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
- Dividing up subgraph determination task has high communication and synchronization overheads.
Background and Motivation

![Graph showing scalability of Baseline and Task Division](image)

- **Ideal Scalability**
- **Task Division**
- **Baseline**

### Problem Description

- Tasks are divided into subtasks for load balance and efficient space utilization.
- The master builds a cost model for tasks.
- Infrequent subgraphs are pruned to improve scalability.
- Additional subgraphs are filled from the approximate phase.
- Workers execute low-cost plans.

### Approach

- **Two-Phase System**: Approximate and exact phases.
- **ScaleMine** introduces a scalable, hybrid approach.
- The master manages a pool of tasks.
- Workers execute tasks with load balancing.

### Contributions

- Scalable parallel system for exact FSM in large graphs.
- Efficient task distribution and execution plans.
- Improved scalability and performance compared to baseline and TaskDivision.
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) $> \tau$

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

Find number of distinct matches for each vertex $v_i$

- Create an $\text{MNI}_{\text{table}}$, where each column ($\text{MNI}_{\text{col}}$) consists of matches for the vertex (called *valid nodes*).
- The number of entries in *all* columns $> \tau$ -> valid subgraph.
MNI Metric

The following optimizations that significantly improve the other embeddings. To support large graphs, GraMi employs a set of embeddings that are sufficient to satisfy the need without maintaining many embeddings. GraMi models the subgraphs in the embeddings, which is less than maintaining many embeddings. GraMi [11] proposed an alternative approach that does not store embeddings unnecessarily. FSM algorithms store embeddings of the previously evaluated subgraphs in order to utilize them in subsequent iterations.

The search space is not known in advance; it is built through a series of evaluation/extension iterations. Each iteration populates the sets: \( S \) for a candidate subgraph \( G \). These statistics are later utilized to optimize the exact FSM phase. These statistics are later utilized to optimize the exact FSM phase.

ScaleMine collects useful statistics for each candidate subgraph \( G \) and accurately estimates the size of each column. Given an approach randomly samples a small fraction of these nodes, it is expensive. Our adaptive sampling approach (Section III-A) to estimate quickly whether a subgraph is frequent. During this phase, ScaleMine introduces a novel approximation phase, based on sampling, that satisfies the aforementioned requirements.

Load balance is essential for scalability. Achieving good load balance is easier when the search space is known in advance; unfortunately, this is not the case for FSM. We tackle the load balancing problem by a novel two-phase approach. The approximate phase resembles the typical FSM algorithm. It begins by finding small frequent subgraphs, which are then extended to larger ones by adding edges. We employ an approximate phase to estimate quickly over the load balance problem by a novel two-phase approach. For a candidate subgraph \( G \), the prediction is representative: the predicted search space should represent the exact search space within acceptable accuracy; (ii) informative: the approximation should be accurate; (iii) prioritization: light-weight node evaluations and expensive ones should be postponed.

For a candidate subgraph \( G \), the prediction is representative: the predicted search space should represent the exact search space within acceptable accuracy; (ii) informative: the approximation should be accurate; (iii) prioritization: light-weight node evaluations and expensive ones should be postponed.
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 $N_i$...

- $MNI_{col}(v_i) = N_i p_i$
- 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

$$\hat{\mu} = n\hat{p} \quad \hat{\sigma} = \frac{\sigma}{\sqrt{n}}$$
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, $\tau$ support threshold, $S$ Candidate Subgraph, $maxS$ Maximum number of samples, $minS$ Minimum number of samples, $bSize$ sample size

Output: $r$ the estimated support

1. $D \leftarrow$ CREATEDOMAINS($G, S$)
2. $r \leftarrow 0$
3. foreach $D_i \in D$ do
4.   $nValid\leftarrow 0$; $totalValid\leftarrow 0$; $nInvalid\leftarrow 0$
5.   counter $\leftarrow 0$
6.   $P_\tau \leftarrow \tau/|D_i|$
7.   Reset distribution $T$
8.   while true do
9.     counter $= counter + 1$ $u \leftarrow$ GETRANDOMNODE($D_i$)
10.    $b \leftarrow$ ISVALID($G, S, u, D_i$)
11.    if $b$ is true then
12.       $nValid\leftarrow nValid\leftarrow 1$
13.       $totalValid\leftarrow totalValid\leftarrow 1$
14.    else $nInvalid\leftarrow nInvalid\leftarrow 1$
15.    if counter (mod bSize)==0 then
16.       $m \leftarrow$ COMPUTEMEAN($nValid, nInvalid$)
17.       Add $m$ to $T$
18.       if $counter \geq minS$ then
19.          $M \leftarrow$ COMPUTEMEAN($T$)
20.          $SD \leftarrow$ COMPUTESD($T$)
21.          if FINISHSAMPLING($T, \tau, maxS$) then break
22.       $nValid\leftarrow 0$
23.       $nInvalid\leftarrow 0$
24.     estimatedSize $\leftarrow (totalValid/counter) \times |D_i|$
25.     if estimatedSize $< r$ then $r \leftarrow$ estimatedSize
Approximation Phase

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

\[
\sum_{D_i \in D} \frac{\text{time}(D_i) \times |D_i|}{N_i}
\]
Exact Phase

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

Keep two task pools
- Approximation pool \((P_{\text{APP}})\) from the approximation phase
- Exact pool \((P_{\text{EX}})\) for the normal FSM algorithm
As such, they are expected to be evaluated in future iterations. Available resources. To avoid having idle workers, ScaleMine small at the beginning and by the end of the evaluation process.

Evaluation of the subgraph. Dispatched tasks are prioritized by corresponding worker. If such statistics are not available for a becomes empty. Statistics of each subgraph are also sent to the dispatching tasks from frequent subgraphs are added to the result set, expanded, and tasks consisting of frequent vertices. Once they are evaluated, from exact evaluation. This phase starts by generating a set of approximations). The second phase handles the exact evaluation both frequent and infrequent predicted subgraphs (i.e., ap-

Due to the nature of FSM, the number of available tasks is loaded, ScaleMine starts its two-phase processing for finding which is utilized by the workers (i.e., threads) running on that loader. Each node has a single copy of the graph index, loaded and dispatched to nodes in the cluster by the graph显现. The exact phase utilizes this information to optimize the execution plan. Note that we only store the invalid columns for the approximated infrequent subgraphs only. The execution plan. Note that we only store this information for approximated infrequent subgraphs because it expected invalid columns.

Expected invalid columns

The master receives the input graph and the user-defined support threshold shows the system architecture. The master prioritizes tasks whenever Graph Loader

Graph Index

Node 1

Graph Data

Node 2

Node m

Graph Index

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 $P_{\text{APP}}$
◦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 $P_{\text{APP}}$

- 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

\[ \lambda = \frac{L_{\text{max}}}{\hat{L}} - 1 \]
Exact Phase - Pruning

Preemptively determining invalid subgraphs

- Know a column does not have sufficient support if the number of valid nodes + number of remaining nodes is less than $\tau$
- 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

![F-Measure graph against Support Threshold τ]

- F-Measure against Support Threshold τ for Patents and Twitter datasets.
- The graph shows F-Measure values ranging from 0.85 to 1.0.
- For Patents, F-Measure remains consistently high across τ1 to τ5.
- For Twitter, F-Measure is slightly lower but still maintains a high accuracy.
- The chart indicates that the approximation phase retains high accuracy regardless of the support threshold.
Evaluation

Approximation Phase is cheap!

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

ScaleMine is scalable

(a) Scalamility: 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 $\tau$ 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.