Expert Systems With Applications 106 (2018) 217–232

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

# Ap-FSM: A parallel algorithm for approximate frequent subgraph mining using Pregel

Vandana Bhatia*, Rinkle Rani

*Department of Computer Science and Engineering, Thapar University, Patiala, India*

ARTICLE INFO

*Article history:*
Received 30 August 2017
Revised 6 April 2018
Accepted 7 April 2018
Available online 9 April 2018

*Keywords:*
Pattern mining
Frequent subgraphs
Graph isomorphism
Large graphs
Sampling

ABSTRACT

Large graphs are scale-free, ubiquitous having irregular relationships and non-trivial topology. Frequent subgraph mining is a popular method for knowledge extraction from graphs. Most of the existing frequent subgraph mining algorithms are centralized algorithms that cannot handle a single large graph efficiently and incur high communication cost. However, to make the task of subgraph mining less expensive computationally, approximate subgraph mining can be applied which will capture similar structure subgraphs as of exact subgraph mining. In this paper, we propose an approximate subgraph mining algorithm named Ap-FSM implemented on distributed graph environment Pregel. The working of Ap-FSM is divided into three phases. The first phase selects the representative graph from the original graph while preserving the original graph properties. The second phase efficiently performs subgraph extension. Phase 3 introduces a novel two-step optimization for performing subgraph pruning. Analyzing such large graph data will be beneficial from the perspective of expert and intelligent systems, as discovered patterns can be used for knowledge discovery and decision making. To evaluate the performance of Ap-FSM, experiments are performed over three real life datasets having up to billion edges. The results show that the proposed Ap-FSM significantly outperforms the state-of-art frequent subgraph mining algorithms and overcome the challenges of performing frequent subgraph mining on a massive large graph. It is also shown that Ap-FSM achieves high scalability and speedup in distributed graph environment and is highly accurate in finding frequent subgraphs from a single large graph.

© 2018 Elsevier Ltd. All rights reserved.

## 1. Introduction

Big data has a good potential to transfigure most of the aspects of society. But, harvesting the valuable knowledge from the huge amount of data originated from various sources is still a remarkable challenge in the field of knowledge discovery and data mining (Aggarwal & Wang, 2010). Representing data as graphs to understand the relationships among data objects is gaining high interest. But, with the increase in the volume of data, the size of the graph is also increasing. The researchers are showing a lot of interest in applying graph analytics on data available from social media (Bonato & Tian, 2013; Halder & Lee, 2017; Leung, Hui, Liò, & Crowcroft, 2009), biological networks (Hill, Srichandan, & Sunderraman, 2012; Karci, 2009), business networks (Wang, Wang, Zhao, & Tan, 2015) etc. Frequent subgraph mining (FSM) is a popular method in the field of graph analytics.

FSM can be either performed on graph transactions or on a single large graph. Most of the existing works focus on graph transactions for performing FSM. But, finding frequent subgraphs from a single massive graph is far more challenging because of high computations cost for performing NP-Complete subgraph isomorphism in a single large graph. In many applications, getting the faster approximate results is significant than getting the exact results. If that large graph is analyzed at small scale, the useful insights can be extracted in lesser time. Analyzing large graph at small scale, while capturing the original graph properties is convenient and relatively easy task for analysis. However, the sample of large graph can be still enough large that could not be efficiently handled by centralized system. Traditional FSM algorithms are centralized and cannot handle such massive large graph. Also, creating a representative but a small sample from a massive large graph is a perplexing task.

Large graphs are handled in distributed environment for efficient processing but applying FSM in distributed environment is challenging due to variation in support values among various computing nodes. In addition, the real-world graphs are dense in nature and traditional FSM algorithms fails in handling dense graphs. Thus, in order to overcome the drawbacks faced

* Corresponding author.
*E-mail addresses:* vbhatia91@gmail.com (V. Bhatia), raggarwal@thapar.edu (R. Rani).

https://doi.org/10.1016/j.eswa.2018.04.010
0957-4174/© 2018 Elsevier Ltd. All rights reserved.

by solely parallelizing the existing traditional frequent subgraph mining approaches, we propose a novel approach called Ap-FSM (Approximate Frequent Subgraph Mining), to iteratively extract frequent subgraphs in a distributed environment. Ap-FSM performs sampling on original single massive graph to select the representative graph for making the task of frequent subgraph mining computationally less expensive in distributed environment. Ap-FSM is designed and implemented on parallel graph processing framework Pregel. Pregel is proficient in dealing with very large graphs and efficiently implementing parallel graph algorithms.

To our knowledge, Ap-FSM is the first algorithm to exploit the aggregation of graph sampling and parallelization for the task of FSM. A number of existing research explored either distributed algorithms (Bhatia & Rani, 2017; Jain, Flick, Pan, Green, & Aluru, 2017) or sampling (Mitzenmacher, Peng, & Xu, 2015; Saha & Al Hasan, 2014; Zou & Holder, 2010) for handling large graph. But, both of the aforementioned approaches have been somewhat orthogonal till now. In Ap-FSM, the shortcoming of either approach is compensated by the advantages of the other. To minimize communication overhead, in-memory computations are performed. The distributed graph data are mined in parallel and their results are collected afterwards.

In summary, following contributions are made by the proposed approach

- An efficient parallel algorithm for approximate frequent subgraph mining on a single large graph.
- An efficient technique named G-Samp for the selection of an approximate subgraph while capturing the original graph properties for convenient and relatively easy analysis.
- A two-step approach for optimizing the subgraph isomorphism task and subgraph pruning.

The rest of the paper is structured as follows. Section 2 presents the literature review of the existing research in the area of frequent subgraph mining. The preliminaries and problem definition are given in Section 3. Section 3 also detail out the Pregel framework. The proposed approach is explained in Section 4. Section 5 discusses the performance evaluations, and the conclusion of the paper is given in Section 6.

## 2. Related work

Frequent subgraph mining has become an imperative area of research because of wide variety of graph mining applications including chemical data analysis (Yan & Han, 2002), drug discovery, communication networks (Hsieh & Li, 2010), computational biology (Chen et al., 2017), Web services (Liu, Zhu, Chu, & Ma, 2018), and many others (Pears, Pisalpanus, & Koh, 2015).

The frequent subgraph mining algorithms can be classified as apriori-based and pattern-growth based approaches. In apriori based approaches, size-($k$+1) subgraphs are generated from the set of frequent $k$ subgraphs. Here, breadth first search strategy is preferred because graph is represented in level-wise manner. Many apriori-based algorithms for mining frequent subgraphs have been proposed in literature (Kuramochi & Karypis, 2001; Kuramochi & Karypis, 2004b; Nijssen & Kok, 2001). Kuramochi and Karypis (Kuramochi & Karypis, 2004a) proposed an apriori based algorithm FSG which uses an edge-based method to grow the size of subgraph by single edge during each iteration. In FSG, the number of edges characterizes the size of graph. FSG was further modified to find frequent patterns in geometric graphs in 2007 (Kuramochi & Karypis, 2007). Although apriori-based algorithms find the frequent subgraphs efficiently in small graphs, they have considerable overheads in joining two subgraphs of size-$k$ to generate candidates of size-($k$+1). To overcome these overheads, non-apriori based algorithms were introduced, which were basically based on pattern-growth approach.

In Pattern growth approach, a new edge $e$ is added to the subgraph for generating a new candidate subgraph $g$. By repeating this procedure, extension of frequent subgraphs is performed iteratively in all possible manners. But, during the generation of frequent subgraph it may possible that same subgraph is generated many times (Cook & Holder, 2007). This problem resulted in design of many other algorithms such as FFSM (Huan, Wang, & Prins, 2003), MOFA (Borgelt & Berthold, 2002), gSpan (Yan & Han, 2002), Gaston (Nijssen & Kok, 2004) and SUBDUE (Ketkar, Holder, & Cook, 2005).

CloseGraph (Yan & Han, 2003) introduced by Yan and Han in 2003 is a closed graph mining algorithm that uses pattern growth approach. It eliminates the generation of the unnecessary subgraphs and increases the mining efficiency up to greater extent, specifically in the case of large patterns. Furthermore, certain algorithms were proposed in literature for handling uncertain graphs also (Wu, Ren, & Sheng, 2017; Zou, Li, Gao, & Zhang, 2010).

Some work have also been done to find the maximal frequent patterns in graph (Huan, Wang, Prins, & Yang, 2004; Ozaki & Etoh, 2011; Thomas, Valluri, & Karlapalem, 2010; Wu et al., 2017). Most of them have used the concept of edit distance for allowing structural differences (Flores-Garrido, Carrasco-Ochoa, & Martínez-Trinidad, 2014a, b; Moussaoui & Zaghdoud, 2016). An algorithm named APGM (Jia, Zhang, & Huan, 2011) is proposed to find inexact patterns by using a pre-defined substitution matrix consisting of similarities among labels. Similarity is measured by a product of probabilities. APGM is extended to allow label replacements of both vertices and edges in VEAM (Acosta-Mendoza, Gago-Alonso, & Medina-Pagola, 2012). However, both APGM and VEAM require prior knowledge in the form of substitution matrix which is available in fewer applications only. Some other algorithms such as MgVEAM (Acosta-Mendoza, Gago-Alonso, Medina-Pagola, Jesús Ariel Carrasco-Ochoa, & Martínez-Trinidad, 2017) and AMgMiner (Acosta-Mendoza, Gago-Alonso, Carrasco-Ochoa, Martínez-Trinidad, & Medina-Pagola, 2016) are designed for dealing with multi-graphs. Both MgVEAM and AMgMiner are also used for image classification.

We intend to find frequent subgraphs from a single large graph. For a single graph, only few approaches for approximate subgraph mining algorithms were proposed in literature (Chen, Yan, Zhu, & Han, 2007; Flores-Garrido et al., 2014a, b). Most of the algorithms work for only small graph datasets and thus, face scalability issue. For finding the patterns from large graph datasets in distributed computing system, Map-Reduce was used by some researchers (Bhuiyan & Al Hasan, 2013; Lin, Xiao, & Ghinita, 2014; Lu, Chen, Tung, & Zhao, 2013). However, they find frequent subgraphs from graph transactions. Only few research have been conducted for mining frequent subgraphs from a single graph (Bringmann & Nijssen, 2008; Flores-Garrido et al., 2014a, b).

Also, most of the existing research considers only sparse graphs for mining (Kuramochi & Karypis, 2005; Reinhardt & Karypis, 2007). But, the real world graphs are dense and have high average degree. The first algorithm which finds the frequent subgraphs from large dense subgraph is GRAMI (Elseidy, Abdelhamid, Skiadopoulos, & Kalnis, 2014). It uses constraint satisfaction problem for finding the minimum set of instances for satisfying support threshold. AGRAMI is proposed by extending GRAMI to find approximate subgraphs. However, both of them are centralized algorithms that cannot handle very large graph. For handling the large graph in distributed environment, many subgraph mining algorithms using Map-Reduce were proposed (Babu, 2016; Wang et al., 2016). A density based partitioning technique was proposed for large scale subgraph mining using Map-Reduce framework for balancing computational load (Aridhi, D'Orazio, Mad-

**Table 1**
Characteristics of subgraph mining algorithms.

| Algorithm | Nature of input | Nature of output | Max. graph size | | Mode of processing | Advantages | Limitations |
|---|---|---|---|---|---|---|---|
| **For set of graphs** | | | **Number of graphs** | | | | |
| REAFUM | Set of Graphs | Approximate | 340 graphs | | Centralized | Allow both structural and label approximations | Handles small graph datasets only |
| FSM-H | Set of Graphs | Exact | 100,000 graphs | | Distributed | Handle large distributed graph datasets | Suffers from I/O overhead |
| MRFSE | Set of Graphs | Exact | 1,000,000 graphs | | Distributed | Isomorphism-testing-free approach is used | Suffers from I/O overhead |
| MUSE | Set of Graphs | Uncertain Graphs | 6 graphs | | Centralized | Uses the concept of error tolerance | Works only for small graph datasets |
| APGM | Set of Graphs | Approximate | 10,000 graphs | | Centralized | Handles noise and distortion | Prior-domain knowledge of dataset is required |
| VEAM | Set of Graphs | Approximate | 700 images | | Centralized | Allow approximation over edge label set | Prior-domain knowledge of dataset is required |
| wgMiner | Set of Graphs | Approximate | 13,036 graphs | | Centralized | Discover all closed and maximal patterns | Considers only upper bound of interestingness measure |
| UMFGAMW | Set of Graphs | Approximate | 10,000 graphs | | Centralized | Standard coding of uncertain graph adjacency matrix | Produces large number of uncertain graphs |
| MgVEAM | Set of Graphs | Approximate | 1000 images | | Centralized | Do not require any preprocessing on multi-graphs | Does not support node scalability |
| **For single graph** | | | **\|V\|** | **\|E\|** | | | |
| AGrAP | Single Graph | Approximate | 200 | – | Centralized | Considers structural differences | Not Scalable |
| MaxAFG | Single Graph | Approximate | 35 | – | Centralized | Considers structural differences in patterns | Centralized algorithm with higher time complexity |
| SSIGRAM | Single Graph | Exact | 11,316,811 | 85,331,846 | Distributed | Employs optimization and load balancing | Cannot handle vertices of high degree |
| Arabesque | Single Graph | Exact | 4,589,876 | 43,968,798 | Distributed | Handles large graph efficiently | Performs costly graph isomorphism and pruning |
| GRAMI | Single Graph | Exact | 11,316,811 | 85,331,846 | Centralized | Performs optimizations for graph isomorphism and pruning | Not Scalable |
| ScaleMine | Single Graph | Exact | 56,692,400 | 1,046,834,000 | Distributed | Performs load balancing for parallel computations | Cannot handle graphs larger than the capacity of a single machine |
| pegi | Single Graph | Exact | 4,847,571 | 68,993,773 | Distributed | Less distribution and communication overhead | Graph isomorphism optimization is not considered |
| gApprox | Single Graph | Approximate | 2637 | 4030 | Centralized | Anti-monotonic constraint is utilized | Not Scalable |
| Proposed Ap-FSM | Single Graph | Approximate | 41,652,230 | 1,468,365,182 | Distributed | Scalable and performs efficient optimizations for subgraph isomorphism and pruning | Some of the important subgraphs may not be identified |

douri, & Mephu Nguifo, 2015). Other subgraph mining algorithms for single large graphs are ScaleMine (Abdelhamid, Abdelaziz, Kalnis, Khayyat, & Jamour, 2016), FSM-H (Bhuiyan & Al Hasan, 2013), Arabesque (Teixeira et al., 2015), MRFSE (Lu et al., 2013), gApprox (Chen et al., 2007), SSIGRAM (Qiao et al., 2018). However, all the aforementioned research performs expensive computation.

To reduce the computational cost, FSM is applied on the representative graph sample. Many graph sampling algorithms were proposed in literature (Edwards, Johnson, Milosavljevic, Gadepally, & Miller, 2015; Mozafari et al., 2015; Ribeiro & Towsley, 2010; Voudigari, Salamanos, Papageorgiou, & Yannakoudakis, 2016). However none of them were applied for the task of subgraph mining and none of the existing FSM algorithm used sampling to lower down the computational cost. In the proposed algorithm Ap-FSM sampling is used along with two-step optimization for cutting the cost of subgraph isomorphism and subgraph pruning. The summary of the characteristics of the proposed algorithm along with the existing algorithms are given in Table 1.

## 3. Preliminaries and background

### 3.1. Preliminaries

A Graph $G$ can be defined as set $\{V, E\}$, where $|V|$ is a set of vertices and $|E| \in [V \text{ x } V]$ is set of edges. Graph can be labeled or unlabeled. A labeled graph can be defined as:

**Definition 1. (Labeled Graph)** A labeled Graph $G_L$ with node label $L_V$ and an edge label $L_E$ is a tuple of four elements $G=(V, E, \mu, \varepsilon)$, where $\mu$ and $\varepsilon$ are the vertex and edge labeling functions respectively such that $\mu{:}V\rightarrow L_V$ and $\varepsilon{:}E\rightarrow L_E$.

In certain applications, a large graph may contain a set of smaller graphs. Or, for an instance, a graph $G=(g_1, g_2, \ldots g_n)$, where the graph $G$ is an aggregation of smaller subgraphs $(g_1, g_2, \ldots g_n)$. More formally:

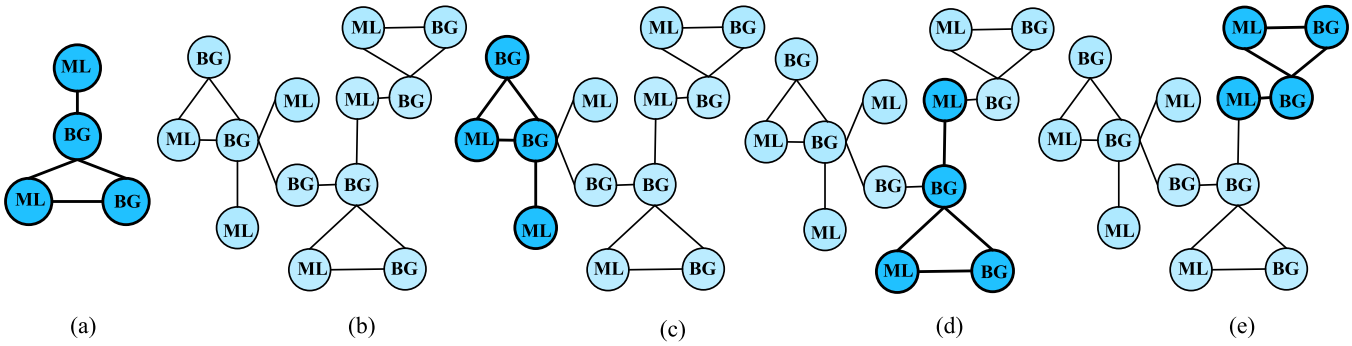


**Fig. 1.** Graph embeddings: (a) subgraph (b) single graph as input (c) Subgraph_Embedding_1 (d) Subgraph _Embedding_2 (e) Subgraph _Embedding_3.

**Definition 2.** (**Subgraph**) A labeled graph $g_l = (v, e, \mu_g, \varepsilon_g)$ is a subgraph of another labeled graph $G_L(V, E, \mu, \varepsilon)$ if the vertices $v \subseteq V$, edges $e \subseteq (E \cap (v \times v))$, $\mu_g(p) = \mu(p)$ for all $p \in v$, and $\varepsilon_g(p,q) = \varepsilon\ (p,q)$ for all $(p,q) \in e$.

On the contrary, graph $G$ is called a supergraph of $g$, if $g$ is subgraph of $G$.

A graph is frequent if the *support*($g$) is no less than $\tau$. Here, $\tau$ is the minimum support. For matching two similar subgraphs, subgraph isomorphism is computed which map every node of subgraph $g_1$ to $g_2$ such that vertex and edge labels are consistent in a way that edge structure can be preserved.

**Definition 3.** (**Subgraph Isomorphism**) Two Subgraphs $g_1 = (V_1, E_1, \mu_1, \varepsilon_1)$ and $g_2 = (V_2, E_2, \mu_2, \varepsilon_2)$ are isomorphic if they have identical topology and there exist a mapping $f_v$: $V_1 \rightarrow V_2$ such that $\mu_1(p) = \mu_2(f(p))$ for all $p \in V_1$ and for all edges $e_1(p,q) \in E_1$, there exist an edge $e_2\ (f\ (p), f(q)) \in E_2$ and $\varepsilon_1(p,q) = \varepsilon(f(p), f(q))$.

**Definition 4.** (**Graph Embedding**) Given a subgraph $g$ of graph $G$, suppose $g \subseteq G$ such that their exist a mapping $f_v$: $V_g \rightarrow V_G$. An embedding $\phi$ is a pattern of vertices $V_G$ that are mapped to the vertices $V_g$ such that$m = [f(v_1), \ldots.., f(v_i), \ldots., f(v_{|V_g|})]$, where $v_i \in V_g$ and $f(v_i) \in V_G$.

**Example 1.** Consider the graphs in Fig. 1, three possible embeddings in Fig. 1(c)–(e) of the subgraph in Fig. 1(a) of the input graph in Fig. 1(b). Embeddings shown in Fig. 1(c) and (e) do not overlap, so are edge-disjoint, whereas the embedding shown in Fig. 1(d) share vertices and edges with the embedding shown in Fig. 1(e). Therefore if overlaps are allowed, the frequency of subgraph of Fig. 1(a) is 3, otherwise it is 2.

Overlapping of frequent subgraphs can occur in both vertex as well as edge. Various anti-monotone metrics for computing support based on overlap are introduced in literature. Kuramochi and Karypis (2005) introduced maximum independent set (*MIS*) for computing support of frequent embedding or occurrence of subgraph $g$ in a single graph setting by considering all possible embeddings $\phi_i$ of subgraph $g$ in graph $G$. Further, binary relationship namely harmful overlap (*HO*) was formalized. A harmful overlap (Fiedler & Borgelt, 2007) of embedding $\phi$ and $\phi'$ of subgraph $g$ exists if $\exists v \in V_g$: $\phi(v), \phi'(v) \in (V_g) \cap \phi'(V_g)$. However, the computations performed in both *MIS* and *HO* are *NP*-complete. For instance in Fig. 1, *MIS* gives support value of two while there are three embeddings of subgraph $g$.

In this paper, we are using minimum image based support (*MNI*) because it avoids expensive computations and provides the superset of results of *MIS* and *HO*. *MNI* (Bringmann & Nijssen, 2008) is based on the number of unique vertices of a graph $G$ to which a vertex of the subgraph $g$ is mapped. It can formally be defined as:

**Definition 5.** Let $g_1, \ldots.., g_m$ be the isomorphic graphs such that the set of embeddings $S(g) = \{g_1(v), \ldots.., g_m(v)\}$. The minimum image based support (*MNI*) $\sigma_i$ of subgraph $g$ in graph $G$ is defined as:

$$\sigma_i(g, G) = \min_{v \in V_g} |\{\varphi_i(v) : \varphi_i = g_i(v) \forall g_i \in S(g)\}| \tag{1}$$

### 3.2. Distributed Pregel framework

Pregel is a graph analytic environment designed for large graphs using Bulk Synchronous Parallel (BSP) algorithm. Google published a paper on Pregel in 2010 (Malewicz et al., 2010). It works in a distributed manner by dividing vertices of a large graph among the workers in a cluster and performing computations individually at each worker node. Communication among the workers occurs by passing messages which are generally small in comparison to data and thus are more efficient for transmission. The algorithm terminates when no messages are produced during iteration or until each worker node votes to halt.

BSP is a paradigm to design parallel algorithms with two basic operations: Communication via sending messages and barrier Synchronization. A BSP computation involves a sequence of supersteps as shown in Fig. 2. Messages from the previous superstep are available in the next superstep. In this paper, iterative BSP based graph processing framework Giraph ("Apache Giraph," 2016) is used to perform offline batch processing of semi-structured graph data. It performs iterative calculations on the top of the Hadoop cluster by avoiding costly disk and network operations mandatory in map-reduce framework by using out of core in-memory execution. The processing of Pregel based Giraph is shown in Fig. 1. In Giraph, only intermediate values in the form of messages are sent across the network. The disk access takes place only while fetching input from Hadoop Distributed file system (HDFS), storing output back in HDFS and for storing checkpoints. Each cycle of an iterative Giraph calculation on Hadoop runs a full map-reduce job with only mappers.

The input graph is partitioned between the worker nodes. Vertices are loaded from HDFS using a user specified input format. Workers call compute() function on the active vertices and collect the messages sent in previous superstep. During each subgraph, every vertex executes by re-computing its values and sends massages to all other vertices. The message transfer takes place over a superstep barrier. Workers then compute aggregators, collect statistics and wait at the synchronization barrier for more vertices and messages to be processed until all workers votes to halt. At last, vertices are offloaded to HDFS through an output format. Checkpoints are used to achieve fault tolerance. Worker saves the state of their partition in HDFS after getting the instruction from the master at the beginning of each superstep. Worker saves their vertex values, edge values and the values of incoming messages, which are then aggregated by master. To overcome communica-

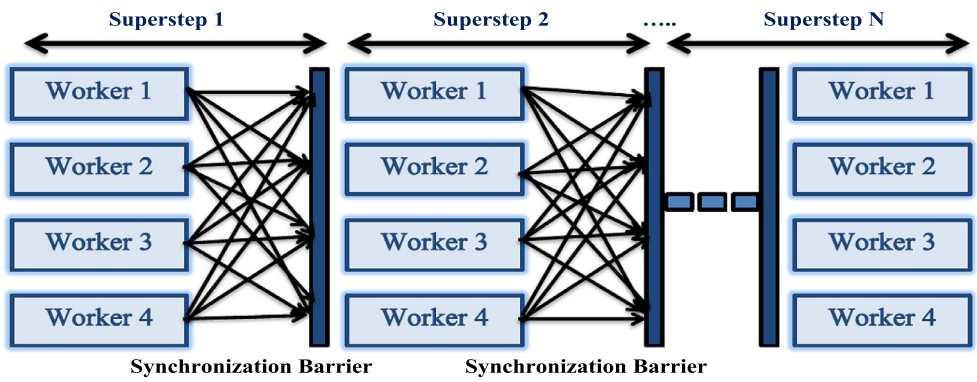


**Fig. 2.** Workflow of Pregel.

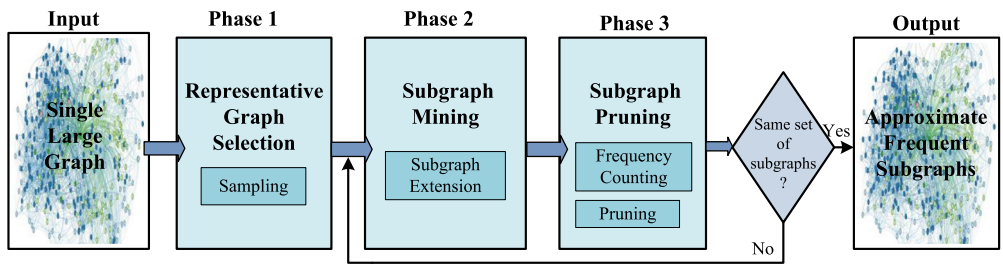


**Fig. 3.** Block diagram of the proposed approach.

tion overhead giraph functions such as Combiner and aggregators can be utilized (Martella, Logothetics, & Shaposhnik, 2015).

### 3.3. Problem definition

Given a graph sample $G_s$ consisting of labeled vertices $V_s \in V$ and a support threshold $\tau$, find the subgraphs $g_s$ which are at least $\sigma$ frequent in graph sample $G_s$

## 4. The proposed algorithm

In this section, the proposed Ap-FSM is presented. Ap-FSM discovers representative approximate frequent subgraphs from a single massive graph. The proposed approach employs a simple generation-and-verification method during each iteration on the top of Pregel. The verification is carried out in two steps in each iteration, first by Workers locally and later by Master node globally.

The block diagram of the proposed Ap-FSM is shown in Fig. 3. The working of Ap-FSM is divided in three phases: representative graph selection, subgraph extension and subgraph pruning. The first phase of representative graph selection phase is executed only once in the beginning whereas the second and third phases are iterative. In phase 1, AP-FSM selects a sample from the original graph to make the computations fast and easy. Second phase performs two steps locally on each Worker node. In the first step, all frequent single vertices are enumerated to form size-1 subgraphs. Then, the iterative computations are performed for subgraph extension. In each iteration, Ap-FSM generates all possible subgraphs which are larger in size by one vertex and edge from the subgraphs generated in previous iteration. Further, in the third phase, Ap-FSM sums the frequency of every candidate subgraph and prunes those subgraphs which are not frequent. Ap-FSM terminates when no extension is performed in a particular iteration. The workflow diagram of the proposed Ap-FSM is given in Fig. 4. In the succeeding subsections, the three phases of proposed Ap-FSM are described in detail.

### 4.1. Phase 1: representative graph selection

A novel graph sampling algorithm named G-Samp is proposed in this section in order to select an appropriate representative graph from a single large graph. Representative graph $G'$ is a sample of a given single large graph $G$ such that $G'$ holds somewhat similar graph properties of the original graph $G$. More formally:

**Definition 6.** The sampling of graph $G=(V,E)$ with sampling factor $\delta$ is to generate a subgraph $G'=(V',E')$ such that $|V'| = \delta|V|$ where, $V' \subseteq V$ and $E' \subseteq E$.

The real-world large graphs follow power-law-degree distribution, thus the sampled graph $G'$ for having similar characteristics should have similar degree distribution as of graph $G$. Thus, the algorithm starts with computing the degree of the vertices of graph $G$. All the vertices with their vertex id's and degree are stored in a list for further processing.

The vertex list is further sorted as per the descending order of their degree values. The vertices with the similar degree are aggregated and stored together in list $l_i$, where $i$ represent the degree of vertices in list. For example, vertices with degree 6 are stored in list $l_6$. The precise working of G-Samp is given in Algorithm 1. A pre-defined sampling factor $\delta$ is used to select the vertices from the graph for sampling. For example, for $\delta = 0.2$, 20% of the vertices are selected from each degree-wise aggregated vertex list $l_i$. From each vertex list $l_i$, $h$% vertices are selected randomly. The selected $h$% vertices from list $l_i$ are stored in a new list $l'_i$.

Vertices stored in lists $l'_i$ are aggregated and are stored in sampled graph $G'$ such that graph $G'$ will have the same number of vertices from each degree-wise aggregated vertex list $l_i$. Thus, the selected vertices will have same degree structure as of the original graph $G=(V,E)$ such that the selected representative graph $G'=(V',E')$ will also follow the similar but less dense power law degree distribution as followed by original graph $G=(V,E)$. These selected vertices set $V'$ with the edges incident on the vertices of $V'$ together form the sampled graph $G'$.

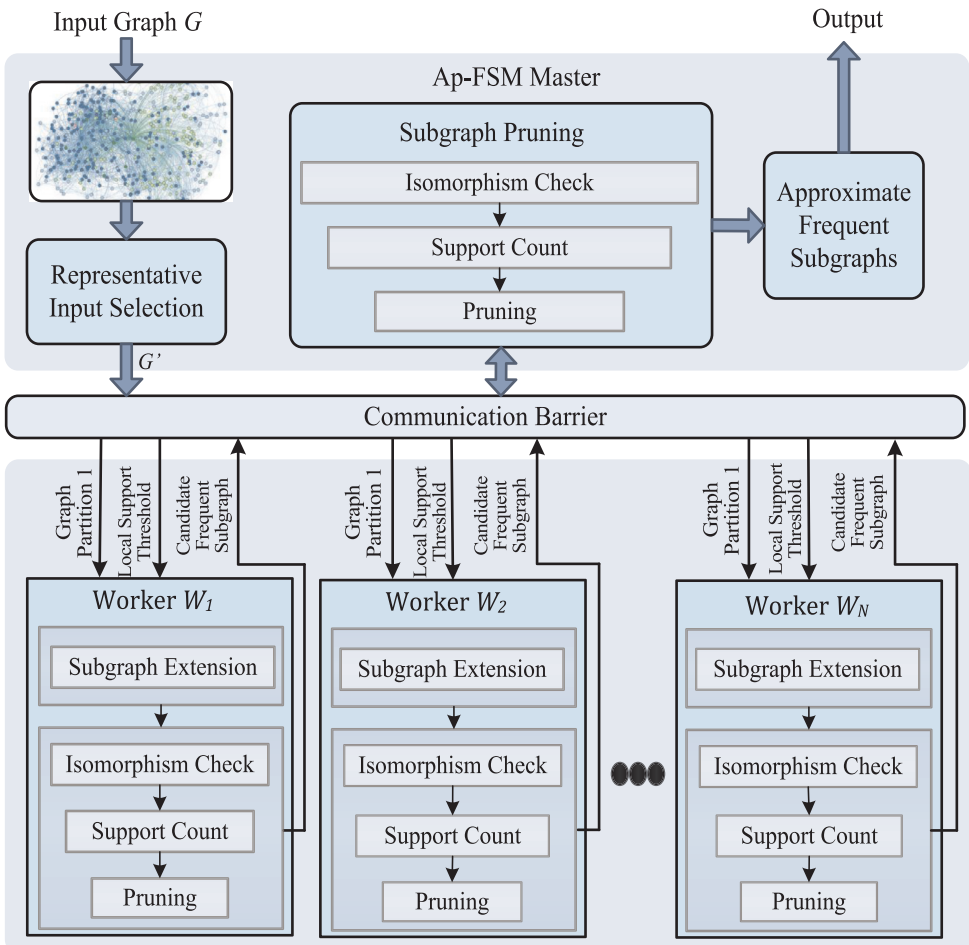


**Fig. 4.** Flow diagram of the proposed approach to extract frequent approximate subgraphs.

**Algorithm 1**: **Phase 1:** ***G-Samp*** **(*G*, *δ* )**

**Input:** A Graph $G$ ($V$,$E$) of size $N$, Sampling Rate $\delta$ ,
**Output**: A sampled graph $G'$=($V'$,$E'$) of size $x$

1. Rank $N$ vertices of graph based on the decreasing order of their degree $d_i$
2. Aggregate vertices with similar degree in list $l_i$.
3. $G' \leftarrow \emptyset$
4. $h = \delta * 100$
5. $x = h\% \ N$
6. **While** $|V'| \leq x$ **do**
7. | **For** each vertex list $l_i$ **do**
8. | | Select $h$% vertices
9. | **End**
10. | $l'_i \leftarrow$ Selected $h$% vertices
11. **End**
12. $G' \leftarrow \sum_{i=0} l'_i$

In this phase, the frequent subgraphs are identified in multiple iterations by performing one node and one edge extension in each iteration. All the computations of this phase are performed on the worker nodes as shown in Fig. 4. The main focus is on generating a superset $\mathcal{F}^g_{i+1}$ from subset $\mathcal{F}^g_i$ for all candidate subgraphs.

### 4.2. Phase 2: subgraph extension

The frequent subgraph discovery in Ap-FSM starts from this phase. The working of Ap-FSM for frequent subgraph discovery is summarized in Algorithm 2. The master and workers perform computations until the set of frequent subgraphs of desired size are discovered.

**Algorithm 2**: **Ap-FSM Subgraph Discovery**

```
Input: A Graph G' (V',E')
Output: Approximate frequent subgraph set of F of G'=(V',E')
While (k ≤ z) do
    Function VertexCompute(msg,superstep)
        If (Superstep=0) then
            F ← ∅
            F^1 ←Frequent size-1 subgraphs in G'
            S(F^1) ←All embeddings of F^1 in G'
        End
        Else
            S(F^k) ←All embeddings of F^k in G'
            Ext(F^k) ←Possible extensions of all the embeddings of F^k
            F^(k+1) ←F^(k+1) ∪ Ap-FSM-EXTEND(F^k,G',S(F^k))
            While (F^(k+1) ≠∅)
                F_Wi ←LocalPrunning(F^(k+1),d(g_i), τ')    //Using Algorithm 4
            End
        End
        (F^(k+1)) ← All distinct subgraphs images from F_Wi from all the Workers
    End
    Function MasterCompute(msg,superstep)
        While (I(F^(k+1)) ≠∅) do
            F ←GlobalPrunning(I(F^(k+1)), d(g_i), τ)        //Using Algorithm 5
        End
    End
End
```

**Algorithm 3:** Ap-FSM-EXTEND($F^k$,$G'$,$S(F^k)$)

```
F^(k+1) ←∅
For each embedding φ of subgraph s' in S(F^k) do
    For each c^(k+1) ∈ candidate subgraph set C^(k+1) do
        If F^k is the parent of C^(k+1) then
            For each possible extension ext(e,v) or ext(e) in Ext(g') do
                If ((g',e)→v) then
                    If (v is frequent in Ext(s')) then
                        C^(k+1) ← C^(k+1) ∪ ext(e,v)
                    End
                    Else if ((s',e)→v(s')) then
                        C^(k+1) ← C^(k+1) ∪ ext(e)
                    End
                End
            End
        End  F^(k+1) ← C^(k+1) ∪ F^(k+1)
    End
End
F^(k+1) ← All size-k+1 subgraphs
```

Phase 2 of Ap-FSM starts by finding all size-1 subgraphs and storing them to frequent subgraph set $F^1$. Further, each size-1 subgraph in $F^1$ is used as the initial point for performing a recursive depth-first-search (DFS) based extension to explore final size-$z$ subgraphs. The recursion terminates when all size-$z$ frequent subgraphs are discovered and there is no change in further iteration.

The extension is performed using Ap-FSM-EXTEND function given in Algorithm 3. Ap-FSM-EXTEND takes as input set $F^k$ of size-$k$ subgraphs and complete embeddings set $S(F^k)$ of $F^k$ in $G'$ and continues as follows. For each embedding $\varphi \in S(F^k)$ of size-$k$, the proposed Ap-FSM recognizes and stores each potential size-($k$+1) subgraph embedding's ($\varphi + (v,e)$).

**Algorithm 4**: ***LocalPruning*** $(F^{k+1}, d(g_i), \tau')$

**Input:** All candidate subgraphs $F^{k+1}$ at level $(k+1)$, Degree $d_i$ , Graph $G'$, Local frequency threshold $\tau'$
**Output:** Local Frequent subgraph set $F_{Wi}$

```
Function VertexCompute(msg,superstep)
  For each subgraph g_i ∈ F^{k+1} do
    | Compute d (g_i)
  End
  Sort all g_i ∈ F^{k+1} in descending order of their d (g_i)
  Agg(g_i) based on d (g_i) similarity
  For each g_i ∈ F^{k+1} do
    | If (d (g_i)= d (g_j)) then
    |   | If (DFS(g_i)= DFS(g_j)) then
    |   |   | Declare g_i and g_j isomorphic
    |   |   | Add g_i and g_j to C^{k+1}_{g_i}
    |   | End
    | End
  End
  F^{k+1} ← ∅
  For each C^{k+1}_{g_i} do
    | If (Count( C^{k+1}_{g_i} ) ≥ τ' ) then
    |   | Add all g_i ∈ C^{k+1}_{g_i} to F^{k+1}
    | End
  End
  I(F^{k+1}) ← A copy of all frequent subgraphs g_i ∈F^{k+1} , d (g_i) , Support (g_i)
```

Ap-FSM does that by exploring every possible extension $Ext(g')$ of each embedding $\varphi$ of size-$k$ subgraph $g'$, where $g' \in F^k$. The extension $Ext(g')$ contains all possible extensions $ext(e,v)$ of vertex $v$ and edge $e$ or extensions $ext(e)$ of only edge $e$ in the case of backtracking. The extension $e' \in Ext(g')$ which is frequent in all the embeddings of subgraph $g'$ is then stored in candidate subgraph set $C^{k+1}$ of size-$(k+1)$.

However, the multiple discovery of same vertex in depth-first tree rooted at that vertex should be avoided. It will be highly beneficial to avoid such extensions as each vertex at level $k$ is linked with up to $k$ different vertices at level $(k-1)$. Thus the algorithm will finish off in performing redundant computations which will adversely affect the overall performance.

The proposed Ap-FSM eliminates such redundant calculations by allocating each vertex at level $k$, a unique parent vertex at level $(k-1)$. Therefore, only subgraphs from $F^{k-1}$ are permitted to create subgraphs in $F^k$. The subgraph stored in $F^{k-1}$ is called as the parent of $F^k$. For each candidate subgraph $c^{k+1}$ from candidate subgraph set $C^{k+1}$, Ap-FSM keeps only those subgraphs in $C^{k+1}$ which have $F^k$ as parent.

The remaining subgraphs in $C^{k+1}$ are size $(k+1)$ frequent subgraphs which are obtained by performing a single edge extension $ext(e)$ or a single vertex and edge extension $ext(e,v)$ of $F^k$ and are forwarded as input for proceeding recursive call. The recursion terminates when the candidate size-$(k+1)$ subgraph set $C^{k+1} = 0$, and backtracking take place in depth-first search.

### 4.3. Phase 3: subgraph pruning

This phase involves three stages. In the first stage, an isomorphism test is performed on candidate subgraphs to determine their frequency. In the second and third stages, the support of the frequent subgraph is calculated. The candidate subgraphs having support lesser than the minimum support are discarded. Rest of the subgraphs are added to the list of frequent subgraphs.

The computations involved in the above three stages can be carried out efficiently over small graphs in a centralized system. But, in distributed environment where the large graph is partitioned, finding of frequent subgraphs from a single graph is a challenging task. To overcome all the challenges, the computation of subgraph pruning in proposed Ap-FSM are performed in two steps in each iteration at local level on each Worker node and then on globally on Master node. The step-wise summary of this phase is given in Algorithm 3.

The frequency threshold is also categorized as local frequency threshold $\tau'$ and global frequency threshold $\tau$, such as $\tau' < \tau$. As, some subgraphs might not be frequent in one graph partition but frequent in other graph partition.

Thus, the value of local frequency threshold $\tau'$ should always be lesser than the global frequency threshold $\tau$. The output of local pruning from each Worker is forwarded to Master node for global pruning. The steps followed for Local pruning and Global pruning are given in Algorithms 4 and 5 respectively. For Local pruning of candidate subgraphs $F^{k+1}$ received from extension phase, an isomorphism test is performed locally at each Worker node first. Detecting isomorphism is computationally expensive and is NP Complete. Let the number of candidate subgraphs $|F^{k+1}|$ at stage $(k+1)$ be $p$. The isomorphism in between $p$ subgraphs is checked $(p*p)$ times.

More than half of computations involved in checking isomorphism between subgraphs are useless and performed to go in vain.

**Algorithm 5**: ***GlobalPruning*** $(I(F^{k+1}), \tau)$

**Input:** Image of frequent subgraphs $I(F^{k+1})$ from Workers $w_i$, Global frequency threshold $\tau$

**Output:** Global Frequent subgraph set $F$

```
1.  Function MasterCompute(msg,superstep)
2.  Sort all I(g_i) ∈ I_w(F^{k+1}) in descending order of their d (g_i)
3.  For each I(F^{k+1}) ∈ Worker w_i do
4.      For each I(g_i) ∈ I(F^{k+1}) do
5.          If (d (g_i)= d (g_j)) then
6.              If (DFS(g_i)= DFS(g_j)) then
7.                  Declare g_i and g_j be isomorphic
8.                  Add g_i and g_j to candidate set f^{k+1}_{g_i}
9.              End
10.         End
11.     End
12.     support( f^{k+1}_{g_i})←support (g_i)+Count (g_j)
13. End
14. If (support( f^{k+1}_{g_i}) ≥ τ ) then
15.     Add all g_i ∈ f^{k+1}_{g_i} to F
16. End
```

To avoid such expensive computations, those subgraphs are identified and aggregated which could be isomorphic using on Lemma 1.

**Lemma 1. (Degree-based Isomorphism)** *Given a subgraph g with an embedding $\varphi$ and another subgraph $g'$ with embedding $\varphi'$ in graph G. The degree of subgraph g and $g'$ are $d(g)=\sum_{v\in g} deg(v)$ and $d(g')=\sum_{v'\in g'} \deg(v')$ respectively. The subgraphs g and $g'$ are not subgraph isomorphic if $d(g)\neq d(g')$.*

For all subgraphs $g_i \in F^{k+1}$, $d(g_i)$ is computed. The subgraph $g_i$ with embedding $\varphi_i$ is stored along with $d(g_i)$ locally on Worker. Further, all stored subgraphs $g_i \in F^{k+1}$ are reorganized by sorting in descending order of their corresponding $d(g_i)$ values. All subgraphs with similar $d(g_i)$ values are aggregated using the Aggregator function of Pregel.

Using Lemma 1, two non-isomorphic subgraphs can be identified, but the subgraph isomorphism cannot be confirmed. For confirming the subgraph isomorphism in between subgraphs g and $g'$, canonical labeling based on depth first subtree (DFS) code is applied. The minimum DFS code is a sequence of labels of vertices and edges (Yan & Han, 2002). The isomorphism test is conducted on the subgraphs with same $d(g_i)$ values using Lemma 2.

**Lemma 2. (Graph Isomorphism)** *Given subgraphs g and $g'$, if the minimum DFS code of g and $g'$ are same, then g and $g'$ are isomorphic.*

Based on Lemma 2 and Definition 5, the isomorphic graphs ($g_1$, $g_2$,.....,$g_i$) are stored in candidate set $C^{k+1}_{g_i}$. The subgraph set $F^{k+1}$ is then again initialized to store the Local frequent subgraphs. If the support of $C^{k+1}_{g_i}$ is greater than local frequency threshold $\tau'$, then all the subgraphs $g_i \in C^{k+1}_{g_i}$ are added to $F^{k+1}$. The algorithm terminates after exploring all the candidate subgraphs.

To reduce the communication cost, a single subgraph image is passed to master for each of the isomorphic frequent subgraph. For example, consider a subgraph $g_1$ with embedding $\varphi_1$ in graph $G$ whose local support is $l$. Instead of passing $l$ number of $g_1$ subgraphs to master, only an image of $g_1$ subgraph is passed. The image $I(g_1)$ consist of single copy of subgraph $g_1$, the local support $l$ of $g_1$ and $d(g_i)$ . The images of all the frequent subgraphs are aggregated in $I(F^{k+1})$ such that:

$$I\left(F^{k+1}\right) = \{I(g_1) \cup I(g_2) \cup \ldots\ldots \cup I(g_h)\}$$

Where, $h$ is the number of isomorphic frequent subgraph sets in $I(F^{k+1})$. The image of locally frequent subgraphs $I(F^{k+1})$ is then passed to the master node by each worker node. Then, the second subgraph pruning step Global Pruning is initiated.

The pseudo code for performing Global Pruning is given in Algorithm 5. The Global Pruning function of Ap-FSM takes as input the images $I(F^{k+1})$ from all $d$ worker nodes ($w_1$ $w_2$,.....,$w_d$) and a global frequency threshold $\tau$. For the images $I(F^{k+1})_{\text{from } d \text{ worker}}$ nodes, the subgraph pruning is performed. The global subgraph pruning is performed in similar way as local subgraph pruning was performed (Lines 5–14) using degree values and DFS code. The isomorphic subgraphs $g_i$ are then stored in set $f^{k+1}_{g_i}$ together.

Then the global frequency of subgraph $g_i$ is computed by adding the local support of $g_i$ from the workers that contain subgraph $g_i$. If the global frequency of frequent isomorphic subgraph $g_i$ is greater than equal to global frequency threshold $\tau$, the subgraph $g_i$ is declared as frequent globally.

The global frequent subgraph set $F$ is created by adding the images of the subgraphs that are frequent globally. The master then passes subgraph set $F$ to all the worker nodes as size-$k$ subgraph for obtaining size ($k$+1) subgraph using subgraph extension in phase 2.

**Table 2**
Features of real world datasets used in experiments.

| Dataset | Original graph description | | Approximate graph description | | #Node labels |
|---|---|---|---|---|---|
| | \|Nodes\| | \|Edges\| | \|Nodes\| | \|Edges\| | |
| US patents | 3,774,768 | 16,518,947 | 1,078,505 | 4,235,627 | 320 |
| LiveJournal | 4,847,571 | 68,993,773 | 1,310,154 | 17,248,443 | 34 |
| Twitter | 41,652,230 | 1,468,365,182 | 8,010,044 | 271,919,547 | 1053 |

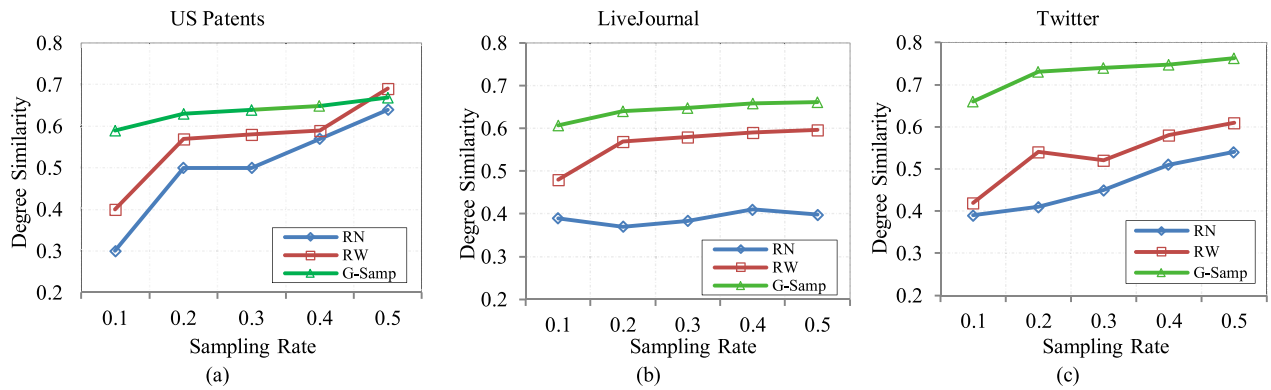


**Fig. 5.** Performance of sampling algorithms in terms of degree similarity.

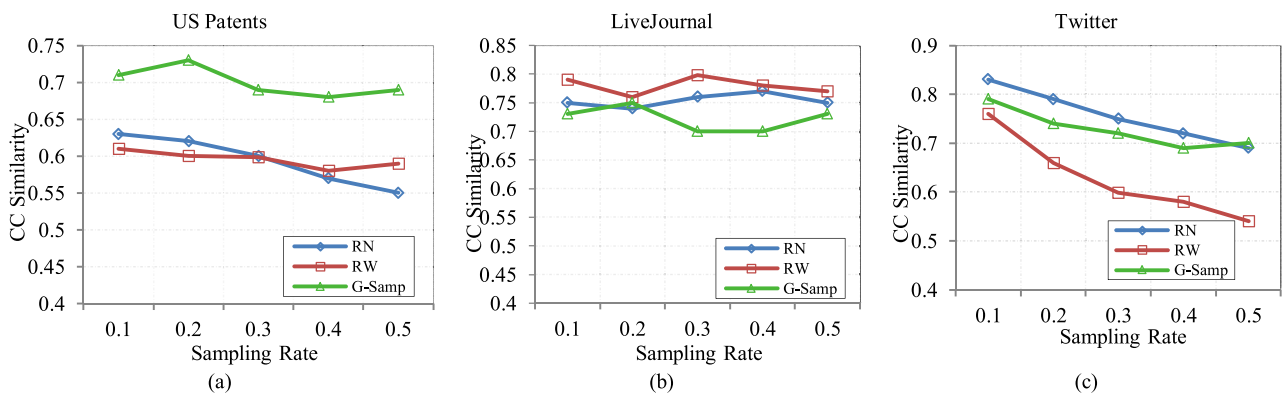


**Fig. 6.** Performance of sampling algorithms in terms of clustering coefficient.

## 5. Performance evaluation

Ap-FSM is implemented using the Giraph version 1.1.0 and Hadoop 2.6.0. Its performance has been evaluated using a Hadoop Cluster composed of 8 Machines equipped with an Intel Xeon processor with 2.9 GHz processor and 8GB of RAM memory with Ubuntu 16.04 operating system installed. One of the machines is taken as Master node and rest are taken as Worker nodes. Master node also acts as a Worker node.

The experiments are performed over one synthetic and three real-life datasets from domains such as social networks and collaboration networks. The statistics of considered graphs are given in Table 2.

Two social graph datasets are considered from LiveJournal and Twitter. LiveJournal is an on-line community which allows its members to maintain journals, group and individual blogs. LiveJournal allows users to announce the friendship with other members. Nodes in this dataset are the users of LiveJournal and the direct edges characterize friendships among members (Leskovec & Andrej, 2014). The twitter graph data (Meeyoung, Haddadi, Benevenuto, & Gummadi, 2010) is based on a snapshot taken in 2009. A vertex represents a user and a directed edge specifies that a user is followed by another user. US patent graph (Hall, Jaffe, & Trajtenberg, 2001) is a citation network of patents registered with the United States Patent and Trademark Office. In this dataset, each node is a patent and an edge represents a citation. The network contains self-citations also which resulted in formation of self-loops.

### 5.1. Performance evaluation of G-Samp

The performance of the proposed G-Samp in representative graph selection phase is compared with the state-of-art sampling algorithms such as Random Node (RN) sampling and Random Walk (RW) sampling. We examined the characteristics of the sampled graphs by varying the sampling rate $\delta$. Let the size of the original graph as 1, sampling rate $\delta = 0.4$ means that the size of sample is 40% of the original graph. The characteristics of the sample $S_i$ are compared with respect to the characteristics of original graph $G$ using:

$$\frac{p(S_i)}{p(G)} \times 100$$

where, $p(S_i)$ and $p(G)$ are particular characteristic of sample $S_i$ and original graph $G$ respectively. The performance of all the algorithms is compared on the basis of quality of samples prepared in terms of Degree similarity, Clustering-Coefficient similarity and Betweenness similarity and is shown in Fig. 5, Fig. 6 and Fig. 7 respectively.

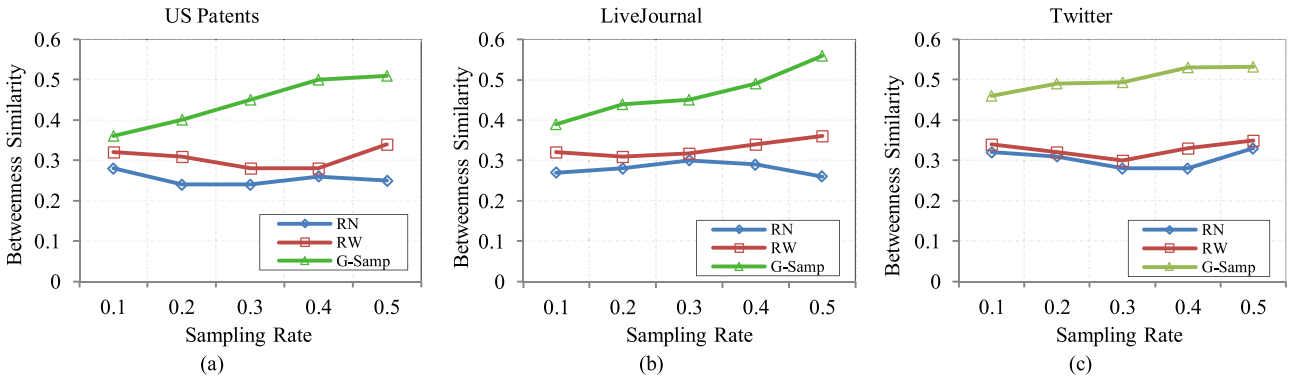


**Fig. 7.** Performance of sampling algorithms in terms of betweenness similarity.

*5.1.1. Degree similarity*

Degree of a vertex is the number of links incident upon a vertex. The degree similarity characteristics of sampling algorithms with the original graph are shown in Fig. 5(a)–(c). In US Patents dataset, for $\delta = 0.1$, the best degree similarity is generated by the proposed approximate phase. However, for $\delta = 0.5$, Random walk sampling produces better samples in terms of the degree distribution. Whereas the performance of proposed approach remains consistently efficient as shown in Fig. 5(a). For LiveJournal graph dataset, the proposed approach outperforms all the competent algorithms for all the sampling rates. In this case, the random node sampling performs worst of all the algorithms. For the Twitter Dataset, the proposed approach again performs well as shown in Fig. 5(c). It can be observed from the Fig. 5 that, the random walk sampling has the most inconsistent results in terms of degree distribution and proposed approach depict high degree similarity.

*5.1.2. Clustering coefficient*

Clustering Coefficient is a measure of the degree to which the vertices of a graph tends to form a cluster. In the case of average global clustering coefficient, there is a significant variation in the performance of the considered sampling algorithms.

The plot in Fig. 6 shows the average fraction of triangles existing around the vertices of degree *d*. Instinctively, it gives an idea about the community structure of the graph. Random node sampling surprisingly has very high clustering coefficient accuracy for the Twitter graph as shown in Fig. 6(c) and has low accuracy for US Patent graph as shown in Fig. 6(a). Whereas Random walk performs better for Twitter dataset as a vertices under a random walk are more connected than the other vertices of the network. However, the proposed G-Samp provides more consistent results for all the sampling rates and is highly accurate for all the considered datasets. For twitter dataset, G-Samp has an average 0.75 value of clustering coefficient while Random node and Random walk has on overage 0.75 and 0.6 value respectively.

*5.1.3. Betweenness similarity*

Betweenness computes the shortest path in between the vertices of the graph. As the proposed approach considers the degree of vertices for sampling thus, has smaller diameter and better betweenness as depicted in Fig. 7. For all the datasets, proposed approach performs better than the random walk sampling and random node sampling.

*5.2. Comparison with existing approaches*

For performance evaluation, proposed Ap-FSM is compared with baseline algorithm, Arabesque (Teixeira et al., 2015), GRAMI (Elseidy et al., 2014), AGRAMI (Elseidy et al., 2014) and gApprox (Chen et al., 2007). Baseline algorithm, Arabesque and GRAMI are exact subgraph mining algorithms while AGRAMI and gApprox are approximate subgraph mining algorithm.

The performance of proposed Ap-FSM along with other competent algorithms is compared by varying the support threshold $\tau$. For fair evaluation, the exact subgraph mining algorithms are evaluated for sample sizes 20%, 30% and 40% as shown in Fig. 8. Ap-FSM outperforms the competent algorithms particularly for smaller values of support threshold $\tau$ in all the cases. Arabesque smashes especially for larger samples and lesser support values because it incurs communication and storage overhead associated with the large number of embeddings. GRAMI, AGRAMI and gApprox are centralized algorithm thus, crashed while handling large data and lower support values.

For $\delta = 20\%$, Ap-FSM is three order of magnitude faster than the GRAMI and an order of magnitude faster than Arabesque, AGRAMI and gApprox for all the considered datasets. For Twitter dataset with $\delta = 20\%$, Arabesque, GRAMI, AGRAMI and gApprox fails to handle smaller support values as shown in Fig. 8(c).

For 30% sampled US Patent dataset, GRAMI crashes for $\tau = 2k$ while AGRAMI and gApprox are found to be computationally expensive. For $\delta = 30\%$, the baseline algorithm also crashes for smaller support values for LiveJournal and Twitter dataset. Approximate subgraph mining algorithms AGRAMI and gApprox crashed for most of the support threshold values. However, the proposed Ap-FSM performs well for all the datasets and is computationally very less expensive in comparison to the other counterparts. When $\delta = 40\%$ is taken, all the competent algorithms suffers in handling such a large amount of data. However, the proposed Ap-FSM performs exceptionally well as shown in Fig. 8(g)–(i). It can be observed that the Ap-FSM performs consistently well for all the data sizes and for all the support values and outperforms all the other considered counterparts.

*5.3. Scalability and speedup*

Fig. 9 shows the execution time of the proposed Ap-FSM with the increased number of worker nodes over the considered graph datasets. As anticipated, increase in the number of processors resulted in decreased run time. As GRAMI is a centralized algorithm, we do not involve it in the computation of node scalability. We are considering $\delta = 20\%$ for Baseline algorithm and Arabesque. For all the datasets, linear run time is observed when the number of worker nodes is increased as shown in Fig. 9. Speedup is a measure for evaluating the efficiency of parallel algorithm with the increasing number of processors. It can be calculated as:

$$Speedup = \frac{Run\ Time_1}{P * Run\ Time_P}$$

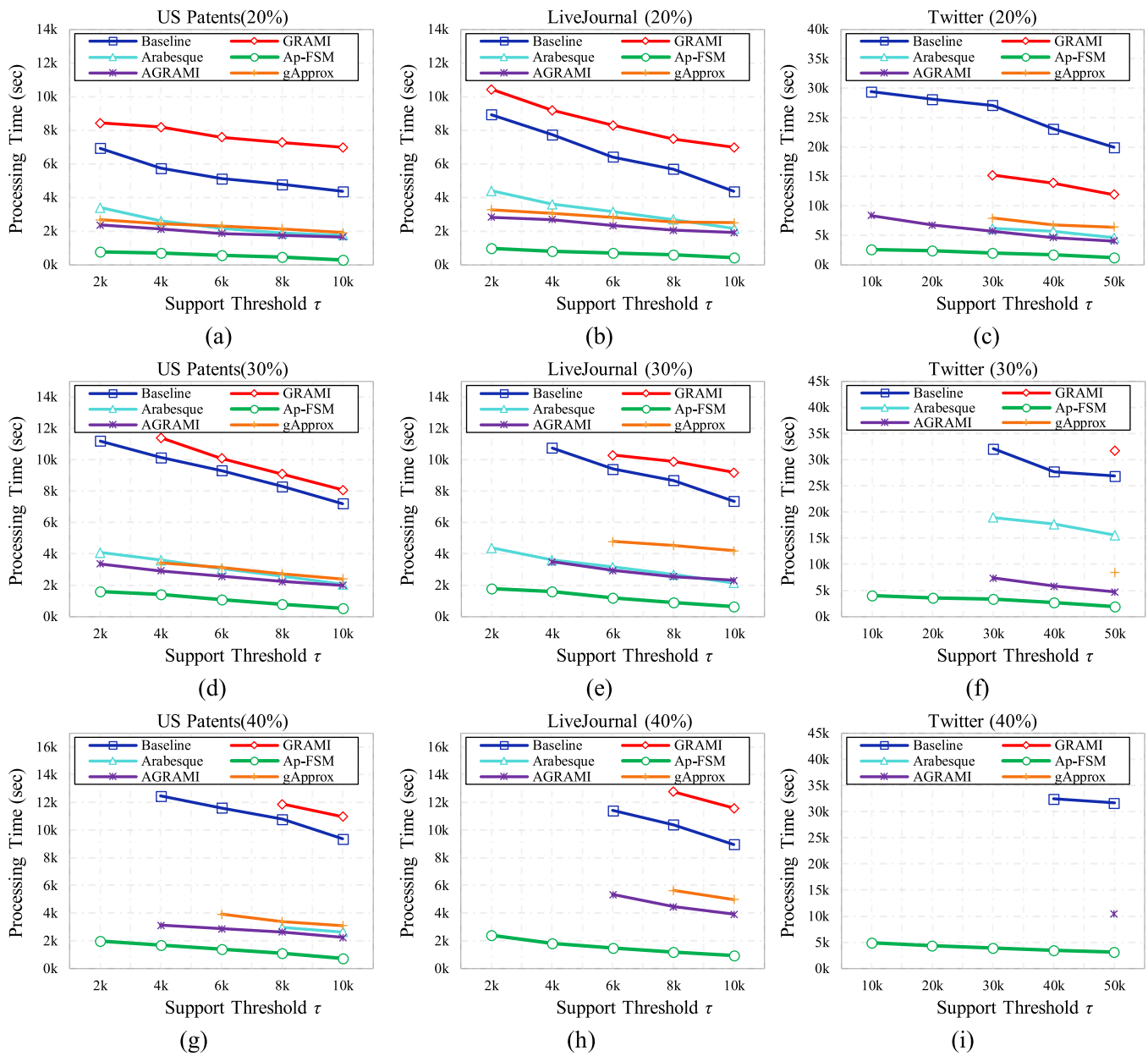


**Fig. 8.** Evaluating performance in terms of processing time (s) on various sample sizes: (a)–(c) 20%, (d)–(f) 30%, (g)–(i) 40%.

where, $P$ is the number of processors used. For ideal parallelization, *Speedup* = 1. *Speedup* behavior of Ap-FSM is calculated using the above formulae. Ap-FSM achieves high *Speedup* efficiency of 74%, 82.5%, and 82.9% for Twitter dataset when frequency values are 30k, 40k and 50k respectively. Arabesque achieves 42%, 41% and 43.5% for Twitter dataset when frequency values are 30k, 40k and 50k respectively. Thus, Ap-FSM achieves an order of magnitude higher speedup than Arabesque for largest dataset used. Therefore, Ap-FSM achieves high speedup efficiency and scalability for all the graph datasets.

### 5.4. Evaluation of optimizations

In this experiment, we measure the effect of each of the optimization used in Ap-FSM. The following versions of the proposed Ap-FSM are implemented on the same distributed environment and are compared with Ap-FSM to discover the advantage of using various optimizations used in Ap-FSM.

- BL: The proposed approximate subgraph mining algorithm is compared with baseline algorithm (BL) which discovers exact subgraphs without any optimization.
- ESM: To measure the advantage of selecting representative graph in phase 1, exact subgraph mining (ESM) is performed by omitting phase 1 of Ap-FSM.
- SSP: To discover the benefit of using two level subgraph pruning, the proposed algorithm is also compared with a single step subgraph pruning (SSP) version of Ap-FSM during phase 3. The sup-graph pruning is performed globally only at Master node in SSP version of Ap-FSM.

The results of all the above mentioned versions of Ap-FSM along with Ap-FSM are shown in Fig. 6 for all the considered datasets US Patents, LiveJournal and Twitter. Each bar represents the processing time in seconds. Bars with red border indicate that the computations do not finish even within the considered maximum elapsed time.

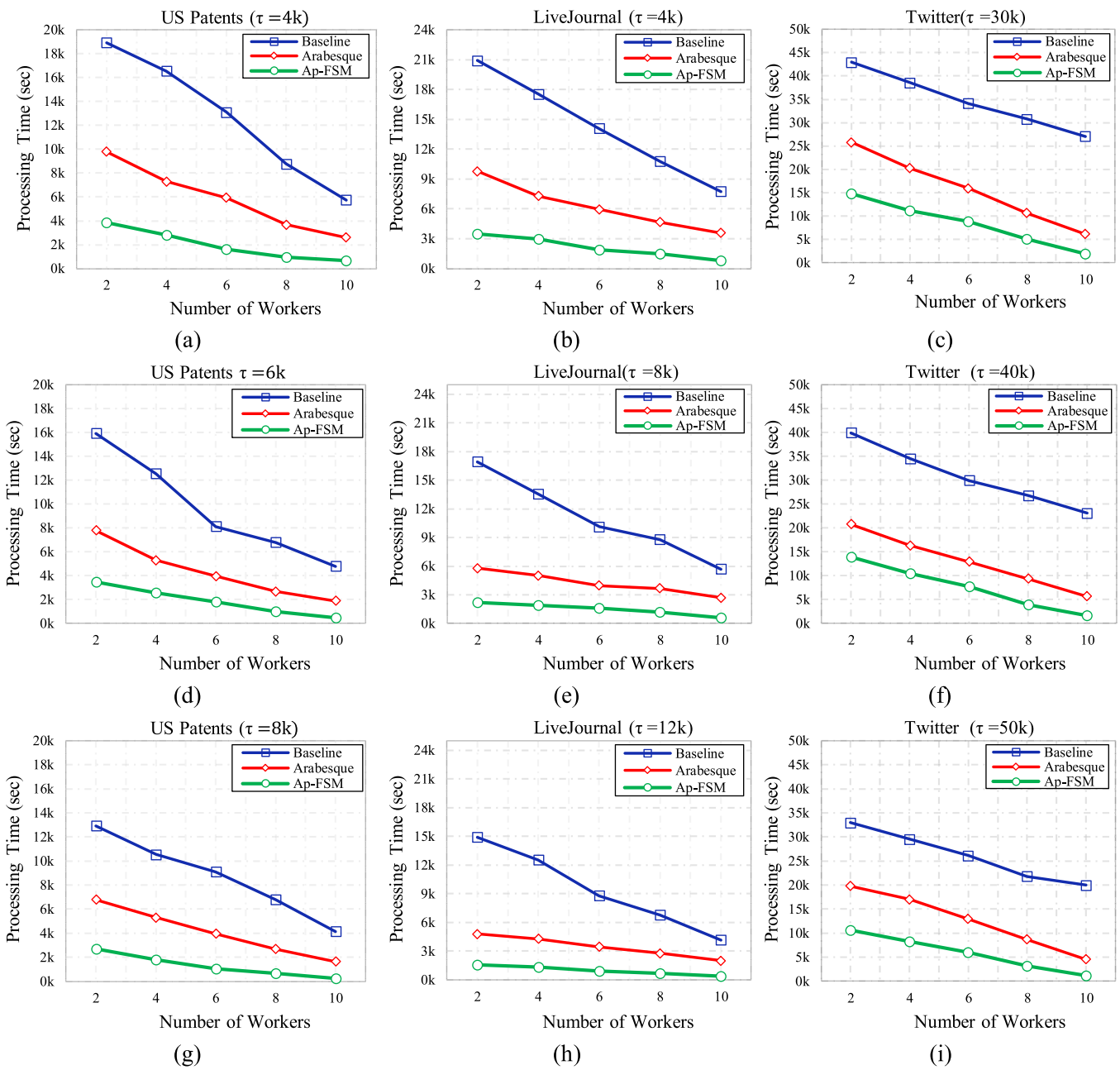


**Fig. 9.** Scalability and speedup efficiency in terms of processing time by varying number of workers: (a)US patents with $\tau = 4$k, (b) LiveJournal with $\tau = 4$k, (c) Twitter with $\tau = 30$k, (d) US Patents with $\tau = 6$k, (e) LiveJournal with $\tau = 8$k, (f) Twitter with $\tau = 40$k, (g) US patents with $\tau = 8$k, (h) LiveJournal with $\tau = 12$k, (i) Twitter with $\tau = 50$k.

Fig. 10(a)–(c) show the processing time for US Patents, LiveJournal and Twitter datasets respectively. It can be observed that the processing time is much less for higher support values in the case of all the datasets. For LiveJournal dataset, the baseline algorithm takes more than 10k seconds to find the exact subgraphs when $\delta = 2$k.

However, the ESM takes an order of magnitude lesser time than the baseline algorithm. However, for higher support values, the difference is lesser in between the two aforementioned versions. For Twitter dataset with more than billion edges, the baseline algorithm takes more than 10k seconds for support values 5k and 10k. Although in comparison, the baseline does not finish its execution within the elapsed time limit. All the three versions of Ap-FSM perform better than the baseline algorithm. The proposed Ap-FSM in comparison to the other two versions is computationally less expensive in terms of processing time for all the datasets.

### 5.5. Accuracy

The accuracy of the subgraphs discovered by Ap-FSM is assessed by comparing the subgraphs produced by the proposed approximate subgraph mining algorithm Ap-FSM having sample rate 0.4 with the exact subgraph mining (ESM) algorithm. We consider F-measure for the evaluation of accuracy of Ap-FSM. If the subgraph is detected as frequent in both Ap-FSM and ESM, we consider it as true positive. If the subgraph is frequent in ESM but not in AP-FSM, it will be treated as false negative and subgraphs frequent in Ap-FSM but not in ESM are considered as false positives. Fig. 11 depicts the F-measures for US Patents and LiveJournal dataset for various support values.

It can be seen that Ap-FSM has high F-Measure for all the datasets. For LiveJournal dataset, F-Measure is high as 0.9 when $\tau = 4$k. Ap-FSM despite of taking much less time in comparison

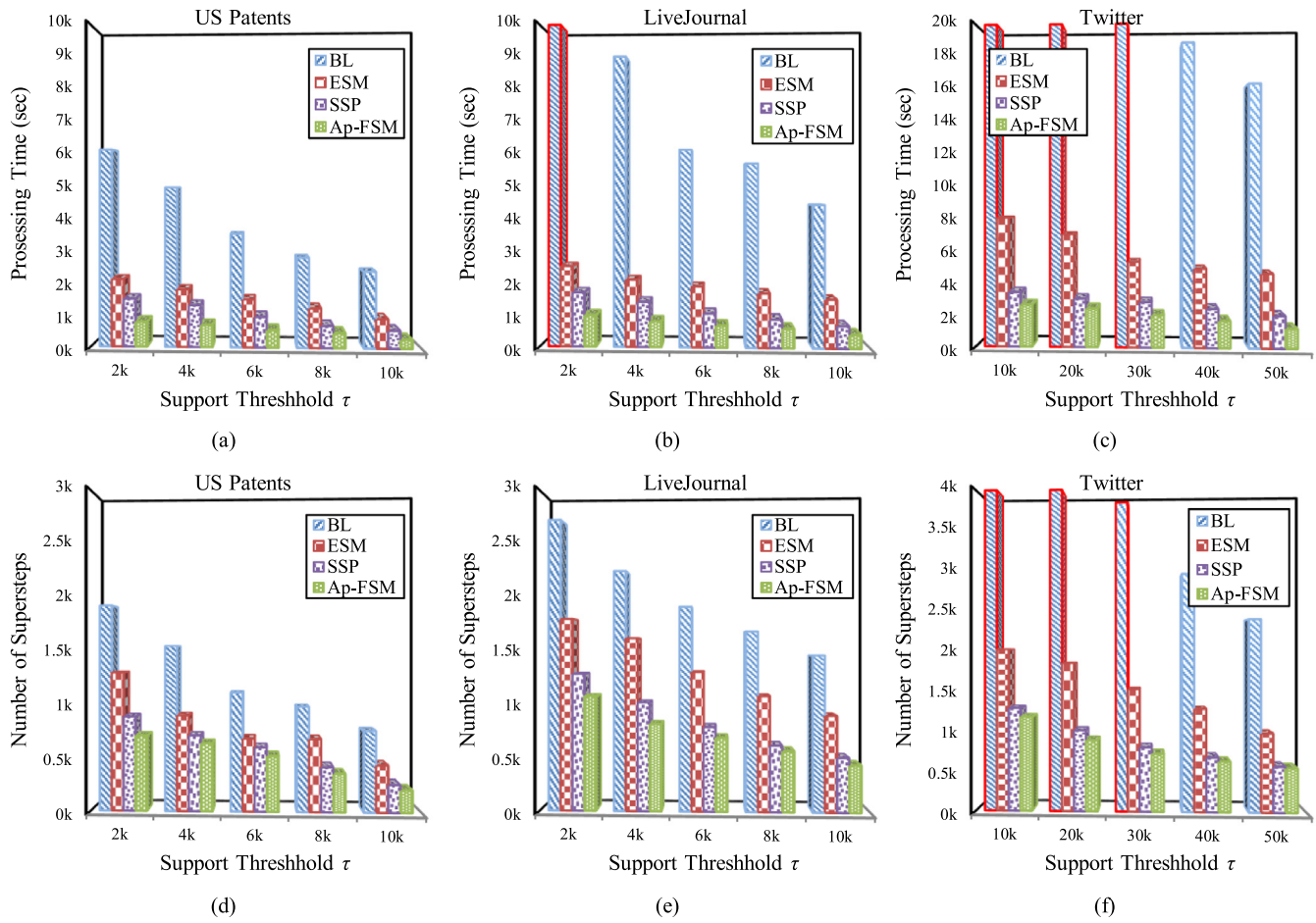


**Fig. 10.** Evaluating performance of optimizations: processing time (a)–(c), number of supersteps (d)–(f).

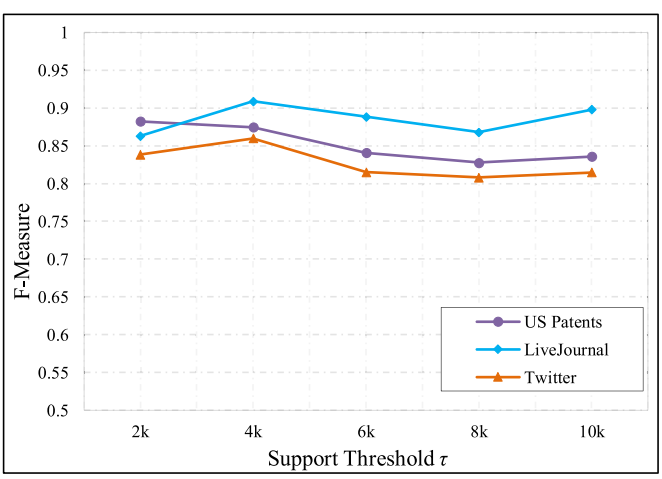


**Fig. 11.** Accuracy of proposed Ap-FSM in terms of F-measure.

to exact subgraph mining algorithm discovers frequent subgraphs with high accuracy.

## 6. Conclusion

In this paper, we propose an approximate frequent subgraph mining algorithm Ap-FSM for a single large graph using distributed graph framework Pregel. Ap-FSM starts with selecting a representative graph which has same properties as of original graph. Further, it discovers the exact subgraphs from the representative graph by performing optimized graph isomorphism and subgraph pruning. A two-step subgraph pruning is used in proposed Ap-FSM to avoid unnecessary communication. The storage overhead is minimized by reducing the number of embeddings to be stored. We performed experiments over real life datasets and compared the performance of proposed Ap-FSM with baseline algorithm, GRAMI, Arabesque and with two approximate subgraph mining algorithm AGRAMI and gApprox. The results show that Ap-FSM outperforms the competent subgraph mining algorithms in terms of processing time and is at least an order of magnitude faster. The efficiency of optimizations used in Ap-FSM is also shown by comparing with its various variants. The results show that each of the optimization provides a significant improvement in the task of frequent subgraph mining. Furthermore, it is shown that Ap-FSM has high data scalability and can efficiently handle graphs with billions of edges.

In future, a suitable partitioning technique may be designed for better load balancing of the proposed Ap-FSM. This work can be extended for handling queries in distributed graph environment. Further, the proposed Algorithm can be applied to various applications like credit card fraud detection, recommendation system, etc. for obtaining interesting patterns.

## References

Abdelhamid, E., Abdelaziz, I., Kalnis, P., Khayyat, Z., & Jamour, F. (2016). ScaleMine: Scalable parallel frequent subgraph mining in a single large graph. In *The proceedings of IEEE international conference for high performance computing, Networking, storage and analysis* (pp. 1–12).

Acosta-Mendoza, N., Gago-Alonso, A., & Medina-Pagola, J. E. (2012). Frequent approximate subgraphs as features for graph-based image classification. *Knowledge-Based Systems, 27*, 381–392. https://doi.org/10.1016/j.knosys.2011.12.002.

Acosta-Mendoza, N., Gago-Alonso, A., Carrasco-Ochoa, J. A., Martínez-Trinidad, J. F., & Medina-Pagola, J. E. (2016). A new algorithm for approximate pattern mining in multi-graph collections. *Knowledge-Based Systems, 109*, 198–207. https://doi.org/10.1016/j.knosys.2016.07.003.

Acosta-Mendoza, N., Gago-Alonso, A., Medina-Pagola, J. E., Carrasco-Ochoa, J. A., & Martínez-Trinidad, J. F. (2017). Extension of canonical adjacency matrices for frequent approximate subgraph mining on multi-graph collections. *International Journal of Pattern Recognition and Artificial Intelligence, 31*(8), 1–33.

Aggarwal, C., & Wang, H. (2010). *Managing and mining graph data*: vol. 40. Database *https://doi.org/10.1007/978-1-4419-6045-0*.

Apache Giraph. (2016). Retrieved March 10, 2017, from http://giraph.apache.org/

Aridhi, S., D'Orazio, L., Maddouri, M., & Mephu Nguifo, E. (2015). Density-based data partitioning strategy to approximate large-scale subgraph mining. *Information Systems, 48*, 213–223. https://doi.org/10.1016/j.is.2013.08.005.

Babu, N. (2016). A distributed approach to weighted frequent subgraph mining. In *The proceedings of IEEE international conference on emerging technology trends* (pp. 1–7).

Bhatia, V., & Rani, R. (2017). A parallel fuzzy clustering algorithm for large graphs using Pregel. *Expert Systems with Applications, 78*, 135–144. https://doi.org/10.1016/j.eswa.2017.02.005.

Bhuiyan, M. A., & Al Hasan, M. (2013). An iterative mapreduce based frequent subgraph mining algorithm. *Transactions on Knowledge and Data Engineering, 4347*(October), 1–17. https://doi.org/10.1109/TKDE.2014.2345408.

Bonato, A., & Tian, Y. (2013). Complex networks and social networks. *Advances in Network Analysis and Its Applications*, 1–18. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-30904-5_12/fulltext.html.

Borgelt, C., & Berthold, M. R. (2002). Mining molecular fragments: Finding relevant substructures of molecules. In *The proceedings of IEEE international conference on data mining* (pp. 51–58). https://doi.org/10.1109/ICDM.2002.1183885.

Bringmann, B., & Nijssen, S. (2008). What is frequent in a single graph? In *Advances in knowledge discovery and data mining* (pp. 1–4). https://doi.org/10.1007/978-3-540-68125-0_84.

Chen, C., Yan, X., Zhu, F., & Han, J. (2007). gApprox: Mining frequent approximate patterns from a massive network. In *The proceedings of IEEE international conference on data mining* (pp. 445–450). https://doi.org/10.1109/ICDM.2007.36.

Chen, Q., Lan, C., Chen, B., Wang, L., Li, J., & Zhang, C. (2017). Exploring consensus RNA substructural patterns using subgraph mining. *IEEE/ACM Transactions on Computational Biology and Bioinformatics, 14*(5), 1134–1146. https://doi.org/10.1109/TCBB.2016.2645202.

Cook, D. J., & Holder, L. B. (2007). *Mining graph data*. Wiley.

Edwards, L., Johnson, L., Milosavljevic, M., Gadepally, V., & Miller, B. A. (2015). Sampling large graphs for anticipatory analytics.

Elseidy, M., Abdelhamid, E., Skiadopoulos, S., & Kalnis, Panos (2014). GRAMI: Frequent subgraph and pattern mining in a single large graph. *The Proceedings of the VLDB Endowment, 7*(7), 517–528. https://doi.org/10.14778/2732286.2732289.

Fiedler, M., & Borgelt, C. (2007). Support computation for mining frequent subgraphs in a single graph. *The Proceedings of the 5th International Workshop on Mining and Learning with Graphs*, 25–30.

Flores-Garrido, M., Carrasco-Ochoa, J. A., & Martínez-Trinidad, J. F. (2014a). Mining maximal frequent patterns in a single graph using inexact matching. *Knowledge-Based Systems, 66*, 166–177. https://doi.org/10.1016/j.knosys.2014.04.040.

Flores-Garrido, M., Carrasco-Ochoa, A. J., & Martnez-Trinidad, J. F. (2014b). AGraP: An algorithm for mining frequent patterns in a single graph using inexact matching. *Knowledge and Information Systems, 44*(2), 385–406. https://doi.org/10.1007/s10115-014-0747-x.

Halder, S., & Lee, Y. K. (2017). Supergraph based periodic pattern mining in dynamic social networks. *Expert Systems with Applications, 72*, 430–442. https://doi.org/10.1016/j.eswa.2016.10.033.

Hall, B. H., Jaffe, A. B., & Trajtenberg, M. (2001). The NBER patent citation data file: Lessons, insights and methodological tools.

Hill, S., Srichandan, B., & Sunderraman, R. (2012). An iterative MapReduce approach to frequent subgraph mining in biological datasets. In *The proceedings of ACM conference on bioinformatics, computational biology and biomedicine* (pp. 661–666). https://doi.org/10.1145/2382936.2383055.

Hsieh, H. P., & Li, C. Te (2010). Mining temporal subgraph patterns in heterogeneous information networks. In *The proceedings of 2nd IEEE international conference on privacy, security, risk and trust* (pp. 282–287). https://doi.org/10.1109/SocialCom.2010.47.

Huan, J., Wang, W., & Prins, J. (2003). Efficient mining of frequent subgraphs in the presence of isomorphism. In *The proceedings of 3rd IEEE international conference on data mining* (pp. 2–5). https://doi.org/10.1109/ICDM.2003.1250974.

Huan, J., Wang, W., Prins, J., & Yang, J. (2004). SPIN: Mining maximal frequent subgraphs from graph databases. In *The proceedings of the 10th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 581–586).

Jain, C., Flick, P., Pan, T., Green, O., & Aluru, S. (2017). An adaptive parallel algorithm for computing connected components. *IEEE Transactions on Parallel and, 28*(9), 2428–2439.

Jia, Y., Zhang, J., & Huan, J. (2011). An efficient graph-mining method for complicated and noisy data with real-world applications. *Knowledge and Information Systems, 28*(2), 423–447. https://doi.org/10.1007/s10115-010-0376-y.

Karci, A. (2009). Efficient automatic exact motif discovery algorithms for biological sequences. *Expert Systems with Applications, 36*(4), 7952–7963. https://doi.org/10.1016/j.eswa.2008.10.087.

Ketkar, N. S., Holder, L. B., & Cook, D. J. (2005). Subdue: Compression-based frequent pattern discovery in graph data. In *The proceedings of 1st international workshop on open source data mining: Frequent pattern mining implementations* (pp. 71–76). https://doi.org/10.1145/1133905.1133915.

Kuramochi, M., & Karypis, G. (2001). Frequent subgraph discovery. In *The proceedings of IEEE international conference on data mining* (pp. 313–320). https://doi.org/10.1109/ICDM.2001.989534.

Kuramochi, M., & Karypis, G. (2004a). An efficient algorithm for discovering frequent subgraphs. *IEEE Transactions on Knowledge and Data Engineering, 16*(9), 1038–1051.

Kuramochi, M., & Karypis, G. (2004b). GREW – A scalable frequent subgraph discovery algorithm. In *The proceedings of 4th IEEE international conference on data mining* (pp. 439–442). https://doi.org/10.1109/ICDM.2004.10024.

Kuramochi, M., & Karypis, G. (2005). Finding frequent patterns in a large sparse graph. *Journal of Data Mining and Knowledge Discovery, 11*(3), 243–271. https://doi.org/10.1007/s10618-005-0003-9.

Kuramochi, M., & Karypis, G. (2007). Discovering frequent geometric subgraphs. *Information Systems, 32*(8), 1101–1120. https://doi.org/10.1016/j.is.2005.05.005.

Leskovec, J., & Andrej, K. Stanford Large Network Dataset Collection. Retrieved March 10, 2017, from https://snap.stanford.edu/data/.

Leung, I. X. Y., Hui, P., Liò, P., & Crowcroft, J. (2009). Towards real-time community detection in large networks. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics, 79*(6), 1–10. https://doi.org/10.1103/PhysRevE.79.066107.

Lin, W., Xiao, X., & Ghinita, G. (2014). Large-scale frequent subgraph mining in MapReduce. In *The proceedings of IEEE international conference on data engineering* (pp. 844–855).

Liu, W., Zhu, L., Chu, L., & Ma, H. (2018). A common subgraph correspondence mining framework for map search services. *Multimedia Tools and Applications*. https://doi.org/10.1007/s11042-017-5553-5.

Lu, W., Chen, G., Tung, A. K. H., & Zhao, F. (2013). Efficiently extracting frequent subgraphs using MapReduce. In *The proceedings of IEEE international conference on big data* (pp. 639–647). https://doi.org/10.1109/BigData.2013.6691633.

Malewicz, G., Austern, M. H., Bik, A. J. C., Dehnert, J. C., Horn, I., Leiser, N., et al. (2010). Pregel : A system for large-scale graph processing. In *The proceedings of the ACM SIGMOD international conference on management of data* (pp. 135–145).

Martella, C., Logothetics, D., & Shaposhnik, R. (2015). Practical graph analytics with apache giraph.

Meeyoung, C., Haddadi, H., Benevenuto, F., & Gummadi, K. P. (2010). Measuring user influence in twitter: The million follower fallacy. *The proceedings of the 4th international AAAI conference on weblogs and social media (ICWSM)*.

Mitzenmacher, M., Peng, R., & Xu, S. C. (2015). Scalable large near-clique detection in large-scale networks via sampling. In *The proceedings of 21th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 815–824).

Moussaoui, M., & Zaghdoud, M. ,A. J. (2016). POSGRAMI: Possibilistic frequent subgraph mining in a single large graph. In *The proceedings of international conference on information processing and management of uncertainty in knowledge-based systems* (pp. 549–561).

Mozafari, B., Sarkar, P., Franklin, M., Jordan, M., Madden, S., & Berkeley, U. C. (2015). Scaling up crowd-sourcing to very large datasets : A case for active learning, 125–136.

Nijssen, S., & Kok, J. (2001). Faster association rules for multiple relations. In *The proceedings of international joint conference on artificial intelligence* (pp. 891–896).

Nijssen, S., & Kok, J. N. (2004). A quickstart in frequent structure mining can make a difference. In *The proceedings of the tenth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 647–652). https://doi.org/10.1145/1014052.1014134.

Ozaki, T., & Etoh, M. (2011). Closed and maximal subgraph mining in internally and externally weighted graph databases. In *The proceedings of 25th IEEE international conference on advanced information networking and applications* (pp. 626–631). https://doi.org/10.1109/WAINA.2011.48.

Pears, R., Pisalpanus, S., & Koh, Y. S. (2015). A graph based approach to inferring item weights for pattern mining. *Expert Systems with Applications, 42*(1), 451–461. https://doi.org/10.1016/j.eswa.2014.07.030.

Qiao, F., Zhang, X., Li, P., Ding, Z., Jia, S., & Wang, H. (2018). A parallel approach for frequent subgraph mining in a single large graph using spark. *Applied Sciences, 8*(2), 1–18. https://doi.org/10.3390/app8020230.

Reinhardt, S., & Karypis, G. (2007). A multi-level parallel implementation of a program for finding frequent patterns in a large sparse graph. In *The proceedings of 21st international symposium on parallel and distributed processing* (pp. 1–8). https://doi.org/10.1109/IPDPS.2007.370404.

Ribeiro, B., & Towsley, D.. *Estimating and sampling graphs with multidimensional random walks* arXiv: 1703.08252 https://doi.org/10.1145/1879141.1879192.

Saha, T. K., & Al Hasan, M. (2014). FS3: A sampling based method for top-k frequent subgraph mining. In *The proceedings of IEEE international conference on big data* (pp. 72–79). https://doi.org/10.1109/BigData.2014.7004359.

Teixeira, C. H. C., Fonseca, A. J., Serafini, M., Siganos, G., Zaki, M. J., & Aboulnaga, A. (2015). Arabesque: A system for distributed graph mining. In *The proceedings of the 25th symposium on operating systems principles* (pp. 425–440). https://doi.org/10.1145/2815400.2815410.

Thomas, L. T., Valluri, S. R., & Karlapalem, K. (2010). MARGIN: Maximal frequent subgraph mining. *ACM Transactions on Knowledge Discovery from Data, 4*(3), 1–42. https://doi.org/10.1145/1839490.1839491.

Voudigari, E., Salamanos, N., Papageorgiou, T., & Yannakoudakis, E. J. (2016). Rank degree : An efficient algorithm for graph sampling. In *The proceedings of IEEE/ACM international conference on advances in social networks analysis and mining (ASONAM)* (pp. 120–129).

Wang, K., Xie, X., Jin, H., Yuan, P., Lu, F., & Ke, X. (2016). Frequent subgraph mining in graph databases based on MapReduce. In *The proceedings of 10th Asia-Pacific services computing conference on advances in services computing, APSCC* (pp. 464–476). https://doi.org/10.1007/978-3-319-49178-3.

Wang, L., Wang, Z., Zhao, S., & Tan, S. (2015). Stock market trend prediction using dynamical Bayesian factor graph. *Expert Systems with Applications, 42*(15–16), 6267–6275. https://doi.org/10.1016/j.eswa.2015.01.035.

Wu, D., Ren, J., & Sheng, L. (2017). Uncertain maximal frequent subgraph mining algorithm based on adjacency matrix and weight. *International Journal of Machine Learning and Cybernetics*, 1–11. https://doi.org/10.1007/s13042-017-0655-y.

Yan, X., & Han, J. (2002). gSpan: Graph-based substructure pattern mining. *Journal of Chemical Information and Modeling, 53*(9), 1689–1699. https://doi.org/10.1017/CBO9781107415324.004.

Yan, X., & Han, J. (2003). CloseGraph: Mining closed frequent graph patterns. In *The proceedings of 9th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 286–295). https://doi.org/10.1145/956750.956784.

Zou, R., & Holder, L. B. (2010). Frequent subgraph mining on a single large graph using sampling techniques. In *The proceedings of the 8th workshop on mining and learning with graphs* (pp. 171–178). https://doi.org/10.1145/1830252.1830274.

Zou, Z., Li, J., Gao, H., & Zhang, S. (2010). Mining frequent subgraph patterns from uncertain graph data. *IEEE Transactions on Knowledge and Data Engineering, 22*(9), 1203–1218. https://doi.org/10.1109/TKDE.2010.80.