ISSN: 1803-3814 (Printed), 2571-3701 (Online)
https://doi.org/10.13164/mendel.2021.2.090

CCGraMi: An Effective Method for Mining Frequent Subgraphs in a Single Large Graph

Lam B.Q. Nguyen[1,2]✉, **Ivan Zelinka**[3,2], **Quoc Bao Diep**[2]

[1]Faculty of Information and Communications, Kien Giang University, Kien Giang, Vietnam
[2]Faculty of Electrical Engineering and Computer Science, VSB-Technical University of Ostrava, Czech Republic
[3]Faculty of Electrical and Electronics Engineering, Ton Duc Thang University, Ho Chi Minh City, Vietnam
nbqlam@gmail.com✉, ivan.zelinka@tdtu.edu.vn, ivan.zelinka@vsb.cz, diepquocbao@gmail.com

Abstract

In modern applications, large graphs are usually applied in the simulation and analysis of large complex systems such as social networks, chemical structures, computer networks, maps, traffic networks. Therefore, graph mining is also an interesting subject attracting many researchers. Frequent subgraph mining (FSM) in a single large graph is one of the most important branches of graph mining, and FSM is defined as finding all subgraphs in a dataset whose occurrences are greater than or equal to a given frequency threshold. Among all algorithms for FSM, the GraMi algorithm is considered the state of the art and many algorithms have been proposed to improve this algorithm. In 2020, the SoGraMi algorithm was proposed to optimize the GraMi algorithm and presented an outstanding performance in terms of runing time and storage space. We propose a new algorithm in this paper to improve SoGraMi based on connected components, called CCGraMi (Connected Components GraMi). Our experiments on four real datasets (both directed and undirected) show that the proposed algorithm can outperform SoGraMi in terms of running time as well as memory requirements.

Keywords: *Data Mining, Pruning Techniques, Single Large Graph, Subgraph Mining, Weighted Subgraph.*

Received: 15 November 2021
Accepted: 20 December 2021
Published: 21 December 2021

1 Introduction

Large graphs are commonly used in practical applications such as social network mining [5], decision support systems [30], web mining [11, 26], map model analysis [6], consulting systems [25], criminal investigations [5], information retrieval systems, structural graph clustering [18], etc. Therefore, FSM plays an important role in research purposes as well as real-life applications [8, 31]. FSM for a large graph has attracted many researchers in recent years with many published studies [1, 10, 17, 23, 24, 29]. Since the graph is a non-linear structure and the complexity is NP-hard (nondeterministic polynomial time) [23], FSM has always been a challenging and interesting research area that attracts researchers [10, 15, 17, 19, 22, 23].

For a real-life example, a sale company collects and analysis data on its customers [3] to find frequent customer groups to fine-tune its business strategies [22]. The single large graph G in Fig. 1 illustrates the list of their customers, in which each customer is a node in the large graph belonging to a group labeled A, B, C, or D, and the relationship (labeled x, y, z, t or w) of two customers is indicated by edges of the two those nodes.

The main task of FSM algorithms is to find a set of all frequent subgraphs S in a large graph G, these are all subgraphs whose number of appearances in a large graph are greater than or equal to a given frequency threshold. A lot of methods search and count the number of isomorphisms for subgraph S in the large graph G. However, almost all the popular FSM approaches require two computationally expensive phases:

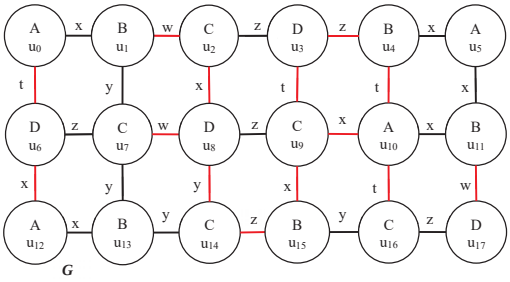

Figure 1: A large graph G

- Generating phase: in this phase, the mining process generates candidate subgraphs, and a frequent subgraph with k edges will generate candidate subgraphs with $(k+1)$ edges.
- Testing phase: this phase checks and counts the isomorphisms of each candidate subgraph, the mining process determines whether this candidate is frequent or not. However, in this phase, the computational cost is significantly high because

isomorphism processing is an NP-complete (non-deterministic polynomial) problem [10, 23].

In 2014, the GraMi [10] was proposed as an FSM algorithm from a single large graph. It is based on a novel approach, only storing the templates of candidates and searching isomorphisms to mark corresponding values in the domains of the templates, this approach does not fully list all of each candidate's appearances on the large graph [13]. GraMi had some optimizations to continuously enhance its performance: (1) *unique labels, push-down pruning, decomposition pruning*; (2) *lazy search*; (3) *automorphisms*. And then there are many methods introduced to improve this algorithm. ScaleMine algorithm [1] was proposed in 2016 as a parallel FSM, and SSIGRAM algorithm [24] was introduced in 2018 based on a new parallel approach using Spark. The two algorithms optimize GraMi by modeling distributed systems. In [23], PaGraMi was proposed as a new parallel approach, in which PaGraMi used multi-threads in a multi-core personal computer. In 2020, SoGraMi [23] was also developed to decrease the search space and improve the performance of the original GraMi.

Although SoGraMi can overcome the weaknesses of the original GraMi algorithm, this algorithm still has some disadvantages:

- The running time: because solving isomorphisms is an NP-complete problem [10, 23], this leads the time for search isomorphisms is extremely long.
- The memory requirements: the mining process consumes a lot of memory to store and evaluate, but the number of candidates is huge [17, 23].

In this paper, based on connected components we propose two effective strategies, which help to reduce the running time as well as the memory requirements and our algorithm improves the performance of the SoGraMi algorithm [23]. We call this algorithm CC-GraMi (Connected Components GraMi).

The rest of our paper is organized as follows: Section 2 surveys many related works of FSM. In Section 3, we present all the concepts and definitions of our strategies. Section 4 describes our two strategies and the algorithm in detail. Section 5 shows our experimental results on four directed/undirected graph datasets. Section 6 is our conclusion and some directions for further works.

2 Related Work

Based on gSpan [29] algorithm, GraMi [10] searches for isomorphisms of a candidate subgraph because gSpan dramatically reduces the memory requirement compared to previous grow-and-store approaches [16, 29]. The gSpan algorithm constructs a tree graph by using the DFS lexical order to represent all patterns, in the search tree, each node represents a DFS code, this is a hierarchical search space, called a DFS code tree [23]. The subgraph with size $(k+1)$ is created by adding an edge to the subgraph with k edge in the tree, a subgraph with size $(k+1)$ is corresponding to the level $(k+1)$ of this tree and these nodes contain the DFS code for subgraph k. Instead of retaining all detected subgraphs, the gSpan algorithm only keeps a list of each detected subgraph, and then the isomorphisms searching process is only applied for subgraphs in the list.

In 2014, the GraMi algorithm [10] was proposed for mining frequent subgraphs and patterns, it had optimizations such as unique labels [9, 23], push-down pruning [10, 23], decomposition pruning; lazy search; automorphisms. In which a new candidate will be constructed by adding an edge to a frequent subgraph, a frequent subgraph is a substructure of its generated candidate subgraphs. Based on minimum image-based support (MNI) the evaluation process of a candidate subgraph will stop when this subgraph has enough valid assignments to determine as frequent and the process ignores all remaining values to reduce the running time.

ScaleMine [1] was proposed in 2016, and SSIGRAM [24] was proposed in 2018. They were novel parallel algorithms for FSM in a single large graph in order to optimize the GraMi algorithm in distributed systems [28]. In which, the process divides mining tasks into separate CPU cores [1] or threads on a cluster [24]. ScaleMine was implemented on the Shaheen II system (a modest cluster on a high-end Cray XC40 supercomputer), while SSIGRAM was run on the Apache Spark framework. Besides, there are some existing parallel approaches [23, 28, 32], such as DistGraph [28], Pregel-based systems [32], they are too computationally expensive systems and complex to carry out. The SIGRAM algorithm [16] need to restore all computational intermediary steps of *MIS*, but their complexity is an NP-hard problem [23], therefore this method is extremely computationally expensive [27]. In a labeled sparse undirected graph, this algorithm uses *maximum independent sets (MIS)* metrics to mine all frequent subgraphs.

There are some algorithms which calculate the support for all mined subgraphs, such as gSpan [29], GraMi [10], O-FSM [7], ScaleMine [1], and SSIGRAM [24]. In isomorphism solving, there are several difficulties [7], because a lot of different appearances of a subgraph (isomorphisms) can overlap [10, 12]. A process called graph compression used by SumISO [21] is to group vertices into super vertices, SumISO only searches isomorphisms on these compressed representations of the graphs. There is some algorithms' goal that finds all frequent patterns by using inexact matching, such as APGM [14], this is an effective algorithm and it can mine frequent patterns with noise in real-life applications [5, 14, 20], the VEAM algorithm extends the APGM algorithm by a definition of approximate subisomorphism [2].

In 2020, the SoGraMi algorithm [23] was proposed to

optimize GraMi by a sorting strategy. It sorts all frequent edges in the large graph by their supports; prioritizes to extend small frequent edges; in each candidate subgraph, the process will prioritize to process small domain edges. This efficient strategy significantly reduces the number of generated candidate subgraphs, the running time, and the memory requirements in comparison to the original GraMi algorithm. In this paper, we continue to improve the SoGraMi algorithm [23] with two proposals based on the definition of connected components.

3 Concepts and Definitions

Definition 1 ([10, 23]). A *large graph* $G = (V, E, L)$ consists of a set of nodes V, a set of edges E, and a function L assigns labels to all the nodes/edges in the graph G.

Definition 2 ([10, 23]). A graph $S = (V_S, E_S, L_S)$ is a *subgraph* of $G = (V, E, L)$ if $V_S \subseteq V, E_S \subseteq E; L_S(v) = L(v), \forall v \in V_S; L_S((u, v)) = L((u, v)), \forall (u, v) \in E_S$.

Definition 3 ([10, 22]). Let $S = (V_S, E_S, L_S)$ be a subgraph in $G = (V, E, L)$. A *subgraph isomorphism* I of S to G is a function $f : V_S \to V$ satisfying: $L_S(v) = L(f(v)), \forall v \in V_S$; $(f(u), f(v)) \in E$ and $L_S((u, v)) = L((f(u), f(v))), \forall (u, v) \in E_S$.

To determine whether S is frequent in G or not, GraMi finds isomorphisms of S in G, evaluates its support [23] based on the number of these isomorphisms. The authors define solution [10] as an isomorphism [4] for a subgraph and a *constraint satisfaction problem* (CSP) is represented as a tuple (X, D, C) in which:

- X: an ordered set of variables (corresponding to nodes v in subgraph S)
- D: a set of domains corresponding to variables X
- C: a set of constraints between the variables in X. A solution for the CSP is an *assignment* to the variables in X, such that all constraints in C are satisfied.

For example, the subgraph S (in Fig. 2) to large graph G (in Fig. 1) CSP is defined:

$$\{(v_0, v_1, v_2),$$
$$\{\{u_0, u_5, u_{10}, u_{12}\}, \{u_1, u_4, u_{11}, u_{13}, u_{15}\}, \{u_2, u_7, u_9, u_{14}, u_{16}\}\},$$
$$\{v_0 \neq v_1 \neq v_2, L(v_0) = A, L(v_1) = B, L(v_2) = C, L(v_0, v_1) = x, L(v_1, v_2) = y\}\}.$$

In the CSP model, for each node $v \in S$ corresponding to each *variable* $x_v \in X$ has a *domain* D containing nodes u (in G) having the same node label as v in subgraph S, it means that "u can be assigned to v". For example subgraph S (in Fig.2) has three nodes v_0, v_1 and v_2; the variable v_0 has a domain $D(v_0) = u_0, u_5, u_{10}, u_{12}$, these nodes u have the same node label "A" with v_0, thus these nodes u can be assigned to v_0.

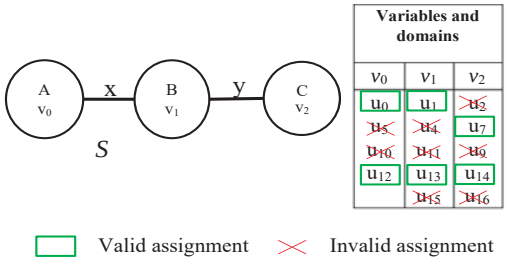

Figure 2: Valid and invalid assignments for subgraph S

Definition 4 ([23]). Assignment for a node u (in the domain D) to a node v (in subgraph S) is *valid* iff there exists an isomorphism I of S in large graph G that corresponding assigns u to v, and *invalid* otherwise.

The GraMi algorithm uses the minimum image-based support (MNI) [10] to evaluate each candidate subgraph whether that candidate is a frequent subgraph or not. The MNI support of S in G satisfies a given frequency threshold τ, $s_G(S) \geq \tau$, if every variable in X has at least τ distinct valid assignments. GraMi only searches for a number of isomorphisms of subgraph S in the large graph G that is enough to determine whether S is frequent [23], and it ignores all remaining isomorphisms to reduce the searching time.

Definition 5 ([23]). The *support* of subgraph S in large graph G (denoted by $s_G(S)$) is the minimum number of all distinct valid assignments in the domains of S.

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

SoGraMi proposes a sorting strategy to optimize the original GraMi algorithm. SoGraMi sort all frequent edges based on ascending order of their support, which means that the process prioritizes mining low-frequency subgraphs first; and in a subgraph, and the node with a smaller domain will be processed first. This sorting strategy [23] can significantly reduce the number of candidate subgraphs, decrease the runtime and the memory requirements.

For example: In Fig. 2, the domain of v_0 and v_1:
$|D(v_0)| = |u_0, u_5, u_{10}, u_{12}| = 4$
$|D(v_1)| = |u_1, u_4, u_{11}, u_{13}, u_{15}| = 5$
SoGraMi algorithm sorts the frequent edges list $fEdges$ (see the detail of $fEdges$ in Section 4.1), therefore the edge $A \overset{x}{-} B$ (smallest domain in the $fEdges$ list) will be processed first, and in edge $A \overset{x}{-} B$, node A (smaller domain in this edge) will be processed first. SoGraMi also searches for MNI support to determine whether S is frequent [23], and it also ignores all remaining isomorphisms to reduce the searching time.

Definition 6 ([23]). A subgraph S is *frequent* in the

large graph G if its support is greater than or equal to a given frequency threshold τ.

For example: valid and invalid assignments for S are shown in Fig. 2.

$$s_G(S) = \min(|D(v_0)|, |D(v_1)|, |D(v_2)|) =$$
$$\min(|u_0, u_{12}|, |u_1, u_{13}|, |u_7, u_{14}|) = \min(2, 2, 2) = 2$$

Because $s_G(S) \geq \tau$, S is a frequent subgraph in G.

Remark. *In this paper, we choose $\tau = 2$ for all our examples.*

Although the performance of SoGraMi significantly overcomes that of the original GraMi algorithm, it still needs a lot of runtime and storage because of searching values in a domain of a candidate subgraph, a node can combine with all other nodes in this domain to form isomorphisms. This leads to the long runtime and the large memory requirements.

In this paper, we continuously optimize our SoGraMi algorithm by two effective strategies based on connected components and we call it CCGraMi. In the first strategy, when evaluating a candidate subgraph, each node in its domain only combines to other nodes in the same connected component instead of combining to all available nodes in this domain, which decreases the runtime for the searching process (see detail in Subsection 4.1). In the second strategy, if a connected component does not have enough nodes to form an isomorphism, the process will delete this connected component to reduce the storage space (see detail in Subsection 4.2).

4 Proposal Algorithms

4.1 Searching Isomorphisms Based On Connected Components

In SoGraMi algorithm, after pruning all infrequent edges in the large graph G (in Fig. 1), we have the remaining nodes and edges are shown as in Fig. 3. Because of deleting some nodes/edges, a large graph can be split into many separated connected components. An isomorphism is a connected subgraph, thus its nodes cannot exist in two or more connected components. In the first contribution of CCGraMi, this algorithm searches and split all nodes in G to separate connected components, the process for searching isomorphism will determine isomorphisms on each connected component instead of searching on an entire large graph.

For example, the remaining nodes/edges in large graph G are split into five separated connected components such as C_1, C_2, C_3, C_4, and C_5.

In the SoGraMi algorithm, the mining process gets a frequent edge list $fEdge$ [23] that contains all edges whose number of appearances is equal to or greater than a given frequency threshold; sorting this list by

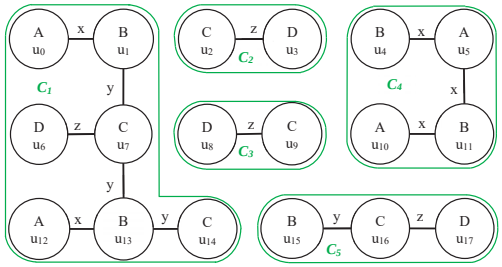

Figure 3: The frequent nodes/edges and five connected components in large graph G

the support of each edge to decrease the number of candidate subgraphs.

In Fig. 3, there are three edges in the $fEdges$ as follow:

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

The SoGraMi algorithm has an advantage in comparison to the original algorithm GraMi, SoGraMi sorts and prioritizes the edges with small domains when generating candidate subgraphs. This sorting strategy [23] can decrease the number of candidates and reduce the time to process each candidate. For example, edge $A \overset{x}{-} B$ has the smallest domain in the $fEdges$ list, and node A has a smaller domain than node B, thus node A will be processed first as in Fig. 4.

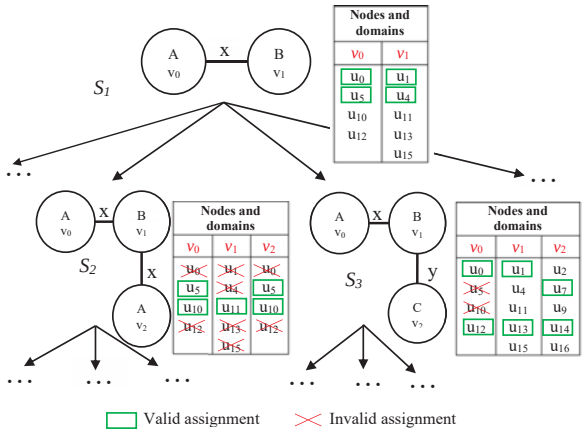

Figure 4: Some subgraphs' domains in the SoGraMi algorithm

Both GraMi and SoGraMi only search isomorphisms for a subgraph S until they find the MNI support of S in G that is enough to evaluate S as a frequent subgraph, they ignore all remaining isomorphisms to reduce the searching time [10, 22]. The domain of v_0 in S_3 has four nodes u_0, u_5, u_{10}, and u_{12}, the process performs sequentially these nodes. There are two isomorphisms of S_3 as u_0, u_1, u_7 and u_{12}, u_{13}, u_{14}, therefore, nodes u_0 and u_{12} are valid assignments, while u_5 and u_{10} are invalid assignments for v_0. After searching isomorphism for u_0, the mining process cannot search isomorphism for u_5 and u_{10} (in connected components C_4) these steps cost time but they do not get any valid

assignment. Meanwhile, u_{12} and u_0 are in the same connected component C_1 (in Fig. 3), but they are very far together in the domain to check together.

We propose the first strategy in this paper, because an isomorphism cannot exist in two different connected components, we split all nodes in the domain based on each connected component. For example, when processing subgraph S_6 in Fig. 5 (this is subgraph S_3 in Fig. 4), all nodes in its domain are split into five corresponding connected components. The mining process only searches isomorphisms on each connected component instead of the entire domain. After finding out two isomorphisms u_0, u_1, u_7 and u_{12}, u_{13}, u_{14}, that is enough to determine that S_6 is frequent, the searching process will stop to reduce running time.

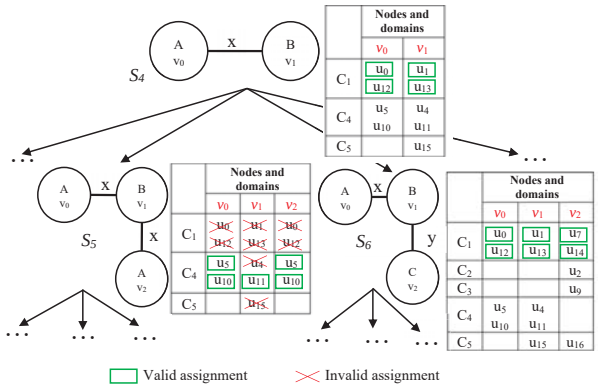

Figure 5: Some subgraphs' domains in CCGraMi

4.2 Pruning Subgraphs' Domains Based On Connected Components

The second contribution in this paper, we propose another strategy to prune subgraphs' domains based on connected components. Any connected component that does not have enough nodes to form an isomorphism will be pruned to reduce the storage space and the time to search isomorphism on it.

For example, subgraph S_7 and S_8 are the same subgraphs (they are S_3 in Fig.4 and S_6 in Fig. 5), but S_6 has four connected components which do not have enough nodes in the domain to form any isomorphism; C_2 and C_3 lack of node with label "A" and "B"; C_4 lacks of node with label "C"; C_5 lack of node with label "A". These connected components cannot form any isomorphism of $A \overset{x}{-} B \overset{y}{-} C$ (for subgraphs S_7 and S_8 in Fig. 6), deleting C_2, C_3, C_4 and C_5, we have the domain as S_8. In that, S_7 is a sample for a subgraph's domain of the SoGraMi algorithm and S_8 is a sample for the CCGraMi algorithm.

Early pruning all nodes in a domain that cannot form any isomorphism (it means they cannot be valid assignments), the mining process of CCGraMi can reduce the storage space as well as the runtime to search isomorphism for these nodes.

In the CCGraMi algorithm 1, after getting frequent

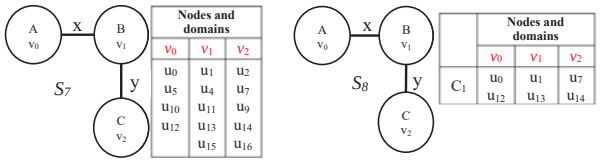

Figure 6: Comparision of subgraphs' domains

Algorithm 1 CCGraMi

Input: A graph G and a frequency threshold τ
Output: All subgraphs S in G where $s_G(S) \geq \tau$

1: resultList $\leftarrow \emptyset$
2: Let $fEdges$ be a set of all frequent edges in G
3: Sort $fEdges$ in ascending order based on the support
4: **for** each edge $ed \in fEdges$ **do**
5: resultList $\leftarrow$ resultList $\cup$ **SubgraphExtend** $(ed, G, \tau, fEdges)$
6: Remove ed from $fEdges$
7: **return** resultList

edges list $fEdges$ (Line 2), CCGraMi sort this list (Line 3), it likely to SoGraMi algorithm. From Line 4 to Line 6, the mining process extends these edges to form candidate subgraphs by **SubgraphExtend()** 2 [23].

In the function **SubgraphExtend()** 2, these are two phases of the mining process:

- Generating phase: From Line 2 to Line 6, this function combines each frequent subgraph with an edge in $fEdge$, the new candidate subgraph will be put into the *candidateSet* list.
- Testing phase: From Line 7 to Line 9, this function tests each generated subgraph in *candidateSet* by **IsFrequent()** function 3. If a candidate subgrpah is determined as frequent, this frequent subgraph will be extended recursively (Line 9) with all remaining edges in $fEdges$.

Algorithm 2 SubgraphExtend

Input: A subgraph S, a large graph G, a frequency threshold τ and a set of frequent edges $fEdges$ in G
Output: All frequent subgraphs in G extended from S

1: SList $\leftarrow S$, candidateSet $\leftarrow \emptyset$
2: **for** each edge $ed \in fEdges$ and node $v \in S$ **do**
3: **if** ed can extend v **then**
4: Let ext be an extended subgraph of S by ed
5: **if** ext is not already generated **then**
6: candidateSet $\leftarrow$ candidateSet $\cup$ ext
7: **for** $c \in$ candidateSet **do**
8: **if IsFrequent**$(c, G, \tau, fEdges)$ **then**
9: SList $\leftarrow$ SList $\cup$ **SubgraphExtend** $(c, G, \tau, fEdges)$
10: **return** SList

Algorithm 3 IsFrequent

Input: Subgraph S, large graph G and a frequency threshold τ and a set of frequent edges $fEdges$ in G
Output: true if S is a frequent subgraph of G, false otherwise

1: Let $ccList$ be a set of all connected components in $fEdges$
2: **for** each node v with domain D **do** count $\leftarrow \emptyset$
3: **if** the size of any domain is less than τ **then**
4: **return** false
5: **for** element u of D **do**
6: **for** each connected component $cc \in ccList$ and $u \in cc$ **do**
7: **if** cc does not have enough nodes for S **then**
8: Remove all node values in cc
9: **if** the size of any domain is less than τ **then**
10: **return** false
11: **if** u is already marked **then**
12: count++
13: **else if** existing an isomorphism I assign u to v **then**
14: Mark all nodes of I in corresponding domains
15: count++
16: **else**Remove u from the domain D
17: **if** count $= \tau$ **then**
18: Move to the next node v (Line 2)
19: **return** false ▷ count $< \tau$ and domain is exhausted
20: **return** true

In the **IsFrequent()** function 3, the process iterates all node v in a candidate subgraph S (Line 2); for each node u in the domain of this subgraph (Line 5), instead of combining this node u with all available nodes in the domain, our strategy only combine this node u with the nodes in the same connected component (Line 6). Moreover, if any connected component does not have enough nodes to form isomorphism I to subgraph S, our second strategy will remove all nodes in this connected component (from Line 7 to Line 10).

Let N and n be the number of nodes in large graph G and subgraph S, respectively, the frequency threshold is τ. We denote p_s and p_c as the possibilities that a node in the domain of a node is valid in the SoGraMi and CCGraMi, respectively. In total, the complexity bound of **IsFrequent()** function 3 in SoGraMi is $O(n.\tau/p_s.N^{n-1})$ [10, 23].

In our strategies, let V be the set of nodes and E be the set of edges in the large graph G, the well-known complexity of BFS for getting all connected components (Line 1 in **IsFrequent()** function 3) is $O(|E|+|V|)$. Because of early deleting node values in the domain, this leads to the smaller domain, thus finding τ valid assignments will be faster and it increases the possibility that p_c is greater than p_s.

Moreover, N is the number of all the nodes in the large graph G, but in CCGraMi, N will be split to k connected components. It means that $N = N_0 + N_1 + \cdots + N_{k-1}$. In total, the complexity bound of **IsFrequent()** function 3 in CCGraMi is $O(n.\tau/p_c.(N_0^{n-1}+N_1^{n-1}+\cdots+N_{k-1}^{n-1}))$. We have that:

$$p_c \geq p_s \quad (1)$$

$$N^{n-1} \geq (N_0^{n-1} + N_1^{n-1} + \cdots + N_{k-1}^{n-1}) \quad (2)$$

according to the property of polynomial expansion.

Based on Eq. 1 and Eq. 2: $O(n.\tau/p_s.N^{n-1}) \geq O(n.\tau/p_c.(N_0^{n-1} + N_1^{n-1} + \cdots + N_{k-1}^{n-1}))$, in the worst case, if $k = 1$ (there is only one connected component in the large graph G), the complexity of the two algorithms is equal.

5 Experimental Evaluation

In this section, we implement and compare the performances of GraMi, SoGraMi, and the new algorithm CCGraMi. All our experiments were conducted with the Java SE Development Kit 8, a system with Windows 10, using an Intel Core i5, 4 threads, 3.2GHz CPU, equipped with 4GB RAM. We compare three algorithms on criteria: runtime and memory requirements, on four datasets (both directed and undirected), our new algorithm CCGraMi shows that it can overcome the two previous algorithms.

We recorded the all results on four datasets (Tab. 1):

- MiCo: This is a large size undirected graph which contains 100,000 nodes and over one million edges. It describes Microsoft's co-author information, in which the nodes are the authors and their labels are the areas of interest, each edge in this graph represents the collaboration among two authors, the number of co-author papers is the edge label of two nodes. We use this dataset with the same information as in the SoGraMi paper to show the efficiency of our optimizations in the new algorithm CCGraMi [23].
- Facebook: This dataset is downloaded from http://snap.stanford.edu/data/ and it is an undi-

Table 1: The features of four datasets

Dataset	Type	Nodes	Node labels	Edges	Edge labels
MiCo	Undirected	100,000	29	1,080,298	106
Facebook	Undirected	4,389	20	88,235	36
p2p-Gnutella09	Directed	8,114	25	26,013	40
CiteSeer	Directed	3,312	6	4,732	101

rected dataset consisting of 4,389 nodes and 88,235 edges collected via the Facebook application from survey respondents. We use this dataset with the same information as in the SoGraMi paper to show the efficiency of the new algorithm CCGraMi. The original dataset does not have nodes and edges labels, we added randomly 20 distinct node labels and 36 distinct edge labels with the ratings in the SoGraMi paper [23].

- p2p-Gnutella09: This dataset is a medium-size directed dataset downloaded from http://snap.stanford.edu/data/. It is a sequence of snapshots for the Gnutella peer-to-peer file-sharing network. There are nine Gnutella network snapshots collected in August 2002. In which, the nodes represent the hosts in the Gnutella network topology, and the connections between the Gnutella hosts are the edges in this graph. Likely to Facebook dataset, we add these randomly 25 distinct node labels and 40 distinct edge labels with the following ratings in Tabs. 2 and 3, respectively.

- CiteSeer: This is a small size directed graph dataset including 3,312 publications, each publication is corresponding to a node and 4,732 citations between the publications (each citation corresponds to an edge). Each node has a label that presents a field of Computer Science and each edge has a label (from 0 to 100), this edge label is the similarity between two publications, in which the smaller the label, the higher the degree of similarity. Likely to MiCo and Facebook datasets, we use the CiteSeer dataset with the same information as in the SoGraMi paper to show the efficiency of the new algorithm CCGraMi [23].

Using these four datasets with undirected and directed graphs, we use different frequency thresholds τ to demonstrate that our new algorithm has outstanding performance in comparison to GraMi and SoGraMi algorithms, and we try to reduce the frequency thresholds τ until our personal computer cannot execute it any more.

First, we record and compare the runtime of three algorithms in all four datasets. In all datasets, the lower the threshold, the better the CCGraMi algorithm is in comparison to the two previous algorithms. Moreover, CCGraMi always crashes at lower thresholds in comparison to GraMi and SoGraMi.

The MiCo dataset is a large undirected dataset (in Fig. 7), because of the computing power and storage capacity of the computer, we only take the tests at high thresholds in our experiments. Our experiments did not complete at $\tau = 9,200$ because of exceeding the internal memory. At the lowest threshold $\tau = 9,250$, CCGraMi can reduce the runtime to 75.5% in comparison to that of SoGraMi, and 21.7% to original GraMi. All three algorithms crash at the threshold $\tau = 9200$.

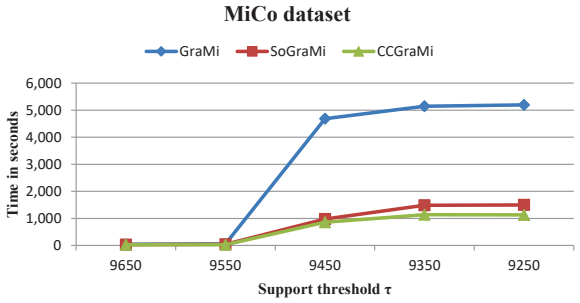

Figure 7: Running time for MiCo dataset

With the Facebook dataset, a medium undirected dataset (in Fig. 8), the mining process can implement at low thresholds. CCGraMi can reduce the runtime to 87.2% of that of SoGraMi, and 56.5% of that of the original GraMi at $\tau = 125$. At $\tau = 120$, CCGraMi's experiment can be implemented with 348.322 seconds, SoGraMi needs 404.538 seconds, while GraMi crashes because it exceeds the available memory.

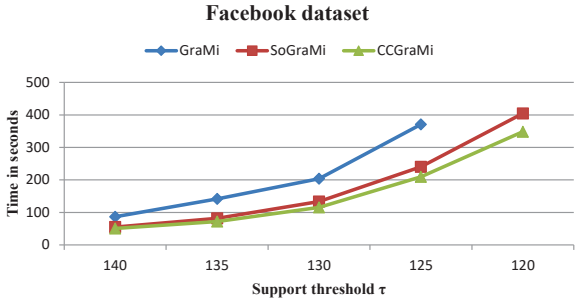

Figure 8: Running time for Facebook dataset

With the p2p-Gnutella09 dataset, this is a medium-directed dataset in Fig. 9. Our proposed algorithm shows that the running time at $\tau = 40$ can be decreased to 85.0% that of the SoGraMi, to 71.9% of the original GraMi. At $\tau = 35$, CCGraMi still runs and takes 442.531 seconds (82.9% in comparison to SoGraMi), SoGraMi costs 535.532 seconds, while GraMi is out of available memory at this threshold.

Table 2: The labels of nodes and their ratings

Labels of nodes	1	2	3	4	5	6	7	8	9	10	11	12	13
Rating (%)	20	15	10	7.5	6.5	6	4	3.5	3	2.5	2.5	2.5	2
Labels of nodes	14	15	16	17	18	19	20	21	22	23	24	25	-
Rating (%)	2	2	2	1.5	1.5	1.5	1	1	1	0.5	0.5	0.5	-

Table 3: The labels of edges and their ratings

Labels of edges	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Rating (%)	22	13	10	8	7	5	3	2.5	2.5	2.5	2	1.8	1.6	1.2
Labels of edges	15	16	17	18	19	20	21	22	23	24	25	26	27	28
Rating (%)	1.2	1.1	1.1	1.1	1	1	1	1	0.9	0.9	0.9	0.9	0.8	0.8
Labels of edges	29	30	31	32	33	34	35	36	37	38	39	40	-	-
Rating (%)	0.8	0.7	0.6	0.6	0.3	0.3	0.2	0.2	0.2	0.1	0.1	0.1	-	-

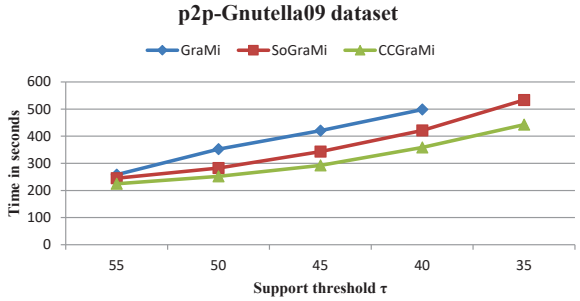

Figure 9: Running time for p2p-Gnutella09 dataset

With the CiteSeer dataset in Fig. 10, this is a small size directed dataset. Our proposed CCGraMi shows that the running time can be decreased to 78.3% that of the SoGraMi, to 66.7% of the original GraMi at the threshold $\tau = 12$. At $\tau = 11$, CCGraMi still runs and takes 490.653 seconds (77.6% in comparison to SoGraMi), SoGraMi costs 631.785 seconds, while GraMi is out of available memory at this threshold.

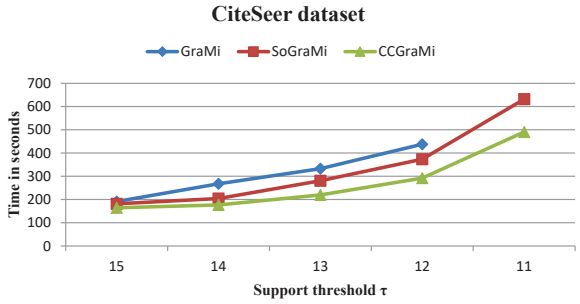

Figure 10: Running time for CiteSeer dataset

Finally, we record and compare the memory requirements for the three algorithms GraMi, SoGraMi, and CCGraMi. Because CCGraMi prunes the domain of each candidate subgraph based on connected components which cannot form any isomorphism for this candidate, the memory requirement is also significantly reduced.

For the MiCo dataset, because the numbers are small, the magnitude of our improvements (in Fig. 11) is not well illustrated. CCGraMi only reduces the memory requirements to 95.3% compared with SoGraMi, and 92.8% of that is needed for GraMi at the last threshold $\tau = 9,250$, and all the versions of algorithms crash at $\tau = 9,200$.

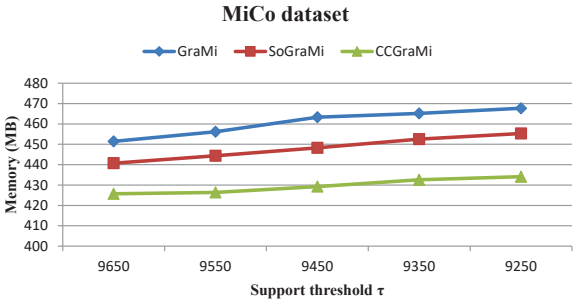

Figure 11: Memory requirements for MiCo dataset

For the medium undirected Facebook dataset (in Fig. 12), there are many candidates and our results show that the improvement of CCGraMi is significant. The memory requirement of the new algorithm can be reduced to 72.1% that of the SoGraMi, and to 39.1% of the original GraMi at the threshold $\tau = 120$. At the last threshold $\tau = 120$, the GraMi algorithm crashes because it exceeds the available memory, CCGraMi can reduce the memory requirements to 71.6% compared to SoGraMi, which only needs 478.402 MB, and SoGraMi needs 667.515 MB.

On the medium-size dataset p2p-Gnutella09, this is a directed dataset (in Fig. 13), because of a large number of generated candidate subgraphs, our approach shows significant results. At $\tau = 40$, the memory requirement can be reduced to 56.2% that of the original GraMi and 76.1% that of SoGraMi. At the last threshold $\tau = 35$, the original GraMi ran out of the available memory, SoGraMi needed 703.470 MB, while CCGraMi needs 75.7% of the storage space of SoGraMi, it only consumed 679.192 MB.

On the CiteSeer dataset, the memory reductions are more significantly at the lower thresholds τ (in Fig. 14).

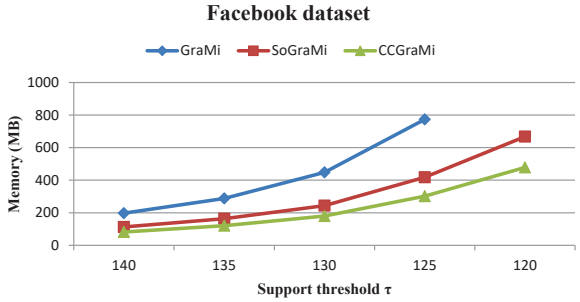

Figure 12: Memory requirements for Facebook dataset

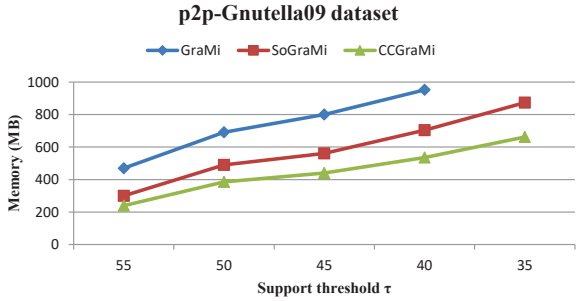

Figure 13: Memory requirements for p2p-Gnutella09 dataset

With $\tau = 12$, CCGraMi only consumes 82.1% and 68.1% in comparison to memory requirements of SoGraMi and GraMi. Especially, at the threshold $\tau = 11$, the original algorithm GraMi crashed because of exceeding available memory but SoGraMi and CCGraMi still worked, our new algorithm reduced the memory to 81.1% in comparison to that of SoGraMi, SoGraMi consumed 959.834 MB, CCGraMi only needed 778.652 MB.

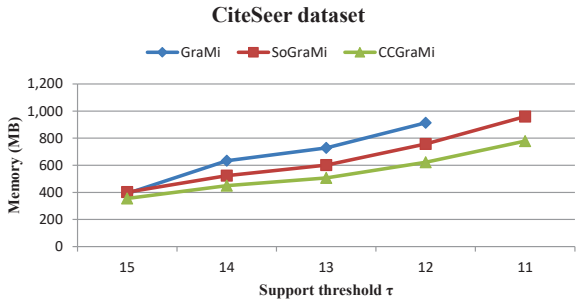

Figure 14: Memory requirements for CiteSeer dataset

6 Conclusions and Future Work

In this paper, we have proposed a new algorithm CCGraMi with two effective strategies based on connected components: Searching isomorphisms and Pruning subgraphs' domains. Our experiments on four real datasets (both directed and undirected graphs) showed that the proposed algorithm CCGraMi has good results compared to the original algorithm GraMi and optimized algorithm SoGraMi in terms of running time and memory requirements.

In the future, we will continue to research new methods based on connected components such as parallel processing on each connected component, arranging connected components by size to prioritize the processing of small-sized connected components to decrease storage space during the mining process, and combining high-powered computer systems to be able to mine larger graphs with smaller frequency thresholds.

Acknowledgement: The following grants are acknowledged for the financial support provided for this research: Grant of SGS No. SP2022/22, VSB-Technical University of Ostrava.

References

[1] Abdelhamid, E., Abdelaziz, I., Kalnis, P., Khayyat, Z., and Jamour, F. Scalemine: Scalable parallel frequent subgraph mining in a single large graph. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2016), pp. 716–727.

[2] Acosta-Mendoza, N., Gago-Alonso, A., and Medina-Pagola, J. E. Frequent approximate subgraphs as features for graph-based image classification. *Knowledge-Based Systems 27* (2012), 381–392.

[3] Amaranatha Reddy, P., and Hazarath Murali Krishna Prasad, M. High utility item-set mining from retail market data stream with various discount strategies using egui-tree. *Journal of Ambient Intelligence and Humanized Computing* (2021), 1–12.

[4] Ansari, Z. A., Jahiruddin, and Abulaish, M. An efficient subgraph isomorphism solver for large graphs. *IEEE Access 9* (2021), 61697–61709.

[5] Basu, K., Zhou, C., Sen, A., and Goliber, V. H. A novel graph analytic approach to monitor terrorist networks. In *2018 IEEE Intl Conf on Parallel Distributed Processing with Applications, Ubiquitous Computing Communications, Big Data Cloud Computing, Social Computing Networking, Sustainable Computing Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)* (2018), pp. 1159–1166.

[6] Daniel, C. B., Saravanan, S., and Mathew, S. Gis based road connectivity evaluation using graph theory. In *Transportation Research* (Singapore, 2020), T. V. Mathew, G. J. Joshi, N. R. Velaga, and S. Arkatkar, Eds., Springer Singapore, pp. 213–226.

[7] Dhiman, A., and Jain, S. Optimizing frequent subgraph mining for single large graph. *Procedia Computer Science 89* (2016), 378–385. Twelfth International Conference on Communication Networks, ICCN 2016, August 19– 21, 2016, Bangalore, India Twelfth International Conference on Data Mining and Warehousing, ICDMW 2016,

August 19-21, 2016, Bangalore, India Twelfth International Conference on Image and Signal Processing, ICISP 2016, August 19-21, 2016, Bangalore, India.

[8] Ding, R.-X., Wang, X., Shang, K., and Herrera, F. Social network analysis-based conflict relationship investigation and conflict degree-based consensus reaching process for large scale decision making using sparse representation. *Information Fusion 50* (2019), 251–272.

[9] El Islem Karabadji, N., Aridhi, S., and Seridi, H. A closed frequent subgraph mining algorithm in unique edge label graphs. In *Machine Learning and Data Mining in Pattern Recognition* (Cham, 2016), P. Perner, Ed., Springer International Publishing, pp. 43–57.

[10] Elseidy, M., Abdelhamid, E., Skiadopoulos, S., and Kalnis, P. Grami: Frequent subgraph and pattern mining in a single large graph. *Proc. VLDB Endow. 7*, 7 (mar 2014), 517–528.

[11] Faroughi, A., Morichetta, A., Vassio, L., Figueiredo, F., Mellia, M., and Javidan, R. Towards website domain name classification using graph based semi-supervised learning. *Computer Networks 188* (2021), 107865.

[12] Fiedler, M., and Borgelt, C. Subgraph support in a single large graph. In *Seventh IEEE International Conference on Data Mining Workshops (ICDMW 2007)* (2007), pp. 399–404.

[13] Iqbal, R., Doctor, F., More, B., Mahmud, S., and Yousuf, U. Big data analytics and computational intelligence for cyber–physical systems: Recent trends and state of the art applications. *Future Generation Computer Systems 105* (2020), 766–778.

[14] Jia, Y., Zhang, J., and Huan, J. An efficient graph-mining method for complicated and noisy data with real-world applications. *Knowledge and Information Systems 28*, 2 (2011), 423–447.

[15] Jung, J. J. Constraint graph-based frequent pattern updating from temporal databases. *Expert Systems with Applications 39*, 3 (2012), 3169–3173.

[16] Kuramochi, M., and Karypis, G. Finding frequent patterns in a large sparse graph. *Data mining and knowledge discovery 11*, 3 (2005), 243–271.

[17] Le, N.-T., Vo, B., Nguyen, L. B., Fujita, H., and Le, B. Mining weighted subgraphs in a single large graph. *Information Sciences 514* (2020), 149–165.

[18] Mai, S. T., et al. Scalable interactive dynamic graph clustering on multicore cpus. *IEEE Transactions on Knowledge and Data Engineering 31*, 7 (2019), 1239–1252.

[19] Matousek, R., Dobrovsky, L., and Kudela, J. How to start a heuristic? utilizing lower bounds for solving the quadratic assignment problem. *International Journal of Industrial Engineering Computations 13*, 2 (2022), 151–164.

[20] Mrzic, A., et al. Grasping frequent subgraph mining for bioinformatics applications. *BioData mining 11*, 1 (2018), 1–24.

[21] Nabti, C. E. *Subgraph Isomorphism Search In Massive Graph Data*. PhD thesis, Université de Lyon, 2017.

[22] Nguyen, L. B., Nguyen, L. T., Zelinka, I., Snasel, V., Nguyen, H. S., and Vo, B. A method for closed frequent subgraph mining in a single large graph. *IEEE Access* (2021), 1–1.

[23] Nguyen, L. B., Vo, B., Le, N.-T., Snasel, V., and Zelinka, I. Fast and scalable algorithms for mining subgraphs in a single large graph. *Engineering Applications of Artificial Intelligence 90* (2020), 103539.

[24] Qiao, F., Zhang, X., Li, P., Ding, Z., Jia, S., and Wang, H. A parallel approach for frequent subgraph mining in a single large graph using spark. *Applied Sciences 8*, 2 (2018).

[25] Ray, A., Holder, L., and Choudhury, S. Frequent subgraph discovery in large attributed streaming graphs. In *Proceedings of the 3rd International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications* (New York, New York, USA, 24 Aug 2014), W. Fan, A. Bifet, Q. Yang, and P. S. Yu, Eds., vol. 36 of *Proceedings of Machine Learning Research*, PMLR, pp. 166–181.

[26] Schenker, A., Last, M., Bunke, H., and Kandel, A. Classification of web documents using a graph model. In *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.* (2003), pp. 240–244 vol.1.

[27] Shahrivari, S., and Jalili, S. Distributed discovery of frequent subgraphs of a network using mapreduce. *Computing 97*, 11 (2015), 1101–1120.

[28] Talukder, N., and Zaki, M. J. A distributed approach for graph mining in massive networks. *Data Mining and Knowledge Discovery 30*, 5 (2016), 1024–1052.

[29] Yan, X., and Han, J. gspan: graph-based substructure pattern mining. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.* (2002), pp. 721–724.

[30] Yu, Y., Zhang, Z., Sun, R., Liu, H., Yuan, S., Jiang, T., Wu, M., Guo, C., Guo, Y., Weng, J., Zheng, X., and Yuan, F. Ai-guided resource allocation and rescue decision system for medical applications. *Future Generation Computer Systems 118* (2021), 485–491.

[31] Zeng, J., Yang, L. T., Lin, M., Ning, H., and Ma, J. A survey: Cyber-physical-social systems and their system-level design methodology. *Future Generation Computer Systems 105* (2020), 1028–1042.

[32] Zhao, X., Chen, Y., Xiao, C., Ishikawa, Y., and Tang, J. Frequent subgraph mining based on pregel. *The Computer Journal 59*, 8 (2016), 1113–1128.