Frequent Closed Subgraph Mining: A Multi-thread Approach

Lam B. Q. Nguyen[1], Ngoc-Thao Le[2], Hung Son Nguyen[3], Tri Pham[4], Bay Vo[2,*]

[1] Faculty of Information and Communications, Kien Giang University, Kien Giang, Vietnam
[2] Faculty of Information Technology, HUTECH University, Ho Chi Minh City, Vietnam
[3] Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Warsaw, Poland
[4] Institute for Computational Science and Technology (ICST) – Ho Chi Minh City, Vietnam

Abstract. Frequent subgraph mining (FSM) is an interesting research field and has attracted a lot of attention from many researchers in recent years, in which closed subgraph mining is a new topic with many practical applications. In the field of graph mining, GraMi (GRAph MIning) is considered state-of-the-art, and many algorithms have been developed based on the improvement of this approach. In 2021, we proposed the CloGraMi algorithm based on GraMi to mine closed frequent subgraphs from a large graph rapidly and efficiently. However, with NP time complexity and extremely high cost in terms of running time, graph mining is always a challenging problem for all researchers. In this paper, we propose a parallel processing strategy aiming to improve the execution speed of our CloGraMi algorithm. Our experiments on six datasets, including both undirected and directed graphs, with different sizes, including large, medium and small, show that the new algorithm significantly reduces running time and improves performance, and has better performance compared to the original algorithm.

Keywords: Subgraph Mining; Closed Subgraph Mining; Multi-thread; Parallel.

1 Introduction

Because graphs have a non-linear structure with NP complexity [1], [2], [3], graph mining has always been an interesting and challenging research area. Graphs are used to simulate, store, analysis and solve a wide variety of problems in both the scientific and commercial fields [4], [5], [6]. Graph mining [7], [8], [9], [10] is the premise for many different studies and practical applications, in areas such as social networks, maps analysis, telephone networks, bioinformatics, chemical compounds, crime investigation, etc. Closed frequent subgraph mining [11], [12], [13] is a relatively new field in this context, with a few studies already having been published [11], [12], [13], [14], [15], [16]. Extending this field is our motivation to study and research this topic.

For example, and as in [12], a commercial company needs to analyze data that has been collected from its customers, and the sales department needs to find the groups of loyal customers (who frequently buy the company's products) and the features of the relationships among the customer groups in order to have a flexible business strategy. In

Figure 1, graph G is used to simulate all the customers in the dataset, and each node represents a customer of this company (labeled A, B, C or D), in which each edge in the graph represents the relationship of each pair of customers (labeled x, y, z, t or w). The two subgraphs S_1, S_2 are two samples of groups of loyal customers.

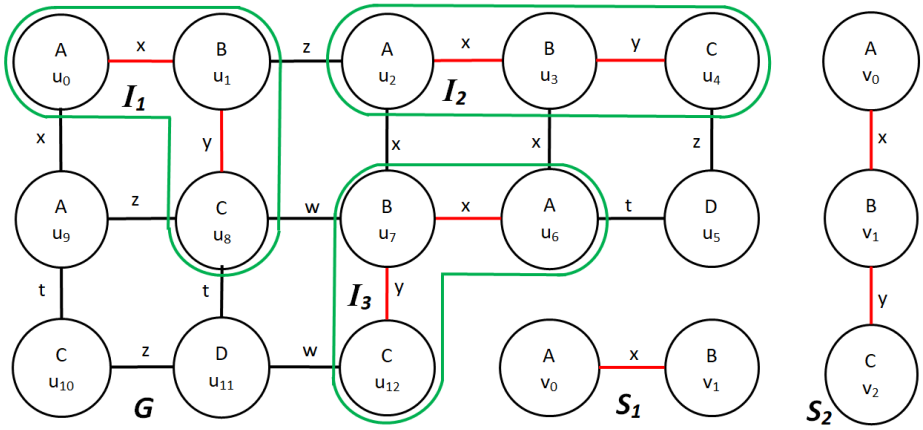

Fig. 1. G – a large graph – and its two subgraphs S_1 and S_2

The above example leads to very useful problems in practice, in which users need to find all the groups of loyal customers as well as the largest number of purchases for that group, as this can help the sales department to adjust the business strategies of the company. This problem is useful not only for business [3], [17], but also in many other fields [18], [19], [20], [21] such as recommendation systems, crime investigation, information retrieval systems, graph clustering, decision support systems, traffic flow, map model analysis, etc.

In 2014, the GraMi algorithm [1], [2], [22], [23] was proposed with the aim to effectively mine all frequent subgraphs in a large graph. And then, in 2020, we introduced the SuGraMi algorithm [2] based on GraMi, which can find all frequent subgraphs and their support. In 2021, we proposed an algorithm to mine closed frequent subgraphs, the CloGraMi algorithm [12], based on SuGraMi. In this paper, our contribution is proposing an effective strategy to improve our CloGraMi algorithm by parallel processing, it aims to reduce the running time, and we call it PCGraMi (Parallel Closed Graph Mining).

Our paper consists of six sections: Section 1 introduces the problem of closed graph mining and our contribution; Section 2 surveys relevant algorithms; Section 3 consists of the main definitions in the field of graph mining; Section 4 presents the PCGraMi algorithm in detail; Section 5 describes the results of our evaluation studies; Finally, the conclusions and future work are discussed in Section 6.

2 Related Work

The GraMi algorithm [1], [2], [3] was proposed in 2014 as a fast frequent subgraph mining algorithm in a single large graph. This algorithm solves the FSM problem by introducing a new technique to store only templates of generated candidates. It searches

isomorphisms and only marks the corresponding values in domains of those templates, without enumerating all the occurrences of each candidate subgraph (corresponding isomorphisms of this subgraph [1], [2], [24], [25]) from a large graph. Because of this new technique, GraMi is different from previous grow-and-store methods [24], [26]. GraMi has modelled the support of a subgraph as a CSP (constraint satisfaction problem) model [1], [2], [22]. The GraMi algorithm iteratively unfolds the CSP model until it discovers a minimum set of occurrences for a candidate subgraph to be correctly identified as a frequent subgraph. To reduce the search time, all remaining occurrences are ignored. This process will repeat by extending (adding new edges) a frequent subgraph until no new frequent subgraphs are found.

In 2016, a parallel FSM algorithm called ScaleMine [27] was proposed, and this applies distributed computing to the GraMi algorithm. ScaleMine is a two-phase approach [27], [28]: first, the algorithm performs an approximation to quickly identify the frequent subgraphs with high occurrence probability in the dataset; then, the exact calculations are performed using the results of the approximation phase to achieve better load balancing [29]. Experiments using this algorithm have extended to 8,192 cores of the Cray XC40 system, enabling it to solve a large graph with one billion edges and having faster mining performance than existing solutions. In 2020, we proposed a parallel approach PaGraMi [2] based on the original algorithm GraMi, using multi-thread processing in a computer with a multi-core CPU, combined with the SoGraMi algorithm [2] to reduce the search space of GraMi by pruning infrequent candidate subgraphs.

The process of generating candidate subgraphs needs lots of memory to store a huge number of candidates [1], [3], [30], [31]. Algorithms such as GraMi [1], SoGraMi [2], WeGraMi [25] and CCGraMi [23] have their own efficient strategies for pruning and thus reducing the number of infrequent candidates. The goals of all the mentioned algorithms are to reduce the costs of the generating phase [3], and improve the efficiency of memory usage and running time. Other parallel algorithms, such as ScaleMine [27] and Arabesque [32], have been proposed as new parallel algorithms using a message transfer interface on distributed systems, but they still lack a load balancing method for clusters or their machines in the system [3].

In 2018, the SSIGRAM algorithm (Spark-based Single Graph Mining) [33] was proposed as a parallel algorithm for FSM based on Spark. It parallelizes the extending and evaluating the support of subgraphs on all distributed "workers". In addition, a new heuristic-based search strategy was proposed with three optimizations: (1) load balancing, (2) top-down pruning and (3) pre-search pruning during support evaluation, helping to significantly improve performance. The support evaluation of candidates are simultaneously carried out at the “executors” in the Spark cluster. The evaluation results demonstrate that the SSIGRAM algorithm can perform significantly better than GraMi algorithm on all four different real datasets. Furthermore, it is capable of operating at lower support thresholds. While the SIGRAM [34] algorithm enumerates all intermediate steps using the maximum independent sets (MIS) measurements. However, these steps have NP complexity, thus the method is extremely expensive in practice [29].

In 2021, we introduced the CCGraMi algorithm [23], which is based on connected components in a large graph with two main contributions: finding isomorphisms of subgraphs in each connected component of the large graph instead of searching in the

whole large graph to reduce searching time, and early pruning of the domains of candidate subgraphs based on the size of the connected components to reduce storage space and searching time on those domains. We also proposed the CloGraMi algorithm in 2021 [12] to mine closed subgraphs, moreover we have three effective strategies aiming to optimize this algorithm: a level order traversal strategy to reduce the time for the search, early determining of closed subgraphs, and setting a constraint for pruning of candidates that are non-closed subgraphs.

In 2022, because the GraMi algorithm [1] lacked an effective strategy to balance its search space, we introduced the BaGraMi algorithm [22] to address this issue. This decreases the size of both the search space for all generated subgraphs and domain for each candidate subgraph [35], and thus helps balance the search space of the original GraMi algorithm [2]. The main contributions of this algorithm are to reduce the invalid assignments in the domain of a candidate and the number of infrequent subgraphs, enhancing the mining performance. Our algorithm not only reduces the running time but also the memory requirements of the mining process.

According to our survey [3] in 2022, most related studies focus on FSM [7], [8], [9], [10], [36], and there are few algorithms in the field of closed frequent subgraph mining [11], [14], [13], [15], [16], [37], [38], and thus we propose to improve our CloGraMi algorithm by using a parallel processing strategy to improve the execution speed of this algorithm on multi-core personal computers. The goal of this algorithm is to boost the performance of the mining closed frequent subgraphs from a single large graph process.

3 Definitions

Definition 1. [1], [2], [12]: Let G be a large *graph,* $G = (V, E, L)$. In which, V denotes the set of all the nodes, E denotes the set of all the edges and L is a function to assign labels to all the nodes/edges, respectively.

Definition 2. [1], [12]**,** [22]: A graph $S = (V_S, E_S, L_S)$ is a *subgraph* of $G = (V, E, L)$ iff $V_S \subseteq V$, $E_S \subseteq E$ and $L_S(v) = L(v)$ for $\forall v \in V_S$; $LS((u, v)) = L((u, v))$ for $\forall (u, v) \in ES$.

Definition 3. [1], [12]**,** [23]: Let $S = (V_S, E_S, L_S)$ be a subgraph of $G = (V, E, L)$. A *subgraph isomorphism* I 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 [12], the subgraph S_2 has three distinct isomorphisms I_1, I_2 and I_3 as presented in Table 1.

Table 1. Three distinct isomorphisms of S_2.

Subgraph S_2	v_0	v_1
Isomorphism I_1	u_0	u_1
Isomorphism I_2	u_2	u_3
Isomorphism I_3	u_6	u_7

The notation u is used to indicate all the nodes in the large graph G and the notation v is used to indicate all the nodes in the subgraph S. Each node $v \in V_S$ has a domain D containing all the nodes u that have the same node label with v and they can be assigned to v. The assignments u to v are used to list, mark and count the number of isomorphisms of subgraph S in the large graph G.

Example 2: As shown in Figure 2, the domain of $v_0 \in S$ is $D = \{u_0, u_2, u_6, u_9\}$.

Definition 4. [22]: An assignment of a node $u \in V_G$ in the domain of node $v \in V_S$ is *valid* if there exists an isomorphism I that assigns u to v, *invalid* otherwise.

Example 3: In Figure 2, the nodes u_0, u_2, and u_6 in the domain are valid assignments because there exist three isomorphisms I_1, I_2 and I_3 assigning u_0, u_2, and u_6 to v_0, respectively; u_9 is an invalid assignment in the domain of S_2 because there is no isomorphism that assigns u_9 to v_0.

Definition 5. [2], [22]: *The support* of S in G (*denoted by* $s_G(S)$) is the minimum number of all distinct valid assignments of $\forall v \in V_S$. In other words:

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

Example 4 [2], [12]: The support of subgraph S_2 in large graph G:

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

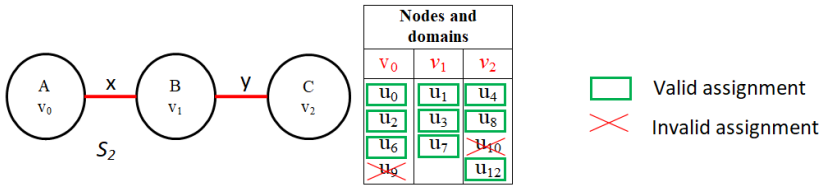

Fig. 2. Valid and invalid assignments for a subgraph

Definition 6. [3]: A subgraph S in a large graph G is called a *frequent subgraph* if $s_G(S) \geq \tau$, in which τ is a given frequency threshold.

Remark: We use $\tau = 2$ for all our examples in this paper.

Example 5 [12]: As $s_G(S_2) = 3 \geq \tau$, it is a frequent subgraph in G.

Definition 7. [3], [12]: A frequent subgraph S is a *closed frequent subgraph* if there does not exist any supergraph S' ($S \subset S'$) whose the support is equal to that of S (called as "closed subgraph" in short).

Example 6 [12]: S_1 is a non-closed subgraph because $S_1 \subset S_2$ and $s_G(S_2) = s_G(S_1) = 3$, and S_2 is a closed subgraph (see detail in [12]).

In the CloGraMi algorithm [12], we propose traversing the search tree by level to identify closed subgraphs early, in which in order to determine if the subgraph S at level n on the search tree is a closed subgraph the mining program only needs to compare S with frequent subgraphs at level (n + 1) in the search tree, instead of comparing this frequent subgraph with all mined frequent subgraphs. In this work, a new parallel processing strategy for our CloGraMi algorithm is proposed. Whereas, each branch in the search tree corresponding to an edge in the *FrequentEdge* list [2], [12] will be assigned to a concurrent thread, thus many branches in the search tree will be executed simultaneously, each frequent subgraph at level n on the branches will be mined and compared with the frequent subgraph at level (n + 1). In this PCGraMi algorithm, many frequent

subgraphs are generated and compared at the same time to find out the closed subgraphs to decrease the running time in the comparison to CloGraMi, our original algorithm.

4 Proposed Method

In the SuGraMi algorithm [2], as $s_G(S) \leq s_G(S')$ with S' be a child of S on the search tree was proven, based on this Downward Closure Property (DCP) [25] we introduced a level-wise traversal strategy [12] on the search tree, which can quickly identify closed subgraphs. Each frequent subgraph S of size k compares its support only with subgraphs S' with size $(k+1)$ being children of S [12]. We propose a parallel processing strategy to improve the speed of the CloGraMi algorithm in this paper, in which each branch of the search tree will be assigned to a thread, the threads will generate candidate subgraphs, check the support, and determine closed subgraphs on the same branch of the search tree.

Example 7: For the graph G in Figure 1, after evaluating all edges' support and pruning the infrequent edges, we have a *FrequentEdges* list consisting of edges as in Figure 3:

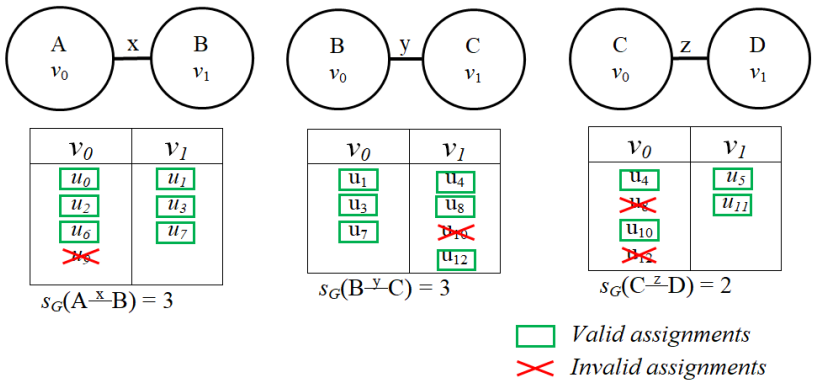

Fig. 3. The frequent edges list

$FrequentEdges = \{A\text{–}^{x}\text{–}B;\ B\text{–}^{y}\text{–}C;\ C\text{–}^{z}\text{–}D\}$

When the mining process generates the i^{th} thread to process the edge e_i in the $FrequentEdges$ list, the edges e_0, e_1,⋯, e_{i-1} have already been processed in other threads, so they will be removed from the $FrequentEdges$ list of this thread [2] as in Figure 4, and thus main program can implement without generating duplicate subgraphs.

Our algorithm consists of three steps.

(1) Each edge in the $FrequentEdges$ list is assigned to a separate thread (Line 4 to Line 8).

(2) These threads are then executed simultaneously (Line 6 and Line 7) based on the number of available processor cores. The running time of this parallel phase will be the time of the largest thread in the list (usually the first thread because it needs to combine with all remaining edges).

(3) Collecting closed subgraphs (Line 9 and Line 10) returned from these threads (collection time is very small compared to mining time).

```
Algorithm: PCGraMi
Input: A graph G, a frequency threshold τ
Output: All closed subgraphs S of G
1  ClosedSubgraphsList ← ∅ //the list of all closed subgraph in graph
G
2  Let FrequentEdges is a set of frequent edges in G
3  Let R be the result set for all simultaneously threads
4  foreach e ∈ FrequentEdges do
5    Generate a new thread and do
6      r ← ∅
7      r ← r ∪ extendSubgraph (e, G, τ, FrequentEdges)
8    Remove e from FrequentEdges
9  foreach r ∈ R do
10   ClosedSubgraphsList ← ClosedSubgraphsList ∪ r
11 return ClosedSubgraphsList
```

At Line 7 in our PCGraMi algorithm, the program needs a function *extendSubgraph*( ) to recursively extend frequent subgraphs, as follows.

```
Algorithm: extendSubgraph
Input: A frequent subgraph S, a frequency threshold τ, a set of frequent
edges FrequentEdges of G
Output: All closed subgraphs in G extended from S
1  cS ← ∅ //closed subgraphs list
2  gS ← ∅ //generated subgraphs list
3  fS ← ∅ //frequent subgraphs list
4  foreach e ∈ FrequentEdges and v ∈ S do
5    if e can extend u then
6      Let Ext be the extension of S by adding e
7      if Ext is not already generated then
8        gS ← gS ∪ Ext
9  clo ← true //a condition for being a closed subgraph
10   foreach g ∈ gS do
11     if s_G(g)≥ f then
12     fS ← fS ∪ g
13     if s_G(g)=s_G(S) then
14       clo ← false
15 if clo = true then
16   cS ← cS ∪ S
17 foreach S' ∈ fS do
18   cS ← cS ∪ extendSubgraph(e,f,FrequentEdges,G)
19 return cS
```

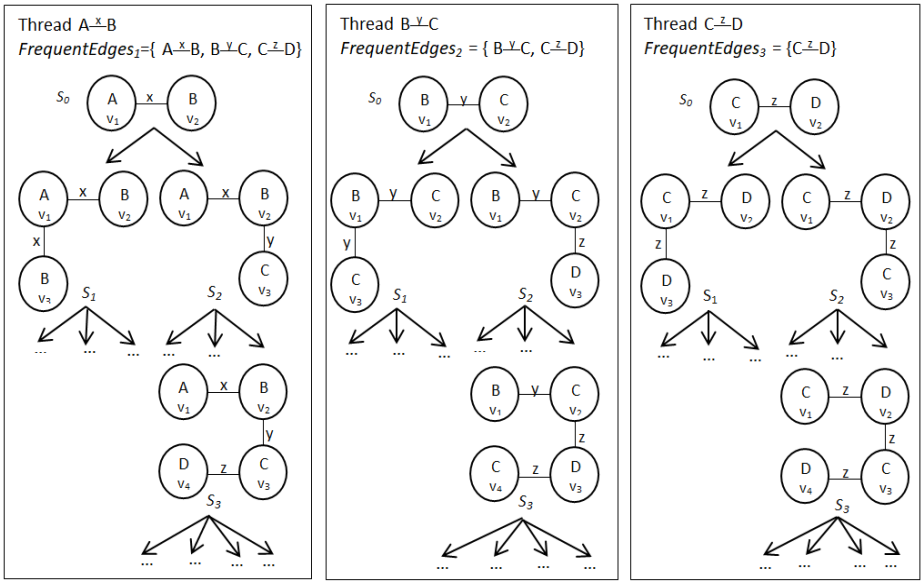

Fig. 4. Multi-threads execute simultaneously

Consider the example given in Figure 4, instead of sequentially processing each edge in this *FrequentEdges* list as the CloGraMi algorithm, our parallel algorithm will generate three threads to process these three edges at the same time. The main program then collects the results from these three threads to get the total results. The three phases are generating all three threads, parallel mining and collecting the results of PCGraMi algorithm. The first and third phases, which are generating threads and collecting the results, require a very short time compared to the mining phase, only from 1% to 2% of the total time for the program. As such, the cost of these two extra phases of our PCGraMi algorithm is insignificant compared to CloGraMi, but they significantly reduce the time of the mining phase. The limitation of parallel approaches is the memory requirements. Because the main program needs to store and process multiple candidate subgraphs at the same time, more storage space is needed for parallel mining than sequential processing.

5 Experimental Results

To demonstrate the effectiveness of our parallel strategy, we record and compare the running times for new algorithm PCGraMi with the original CloGraMi algorithm. All of our experiments were carried out on a multi-core personal computer running the Windows 10 operating system, equipped with a Core i5 CPU having four physical cores 3.2Ghz, eight threads, JavaSE Development Kit 8. A computer with four physical cores can handle up to four threads at the same time, so we implemented and compared the performance of the new algorithm PCGraMi with two threads and four threads and compared the results to those obtained with the sequential algorithm CloGraMi. With the two versions of PCGraMi (PCGraMi_2 and PCGraMi_4), we try to reduce the

frequency thresholds τ until they cannot execute, then we collect the results to show the efficiency of our method. Our experiments were performed on six different sized graph datasets, including directed and undirected graphs, as shown in Table 2.

Table 2. The features of six real datasets.

Datasets	Type	Nodes	Node labels	Edges	Edge labels
MiCo	Undirected	100,000	29	1,080,298	106
GitHub social network	Undirected	37,700	60	289,003	60
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
Email-Eu-core network	Directed	1,005	50	25,571	50

- With the MiCo dataset (Figure 5.a) [1], [2]: This is an undirected data set describing Microsoft co-authoring information with over one million edges and 100,000 nodes. The nodes in this dataset represent all Microsoft authors and their labels are the authors' interests. A collaboration between a pair of authors is represented as an edge, the edge's label represents the number of co-authored articles. This is a large dataset, so the mining thresholds are also high. There are only two edges in the $FrequentEdges$ list, so our parallel strategy generates only two processing threads corresponding to these two edges. The running time of the PCGraMi parallel algorithm with two threads is from 80% to 83% of that needed by the CloGraMi algorithm at the survey thresholds, and with four threads is roughly 71%. At the final threshold $\tau = 9{,}250$, CloGraMi needs 1,505.372 seconds for the mining process, while PCGraMi (with 4 threads) only needs 1,081.115 seconds.

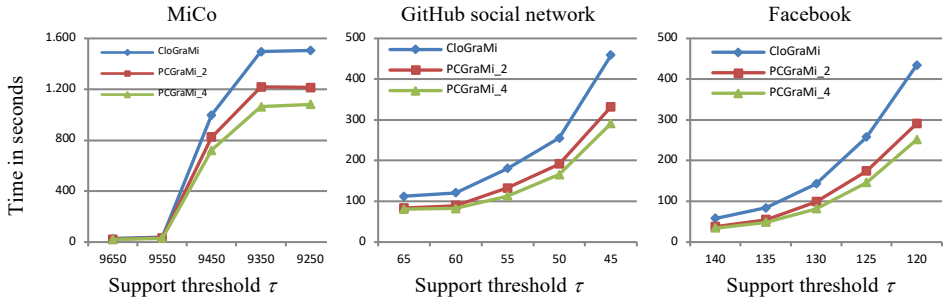

Fig. 5. The running time on the undirected datasets.

- With the GitHub social network dataset (Figure 5.b) [12]: This is an undirected medium size dataset collected in June 2019 using the public API. This dataset represents the network of GitHub developers, the nodes in this graph represent developers, while the edges describe the mutual follower relationships between a pair of developers. This network dataset can be obtained from https://snap.stanford.edu/data/. It consists of 37,700 nodes and 289,003 edges. PCGraMi with two threads can reduce the running

time to 72% of that needed by CloGraMi. The maximum running time (with four threads) can be decreased to 61% of that needed by the CloGraMi algorithm at $\tau = 55$, at the last threshold $\tau = 45$, CloGraMi needs 458.485 seconds, while PCGraMi only needs 300.211 seconds to complete the process.

- With the Facebook dataset (Figure 5.c) [2], [25]: This directed dataset was obtained at: http://snap.stanford.edu/data/, and consists of 4,389 nodes, 88,235 edges, which are the 'circles' (or 'friends lists') of the Facebook social network. The data is collected from the social network user surveyed through the Facebook app. User data has been anonymized by Facebook, replacing each user's internal ids with a new value. Our new PCGraMi algorithm can reduce the running time to 66% that needed by the original CloGraMi algorithm with two processing threads, and to 56% with four threads, at the last threshold $\tau = 120$, CloGraMi needs 434.768 seconds to process while PCGraMi with four threads takes only 251.663 seconds.

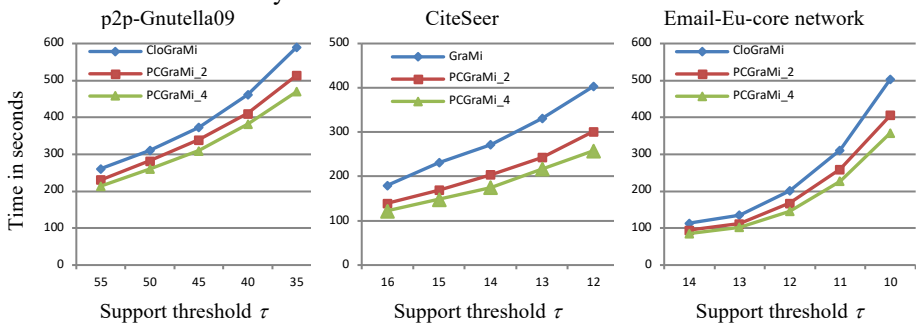

Fig. 6. The running time on three directed datasets.

- With p2p-Gnutella09 dataset (in Figure 6.a) [2], [22]: This graph dataset was obtained from http://snap.stanford.edu/data/, and is a directed graph dataset of a sequence of snapshots from the Gnutella peer-to-peer file sharing network. All nine snapshots were collected in August 2002 from the Gnutella network. Hosts in this network are represented as nodes in the dataset, while the connections among the Gnutella hosts are the edges of pairs of nodes. This dataset has 8,114 nodes with 26,013 directed edges. Our improved algorithm has a running time equal to approximately 87% (with two simultaneous threads) and 79% (with four processing threads) of that needed by CloGraMi. At the threshold $\tau = 35$, the original algorithm CloGraMi needs 590.122 seconds, while PCGraMi_4 needs 473.134 seconds to complete the mining process.

- With the CiteSeer dataset (Figure 6.b) [1], [23]: This directed graph dataset includes 3,312 nodes (each node in this graph corresponds to a publication) and 4,732 directed edges, they are citations between a pair of publications (citations are directed edges in this graph). In this graph, each node has a label representing a field of computer science. Each directed edge evaluates the similarity (labelling from 0 to 100) to between a pair of publications. Our proposed algorithm has the running time reduced by approximately 73% (with two threads) and 63% (with four threads) of CloGraMi's consumption, at the last threshold $\tau = 12$, the original algorithm CloGraMi needs 403.475 seconds but the PCGraMi parallel algorithm with four threads only needs 258.119 seconds.

- With the Email-Eu-core network dataset (Figure 6.c) [12]: This dataset can be acquired from https://snap.stanford.edu/data/. It is a medium size directed graph consisting of 1,005 nodes with 25,571 directed edges. This network in a large European research institution was generated by the users using email data, but the users' information was anonymized for the sake of privacy. Each directed edge means that a user in this institution sent at least one email to another user. The parallel algorithm's running time can be reduced to 80% (with two parallel threads) and 71% (with four processing threads) of CloGraMi's, at the last threshold $\tau = 10$, CloGraMi needs 504.182 seconds while PCGraMi with 4 threads only needs 358,198 seconds.

6 Conclusion and Future Work

We have surveyed new algorithms and the literature on frequent subgraph mining and closed subgraph mining in recent years. There are few studies on closed subgraph mining, and such algorithms are very costly to implement. Therefore, we improved our closed subgraph mining algorithm CloGraMi by proposing an efficient parallel strategy, which is to concurrently explore multiple subgraphs at the same time using multithreading. This parallel strategy can reduce the running time of the mining process. We conducted several experiments and compared PCGraMi with the original algorithm CloGraMi on six real datasets with different sizes (both directed and undirected), our new parallel algorithm has proved to overcome at all thresholds and on all selected datasets.
In the future, we have two promising research directions for closed subgraph mining: applying a parallel strategy to high-performance computing (HPC) to use the computing power to mine larger graphs with a smaller frequency threshold, and proposing an efficient pruning strategy for non-closed subgraphs that reduces the memory requirements of current algorithms.

Acknowledgement
This work was supported by Institute for Computational Science and Technology (ICST) – Ho Chi Minh City and the Department of Science and Technology (DOST) – Ho Chi Minh City under grant no. 23/2021/HĐ-QKHCN.

References

[1] M. Elseidy, E. Abdelhamid, S. Skiadopoulos, and P. Kalnis, "Grami: Frequent subgraph and pattern mining in a single large graph," *Proc. VLDB Endow.*, vol. 7, no. 7, pp. 517–528, 2014.
[2] 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, p. 103539, 2020.
[3] L. B. Q. Nguyen, I. Zelinka, V. Snasel, L. T. T. Nguyen, and B. Vo, "Subgraph mining in a large graph: A review," *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, p. e1454, 2022.
[4] S. Velampalli and V. R. M. Jonnalagedda, "Frequent SubGraph Mining Algorithms: Framework, Classification, Analysis, Comparisons," in *Data Engineering and Intelligent*

Computing, Springer, 2018, pp. 327–336.
[5] 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, p. 104302, 2021.
[6] 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, 2020.
[7] 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, 2018.
[8] M. H. Chehreghani, T. Abdessalem, A. Bifet, and M. Bouzbila, "Sampling informative patterns from large single networks," *Futur. Gener. Comput. Syst.*, vol. 106, pp. 653–658, 2020.
[9] 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, 2018.
[10] 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," *Futur. Gener. Comput. Syst.*, vol. 105, pp. 766–778, 2020.
[11] 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, 2017.
[12] L. B. Q. Nguyen, L. T. T. Nguyen, I. Zelinka, V. Snasel, H. S. Nguyen, and B. Vo, "A Method for Closed Frequent Subgraph Mining in a Single Large Graph," *IEEE Access*, 2021.
[13] N. E. I. Karabadji, S. Aridhi, and H. Seridi, "A closed frequent subgraph mining algorithm in unique edge label graphs," in *International Conference on Machine Learning and Data Mining in Pattern Recognition*, 2016, pp. 43–57.
[14] X. Yan and J. Han, "Closegraph: mining closed frequent graph patterns," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2003, pp. 286–295.
[15] A. Bendimerad, M. Plantevit, and C. Robardet, "Mining exceptional closed patterns in attributed graphs," *Knowl. Inf. Syst.*, vol. 56, no. 1, pp. 1–25, 2018.
[16] 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 *Iberoamerican Congress on Pattern Recognition*, 2017, pp. 10–18.
[17] Y. Jia, J. Zhang, and J. Huan, "An efficient graph-mining method for complicated and noisy data with real-world applications," *Knowl. Inf. Syst.*, vol. 28, no. 2, pp. 423–447, 2011.
[18] 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.
[19] 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.
[20] 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.
[21] 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, 2014.
[22] L. Nguyen *et al.*, "An efficient and scalable approach for mining subgraphs in a single large graph," *Appl. Intell.*, pp. 1–15, 2022.
[23] L. B. Q. Nguyen, I. Zelinka, and Q. B. Diep, "CCGraMi: An Effective Method for Mining Frequent Subgraphs in a Single Large Graph," in *MENDEL*, 2021, vol. 27, no. 2, pp. 90–99.
[24] J. R. Ullmann, "An algorithm for subgraph isomorphism," *J. ACM*, vol. 23, no. 1, pp. 31–42, 1976.

[25] N.-T. Le, B. Vo, L. B. Q. Nguyen, H. Fujita, and B. Le, "Mining weighted subgraphs in a single large graph," *Inf. Sci. (Ny).*, vol. 514, pp. 149–165, 2020.
[26] M. Seeland, T. Girschick, F. Buchwald, and S. Kramer, "Online structural graph clustering using frequent subgraph mining," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2010, pp. 213–228.
[27] E. Abdelhamid, I. Abdelaziz, P. Kalnis, Z. Khayyat, and F. Jamour, "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.
[28] X. Yan and J. Han, "gspan: Graph-based substructure pattern mining," in *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, 2002, pp. 721–724.
[29] L. Dhulipala, G. E. Blelloch, and J. Shun, "Theoretically efficient parallel graph algorithms can be fast and scalable," *ACM Trans. Parallel Comput.*, vol. 8, no. 1, pp. 1–70, 2021.
[30] L. T. Thomas, S. R. Valluri, and K. Karlapalem, "Margin: Maximal frequent subgraph mining," *ACM Trans. Knowl. Discov. from Data*, vol. 4, no. 3, pp. 1–42, 2010.
[31] A. Farag, H. Abdelkader, and R. Salem, "Parallel graph-based anomaly detection technique for sequential data," *J. King Saud Univ. Inf. Sci.*, vol. 34, no. 1, pp. 1446–1454, 2022.
[32] 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 *Proceedings of the 25th Symposium on Operating Systems Principles*, 2015, pp. 425–440.
[33] 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, 2018.
[34] M. Kuramochi and G. Karypis, "Finding frequent patterns in a large sparse graph," *Data Min. Knowl. Discov.*, vol. 11, no. 3, pp. 243–271, 2005.
[35] J. Kepner, "Keynote Talk: Large Scale Parallel Sparse Matrix Streaming Graph/Network Analysis," in *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures*, 2022, p. 61.
[36] S. Bouhenni, S. Yahiaoui, N. Nouali-Taboudjemat, and H. Kheddouci, "A survey on distributed graph pattern matching in massive graphs," *ACM Comput. Surv.*, vol. 54, no. 2, pp. 1–35, 2021.
[37] B. Güvenoglu and B. E. Bostanoglu, "A qualitative survey on frequent subgraph mining," *Open Comput. Sci.*, vol. 8, no. 1, pp. 194–209, 2018.
[38] P. Fournier-Viger *et al.*, "A survey of pattern mining in dynamic graphs," *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, vol. 10, no. 6, p. e1372, 2020.