i|
    Reset distribution T
    while true do
        counter = counter + 1 u ← GetRandomNode(D_i)
        b ← IsValid(G,S,u,D_i)
        if b is true then
            nValids = nValids + 1
            totalValids = totalValids + 1
        else nInvalids = nInvalids + 1
        if counter (mod bSize)=0 then
            m ← ComputeMean(nValids,nInvalids)
            Add m to T
            if counter ≥ minS then
                M ← ComputeMean(T)
                SD ← ComputeSD(T)
                if FinishSampling(T,τ,maxS) then break
            nValids ← 0
            nInvalids ← 0
    estimatedSize ← (totalValids/counter) ∗ |D_i|
    if estimatedSize < r then r ← estimatedSize
```

Approximation Phase

Also collect useful statistics

- Estimates support of subgraph
- Number of valid nodes per MNI_{col}
- Expected invalid columns
- Subgraph evaluation time

$$\sum_{D_i \in D} \frac{time(D_i) * |D_i|}{N_i}$$

Exact Phase

Master-Worker paradigm

- Master keeps track of task pool, task dispatch and synchronization
- MPI for communication

Keep two task pools

- **Approximation pool (P_{APP})** from the approximation phase
- **Exact pool (P_{EX})** for the normal FSM algorithm

Exact Phase

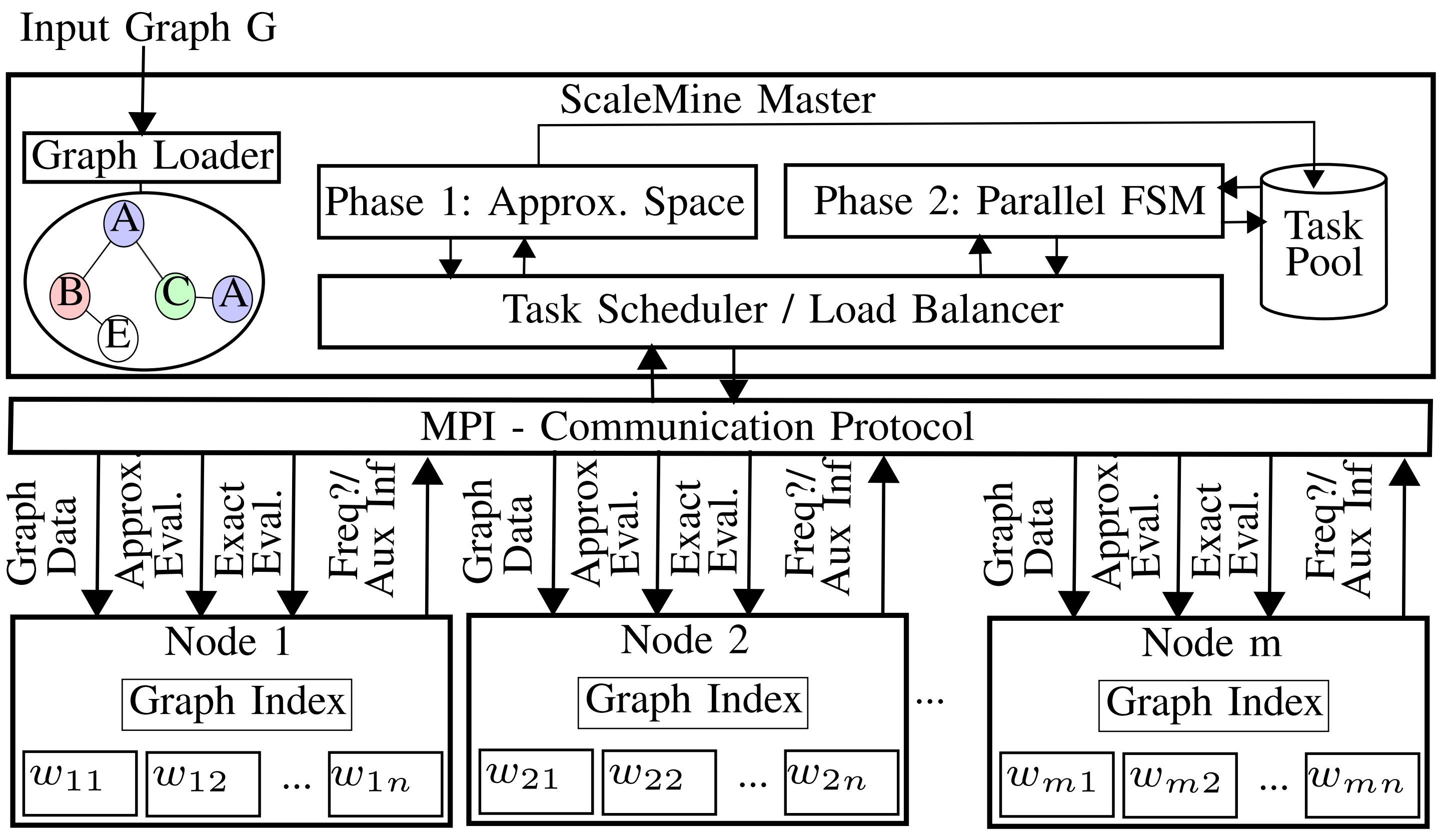


Fig. 4. ScaleMine System Architecture

Exact Phase – Load Balancing

FSM often runs out of work in its exact pool in the beginning and at the end

- Results in load imbalance

When out of work, dispatch tasks from P_{APP}

- These are high likelihood of frequent subgraph tasks
- Minimizes wasted work

Exact Phase – Load Balancing

FSM often runs out of work in its exact pool in the beginning and at the end

- Results in load imbalance

When out of work, dispatch tasks from P_{APP}

- These are high likelihood of frequent subgraph tasks
- Minimizes wasted work

Exact Phase – Subtasking

Use estimated evaluation time to partition long-running tasks

- Vertical or Horizontal

Manage imbalance caused by partitioning based on predicted workload distribution

$$\lambda = \frac{L_{max}}{\hat{L}} - 1$$

Exact Phase - Pruning

Preemptively determing invalid subgrahs

- Know a column does not have sufficient support if number of valid nodes + number of remaining nodes is less than τ
- Can also be used for subtasks

Prune large, expensive nodes by delaying their computation until necessary

Evaluation

Evaluated on 4 graphs

Comparison with prior work

- GraMi (single-threaded)
- Arabesque (distributed)

Evaluated on a cluster of 16 machines

Evaluation

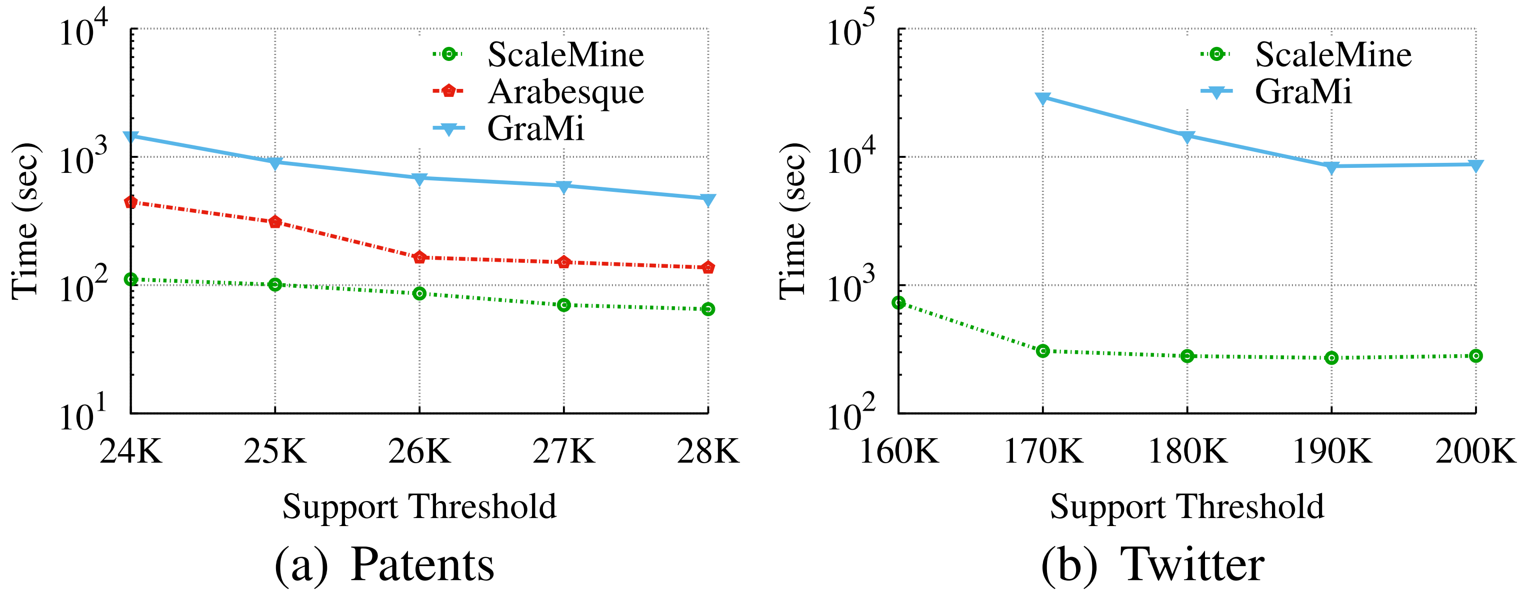


Fig. 5. Performance of ScaleMine vs. existing FSM systems on a cluster of 16 machines (256 workers) using two datasets: (a) Patents and (b) Twitter

Evaluation

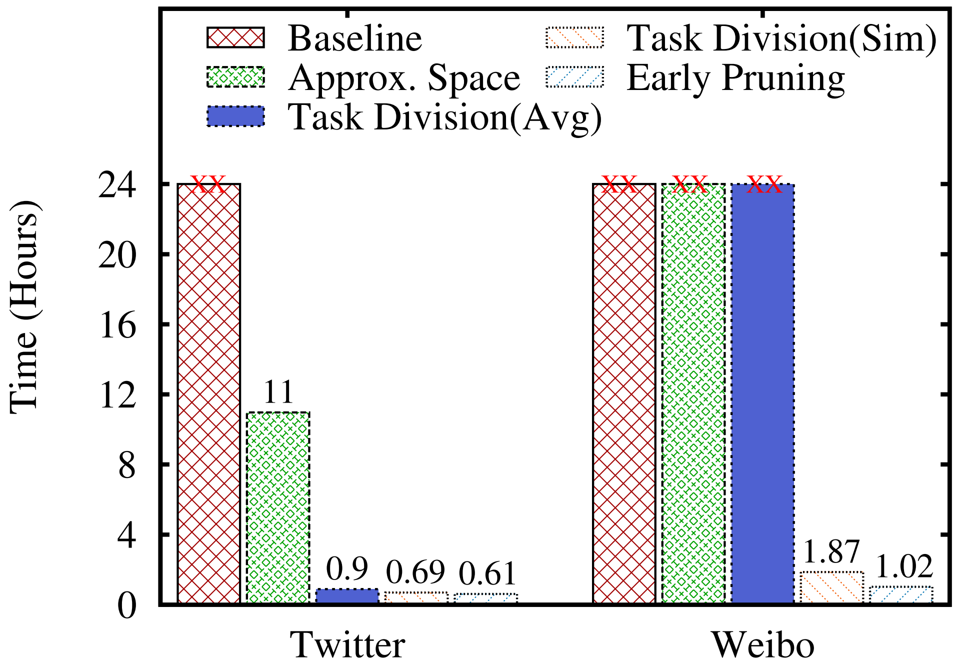
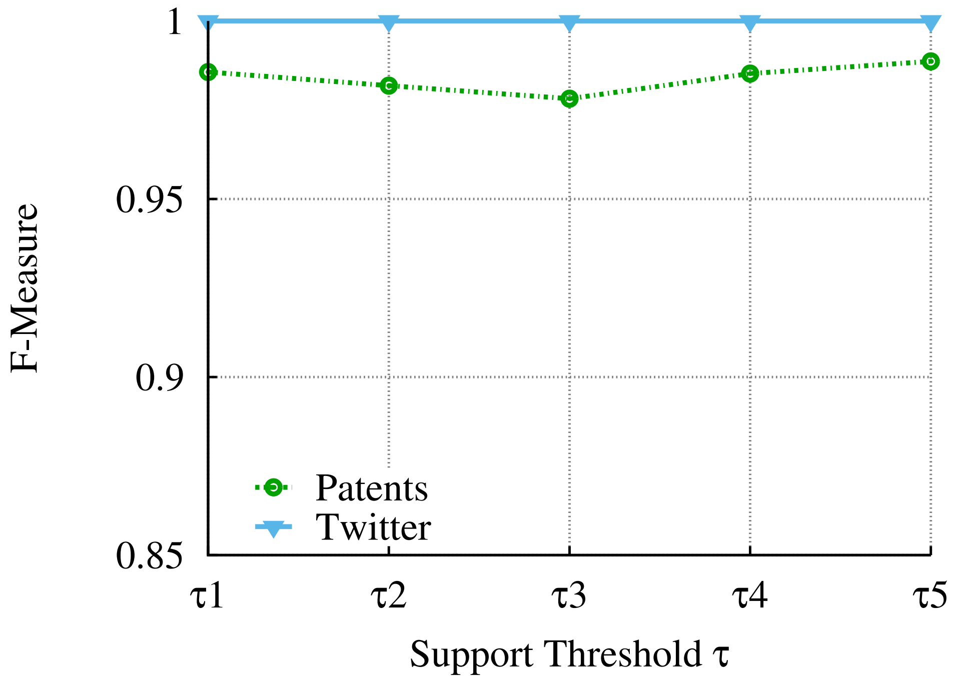


Fig. 6. Effect of ScaleMine's optimizations using Shaheen II with 512 cores on both Twitter ($\tau = 155k$) and Weibo ($\tau = 490k$, maximum size = 5 edges)

Evaluation

Approximation Phase retains high accuracy



Evaluation

Approximation Phase is cheap!

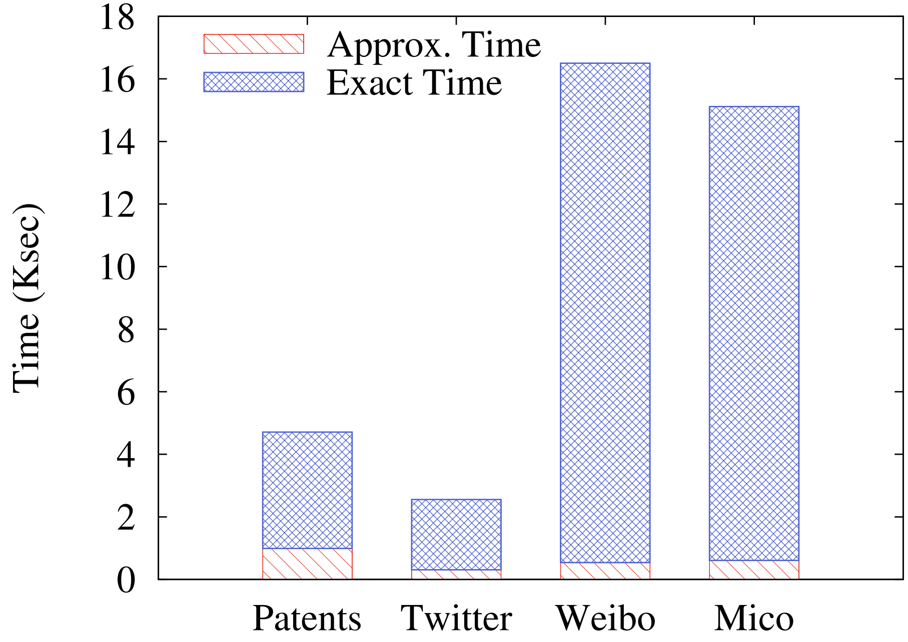
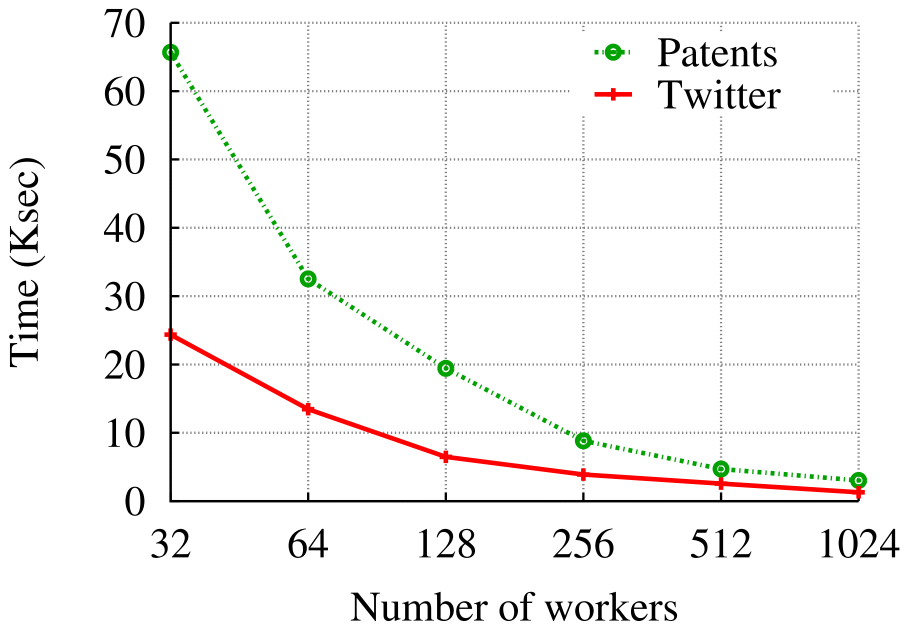


Fig. 8. Approximation phase time w.r.t the exact time

Evaluation

ScaleMine is scalable



(a) Scalability: Twitter and Patents

Limitations

How much wasted work is there from added communication/synchronization overheads of subtasking?

Priority within a pool?

Some key terms not explained (F-score? Which values of τ used?

Conclusion

Prior subgraph mining systems do not scale well

- Single-thread: Insufficient for large graphs
- Distributed: Suffer from synchronization overheads and load imbalance

SclaeMine uses a novel 2-phase technique to provide scalable subgraph mining

- Approximation phase for finding useful work quickly
- Pruning to remove invalid subgraphs early.