IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received November 8, 2021, accepted November 27, 2021, date of publication December 7, 2021,
date of current version December 23, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3133666

A Method for Closed Frequent Subgraph Mining in a Single Large Graph

LAM B. Q. NGUYEN[1,2], LOAN T. T. NGUYEN[3,4], IVAN ZELINKA[2,5], (Member, IEEE), VACLAV SNASEL[2], (Senior Member, IEEE), HUNG SON NGUYEN[6], AND BAY VO[7]

[1]Faculty of Information and Communications, Kien Giang University, Kien Giang 920000, Vietnam
[2]Faculty of Electrical Engineering and Computer Science, VŠB-Technical University of Ostrava, 700 30 Ostrava, Poruba, Czech Republic
[3]School of Computer Science and Engineering, International University—VNU-HCM, Ho Chi Minh City 700000, Vietnam
[4]Vietnam National University, Ho Chi Minh City, Ho Chi Minh City 700000, Vietnam
[5]Faculty of Electrical and Electronics Engineering, Ton Duc Thang University, Ho Chi Minh City 700000, Vietnam
[6]Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, 00-927 Warsaw, Poland
[7]Faculty of Information Technology, HUTECH University, Ho Chi Minh City 700000, Vietnam

Corresponding authors: Loan T. T. Nguyen (nttloan@hcmiu.edu.vn) and Bay Vo (vd.bay@hutech.edu.vn)

This work was supported in part by the Institute for Computational Science and Technology (ICST)—Ho Chi Minh City, and in part by the Department of Science and Technology (DOST)—Ho Chi Minh City under Grant 23/2021/HĐ-QKHCN.

ABSTRACT Mining frequent subgraphs is an interesting and important problem in the graph mining field, in that mining frequent subgraphs from a single large graph has been strongly developed, and has recently attracted many researchers. Among them, MNI-based approaches are considered as state-of-the-art, such as the GraMi algorithm. Besides frequent subgraph mining (FSM), frequent closed frequent subgraph mining was also developed. This has many practical applications and is a fundamental premise for many studies. This paper proposes the CloGraMi (Closed Frequent Subgraph Mining) algorithm based on GraMi to find all closed frequent subgraphs in a single large graph. Two effective strategies are also developed, the first one is a new level order traversal strategy to quickly determine closed subgraphs in the searching process, and the second is setting a condition for early pruning a large portion of non-closed candidates, both of them aim to reduce the running time as well as the memory requirements, improve the performance of the proposed algorithm. Our experiments are performed on five real datasets (both directed and undirected graphs) and the results show that the running time as well as the memory requirements of our algorithm are significantly better than those of the GraMi-based algorithm.

INDEX TERMS Data mining, frequent closed subgraph, social network, pruning strategy.

I. INTRODUCTION

In the data mining field [1]–[3], closed itemset mining [4] has been developed for a long time and been the focus of many studies [5]–[10]. But in the graph mining field [11]–[14], closed frequent subgraph mining is a new problem with little research [15], [16]–[19]. Because graphs are non-linear data structures [20], [21] they are a challenging and interesting research area that can be used to organize, simulate, model and solve a lot of real world problems [22]–[24] and thus it has become more popular both in scientific as well as commercial fields. Graph analysis [25] has been studied for a long time, and is the premise for many applications such as those associated with social networks, telephone networks, program flows, bio-informatics, chemical compounds, terrorist networks, etc., with closed frequent subgraph mining forming the fundamental basis for graph clustering, graph based anomaly detection, and graph classification [26]–[28].

For a real-life example, a sales company collects customer data and wants to find frequent customer groups to fine-tune its business strategies. The graph G in Figure 1 demonstrates the list of all the customers, each customer is represented by a node belonging to a group labeled A, B, C or D, and each edge of the two nodes indicates the relationship (labeled x, y, z, t or w) of those two customers. Node labels indicate the kinds of the customers and edge labels illustrate the kinds of customers' relationships. Two subgraphs S_1, S_2 in Figure 1 are two samples for the company's frequent purchasing groups.

This leads to a very practical and useful problem, which is to find all frequent purchasing groups with their largest

The associate editor coordinating the review of this manuscript and approving it for publication was Senthil Kumar.

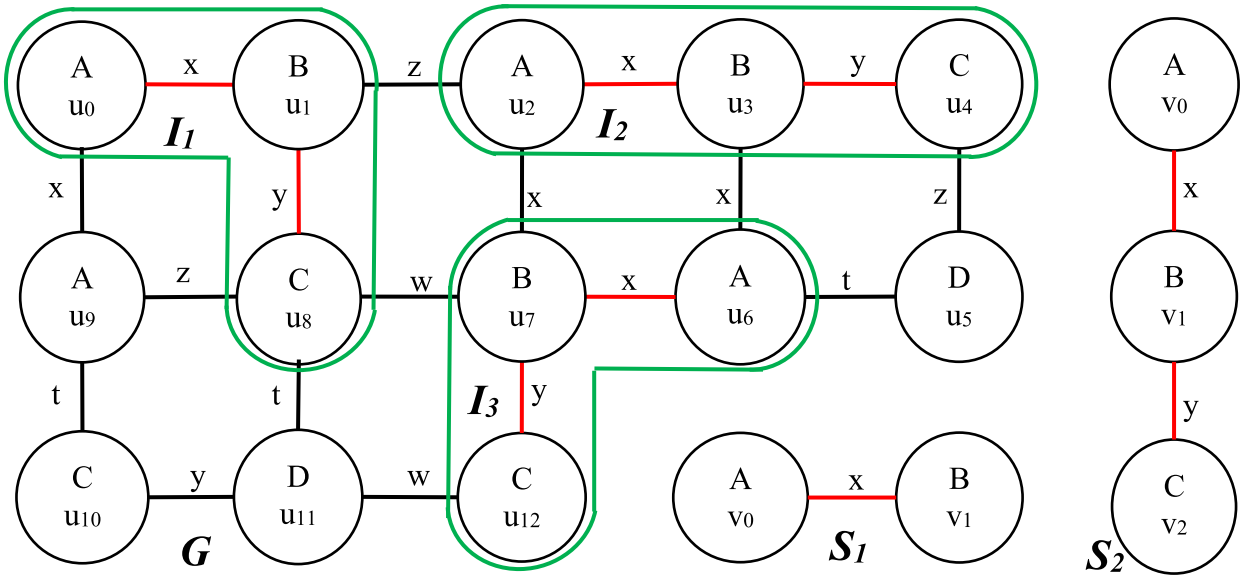


FIGURE 1. A large graph G and two subgraphs S_1 and S_2.

possible actual number of purchasing times. The results of the search help to adjust the company's business strategies. This is useful not only in business management [10] but also in many other fields [29], such as criminal investigations, decision support systems, information retrieval systems, map model analysis, consulting systems, structural graph clustering, and so on.

In 2014, GraMi [30], [31] was proposed as a novel approach to efficiently mine frequent subgraphs [30]. Most methods for subgraph mining in a large graph G work by searching and counting the number of isomorphisms of a subgraph S in G [30], [32]–[34], if this number is not less than a given frequency threshold τ, this means S is a frequent subgraph. These subgraph mining algorithms suffer two main costs: (1) generating candidates [33] and (2) checking candidates' isomorphisms [21], [35].

The approach in GraMi is novel in comparison to the previous grow-and-store methods, because it only stores templates of all candidate subgraphs, not each candidate's appearance in the entire large graph in order to reduce the huge storage space of candidate generation. In addition, it uses a method to model a candidate's support, the constraint satisfaction problem (CSP) [30]. To reduce running time when checking isomorphisms of a subgraph, GraMi only searches the number of appearances (isomorphisms), or the minimum image based support (MNI) [30] of S in G, and this number is enough to determine whether a subgraph S is frequent or not, and thus it ignores all the remaining appearances (in Section III, we describe them in detail). However, in the worst case the time needed by the CSP model still increases exponentially.

In 2020, we improved GraMi to be able to find the supports for frequent subgraphs, with an approach called SuGraMi [33]. Because of finding just enough isomorphisms to determine that a candidate subgraph S is frequent, GraMi can only find all frequent subgraphs in a graph dataset, but not their support. In 2020, we improved GraMi to be able to find the supports for frequent subgraphs, with an approach called SuGraMi [33]. Although SuGraMi can find all frequent subgraphs with their useful supports, they are not closed frequent subgraphs yet. For this problem, we proposal an algorithm for closed frequent subgraph mining and two effective strategies to optimize this algorithm:

(1) We state the problem of mining closed frequent subgraphs from a single large graph.

(2) With the aim to reduce the processing time as well as storage space for our proposed method, we propose a level-wise traversal strategy for the search tree to determine closed frequent subgraphs early, and this helps improve the performance of the algorithm. Based on this strategy, we set one more condition for non-closed subgraphs, if non-closed subgraphs violate that condition, they will be pruned early to reduce storage space.

(3) We propose an efficient algorithm, named CloGraMi, to mine all closed frequent subgraphs based on (2).

Our paper is organized as follows: We survey related studies related to subgraph mining in Section II. The concepts and definitions for the new algorithm are presented in Section III, while in Section IV our new algorithm and optimizations will be demonstrated in detail. All our experimental results with five datasets (directed/undirected graph) are shown in Section V. Finally, Section VI presents our conclusion and proposes some directions for future research.

II. RELATED STUDIES

In 2002, the gSpan [36] algorithm was proposed as a new approach, it can efficiently mine frequent graph-based patterns in a graph dataset by building a new lexicographic order for graphs, then it maps each subgraph to a unique minimum

depth-first search (DFS) code as its canonical label. Then, based on this lexicographic order, a subgraph with $(k + 1)$ edges will be generated by adding a new edge to a subgraph with k edges on the search tree. This $(k + 1)$ edges subgraph corresponds to the level $(k + 1)$ in the search tree, and has nodes that contain the DFS code of subgraph k edge. GraMi uses gSpan to find subgraph candidate isomorphisms because gSpan can greatly reduce memory requirements. Instead of keeping all candidates' appearances in the large graph, GraMi only stores these candidates' templates in order to reduce storage space in the search tree. GraMi also finds just enough isomorphisms to determine whether a candidate subgraph is frequent or not [35], and ignores all the remaining appearances to reduce the search time.

Continuing to optimize GraMi using parallel and distributed computing, ScaleMine [21], [31], [37] and SSIGRAM [38] were proposed in 2016 and 2018 as parallel frequent subgraph mining algorithms for a single large graph. The ScaleMine algorithm splits all mining tasks to separate CPU cores and is performed on the Shaheen II (a modest cluster on the Cray XC40 supercomputer), while the SSIGRAM algorithm uses threads on a cluster [38] and is implemented on the Apache Spark framework. ParGraph [39] includes a new hybrid load balancing scheme to distribute tasks/threads more efficiently and processes the data using a message passing interface (MPI) and OpenMP. This scheme for hybrid load balancing is different from current parallel approaches based on load balancing, as seen with MapReduce [40], OpenMP [41], Arabesque [42], DistGraph [43] and Pregel [44].

Meanwhile, CloseGraph [15] generates closed frequent patterns with a smaller support value than any of its subpatterns, Graphsig [45] also has a similar approach when using a specific subgraph for mining. SMPCS (Steiner Maximum Path-Connected Subgraph) [46] represent the SIoT as the heterogeneous information networks (HINs) with multi types of entities and relations, and it resolves this problem from the perspective of cohesive subgraph search develop efficient algorithms based on an index tree for searching. In 2016, Karabadji *et al.* presented ECE-CloseSG [16]. This is a mining algorithm for closed frequent unique edge label subgraphs. It has a method for search space pruning, and applies a strong accessibility property which allows the algorithm to ignore uninteresting subgraphs. Instead of mining all subgraphs, CloseGraph [15] mines only closed frequent subgraph patterns from a multi-graph dataset by exploring several interesting pruning methods. Its performance shows that CloseGraph not only decreases the number of unnecessarily generated subgraphs significantly, but also substantially enhances the efficiency of the mining process. The success of the algorithm is based on the development of the novel concepts that help CloseGraph prune the search space with regard to the equivalent occurrence of candidate subgraphs, and carry out early termination with a small additional cost.

With regard to closed frequent subgraph mining, the PSI-CFSM algorithm [19] was introduced in 2017 as an optimization for closed frequent subgraph mining algorithm for a set of graphs, and this efficient method aims to reduce the costly process of subgraph isomorphism testing. PSI-CFSM can obtain polynomial time complexity because it uses a binary search to search string code for a unique representation of a k-subgraph in an array of ordered unique string code for a k-subgraph set in a random-access machine model. Frequent approximate subgraph (FAS) mining is also an important technique in the field of graph mining. The paper [18] introduced an algorithm for mining generalized closed FASs from multi-graph collections [47]. This is the first algorithm reported in the literature for mining this kind of pattern over multi-graph collections. One direction that has emerged recently is to use Topological Data Analysis as a method of re-representing a complex graph into a simple graph while preserving its topological characteristics [47]. This is an idea that has a lot of promise which could be exploited in the near future, but it still needs to be thoroughly evaluated for the theoretical framework in approximation [48], [49]. In 2018, the ENERGETICS, CENERGETICS and EXCESS [17] algorithms introduced the novel problem of exceptional closed pattern mining in attributed graphs. This approach identifies all neighborhoods with homogeneous and exceptional characteristics. It takes closure operators, upper bounds and pruning properties. Their experiments on ten real datasets demonstrated the relevance of both approaches, and also showed their limits. The effectiveness of CENERGETICS overcomes that of its previous version, ENERGETICS, but it still has difficulties to scale with the number of attributes. EXCESS can fix this problem because it can mine graphs with hundreds of attributes.

In the field of maximal frequent subgraph mining, a maximal frequent subgraph is defined as a frequent subgraph if it does not have a super-graph that is frequent. In 2018, a novel algorithm UMFGAMW [50] was proposed for uncertain maximal frequent subgraph mining based on the adjacency matrix and weight. It presents an adjacency matrix as well as a standard matrix coding for uncertain graphs. In this method, the complexity is reduced for uncertain graph standard coding, which improves the matching speed for the uncertain subgraph standard coding. In order to reduce the number of total mining results researchers proposed using a limiting condition of the mean weight for weighted uncertain edges and the minimum support threshold under uncertain meaning. In 2021, the RASMA algorithm [51] was proposed for mining frequent and maximal frequent subgraphs in a given collection of graphs [18]. A key innovation in this algorithm is using a connected subgraph enumerator with a reverse-search strategy to enumerate the connected subgraphs of an undirected graph. RASMA uses this enumeration strategy to very efficiently obtain all maximal frequent subgraphs, and employs several pruning strategies to overcome the prohibitively costly task of storing all frequent subgraphs.

According to our survey, there are currently very few studies on mining closed frequent subgraphs [15]–[19], [52]–[54], although CloseGraph [15] is an efficient algorithm for mining

closed graph patterns in a labeled graphs dataset. ECE-CloseSG [16] is an algorithm which finds closed frequent subgraphs in Unique Edge Label Graphs, and it uses search space pruning and applies the strong accessibility property [53], [54] to ignore a large portion of uninteresting subgraphs. TGP [52] was developed to mine closed patterns without minimum support, because it introduces a novel structure to store graph patterns called a Lexicographic Pattern Net, which makes the verification of the closed frequent patterns more efficient and can speed up the process of raising the frequency threshold dynamically.

Almost algorithms mentioned above mine for a set of graphs. There are very few algorithms for mining closed subgraphs in a single large graph, and this is our motivation to propose a new algorithm for this issue. Besides, GraMi [30] is one of the most efficient algorithms for FSM from large graphs [55], but it only finds all frequent subgraphs from a single graph. Therefore, our first contribution is based on the GraMi algorithm, and we propose using CloGraMi to mine closed frequent subgraphs. Our second contribution is to further enhance the performance of the CloGraMi algorithm by early determination of the closed frequent subgraphs. The third contribution is to use a condition for early pruning of non-closed subgraphs which are violated. With these contributions, the CloGraMi algorithm is able to reduce the storage space as well as running time, showing better performance than GraMi.

III. DEFINITIONS, PROPOSITIONS AND PROBLEM STATEMENT

Definition 1 [30]: A *graph* is a triplet $G = (V, E, L)$, in which V is a set containing all the nodes of graph G, E is a set containing all the edges of graph G and L is a function assigning labels to all nodes/edges in the graph.

Definition 2 [30]: Let $S = (V_S, E_S, L_S)$ be a *subgraph* of a graph $G = (V, E, L)$. A subgraph isomorphism of S to G is an injective function $f: V_S \rightarrow V$ satisfying

(a) $L_S(v) = L(f(v)), \forall v \in V_S$

(b) $(f(u), f(v)) \in E$ and $L_S(u, v) = L(f(u), f(v)), \forall (u, v) \in E_S$.

Example 1: In Figure 1, subgraph S_2 has three distinct isomorphisms I_1, I_2 and I_3 with corresponding nodes as following:

Subgraph S_2	v_0	v_1	v_2
Isomorphism I_1	u_0	u_1	u_8
Isomorphism I_2	u_2	u_3	u_4
Isomorphism I_3	u_6	u_7	u_{12}

In our algorithm, each node $v \in S$ has an own domain containing all nodes u which have the same node label and can be assigned to v. We use u to list the nodes in the large graph G and v to list the nodes in the subgraph S. These domains for all nodes in a subgraph are used to mark and count the corresponding occurrences of these nodes v on the large graph G.

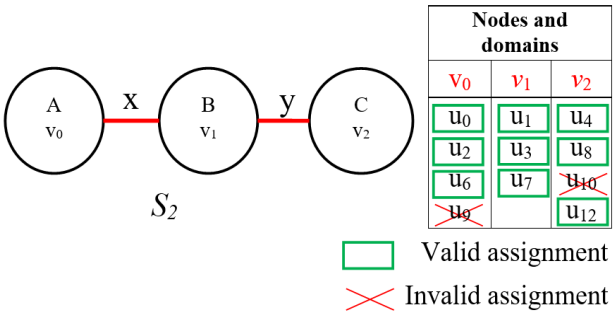


FIGURE 2. Valid and invalid assignments for subgraph S_2.

Example 2: In Figure 2, node v_0 (of subgraph S) has a domain including nodes $\{u_0, u_2, u_6, u_9\}$ (on large graph G).

Proposition 1 [33], [34]: Let $S = (V_S, E_S, L_S)$ be a subgraph of a graph $G = (V, E, L)$, $u \in V$ and $v \in V_S$. An assignment of a node u is a valid assignment in the domain of v if there exists an isomorphism I that u is assignable to v, otherwise u is an invalid assignment.

Example 3: In Figure 2, node v_0 has a domain $D(v_0) = \{u_0, u_2, u_6, u_9\}$, in which u_0, u_2, u_6 are valid assignments (they are corresponding to I_1, I_2 and I_3 in Example 1), the last node u_9 is an invalid assignment because there do not exist corresponding isomorphisms that assign u_9 to v_0.

Because subgraph S_2 has three distinct isomorphisms I_1, I_2, and I_3, the corresponding nodes u (in G) of these isomorphisms are valid assignments. Although S_2 has other isomorphisms such as (u_2, u_7, u_{12}) and (u_6, u_3, u_4), these nodes u are already valid assignments in the domain, therefore finding these isomorphisms does not affect S_2's domain so we do not list them in Example 1.

The *CSP model* [30], [33] in GraMi is represented as a tuple (X, D, C):

(a) X is the set of variables (corresponding to nodes in a subgraph),

(b) D is the set of domains of variables in X,

(c) C is the set of constraints between all variables in X.

GraMi searches for isomorphisms of a subgraph for the *CSP* to get assignments to the variables in X, such that all the constraints in C are satisfied.

Example 4: In Figure 2, the CSP of S_2 is represented as:

$(X = \{v_0, v_1, v_2\},$

$D = \{\{u_0, u_2, u_6, u_9\}, \{u_1, u_3, u_7\}, \{u_4, u_8, u_{10}, u_{12}\}\},$

$C = \{v_0 \neq v_1 \neq v_2; L(v_0) = A, L(v_1) = B, L(v_2) = C; L(v_0, v_1) = x, L(v_1, v_2) = y\})$

GraMi solves the *CSP* model by searching isomorphisms until it finds the MNI support of S in G that is enough to evaluate S as a frequent subgraph, and it ignores all remaining appearances. The MNI-based support is only used to evaluate a candidate subgraph, it is not the exact number of appearances of a subgraph in the large graph. In this paper, we need to compare the support of frequent subgraphs to find out all closed subgraphs, therefore we do not use the MNI-based support. Instead, we search all isomorphisms and use the full support [33] of candidate subgraphs.

Proposition 2 [33]: The support of a subgraph S in a large graph G (denoted by $s_G(S)$) is the minimum number of all distinct valid assignments (corresponding to all isomorphisms of S) for every node v in the domain of S. In other words:

$s_G(S) = \min\{t|t = |D(v)|, \forall v \in V_S\}$.

Example 5: The support of S_2 in G

$s_G(S_2) = \min(|D(v_0)|, |D(v_1)|, |D(v_2)|) = \min(3, 3, 3) = 3$.

Definition 3 [33]: A subgraph S in G is *frequent* if $s_G(S) \geq \tau$ with τ is a given frequency threshold.

If nothing changes, the frequency threshold is chosen $\tau = 2$ for all the examples in this paper.

Example 6: Because $s_G(S_2) = 3$ and $s_G(S_2) \geq \tau$, therefore S is a frequent subgraph in G.

If S is a subgraph of S', then S' is a super graph of S, denoted by $S \in S'$ (if proper super graph, $S \subset S'$). The set of *frequent graph patterns*, *FS*, consists of all the graphs whose support is equal to or greater than a given frequency threshold. The set of *closed frequent graph patterns*, *CS*, is defined as follow [15]:

$CS = \{S|S \in FS$ and $S' \in FS$ such that $S \subset S'$ and support(S) = support$(S')\}$

Definition 4 [15]: A frequent subgraph S is a *closed frequent subgraph* if and only if there does not exist any proper supergraph S' whose support is equal to that of S (and we call as ''closed subgraph'' in short).

Example 7: S_2 is a closed subgraph and S_1 is a non-closed subgraph (see the details in Figure 3 and Subsection IV.A).

Our first contribution is to propose a GraMi-based algorithm to find out all closed subgraphs from the list of mined frequent subgraphs. It consists of two main steps: 1) the algorithm finds all frequent subgraphs; and 2) the program filters out all closed subgraphs in the list of frequent subgraphs (see Subsection IV.B for details).

As in [33], we have proved that $s_G(S') \leq s_G(S)$ if subgraph S' is generated from subgraph S. Thus, when evaluating whether a subgraph S (with size n) is closed or not, the algorithm only needs to check the subgraphs S' with size $(n + 1)$ that are directly generated from S (proven in Subsection IV.B). This proposal is also our second contribution in this paper.

The mining closed frequent subgraphs problem in a single large graph G is stated to find all frequent subgraphs S if there exists no proper supergraph S' (of S in G) that has the same support as S. To solve it, we propose the CloGraMi algorithm, based on an advanced version of the GraMi algorithm, which can accurately find all closed frequent subgraphs through two phases (see Subsection IV.A). However, this algorithm still needs a long time and large storage space to be executed. To reduce the search time, we propose a new traversal strategy for CloGraMi (detailed in Subsection IV.B). Moreover, we propose a condition for an effective strategy to quickly determine whether a frequent subgraph is not a closed subgraph and remove it from the storage space of the mining process. This strategy helps to reduce the search time and reduce the storage space (detailed in Subsection IV.C). This is our third contribution.

IV. PROPOSED ALGORITHMS

In this section, we describe our three contributions in detail with specific examples for each contribution.

A. A CLOSED SUBGRAPH MINING ALGORITHM BASED ON GRAMI

The GraMi algorithm uses a novel approach and can quickly and efficiently mine frequent subgraphs, and many other algorithms have been proposed to improve its performance. However, with regard to mining frequent subgraphs, no other algorithms have been proposed to mine closed subgraphs based on GraMi. Indeed, there are very few closed subgraph mining algorithms that aim to work with single large graphs, and there are none based on GraMi.

In this subsection, we propose the GraMi-based algorithm as a baseline algorithm to mine closed subgraphs with the following two main steps:

(1) Find frequent subgraphs (from Line 2 to Line 6): Like GraMi, our algorithm puts all frequent edges into a list, called *fEdges*. These edges are frequent subgraphs with 1-edge, then the program will simultaneously extend these subgraphs by adding a new edge (from the *fEdges* list) to generate a new subgraph candidate and evaluate whether this candidate is frequent or not. If the candidate is frequent, the program continues to extend and evaluates it recursively.

Example 8: Given the graph G in Figure 1, assuming the frequent threshold is $\tau = 2$, we have a list of frequent edges:

$$fEdges = \{A \overset{x}{—} B; B \overset{y}{—} C; C \overset{z}{—} D\}$$

The program will process the edges in *fEdges* sequentially. In Figure 3, with $\tau = 2$, S_{11} (corresponding to S_1 in Figure 1) is a frequent subgraph because $s_G(S_{11}) = 3$, so the program will generate and evaluate recursively the children of S_{12}. Because S_{12} is also a frequent subgraph because $s_G(S_{12}) = 2$, the process will generate and evaluate recursively for all the children of S_{12} on the search tree, and when all the children on that branch are generated and evaluated the program will move to the next branch (S_{14} in this case).

(2) Filter out all closed subgraphs from the list of frequent subgraphs (from Line 7 to Line 14): For each subgraph S in the list of frequent subgraphs obtained in Step 1, the program checks for the existence of a subgraph S' which is a proper supergraph of S and that has the same support. If no subgraph S' exists, then S is a closed subgraph and it will be put into the result list.

Example 9: In Figure 3, S_{11} (S_1 in Figure 1) is a frequent subgraph but not a closed subgraph, because there exists S_{15} as a proper supergraph and $s_G(S_{11}) = s_G(S_{15}) = 3$. But S_{15} (S_2 in Figure 1) is a closed subgraph.

In the first step of the mining process, at Line 5 we use the recursive function SUBGRAPHEXTENSION() as in [30], and if a subgraph candidate is determined to be frequent then it will be extended and evaluated recursively to find all

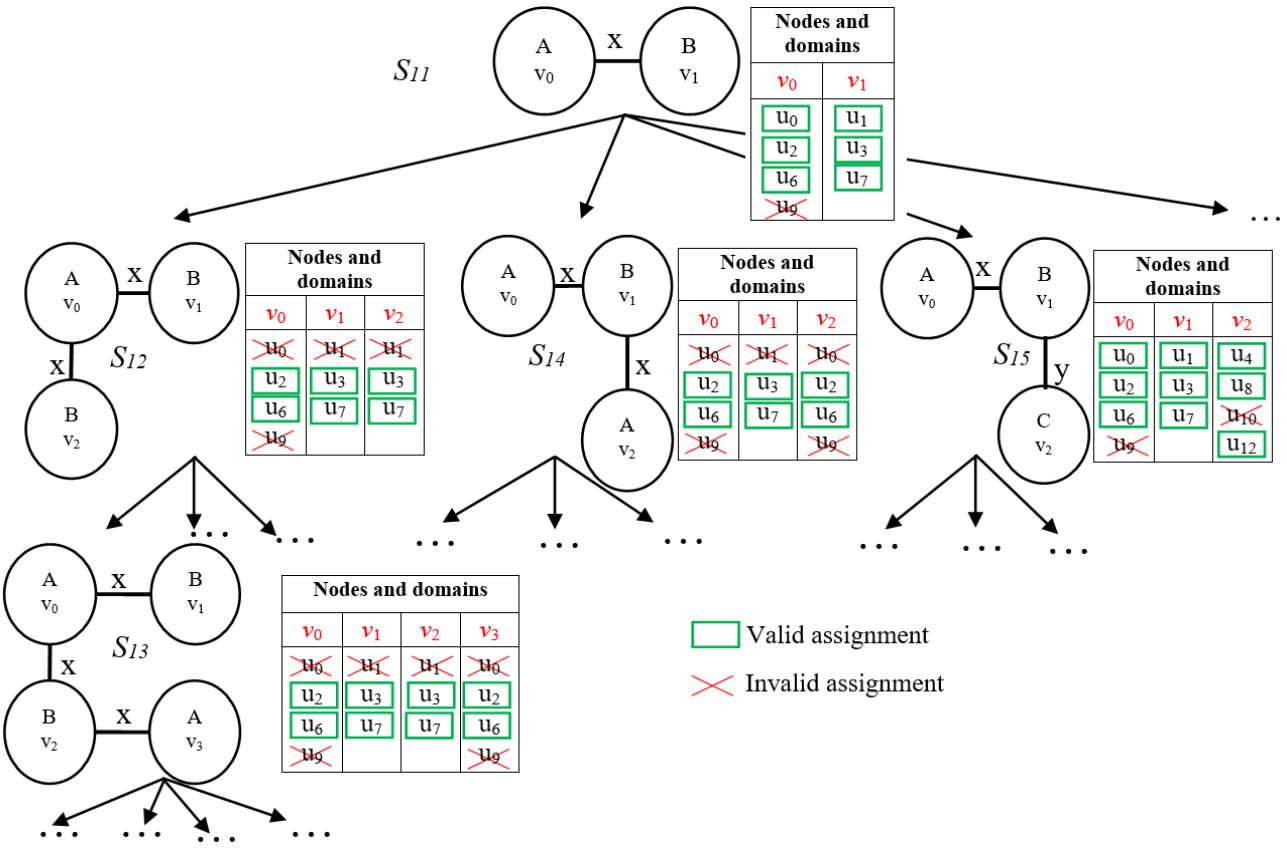


FIGURE 3. Valid and invalid assignments for subgraph A $\stackrel{x}{-}$ B.

Algorithm 1 GraMi-Based

Input: G : single large graph, τ : frequency threshold
Output: rList: closed subgraphs of G
1 rList $\leftarrow \emptyset$ *//the result list of all closed subgraphs*
2 fSubgraphs $\leftarrow \emptyset$ *//the list of all frequent subgraphs*
3 Let *fEdges* be a set containing frequent edges in G
4 **foreach** $e \in$ *fEdges* **do**
5 fSubgraphs $\leftarrow$ fSubgraphs $\cup$ SUBGRAPHEXTENSION(e, G, τ, *fEdges*)
6 Remove e from *fEdges*
7 **foreach** $S \in$ fSubgraphs **do**
8 closed $\leftarrow$ true
9 **foreach** $S' \in$ fSubgraphs **do**
10 **if** $(S' \supset S)$ and $s_G(S') = s_G(S)$ **then**
11 closed $\leftarrow$ false
12 break
13 **if** closed = true **then**
14 rList $\leftarrow$ rList $\cup S$ 15 return rList

frequent subgraphs which are generated from that subgraph. All frequent subgraphs will be restored with their support to compare in the second step.

In the second step, for each mined frequent subgraph S (Line 7), we need to check if there exists a frequent subgraph S' in the list of frequent subgraphs which is the proper supergraph of S; and S' has the support equal to that of S (Line 10). If no subgraph S' exists, then S is a closed subgraph and is put in the result list (Lines 13 and 14).

However, those two steps in our CloGraMi algorithm are still costly in terms of running time and memory requirement. We continuously optimize our algorithm by two efficient strategies as shown in the following subsections.

B. LEVEL ORDER TRAVERSAL STRATEGY TO QUICKLY DETERMINE CLOSED SUBGRAPHS

With the baseline approach as in Subsection IV.A, the mining process needs to find and store all frequent subgraphs in G, then perform a filtering process for all closed subgraphs by comparing each frequent subgraph S with all remaining frequent subgraphs in the result list, and this is costly and unnecessary.

Proposition 3: Frequent subgraph S with size n is not a closed subgraph if there exists a subgraph S' size $(n+1)$ generated from S in the search tree, whose the support is equal to that of S.

Proof:

- Let S be a frequent subgraph with size n.
- Let S' be a frequent subgraph with size $(n+1)$, let S'' be a frequent subgraph with size $(n+2)$, and so on.

Algorithm 2 CloGraMi

Input: G: single large graph, τ: frequency threshold
Output: rList: closed subgraphs of G 1 rList $\leftarrow \emptyset$ *//the result list of all closed subgraphs*
2 Let *fEdges* is a set contains frequent edges of G
3 **foreach** $e \in$ *fEdges* **do**
4 rList $\leftarrow$ rList $\cup$ ExtendByLevel(e, τ, *fEdges*, G)
5 Remove e from *fEdges*
6 **return** rList

Algorithm 3 ExtendByLevel

Input: S: frequent subgraph, τ: frequency threshold, *fEdges*: set of frequent edges of large graph G
Output: cSubgraphs: closed subgraphs in G that are extended from S
1 cSubgraphs $\leftarrow \emptyset$ *//the list of closed subgraphs*
2 gSubgraphs $\leftarrow \emptyset$ *//the list of generated subgraphs*
3 fSubgraphs $\leftarrow \emptyset$ *//the list of frequent subgraphs*
4 **foreach** edge $e \in$ *fEdges* **and** node $v \in S$ **do**
5 **if** e can extend u **then**
6 Let E be the extension of S by e
7 **if** E is not already generated **then**
8 gSubgraphs $\leftarrow$ gSubgraphs $\cup E$
9 closed $\leftarrow$ true
10 **foreach** $g \in$ gSubgraphs **do**
11 **if** $s_G(g) \geq \tau$ **then**//*g is a frequent subgraph*
12 fSubgraphs $\leftarrow$ fSubgraphs $\cup g$
13 **if** $s_G(g) = s_G(S)$ **then**//*compare the support*
14 closed $\leftarrow$ false
15 **if** closed = true **then**
16 cSubgraphs $\leftarrow$ cSubgraphs $\cup S$
17 **foreach** $S' \in$ fSubgraphs **do**
18 cSubgraphs $\leftarrow$ cSubgraphs $\cup$ ExtendByLevel(S', τ, *fEdges*, G)
19 **return** cSubgraphs

- In [33], we have proved:

$s_G(S) \geq s_G(S')$ where S' is directly generated from S in the search tree (1)

- Thus, by extension, we have:

$s_G(S) \geq s_G(S') \geq s_G(S'')$, where S'' is the directly generated subgraph from S' (2)

- From (2), in the search tree, if there does not exist S' (subgraph with size $n+1$) whose support is equal to that of S (subgraph with size n), then there cannot exist a subgraph S'' (subgraph with size $n+2$) that has the same support as that of S.

- Therefore, rather than having to compare it to all mined frequent subgraphs in the result list, when determining whether a subgraph S (size n) is closed or not we only need to compare it with the frequent subgraphs that are directly generated children (size $n+1$) on the search tree (1): if there exist S' whose support is equal to the support of S then S is not closed.

With the GraMi algorithm, the process of extending and evaluating frequent subgraphs will be implemented recursively using a DFS on the search tree with the function SUBGRAPHEXTENSION(), we propose implementing a strategy of frequent extension and evaluation for a "level order traversal" in the search tree, namely the function ExtendByLevel A frequent subgraph with size n will generate all its candidate subgraphs size $(n+1)$, and we only compare its support with the candidates' size $(n+1)$ to determine whether it is a closed subgraph or not.

Example 10: In Figure 4, S_{21} (corresponding to S_1 in Figure 1) is a frequent subgraph, and the algorithm will generate all children of S_{21} in the search tree (including S_{22}, S_{23}, S_{24} and others). After calculating the support for these children, the mining process will compare the support of S_{21} to its children's support, if there exists one that has the same support as S_{21} (in this case $s_G(S_{21}) = s_G(S_{24}) = 3$), it determines S_{21} as a frequent subgraph but not a closed subgraph. All frequent subgraphs generated from S_{21} will also be generated and evaluated recursively by the program. With this strategy, instead of comparing the support of S_{21} with all the mined frequent subgraphs, we only compare S_{21} with its children to reduce the processing time.

In the CloGraMi algorithm, we only need one list to store the closed subgraphs (Line 1) during entire the computation, and a list for all frequent subgraphs is unnecessary. At Line 4, the recursive function ExtendByLevel() has been optimized, which will generate all the children of a frequent subgraph S and evaluate them sequentially.

In the function ExtendByLevel(), from Line 4 to Line 8, the program will generate all candidate subgraphs of the subgraphs of S. Line 11 and Line 12 check whether these subgraphs are frequent or not. From Line 10 to Line 16, the program only compares the support of subgraph S with the frequent subgraphs g ($s_G(g) \geq \tau$ at Line 11) generated from it instead of comparing with all mined frequent subgraphs. If S is closed, it will be put in the result list. Line 17 and Line 18 are used to recursively generate and evaluate frequent subgraphs that are children of subgraph S.

Thus, after finishing the recursive process for each branch in the search tree, the main program can free the storage for frequent subgraphs and allocate it to another search branch instead of having to store the entire frequent subgraphs of all the branches in the search tree, as in Subsection IV.A. After applying the level order traversal strategy for the search tree, we propose another strategy to identify non-closed subgraph early in subsection IV.C to enhance the algorithm performance.

C. EARLY PRUNING NON-CLOSED CANDIDATES

With our level order traversal strategy, when determining whether a frequent subgraph S' is closed or not, in the recursive ExtendByLevel() function in Subsection IV.B, the program will generate all candidate subgraphs S' and store

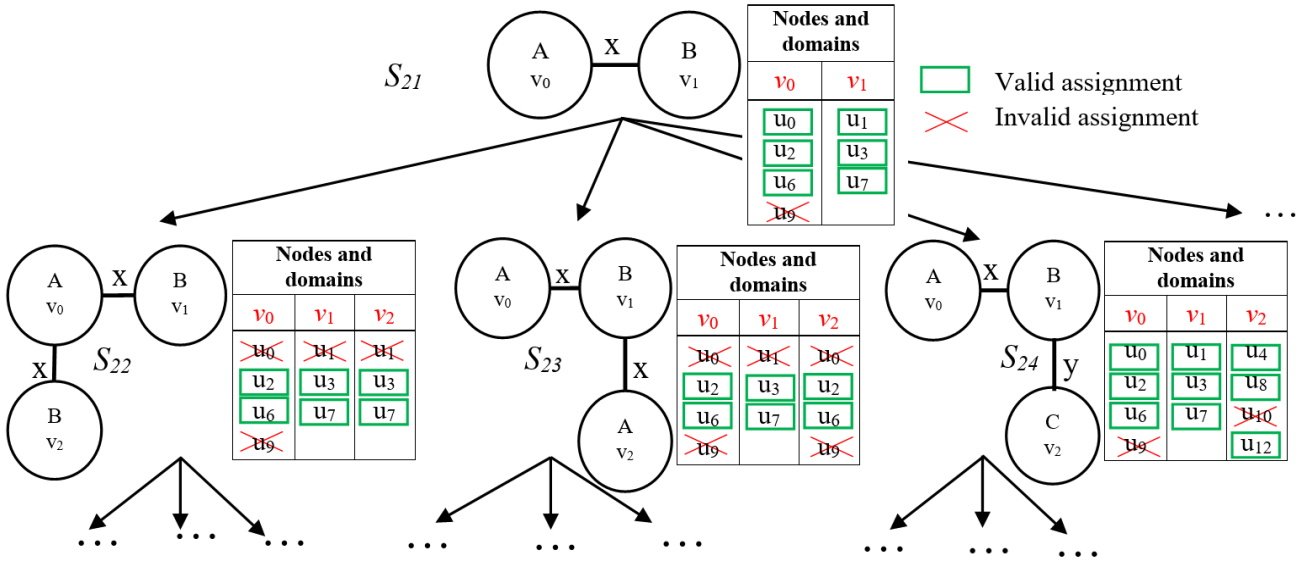


FIGURE 4. Some subgraphs generated from A $\overset{x}{-}$ B.

them all; after the generation process is the process of calculating the support for these subgraphs S' to determine the frequent subgraphs (which will continue to recursively expand in a later step); Finally, compare the support of S and that of S' to determine whether S is a closed subgraph or not. However, this still requires a lot of computer memory.

Our third contribution is to set a condition to function ExtendByLevel() for early pruning a non-closed subgraph S as soon as it is determined that there is a candidate S' whose support is equal to that of S and we called it as ExtendByCondition Remove the domain of S from storage memory and save only the template of S to generate the next candidates S', in addition we also arrange the edges in the *fEdges* list to increase performance.

In the ExtendByCondition() function we set a "closed" variable at Line 3, and this is the condition to determine S is a non-closed subgraph, from Line 4 to Line 6, the program will generate the children subgraph S' of S in the search tree. In this, the program does not have to wait until all children of have been generated before starting the process of evaluating each of them. When generating any child, the program will calculate support for that subgraph. From Line 8 to Line 9, only frequent subgraphs S' will be stored to be evaluated recursively (Line 15 to Line 16), and if at any time the program detects the existence of a frequent subgraph S' whose support is equal to that of S (Line 10), then S is not a closed subgraph (Line 11) and the process will delete the domain of a subgraph S (Line 12) to reduce the storage space, keeping only the template (including the list of vertices and edges of S) to generate the next S'. All frequent subgraphs generated from S will be extended recursively (Line 15 and Line 16).

Algorithm 4 ExtendByCondition

Input: S: frequent subgraph, τ: frequency threshold, *fEdges*: set of frequent edges of large graph G
Output: cSubgraphs: closed subgraphs in G that are extended from S
1 cSubgraphs ← ∅ *//the list of closed subgraphs*
2 fSubgraphs ← ∅ *//the list of frequent subgraphs*
3 closed ← true
4 **foreach** edge $e \in fEdges$ **and** node $u \in S$ **do**
5 **if** e can extend u **then**
6 Let S' be the extension of S with e
7 **if** S' is not already generated **then**
8 **if** $s_G(S') \geq \tau$ **then**
9 fSubgraphs ← fSubgraphs $\cup S'$
10 **if** $s_G(S') = s_G(S)$ **then**
11 closed ← false
12 Delete domain of S
13 **if** closed = true **then**
14 cSubgraphs ← S
15 **foreach fS** ∈ fSubgraphs **do**
16 cSubgraphs ← cSubgraphs ∪ ExtendByCondition(*fS*, τ, *fEdges*, G)
17 **return** cSubgraphs

Let N and n be the number of nodes in G and S, respectively, both GraMi-based and CloGraMi and the complexity of the mining process needs $O(2^{N^2}.N^n)$ time [30], which is exponential in the problem size. But the GraMi-based approach needs a step to filter out closed subgraphs from the list of all frequent subgraphs (from Line 7 to Line 14) after the mining process. Let M be the number of frequent

subgraphs, at Line 7 and Line 9, we have $O(M^2)$ for iteration of the frequent subgraphs list. Let k is the number of nodes in S, K is the number of nodes in S', in the filtering process S is a subgraph and S' is a large graph. At Line 10, we need to check whether S is an subgraph isomorphism in S', and this problem takes $O(K^k)$ time (a well-known NP-hard problem) [30]. Overall, the complexity of the filtering process is $O(M^2.K^k)$.

In contrast, our CloGraMi only needs the mining process, without the filtering process. In the function ExtendByCondition() of the mining process, because S' is generated from S in the search tree (S is always a subgraph isomorphism of S'). At Line 8, getting the support of all candidate subgraphs in G (NP hard complete) is the same as with the function SUBGRAPHEXTENSION() [30] in GraMi-based, we only need to compare these supports at Line 10, $s_G(S') = s_G(S)$, which only takes the complexity O(1).

With large datasets consisting of millions of nodes/edges, the domain of a frequent subgraph is very large, but the template of a subgraph is always very small, so deleting the domain will reduce storage space. Because the time for support calculation of the candidates is constant, with our early pruning non-closed strategy the program can reduce the time needed to filter closed subgraphs, reducing the storage space of non-closed subgraphs.

V. EXPERIMENTAL STUDIES

We use five datasets (including three undirected graph datasets and two directed graph datasets), a personal computer with a CPU i5, 4 cores, 3.2GHz, 4GB RAM, Windows 10 operating system and Java programming language. In this section we record and compare the performance of the GraMi-based algorithm with our new algorithm CloGraMi based on five real datasets.

The details of our five datasets are as follows:

+ Bitcoin Alpha: This is downloaded from the website https://snap.stanford.edu/data/, and is the dataset of an undirected graph, which is a network of people who use Bitcoin to trade on the Bitcoin Alpha platform. Because Bitcoin users are anonymous, this needs to maintain a record of user's reputation to prevent transactions with fraudulent and risky users. It includes 3,783 nodes and 24,186 edges; initially these vertices and edges have no labels, so we add 50 different labels randomly with the rate shown in Table 1.

+ LastFM Asia Social Network: Like the Bitcoin Alpha dataset, this dataset is also downloaded from the website https://snap.stanford.edu/data/. It is a social network of LastFM users, consisting of 7,624 nodes and 27,806 edges. It was collected in March 2020 from the public API. Its nodes are the users of LastFM from Asian countries and its edges are relationships between the users. The nodes and edges are also randomly labeled by us, with 60 different labels, with the details shown in Table 2.

+ CiteSeer: This dataset is a directed graph, there are 3,312 publications (each publication corresponds to a node in this dataset) and 4,732 citations for these publications (each citation is a directed edge between two nodes). Each node in this dataset has a single label (it is a field of Computer Science) and each edge has a value from 0 to 100, with these edge labels indicating the similarity between two publications. As in [33], [34], we only take the integer parts of the decimal numbers of each edge label.

TABLE 1. 50 distinct labels for nodes/edges and their ratings.

Labels of nodes/edges	1	2	3	4	5	6	7	8	9	10
Rating (%)	12	9	8	6	5	4	3	3	3	2
Labels of nodes/edges	11	12	13	14	15	16	17	18	19	20
Rating (%)	2	2	2	2	2	1	1	1	1	1
Labels of nodes/edges	21	22	23	24	25	26	27	28	29	30
Rating (%)	1	1	1	1	1	1	1	1	1	1
Labels of nodes/edges	31	32	33	34	35	36	37	38	39	40
Rating (%)	1	1	1	1	1	1	1	1	1	1
Labels of nodes/edges	41	42	43	44	45	46	47	48	49	50
Rating (%)	1	1	1	1	1	1	1	1	1	1

+ Email-Eu-core network: This directed graph dataset was downloaded from the website https://snap.stanford.edu/data/. The network includes 1,005 nodes and 25,571 edges, it was generated from a large European research institution by using email data, but it has anonymized member information about all incoming/outgoing email among users. There is a directed edge if a person sent another person at least one email in the network. Just like the Bitcoin Alpha dataset above, we randomly added 50 different labels to the nodes/edges at the rate shown in Table 1.

+ GitHub Social Network: This graph was downloaded from the website https://snap.stanford.edu/data/. It is an undirected graph dataset, it is a large social network collected in June 2019 from the public API. This is a network of GitHub developers, in which the nodes present developers (have starred at least 10 repositories), the edges illustrate mutual follower relationships between developers. The node features are extracted based on the email address, location and repositories starred. This dataset includes 37,700 nodes (10 times bigger than the Bitcoin Alpha dataset) and 289,003 edges (over 10 times bigger than the LastFM Asia Social Network dataset). Each node is a user with their own name, so to anonymize the names we randomly added 60 different labels to the nodes/edges at the rate shown in Table 2.

The features of our five datasets are shown in Table 3:

Using these five datasets consisting of undirected and directed graphs we conduct the experiments and record the results, using different frequency thresholds τ (to demonstrate the performance of the new algorithm with a large number of frequent subgraphs and closed subgraphs, we try to reduce the thresholds τ until our computer cannot execute it anymore). Specifically for the last dataset, GitHub Social Network, we divide it into four parts of 22,000 nodes, 27,000 nodes, 32,000 nodes and 37,700 nodes (entire this dataset) to test the scalability of our CloGraMi algorithm and compare

TABLE 2. 60 distinct labels for nodes/edges and their ratings.

Labels of nodes/edges	1	2	3	4	5	6	7	8	9	10
Rating (%)	10	6	5	4	4	3	3	3	3	3
Labels of nodes/edges	11	12	13	14	15	16	17	18	19	20
Rating (%)	2	2	2	2	2	2	1	1	1	1
Labels of nodes/edges	21	22	23	24	25	26	27	28	29	30
Rating (%)	1	1	1	1	1	1	1	1	1	1
Labels of nodes/edges	31	32	33	34	35	36	37	38	39	40
Rating (%)	1	1	1	1	1	1	1	1	1	1
Labels of nodes/edges	41	42	43	44	45	46	47	48	49	50
Rating (%)	1	1	1	1	1	1	1	1	1	1
Labels of nodes/edges	51	52	53	54	55	56	57	58	59	60
Rating (%)	1	1	1	1	1	1	1	1	1	1

TABLE 3. The features of the five datasets.

Datasets	Type	Nodes	Node labels	Edges	Edge labels
Bitcoin Alpha	Undirected	3,783	50	24,186	50
LastFM Asia Social Network	Undirected	7,624	60	27,806	60
CiteSeer	Directed	3,312	6	4,732	101
Email-Eu-core network	Directed	1,005	50	25,571	50
GitHub Social network	Undirected	37,700	60	289,003	60

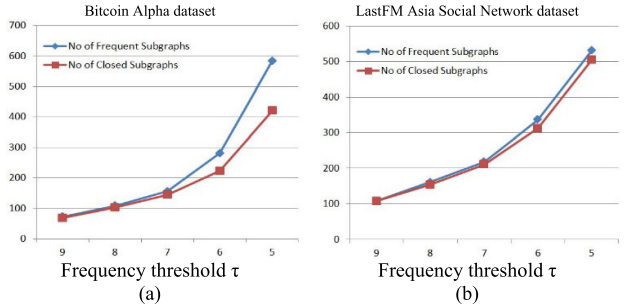


FIGURE 5. The numbers of frequent and closed subgraphs of two undirected graphs.

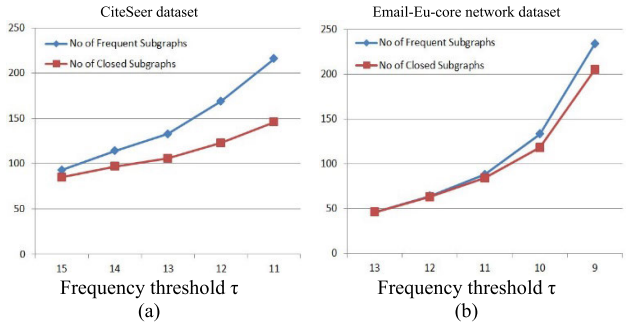


FIGURE 6. The numbers of frequent and closed subgraphs of two directed graphs.

the results. We also use a sorting strategy [33] for the new algorithm to reduce the search space. In this section we will compare the GraMi-based algorithm with the CloGraMi algorithm on three criteria: the number of candidates for filtering closed subgraphs, the running time and the computer memory requirements. As in all our experiments, the lower the threshold, the better the CloGraMi algorithm is in comparison to the GraMi-based one.

With the first criterion the GraMi-based algorithm finds all frequent subgraphs, and these are candidates to filter out the necessary closed subgraphs, but for our CloGraMi the program filters out closed subgraphs while testing them, so all frequent subgraphs in the result list are closed subgraphs.

+ For the Bitcoin Alpha dataset (Figure 5.a): In this dataset, CloGraMi proved to be significantly more efficient than the GraMi-based algorithm. The number of candidates of CloGraMi (also the number of closed subgraphs) can be reduced to 71.9% that needed when compared to the GraMi-based algorithm. In particular, at threshold $\tau = 5$, GraMi finds 584 frequent subgraphs and filters out 420 closed subgraphs, but CloGraMi finds 420 closed subgraphs directly.

+ For the LastFM Asia Social Network dataset (Figure 5.b): The effect of CloGraMi with this dataset is not as significant as that of Bitcoin Alpha dataset, at our lowest possible threshold $\tau = 5$, CloGraMi is only reduced to 94.9% compared to GraMi. CloGraMi finds 504 closed subgraphs directly, while GraMi finds 531 frequent subgraphs, and then has to perform the filtering process.

+ For the dataset CiteSeer (Figure 6.a): On this directed graph dataset, our CloGraMi proved to be the most efficient of the five datasets, as the number of candidates of CloGraMi can be reduced to 67.5% at threshold $\tau = 11$. Specifically, at this threshold, GraMi cannot perform because of exceeding the available memory, but CloGraMi was still able to find 146 closed subgraphs directly (and we used a statistical variable to record the number of frequent subgraphs that can be mined at this threshold as 216).

+ For the Email-Eu-core network dataset (Figure 6.b): Although not as effective as with the CiteSeer dataset above, CloGraMi can also reduce the number of candidates to 87.6% compared to that needed by GraMi. Specifically, at the threshold $\tau = 9$ GraMi also exceeds the available memory and crashes, CloGraMi can directly find 205 closed subgraphs, and we also use a statistical variable to record the number of frequent subgraphs at this threshold as 234.

The second criterion for our comparison is the running time for the two algorithms. As with the first criterion, the number of candidates is reduced, and there is no time to filter closed subgraphs, so CloGraMi can reduce the running time significantly compared to the GraMi-based algorithm, the greatest reduction is to 73.7% for the first dataset. On the three undirected datasets, the running time is reduced much more than on the two directed datasets.

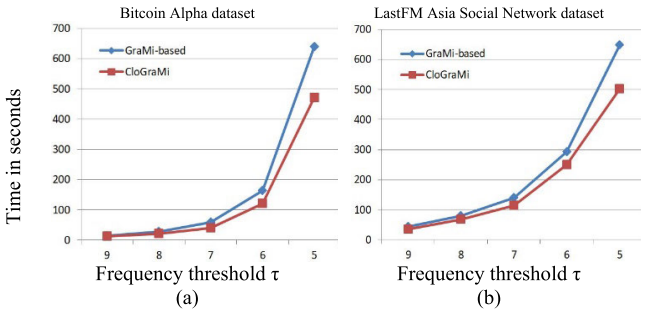


FIGURE 7. Running time for the two undirected datasets.

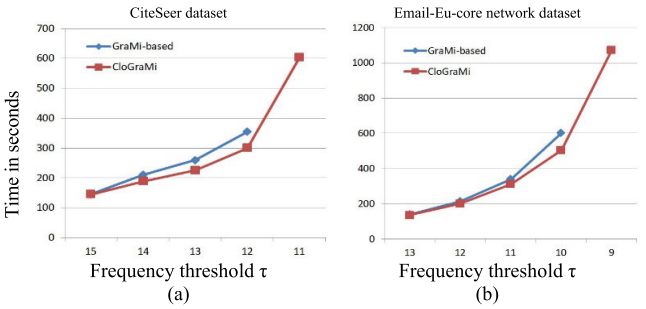


FIGURE 8. Running time for the two directed datasets.

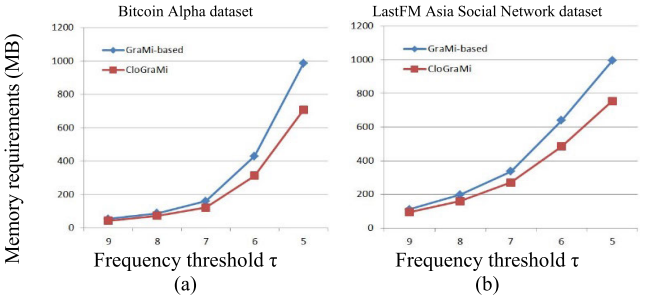


FIGURE 9. Memory requirements for two undirected datasets.

+ For the Bitcoin Alpha dataset (Figure 7.a): The efficiency of CloGraMi increases gradually by the experimental thresholds, the running time can be reduced to 73.7% when compared to GraMi. At threshold $\tau = 5$, GraMi needs 639.411 seconds to find 584 frequent subgraphs and filter out 420 closed subgraphs (in Figure 5.a), while CloGraMi needs 471.808 seconds to find closed subgraphs directly without filtering.

+ For the LastFM Asia Social Network dataset (Figure 7.b): Because the number of candidates and closed subgraphs in this dataset is not as great as with the Bitcoin Alpha dataset (in Figure 5.b), the efficiency of CloGraMi increases gradually when we reduce the experimental threshold, but it is not as good as that with the previous dataset. CloGraMi can reduce running time by 77.3% in comparison to GraMi. In particular, at the threshold $\tau = 5$, it took GraMi 648.453 seconds to find 531 frequent subgraphs and filter out 504 closed subgraphs, while CloGraMi needed 501.594 seconds to find closed subgraphs directly.

+ For CiteSeer dataset (Figure 8.a): CloGraMi only reduced the running time to 84.4% that needed for GraMi at threshold $\tau = 12$, as CloGraMi required 300.304 seconds and GraMi needed 355.075 seconds. In particular, at the end threshold $\tau = 11$ GraMi could not run because it was out of available memory, but CloGraMi still found 146 closed subgraphs (Figure 6.a) with 602.698 seconds for the running time.

+ For the Email-Eu-core network dataset (Figure 8.b): Similar to the directed graph dataset above, CloGraMi can only need 83.7% of the running time of GraMi at threshold $\tau = 10$, as GraMi needed 602.008 seconds to execute but CloGraMi only needed 504.182 seconds. Especially at the last threshold $\tau = 9$, GraMi exceeded the available memory and could not execute, CloGraMi still found closed subgraphs with 1,072.109 seconds.

Our final comparison criterion is the memory requirements. As in Subsections IV.B and IV.C, CloGraMi does not need to store frequent subgraphs by execution steps, so the storage space is also more optimized than with the GraMi-based algorithm. As with the running time criterion, the greatest reduction of memory requirements is to 71.6% for the first dataset. On the three undirected datasets, the running time is reduced much more than on the two directed datasets. However, at the last thresholds of the two directed graphs, only CloGraMi can execute, and GraMi-base crashes because of exceeding available memory.

+ For the Bitcoin Alpha dataset (Figure 9.a): In this undirected dataset, CloGraMi shows the best performance of our five datasets, and the memory requirements can be reduced to 71.6% in comparison to the memory needed by GraMi. Particularly, at the threshold $\tau = 5$, GraMi needs 986.588 MB and finds 584 frequent subgraphs, while CloGraMi only needs 706.717 MB and directly finds 420 closed subgraphs.

+ For the LastFM Asia Social Network dataset (Figure 9.b): CloGraMi requires 75.8% of the computer memory that GraMi needs at threshold $\tau = 5$, CloGraMi needs 755.547 MB and finds 504 closed subgraphs but GraMi needs 996.152MB to find 531 frequent subgraphs and carry out filtering of the closed subgraphs after searching.

+ For the CiteSeer dataset (Figure 10.a): CloGraMi requires 80.3% of the memory of GraMi with a small threshold. At threshold $\tau = 12$, GraMi needs 936.122 MB while CloGraMi only needs 752.315 MB, and at the last threshold, $\tau = 11$, GraMi could not execute because it exceeded the available memory, while CloGraMi needed 986.142 MB and found 146 closed subgraphs.

+ For the Email-Eu-core network dataset (Figure 10.b): CloGraMi can reduce memory to 76.8% that needed by GraMi at the threshold $\tau = 10$, as GraMi needs 949.832 MB to perform two steps: searching for frequent subgraphs and filtering closed subgraphs, while CloGraMi consumes only 730.237 MB. Specifically, at the end threshold $\tau = 9$, GraMi is no longer executable because of exceeding the

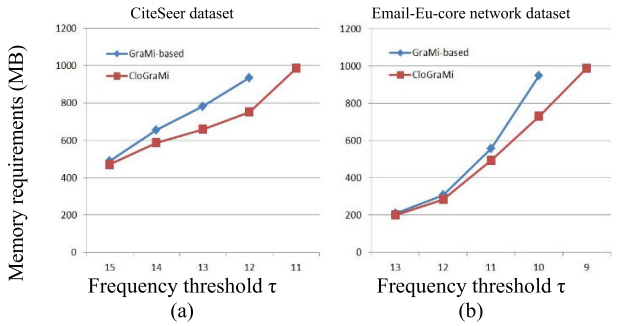


FIGURE 10. Memory requirements for two directed datasets.

available memory, while CloGraMi needs 989.667 MB of the computer's memory.

To test the scalability of our CloGraMi algorithm, we use the GitHub Social network dataset, which is the largest dataset in Table 3, and divide it into four parts of increasing size. We implement on all of them and record and compare the results on three criteria, as with all the aforementioned datasets.

With the first criterion, our CloGraMi algorithm can reduce the number of candidates in the filtering process of mined frequent subgraphs to get all closed subgraphs. The larger the dataset, the lower the threshold, the better the CloGraMi algorithm is in comparison to the GraMi-based one. Moreover, with regard to three large parts – 27,000 nodes, 32,000 nodes and 37,700 nodes – the GraMi-based approach cannot execute at the last threshold because of exceeding the available memory, but CloGraMi is still able to run and we are still able to record the number of frequent and closed subgraphs. CloGraMi can reduce the number of candidates to 88.1% in comparison to the number with the GraMi-based approach, as seen in Figure 11.

As with the four above datasets, CloGraMi also reduces the running time in all the four parts of the large dataset. Running time can be reduced to 85.8% compared to that of GraMi-based as shown in Figure 12. In each part of the GitHub Social Network dataset, the lower the frequency threshold, the greater the reduction in running time due to the large number of pruned candidates.

The final criterion for our comparison is the memory requirements. Because CloGraMi does not need to store the list of frequent subgraphs, passes by the filtering process of closed subgraphs, and it has an efficient strategy of early pruning of non-closed subgraphs, memory requirements can be significantly reduced. On all four parts of the dataset the memory needed is reduced at lower thresholds, and CloGraMi can be reduced to between 70% and 80%, as shown in Figure 13. In particular, for the three large parts 27,000 nodes, 32,000 nodes and 37,000 nodes, at the last frequency thresholds GraMi-based always crashes because of exceeding the available memory, but CloGraMi can still execute.

Finally, we compare the performances of the CloGraMi to GraMi-based algorithm by increasing size of the GitHub

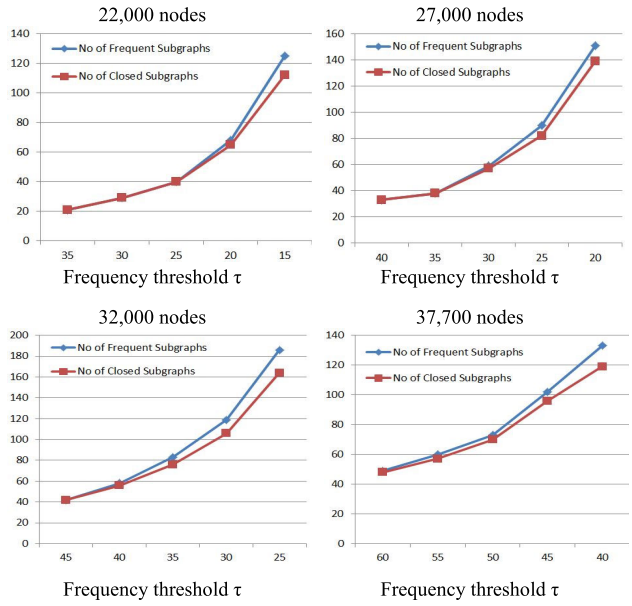


FIGURE 11. The numbers of frequent and closed subgraphs of four parts of the GitHub Social Network dataset.

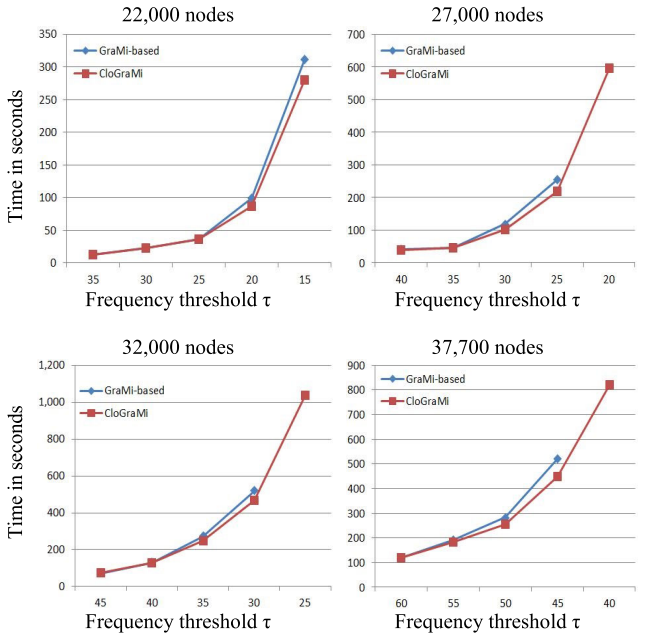


FIGURE 12. Running time for the four parts of GitHub Social Network dataset.

Social Network dataset (22,000 nodes, 27,000 nodes, 32,000 nodes and 37,700 nodes). With different dataset sizes, the algorithms can run on different frequency thresholds, therefore, we choose a frequency threshold as 0.11% of the dataset sizes to unify frequency thresholds to test. The tested thresholds are rounded based on the size of the parts of dataset and the selected percentage as shown in Table 4.

We choose these thresholds to illustrate and compare the performances including the runtime (in the Figure 14.a) and

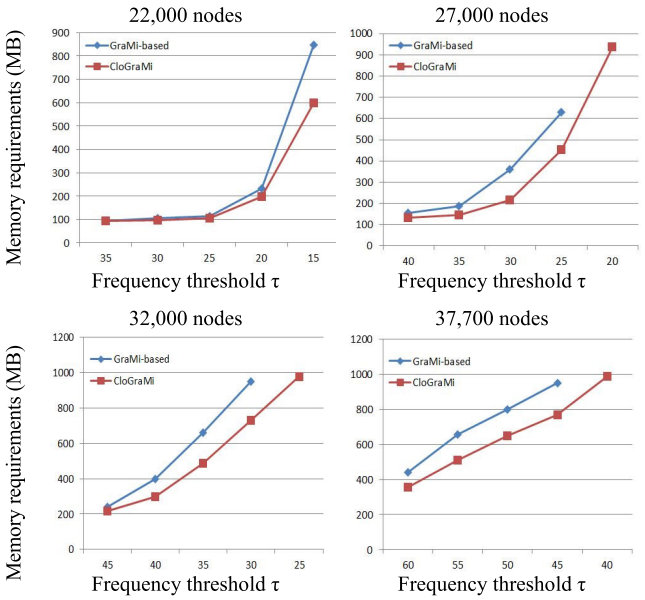


FIGURE 13. Memory requirements for four parts of GitHub Social Network dataset.

TABLE 4. The tested thresholds for four parts of GitHub Social Network dataset.

The size of dataset	The percentage rate (%)	The expected thresholds	The tested thresholds
22,000 nodes	0.11	24.2	24
27,000 nodes	0.11	29.7	30
32,000 nodes	0.11	35.2	35
37,700 nodes	0.11	41.47	41

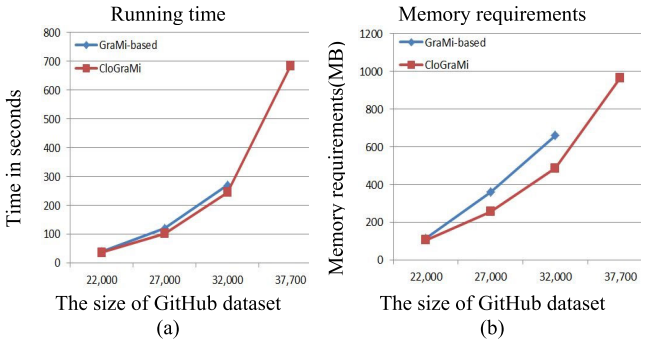


FIGURE 14. Performances of the algorithms on GitHub Social Network dataset with increasing size.

the memory requirements (in the Figure 14.b) of the two algorithms.

At the same threshold 0.11%, the larger the data size, the more effective CloGraMi is. Similarly to the comparisons above, CloGraMi can reduce the running time to 85.6% and the memory requirements to 73.8% those of GraMi-based, moreover, GraMi-based always has lower performance and crashes early in comparison to CloGraMi.

VI. CONCLUSIONS AND FUTURE WORKS

We propose an important problem of finding closed subgraphs based on the GraMi algorithm. In this paper, we also apply two more effective strategies to improve the performance of our algorithm CloGraMi, which are the level order traversal strategy to quickly determine closed subgraph, and a condition is set to prune non-closed subgraphs early. With three contributions, our CloGraMi algorithm has shown to be more efficient than the original algorithm with all three comparison criteria: the number of candidates to check, the running time and the memory requirements compared to the baseline (GraMi-based) algorithm.

In the future, we will continue to research and propose new methods to innovate and improve the performance of the closed subgraphs mining algorithm in a single large graph, such as: defining a feature ensuring the Downward Closure Property (DCP) in order to prune non-closed subgraphs, parallel processing on many branches in the search tree at the same time, and defining and mining closed subgraphs on a weighted single large graph. In GraMi approach, the domain of all subgraph candidates stores only values of the nodes in the large graph, this leads to some inconveniences of lacking information of edges. Thus a new approach including nodes/edges of candidates is very important for further researches.

REFERENCES

[1] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proc. 20th Int. Conf. Very Large Data Bases, (VLDB)*, vol. 1215, 1994, pp. 487–499.
[2] Z.-H. Deng, "DiffNodesets: An efficient structure for fast mining frequent itemsets," *Appl. Soft Comput.*, vol. 41, pp. 214–223, Apr. 2016.
[3] N. Aryabarzan, B. Minaei-Bidgoli, and M. Teshnehlab, "negFIN: An efficient algorithm for fast mining frequent itemsets," *Expert Syst. Appl.*, vol. 105, pp. 129–143, Sep. 2018.
[4] B. Vo, T.-P. Hong, and B. Le, "DBV-miner: A dynamic bit-vector approach for fast mining frequent closed itemsets," *Expert Syst. Appl.*, vol. 39, no. 8, pp. 7196–7206, Jun. 2012.
[5] M. J. Zaki and C. J. Hsiao, "Efficient algorithms for mining closed itemsets and their lattice structure," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 4, pp. 462–478, Apr. 2005.
[6] L. T. T. Nguyen, V. V. Vu, M. T. H. Lam, T. T. M. Duong, L. T. Manh, T. T. T. Nguyen, B. Vo, and H. Fujita, "An efficient method for mining high utility closed itemsets," *Inf. Sci.*, vol. 495, pp. 78–99, Aug. 2019.
[7] T. Le and B. Vo, "An N-list-based algorithm for mining frequent closed patterns," *Expert Syst. Appl.*, vol. 42, no. 19, pp. 6648–6657, Nov. 2015.
[8] B. Vo, L. V. Nguyen, V. V. Vu, M. T. H. Lam, T. T. M. Duong, L. T. Manh, T. T. T. Nguyen, L. T. T. Nguyen, and T.-P. Hong, "Mining correlated high utility itemsets in one phase," *IEEE Access*, vol. 8, pp. 90465–90477, 2020.
[9] M. Nouioua, P. Fournier-Viger, C.-W. Wu, J. C.-W. Lin, and W. Gan, "FHUQI-Miner: Fast high utility quantitative itemset mining," *Appl. Intell.*, vol. 51, pp. 6785–6809, Feb. 2021.
[10] P. A. Reddy and M. H. M. K. Prasad, "High utility item-set mining from retail market data stream with various discount strategies using EGUI-tree," *J. Ambient Intell. Humanized Comput.*, pp. 1–12, 2021, doi: 10.1007/s12652-021-03341-3.
[11] Q. Song, Y. Wu, P. Lin, L. X. Dong, and H. Sun, "Mining summaries for knowledge graph search," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 10, pp. 1887–1900, Oct. 2018.
[12] M. Haghir Chehreghani, T. Abdessalem, A. Bifet, and M. Bouzbila, "Sampling informative patterns from large single networks," *Future Gener. Comput. Syst.*, vol. 106, pp. 653–658, May 2020.

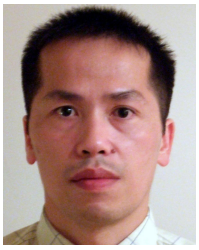
[13] Y. Chen, X. Zhao, X. Lin, Y. Wang, and D. Guo, "Efficient mining of frequent patterns on uncertain graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 2, pp. 287–300, Feb. 2019.
[14] R. Iqbal, F. Doctor, B. More, S. Mahmud, and U. Yousuf, "Big data analytics and computational intelligence for cyber–physical systems: Recent trends and state of the art applications," *Future Gener. Comput. Syst.*, vol. 105, pp. 766–778, Apr. 2020.
[15] X. Yan and J. Han, "CloseGraph: Mining closed frequent graph patterns," in *Proc. 9th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2003, pp. 286–295.
[16] N. E. I. Karabadji, S. Aridhi, and H. Seridi, "A closed frequent subgraph mining algorithm in unique edge label graphs," in *Proc. Int. Conf. Mach. Learn. Data Mining Pattern Recognit.*, 2016, pp. 43–57.
[17] A. Bendimerad, M. Plantevit, and C. Robardet, "Mining exceptional closed patterns in attributed graphs," *Knowl. Inf. Syst.*, vol. 56, no. 1, pp. 1–25, Jul. 2018.
[18] N. Acosta-Mendoza, A. Gago-Alonso, J. A. Carrasco-Ochoa, J. F. Martínez-Trinidad, and J. E. Medina-Pagola, "Mining generalized closed patterns from multi-graph collections," in *Proc. Iberoamerican Congr. Pattern Recognit.* Cham, Switzerland: Springer, 2017, pp. 10–18.
[19] J. Demetrovics, H. M. Quang, N. V. Anh, and V. D. Thi, "An optimization of closed frequent subgraph mining algorithm," *Cybern. Inf. Technol.*, vol. 17, no. 1, pp. 3–15, Mar. 2017.
[20] Z. A. Ansari and M. Abulaish, "An efficient subgraph isomorphism solver for large graphs," *IEEE Access*, vol. 9, pp. 61697–61709, 2021.
[21] E. Abdelhamid, I. Abdelaziz, Z. Khayyat, P. Kalnis, X. Wang, and P. Kapanipathi, "Pivoted subgraph isomorphism: The optimist, the pessimist and the realist," in *Proc. EDBT*, 2019, pp. 361–372.
[22] S. J. Nejad, F. Ahmadi-Abkenari, and P. Bayat, "A combination of frequent pattern mining and graph traversal approaches for aspect elicitation in customer reviews," *IEEE Access*, vol. 8, pp. 151908–151925, 2020.
[23] F. Jie, C. Wang, F. Chen, L. Li, and X. Wu, "A framework for subgraph detection in interdependent networks via graph block-structured optimization," *IEEE Access*, vol. 8, pp. 157800–157818, 2020.
[24] H. Guan, Q. Zhao, Y. Ren, and W. Nie, "View-based 3D model retrieval by joint subgraph learning and matching," *IEEE Access*, vol. 8, pp. 19830–19841, 2020.
[25] V. Karwa, S. Raskhodnikova, A. Smith, and G. Yaroslavtsev, "Private analysis of graph structure," *ACM Trans. Database Syst.*, vol. 39, no. 3, pp. 1–33, Oct. 2014.
[26] S. Velampalli and V. R. M. Jonnalagedda, "Frequent SubGraph mining algorithms: Framework, classification, analysis, comparisons," in *Data Engineering and Intelligent Computing*. Singapore: Springer, 2018, pp. 327–336.
[27] A. Borrego, D. Ayala, I. Hernández, C. R. Rivero, and D. Ruiz, "CAFE: Knowledge graph completion using neighborhood-aware features," *Eng. Appl. Artif. Intell.*, vol. 103, Aug. 2021, Art. no. 104302.
[28] J. Fox, T. Roughgarden, C. Seshadhri, F. Wei, and N. Wein, "Finding cliques in social networks: A new distribution-free model," *SIAM J. Comput.*, vol. 49, no. 2, pp. 448–464, Jan. 2020.
[29] B. P. L. Lau, A. K. Singh, and T. P. L. Tan, "A review on dependence graph in social reasoning mechanism," *Artif. Intell. Rev.*, vol. 43, no. 2, pp. 229–242, Feb. 2015.
[30] M. Elseidy, E. Abdelhamid, S. Skiadopoulos, and P. Kalnis, "GraMi: Frequent subgraph and pattern mining in a single large graph," *Proc. VLDB Endowment*, vol. 7, no. 7, pp. 517–528, Mar. 2014.
[31] E. Abdelhamid, "Scalable frequent subgraph mining," Ph.D. dissertation, King Abdullah Univ. Sci. Technol., Saudi Arabia, 2017.
[32] J. R. Ullmann, "An algorithm for subgraph isomorphism," *J. ACM*, vol. 23, no. 1, pp. 31–42, Jan. 1976.
[33] L. B. Q. Nguyen, B. Vo, N.-T. Le, V. Snasel, and I. Zelinka, "Fast and scalable algorithms for mining subgraphs in a single large graph," *Eng. Appl. Artif. Intell.*, vol. 90, Apr. 2020, Art. no. 103539.
[34] N.-T. Le, B. Vo, L. B. Q. Nguyen, H. Fujita, and B. Le, "Mining weighted subgraphs in a single large graph," *Inf. Sci.*, vol. 514, pp. 149–165, Apr. 2020.
[35] E. Abdelhamid, M. Canim, M. Sadoghi, B. Bhattacharjee, Y. Chang, and P. Kalnis, "Incremental frequent subgraph mining on large evolving graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 12, pp. 2710–2723, Dec. 2017.
[36] X. Yan and J. Han, "gSpan: Graph-based substructure pattern mining," in *Proc. IEEE Int. Conf. Data Mining*, Dec. 2002, pp. 721–724.
[37] E. Abdelhamid, I. Abdelaziz, P. Kalnis, Z. Khayyat, and F. Jamour, "ScaleMine: Scalable parallel frequent subgraph mining in a single large graph," in *Proc. SC Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2016, pp. 716–727.
[38] F. Qiao, X. Zhang, P. Li, Z. Ding, S. Jia, and H. Wang, "A parallel approach for frequent subgraph mining in a single large graph using spark," *Appl. Sci.*, vol. 8, no. 2, p. 230, Feb. 2018.
[39] N. Talukder and M. J. Zaki, "Parallel graph mining with dynamic load balancing," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2016, pp. 3352–3359.
[40] J. Li, Y. Liu, J. Pan, P. Zhang, W. Chen, and L. Wang, "Map-balance-reduce: An improved parallel programming model for load balancing of Mapreduce," *Future Gener. Comput. Syst.*, vol. 105, pp. 993–1001, Apr. 2020.
[41] S. Reinhardt and G. Karypis, "A multi-level parallel implementation of a program for finding frequent patterns in a large sparse graph," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, Mar. 2007, pp. 1–8.
[42] C. H. C. Teixeira, A. J. Fonseca, M. Serafini, G. Siganos, M. J. Zaki, and A. Aboulnaga, "Arabesque: A system for distributed graph mining," in *Proc. 25th Symp. Operating Syst. Princ.*, Oct. 2015, pp. 425–440.
[43] N. Talukder and M. J. Zaki, "A distributed approach for graph mining in massive networks," *Data Mining Knowl. Discovery*, vol. 30, no. 5, pp. 1024–1052, Sep. 2016.
[44] X. Zhao, Y. Chen, C. Xiao, Y. Ishikawa, and J. Tang, "Frequent subgraph mining based on Pregel," *Comput. J.*, vol. 59, no. 8, pp. 1113–1128, Aug. 2016.
[45] S. Ranu and A. K. Singh, "GraphSig: A scalable approach to mining significant subgraphs in large graph databases," in *Proc. IEEE 25th Int. Conf. Data Eng.*, Mar. 2009, pp. 844–855.
[46] X. Fan, Y. Li, J. Sun, Y. Zhao, and G. Wang, "Effective and efficient Steiner maximum path-connected subgraph search in large social Internet of Things," *IEEE Access*, vol. 9, pp. 72820–72834, 2021.
[47] C. Bodnar, C. Cangea, and P. Liò, "Deep graph mapper: Seeing graphs through the neural lens," *Frontiers Big Data*, vol. 4, Jun. 2021, Art. no. 680535, doi: 10.3389/fdata.2021.680535.
[48] Q.-T. Bui, B. Vo, V. Snasel, W. Pedrycz, T.-P. Hong, N.-T. Nguyen, and M.-Y. Chen, "SFCM: A fuzzy clustering algorithm of extracting the shape information of data," *IEEE Trans. Fuzzy Syst.*, vol. 29, no. 1, pp. 75–89, Jan. 2021.
[49] Q.-T. Bui, B. Vo, H.-A.-N. Do, N. Q. V. Hung, and V. Snasel, "F-mapper: A fuzzy mapper clustering algorithm," *Knowl.-Based Syst.*, vol. 189, Feb. 2020, Art. no. 105107.
[50] D. Wu, J. Ren, and L. Sheng, "Uncertain maximal frequent subgraph mining algorithm based on adjacency matrix and weight," *Int. J. Mach. Learn. Cybern.*, vol. 9, no. 9, pp. 1445–1455, Sep. 2018.
[51] S. Salem, M. Alokshiya, and M. A. Hasan, "RASMA: A reverse search algorithm for mining maximal frequent subgraphs," *BioData Mining*, vol. 14, no. 1, pp. 1–23, Dec. 2021.
[52] Y. Li, Q. Lin, R. Li, and D. Duan, "TGP: Mining top-K frequent closed graph pattern without minimum support," in *Proc. Int. Conf. Adv. Data Mining Appl.*, 2010, pp. 537–548.
[53] B. Güvenoglu and B. E. Bostanoglu, "A qualitative survey on frequent subgraph mining," *Open Comput. Sci.*, vol. 8, no. 1, pp. 194–209, Dec. 2018.
[54] P. Fournier-Viger, G. He, C. Cheng, J. Li, M. Zhou, J. C. Lin, and U. Yun, "A survey of pattern mining in dynamic graphs," *WIREs Data Mining Knowl. Discovery*, vol. 10, no. 6, Nov. 2020, Art. no. e1372.
[55] W. Fan, C. Hu, X. Liu, and P. Lu, "Discovering graph functional dependencies," *ACM Trans. Database Syst.*, vol. 45, no. 3, pp. 1–42, Sep. 2020.

LAM B. Q. NGUYEN received the M.Sc. degree in information systems from the University of Science, Vietnam National University, Ho Chi Minh City, Vietnam, in 2015. He is currently pursuing the Ph.D. degree with the Technical University of Ostrava, Czech Republic. His research interests include graph mining, data mining, parallel processing, machine learning, and distributed computing.

LOAN T. T. NGUYEN received the B.Sc. and M.Sc. degrees in computer science from Vietnam National University, Ho Chi Minh City, Vietnam, in 2002 and 2008, respectively, and the Ph.D. degree in computer science from the Wroclaw University of Technology, Poland, in 2015. From October 2016 to September 2017, she was an ERCIM Postdoctoral Researcher with the University of Warsaw, Poland. She was a Visiting Researcher with NTNU, Norway, in March 2017. She is currently a Lecturer with the School of Computer Science and Engineering, International University—VNU-HCM, Vietnam. Her research interests include association rules, classification, and mining in incremental databases.

IVAN ZELINKA (Member, IEEE) is currently a Professor of computer science at Modeling Evolutionary Algorithms Simulation and Artificial Intelligence, Faculty of Electrical and Electronics Engineering, Ton Duc Thang University, Ho Chi Minh City, Vietnam, and the Department of Computer Science, FEECS, VSB-Technical University of Ostrava, Ostrava, Czech Republic. He is a Chartered IT Professional at the British Computer Society (www.bcs.org/). His field of expertise is artificial intelligence, soft-computing, and cybersecurity. He is a supervisor of a few research projects, including international projects. He is also the supervisor of a research lab that consists of three professors and Ph.D. students working on different projects. He is an editor of a few journals and an editor and a founder of a series of books (http://www.springer.com/series/10624).

VACLAV SNASEL (Senior Member, IEEE) currently works in a multi-disciplinary environment involving artificial intelligence, multidimensional data indexing, conceptual lattice, information retrieval, semantic web, knowledge management, data compression, machine intelligence, neural networks, web intelligence, data mining, and applied to various real-world problems. His research and development experience includes over 25 years in the industry and academia. He has given more than ten plenary lectures and conference tutorials in these areas. He has authored/coauthored several refereed journal/conference papers and book chapters. He has published more than 400 papers (147 is recorded at Web of Science). He has supervised many Ph.D. students from Czech Republic, Jordan, Yemen, Slovakia, Ukraine, and Vietnam.



HUNG SON NGUYEN received the Ph.D. and D.Sci. (Habilitation) degrees, in 1997 and 2008, respectively. He is currently working as a Professor at the University of Warsaw. His main research interests include fundamentals and applications of rough set theory, data mining, text mining, bioinformatics, intelligent multiagent systems, soft computing, and pattern recognition. On these topics, he has published more than 140 research papers in edited books and international journals and conferences. He was involved in numerous research and commercial projects, including dialog-based search engine (Nutech), fraud detection for Bank of America (Nutech), logistic project for General Motors (Nutech), Semantic Search Engine, Intelligent Decision Support System for Firefighting in Poland, and RID—Development of Innovative Transport System and Recommendation System for Fashion and Cosmetic Branches. He is a fellow of the International Rough Set Society and a member of the Editorial Board of international journals, including, *Transactions on Rough Sets*, *Data mining and Knowledge Discovery* (2005–2008), and *ERCIM News*, and *Computational Intelligence*. He is the Manager Editor of *Fundamenta Informaticae*. He has served as the Program Co-Chair for RSCTC'06, RSKT2012, and IJCRS2018, a PC Member for various other conferences, including PKDD, PAKDD, AAMAS, RSCTC, RSFDGrC, and RSKT, and a reviewer for many other journals.

BAY VO received the B.Sc., M.Sc., and Ph.D. degrees in computer science from the University of Science, Vietnam National University, Ho Chi Minh City, Vietnam, in 2002, 2005, and 2011, respectively. He is currently an Associate Professor and the Dean of the Faculty of Information Technology, Ho Chi Minh City University of Technology (HUTECH), Vietnam. His research interests include association rules, classification, mining in incremental database, distributed databases, graph mining, social network analysis and mining, and preserving privacy in data mining. He was a PC Member of several conferences, such as IJCAI, PAKDD, ICONIP, SMC, ICCCI, and ACIIDS. He is a Manager Editor of *Vietnam Journal of Computer Science* and an Associate Editor of *ICIC Express Letters, Part B: Applications*.