

# Near-optimal Top- $k$ Pattern Mining

Xin Wang<sup>a</sup> (xinwang@swpu.edu.cn), Zhuo Lan<sup>a</sup>  
(202022000326@stu.swpu.edu.cn), Yu-Ang He<sup>a</sup>  
(202022000311@stu.swpu.edu.cn), Yang Wang<sup>a</sup> (wangyang@swpu.edu.cn),  
Zhi-Gui Liu<sup>b</sup> (liuzhigui@swust.edu.cn), Wen-Bo Xie<sup>a,\*</sup>  
(wenboxie@swpu.edu.cn)

<sup>a</sup> School of Computer Science, Southwest Petroleum University,  
Chengdu 610500, China

<sup>b</sup> School of Information Engineering, Southwest University of Science and Technology,  
Mianyang 621010, China

**Abstract.** Nowadays, frequent pattern mining (FPM) on large graphs receives increasing attention, since it is crucial to a variety of applications, e.g., social analysis. Informally, the FPM problem is defined as finding all the patterns in a large graph with frequency above a user-defined threshold. However, this problem is nontrivial due to the unaffordable computational and space costs in the mining process. In light of this, we propose a cost-effective approach to mining near-optimal top- $k$  patterns. Our approach applies a “level-wise” strategy to incrementally detect frequent patterns, hence is able to terminate as soon as top- $k$  patterns are discovered. Moreover, we develop a technique to compute the lower bound of support with smart traverse strategy and compact data structures. Extensive experimental studies on real-life and synthetic graphs show that our approach performs well, i.e., it outperforms traditional counterparts in efficiency, memory footprint, recall and scalability.

**Keywords:** Frequent Pattern Mining, Graph Mining, Social analysis

## 1 Introduction

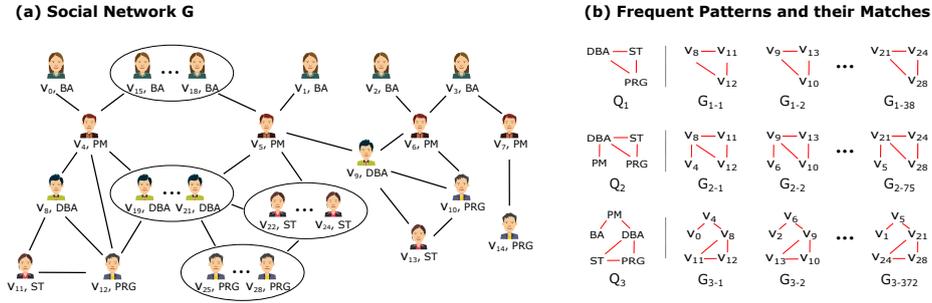
Frequent pattern mining is one of the most important problems in knowledge discovery and graph mining, of which the main task is to find subgraphs with support above a threshold, from a dataset. There are two main types of settings considered to detect frequent patterns in previous researches, i.e., transactional-based and single-graph-based. Recently, the single-graph-based setting has given rise to a high degree of academic attention, owing to its wide applications in e.g., bioinformatics (Xue, Klabjan, & Luo, 2019), cheminformatics (Sabe et al., 2021), web analysis and social network analysis (Daud, Ab Hamid, Saadoon, Sahran, & Anuar, 2020). Methods that rely on the single-graph-based setting mostly follow the combinatorial pattern enumeration paradigm. However, it is costly and unnecessary to enumerate all the patterns in real-world applications such as social network analysis (Huan, Wang, Prins, & Yang, 2004; X.-F. Yan & Han, 2003).

---

\* Corresponding author at: School of Computer Science, Southwest Petroleum University, Chengdu 610500, China. E-mail: wenboxie@swpu.edu.cn (Wen-Bo Xie)

The minimum-image-based support (MNIS for short) (Bringmann & Nijssen, 2008) is widely used in traditional FPM algorithms due to its simplicity of calculation. Generally, the traditional algorithms maintain all the matches of a pattern to calculate its MNIS support. This brings big challenges to the mining evaluation on large graphs, as there may exist (potentially) exponentially many matches of a pattern in a large graph, which leads to an unsatiable memory cost and low scalability.

In addition to the scalability, the practicability is also considerable. In most real-world applications, it is unnecessary to enumerate all the patterns. On one hand, people prefer to focus on some typical patterns rather than scan the dazzling low-value ones (Zhu et al., 2011). On the other hand, given a frequent pattern, all of its sub-patterns must be frequent as well, thus these sub-patterns are to some extent considered as “redundant” patterns.



**Fig. 1.** A snapshot of a social graph  $G$  & three patterns along with their matches

*Example 1.* A fraction of a social graph  $G$  is shown in Fig. 1 (a), where each node denotes a person with ID and job title (e.g., project manager (PM), database administrator (DBA), programmer (PRG), business analyst (BA) and software tester (ST)); and each edge indicates friendship, e.g.,  $(v_0, v_4)$  indicates that  $v_0$  and  $v_4$  are friends. From graph  $G$ , one can discover a few typical patterns, e.g.,  $Q_1$ ,  $Q_2$  and  $Q_3$  as well as their matches (Fig. 1 (b)). Note that  $Q_1$  and  $Q_2$  are both the subgraphs of  $Q_3$ , if they are considered frequent and returned, then we will have to face a large set of frequent patterns, which not only includes “redundancy” but also is costly for inspection. Instead, we only need top- $k$  patterns. Then the cost for inspection and mining can be greatly reduced. For example, when  $k = 1$ ,  $Q_3$  is considered more interesting than  $Q_1$  and  $Q_2$  from the perspective of closeness (X.-F. Yan & Han, 2003) and hence is more preferred.  $\square$

The example suggests us to investigate *top- $k$  pattern mining* problem. While two crucial questions have to be answered:

(1) What metrics for measuring support and interestingness of a pattern shall we choose?

(2) How to develop an efficient algorithm such that (i) mining computation can terminate as soon as  $k$  patterns are identified and (ii) support evaluation can be processed less costly in both evaluation time and memory footprint?

**Contributions.** This paper investigates the *top-k pattern mining* problem, and provides an effective approach to mining *near-optimal top-k* patterns. Our contributions are as follows.

(1) We adopt minimum-image-based support and propose a metric for measuring “interestingness” of a pattern. Based on the metrics, we formalize the *top-k pattern mining* (TOPKPM) problem and show the intractability of the problem (Section 3).

(2) We investigate the TOPKPM problem and develop an approach to identifying *near-optimal top-k* patterns. The algorithm has following desirable performances: (a) it preserves *early termination property*, hence can terminate as soon as  $k$  preferred patterns are discovered; and (b) the pattern set shows high recall value, compared with the optimal solution via intensive tests (Section 4.1).

(3) To facilitate support evaluation, we devise a novel technique for fast estimation. Our technique, which captures the essential feature of MNIS-based metric, well plugs into our main algorithm that works in a “level-wise” manner, hence is able to estimate the MNIS support efficiently and accurately, while consuming much less memory space (Section 4.2).

(4) Using real-life and synthetic graphs, we experimentally verify the performances of our algorithm and find the following (Section 5). (a) Our algorithm shows excellent performance *w.r.t.* response time and memory cost on various real-life graphs. In particular, the required response time of our algorithm is about one order of magnitude faster than its counterparts. (b) Our algorithm, though incorporates approximation scheme, is able to obtain desired recalls, i.e., the set of top- $k$  patterns identified is *near-optimal*. For example, on two real-life graphs, our algorithm even achieves 100% recall. (c) Our algorithm scales much better than its counterparts, *w.r.t.* response time and memory footprint.

## 2 Related Work

The FPM problem on single large graphs has been well studied and a host of techniques have been proposed. We next review them as follows.

**Exact mining.** A large part of prior works focus on mining exact results. On static graphs, (Elseidy, Abdelhamid, Skiadopoulou, & Kalnis, 2014) formulated the FPM as a constrained satisfaction problem, and proposed an efficient algorithm called GraMI. (D. Yan, Qu, Guo, & Wang, 2020) divided the workload by prefix projection to achieve efficient frequent pattern mining on multicore machines. A framework (Ur Rehman, Liu, Ali, Nawaz, & Fong, 2021) was proposed to effectively reduce the duplicate and enormous frequent patterns through the initiation of a new ranking measurement called FSP-Rank. On weighted graphs, (Ashraf et al., 2019; N. Le, Vo, Nguyen, Fujita, & Le, 2020) proposed approaches to detecting frequent patterns with weights. Over evolving graphs, (Abdelhamid et al., 2017) introduced another dynamic algorithm IncGM+, which divides an input graph into frequent and infrequent updated subgraphs and prunes the update area by adjusting the boundary subgraphs named “fringe”. This approach keeps small memory overhead. To tackle the distributive FPM problem

and leverage parallel computation, DISTGRAPH (Talukder & Zaki, 2016) uses a set of optimizations and efficient collective communication operations to minimize the total amount of messages shipped among different sites. ScaleMine (Abdelhamid, Abdelaziz, Kalnis, Khayyat, & Jamour, 2016) leverages the approximate and exact phases to achieve better load balance and more efficient evaluation when mining candidate patterns. (T. Wang, Huang, Lu, Peng, & Du, 2018) adopts a message-passing-free scheme among workers and utilizes a task scheduler to dynamically balance the workload for frequent subgraph mining on distributed systems. For the methods with depth-first order, gSpan (X. Yan & Han, 2002) designs a DFS lexicographic order to support the mining algorithm. FFSM (Huan, Wang, & Prins, 2003) develops a new graph canonical form and completely avoids subgraph isomorphism testing by maintaining an embedding set for each frequent subgraph. Gaston (Nijssen & Kok, 2004) adopts a step-wise approach that uses combinations of frequent paths, frequent free trees, and cyclic graphs to discover frequent subgraphs.

**Approximate mining.** To support practical applications, a host of techniques were developed for approximate pattern mining, under various settings. In (Elseidy et al., 2014), an approximate solution called AGRAMI was also proposed to produce an incomplete set of frequent patterns with no false positives. On graphs with noise, exact matching is no longer feasible, (Driss, Boulila, Leborgne, & Gançarski, 2021) introduced an approach, which allows inexact matching, to mining frequent patterns. Sampling-based algorithms have been proposed for the issue. (Nasir, Aslay, Morales, & Riondato, 2021) presented TipTap, a collection of sampling-based approximation algorithms for mining frequent  $k$ -vertex patterns in fully-dynamic graphs. (Preti, De Francisci Morales, & Riondato, 2021) proposed another sampling-based randomized algorithm called MaNIACS, of which the accuracy can be guaranteed by empirical Vapnik-Chervonenkis (VC) dimension. (Zheng & Wang, 2021) introduced a graph sampling algorithm RASI to reduce the unessential structure of a data graph. RASI demonstrates higher efficiency and greater accuracy than its counterparts for FPM. REAFUM (Li & Wang, 2015) focuses on finding non-redundant representative frequent patterns that summarize the frequent patterns using approximate matching in a graph database. APGM (Jia, Zhang, & Huan, 2011) models the noise distribution through a probability matrix, and then uses an approximate matching strategy to mine useful patterns from the noise map database. VEAM (Acosta-Mendoza, Gago-Alonso, & Medina-Pagola, 2012) mines frequent subgraphs under the semantic of inexact matching. The approach identifies frequent patterns from a collection of images with slight angular differences between the positions of image segments. On uncertain graphs, (Chen, Zhao, Lin, Wang, & Guo, 2019) developed an approximation algorithm with accuracy guarantee for the FPM problem under probabilistic semantic.

**Top- $k$  mining.** The topic of identifying  $k$  best patterns arose much attention in recent years. (Semertzidis & Pitoura, 2019) proposed an algorithm for mining top- $k$  durable matches in dynamic graphs, which uses a compact representation of the graph snapshots and appropriate time indexes to prune the search space.

(X. Wang et al., 2021) proposed a metric to measure the quality of a pattern and developed a parallel algorithm with early termination property to efficiently discover  $k$  best patterns in a distributed large graph. FastPat framework (Zeng, U, Yan, Han, & Tang, 2021) utilizes the meta index and an upper bound of the frequency score to prune unqualified candidates. In particular, FastPat efficiently calculates the support of candidates through a join-based approach. (Prateek, Khan, Goyal, & Ranu, 2020) uses a holistic best-first exploration strategy along with a compressed data structure called Replica to identify pairs of subgraph patterns that frequently co-occur in proximity within a single graph. RESLING (Natarajan & Ranu, 2018) is a framework to mine the top- $k$  representative patterns. It evaluates patterns from the edit map and performs diversified ranking through two random-walk-based algorithms. (Aslay, Nasir, De Francisci Morales, & Gionis, 2018) addressed the problem of approximate  $k$ -vertex frequent pattern mining on a dynamic graph with high probability in a given time. To mine the top- $k$  uncertain frequent patterns from uncertain databases, (T. Le, Vo, Huynh, Nguyen, & Baik, 2020) introduced an approach that combines the mining and ranking phases as a whole to improve efficiency and reduce the memory cost.

**Our work differs from earlier works in two main aspects:** (1) a “level-wise” strategy is employed in the mining process to ensure the *early termination property*; (2) a novel support evaluation technique, that leverages wise traversal strategy and compact data structures is incorporated in the mining process. As a result, our method is committed to delivering *near-optimal* results (the recall is up to 100%) with low computational and memory costs.

### 3 Graphs, Patterns and Top- $k$ Pattern Mining

In this section, we first review graphs, patterns, graph pattern matching; we then formalize the *top- $k$  pattern mining* problem.

#### 3.1 Graph Pattern Matching

**Definition 1. Graph & Subgraph.** A data graph (or simple graph) is defined as  $G = (V, E, L)$ , where (1)  $V$  is a set of nodes; (2)  $E \subseteq V \times V$  is a set of undirected edges; and (3) each node  $v \in V$  carries a tuple  $L(v) = (A_1 = a_1, A_2 = a_2, \dots, A_n = a_n)$ , in which  $A_i = a_i (i \in [1, n])$  represents that the node  $v$  has a value  $a_i$  for the attribute  $A_i$ , and is denoted as  $v.A_i = a_i$ .

A graph  $G_s = (V_s, E_s, L_s)$  is a subgraph of  $G = (V, E, L)$ , denoted by  $G_s \subseteq G$ , if  $V_s \subseteq V$ ,  $E_s \subseteq E$ , and moreover, for each  $v \in V_s$ ,  $L_s(v) = L(v)$ .  $\square$

**Definition 2. Pattern & Sub-pattern.** A pattern  $Q$  is defined as a graph  $(V_p, E_p, f_v)$ , where  $V_p$  and  $E_p$  are the set of nodes and edges, respectively; for each  $u$  in  $V_p$ , it is associated with a predicate  $f_v(u)$  defined as a conjunction of atomic formulas of the form of ‘ $A = a$ ’ such that  $A$  denotes an attribute of the node  $u$  and  $a$  is a value of  $A$ . Intuitively,  $f_v(u)$  specifies search conditions imposed by  $u$ , that is, for a node  $v$  in  $G$ , if for each atomic formula ‘ $A = a$ ’ in

$f_v(u)$ , there is an attribute  $A$  in  $L(v)$  with  $v.A = a$ , then the node  $v$  satisfies  $f_v(u)$  (denoted as  $v \sim u$ ).

A pattern  $Q' = (V'_p, E'_p, f'_v)$  is subsumed by another pattern  $Q = (V_p, E_p, f_v)$ , denoted by  $Q' \sqsubseteq Q$ , if  $(V'_p, E'_p)$  is a subgraph of  $(V_p, E_p)$ , and function  $f'_v$  is a restriction of  $f_v$ . Then,  $Q'$  is referred to as a sub-pattern of  $Q$  if  $Q' \sqsubseteq Q$ .  $\square$

**Definition 3. Pattern Matching.** We adopt the subgraph isomorphism (Cordella, Foggia, Sansone, & Vento, 2004) as the matching semantic. A subgraph  $G_s$  of  $G$  matches a pattern  $Q$  via isomorphism, iff there exists a bijective function  $\rho: V_s \Rightarrow V_p$ , such that (i) for each  $v \in V_s$ ,  $v \sim \rho(v)$  and (ii)  $(v_i, v_j) \in E_s$  iff  $(\rho(v_i), \rho(v_j)) \in E_p$ .

In a graph  $G$ , if there exists a subgraph  $G_s$  that is mapped from  $Q$  via  $\rho$ , then  $G_s$  is referred to as a *match* of  $Q$  in  $G$ , and the match set  $M(Q, G)$  includes all the matches  $G_s$  of  $Q$  in  $G$ . Abusing the notation of *match*, we denote  $v$  in  $G_s$  as a *match* of  $u$  in  $Q$ , if  $\rho(u) = v$ . Then for each node  $u$  in  $E_p$ , one can derive a set  $\{v | v \in G_s, G_s \in M(Q, G), v \sim u\}$  from  $M(Q, G)$ , and denote it by  $\text{IMG}(u)$ . One may verify that  $\text{IMG}(u)$  consists of a set of *distinct* nodes  $v$  in  $G$  as *matches* of  $u$  in  $Q$ .

**Definition 4. Forward & Backward Expansions.** Given a pattern  $Q$ , its DFS tree  $T_Q$  can be built via a depth-first search on  $Q$  from one of its node  $u$ . Then, edges in  $T_Q$  are referred to as *forward edges* and the remaining edges in  $Q$  are denoted as *backward edges*. Thus, the *Forward expansion* enlarges  $Q$  by including a new edge from an existing node in  $Q$  to a newly introduced node; while the *Backward Expansion* includes a new edge from two existing nodes of  $Q$ .  $\square$

For example, a pattern  $Q_c$  with edge set  $\{(ST, DBA), (DBA, PRG)\}$  can be generated via *forward expansion* from a pattern with edge  $(ST, DBA)$ ; with  $Q_c$ , another pattern  $Q_1$  (shown in Fig. 1(b)) is generated via *backward expansion*.

**Other Notations.** (1) The total size  $|G|$  of  $G$  (resp.  $|Q|$  of  $Q$ ) is  $|V| + |E|$  (resp.  $|V_p| + |E_p|$ ), i.e., the total number of nodes and edges in  $G$  (resp.  $Q$ ). (2) For a pattern  $Q$ , its *complete* pattern  $\hat{Q}$  is such a pattern that takes the same set of nodes as  $Q$ , and moreover, has an edge for each pair of nodes in  $\hat{Q}$ . (3) The height of a node  $v$  in a rooted and directed tree  $\mathcal{T}$  is the length of the longest path from  $v$  to a leaf node of  $\mathcal{T}$ . Similarly, the height  $h$  of  $\mathcal{T}$  is the maximum height among all nodes in  $\mathcal{T}$ .

A summary of notations are listed in Table 1.

### 3.2 Top- $k$ Pattern Mining Problem

Below, we first review the frequent pattern mining problem, and then formalize the *top- $k$  pattern mining* (TOPKPM) problem. We start from the support metric.

**Definition 5. Support.** The support of a pattern  $Q$  in a single graph  $G$ , denoted by  $\text{Sup}(Q, G)$ , indicates the appearance frequency of  $Q$  in  $G$ .  $\square$

**Table 1.** A summary of notations

Symbols	Notations
$G = (V, E, L)$	a data graph
$Q = (V_p, E_p, f_v)$	a pattern
$G_s \subseteq G$	$G_s$ is a subgraph of $G$
$Q' \sqsubseteq Q$	$Q'$ is a sub-pattern of $Q$
$M(Q, G)$	the set of matches of $Q$ in $G$
$\text{IMG}(u)$	the set of matches of node $u$ of $Q$ in $G$ , derived from $M(Q, G)$
$ V  +  E $	$ G $ , the size of $G$
$ V_p  +  E_p $	$ Q $ , the size of $Q$
$\mathcal{T}$	a rooted and directed tree for maintaining frequent patterns
$h$	the height of tree $\mathcal{T}$
$\text{Sup}(Q, G)$ (resp. $\theta$ )	the support of a pattern $Q$ in $G$ (resp. threshold of support)
$\text{ITRS}(Q)$	the interestingness of a pattern $Q$
$\widehat{Q}$	the <i>complete</i> pattern of $Q$
$D(Q)$ (resp. $D_i(Q)$ )	the domain of a pattern $Q$ (resp. a pattern node $u_i$ in $Q$ )
$e_x = (u_i, u_j)$	an edge for pattern extension

Analogous to the association rules for itemsets, the support metric for patterns should be anti-monotonic, i.e., for patterns  $Q$  and  $Q'$ , if  $Q' \sqsubseteq Q$ , then  $\text{Sup}(Q', G) \geq \text{Sup}(Q, G)$  for any  $G$ , to facilitate search space pruning. Various pattern-based anti-monotonic support metrics exist, e.g., **Minimum-Image-based Support** (MNIS) (Bringmann & Nijssen, 2008), harmful overlap (Fiedler & Borgelt, 2007) and maximum independent sets (Gudes, Shimony, & Vanetik, 2006). In this paper, MNIS is chosen as the support metric owing to the merit of fast evaluation.

Formally, the metric is defined as,

$$\text{Sup}(Q, G) = \min \{ |\text{IMG}(u)| \mid u \in V_p \}, \quad (1)$$

where  $\text{IMG}(u)$  is the image of a pattern node  $u$  in  $G$ .

*Example 2.* Recall graph  $G$ , pattern  $Q_1$  and its matches in Fig. 1. It is easy to see that  $\text{IMG}(\text{DBA}) = \{v_8, v_9, v_{19}, v_{20}, v_{21}\}$ ,  $\text{IMG}(\text{ST}) = \{v_{11}, v_{13}, v_{22}, v_{23}, v_{24}\}$ ,  $\text{IMG}(\text{PRG}) = \{v_{10}, v_{12}, v_{25}, v_{26}, v_{27}, v_{28}\}$ , which leads to  $\text{Sup}(Q_1, G) = 5$ .  $\square$

**Definition 6. Frequent Pattern Mining.** *Given a graph  $G$  and an integer  $\theta$  as the support threshold, it is to discover a set  $\mathbb{S}$  of frequent patterns  $Q$  in  $G$  such that  $\text{Sup}(Q, G) \geq \theta$  for any  $Q$  in  $\mathbb{S}$ .*  $\square$

In practice, the task of FPM faces three challenges: (1) the underlying graphs  $G$  are typically very large, and in the meanwhile, the FPM problem is intractable, it is hence very costly to identify all the frequent patterns on such large graphs; (2) there may return excessive patterns which bring trouble to users' inspection

and application, moreover people are more interested in those patterns which are top ranked (X.-F. Yan & Han, 2003); and (3) it is not easy to set a viable support threshold  $\theta$ , because a large (resp. small)  $\theta$  will lead to too few (resp. many) patterns (X.-F. Yan & Han, 2003). In light of these, it is necessary to investigate the *top-k pattern mining* problem. While, to do this, it is crucial to develop a metric for measuring the interestingness of a pattern.

Existing metrics for measuring patterns’ interestingness can be divided into two types: subjective metrics and objective metrics. A formalization of subjective metric was first introduced by (van Leeuwen, Bie, Spyropoulou, & Mesnage, 2016), followed by several similar counterparts. All these metrics, however, are based on information theory and are computationally expensive. In contrast, objective metrics (X.-F. Yan & Han, 2003; Huan et al., 2004; Chi, Xia, Yang, & Muntz, 2005) consider the structural containment relationship among patterns, on the basis of the “closeness” property, resulting in better efficiency. Inspired by the objective metrics, in this paper, we evaluate the interestingness of a pattern  $Q = (V_p, E_p, f_v)$  as,

$$\text{ITRS}(Q) = |Q| = |V_p| + |E_p|. \quad (2)$$

*Example 3.* Recall patterns  $Q_1$ ,  $Q_2$  and  $Q_3$  in Fig. 1 (b). One may verify that  $\text{ITRS}(Q_1) = 6$ ,  $\text{ITRS}(Q_2) = 8$  and  $\text{ITRS}(Q_3) = 10$ . Among three patterns,  $Q_3$  is considered more interesting, as it subsumes others; in addition, it is frequent entails that the others are frequent as well.  $\square$

Indeed, the metric is a simplified closeness-based metric, as it simplifies evaluation of pattern containment with pattern size. Moreover, it is cheaper to evaluate and can be adapted based on practical requirements, e.g., by integrating to developing a *top-k pattern mining* algorithm with *early termination* property.

**Problem formulation.** The TOPKPM problem is formalized as follows.

- Input: A single large graph  $G$ , support threshold  $\theta$  and integer  $k$ .
- Output: A set  $\mathbb{S}_k$  of patterns  $Q$  discovered from  $G$  such that  $|\mathbb{S}_k| \leq k$ ,  $\text{Sup}(Q, G) \geq \theta$  for any  $Q$  in  $\mathbb{S}_k$  and  $\arg \max_{\mathbb{S}_k \subseteq \mathbb{S}} \sum_{Q \in \mathbb{S}_k} \text{ITRS}(Q)$ .

Intuitively, the problem is to find a set of  $k$  (specified by users) patterns that not only satisfy support constraint but also take the largest sum of interestingness values. However, the problem is nontrivial.

**Proposition 1:** *The decision problem of TOPKPM is NP-hard.*  $\square$

To see Prop. 1, observe that the subgraph isomorphism (ISO) problem is embedded in TOPKPM problem, thus TOPKPM problem must be at least as hard as ISO problem. Since ISO is an NP-complete problem (Cordella et al., 2004), thus TOPKPM problem must be NP-hard.

To tackle the issue, one may develop an algorithm (NAIVE) that applies a “find-all-select” strategy to identify top- $k$  patterns. In a nutshell, NAIVE discovers a complete set  $\mathbb{S}$  of frequent patterns by using any existing frequent pattern

mining algorithm, ranks frequent patterns according to their interestingness values and picks  $k$  best ones. Though straightforward, NAIVE has to mine all the frequent patterns, hence is prohibitively expensive and even not doable on large graphs. To rectify this, one can incorporate both early termination strategy and approximation scheme. We next illustrate more in Section 4.

## 4 Mining Near-Optimal Top- $k$ Patterns

In this section, we first outline an algorithm that preserves early termination property, for identifying near-optimal top- $k$  patterns. We then present a novel method for estimating MNIS.

### 4.1 Mining with Early Termination

By Prop. 1, we know that identifying the optimal top- $k$  patterns requires extremely high computational costs, which is infeasible in practice. Hence, an algorithm that is able to efficiently discover near-optimal top- $k$  patterns is more desired. Motivated by this, we develop such an algorithm, denoted as APRTOPK.

In contrast to traditional methods, APRTOPK works in an incremental manner to identify top- $k$  patterns, during the period, compact data structures are used for estimating pattern supports. These together significantly lower both computational and space costs while retaining near-optimal recall.

**Framework.** As shown in the Pseudo-Code in Algorithm 1, APRTOPK takes a single (possibly large) graph  $G$ , a support threshold  $\theta$ , an integer  $k$  and a parameter  $m$  as input and returns a set  $\mathbb{S}_k$  of qualified patterns that are close to the optimal solution as output. Here, parameter  $m$  is used to limit the operation times of procedure NODECHOOSE (see Eq. 3), thereby improving efficiency. During mining, APRTOPK performs three main tasks, i.e., Initialization (lines 1-2), Tree patterns identification (lines 3-9), and Top- $k$  patterns mining (line 10). All the frequent patterns are organized in a directed tree  $\mathcal{T}$ , which is dynamically maintained. In particular, the growth of  $\mathcal{T}$  follows a bottom-up manner, starting from “seed” patterns (see below for explanations). To simplify discussion, we use “parent” (resp. “child”) to denote relationship of two patterns which correspond to parent-child nodes in  $\mathcal{T}$ .

*Initialization.* Four parameters are initialized, i.e., a boolean variable  $flag$  to control **while** loop, an empty set  $\mathbb{S}_k$  for keeping track of top- $k$  patterns, an empty set  $L$  for maintaining candidate patterns and an empty tree  $\mathcal{T}$  to record frequent patterns (line 1). Later on, frequent single-edge patterns (*a.k.a.* “seed patterns”) are identified. They are included in a set  $fEdges$  and used to update  $\mathcal{T}$  (line 2). Note that, after initialization, tree  $\mathcal{T}$  is consisted of isolated nodes that correspond to “seed patterns” in  $fEdges$ .

*Tree patterns identification.* In this stage, APRTOPK iteratively identifies frequent “tree” patterns, following a *level-wise* strategy (lines 3-9). In each round

---

**Algorithm 1** APRTOPK

---

*Input:* Graph  $G$ , support threshold  $\theta$ , integers  $k$  and  $m$ .

*Output:* A set of no more than  $k$  patterns.

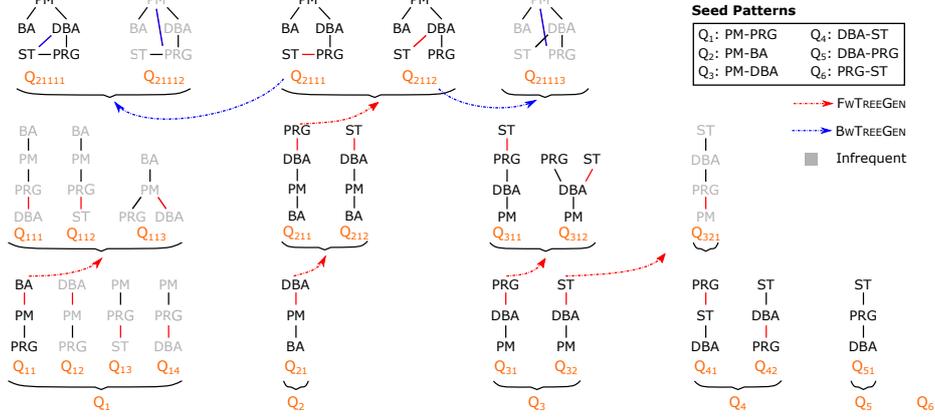
```
1: initialize  $flag := \mathbf{false}$ ;  $\mathbb{S}_k := \emptyset$ ;  $L := \emptyset$ ;  $\mathcal{T}$  as an empty tree;
2: initialize  $fEdges$ ; update  $\mathcal{T}$ ;
3: while  $flag \neq \mathbf{true}$  do
4:    $L := \text{FWTREEGEN}(fEdges, \mathcal{T})$ ;
5:   for each pattern  $Q_c$  in  $L$  do
6:     if  $\text{FRQCHK}(G, Q_c, D(Q_p), \theta, m) \geq \theta$  then
7:       update  $\mathcal{T}$  with  $Q_c$ ;
8:   if  $\mathcal{T}$  was not updated then
9:      $flag := \mathbf{true}$ ;
10:  $\mathbb{S}_k := \text{ETSEARCH}(\mathcal{T}, \theta, k, m)$ ;
11: return  $\mathbb{S}_k$ ;

12: function  $\text{ETSEARCH}(\mathcal{T}, \theta, k, m)$ 
13:   initialize  $Terminate := \mathbf{false}$ ;  $\mathbb{S}_k := \emptyset$ ;  $L := \emptyset$ ;  $h$  as the height of  $\mathcal{T}$ ;
14:   while  $Terminate \neq \mathbf{true}$  do
15:     for each  $v$  at level  $h$  of  $\mathcal{T}$  do
16:        $L := \text{BWTREEGEN}(Q_{[v]}, fEdges)$ ;
17:       for each pattern  $Q_c$  in  $L$  do
18:         if  $\text{FRQCHK}(G, Q_c, D(Q_p), \theta, m) \geq \theta$  then
19:            $\mathbb{S}_k := \mathbb{S}_k \cup \{Q_c\}$ ;
20:         if termination condition is satisfied then
21:            $Terminate := \mathbf{true}$ ;
22:           update  $\mathbb{S}_k$ ;
23:           break;
24:   update  $h$ ;
25:   return  $\mathbb{S}_k$ ;
```

---

iteration, APRTOPK performs as follows. (1) It generates a set  $L$  of “tree” patterns as candidates with procedure FWTREEGEN (line 4). Note that FWTREEGEN (not shown) produces candidate patterns by expanding “tree” patterns that locate at the top level of  $\mathcal{T}$  with “seed patterns”, following forward expansion (See Def. 4). (2) For each candidate pattern  $Q_c$ , APRTOPK employs a procedure FRQCHK to calculate its support and updates  $\mathcal{T}$  with  $Q_c$  if it is frequent (lines 6-7). The details of FRQCHK for support estimation will be given shortly. (3) After above process, if  $\mathcal{T}$  remains unchanged, the flag  $flag$  is then updated as **true**, indicating that no new pattern was generated and the **while** loop no longer needs to continue (lines 8-9). By now,  $\mathcal{T}$  grows into a tree with nodes corresponding to frequent tree patterns, and will be used for further processing.

*Example 4.* On graph  $G$  of Fig. 1 (a), APRTOPK first identifies frequent edges as seed patterns  $Q_1$ - $Q_6$  (shown in Fig. 2), as their supports are no less than 3. Then, APRTOPK applies FWTREEGEN to generate candidate patterns following *forward expansion*, in a *level-by-level* manner. For example, using pattern



**Fig. 2.** Growth of  $\mathcal{T}$ , via *forward* and *backward* expansions. Infrequent patterns are marked in grey.

$Q_1$ , APRTOPK generates candidate patterns by enlarging  $Q_1$  with other frequent “seed” patterns and produces  $L = \{Q_{11}, Q_{12}, Q_{13}, Q_{14}\}$ . As patterns in  $L$  are generated via *forward expansion*, their forward edges in Fig. 2 are marked in red. Four levels of “nontrivial” candidate patterns (patterns without duplicate node labels) are listed in Fig. 2.  $\square$

*Top-k patterns mining.* Based on frequent tree patterns, the procedure ETSEARCH is employed to discover top- $k$  patterns. Specifically, ETSEARCH first initializes necessary parameters: a Boolean variable *Terminate* as a flag for loop control, an empty set  $\mathbb{S}_k$  to store  $k$  chosen patterns and an integer  $h$  as the height of  $\mathcal{T}$  (line 13). ETSEARCH then iteratively generates non-tree patterns and identifies top- $k$  ones, where the pattern generation process starts from tree-patterns located at the top level of  $\mathcal{T}$ , and follows a top-down manner (lines 14-24). In each round iteration, ETSEARCH selects a node  $v$  (corresponding to a “tree” pattern  $Q_{[v]}$ ) at level  $h$  of  $\mathcal{T}$ , and generates a set  $L$  of candidate patterns with BWTREEGEN (line 16). Note that BWTREEGEN (not shown) works along the same line as FWTREEGEN, but only enlarges a pattern  $Q_{[v]}$  with “seed patterns” via *backward expansion* (See Def. 4). For each candidate pattern  $Q_c$  in  $L$ , ETSEARCH verifies its support still with FRQCHK and enlarges  $\mathbb{S}_k$  with  $Q_c$  if it is a qualified pattern (line 19). ETSEARCH next verifies whether the termination condition, specified by Proposition 2, is satisfied (line 20).

**Proposition 2:** *Given parameters  $\theta$ ,  $k$  and a tree  $\mathcal{T}$ , whose nodes correspond to the set  $\mathbb{S}^t$  of frequent “tree” patterns, a  $k$ -element set  $\mathbb{S}_k$  is the top- $k$  pattern set, if (1)  $\text{Sup}(Q, G) \geq \theta$  for each  $Q$  in  $\mathbb{S}_k$ , and (2)  $\min\{\text{ITRS}(Q) | Q \in \mathbb{S}_k\} \geq \max\{\text{ITRS}(\widehat{Q}_t) | Q_t \in \overline{\mathbb{S}^t}\}$ .  $\square$*

Here,  $\overline{\mathbb{S}^t}$  is a subset of  $\mathbb{S}^t$  and includes those tree patterns that have not been used for pattern expansion, and  $\widehat{Q}_t$  is a *complete* pattern of a tree pattern  $Q_t$  in

$\overline{\mathbb{S}}^t$ . Observe that  $\text{ITRS}(\widehat{Q}_t)$  must be larger than interestingness value of any other pattern that is expanded from  $Q_t$ , as a result, when the minimum interestingness value of a pattern  $Q$  in  $\mathbb{S}_k$  is already no less than the maximum interestingness value of the complete pattern of a tree pattern  $Q_t$  in  $\overline{\mathbb{S}}^t$ , then  $\sum_{Q \in \mathbb{S}_k} \text{ITRS}(Q)$  is already maximized and no further exploration is needed.

Indeed, Prop. 2 enables algorithm APRTOPK to terminate earlier. As top- $k$  patterns mining are performed in a top-down manner, the set  $\overline{\mathbb{S}}^t$  (initially the same as  $\mathbb{S}^t$ ) is hence dynamically updated with a seen  $Q_t$ .

If the termination condition is satisfied, ETSEARCH sets *Terminate* as **true**, eliminates redundant patterns in  $\mathbb{S}_k$  if  $|\mathbb{S}_k| > k$ , breaks the **while** loop (lines 21-23) and returns  $\mathbb{S}_k$  as final result (line 25). Otherwise, ETSEARCH decreases  $h$  by 1 (line 24), indicating that a new round selection will start from level  $h-1$  of  $\mathcal{T}$ .

*Example 5.* Recall Example 4. To mine the top-1 pattern on graph  $G$  of Fig. 1 (a), a pattern  $Q_{21111}$  is generated via backward expansion (marked in blue line for backward edges) from its parent  $Q_{2111}$  at level 4. ETSEARCH then evaluates its support and enlarges  $\mathbb{S}_k$  with it. The above process terminates until candidates generated from patterns at level 4 of  $\mathcal{T}$  (Fig. 2) are all processed, as the remaining candidates can not have higher ITRS values.  $\square$

## 4.2 Supports Evaluation

During mining, supports evaluation brings two challenges: (a) high computational cost, since it involves expensive isomorphism checking, which may even be performed exponentially many times; and (b) high space cost for recording all the matches and then calculating MNIS for each pattern. On large graphs, such high costs are often not affordable. This calls for effective methods to estimate pattern supports efficiently and accurately.

To this end, we introduce a novel method FRQCHK, which incorporates an approximation scheme for support estimation. The Pseudo-Code of FRQCHK is shown in Algorithm 2. We next introduce its details, starting from the auxiliary structures it uses.

*Auxiliary Structures.* To facilitate the calculation of MNI-based support, an auxiliary structure, called Domain, is used to keep track of matches of a pattern.

**Definition 7. Domain.** *Given a graph  $G$  and a pattern  $Q$  with node set  $V_p$ , the Domain of  $Q$ , denoted by  $D(Q)$ , reorganizes all the matches  $M(Q, G)$  of  $Q$  in  $G$  with a table, whose column head and body correspond to a pattern node  $u_i$  ( $u_i \in V_p$ ) and its image  $\text{IMG}(u_i)$ , respectively.  $\square$*

Abusing the notation of domain, we use  $D_i(Q)$  to indicate the  $i$ -th domain of  $D(Q)$ , which essentially corresponds to  $\text{IMG}(u_i)$ .

*Example 6.* As shown in Fig. 3, the match set  $M(Q_1, G) = \{(v_0, v_4, v_6), (v_0, v_4, v_7), (v_0, v_4, v_8), (v_1, v_4, v_6), (v_1, v_4, v_7), (v_1, v_4, v_8), (v_2, v_4, v_6), (v_2, v_4, v_7), (v_2, v_4, v_8), (v_1, v_5, v_7), (v_1, v_5, v_8), (v_1, v_5, v_9), (v_2, v_5, v_7), (v_2, v_5, v_8), (v_2, v_5, v_9), (v_3, v_5,$

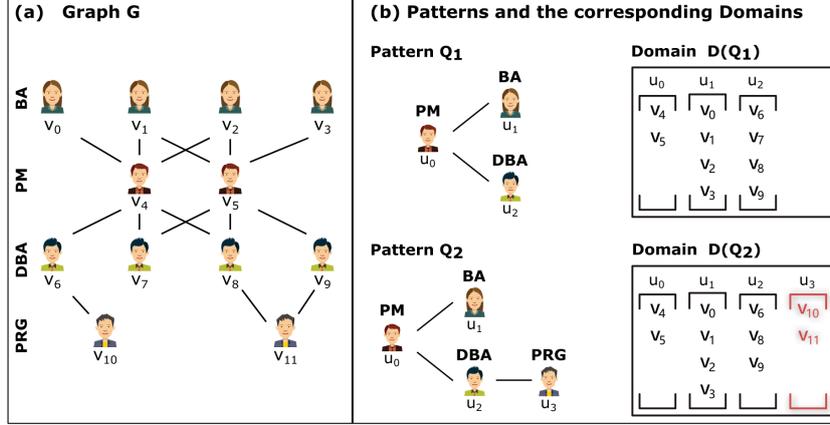


Fig. 3. A sample graph, typical patterns and their domains

$v_7$ ),  $(v_3, v_5, v_8)$ ,  $(v_3, v_5, v_9)$  includes in total 18 distinct matches of  $Q$  in  $G$ ; while the domain  $D(Q_1)$  of  $Q_1$  in  $G$ , shown in Fig. 3(b) is a more compact data structure, compared with  $M(Q_1, G)$ .  $\square$

Obviously, the domain of a pattern  $Q$  in a graph  $G$  is of linear size of  $|G|$  and  $|Q|$ , which is much smaller than  $M(Q, G)$  (potentially in exponential size of  $|G|$ ). Apart from compact structure, domains can be used for support estimation efficiently and accurately.

**Support Estimation.** The support estimation is fulfilled by the procedure FRQCHK, which leverages a recursive function TRAVERSE to update domains of candidate patterns. The Pseudo-Code of FRQCHK is shown in Algorithm 2. As can be seen, the input of FRQCHK includes a graph  $G$ , a candidate pattern  $Q_c$ , whose support needs to be verified, a pattern  $Q_p$  with node set  $V_{p_p}$  along with its domain  $D(Q_p)$ , a support threshold  $\theta$  and an integer  $m$ . Here  $Q_c$  is deemed as the “child” of  $Q_p$ , as the corresponding node of  $Q_c$  on  $\mathcal{T}$  is a child of that of  $Q_p$ . Indeed,  $Q_c$  is extended with an edge  $e_x = (u_i, u_j)$  from  $Q_p$ . If the expansion is a backward expansion, then  $u_j$  is already in  $Q_c$ , otherwise,  $u_j$  is a newly introduced node. As will be seen, the parameter  $m$  is involved for controlling candidate selection.

First of all, FRQCHK initializes an empty domain for  $Q_c$ , an empty stack  $S_d$  for loop and an integer *counter* as  $|D_0(Q_p)|$  for fast verification (line 1). As  $Q_c$  is extended from its *parent*  $Q_p$  with the edge  $e_x = (u_i, u_j)$ , FRQCHK utilizes this property to conduct a preliminary pruning by referencing  $e_x$ ,  $D(Q_p)$  and  $G$  (line 2). Specifically, FRQCHK checks each node  $v_k$  in  $D_i(Q_p)$  and see whether there exists an edge  $(v_k, v'_k)$  in  $G$ , where  $v_k$  is a *match* of  $u_i$  of  $Q_p$  and  $v'_k \sim u_j$ . If  $v_k$  does not have such a neighbor  $v'_k$ , then  $v_k$  can not be a *match* of  $u_i$  of  $Q_c$  and is marked with a special symbol indicating its invalidity.

*Example 7.* Taking  $G$ ,  $Q_1$  as well as its domain  $D(Q_1)$  given in Fig. 3 as input, FRQCHK first checks whether each node in  $D_2(Q_1)$  has a neighbor  $v$  such that

---

**Algorithm 2** FRQCHK

---

*Input:* Graph  $G$ , a pattern  $Q_c$ , the domain  $D(Q_p)$  of  $Q_p$ , parameters  $\theta$  and  $m$ .

*Output:* The minimum-image-based support MNIS of pattern  $Q_c$ .

```
1: initialize domain  $D(Q_c)$ , a stack  $S_d := \emptyset$ , an integer  $counter := |D_0(Q_p)|$ ;
2: update  $D(Q_p)$ ;
3: for each node  $v$  in  $D_0(Q_p)$  do
4:   restore  $S_d$ ;  $counter := counter - 1$ ;
5:    $D(Q_c) := \text{TRAVERSE}(G, Q_c, D(Q_c), D(Q_p), S_d)$ ;
6:   if  $|D_0(Q_c)| + counter < \theta$  then
7:     break;
8: calculate  $Sup(Q_c)$  with  $D(Q_c)$ ;
9: return  $Sup(Q_c)$ ;

10: function  $\text{TRAVERSE}(G, Q_c, D(Q_c), D(Q_p), S_d)$ 
11:    $H_c := \text{CONSEXTR}(G, Q_c, D(Q_p), S_d)$ ;  $c := 0$ ;
12:    $v := \text{NODECHOOSE}(H_c, D(Q_c), S_d, c)$ ;
13:   while  $v \neq null$  do
14:     update  $c$ ;
15:      $S_d.push(v)$ ;
16:     if  $|S_d| == |V_{p_p}|$  then
17:        $\text{EXPAND}(G, Q_c, D(Q_c), S_d)$ ;
18:     else
19:        $D(Q_c) := \text{TRAVERSE}(G, Q_c, D(Q_c), D(Q_p), S_d)$ ;
20:      $S_d.pop()$ ;
21:      $v := \text{NODECHOOSE}(H_c, D(Q_c), S_d, c)$ ;
22:   return  $D(Q_c)$ ;
```

---

$v \sim u_3$ , as  $Q_2$  is expanded with an edge  $e_x = (u_2, u_3)$  from  $Q_1$ . Then, node  $v_7$  is identified and marked as invalid, since it has no neighbor labeled as PRG.  $\square$

FRQCHK next iteratively updates  $D(Q_c)$  via guided traversal from each  $v$  in  $D_0(Q_p)$  (lines 3-7). During the iteration, FRQCHK restores the stack  $S_d$  by pushing  $v$  on top of it after cleaning, in addition, FRQCHK also decreases the *counter* by 1, indicating that  $v$  has been used for verification (line 4). Afterwards, FRQCHK calls  $\text{TRAVERSE}$  to identify qualified matches of  $Q_c$  (line 5, details of  $\text{TRAVERSE}$  will be elaborated shortly). After the traverse, an updated domain  $D(Q_c)$  is returned, FRQCHK then verifies the satisfiability of a simple rule, i.e.,  $|D_0(Q_c)| + counter < \theta$ . Intuitively, the rule states that if the total number of qualified matches of  $u_0$  of  $Q_c$  and unverified matches of  $u_0$  of  $Q_p$  is already less than  $\theta$ , then  $Sup(Q_c)$  must be less than  $\theta$  (property of MNIS). If the rule is satisfied, FRQCHK breaks the loop immediately, since further verification is no longer needed (line 7). When all the candidates of  $u_0$  are verified, FRQCHK calculates  $Sup(Q_c)$  by using  $D(Q_c)$  (line 8) and returns it as final result (line 9).

Procedure  $\text{TRAVERSE}$ . Recall that the **Minimum-Image-based Support** only concerns the image of each distinguished node of a candidate pattern  $Q_c$  rather than the total number of matches of  $Q_c$ . Thus, we do not need to enumerate all the

matches, but try to obtain a domain of  $Q_c$ , which is as accurate as possible. Based on this observation, our procedure applies a wise strategy to guide search accurately and economically. We next present details of TRAVERSE.

Given a stack  $S_d$  that contains match candidates, TRAVERSE works as follows.

**Stage(I):** Based on current status (determined by  $S_d$ ), TRAVERSE identifies a set of nodes  $H_c$  for further exploration with a procedure CONSEXTR (line 11). To do this, CONSEXTR (not shown) first identifies an *unvisited* edge  $e_u = (u_i, u_j)$  of  $Q_c$  for guiding next step exploration. The identification of  $e_u$  is based on  $v_i$ , which locates at  $S_d$  and exists in  $D_i(Q_c)$  (the corresponding domain of  $u_i$ ). CONSEXTR next collects those nodes  $v_j$  in  $G$  (resp.  $D_j(Q_p)$ ), that are neighbors of  $v_i$  and satisfy  $v_j \sim u_j$  if  $e_u$  is a forward (resp. backward) edge. Note that the nodes in  $D(Q_p)$  that are marked with invalid symbols will be omitted. For each seen edge  $e_u$ , it is then marked as *visited* to avoid repeated visit. In addition, TRAVERSE also initializes a parameter  $c$  as 0 for controlling node selection.

**Stage(II):** TRAVERSE picks a node from  $H_c$ , with procedure NODECHOOSE (not shown) by using below selection criteria (line 12).

$$v := \begin{cases} v_h \in H_c \setminus D(Q_c), & H_c \setminus D(Q_c) \neq \emptyset & (A) \\ v_h \in H_c \cap D(Q_c), & H_c \setminus D(Q_c) = \emptyset \wedge S_d \not\subseteq D(Q_c) \wedge c < m & (B) \\ null, & \text{otherwise} & (C) \end{cases} \quad (3)$$

Intuitively, Condition A states that NODECHOOSE prefers to pick a node  $v$  that is not in  $D(Q_c)$ . The reason for the preference lies in that a node that is not in  $D(Q_c)$  is beneficial to enlarge  $Sup(Q_c)$ , since an unvisited node will lead the traversal to a large part of unvisited area with higher possibility. If  $H_c$  is already contained by  $D(Q_c)$  (i.e.,  $H_c \setminus D(Q_c) = \emptyset$ ), NODECHOOSE selects a node from  $H_c \cap D(Q_c)$  if Condition B is satisfied. Here, Condition B enforces extra two restrictions i.e.,  $S_d \not\subseteq D(Q_c)$  and  $c < m$ . For the first restriction, it requires that  $S_d$  should contain nodes that are not in  $D(Q_c)$ , since otherwise, current traversal will not bring any new element to  $D(Q_c)$ . The second restriction imposes a number constraint, that is TRAVERSE only picks no more than  $m$  nodes from  $H_c \cap D(Q_c)$ . The number of selected nodes is recorded by a parameter  $c$ , which is updated when  $v$  is used for next round traversal (line 14). By introducing an adjustable parameter  $m$ , the exploration area at current iteration is restricted, thereby reducing the computational costs. When both of two conditions can not be satisfied, NODECHOOSE returns a null value. Indeed, the selection criteria given above effectively helps TRAVERSE to efficiently obtain a domain of  $Q_c$  with high quality.

*Example 8.* Continuing Example 7, FRQCHK restores stack  $S_d$  by pushing  $v_4$  onto it and calls TRAVERSE to update  $D(Q_2)$  in a depth-first manner. Firstly, CONSEXTR is invoked. It identifies an *unseen* edge  $e_u = (u_0, u_1)$ , and obtains a set  $H_c = \{v_0, v_1, v_2\}$ , as  $v_4 \in D_0(Q_1)$  and  $v_i \sim u_1$  ( $i \in [0, 2]$ ). Afterwards, TRAVERSE calls NODECHOOSE to pick a node for further exploration, and  $v_0$  is chosen due to Condition A. TRAVERSE next calls itself for traverse at a deeper level.  $\square$

**Stage(III):** Starting from a valid node  $v$ , TRAVERSE proceeds by referencing  $S_d$  (lines 13-21). During the traversal, it first updates  $c$  as  $c+1$  if  $v$  is picked from  $H_c \cap$

$D(Q_c)$  (line 14). Then, it keeps detecting a discriminant condition  $|S_d| == |V_{p_p}|$  and invokes EXPAND to update  $D(Q_c)$  if the condition is satisfied (lines 16-17).

Procedure EXPAND (not shown) works as follows. If  $Q_c$  is generated with  $e_x = (u_i, u_j)$  via *forward expansion*, EXPAND searches neighbors  $v'$  of  $v$ , where  $v \in S_d$ ,  $v \sim u_i$  and  $v' \sim u_j$ , and puts  $v'$  along with nodes in  $S_d$  in corresponding domain of  $D(Q_c)$ , as these nodes together already form valid matches of  $Q_c$ . For *backward expansion*, EXPAND verifies whether nodes in  $S_d$  already form a match of  $Q_c$  by referencing  $e_x$  and  $D_i(Q_c)$ ,  $D_j(Q_c)$ . If true, it updates  $D(Q_c)$  along the same line as that for *forward expansion*.

*Example 9.* Recall Examples 7 and 8. At a deeper level, an *unseen* edge  $e_u = (u_0, u_2)$  is used to guide traversal. Then  $H_c = \{v_6, v_7, v_8\}$  is obtained as they are neighbors of  $v_4$  that is in  $S_d$ . NODECHOOSE then picks  $v_6$  (Condition A) and pushes it onto  $S_d$ . The status of  $S_d$  is shown in Fig. 4 (a) (right most stack). At this moment, EXPAND is invoked, as the discriminant condition  $|S_d| == |V_{p_p}|$  is satisfied. EXPAND identifies a node  $v_{10}$  (as the neighbor of  $v_6$ ) such that  $v_{10} \sim u_3$  of  $Q_2$ , according to the forward edge  $(u_2, u_3)$ . Then, a valid match of  $Q_2$  forms. EXPAND puts  $v_{10}$  as well as nodes in  $S_d$  in corresponding columns of  $D(Q_2)$ . The updated  $D(Q_2)$  is depicted in Fig.4(a).  $\square$

Otherwise, TRAVERSE invokes itself for deeper exploration (line 18). Afterwards, TRAVERSE pops the upper-most node of  $S_d$  (line 20), and picks a different node with NODECHOOSE for next round exploration (line 21).

*Example 10.* Following previous examples, TRAVERSE pops  $v_6$ , which is the upper-most node, out of  $S_d$  and picks  $v_8$  (Condition A) for next round iteration. Figure 4 (b) shows how  $S_d$  is changed, where  $S_d$  after popup operation is colored in blue. As  $|S_d|$  equals to  $|V_{p_p}|$  now, EXPAND found that a new match can be formed with  $v_{11}$  and updates  $D(Q_2)$  with the new match. The updated  $D(Q_2)$  is also shown in the middle of Fig. 4 (b).

One may refer to Fig. 4 for the entire process of FRQCHK, where detailed changes of stack  $S_d$ , domain  $D(Q_2)$ , and traversal paths are provided. Detailed explanation is omitted due to space constraint.  $\square$

**Remarks.** (1) The parameter  $m$  controls total number of nodes used for exploration. As verified via experimental studies, guided traversal with limited numbers substantially improves efficiency of mining process, taking only 11.7% time and 31.5% memory of its counterpart, while obtaining 100% recall. (2) When performing support estimation for a candidate pattern  $Q_c$ , FRQCHK leverages the cached domain of  $Q_p$  (as the “parent” of  $Q_c$  and computed in earlier), which substantially improves efficiency. (3) Note that the decision problem of TOPKPM is already NP-hard, so no matter how desired, the TOPKPM problem can not be solved in PTIME. Despite high computational cost, APRTOPK works more efficiently than its counterparts, owing to its approximation scheme for supports estimation and early termination property.

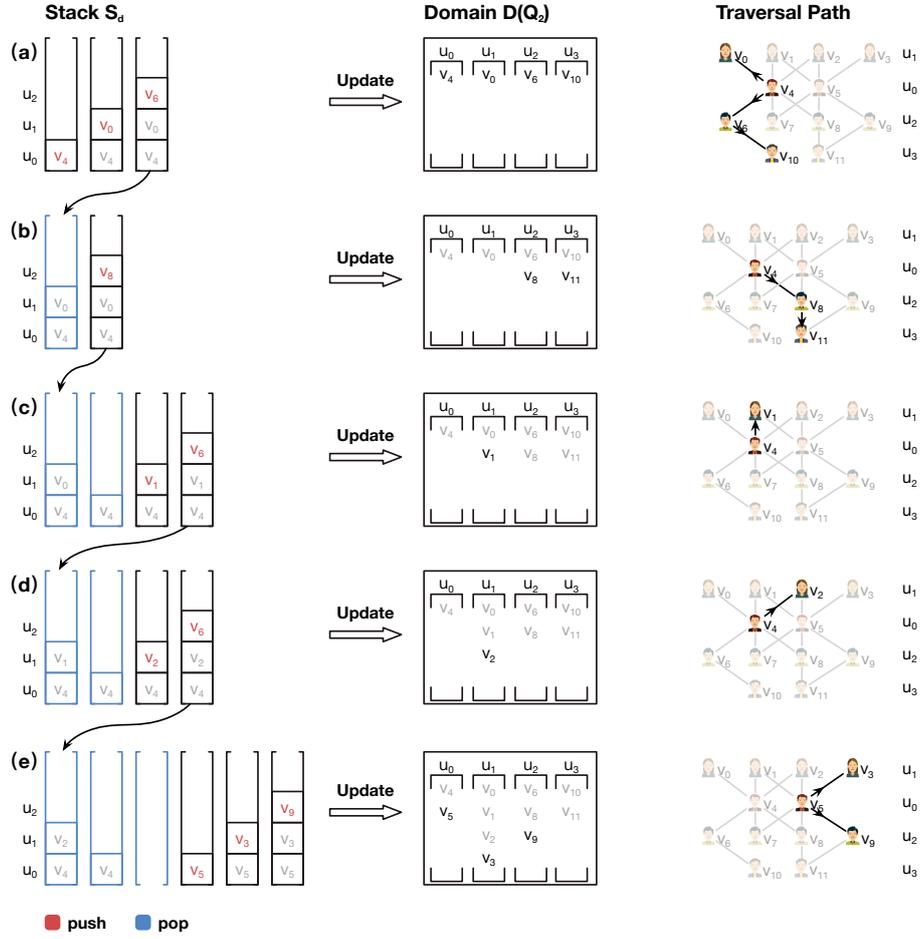


Fig. 4. FRQCHK Running Process

## 5 Experimental Study

Using real-life graphs and synthetic data, we conducted comprehensive experimental studies to evaluate: efficiency, memory cost, effectiveness (recall) and scalability of our algorithm APRTOPK, compared with baseline methods.

### 5.1 Experimental setting

**Real-life graphs.** We used three real-life graphs: (a) *Amazon* (Leskovec, Adamic, & Huberman, 2007), a product co-purchasing network with 0.41 million nodes and 3.35 million edges. (b) *Mico* (Elseidy et al., 2014), a dataset models the Microsoft co-authorship information with 0.1 million nodes and 1.08 million edges.

(c) *Youtube* (Cheng, Dale, & Liu, 2008), a network of videos and their related videos from *Youtube* with 0.15 million nodes and 1.05 million edges.

**Synthetic graphs.** We also designed a generator to produce synthetic graphs  $G = (V, E, L)$ , controlled by the numbers of nodes  $|V|$  and the number of edges  $|E|$ , where  $L$  is taken from an alphabet of  $1K$  labels. We generated synthetic graphs following the evolution model (Garg, Gupta, Carlsson, & Mahanti, 2009): an edge was attached to the high degree nodes with higher probability. The size of  $G$  is up to 0.5 million nodes and 5 million edges.

**Implementations.** We implemented algorithm APRTOPK and the following counterparts, all in Java.

- GRAMI, which identifies frequent patterns with the algorithm in (Elseidy et al., 2014), ranks patterns based on our interestingness metric and selects top- $k$  ones.
- AGRAMI, the approximate version of GRAMI. Along the same line as GRAMI, AGRAMI first discovers frequent patterns with the approximate version of GRAMI. During this period, it sets the time-out to occur after  $f(\alpha)$  iterations of the search, where  $f(\alpha) = \alpha^n \prod_1^n |D_i| + \beta$ ,  $\alpha \in (0, 1]$  is a user-defined parameter,  $\beta$  is a constant,  $D_i$  is the image  $\text{IMG}(u_i)$  of pattern node  $u_i$  and  $n$  is the number of pattern nodes. In this way, AGRAMI achieves better efficiency, at the cost of missing false negatives. After frequent patterns are discovered, the top- $k$  pattern selection is processed in the same way as GRAMI.

In our test, the testbed includes a machine with 2.3 GHz CPU and 16 GB RAM, running JDK v11.0.9 on Windows 10. Each test was run five times and the average is reported.

**Parameters.** For APRTOPK, we fixed parameter  $m = 2$  (used in procedure NODECHOOSE). For AGRAMI, we fixed  $\alpha$  as  $2 * 10^{-5}$ ,  $2 * 10^{-4}$ ,  $7 * 10^{-3}$  and  $10^{-5}$  on *Amazon*, *Mico*, *Youtube* and synthetic graphs, respectively.

## 5.2 Experimental results

**Exp-1: Influence of  $\theta$ .** To see the influence of  $\theta$ , we fixed  $k$  as a large number. It is a fair setting, since a complete (resp. incomplete but still large) set of frequent patterns need to be mined in GRAMI (resp. AGRAMI) in spite of the increase of  $k$ , while the increase of  $k$  weakens APRTOPK.

We then varied the support threshold  $\theta$  from  $2K$  to  $4K$  in  $0.5K$  increments,  $2.9K$  to  $3.3K$  in  $0.1K$  increments and  $0.3K$  to  $0.7K$  in  $0.1K$  increments on *Amazon*, *Mico* and *Youtube*, respectively.

Efficiency. Figures 5(a)-5(c) report the response time of all the algorithms on *Amazon*, *Mico* and *Youtube*, respectively, which tell us the following. (1) With the increase of support threshold  $\theta$ , all the algorithms take shorter time, because fewer candidate patterns and their matches have to be verified. (2) APRTOPK

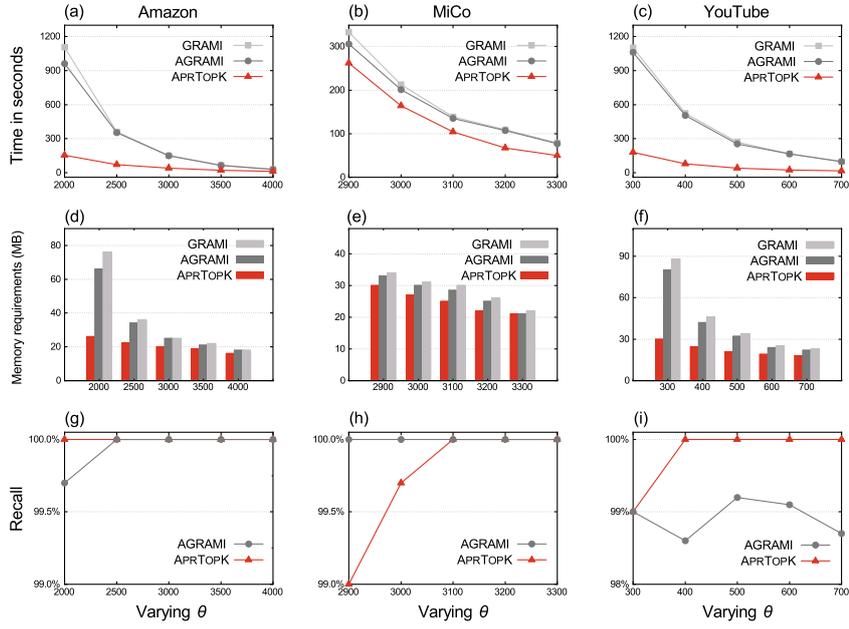


Fig. 5. Exp-1: Influence of  $\theta$

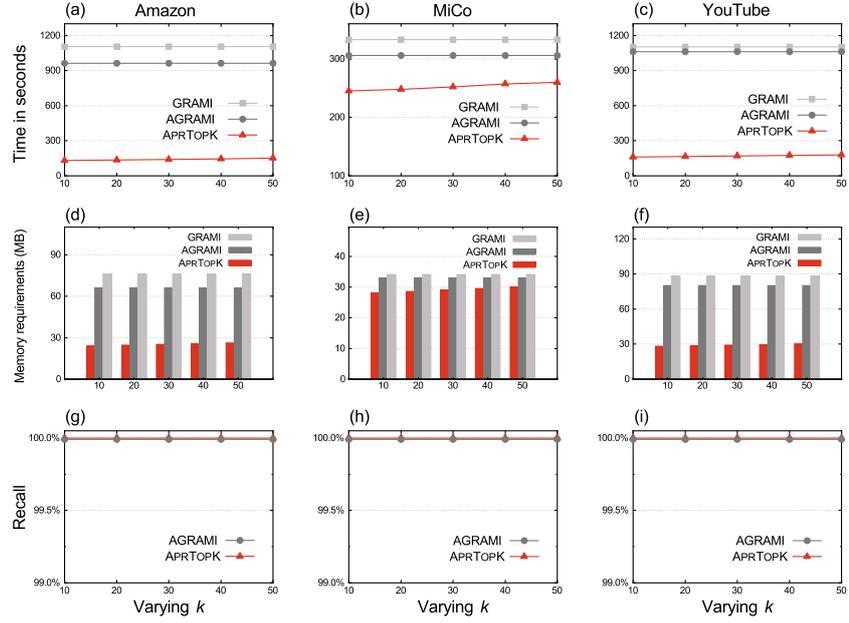


Fig. 6. Exp-2: Influence of  $k$

outperforms GRAMI and AGRAMI in all cases and is less sensitive to the increase of  $\theta$ , since APRTOPK is able to dramatically reduce the cost for candidate patterns verification. On *Amazon*, *Mico* and *Youtube* APRTOPK only takes on average 17.4%, 74.1% and 15.8% time of GRAMI, respectively. In particular, our algorithm takes only 13.8% time of GRAMI, when  $\theta = 2K$ , while obtaining 100% recall, on *Amazon*.

Memory cost. Figures 5(d)-5(f) show the memory footprint of the algorithms over *Amazon*, *Mico* and *Youtube*, respectively. We find that (1) the memory cost of all the algorithms drops with the increase of  $\theta$ , as fewer candidates need to be verified. (2) APRTOPK consumes less memory than GRAMI and AGRAMI on three graphs, as expected; it incurs 59.6%, 87.4% and 52.3% memory cost of GRAMI, on average, at *Amazon*, *Mico* and *Youtube*, respectively.

Recall. Figures 5(g)-5(i) show the recall of APRTOPK and AGRAMI, i.e., the ratio of patterns returned by APRTOPK and AGRAMI vs. the complete set of frequent patterns. We find the following: (1) the recall of APRTOPK is always higher than 99% on three graphs. In particular, the recall of APRTOPK even maintains 100% on *Amazon*. (2) Overall, when  $\theta$  grows, the recall of both algorithms grows as well (not monotonically increasing). This is because, for a large  $\theta$ , the set of frequent patterns becomes smaller, which favors top- $k$  selection. (3) The recall of AGRAMI is influenced not only by  $\theta$ , but also by a set of parameters (e.g.,  $\alpha$  etc.). We have tested AGRAMI with smaller  $\alpha$  and find that its recall and efficiency are mutually restricted. Due to space constraint, we omit details here.

It is noted that, FRQCHK gets less efficiency and memory cost advantages on *Mico*. The main reason lies in that the effectiveness of traversal strategy on *Mico* is not as good as that on *Amazon* and *Youtube*. Indeed, FRQCHK prefers to traverse from nodes that have not been seen since these unseen nodes can contribute the support of a pattern. This underlying feature results in that FRQCHK can achieve better performance to evaluate supports of those patterns whose corresponding matches have a large part of overlap (see Example 6). As the graph structure of *Mico* does not favor FRQCHK very well from the perspective of match overlap, the performance gains from FRQCHK hence become less.

**Exp-2: Influence of  $k$ .** Fixing  $\theta = 2K$ ,  $2.9K$  and  $0.3K$  for *Amazon*, *Mico* and *Youtube*, respectively, we varied  $k$  from 10 to 50 in 10 increments and compared APRTOPK with GRAMI and AGRAMI.

Efficiency. Results shown in Figures 6(a)-6(c) tell us the following. (1) APRTOPK performs much more efficiently than GRAMI, owing to its approximation scheme employed by support estimation and *early termination* property. For example, APRTOPK only takes on average 13.8%, 78.6% and 16.4% time of GRAMI at *Amazon*, *Mico* and *Youtube*, respectively. (2) APRTOPK is sensitive to the increase of  $k$  since it has to verify more candidate patterns before the termination condition can be satisfied, while GRAMI and AGRAMI are not influenced by  $k$  w.r.t. efficiency, as both of them apply the “find-all-select” strategy.

Memory cost. Figures 6(d)-6(f) show the memory footprint of all the algorithms. We find the following. All the algorithms are not sensitive to the varying of  $k$ .

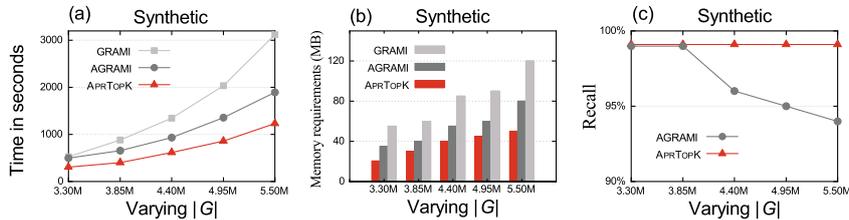


Fig. 7. Exp-3: Scalability

The reasons are twofold: (1) GRAMI and AGRAMI are almost not influenced by the change of  $k$ , hence the memory requirements remain unchanged for both of them; and (2) APRTOPK stops only when termination condition is satisfied, however a larger  $k$  does not dramatically increase the memory cost, meanwhile we find that APRTOPK consumes, on average, 34.2% (resp. 88.2%, 34.1%) memory of GRAMI, on *Amazon* (resp. *Mico* and *Youtube*).

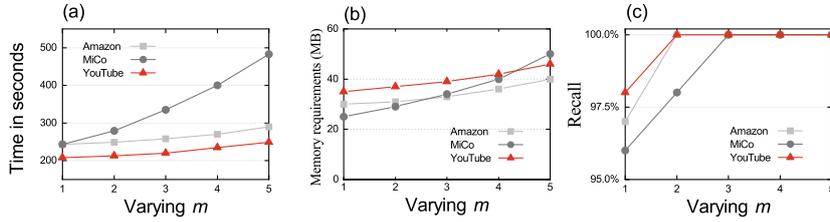
*Recall.* Figures 6(g)-6(i) report the recall of APRTOPK and AGRAMI. We find the following. Both APRTOPK and AGRAMI perform very well on all datasets and the recall of them remains 100% when  $k$  increases from 10 to 50.

**Exp-3: Scalability.** Fixing  $\theta = 1K$  and  $k = 50$ , we varied  $|G|$  from  $(0.3M, 3M)$  to  $(0.5M, 5M)$  with  $0.05M$  and  $0.5M$  increments on  $|V|$  and  $|E|$ , respectively, and compared APRTOPK with GRAMI and AGRAMI. As shown in Figures 7(a)-7(c), (1) all the algorithms take longer time and consume more memory on larger graphs, as expected; (2) APRTOPK is less sensitive to  $|G|$  than others, *w.r.t.* response time and memory footprint, showing its better scalability; and (3) APRTOPK shows a more steady recall than AGRAMI, with the increase of  $|G|$ .

**Exp-4: Influence of  $m$ .** Fixing  $k = 100$  and  $\theta = 1.8K, 2.9K$  and  $0.29K$  for *Amazon*, *Mico* and *Youtube*, respectively, we varied  $m$  from 1 to 5 in 1 increments to test its influence *w.r.t.* efficiency, memory cost and recall for APRTOPK. As shown in Figures 8(a)-8(c), (1) with the increase of  $m$ , the time overhead and memory cost of APRTOPK grow up as well; (2) on *Amazon* and *Youtube* (resp. *Mico*), when  $m$  reaches 2 (resp. 3), the improvement on recall becomes insignificant, since recall values already approach 100%. Hence, it is more appropriate to set  $m$  as 2 or 3 on real life graphs.

## 6 Conclusion

In this paper, we developed an approach to mining *near optimal* top- $k$  patterns. We first formalize the TOPKPM problem by incorporating viable metrics to measure support and interestingness of patterns. We then develop an algorithm APRTOPK to identify top- $k$  patterns efficiently and accurately. The algorithm applies a “level-wise” strategy, which ensures *early termination property*, to discover top-ranked patterns that are not only frequent but also interesting. To fa-



**Fig. 8.** Exp-4: Influence of  $m$

to facilitate support evaluation, we devised a technique to compute the lower bound of support with smart traverse strategy and compact data structure. Our experimental study has verified the efficiency, memory footprint, recall and scalability of our algorithm. We hence contend that our approach yields a promising tool for big graph analysis.

The study of TOPKPM is still in its infancy. One direction concerns pruning technique that may lead to the decrease of costs (computational and space costs) without sacrificing recall. Metrics for measuring importance of patterns also need to investigate. Another interesting topic is to identify top- $k$  patterns with different matching semantics, e.g., graph simulation, inexact matching, etc. It is also worth extending APRTOPK under distributed scenario, to leverage parallel computation.

## CRedit authorship contribution statement

Xin Wang: Conceptualization, Methodology, Formal analysis, Writing – original draft, Writing – review & editing. Zhuo Lan: Methodology, Software, Writing – original draft, Visualization. Yu-Ang He: Software, Validation, Writing – original draft, Visualization. Yang Wang: Methodology, Supervision. Zhi-Gui Liu: Methodology, review & editing. Wen-Bo Xie: Conceptualization, Formal analysis, Supervision, Writing – review & editing.

## Acknowledgments

This work is supported by National Natural Science Foundation of China [grant number 62172102], and National Key Research and Development Program of China [grant number 2017YFA0700800], and Young Scholars Development Fund of SWPU [grant number 202199010142].

## References

- Abdelhamid, E., Abdelaziz, I., Kalnis, P., Khayyat, Z., & Jamour, F. (2016). ScaleMine: Scalable parallel frequent subgraph mining in a single large

- graph. In *Proceedings of the international conference for high performance computing, networking, storage and analysis* (pp. 716–727). New York, NY, USA: IEEE.
- Abdelhamid, E., Canim, M., Sadoghi, M., Bhattacharjee, B., Chang, Y., & Kalnis, P. (2017). Incremental frequent subgraph mining on large evolving graphs. *IEEE Transactions on Knowledge and Data Engineering*, 29(12), 2710–2723.
- 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.
- Ashraf, N., Haque, R. R., Islam, M. A., Ahmed, C. F., Leung, C. K., Mai, J. J., & Wodi, B. H. (2019). WeFreS: weighted frequent subgraph mining in a single large graph. In *Industrial conference on data mining* (pp. 201–215). New York, USA: ibai Publishing.
- Aslay, C., Nasir, M. A. U., De Francisci Morales, G., & Gionis, A. (2018). Mining frequent patterns in evolving graphs. In *Acm international conference on information and knowledge management* (pp. 923–932). New York, NY, USA: ACM.
- Bringmann, B., & Nijssen, S. (2008). What is frequent in a single graph? In *Pacific-asia conference on knowledge discovery and data mining* (pp. 858–863). Berlin Heidelberg: Springer.
- Chen, Y., Zhao, X., Lin, X., Wang, Y., & Guo, D. (2019). Efficient mining of frequent patterns on uncertain graphs. *IEEE Transactions on Knowledge and Data Engineering*, 31(2), 287–300.
- Cheng, X., Dale, C., & Liu, J. (2008). Statistics and social network of youtube videos. In *16th international workshop on quality of service* (pp. 229–238). Enschede, Netherlands: IEEE.
- Chi, Y., Xia, Y., Yang, Y., & Muntz, R. R. (2005). Mining closed and maximal frequent subtrees from databases of labeled rooted trees. *IEEE Transactions on Knowledge and Data Engineering*, 17(2), 190–202.
- Cordella, L. P., Foggia, P., Sansone, C., & Vento, M. (2004). A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10), 1367–1372.
- Daud, N. N., Ab Hamid, S. H., Saadoon, M., Sahran, F., & Anuar, N. B. (2020). Applications of link prediction in social networks: A review. *Journal of Network and Computer Applications*, 166, 102716.
- Driss, K., Boulila, W., Leborgne, A., & Gançarski, P. (2021). Mining frequent approximate patterns in large networks. *International Journal of Imaging Systems and Technology*, 31(3), 1265–1279.
- Elseidy, M., Abdelhamid, E., Skiadopoulos, S., & Kalnis, P. (2014). GRAMI: Frequent subgraph and pattern mining in a single large graph. *Proceedings of the VLDB Endowment*, 7(7), 517–528.
- Fiedler, M., & Borgelt, C. (2007). Subgraph support in a single large graph. In *IEEE international conference on data mining workshops* (pp. 399–404). IEEE Computer Society.

- Garg, S., Gupta, T., Carlsson, N., & Mahanti, A. (2009). Evolution of an online social aggregation network: an empirical study. In *ACM SIGCOMM conference on Internet measurement* (pp. 315–321). New York, NY, USA: ACM.
- Gudes, E., Shimony, S., & Vanetik, N. (2006). Discovering frequent graph patterns using disjoint paths. *IEEE Transactions on Knowledge and Data Engineering*, 18(11), 1441–1456.
- Huan, J., Wang, W., & Prins, J. (2003). Efficient mining of frequent subgraphs in the presence of isomorphism. In *IEEE international conference on data mining* (pp. 549–552). New York, USA: IEEE.
- Huan, J., Wang, W., Prins, J., & Yang, J. (2004). SPIN: Mining maximal frequent subgraphs from graph databases. In *ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 581–586). New York, NY, USA: ACM.
- 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.
- Le, N., Vo, B., Nguyen, L. B. Q., Fujita, H., & Le, B. (2020). Mining weighted subgraphs in a single large graph. *Information Sciences*, 514, 149–165.
- Le, T., Vo, B., Huynh, V., Nguyen, N. T., & Baik, S. W. (2020). Mining top-k frequent patterns from uncertain databases. *Applied Intelligence*, 50(5), 1487–1497.
- Leskovec, J., Adamic, L. A., & Huberman, B. A. (2007). The dynamics of viral marketing. *ACM Transactions on the Web*, 1(1), 5.
- Li, R., & Wang, W. (2015). REAFUM: Representative approximate frequent subgraph mining. In *SIAM international conference on data mining* (pp. 757–765). SIAM.
- Nasir, M. A. U., Aslay, C., Morales, G. D. F., & Riondato, M. (2021). Tiptap: Approximate mining of frequent k-subgraph patterns in evolving graphs. *ACM Transactions on Knowledge Discovery from Data*, 15(3), 1–35.
- Natarajan, D., & Ranu, S. (2018). Resling: a scalable and generic framework to mine top-k representative subgraph patterns. *Knowledge and Information Systems*, 54(1), 123–149.
- Nijssen, S., & Kok, J. N. (2004). A quickstart in frequent structure mining can make a difference. In *ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 647–652). New York, NY, USA: ACM.
- Prateek, A., Khan, A., Goyal, A., & Ranu, S. (2020). Mining top-k pairs of correlated subgraphs in a large network. *Proceedings of the VLDB Endowment*, 13(9), 1511–1524.
- Preti, G., De Francisci Morales, G., & Riondato, M. (2021). MaNIACS: Approximate mining of frequent subgraph patterns through sampling. In *ACM SIGKDD conference on knowledge discovery and data mining* (pp. 1348–1358). New York, NY, USA: ACM.
- Sabe, V. T., Ntombela, T., Jhamba, L. A., Maguire, G. E., Govender, T., Naicker, T., & Kruger, H. G. (2021). Current trends in computer aided drug design

- and a highlight of drugs discovered via computational techniques: A review. *European Journal of Medicinal Chemistry*, 224, 113705.
- Semertzidis, K., & Pitoura, E. (2019). Top- $k$  durable graph pattern queries on temporal graphs. *IEEE Transactions on Knowledge and Data Engineering*, 31(1), 181-194.
- Talukder, N., & Zaki, M. J. (2016). A distributed approach for graph mining in massive networks. *Data Mining and Knowledge Discovery*, 30(5), 1024–1052.
- Ur Rehman, S., Liu, K., Ali, T., Nawaz, A., & Fong, S. J. (2021). A graph mining approach for ranking and discovering the interesting frequent subgraph patterns. *International Journal of Computational Intelligence Systems*, 14(1), 152.
- van Leeuwen, M., Bie, T. D., Spyropoulou, E., & Mesnage, C. (2016). Subjective interestingness of subgraph patterns. *Machine Learning*, 105(1), 41–75.
- Wang, T., Huang, H., Lu, W., Peng, Z., & Du, X. (2018). Efficient and scalable mining of frequent subgraphs using distributed graph processing systems. In *Database systems for advanced applications* (pp. 891–907). Berlin, Heidelberg: Springer.
- Wang, X., Xiang, M., Zhan, H., Lan, Z., He, Y., He, Y., & Sha, Y. (2021). Distributed top- $k$  pattern mining. In *Web and big data* (pp. 203–220). Cham: Springer.
- Xue, Y., Klabjan, D., & Luo, Y. (2019). Predicting ICU readmission using grouped physiological and medication trends. *Artificial Intelligence in Medicine*, 95, 27-37.
- Yan, D., Qu, W., Guo, G., & Wang, X. (2020). PrefixFPM: A parallel framework for general-purpose frequent pattern mining. In *IEEE international conference on data engineering* (pp. 1938–1941). New York, NY, USA: IEEE.
- Yan, X., & Han, J. (2002). gSpan: Graph-based substructure pattern mining. In *IEEE international conference on data mining* (pp. 721–724). New York, NY, USA: IEEE.
- Yan, X.-F., & Han, J.-W. (2003). CloseGraph: Mining closed frequent graph patterns. In *ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 286–295). New York, NY, USA: ACM.
- Zeng, J., U, L. H., Yan, X., Han, M., & Tang, B. (2021). Fast core-based top- $k$  frequent pattern discovery in knowledge graphs. In *IEEE international conference on data engineering* (pp. 936–947). New York, NY, USA: IEEE.
- Zheng, T.-Y., & Wang, L. (2021). Large graph sampling algorithm for frequent subgraph mining. *IEEE Access*, 9, 88970-88980.
- Zhu, F., Qu, Q., Lo, D., Yan, X., Han, J., & Yu, P. S. (2011). Mining top- $k$  large structural patterns in a massive network. *Proceedings of the VLDB Endowment*, 4(11), 807–818.