ICIC Express Letters
Volume **10**, Number **9**, September **2016**

ICIC International ©2016 ISSN 1881-803X
pp. 2105–2113

PARALLEL FREQUENT SUBGRAPH MINING ON MULTI-CORE PROCESSOR SYSTEMS

Bao Huynh[1], Dang Nguyen[2] and Bay Vo[3]

[1]Center for Applied Information Technology
[2]Division of Data Science
Ton Duc Thang University
19 Nguyen Huu Tho Street, Tan Phong Ward, District 7, Ho Chi Minh City 70000, Vietnam
{ huynhquocbao; nguyenphamhaidang }@tdt.edu.vn

[3]Faculty of Information Technology
Ho Chi Minh City University of Technology
475A Dien Bien Phu Street, Ward 25, Binh Thanh District, Ho Chi Minh City 70000, Vietnam
bayvodinh@gmail.com

Received March 2016; accepted June 2016

Abstract. *Frequent subgraph mining is an important topic of graph mining, which has many applications such as web link analysis, fraud detection, and social networks. Numerous algorithms for mining frequent subgraphs have been proposed; most of them, however, used sequential strategies, which under-utilize the power of multi-core processor computers. In this paper, we propose a parallel algorithm to overcome this weakness. Firstly, the multi-core processor architecture is introduced. Secondly, we present the gSpan algorithm as the basic framework of our algorithm. Finally, we propose an efficient parallel algorithm for mining frequent subgraphs. The performance and scalability of the proposed algorithm are illustrated through extensive experiments on two datasets, namely chemical and compound.*

Keywords: Data mining, Frequent subgraph mining, Parallel computing, Multi-core processor

1. Introduction.

Graph mining is an important topic in data mining with many applications such as web link analysis [1], fraud detection [2] and social networks [3,4]. Frequent subgraph mining (FSM) is an essential part of graph mining. The goal of FSM is to discover all frequent subgraphs over a collection of graphs. A subgraph is called frequent if its occurrence is above a user-specified threshold. Numerous algorithms for mining frequent subgraphs have been proposed in recent years [5-14]. The AGM [14] is an algorithm which shares similar characteristics with the Apriori-based frequent itemset mining [15]. The Apriori property was also used in other algorithms for FSM such as FSG [11,16], AcGM [17] and AGM-Hash [18]. These algorithms inherit two weaknesses from Apriori: (1) joining two k-frequent subgraphs to generate $(k+1)$-subgraph candidates; (2) checking the frequency of these candidates separately. In order to avoid the overheads occurred in Apriori-based algorithms, several algorithms without candidate generation have been developed such as gSpan [7], FP-GraphMiner [13] and G-Tries [19]. These algorithms adopt the concept of pattern growth mentioned in [20], which expands patterns from a single pattern directly. While Apriori-based approach must use the breath-first search (BFS) strategy because of its level wise candidate generation, the pattern growth approach can use both BFS and depth-first search (DFS). Most existing FSM methods are sequential algorithms, causing that they require much effort and time to mine large graph datasets. Recently, researchers have begun using parallel and distributed computing techniques to accelerate the computation [21-27]. Along with the development of modern hardware, multi-core CPUs, GPUs, and Map/Reduce become potential and feasible tools for parallel computing [8,19,24,28,29]. Thus, some parallel algorithms have been developed for FSM recently. For example, Buehrer et al. [22] proposed a parallel algorithm for graph mining on the shared memory architecture. In 2015, Vo et al. [28] proposed a parallel algorithm for frequent subgraph mining on the multi-core processor. Kessl et al. [24] used CUDA to mine graph-based substructure patterns on GPUs. In addition, some studies have tried to parallelize FSM algorithms in the Map/Reduce paradigm [25-27,30]. It can be seen that applying parallelism to FSM is an emerging trend.

This study aims to propose an efficient parallel strategy for frequent subgraph mining on multi-core processor computers. Firstly, we introduce the multi-core processor architecture and its applications in data mining. Secondly, we present gSpan (graph-based Substructure pattern mining) which explores frequent substructure without candidate generation [7] as the basic framework of our proposed algorithms. Finally, we propose a parallel strategy for gSpan based on the parallelism model in Microsoft .NET Framework 4.0.

The rest of paper is organized as follows. Section 2 introduces the multi-core processor architecture and benefits of parallel computing in data mining. Section 3 reviews gSpan while Section 4 describes the proposed parallel algorithm. Section 5 presents experiments to show the performance of our algorithm. Conclusions and future work are discussed in Section 6.

2. Multi-core Processor Architecture.

A multi-core processor (Figure 1) is a single computing component with two or more independent central processing units (cores) in the same physical package [31]. Compared to a computer cluster (Figure 2) or a symmetric multi-processor system (Figure 3), the multi-core processor architecture has many desirable properties, for example, each core has direct and equal access to all the system's memory and the multi-core chip also allows higher performance at lower energy and cost. Parallel mining on multi-core processor computers has been widely adopted in many research fields, such as frequent itemset mining [29,32], class association rule mining [33], correlated pattern mining [34], tree-structured data mining [35], and generic pattern mining [36].

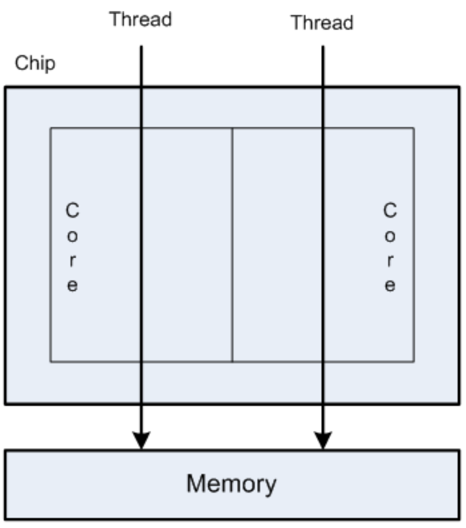

FIGURE 1. Multi-core processor: one chip, two cores, two threads[1]

[1]Source: http://software.intel.com/en-us/articles/multi-core-processor-architecture-explained

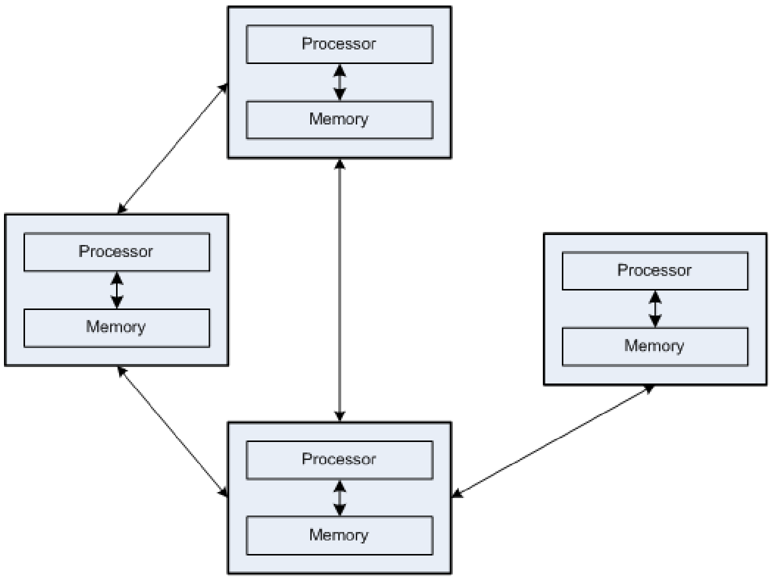

FIGURE 2. Computer cluster[2]

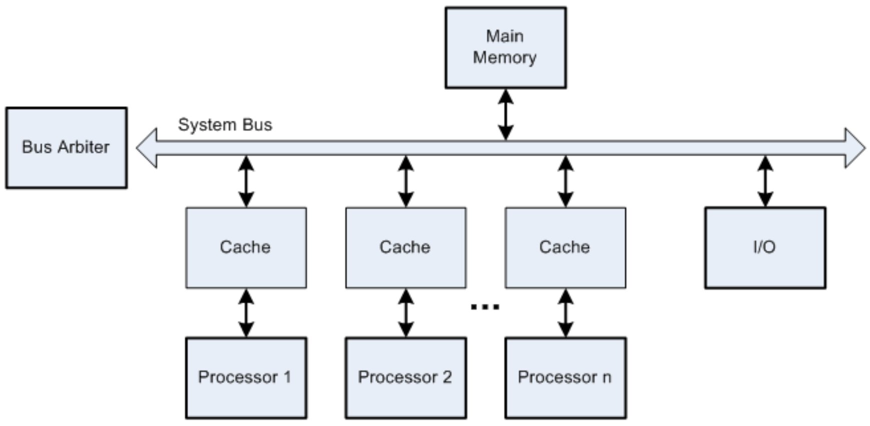

FIGURE 3. Symmetric multi-processor system[3]

3. gSpan Algorithm.

In this section, we briefly summarize the gSpan algorithm because it forms the basic framework of our proposed parallel algorithms.

gSpan uses the depth-first search (DFS) strategy to traverse its search tree. To generate a child node, gSpan extends the parent node by adding one new edge. Each node in the tree is assigned a unique DFS-code and this code is used to determine the isomorphism of subgraphs. The idea of gSpan is to label a subgraph by a DFS-code and create children DFS-codes from the right-most path of DFS tree. If a subgraph has a minimal DFS-code, it is added to the result and used to find the next subgraph. This process is recursively executed until the DFS-code of subgraph is non-minimal.

Definition 3.1. (DFS code) [7]: *A DFS tree T is built through the DFS of a graph G. The depth-first traverse of the vertices forms a linear order, which can be used to label these vertices. An edge sequence (e_i) is produced as follows. Assume $e_1 = (i_1, j_1)$ and $e_2 = (i_2, j_2)$:*

(1) if $i_1 = i_2$ and $j_1 < j_2$, then $e_1 < e_2$

(2) if $i_1 < i_2$ and $j_1 = j_2$, then $e_1 < e_2$

(3) if $e_1 < e_2$ and $e_2 < e_3$, then $e_1 < e_3$

The sequence e_i in which $i = 0, \ldots, |E| - 1$ is called a DFS-code, denoted as code(G, T).

[2]Source: http://en.wikipedia.org/wiki/Distributed_computing

[3]Source: http://en.wikipedia.org/wiki/Symmetric_multiprocessing

Definition 3.2. (DFS Lexicographic Order) [7]: *We suppose that* $Z = \{code(G,T)|$ *T is a DFS tree of $G\}$ and Z is a set which contains all DFS-codes for all graphs. DFS Lexicographic Order is a linear order of DFS-codes defined as follows.*

If $\alpha = code(G_\alpha, T_\alpha) = (a_0, a_1, \ldots, a_m)$ *and* $\beta = code(G_\beta, T_\beta) = (b_0, b_1, \ldots, b_m)$ *with* $\alpha, \beta \in Z$*, then* $\alpha \leq \beta$ *if one of two conditions is true:*

(1) $\exists t,\ 0 \leq t \leq \min(m,n),\ a_k = b_k$ *for* $k < t,\ a_t < b_t$*;*

(2) $a_k = b_k$ *for* $0 \leq k \leq m$ *and* $n \geq m$*.*

Definition 3.3. (Minimal DFS-code) [7]: *Given a graph G and $Z(G)$. Regarding DFS Lexicographic Order, the minimal one,* $\min(Z(G))$*, is called Minimal DFS-code of G.*

Theorem 3.1. [7]: *Given two graphs G and G', G is isomorphic to G' if and only if* $\min(G) = \min(G')$*.*

Definition 3.4. (Right-most path extension rules) [23]: *Given a DFS-code s and an edge e, in either of the following two cases, $s \cup e$ is called the right-most path extension:*

(1) e connects the right-most vertex and the vertices on the right-most path in the DFS tree

(2) e connects the right-most path in the DFS tree and a new vertex

Based on four definitions and Theorem 3.1, the sequential version of gSpan is represented in Figure 4.

```
Input: Graph dataset $D$ and minimum support threshold $minSup$
Output: All frequent subgraphs in $D$
Procedure: GraphSet_Projection($D$, $S$, $minSup$)
1.   sort labels of vertices and edges in $D$ by their frequency;
2.   remove infrequent vertices and edges;
3.   re-label the remaining vertices and edges in descending frequency;
4.   $S1$ = all frequent 1-edge graphs in $D$;
5.   sort $S1$ in DFS lexicographic order;
6.   $S = S \cup S1$;
7.   for each edge $e \in S1$ do
8.     initialize $s$ with $e$, set $s.D = \{g|\forall g \in D, e \in E(g)\}$; (only graph $id$ is stored)
9.     SubGraph_Mining($D$, $S$, $s$);
10.    $D = D \cup D \backslash e$;
11.    if $|D| < minSup$ then
12.      break;
SubGraph_Mining($D$, $S$, $s$)
13.  if $s \neq \min(s)$ then
14.    return;
15.  $S = S \cup \{s\}$;
16.  generate all $s$' potential children with one edge growth;
17.  enumerate($s$);
18.  for each $c$, $c$ is $s$' child do
19.    if support($c$) $\geq minSup$ then
20.      $s = c$;
21.      SubGraph_Mining($D$, $S$, $s$);
```

FIGURE 4. gSpan algorithm [7]

4. Proposed Parallel Algorithm.

In this section, we introduce our parallel versions for gSpan. We adopt key features of gSpan such as isomorphism test and children subgraph generation in our algorithms. A graph can be considered as an object so that we can find a graph of $k+1$-edge from a graph of k-edge.

4.1. Parallel mining frequent subgraphs with shared branch strategy.

Vo et al. [28] proposed a parallel algorithm for frequent subgraph mining based on multi-cores, named PMFS-IB. However, the proposed method has a weakness that it is not to shrink

```
Input: Graph dataset $D$ and minimum support threshold $minSup$
Output: All frequent subgraphs in $D$
Procedure: FIND-SubGraph($D$, $S$, $minSup$)
1.   sort labels of vertices and edges in $D$ by their frequency;
2.   remove infrequent vertices and edges;
3.   re-label the remaining vertices and edges in descending frequency;
4.   $S1$ = all frequent 1-edge graphs in $D$;
5.   sort $S1$ in DFS lexicographic order;
6.   $S = S \cup S1$;
7.   for each edge $e \in S1$ do
8.     initialize $s$ with $e$, set $s.D = \{g | \forall g \in D, e \in E(g)\}$; (only graph $id$ is stored)
9.     EXPAND-SubGraph($D$, $S$, $s$);
10.    $D = D \cup D \backslash e$;
11.    if $|D| < minSup$ then
12.      break;
EXPAND-SubGraph($D$, $S$, $s$)
13.  if $s \neq \min(s)$ then
14.    return;
15.  $S = S \cup \{s\}$;
16.  generate all $s$' potential children with one edge growth;
17.  enumerate($s$);
18.  for each $c$, $c$ is $s$' child do
19.    if support($c$) $\geq minSup$ then
20.      $s = c$;
21.      Task $t_i$ = new Task(() $\Rightarrow${
22.      $FS = \emptyset$;  // list of frequent subgraphs returned by this task
23.      MINE-SubGraph($D$, $FS$, $s$)});
24.      for each task in the list of created tasks do
25.      collect the set of frequent subgraph ($FS$) returned by each task;
26.      $S = S \cup FS$;
MINE-SubGraph($D$, $FS$, $s$)
27.  if $s \neq \min(s)$ then
28.    return;
29.  $FS = FS \cup \{s\}$;
30.  generate all $s$' potential children with one edge growth;
31.  enumerate($s$);
32.  for each $c$, $c$ is $s$' child do
33.    if support($c$) $\geq minSup$ then
34.      $s = c$;
35.      MINE-SubGraph($D$, $FS$, $s$);
```

Figure 5. PMFS-SB algorithm

the dataset. To overcome the weakness of PMFS-IB, we propose a new parallel strategy for gSpan (called PMFS-SB) as shown in Figure 5.

PMFS-SB assigns each branch of the DFS tree to multiple tasks. Firstly, PMFS-SB finds all frequent 1-edge subgraphs (lines 1-6). For a given 1-edge subgraph, the algorithm calls procedure EXPAND-SubGraph to find all frequent subgraphs grown from this 1-edge subgraph (lines 7-9). The step at line 10 projects the dataset into a smaller graph set with fewer vertices, edges, and graphs. In procedure EXPAND-SubGraph, all potential children subgraphs of a single 1-edge subgraph s are generated (lines 16 and 17). The algorithm then calls procedure MINE-SubGraph to mine all children subgraphs in parallel (lines 21-23). The function of MINE-SubGraph is the same as EXPAND-SubGraph except that the former is called recursively whereas the latter is called only one time. Finally, after all tasks finish their work, their results are collected to form the complete set of frequent subgraphs generated by this branch (lines 24-26).

4.2. **Example.** We apply PMFS-SB to a sample dataset shown in Figure 6 with *minSup* = 100% to compare their different process. The DFS tree constructed by PMFS-SB is shown in Figure 7. It can be seen that PMFS-SB creates four tasks $t1$, $t2$, $t3$, and $t4$ to parallel process the same branch "a-b" (Figure 7).

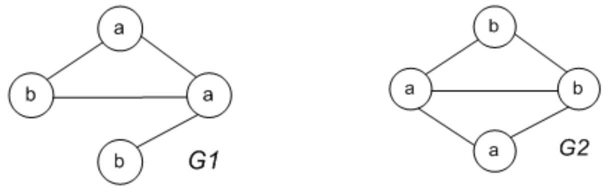

FIGURE 6. Example of a graph dataset

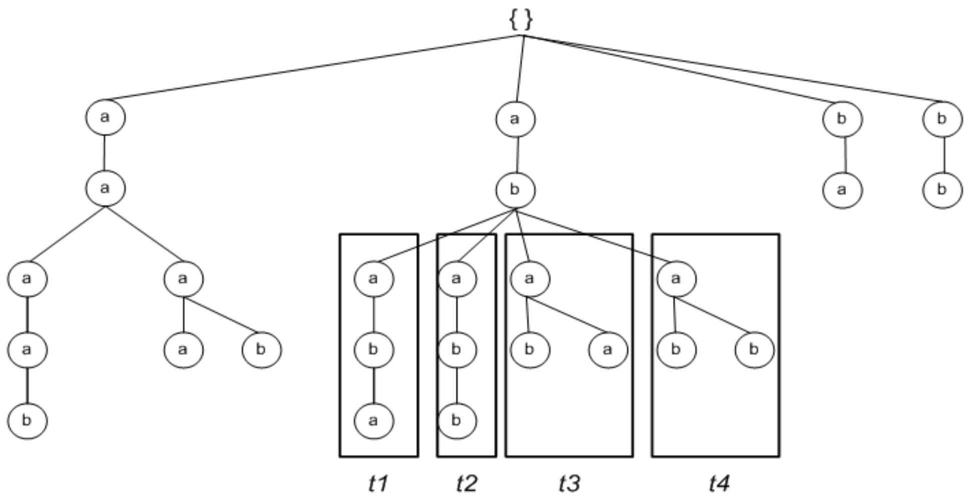

FIGURE 7. DFS tree constructed by PMFS-SB for the sample dataset in Figure 6

5. **Experiments.** All experiments were conducted on a computer with an Intel Core i7-2600 CPU at 3.4 GHz and 4 GB of RAM, which runs Windows 7 Enterprise (64-bit) SP1. The processor has 4 cores and an 8 MB L3-cache; it also supports Hyper-threading. The experimental datasets were obtained from http://www.cs.ucsb.edu/∼xyan/software/. The algorithms were coded in C# by using Visual Studio .NET 2013. Characteristics of the experimental datasets are described in Table 1. The table represents the number of graphs, the average graph size, and the largest graph size in the dataset.

TABLE 1. Characteristics of the experimental datasets

Dataset	#graphs	Average size		Largest size	
		#nodes	#edges	#nodes	#edges
Chemical	340	27	28	214	214
Compound	422	40	42	189	196

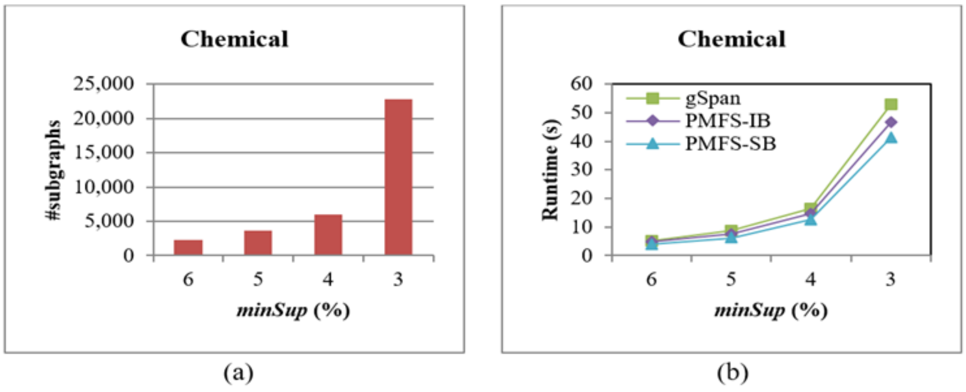

FIGURE 8. #subgraph and runtimes of gSpan, PMFS-IB, and PMFS-SB for the Chemical dataset

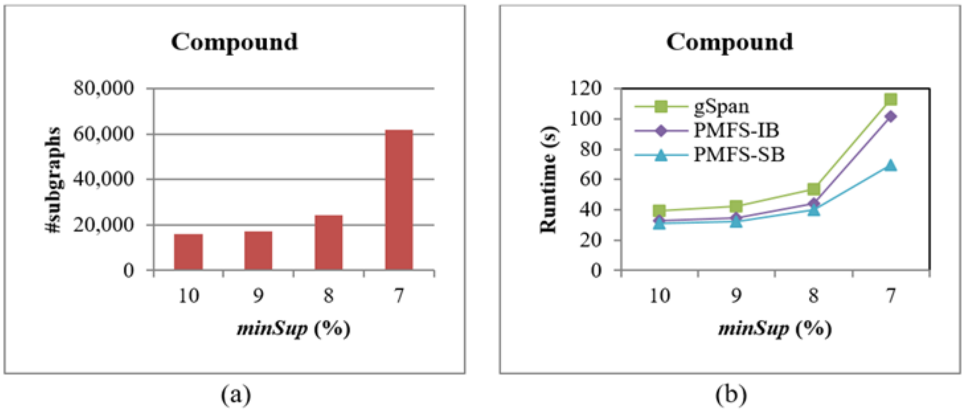

FIGURE 9. #subgraph and runtimes of gSpan, PMFS-IB, and PMFS-SB for the Compound dataset

To demonstrate the efficiencies of PMFS-SB, we compared its execution time with those of sequential gSpan [7] and PMFS-IB [28]. Figures 8(a) and 9(a) provide information about the number of frequent subgraphs which is found in Chemical and Compound respectively while Figures 8(b) and 9(b) compare the runtimes of three algorithms.

The results show that PMFS-SB outperforms gSpan and PMFS-IB in all experiments in terms of mining time. This is because the PMFS-SB can utilize the power of the multi-core processor architecture. When the values of *minSup* are low, the mining times of gSpan and PMFS-IB dramatically rise while the figures for PMFS-SB are smaller. For example, consider the Compound dataset with *minSup* = 7%. The runtime of gSpan was 112.927(s) while that of PMFS-IB was 101.430(s) and PMFS-SB was only 69.942(s). Similarly, consider the Chemical dataset with *minSup* = 3%. The execution times of gSpan, PMFS-IB, and PMFS-SB were 53.037(s), 46.657(s), and 41.237(s) respectively.

6. Conclusions and Future Work.

This paper has introduced an efficient parallel strategy for mining frequent subgraphs in graph datasets. The primary idea of the proposed algorithm is to adapt gSpan to parallel versions based on the parallelism model in

.NET Framework 4.0. The parallel feature is implemented on the multi-core processor architecture which does not require an extra cost for synchronization among processors like a computer cluster.

To validate the efficiencies of the proposed algorithms, experiments were conducted on two popular datasets, namely Chemical and Compound. The experimental results show that the proposed method is superior to the sequential algorithm gSpan and the parallel algorithm PMFS-IB. However, when minimum support values are very low, the cost for context switching between tasks is still very high. The memory consumption is also high, which may cause the memory leakage. We will study the solutions to reduce the memory consumption and to speed up the mining time. We also expand our work to closed subgraph and maximal subgraph mining in the future.

REFERENCES

[1] J. Punin, M. Krishnamoorthy and M. Zaki, LOGML: Log markup language for web usage mining, in *Mining Web Log Data Across All Customers Touch Points*, Springer, pp.88-112, 2002.

[2] W. Eberle and L. Holder, Mining for insider threats in business transactions and processes, *The IEEE Symposium on Computational Intelligence and Data Mining*, 2009.

[3] D. Nettleton, Data mining of social networks represented as graphs, *Computer Science Review*, vol.7, pp.1-34, 2013.

[4] Y. Sylla and P. Morizet-Mahoudeaux, Fraud detection on large scale social networks, *The IEEE International Congress on Big Data*, 2013.

[5] L. Dehaspe, H. Toivonen and R. King, Finding frequent substructures in chemical compounds, *KDD*, 1998.

[6] C. Borgelt and M. Berthold, Mining molecular fragments: Finding relevant substructures of molecules, *The IEEE International Conference on Data Mining*, 2002.

[7] X. Yan and J. Han, gSpan: Graph-based substructure pattern mining, *The IEEE International Conference on Data Mining*, 2002.

[8] J. Huan, W. Wang and J. Prins, Efficient mining of frequent subgraphs in the presence of isomorphism, *The IEEE International Conference on Data Mining*, 2003.

[9] S. Nijssen and J. Kok, Frequent graph mining and its application to molecular databases, *The IEEE International Conference on Systems, Man and Cybernetics*, 2004.

[10] K. Jahn and S. Kramer, Optimizing gSpan for molecular datasets, *The 3rd International Workshop on Mining Graphs, Trees and Sequences*, 2005.

[11] M. Kuramochi and G. Karypis, Frequent subgraph discovery, *The IEEE International Conference on Data Mining*, 2001.

[12] A. Alonso et al., Mining frequent connected subgraphs reducing the number of candidates, in *Machine Learning and Knowledge Discovery in Databases*, Springer, pp.365-376, 2008.

[13] R. Vijayalakshmi et al., FP-GraphMiner – A fast frequent pattern mining algorithm for network graphs, *Journal of Graph Algorithms and Applications*, vol.15, no.6, pp.753-776, 2011.

[14] A. Inokuchi, T. Washio and H. Motoda, An apriori-based algorithm for mining frequent substructures from graph data, in *Principles of Data Mining and Knowledge Discovery*, Springer, pp.13-23, 2000.

[15] R. Agrawal and R. Srikant, Fast algorithms for mining association rules in large databases, *The 20th International Conference on Very Large Data Bases*, pp.487-499, 1994.

[16] M. Kuramochi and G. Karypis, An efficient algorithm for discovering frequent subgraphs, *IEEE Trans. Knowledge and Data Engineering*, vol.16, no.9, pp.1038-1051, 2004.

[17] A. Inokuchi et al., *A Fast Algorithm for Mining Frequent Connected Subgraphs*, Technical Report, Tokyo Research Laboratory, IBM Research, 2002.

[18] P. C. Nguyen et al., Using a hash-based method for apriori-based graph mining, in *Knowledge Discovery in Databases: PKDD*, Springer, pp.349-361, 2004.

[19] P. Ribeiro and F. Silva, G-Tries: A data structure for storing and finding subgraphs, *Data Mining and Knowledge Discovery*, vol.28, no.2, pp.337-377, 2014.

[20] J. Han, J. Pei and Y. Yin, Mining frequent patterns without candidate generation, *ACM SIGMOD Record*, ACM, 2000.

[21] D. Cook et al., Approaches to parallel graph-based knowledge discovery, *Journal of Parallel and Distributed Computing*, vol.61, no.3, pp.427-446, 2001.

[22] G. Buehrer et al., *Parallel Graph Mining on Shared Memory Architectures*, Technical Report, Columbus, OH, USA, 2005.

[23] F. Wang, J. Dong and B. Yuan, Graph-based substructure pattern mining using CUDA dynamic parallelism, in *Intelligent Data Engineering and Automated Learning*, Springer, pp.342-349, 2013.

[24] R. Kessl et al., Parallel graph mining with GPUs, *The 3rd International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, 2014.

[25] B. Wu and Y. Bai, An efficient distributed subgraph mining algorithm in extreme large graphs, in *Artificial Intelligence and Computational Intelligence*, Springer, pp.107-115, 2010.

[26] W. Lu et al., Efficiently extracting frequent subgraphs using MapReduce, *The IEEE International Conference on Big Data*, 2013.

[27] W. Lin, X. Xiao and G. Ghinita, Large-scale frequent subgraph mining in MapReduce, *The 30th IEEE International Conference on Data Engineering*, 2014.

[28] B. Vo, D. Nguyen and T.-L. Nguyen, A parallel algorithm for frequent subgraph mining, in *Advanced Computational Methods for Knowledge Engineering*, Springer, pp.163-173, 2015.

[29] B. Schlegel et al., Scalable frequent itemset mining on many-core processors, *The 9th International Workshop on Data Management on New Hardware*, 2013.

[30] S. Hill, B. Srichandan and R. Sunderraman, An iterative MapReduce approach to frequent subgraph mining in biological datasets, *The ACM Conference on Bioinformatics, Computational Biology and Biomedicine*, 2012.

[31] B. Andrew, *Multi-Core Processor Architecture Explained*, http://software.intel.com/en-us/articles/multi-core-processor-architecture-explained, 2008.

[32] L. Liu et al., Optimization of frequent itemset mining on multiple-core processor, *The 33rd International Conference on Very Large Data Bases*, 2007.

[33] D. Nguyen, B. Vo and B. Le, Efficient strategies for parallel mining class association rules, *Expert Systems with Applications*, vol.41, no.10, pp.4716-4729, 2014.

[34] A. Casali and C. Ernst, Extracting correlated patterns on multicore architectures, in *Availability, Reliability, and Security in Information Systems and HCI*, Springer, pp.118-133, 2013.

[35] S. Tatikonda and S. Parthasarathy, Mining tree-structured data on multicore systems, *Proc. of the VLDB Endowment*, vol.2, no.1, pp.694-705, 2009.

[36] B. Negrevergne et al., Para miner: A generic pattern mining algorithm for multi-core architectures, in *Data Mining and Knowledge Discovery*, Springer, pp.593-633, 2013.