Representative Frequent Approximate Subgraph Mining on Multi-Graph Collections

by

MSc. Niusvel Acosta-Mendoza

Thesis submitted as a partial requirement for the degree of

Ph.D. in Computational Sciences

at the

Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE)
2018, Tonantzintla, Puebla, Mexico

Advisors:

PhD. Jesús Ariel Carrasco-Ochoa
Computational Science Coordination
INAOE
Ph.D. Andrés Gago-Alonso
Data Mining Department
Centro de Aplicaciones de Tecnologías de Avanzada (CENATAV)

©INAOE 2018
All right reserved
The author gives to the INAOE the permission for reproducing and distributing this document

Abstract

Nowadays, there has been an increase in the use of frequent approximate subgraph (FAS) mining for different real-world applications such as image classification, social network analysis and natural language processing, among others. In several of these applications, in the last years, multi-graphs have been used to model data, because, in the real-world, commonly there are more than one relation (edge) between the entities represented as vertices. However, the reported FAS miners have been designed to work with simple-graphs. Therefore, in order to solve the problem of mining FASs from multi-graph collections, we explore two alternatives in this research: (1) transforming the multi-graphs into simple-graphs and the FASs are obtained by applying conventional FAS miners over the transformed simple-graph collection, and (2) proposing algorithms for mining FAS directly from multi-graph collections. Following the first alternative, a method, called allEdges, based on graph transformations for mining all FASs on multi-graph collections by means of applying simple-graph FAS miners was proposed. Later, for speeding up the mining process, an alternative method, called onlyMulti, based on graph transformation for mining some FASs over multi-graph collections was proposed. Despite the fact that both allEdges and onlyMulti allow using simple-graph FAS miners for mining multi-graph FASs, the graph transformation processes increase the size of the graph collection and therefore, the mining process cost is increased. Thus, an algorithm, called MgVEAM, for mining all FASs directly over multi-graph collections without graph transformations was proposed. After, in order to accelerate the mining process, another algorithm, called AMgMiner, for directly mining all multi-graph FASs was proposed. AMgMiner is faster than MgVEAM, but the former requires more memory than the later. All the proposed methods and algorithms were evaluated and compared by using different

multi-graph datasets.

The large number of mined FASs is one of the fundamental drawbacks of FAS mining, which makes difficult the further use of the mined FASs. Therefore, in order to mine only a subset of representative FASs from multi-graph collections, we proposed two algorithms; one for mining generalized closed FASs and another for mining clique FASs.

Experiments on different databases were carried out to show the performance of our proposals. We also analyze the computational complexity of our transformation methods and mining algorithms. In order to show how to use the patterns computed by our algorithms, we include some experiments on using FASs for image classification, where the images are represented as multi-graphs.

Based on our experiments, we conclude that it is possible to mine multi-graph FASs from multi-graph collections with simple-graph FAS miners by applying our methods based on graph transformations. We also conclude that it is possible to mine all multi-graph FASs directly from multi-graph collections by means of AMgMiner and MgVEAM. Finally, with our representative FAS miners, it is possible to mine maximal, closed and clique FASs directly from multi-graph collections without increasing the computational cost of the mining process.

Resumen

Actualmente existe un incremento del uso de la minería de subgrafos frecuentes aproximados en diferentes aplicaciones, por ejemplo clasificación de imágenes, análisis de redes sociales y procesamiento del lenguaje natural, entre otros. En varias de estas aplicaciones, en los últimos años, los multi-grafos han sido utilizados para modelar los datos, porque en la realidad, comúnmente existen más de una relación (arista) entre las entidades representadas como vértices. Sin embargo, los algoritmos reportados para la minería de este tipo de patrones han sido diseñados para trabajar con grafos simples. Por tanto, con el objetivo de solucionar el problema de minar subgrafos frecuentes aproximados en colecciones de multi-grafos, en esta tesis se exploran dos alternativas: (1) transformar los multi-grafos en grafos simples, obteniendo los subgrafos frecuentes aproximados al aplicar algoritmos convencionales sobre los grafos simples transformados, y (2) proponer algoritmos para la minería de subgrafos frecuentes aproximados directamente sobre colecciones de multi-grafos. Siguiendo la primera alternativa se propone un método, llamado allEdges, que se basa en transformaciones de grafos para la minería de todos los subgrafos frecuentes aproximados en colecciones de multi-grafos mediante la aplicación de algoritmos que minan grafos simples. Luego, para acelerar el proceso de minería se propuso un método alternativo, llamado onlyMulti, el cual está basado en transformaciones de grafos para minar algunos subgrafos frecuentes aproximados en colecciones de multi-grafos. A pesar del hecho de que los métodos allEdges y onlyMulti permiten usar algoritmos que mina grafos simples para minar subgrafos frecuentes aproximados en el contexto de multi-grafos, los procesos de transformación incrementan el tamaño de la colección de grafos y por consiguiente se incrementa el costo del proceso de minería. Por lo que se propone un algoritmo, llamado MgVEAM, para minar todos los subgrafos frecuentes

aproximados directamente de la colección de multi-grafos sin procesos de transformación de grafos. Después, con el objetivo de acelerar el proceso de minería, se propone otro algoritmo, llamado AMgMiner, para minar todos los subgrafos frecuentes aproximados directamente en colecciones de multi-grafos. AMgMiner es más rápido que MgVEAM, pero el primero requiere más memoria que el segundo. Todos los métodos y algoritmos propuestos fueron evaluados y comparados utilizando diferentes colecciones de multi-grafos.

Por otro lado, el elevado número de subgrafos frecuentes que se encuentran es uno de los inconvenientes de la minería de subgrafos frecuentes aproximados, el cual dificulta el uso dichos subgrafos. Por lo tanto, con el objetivo de minar solo un subconjunto de patrones representativos en colecciones de mutli-grafos, se propusieron dos algoritmos; uno para identificar patrones cerrados generalizados, y otro para encontrar patrones cliques.

Se realizaron experimentos sobre diferentes bases de datos para mostrar el comportamiento de los métodos basados en transformaciones de grafos y de los algoritmos para la minería propuestos. Además, se realiza un análisis de la complejidad computacional de las propuestas. Con el objetivo de mostrar cómo se usan los patrones encontrados por nuestros algoritmos, se incluyen algunos experimentos usando los subgrafos frecuentes aproximados para la clasificación de imágenes, donde las imágenes están representadas como multi-grafos.

Basados en nuestros experimentos se puede concluir que es posible minar subgrafos frecuentes aproximados de colecciones de multi-grafos con algoritmos convencionales para la minería de patrones frecuentes en colecciones de grafos simples aplicando nuestros métodos basados en transformaciones de grafos. Además, se puede concluir que es posible minar todos los subgrafos frecuentes aproximados directamente de dichas colecciones mediante AMgMiner y MgVEAM. Finalmente, con nuestros algoritmos para la minería de patrones representativos es posible minar los subgrafos frecuentes aproximados maximales, cerrados y cliques directamente de las colecciones de multi-grafos sin incrementar el costo computacional del proceso de minería.

Contents

Abstract iii

Resumen v

1 Introduction 1

1.1 Motivation . . . 6

1.2 Aims . . . 6

1.3 Overview and Results . . . 7

1.4 Document Description . . . 9

2 Basic Concepts 10

2.1 Labeled Simple-Graph and Multi-Graph . . . 10

2.2 Graph Similarity . . . 12

2.3 Approximate Subgraph Mining . . . 18

2.4 Summary . . . 19

3 Related Work 20

3.1 Frequent Subgraph Mining . . . 20

3.2 VEAM . . . 24

3.3 Summary . . . 28

4 Multi-Graph Pattern Mining Based on Graph Transformations 30

4.1 Mining All FASs from a Multi-graph Collection . . . 31

4.2 Mining a Subset of FASs from a Multi-graph Collection . . . 36

4.3 Experiments and Results 39
4.4 Summary and Conclusions 45

5 Mining Patterns Directly from Multi-Graph Collections **46**
5.1 Algorithm based on Canonical Adjacency Matrices 47
5.1.1 Canonical Adjacency Matrix for Multi-Graph Mining 47
5.1.2 The MgVEAM Algorithm 52
5.2 Algorithm based on Depth-First Search canonical forms 56
5.2.1 Depth-First Search Canonical Form for Multi-Graph Mining 57
5.2.2 The AMgMiner Algorithm 61
5.3 Experiments and Results 68
5.4 Summary and Conclusions 77

6 Mining Representative Patterns **78**
6.1 Maximal and Closed FASs 79
6.1.1 The GenCloMgVEAM Algorithm 82
6.2 Clique FASs 85
6.2.1 The CliqueAMgMiner algorithm 86
6.3 Experiments and Results 88
6.4 Summary and Conclusions 95

7 Conclusions and Future Work **96**
7.1 Conclusions 98
7.2 Contributions 100
7.3 Publications 101
7.4 Future Work 102

Appendix A (Using Multi-Graph FASs) **115**

Appendix B (Graph Transformation Correctness) **122**

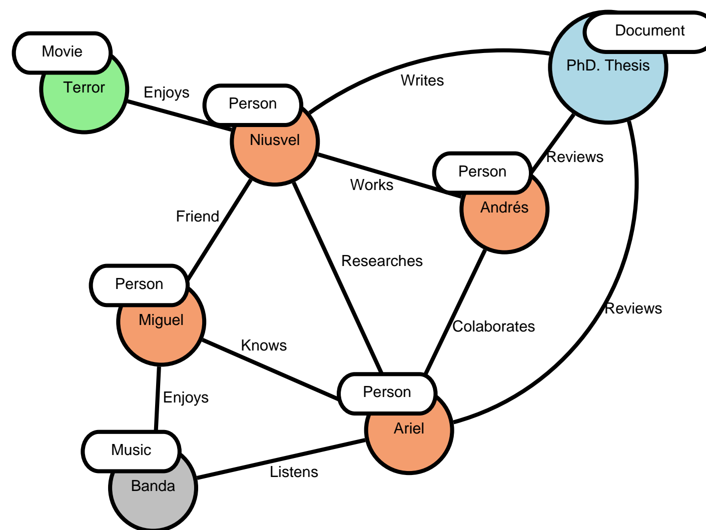
Chapter 1

Introduction

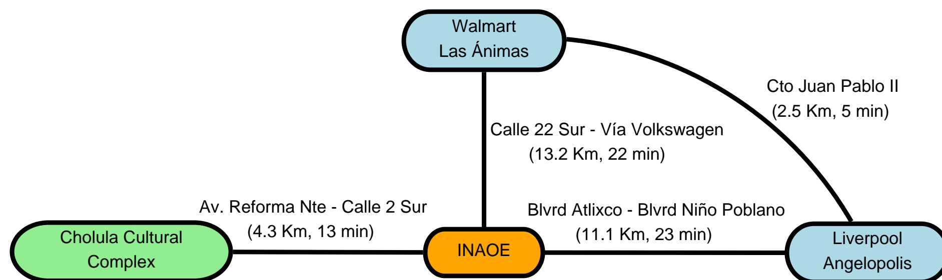
In this chapter, for properly contextualizing our research problem, we present an introduction to the area of mining approximate frequent subgraphs; mentioning some of the different applications where they are applied. Later, the importance and motivation of our research is discussed, and finally, the aims of this Ph.D. research are presented.

In data mining, frequent pattern mining has become an important topic with a wide range of applications in several domains of science, such as: biology, chemistry, social sciences and linguistics, among others (Emmert-Streib et al., 2016; Muñoz-Briseño et al., 2016; Wang et al., 2016; Appel and Moyano, 2017; Deore et al., 2017; Petermann et al., 2017; Senthilkumaran and Thangadurai, 2017; Herrera-Semenets and Gago-Alonso, 2017). This topic includes different techniques for frequent pattern mining, where frequent subgraph mining techniques should be highlighted. These techniques search for subgraphs which appear frequently in a graph database. Graphs are commonly used to model data, since in real-world applications there are entities or objects which can be naturally represented as vertices, and their relationships can be represented as edges (Riesen and Bunke, 2008; Aoun et al., 2014; Manzo et al., 2015; Rousseau et al., 2015; Acosta-Mendoza et al., 2016b; Shi and Weninger, 2016; Appel and Moyano, 2017). Figure 1.1 shows three examples of data modeled using graphs.

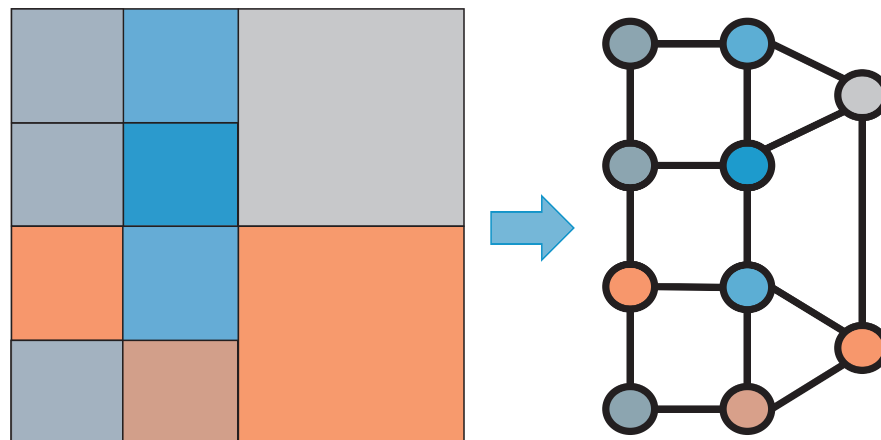
Several algorithms have been developed for mining all frequent subgraphs in a graph collection (Borgelt, 2002; Inokuchi et al., 2002; Kuramochi and Karypis, 2002; Yan and Han, 2002; Huan et al., 2003; Nijssen and Kok, 2004; Wang et al., 2004; Zhu et al., 2007; Gago-Alonso et al., 2008; Thomas et al., 2009; Gago-Alonso et al., 2010a,b; Gago-Alonso, 2015; Alam et al., 2017; Petermann et al., 2017). These algorithms use exact matching for computing frequent subgraphs, but there are several real-world problems where some variations in the



(a) A social network.



(b) A transportation network.



(c) An image represented by the relations between its color regions.

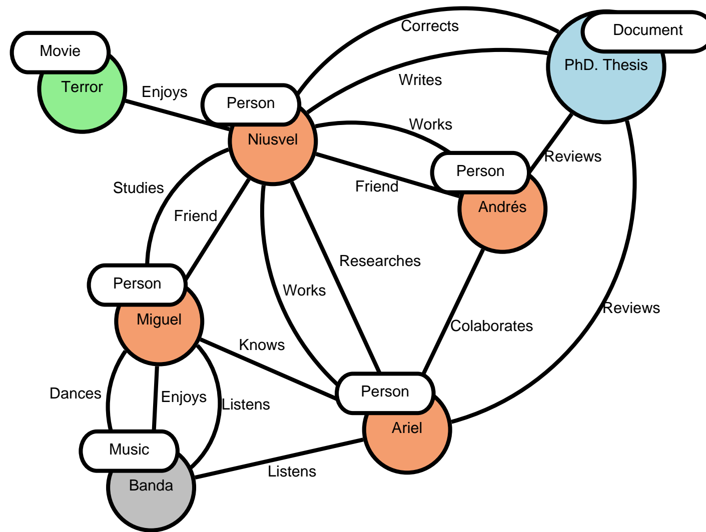
Figure 1.1: Examples of data modeled as graphs.

data are allowed, for example: analysis of links, social networks and routers of package delivery, image classification, and intrusion detection, among others (Holder et al., 1992;

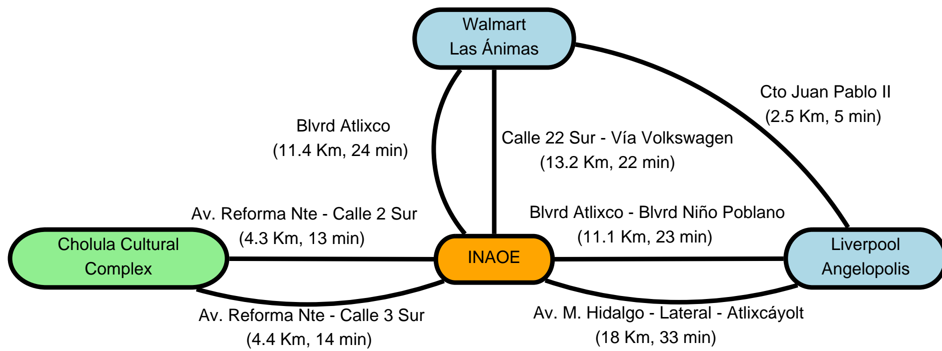
Flores-Garrido et al., 2014; Ramraj and Prabhakar, 2015; Santhi and Padmaja, 2015; Emmert-Streib et al., 2016; Muñoz-Briseño et al., 2016; Herrera-Semenets and Gago-Alonso, 2017). In these problems, where approximations between similar graphs must be considered, exact matching does not produce a positive outcome (Holder et al., 1992; Chen et al., 2008; Acosta-Mendoza et al., 2012a; Li et al., 2012; Elseidy et al., 2014; Flores-Garrido et al., 2015; Gao et al., 2015; Li and Wang, 2015; Moussaoui et al., 2016; Muñoz-Briseño et al., 2016). For this reason, several algorithms have been developed for frequent approximate subgraph (FAS) mining. These algorithms use different approximate graph matching for mining FASs (Jia et al., 2011; Acosta-Mendoza et al., 2012a,b; Morales-González et al., 2014; Elseidy et al., 2014; Flores-Garrido et al., 2015; Gao et al., 2015; Li and Wang, 2015; Moussaoui et al., 2016; Wu et al., 2017).

FAS mining algorithms have become important tools in several applications, such as: analysis of biochemical structures (Chen et al., 2007; Xiao et al., 2008; Jia et al., 2011; Li and Wang, 2015); genetic networks analysis (Song and Chen, 2006); circuits, links and social networks analysis (Holder et al., 1992; Moussaoui et al., 2016); and image classification (Acosta-Mendoza et al., 2012a; Gao et al., 2015; Flores-Garrido et al., 2015), among others. In some of these applications there could be more than one relationship between two vertices, producing a multi-graph representation (see Figure 1.2). A multi-graph is a graph that allows having more than one edge between a pair of vertices (multi-edges), as well as edges connecting a vertex to itself (loops).

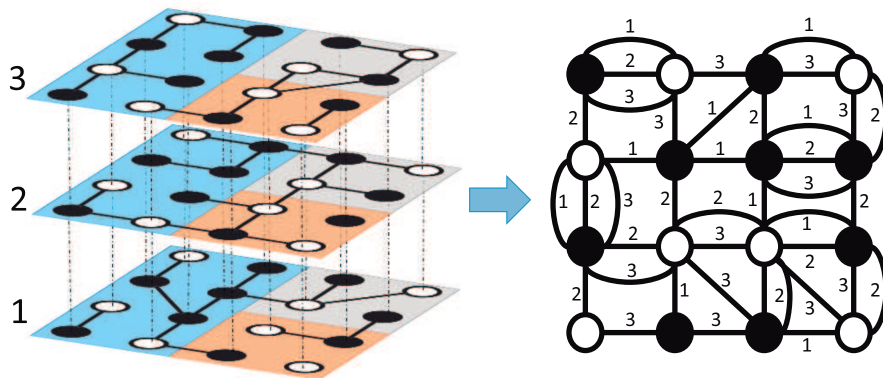
An example of applications that use multi-graph representation can be seen in social network analysis, as it is illustrated in Figure 1.2(a), where the entities (persons, videos, objects, etc.) can be modeled as vertices and multi-edges may represent the different interactions among entities (Jabeur et al., 2012; Papalexakis et al., 2013; Cazabet et al., 2015; Goonetilleke et al., 2015; Verma and Bharadwaj, 2017). Other networks such as transportation, routing, railway and traveling can be modeled with multi-graphs (see Figure 1.2(b)) for determining the minimum cost of deliveries (Setak et al., 2015) by predicting the contacts between bus stations (Wang et al., 2015); or finding the cheapest path for traveling via plane



(a) A social network with more than one interaction between some entities.



(b) A transportation network with more than one path between places.



(c) An image with three different viewpoints.

Figure 1.2: Examples of data modeled as multi-graphs.

(Hulianytsky and Pavlenko, 2015), among others (Terroso-Saez et al., 2015; Wei et al., 2015). Likewise, several works use multi-graphs for representing images (see Figure 1.2(c)) in different applications (Kropatsch et al., 2005; Morales-González and García-Reyes, 2010, 2013; Youssef et al., 2015). In these works, the authors have stated that by using multi-graphs the nature of the problem can be better modeled than by using simple-graphs.

However, multi-graph representations are not being properly exploited because of the lack of algorithms for handling multi-graph collections. Thus, this research is focused on developing algorithms for mining FASs from multi-graph collections. For this purpose, we explore two alternatives: (1) transforming multi-graphs into simple-graphs for applying traditional FAS miners, and (2) developing new algorithms for mining FASs directly from multi-graph collections.

On the other hand, when FASs are mined, usually a large number of subgraphs is obtained (Jia et al., 2011; Acosta-Mendoza et al., 2013; Li and Wang, 2015; Acosta-Mendoza et al., 2016b; El Islem Karabadji et al., 2016; Wu et al., 2017) and discovering a subset of FASs that could be used for representing the whole collection of FASs is a challenge (Jiang et al., 2013; Ramraj and Prabhakar, 2015; Emmert-Streib et al., 2016). In order to mine a subset of FASs, some authors (Flores-Garrido et al., 2014; Li and Wang, 2015; Liu and Gribskov, 2015; Chalupa, 2016; Chen et al., 2016; El Islem Karabadji et al., 2016; Hahn et al., 2016; Hao et al., 2016; Segundo et al., 2016; Salma, 2016; Demetrovics et al., 2017; Wu et al., 2017) have proposed computing only a representative subset from the whole set of FASs, for example maximal[1], clique[2] or closed[3] FASs, among others. We denote these maximal, clique and closed subgraphs as representative patterns because they are used for representing the whole set of FASs. In this research, we are also interested in developing algorithms for mining only a representative subset of FASs from a multi-graph collection.

[1]A maximal FAS is a FAS that is not sub-isomorphic to another FAS (Flores-Garrido et al., 2014).
[2]A clique FAS is a FAS such that every vertex is connected to every other by an edge (Rahman, 2017).
[3]A closed FAS is a FAS that is not sub-isomorphic to another FAS with the same frequency (Yan and Han, 2003).

1.1 Motivation

FAS mining is an important problem in graph mining. In this type of mining, variations in vertex and edge labels, as well as changes in the structure of graphs are taken into account for detecting FASs. Better results have been reported using approximate graph miners (Jia et al., 2011; Li et al., 2012; Acosta-Mendoza et al., 2012c; Acosta-Mendoza, 2013; Flores-Garrido et al., 2015; Gao et al., 2015; Acosta-Mendoza et al., 2016b; Moussaoui et al., 2016; Muñoz-Briseño et al., 2016; Wu et al., 2017) than those results reported by the exact ones. However, all reported FAS mining algorithms have been designed to work with simple-graphs, and, as we have previously mentioned, there are some real-world applications where multi-graphs are necessary for modeling the nature of data (Cazabet et al., 2015; Goonetilleke et al., 2015; Hulianytsky and Pavlenko, 2015; Setak et al., 2015; Terroso-Saez et al., 2015; Wang et al., 2015; Wei et al., 2015; Youssef et al., 2015; Verma and Bharadwaj, 2017).

On the other hand, the FAS mining algorithms reported for simple-graphs commonly mine large sets of FASs. There are some works focused on this problem, where the main idea is to develop methods for identifying a representative subset of FASs (Acosta-Mendoza, 2013; Acosta-Mendoza et al., 2013, 2016b; El Islem Karabadji et al., 2016; Hao et al., 2016; Salma, 2016). However, these works are based on a post-processing stage taking into account the information provided by the problem context. Therefore, another important challenge is to compute only representative (in this PhD. we focused on maximal, closed, and clique FASs) FASs from multi-graph collections during the mining process.

1.2 Aims

The general aim of this thesis is:

- Proposing algorithms for mining representative frequent approximate subgraphs in multi-graph collections, which must be competitive in time with the FAS mining al-

gorithms for simple-graph collections.

For accomplishing the general aim, five specific objectives are proposed; the first focused on developing algorithms for mining all FASs. From the second to the fourth the focus is developing algorithms for mining different types of representative FASs. Finally the last one is focused on shown hoy to use the mined FASs on a specific task.

1. Proposing an algorithm for frequent approximate subgraph mining in multi-graph collections.
2. Proposing an algorithm for mining maximal frequent approximate subgraphs in multi-graph collections.
3. Proposing an algorithm for mining closed frequent approximate subgraphs in multi-graph collections.
4. Proposing an algorithm for mining clique frequent approximate subgraphs in multi-graph collections.
5. Adapting a classification method based on frequent approximate subgraphs, for evaluating the accuracy and performance of the representative subgraphs mined by our algorithms.

1.3 Overview and Results

The main contribution of this research is the introduction of algorithms for mining all FASs and representative (maximal, closed and clique) FASs over multi-graph collections. We also extend the canonical adjacency matrix and depth-first search canonical forms for representing isomorphic multi-graphs.

In this thesis, we propose allEdges for mining all FASs on multi-graph collections by means of transforming multi-graphs into simple-graphs, applying a simple-graph FAS miner

and translating the identified simple-graph FASs to multi-graph FASs. Then, with the aim of speeding up the multi-graph mining process, we propose an alternative method, called onlyMulti, which is also based on graph transformations. onlyMulti allows mining FASs on multi-graph collections faster than allEdges. However, when onlyMulti is used some FASs are missed, while allEdges always mines all FASs of a multi-graph collection. These methods allow us to use any simple-graph FAS miner for mining multi-graph FASs.

In order to accelerate the multi-graph mining process, we propose MgVEAM for directly mining all FASs over multi-graph collections without a transformation process. We introduce an extension of the canonical form based on Canonical Adjacency Matrices (CAM) for representing multi-graphs, which was used in MgVEAM. We also extend the Depth-First Search (DFS) canonical form for representing isomorphic multi-graphs, and used it to introduce a new algorithm, called AMgMiner, for mining all FASs on multi-graph collections. AMgMiner is faster than MgVEAM, but requires more memory for mining all the FASs. All our proposals were evaluated with several experiments on different multi-graph collections.

With the aim of reducing the amount of identified FASs, we propose two algorithms, called GenCloMgVEAM and CliqueAMgMiner, for mining representative (i.e., maximal, closed or clique) FASs directly from multi-graph collections. GenCloMgVEAM is an extension of MgVEAM for mining generalized closed FASs on multi-graph collections. GenCloMgVEAM is able to mine maximal FASs and traditional closed FASs. In this direction, we also propose CliqueAMgMiner, which is an extension of AMgMiner for mining clique FASs on multi-graph collections. Through several experiments over different multi-graph collections we were able to show that our representative FAS miners allows reducing the amount of FASs without increasing the computational cost of MgVEAM and AMgMiner.

1.4 Document Description

This Ph.D. thesis is structured as follows. In Chapter 2, some concepts, needed for understanding the rest of this thesis, are provided. In Chapter 3, the related work is discussed. In Chapter 4, two new methods based on graph transformations for mining FASs from multi-graph collections are introduced. Later, two new algorithms for directly mining FASs from multi-graph collections, without graph transformations, are proposed in Chapter 5. Chapters 4 and 5 address the first specific aim. Next, in Chapter 6, we introduce two algorithms for mining representative FASs (generalized closed FASs and clique FASs) from multi-graph collections; this chapter addresses the second, third and fourth specific objectives. Our conclusions and some future work directions, as well as the contributions and publications derived from this Ph.D. research are presented in Chapter 7. In Appendix A, for addressing the fifth specific objective, we show some experiments about how to use the FASs mined by our proposed algorithms. Finally, in Appendix B, we include proofs of the correctness of our methods based on graph transformations.

CHAPTER 2

BASIC CONCEPTS

In this chapter, some basic concepts needed to define the frequent approximate subgraph (FAS) mining problem in multi-graphs are presented. Additionally, some concepts used to define the representative frequent approximate subgraph mining problem are also provided.

This chapter is structured as follows. In Section 2.1, we present basic concepts on labeled graph, simple-graph and multi-graph. In Section 2.2, basic concepts related to isomorphism, sub-isomorphism, similarity between graphs, approximate isomorphism and sub-isomorphism are defined. In Section 2.3, we present basic concepts on approximate support, FAS, FAS mining and representative FAS mining. Finally, in Section 2.4, we include a summary of this chapter.

2.1 Labeled Simple-Graph and Multi-Graph

In this research, as a first approximation to the FAS mining on multi-graphs, we will focus on undirected labeled multi-graphs and directed labeled multi-graphs will be treated as future work. Thus, the first concepts to be defined are labeled graph, simple-graph and multi-graph.

Definition 2.1 (Labeled graph). *Let L_V and L_E be two label sets for vertices and edges, respectively, a labeled graph G is a 5-tuple $(V_G, E_G, \phi_G, I_G, J_G)$ where:*

- *V_G is a set of vertices,*
- *E_G is a set of edges,*
- *$\phi_G : E_G \to V_G^\bullet$ is a function that returns the pair of vertices of V_G which are connected by a given edge, where $V_G^\bullet = \{\{u, v\} | u, v \in V_G\}$,*

- *$I_G : V_G \rightarrow L_V$ is a labeling function for assigning labels to vertices in V_G,*
- *$J_G : E_G \rightarrow L_E$ is a labeling function for assigning labels to edges in E_G.*

In Figure 2.1, a labeled graph G with $V_G = \{v_0, v_1, v_2\}$ and $E_G = \{e_0, e_1, e_2\}$ is shown. In this example, according to Definition 2.1, $\phi_G(e_0) = \{v_0, v_2\}$, $\phi_G(e_1) = \{v_0, v_1\}$ and $\phi_G(e_2) = \{v_1, v_2\}$, as well as $I_G(v_0) = A$, $I_G(v_1) = B$, $I_G(v_2) = C$, $J_G(e_0) = 0$, $J_G(e_1) = 2$ and $J_G(e_2) = 1$.

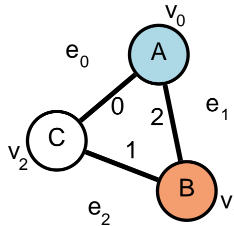


Figure 2.1: Example of a labeled graph G, where $V_G = \{v_0, v_1, v_2\}$, $E_G = \{e_0, e_1, e_2\}$, $L_V = \{A, B, C\}$ and $L_E = \{0, 1, 2\}$.

For undirected labeled graphs, the domain of all possible labels is denoted as $L = L_V \cup L_E$. Henceforth, when we refer to a graph we assume an undirected labeled graph unless we specify the contrary. In Figure 2.2, we show examples of undirected labeled graphs with $L_V = \{A, B, C\}$ and $L_E = \{0, 1, 2\}$.

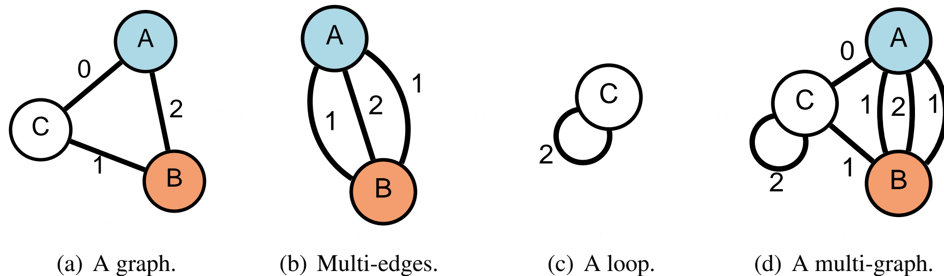


Figure 2.2: Example of different type of undirected labeled graphs with $L_V = \{A, B, C\}$ and $L_E = \{0, 1, 2\}$.

Multi-edges, as it is shown in Figure 2.2(b), are different edges connecting the same pair of vertices (i.e., e and e' are multi-edges if $e \neq e'$ and $\phi_G(e) = \phi_G(e') = \{u, v\}$ such that

$u, v \in V_G$, $u \neq v$). A loop, as it can be seen in Figure 2.2(c), is an edge connecting a vertex to itself (i.e., when $\phi_G(e) = \{u\}$ since $\phi_G(e) = \{u, v\}$ with $v = u$; in a loop $|\phi_G(e)| = 1$). Then, a multi-graph is a graph where more than one edge between a pair of vertices (multi-edges) is allowed, including loops (edges connecting a vertex to itself). In Figure 2.2(d), an example of a multi-graph is shown, where there are multi-edges connecting the vertices A and B, and the vertex C contains a loop. Then, the concepts of simple-graph and multi-graph are defined as follows:

Definition 2.2 (Simple-graph and multi-graph)**.** *A graph G is a* simple-graph *if it has no loops and no multi-edges; otherwise, G is a* multi-graph.

2.2 Graph Similarity

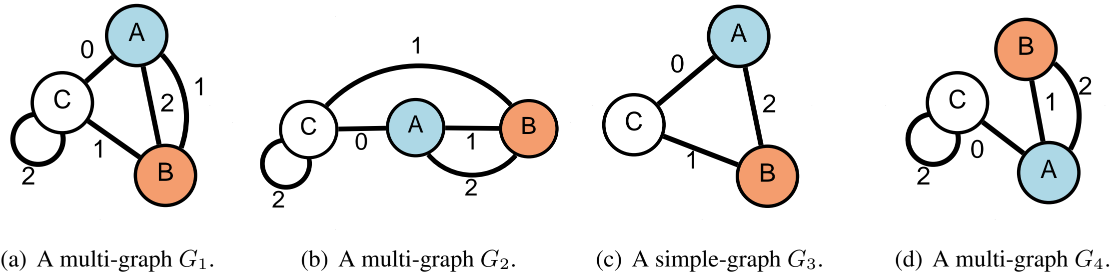
In exact graph mining, graph matching is performed by means of graph isomorphism. For both, simple-graphs and multi-graphs, isomorphism and sub-isomorphism between two graphs are defined as follows:

Definition 2.3 (Isomorphism and sub-isomorphism)**.** *Given two graphs $G_1 = (V_{G_1}, E_{G_1}, \phi_{G_1}, I_{G_1}, J_{G_1})$ and $G_2 = (V_{G_2}, E_{G_2}, \phi_{G_2}, I_{G_2}, J_{G_2})$, the pair of functions (f, g) is an isomorphism between these graphs iff $f : V_{G_1} \rightarrow V_{G_2}$ and $g : E_{G_1} \rightarrow E_{G_2}$ are bijective functions, such that:*

- $\forall u \in V_{G_1} : f(u) \in V_{G_2}$ *and* $I_{G_1}(u) = I_{G_2}(f(u))$
- $\forall e_1 \in E_{G_1}$, *where* $\phi_{G_1}(e_1) = \{u, v\}$*:* $e_2 = g(e_1) \in E_{G_2}$, *and* $\phi_{G_2}(e_2) = \{f(u), f(v)\}$ *and* $J_{G_1}(e_1) = J_{G_2}(e_2)$.
- $\forall e_1 \in E_{G_1}$, *where* $\phi_{G_1}(e_1) = \{v\}$*:* $e_2 = g(e_1) \in E_{G_2}$, *and* $\phi_{G_2}(e_2) = \{f(v)\}$ *and* $J_{G_1}(e_1) = J_{G_2}(e_2)$.

If there is an isomorphism between G_1 and G_2, then we say that G_1 and G_2 are isomorphic. Besides,

if G_1 is isomorphic to a subgraph of G_2, then there is a sub-isomorphism between G_1 and G_2; in this case we say that G_1 and G_2 are sub-isomorphic (see Figure 2.3).



(a) A multi-graph G_1. (b) A multi-graph G_2. (c) A simple-graph G_3. (d) A multi-graph G_4.

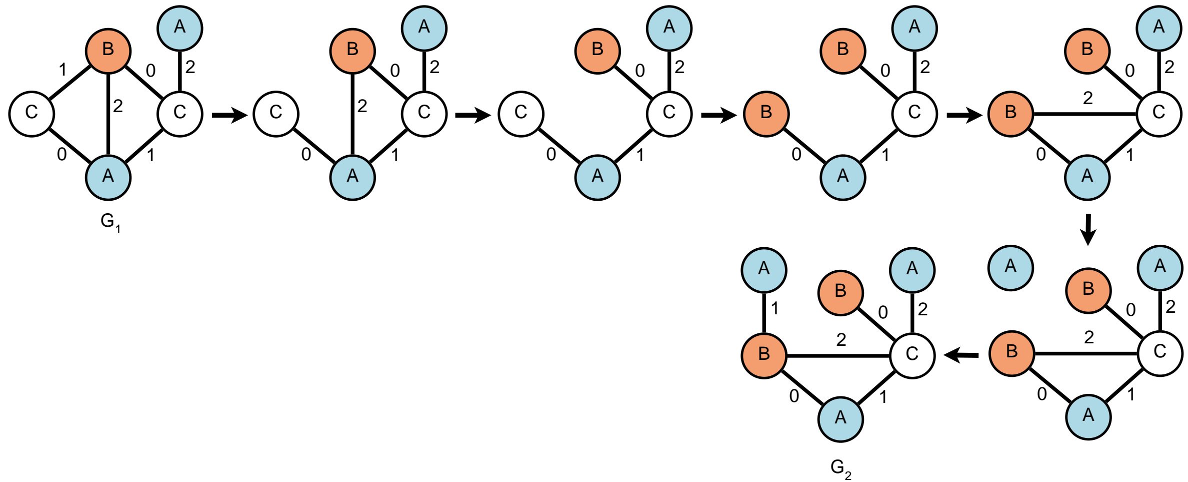
Figure 2.3: Example of three multi-graphs and a simple-graph, where there is an isomorphism between G_1 and G_2, and G_3 and G_4 are sub-isomorphic to both G_1 and G_2.

Exact graph mining algorithms use isomorphism (see Definition 2.3) between graphs for graph matching. However, sometimes graph databases contain noise, and the graphs have small variations in vertices, edges and labels. For this reason, in order to deal with these variations, certain flexibility at the graph matching is required. Approximate graph matching allows identifying patterns that could be missed by using exact graph matching (Cook and Holder, 1994; Jia et al., 2009; Morales-González et al., 2014; Flores-Garrido et al., 2015).

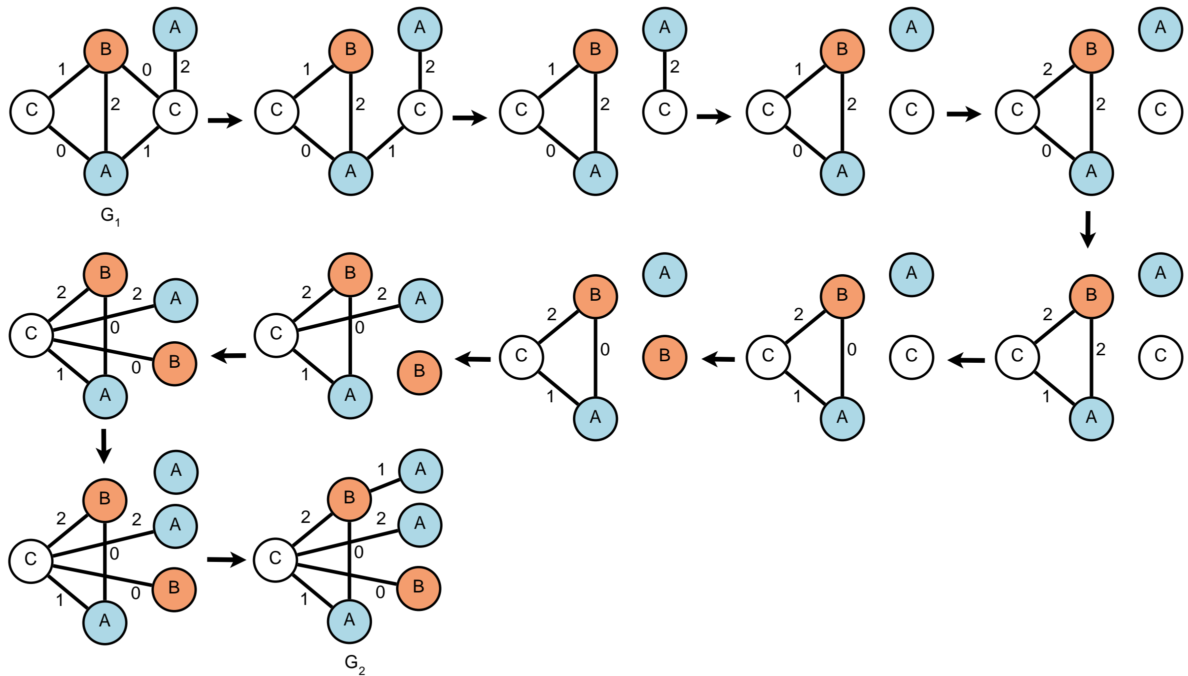
We are interested in graph mining based on approximate graph matching, where *approximate graph matching* consists in identifying graphs that are similar but not identical. Several proposals for computing approximate graph matching have been reported, such as: edit distance (Holder et al., 1992; Flores-Garrido et al., 2015; Gao et al., 2015), homeomorphism (Xiao et al., 2007, 2008), label substitutions (Jia et al., 2009, 2011; Acosta-Mendoza et al., 2012a), among others (Zhang et al., 2007; Zhang and Yang, 2008; Zou et al., 2010a,b; Li et al., 2012). From these proposals, the edit distance is the most used, which is defined as follows:

Definition 2.4 (Similarity between two graphs based on the edit distance)**.** *Let G_1 and G_2 be two labeled multi-graphs, the edit distance between G_1 and G_2, denoted as $d(G_1, G_2)$, is the minimum number of edit operations (i.e., insertion, deletion, and vertex or edge substitution) needed to trans-*

form G_1 into G_2.



(a) Example of 6 edit operations for transforming G_1 into G_2, where two edges are deleted, a vertex is substituted, and a vertex and two edges are inserted.



(b) Example of 11 edit operations for transforming G_1 into G_2, where three edges are deleted, a vertex and three edges are substituted, and a vertex and three edges are inserted.

Figure 2.4: Example of two different edit operation sequences for transforming a graph G_1 into another one G_2: (a) a sequence of 6 edit operations and (b) a sequence of 11 edit operations.

An example of the edit distance is shown in Figure 2.4, supposing that the two sequences of edit operations illustrated in Figures 2.4(a-b) are the only two possible ways for

transforming G_1 into G_2. The edit distance between G_1 and G_2, according to Definition 2.4, is the number of edit operations in the sequence shown in Figure 2.4(a), i.e., $d(G_1, G_2) = 6$; while in Figure 2.4(b), $d(G_1, G_2) = 11$ since eleven edit operations are needed for transforming G_1 into G_2. Notice that the graphs G_2 in both Figures 2.4(a-b) are isomorphic.

As we can see in Figure 2.4, by using the edit distance it is possible to evaluate the similarity between two graphs (G_1 and G_2) which have different numbers of vertices and edges, as well as different vertex and edge labels. In this way, variations in vertex and edges labels (i.e., substitution of vertices and edges), as well as variations in the graph structure (i.e., deletion and insertion of vertices and edges) can be allowed in the graph matching process. However, allowing these variations highly increases the computational cost of the algorithms for mining frequent subgraphs. This happens because allowing label substitutions combined with allowing variations in the graph structure produces a combinatorial explosion of the number of candidate subgraphs. For this reason, in this Ph.D. research, only variations in vertex and edge labels will be allowed but preserving the graph structure. In this scenario, a similarity function that allows performing approximate comparisons between labeled graphs but preserving the structure is required. Therefore, the following definition is introduced.

Definition 2.5 (Similarity between labeled graphs preserving the structure)**.** *Let G_1 and G_2 be two labeled multi-graphs, where V_{G_1}, E_{G_1}, V_{G_2}, and E_{G_2} are their sets of vertices and edges, respectively. The similarity between G_1 and G_2, preserving the graph structure, is defined as:*

$$sim(G_1, G_2) = \begin{cases} \max\limits_{(f,g)\in\Upsilon(G_1,G_2)} \Theta_{(f,g)}(G_1, G_2) & if\ \Upsilon(G_1, G_2) \neq \emptyset \\ 0 & otherwise \end{cases} \quad (2.1)$$

where $\Upsilon(G_1, G_2)$ is the set of all possible (can be more than one) isomorphisms between G_1 and G_2 without taking into account the labels, and $\Theta_{(f,g)}(G_1, G_2)$ is a similarity function for comparing the label information between G_1 and G_2, according to the isomorphism (f, g).

The similarity function $\Theta_{(f,g)}$ can be defined through different operations in vertex and edge labels, for example: $\Theta_{(f,g)}$ may be defined as the product of the label similarity values.

In this way, by using this $\Theta_{(f,g)}$, considering the two multi-graphs (G_1 and G_2) illustrated in Figure 2.5, and supposing that the labels A, C, and 1 can replace the labels C, B, and 2 with a similarity of 0.7, 0.6, and 0.8 respectively; if we apply a similarity based on exact matching then G_1 and G_2 are not similar; while applying a similarity based on approximate matching as the one defined above (see Definition 2.5, computing $\Theta_{(f,g)}$ as the product of the label similarity values) G_1 and G_2 are similar with $sim(G_2, G_1) = 0.7 * 0.6 * 0.8 = 0.336$.

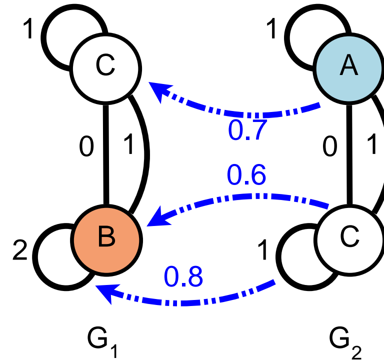


Figure 2.5: Example of the graph matching between two multi-graphs G_1 and G_2, where the label 2 can be replaced by the label 1 with a similarity of 0.8, the label B can be replaced by the label C with a similarity of 0.6 and the label C can be replaced by the label A with a similarity of 0.7.

As can be seen from Definition 2.6, by using $sim(G_1, G_2)$ in the isomorphism and sub-isomorphism between two multi-graphs, we can allow some variations handled by a similarity threshold. In this way, the approximate isomorphism and approximate sub-isomorphism can be defined.

Definition 2.6 (Approximate isomorphism and approximate sub-isomorphism)**.** *Let G_1, G_2 and G_3 be three labeled multi-graphs, let $sim(G_1, G_2)$ be a similarity function, preserving the graph structure, and let $\tau \in [0, 1]$ be a similarity threshold, there is an* approximate isomorphism *between G_1 and G_2 if $sim(G_1, G_2) \geq \tau$. Also, if there is an approximate isomorphism between G_1 and G_2, and G_2 is a subgraph of G_3, then there is an* approximate sub-isomorphism *between G_1 and G_3, denoted as $G_1 \subseteq_A G_3$.*

In Figures 2.5 and 2.6, two different ways for computing the approximate similarity between the same pair of multi-graphs (G_1 and G_2) is illustrated. Supposing that $\tau =$

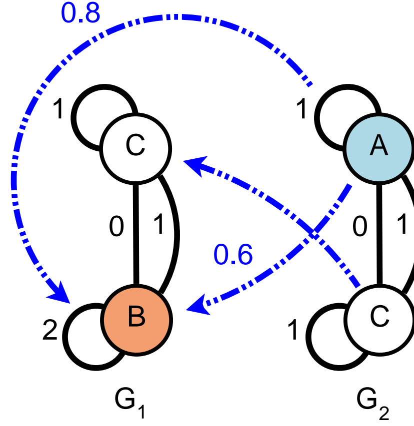


Figure 2.6: Example of the graph matching between two multi-graphs G_1 and G_2, where the label 2 can be replaced by the label 1 with a similarity of 0.8 and the label B can be replaced by the label A with a similarity of 0.6.

0.3 and $\Theta_{(f,g)}$ is the same as for the previous example, both similarities $sim(G_2, G_1) = 0.336$ (see Figure 2.5) and $sim_2(G_2, G_1) = 0.8 * 0.6 = 0.48$ (see Figure 2.6) fulfill with the similarity threshold τ. Therefore, according to Definition 2.6, these similarities can be used for computing two different approximate isomorphisms between G_1 and G_2. As we can notice, between two multi-graphs, more than one approximate similarity with different values can be computed. Thus, in order to have only one similarity value between two graphs, the following definition is used.

Definition 2.7 (Maximum inclusion degree)**.** *Let G_1 and G_2 be two labeled multi-graphs, let $sim(G_1, G_2)$ be a similarity function, preserving the graph structure; the* maximum inclusion degree *of G_1 in G_2 is defined as:*

$$maxID(G_1, G_2) = \max_{G \subseteq G_2} sim(G_1, G), \tag{2.2}$$

where $maxID(G_1, G_2)$ means the maximum value of similarity at comparing G_1 with all of the subgraphs of G_2.

Returning to the previous example (see Figures 2.5 and 2.6), supposing that these figures show all possible graph matching for computing similarities between G_1 and G_2, the

maximum inclusion degree is 0.48 because it is the maximum value of similarity at comparing G_2 with all of the subgraphs of G_1.

2.3 Approximate Subgraph Mining

For mining the FASs in our approximate approach, the approximate support of a subgraph in a multi-graph collection is defined as follows.

Definition 2.8 (Approximate support). *Let $D = \{G_1, \ldots, G_{|D|}\}$ be a multi-graph collection, let $sim(G_1, G_2)$ be a similarity function among graphs, let τ be a similarity threshold, and let G be a labeled multi-graph. Thus, the approximate support (denoted by appSupp) of G in D is obtained through Equation (2.3):*

$$appSupp(G, D) = \frac{\sum_{G_i \in D, G \subseteq_A G_i} maxID(G, G_i)}{|D|} \tag{2.3}$$

By using Equation (2.3), frequent approximate subgraphs can be defined as in the next definition.

Definition 2.9 (Frequent approximate subgraph (FAS)). *Let D be a multi-graph collection, let G be a multi-graph and let $minsupp$ be a support threshold, G is a frequent approximate subgraph in D iff $appSupp(G, D) \geq minsupp$.*

It is important to highlight that $minsupp$ must take values in the interval $[0, 1]$ since $appSupp(G, D)$ only gets values in the interval $[0, 1]$.

Taking into account the FAS definition, *frequent approximate subgraph mining* in a multi-graph collection consists in, given a support threshold, a similarity function between multi-graphs, and a similarity threshold, computing all the FASs in the multi-graph collection.

When all the FASs of a graph collection are mined, usually a large number of FASs is obtained. For this reason, some kinds of representative FASs have been proposed. Two

of these types of representative FASs, which allow to recompute the whole set of FASs, are maximal and closed FASs. A *maximal FAS* in a multi-graph collection is a FAS that is not sub-isomorphic to any other FAS, while a *closed FAS* in a multi-graph collection is a FAS that is not sub-isomorphic to any other FAS with the same approximate support. Another kind of representative FAS is *clique FAS*, which is a FAS where every vertex is connected to every other vertex.

The problem addressed in this Ph.D. research is *representative FAS mining* in multi-graph collections, which consists in, given a support threshold, a similarity function between multi-graphs and a similarity threshold, computing all the representative (maximal, closed or clique) FASs in a multi-graph collection.

2.4 Summary

In this chapter, some definitions and concepts to support the proposals of this research were provided. Starting with the labeled graph concept, we were able to differentiate simple-graphs from multi-graphs. Later, isomorphism and sub-isomorphism were defined for introducing the similarity between labeled graphs. This similarity concept is used as the basis for defining approximate isomorphism and approximate sub-isomorphism, which were used for defining the approximate support. Based on the approximate support, we introduce the concepts of Frequent Approximate Subgraph (FAS) and FAS mining. Finally, the concepts of maximal, closed and clique FASs were introduced for defining the representative FAS mining problem.

CHAPTER 3

RELATED WORK

In this chapter, for contextualizing the research problem at which this Ph.D. thesis is directed, we first review the most relevant algorithms for mining frequent subgraphs in graph databases. We separate these algorithms into those that work with a single graph and those that work with graph collections. We are interested in the latter kind of algorithms. Next, according to the used graph matching strategy, we separate the algorithms for mining frequent subgraphs in graph collections as exact and approximate algorithms. Also, this research is focused on the latter kind of algorithms. Then, the main Frequent Approximate Subgraph (FAS) mining algorithms, as well as those for mining representative FAS are reviewed. Finally, the only algorithm proposed for mining FASs in graph collections, which allows variations in vertex and edge labels but keeping the graph structure, will be described in detail since the algorithms developed in this Ph.D. research are based on this algorithm.

This chapter is structured as follows. In Section 3.1, we describe the main reported frequent subgraph mining algorithms reported in the literature. In Section 3.2, the algorithm most related to our research is detailed. Finally, a summary of this chapter is presented in Section 3.3.

3.1 Frequent Subgraph Mining

Many algorithms for mining frequent subgraphs have been developed to work on data represented as a single graph (Holder et al., 1992; Cook and Holder, 1994; Ketkar, 2005; Chen et al., 2007; Thomas et al., 2010; Zou et al., 2010b; Li et al., 2012; Zou et al., 2010a; Elseidy et al., 2014; Flores-Garrido et al., 2014, 2015; Abdelhamid et al., 2016; Hao et al., 2016; Moussaoui et al., 2016), and graph collections (Yan and Han, 2002, 2003; Huan et al., 2004;

Gago-Alonso et al., 2010b; Acosta-Mendoza et al., 2012a; Chen et al., 2012; Gao et al., 2015; Li and Wang, 2015; El Islem Karabadji et al., 2016; Alam et al., 2017). In this Ph.D. thesis, we are focused on algorithms for mining frequent subgraphs in graph collections, therefore we will refer only to this kind of algorithms.

Several algorithms for mining frequent subgraphs in graph collections have been proposed (Inokuchi et al., 2002; Kuramochi and Karypis, 2002; Borgelt, 2002; Huan et al., 2003; Thomas et al., 2009; Gago-Alonso et al., 2010b). These algorithms mine frequent subgraphs using breadth-first search by growing the subgraphs one vertex or one edge at a time. However, these algorithms are computationally expensive due to the process of generating candidate subgraphs, and verifying the frequency. In order to avoid this overhead, other algorithms based on the depth-first search (pattern-growth) have been developed (Yan and Han, 2002; Wang et al., 2004; Nijssen and Kok, 2004; Zhu et al., 2007; Gago-Alonso et al., 2008, 2010a; Gago-Alonso, 2015; El Islem Karabadji et al., 2016; Alam et al., 2017). These algorithms extend a frequent subgraph by adding a new edge in every possible position. However, a problem with this candidate generation process is that the same subgraph can be obtained many times (i.e., duplicate graph candidates). Thus, to reduce the generation of duplicate graphs, each frequent subgraph is extended as conservatively as possible.

The aforementioned algorithms were designed for mining frequent subgraphs using exact graph matching. However, in many real-world applications it is common for data to have some variations, which means that exact matching cannot be successfully applied (Holder et al., 1992; Chen et al., 2008; Acosta-Mendoza et al., 2012a; Li et al., 2012; Elseidy et al., 2014; Flores-Garrido et al., 2015; Gao et al., 2015; Li and Wang, 2015; Moussaoui et al., 2016). Therefore, it is important to allow certain level of variability, for example variations in vertex and edge labels, and mismatching in vertices and edges. For this reason, in the context of graph mining, it is necessary to evaluate the similarity between graphs considering approximate graph matching (Conte et al., 2004; Gao et al., 2010; Santhi and Padmaja, 2015; Emmert-Streib et al., 2016). In this way, several algorithms based on approximate graph matching have been developed for mining FASs (González et al., 2001; Song and Chen, 2006;

Xiao et al., 2008; Jia et al., 2011; Acosta-Mendoza et al., 2012a,b; Morales-González et al., 2014; Gao et al., 2015; Li and Wang, 2015; Wu et al., 2017). Different approximate graph matching approaches have been used as basis for frequent subgraph mining algorithms, for example: edit distance (González et al., 2001; Song and Chen, 2006; Gao et al., 2015; Li and Wang, 2015), vertex/edge disjoint homeomorphism (Xiao et al., 2008), uncertain graphs (Han et al., 2010; Wang and Li, 2013; Liu et al., 2014; Hu et al., 2015; Santhi and Padmaja, 2015; Wu et al., 2017), and allowing only variations in labels (Jia et al., 2011; Acosta-Mendoza et al., 2012a). In Figure 3.1, we show a time line with the most important contributions in exact and approximate frequent subgraph mining.

In the edit distance approach, different heuristics based on edit operations have been used for comparing graphs in order to mine FASs in graph collections (González et al., 2001; Song and Chen, 2006; Zhang et al., 2007; Zhang and Yang, 2008; Gao et al., 2015; Li and Wang, 2015). However, it is common that FAS miners based on the edit distance do not mine all FASs, as it is the case of SUBDUECL (González et al., 2001) and FASMGED (Gao et al., 2015). Only a few FAS miners, for example CSMiner (Xiao et al., 2007, 2008), RAM (Zhang et al., 2007; Zhang and Yang, 2008) and REAFUM (Li and Wang, 2015), mine all FASs allowing variations in the graph structure.

We will focus on algorithms for mining FASs in multi-graph collections that allow variations in vertex and edge labels, keeping the graph structure. We are focused on this kind of algorithms for avoiding the combinatorial explosion of the number of candidates and their occurrences obtained when label substitutions and variations in the graph structure are combined in the mining process. However, from the algorithms reported, only VEAM (Acosta-Mendoza et al., 2012a,b) allows variations in both vertex and edge labels but keeping the graph structure.

Another interesting research line is the development of algorithms for mining representative FASs. This research line has been little studied where only RNGV (Song and Chen, 2006) for mining closed FASs, and APGM (Jia et al., 2009, 2011) that mines clique FASs have

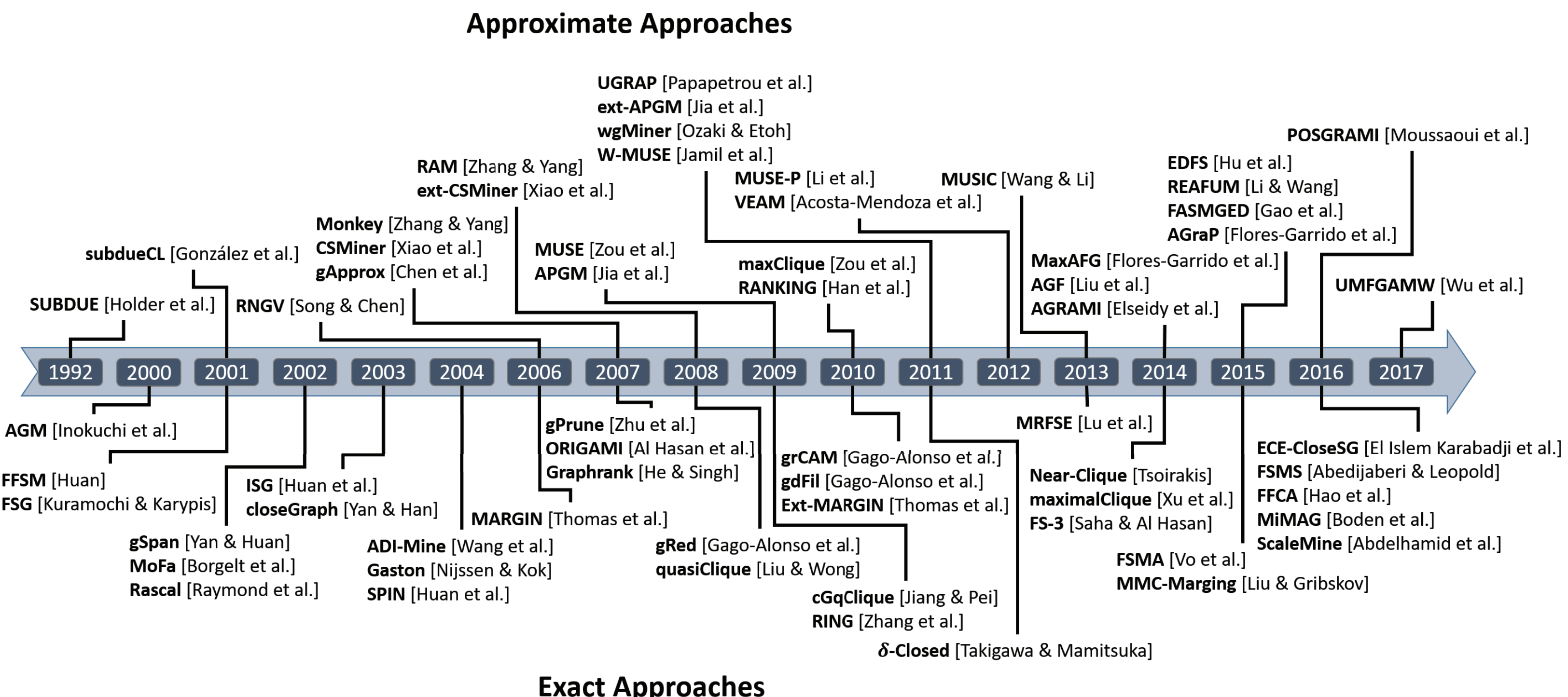


Figure 3.1: Time line of the most important contributions on frequent (exact and approximate) subgraph mining algorithms.

been reported. These algorithms mine representative FASs directly from simple-graph collections. However, none of them allows variations in both vertex and edge labels maintaining the graph structure.

As we can see, there are no representative FAS miners that allow variations in vertex and edge labels, keeping the graph structure. Only VEAM, which mines all the FASs in simple-graph collections, allows this kind of variations but just in simple-graph collections. Thus, in this work, VEAM will be used as basis for developing algorithms that mine representative FASs in multi-graph collections. For this reason, in the next section, the VEAM algorithm will be described.

3.2 VEAM

The VEAM (Vertex and Edge Approximate graph Miner) algorithm (Acosta-Mendoza et al., 2012a,b) mines all FASs in a simple-graph collection allowing variations in vertex and edge labels, keeping the graph structure. For allowing these variations, VEAM uses substitution matrices, which contain the probability of a label to be replaced by another one (see Definition 3.1).

Definition 3.1 (Substitution matrix). *A substitution matrix $M = (m_{i,j})$ is an $|L| \times |L|$ matrix indexed by a label set L, where $m_{i,i} > m_{i,j}, \forall j \neq i$. An entry $m_{i,j}$ $(0 \leq m_{i,j} \leq 1, \sum_j m_{i,j} = 1)$ in M contains the probability of the label i to be replaced by the label j.*

It is important to highlight that a substitution matrix can be non-symmetric because it is possible that a label v can replace another label w but the label w cannot replace the label v.

In VEAM, two substitution matrices are used: one for edge labels (ME) and another one for vertex labels (MV). Based on these matrices, the similarity function between graphs used by VEAM is defined as follows.

Definition 3.2 (Similarity function $\Theta_{(f,g)}$ based on substitution matrices)**.** *Let G_1 and G_2 be two graphs, and let MV and ME be two substitution matrices in L_V and L_E, respectively. The similarity function is defined as:*

$$\Theta_{(f,g)}(G_1, G_2) = \prod_{v \in V_{G_1}} \frac{MV_{I_{G_1}(v), I_{G_2}(f(v))}}{MV_{I_{G_1}(v), I_{G_1}(v)}} * \prod_{e \in E_{G_1}} \frac{ME_{J_{G_1}(e), J_{G_2}(g(e))}}{ME_{J_{G_1}(e), J_{G_1}(e)}} \tag{3.1}$$

where (f, g) is the isomorphism between G_1 and G_2, $MV_{I_{G_1}(v), I_{G_2}(f(v))}$ and $MV_{I_{G_1}(v), I_{G_1}(v)}$ are the cells $MV_{i,j}$ and $MV_{i,i}$ respectively of the vertex substitution matrix with $i = I_{G_1}(v)$ and $j = I_{G_2}(f(v))$, and $ME_{J_{G_1}(e), J_{G_2}(g(e))}$ and $ME_{J_{G_1}(e), J_{G_1}(e)}$ are the cells $ME_{q,r}$ and $ME_{q,q}$ respectively of the edge substitution matrix with $q = J_{G_1}(e)$ and $r = J_{G_2}(g(e))$. Notice that, as the function $\Theta_{(f,g)}$ is based on substitution matrices and these matrices are non-symmetric, then this similarity function is non-symmetric.

Using Definition 3.2 the occurrences (see Definition 3.3) of each FAS into the graph collection can be computed.

Definition 3.3 (Occurrence)**.** *Let G_1, G_2 and T be three graphs, where T is a subgraph of G_2, and let $sim(G_1, T)$ a similarity function according to Definition 2.5, using $\Theta_{(f,g)}$ as in Definition 3.2, then T is an occurrence of G_1 in G_2, using a similarity threshold τ, if $sim(G_1, T) \geq \tau$.*

Based on the definitions 3.2 and 3.3, VEAM computes and stores all the occurrences of each subgraph candidate P_j in a simple-graph collection D. Then, taking into account the occurrences of P_j, only the subset of simple-graphs $D_j \subseteq D$, where P_j has at least one occurrence, is traversed for growing P_j. VEAM reduces the search space to D_j because a FASs only can be grown in a simple-graph where it has occurrences.

In VEAM, adjacency matrices for representing each simple-graph of a collection are used. Notice that, since the adjacency matrix of an undirected simple-graph is symmetric, only the lower or upper triangular adjacency matrices are needed for representing simple-graphs.

Definition 3.4 (Adjacency matrix of a labeled simple-graph). *Let $v_i, v_j \in V_G$ be two arbitrary vertices of a given simple-graph G, the adjacency matrix $M = (m_{i,j})_{|V_G| \times |V_G|}$ for G is defined by:*

$$m_{i,j} = \begin{cases} I_G(v_i) & if\ i = j \\ J_G(e) & if\ i \neq j, e \in E_G, \phi_G(e) = \{v_i, v_j\} \\ - & otherwise \end{cases} \tag{3.2}$$

The symbol "$-$" is used for representing the edge label absence.

In order to simplify the graph representation based on adjacency matrices, an adjacency matrix *code* (a sequence of labels) for each matrix can be built (Kuramochi and Karypis, 2002; Gago-Alonso et al., 2010b). This code is built concatenating lower (upper) rows of a triangular adjacency matrix. Equation (3.3) is used for obtaining the code of an adjacency matrix M of a graph with n vertices.

$$code(M) = m_{1,1}m_{2,1}m_{2,2}m_{3,1}m_{3,2}m_{3,3} \ldots m_{n,n} \tag{3.3}$$

A simple-graph G may have more than one adjacency matrix code, according to each permutation of the vertices into the matrix. Then, for achieving a unique representation for isomorphic graphs, the canonical adjacency matrix (CAM) code is defined as follows.

Definition 3.5 (Canonical adjacency matrix of a labeled simple-graph). *The canonical adjacency matrix (CAM) of a graph G is the adjacency matrix of G that has the maximal (minimal) code (i.e., CAM code) among all its possible codes.*

The process to compute the minimal (maximal) CAM code is computationally expensive. Then, for speeding up this process, an alternative was proposed in (Kuramochi and Karypis, 2002). In this alternative, as it is illustrated in Figure 3.2, the vertex label set, as well as the degree-based order are used as vertex invariants[1]. Using these vertex invariants,

[1]Vertex invariants are properties useful for keeping the same vertex ordering in different isomorphism mappings (Read and Corneil, 1977; Kuramochi and Karypis, 2002).

all vertices of a graph can be partitioned into equivalence classes, where vertices of the same class have equivalent vertex invariant values. This criterion allows only performing permutations of the vertices into the same equivalence partition instead of performing permutations into the whole set of vertices (see Figure 3.2(c)). This can be done because two isomorphic graphs will lead to the same partitioning of the vertices and therefore the same canonical code is computed from them. This alternative, for computing the canonical code of simple-graphs, in most cases, reports a considerable reduction of permutations between vertices (Kuramochi and Karypis, 2002). Therefore, VEAM uses the CAM proposed in (Kuramochi and Karypis, 2002) for representing the FAS candidates. An example of how VEAM computes the CAM code of a simple-graph G is illustrated in Figure 3.2. In this example, only thee permutations between the cells of the starting adjacency matrix are performed for obtaining the CAM code of G; instead of the 6! permutations required if no equivalence classes were used.

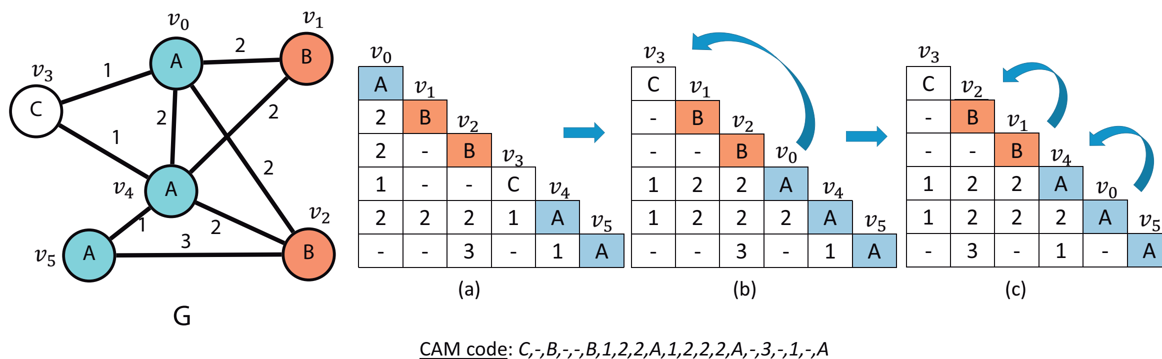


Figure 3.2: Adjacency matrices for a simple-graph G, where its CAM code is obtained from the matrix (c), which is the CAM of G according to the alternative proposed in (Kuramochi and Karypis, 2002).

The VEAM algorithm (Acosta-Mendoza et al., 2012a,b) starts mining all frequent approximate single-vertex subgraphs. Then, following a *Depth-First Search* (DFS) approach, each frequent single-vertex is extended by recursively adding a single-edge at a time.

In the recursive pattern-growth step of VEAM, all children of each FAS G, which satisfy the similarity constraint using Definition 3.2, are computed; each child of G is a candidate graph. As the same subgraph can be obtained from different candidate graphs,

an isomorphism test over each computed candidate should be performed. For speeding up these isomorphism tests, each FAS is represented by a canonical form based on adjacency matrices (CAM). By comparing the CAMs of the subgraphs, the isomorphic candidates (i.e., duplicate candidates) are identified and their occurrences are assigned to only one of them. These comparisons between CAMs allow us to eliminate duplicities in the candidate set. Once the candidate set is computed, only those frequent candidates, which were not identified in previous steps, are stored as FASs in the collection. The stop condition in the recursion is supported by the downward closure property, which ensures that a non-frequent subgraph will just produce non-frequent children.

3.3 Summary

In this chapter, we have reviewed the main algorithms for frequent subgraph mining, specially those algorithms for mining FASs in graph collections. We focused on the VEAM algorithm because it allows variations in vertex and edge labels keeping the graph structure, which constitutes the research line of this research.

As it can be seen from our state-of-the-art review, there is no FAS mining algorithm designed for working on multi-graph collections. However, as we mentioned in the introduction, there are real-world applications using multi-graphs for modeling the objects under study. Then, in these applications, it would be useful to apply algorithms for mining FAS in multi-graphs collections.

On the other hand, aiming at reducing the number of mined FASs, some FAS miners have been proposed for mining representative FASs, as RNGV and APGM. However, as it was mentioned, RNGV allow variations in graph structure, which is too computationally expensive; while APGM maintains the graph structure but it does not allow variations in both vertex and edge labels. Thus, to the best of our knowledge, there is no FAS miner, which only allows variations in vertex and edge labels, that mines representative FASs in

multi-graph collections.

In the next chapter, we present a variant based on graph transformations for mining FAS over multi-graph collections.

CHAPTER 4

MULTI-GRAPH PATTERN MINING BASED ON GRAPH TRANSFORMATIONS

To the best of our knowledge, there is not a Frequent Approximate Subgraph (FAS) mining algorithm designed for working with multi-graphs. However, several researchers have focused their efforts on developing algorithms for mining FASs in simple-graph collections. Thus, we propose a solution for mining FASs in multi-graph collections taking advantage of these efforts. Our solution consists in transforming a multi-graph collection into a simple-graph collection, mining FASs from the simple-graph collection by applying a FAS miner, and transforming the FASs into multi-graphs. Following this idea, we propose a method, called allEdges, (see Section 4.1) based on graph transformations that allows mining all FASs from a multi-graph collection. allEdges comprises: (1) $M2Simple1$ transformation algorithm, (2) a FAS mining algorithm, and (3) $S2Multi$ transformation back algorithm. By applying this method, we can mine all FASs from a multi-graph collection, but the graph transformation process increases the size of the graphs in the collections, which increases the computational cost of the mining process. Thus, with the aim of reducing this computational cost, we propose an alternative method, called onlyMulti, (see Section 4.2), which is faster than allEdges but only mines a subset of FASs from a multi-graph collection. This alternative method comprises: (1) $M2Simple2$ transformation algorithm, which transforms multi-graphs into simple-graphs differently than allEdges, (2) a FAS mining algorithm, and (3) $S2Multi$ transformation back algorithm.

4.1 Mining All FASs from a Multi-graph Collection

In this section, for introducing our proposed allEdges method based on graph transformations, we present the $M2Simple1$ transformation algorithm and the $S2Multi$ transformation back algorithm. The $M2Simple1$ algorithm allows transforming each multi-graph into a simple-graph, and $S2Multi$ performs a transformation back for returning a simple-graph FAS to a multi-graph context.

allEdges, as it is demonstrated in Appendix B, allows mining all FASs from a multi-graph collection. The main idea of allEdges consists in transforming each loop and non-loop edge (simple-edge or multi-edge) of a multi-graph collection into a new simple-edge and two new simple-edges, respectively. Notice that, during the mining process, a multi-edge can have occurrences on a simple-edge and vice versa. Thus, all edges (simple-edges or multi-edges) are transformed to guarantee the complete occurrence count, allowing to identify all FASs on a multi-graph collection. In this way, all edges are transformed into simple-edges without losing information of multi-graphs. Once the multi-graph collection has been transformed into a simple-graph one, we can apply any traditional FAS miner (i.e., APGM (Jia et al., 2011), VEAM (Acosta-Mendoza et al., 2012a) and REAFUM (Li and Wang, 2015), among others) for mining FASs, taking advantage of the large number of reported FAS mining algorithms. Later, all identified FASs are transformed, through a reverse process, into multi-graphs.

In summary, allEdges comprises three steps: (1) a multi-graph collection is transformed into a simple-graph collection (using $M2Simple1$), (2) the FASs are mined from this simple-graph collection by applying a traditional FAS miner, and (3) the mined FASs are transformed into multi-graphs for obtaining the FASs of the multi-graph collection (using $S2Multi$).

In the first step of $M2Simple1$, all the edges of each multi-graph in the collection are visited; identifying the loops and non-loop edges (multi-edges and simple-edges). Each loop of a multi-graph G' that connects a vertex $v \in V_{G'}$ is replaced by a new vertex w with a special label (k) and a simple-edge with the label of the loop, connecting v to w. This process

is shown in Figure 4.1 where each loop in G' is turned into a simple-edge in G. The special label k cannot be used as label in the multi-graph collection and during the mining process it cannot be replaced by any other label, except by itself; in this way, a loop will only match with other loops.

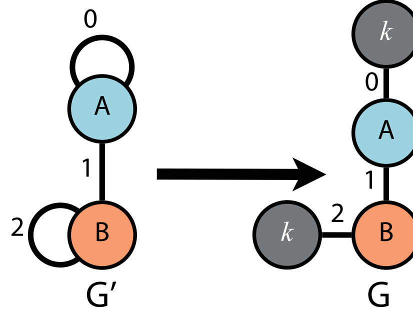


Figure 4.1: Example of the transformation of a multi-graph (G') with two loops into a simple-graph (G) using $M2Simple1$.

Each non-loop edge (i.e., a multi-edge or a simple-edge) e in G', with $\phi_{G'}(e) = \{u, v\}$ and $u \neq v$, is transformed into a new vertex w with a special label (p) and two edges (e_1 and e_2) both with the label of e; connecting u and v, respectively, to w. This process is shown in Figure 4.2, where the simple-graph G is obtained from the multi-graph G' by transforming each non-loop edge in G' into two simple-edges in G. The special label p, in the same way as k, cannot be used as label in the multi-graph collection and during the mining process it cannot be replaced by any other label, except by itself; in this way, a non-loop edge will only match with other non-loop edges.

Following the ideas above described, by traversing the edges of a given multi-graph G', we can transform them into new vertices and simple-edges, obtaining a simple-graph. The computational complexity of this transformation process is $O(m)$, where m is the number of the edges of G'; since transforming a single edge is $O(1)$. When this transformation process is applied over a multi-graph collection D' the complexity is $O(qd)$, where q is the average number of the edges in the graphs of D' and d is the number of multi-graphs of D'.

Once a multi-graph collection D' has been transformed into a simple-graph collection

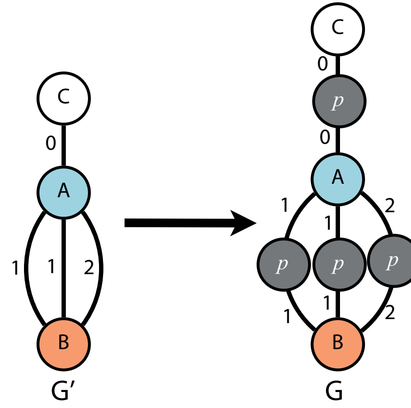


Figure 4.2: Example of the transformation of a multi-graph (G') with three multi-edges into a simple-graph (G).

D, a FAS miner can be applied for mining all FASs from D. In order to obtain the FASs of D', the mined FASs (simple-graphs) must be transformed back into multi-graphs. For doing that, a transformation process is required.

For transforming a FAS G (a simple-graph) into a multi-graph G', each edge $e \in E_G$ with $\phi_G(e) = \{u, v\}$ that has a vertex v with label k is transformed into a loop $\phi_{G'}(e') = \{u\}$ keeping the label of e. Each pair of edges e_1 and e_2 with $\phi_G(e_1) = \{u, w\}$ and $\phi_G(e_2) = \{v, w\}$ that have a common vertex w with label p are replaced by an edge e' with $\phi_{G'}(e') = \{u, v\}$ keeping the label of e_1 and e_2, which have the same label.

Following the aforementioned idea, by traversing all edges of a FAS G (a simple-graph) and replacing those edges that contain vertices with label p or k by multi-edges or loops, respectively, we can transform a simple-graph FAS into a multi-graph FAS. However, there are two cases where a simple-graph FAS should not be returned to a multi-graph. The first one is when a FAS is just a vertex with one of the special labels k or p, because this kind of vertices are produced by our transformation method but they are not part of the multi-graphs. The second case is when a vertex with the special label p is not connected with exactly two vertices, because the transformation process inserts this kind of vertices for replacing a multi-edge between a pair of vertices; therefore, if one of the connections is missing, the multi-edge cannot be rebuilt. An example of this last kind of FASs that cannot be returned to multi-

graphs is shown in Figure 4.3, where G_1 and G_2 have special vertices with label p that are not connected with two vertices. When a FAS does not contain any of these two cases, we say that it is returnable.

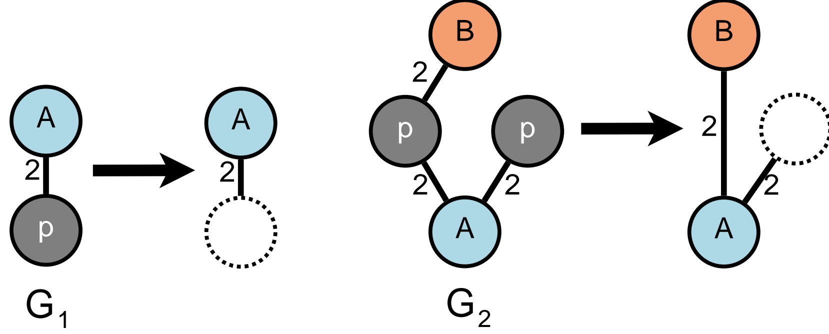


Figure 4.3: Example of two simple-graph FASs, G_1 and G_2, which cannot be returned into multi-graphs.

With the aim of identifying the FASs from the multi-graph collection, we introduce some conditions that the mined simple-graph FASs must fulfill for being returned to a multi-graph (see Definition 4.1).

Definition 4.1 (Returnable graph)**.** *Let k and p be the special labels used for representing loops and multi-edges, respectively. A simple-graph G is* returnable *to a multi-graph if it fulfills the following conditions:*

1. *Each vertex $v \in V_G$ with $I_G(v) = p$ has exactly two incident edges e_1 and e_2, such that $J_G(e_1) = J_G(e_2)$*
2. *Each vertex $v \in V_G$ with $I_G(v) = k$ has exactly one incident edge.*

In Figure 4.4, an example of the transformation process performed by our proposed method for mining all FASs from a multi-graph collection is illustrated. As we can see in this figure, the multi-graph collection $D' = \{G'_1, G'_2, G'_3\}$ is transformed into the simple-graph collection $D = \{G_1, G_2, G_3\}$. We called $M2Simple1$ to this transformation process. Next, by applying a FAS miner over the simple-graph collection D, all FASs are mined. Then,

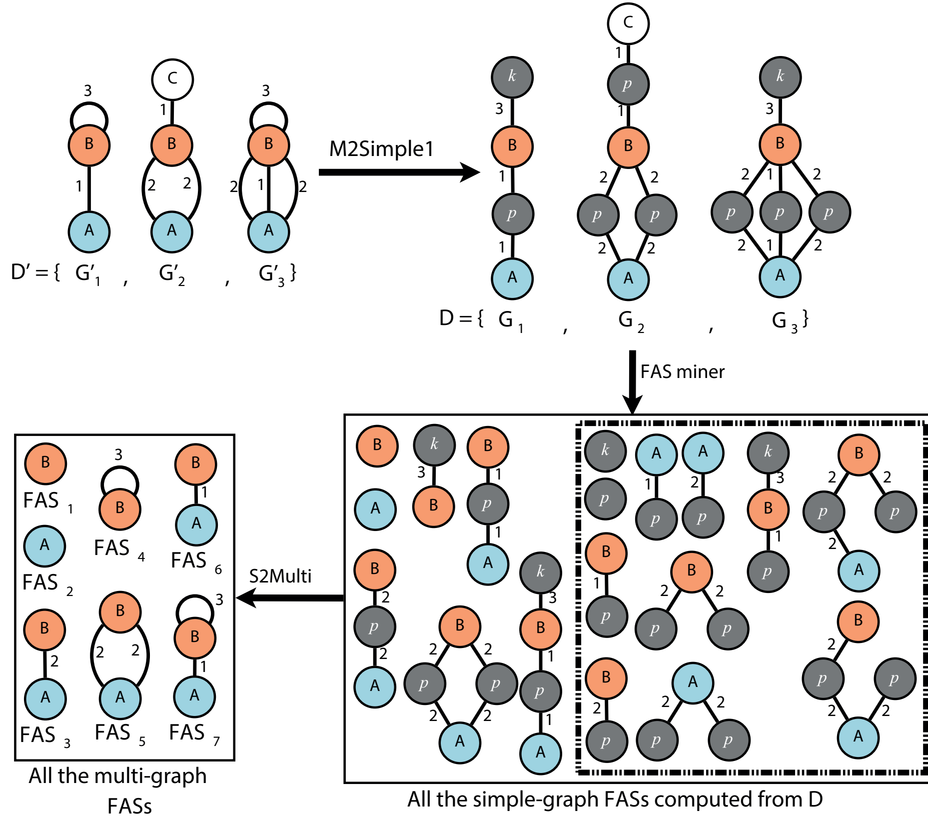


Figure 4.4: Example of FASs mined by using the proposed method for mining all the FASs from a multi-graph collection $D' = \{G'_1, G'_2, G'_3\}$ using the support threshold $minsup = 2/3$.

each computed FAS, which is a simple-graph, is transformed into a multi-graph. We called $S2Multi$ to this last transformation process. In fact, as we can see in Figure 4.4, the FASs not in the dashed square, obtained by applying a FAS miner, are returnable since the conditions of Definition 4.1 are fulfilled; while the FASs inside the dashed square cannot be transformed to multi-graphs because they do not represent subgraphs of the multi-graph collection. In this way, as we demonstrate in Appendix B, all multi-graph FASs can be mined by applying allEdges method.

The process of transforming a simple-graph FAS into a multi-graph ($S2Multi$) has a computational complexity $O(r)$, where r is the number of edges of the input FAS. When this process is applied over a FAS set C, it has a computational complexity $O(sc)$, where c is the number of FASs in C and s is the average number of edges in the FASs of C.

Summarizing, the proposed method for mining all the FASs from a multi-graph collection uses $M2Simple1$ for transforming the multi-graph collection into a simple-graph collection, and the FASs obtained by applying a FAS miner over the simple-graph collection are transformed back to multi-graphs by using $S2Multi$.

4.2 Mining a Subset of FASs from a Multi-graph Collection

In Section 4.1, we introduced a method that allows applying traditional FAS miners for mining all FASs from multi-graph collections. However, the process of transforming every edge into two simple-edges increases the size of the graphs in the collection and, therefore, the computational cost of the mining process. Thus, in this section, we propose an alternative method, which does not transform all edges, but not always is able to mine all FASs from a multi-graph collection (called *onlyMulti*). The difference between onlyMulti and allEdges is the way each one transforms a multi-graph into a simple-graph. Thus, in this section, we introduce the $M2Simple2$ algorithm, which is embedded into onlyMulti, for transforming multi-graphs into simple-graphs.

This method consists in only transforming loops and multi-edges, while simple-edges are kept without changes. Following this alternative, the process for transforming a multi-graph G' into a simple-graph G consists in replacing each loop and each multi-edge by new vertices and simple-edges in the same way as in allEdges. Notice that, in this case, the simple-edges are kept without change. In this way, simple-edges are not used as occurrences for the multi-edges and vice versa; thus, simple-edges cannot be used for computing the support of multi-edges and vice versa. Therefore, in this transformation method, the frequency of both multi-edges and simple-edges would be reduced; resulting in fewer FASs than those mined by allEdges. An example of this reduction will be presented, at the end of this section.

An example of how a multi-graph is transformed into a simple-graph is shown in Figure 4.5, where each loop in G' is transformed into a new vertex and a simple-edge in

G, and each multi-edge in G' is transformed into a new vertex and two simple-edges in G, obtaining the simple-graph G from the multi-graph G'.

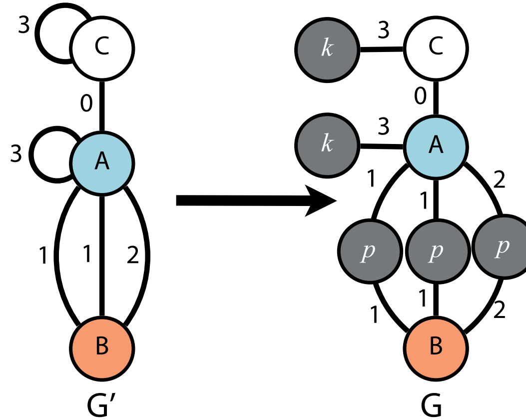


Figure 4.5: Example of the transformation of a multi-graph (G') with three multi-edges and two loops into a simple-graph (G).

The computational complexity of applying this transformation process over a multi-graph G' is $O(|E_{G'}|)$, where $|E_{G'}|$ is the number of edges of G'. This complexity is obtained considering that each edge of G' should be visited for deciding if the edge will be or not transformed. Then, this transformation process should be applied over each graph in the multi-graph collection. Thus, the computational complexity of the process for transforming a multi-graph collection into a simple-graph collection is $O(qd)$, where q is the average number of edges in the multi-graphs of the collection, and d is the number of multi-graphs in the collection.

Given a multi-graph collection, through the process above described, we can get a simple-graph collection. Then, in a similar way as in Section 4.1, we can apply a traditional FAS miner, and we can use the same process for transforming the returnable FASs into multi-graphs.

In Figure 4.6, an example of applying the transformation method (onlyMulti), proposed in this section, over a multi-graph collection D' is shown. In this figure, $D' = \{G'_1, G'_2, G'_3\}$ is transformed into a simple-graph collection $D = \{G_1, G_2, G_3\}$ by applying the first transfor-

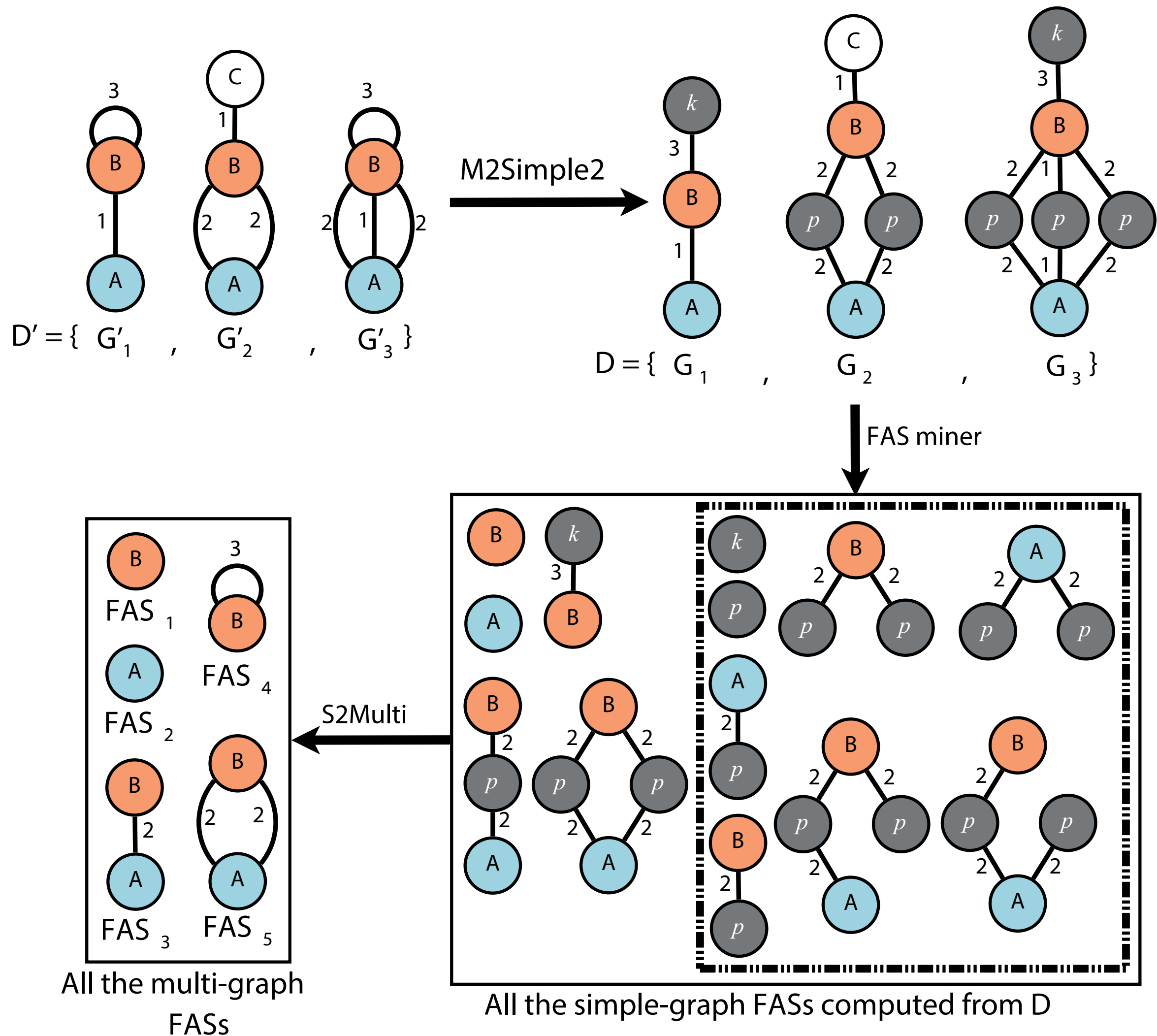


Figure 4.6: Example of FASs mined by using the proposed method for mining a subset of FASs from the multi-graph collection $D' = \{G'_1, G'_2, G'_3\}$ shown in Figure 4.4 using a support threshold $minsup = 2/3$.

mation process ($M2Simple2$) of our proposed method, then a FAS miner is applied over D. Later, the returnable FASs are transformed into multi-graphs.

By comparing this example with the one discussed in Section 4.1, which was illustrated in Figure 4.4, we can see that both proposed methods (onlyMulti and allEdges) for mining FASs from a multi-graph collection differ in the process for transforming a multi-graph into a simple-graph. Thus, different sets of FASs from the same multi-graph collection are obtained.

An example of this fact can be seen in Figures 4.4 and 4.6, where both onlyMulti and allEdges compute $\{FAS_1, \ldots, FAS_5\}$ as frequent approximate subgraphs. However, allEdges also mines frequent approximate subgraphs FAS_6 and FAS_7; while for onlyMulti these patterns are missed.

4.3 Experiments and Results

In this section, the performance of both allEdges and onlyMulti is evaluated, we cannot compare against state-of-the-art algorithms like VEAM, because they cannot process multi-graph collections. For carrying out our experiments we use synthetic multi-graph collections because this kind of collections have been commonly used for evaluating the performance of subgraph mining algorithms. These collections allow for studying the performance of the algorithms in controlled conditions; according to the number of graphs in the collections or the number of edges and vertices in the graphs. The used synthetic collections were randomly generated using the PyGen[1] graph emulation library. For building these collections we vary only one parameter at a time. First, we fix the size of the collection $|D| = 1000$ and the number of edges $|E| = 40$, varying the number of vertices $|V|$ from 10 to 50, with increments of 10. Next, we fix $|V| = 20$, maintaining $|D| = 1000$ and varying $|E|$ from 10 to 50, with increments of 10. Finally, we vary $|D|$ from 1000 to 5000, with increments of 1000, keeping $|V| = 20$ and $|E| = 40$. Notice that, we assign a descriptive name for each synthetic collection, for example, $D5kV20E40$ means that the collection has $|D| = 5000$, $|V| = 20$ and $|E| = 40$. Our experiments were carried out on a personal computer with an Intel(R) Core(TM) i5-3317U CPU @ 1.70 GHz with 4 GB of RAM. All the algorithms were implemented in ANSI-C and executed on Microsoft Windows 10.

In Figures 4.7, 4.8 and 4.9, the performance results obtained by the proposed transformation algorithms (allEdges and onlyMulti) over the previously described multi-graph collections are shown. These figures are split in three sub-figures: (a) the runtime, (b) the

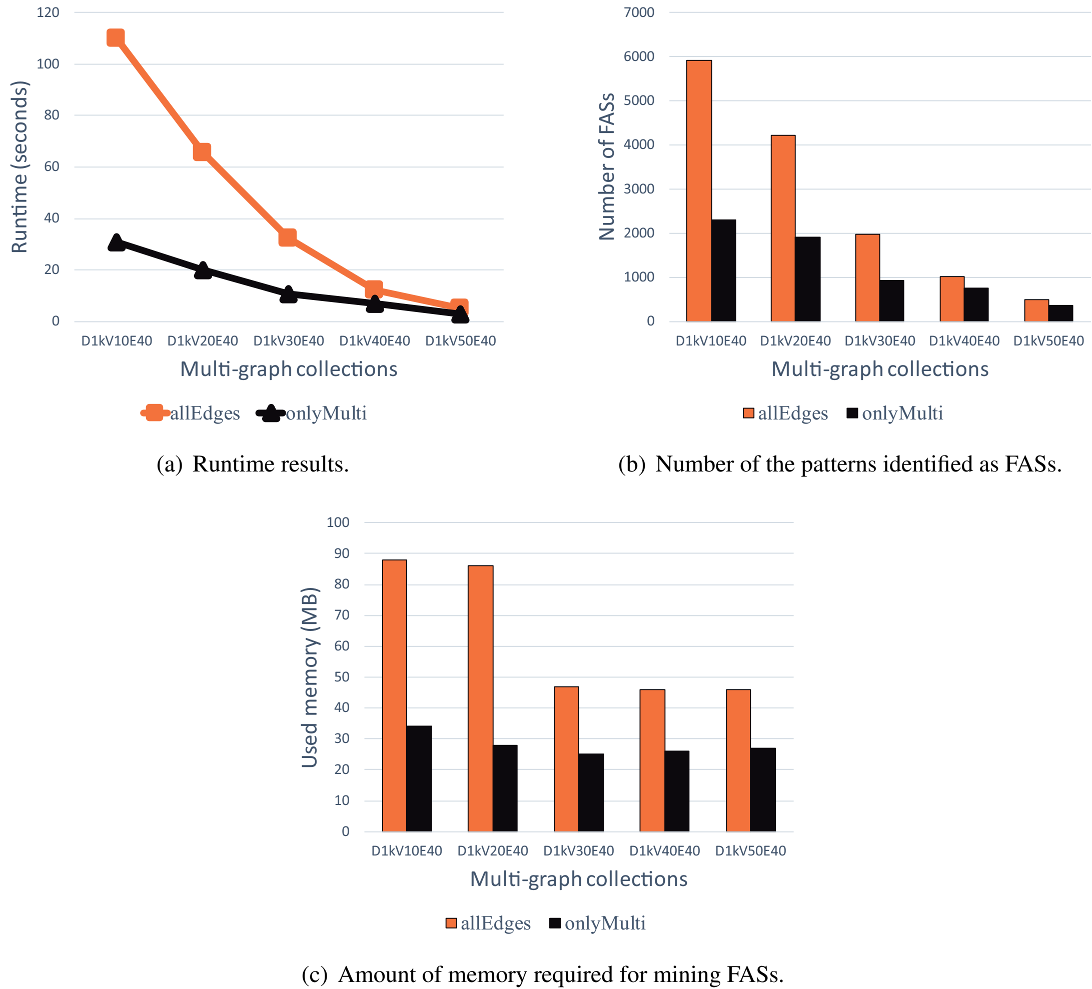
[1] PyGen is available in http://pywebgraph.sourceforge.net.

number of identified FASs and (c) memory required for mining the FASs. It is important to highlight that the FAS miner used by allEdges and onlyMulti in our experiments was VEAM (Acosta-Mendoza et al., 2012a). This is because as we mentioned in Chapter 3, VEAM is the only one algorithm that mines all the FASs in simple-graph collections, allowing variations in vertex and edge labels but keeping the graph structure. All the results reported in these figures were achieved with a similarity threshold $\tau = 0.55$ and a support threshold $minsup = 0.02$. This value for τ was selected because for values smaller than 0.55 almost all the mined subgraphs become similar to each other, while greater values for τ produce exact occurrences for most patterns (i.e., exact pattern mining). On the other hand, by using values for $minsup$ smaller than 0.02 makes almost all subgraphs become frequent, while greater values for $minsup$ produce only a few FASs.

As it can be seen from Figure 4.7, the runtime for both transformation methods decreases when the number of vertices $|V|$ grows (see Figure 4.7(a)). This improvement is achieved since for greater values of $|V|$, less dense multi-graphs are obtained and thus fewer subgraphs are identified as frequent (see Figure 4.7(b)). This decrement of the number of FASs implies that less memory is required for the mining process (see Figure 4.7(c)).

The performance of allEdges and onlyMulti over different multi-graph collections, where the number of vertices and multi-graphs are fixed but varying the number of edges, is illustrated in Figure 4.8. In this case, unlike in the previous set of experiments, as the number of edges grows more dense multi-graphs are obtained. Thus, more patterns are identified and increases in the runtime and the memory required by both methods for mining FASs recorded.

In Figure 4.9, the performance of the proposed methods over multi-graph collections which were obtained by varying the number of multi-graphs, but keeping the number of vertices and the number of edges, is illustrated. In this experiment, when the number of multi-graphs grows the number of FASs does not vary much, but the runtime and the memory required for storing the FAS information increases. This is because the number of multi-graphs to be processed increases.



(a) Runtime results.

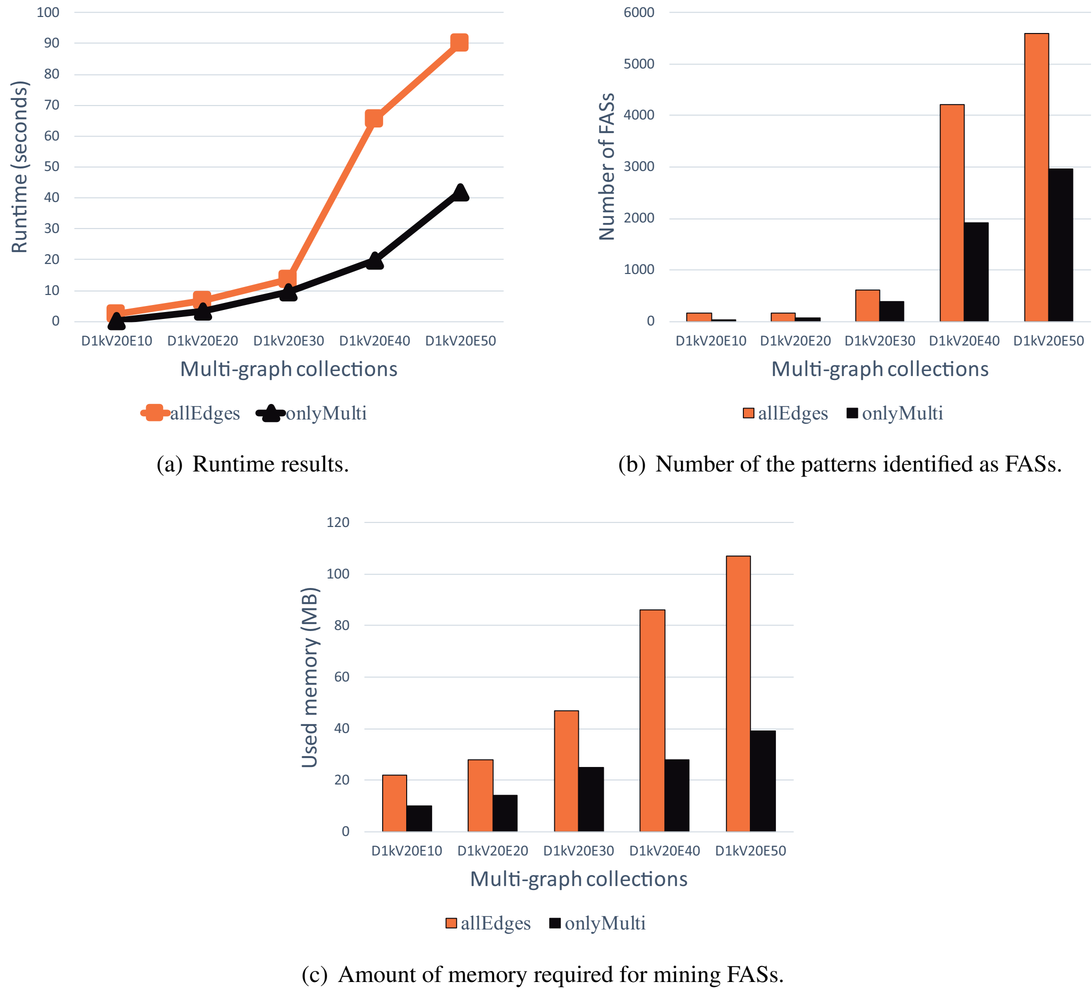
(b) Number of the patterns identified as FASs.

(c) Amount of memory required for mining FASs.

Figure 4.7: Performance of both allEdges and onlyMulti methods, using the similarity threshold $\tau = 0.55$ and the support threshold $minsup = 0.02$, over different synthetic multi-graph collections obtained varying the number of vertices from $|V| = 10$ to $|V| = 50$ with increments of 10 keeping the number of multi-graph $|D| = 1000$ and the number of edges $|E| = 40$.

According to the results shown in Figures 4.7, 4.8 and 4.9, on average, the onlyMulti method is 2 times faster than allEdges. This is because the graph transformation process of allEdges increases, more than in onlyMulti, the size of the graphs in the collections, which also increases the computational cost of the mining process. However, it is important to highlight that, in the worst case (i.e., when all edges of the collection are multi-edges and loops), the performance and results achieved by both onlyMulti and allEdges is the same.

It is important to highlight that these results were achieved by transforming each multi-



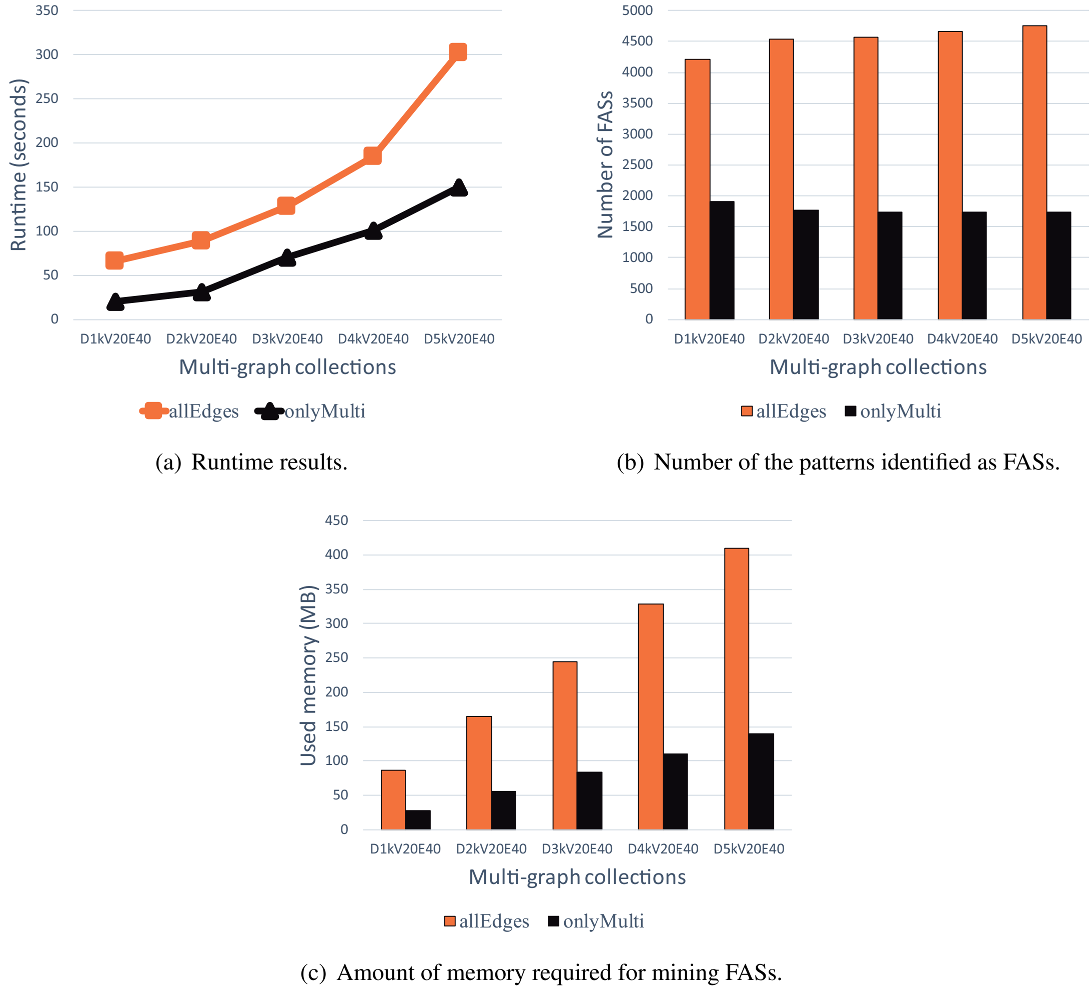
(a) Runtime results.

(b) Number of the patterns identified as FASs.

(c) Amount of memory required for mining FASs.

Figure 4.8: Performance of both allEdges and onlyMulti methods, using the similarity threshold $\tau = 0.55$ and the support threshold $minsup = 0.02$, over different synthetic multi-graph collections obtained varying the number of edges from $|E| = 10$ to $|E| = 50$ with increments of 10 keeping the number of multi-graph $|D| = 1000$ and the number of vertices $|V| = 20$.

graph collection into a simple-graph collection using $M2Simple1$ for allEdges and $M2Simple2$ for onlyMulti. Then, the FASs are mined by applying VEAM over the simple-graph collection, and each obtained FAS was transformed into a multi-graph using $S2Multi$. Thus, in Table 4.1, the performance results, in terms of runtime, and the average number of vertices and edges obtained by the proposed transformation processes ($M2Simple1$, $M2Simple2$ and $S2Multi$) are shown. Table 4.1 is split into three sub-tables according to the analyzed parameter of the collections. In these sub-tables, the first column shows the collection identifier.



(a) Runtime results.

(b) Number of the patterns identified as FASs.

(c) Amount of memory required for mining FASs.

Figure 4.9: Performance of both allEdges and onlyMulti methods, using the similarity threshold $\tau = 0.55$ and the support threshold $minsup = 0.02$, over different synthetic multi-graph collections obtained varying the number of multi-graphs from $|D| = 1000$ to $|D| = 5000$ with increments of 1000 keeping the number of multi-graph $|V| = 20$ and the number of vertices $|E| = 40$.

The other two consecutive blocks with four columns each one show the results obtained by applying the transformation method specified on the top. The first and fourth columns of each block show the runtime in seconds of the process for transforming a multi-graph collection into a simple-graph collection ($M2Simple1$ for allEdges and $M2Simple2$ for onlyMulti) and $S2Multi$ applied over the mined FASs, respectively. The other two columns of each block specify the average number of vertices and edges for each collection, after the transformation from multi-graphs to simple-graphs, respectively.

Table 4.1: Performance of the proposed transformation algorithms over different synthetic multi-graph collections.

(a) Varying $|V|$ from 10 to 50 with $|D| = 1000$ and $|E| = 40$.

Collection	**allEdges**				**onlyMulti**			
	$M2Simple1$	$\|V\|$	$\|E\|$	$S2Multi$	$M2Simple2$	$\|V\|$	$\|E\|$	$S2Multi$
D1kV10E40	0.081s	50	69	0.012s	0.060s	39	58	0.004s
D1kV20E40	0.082s	60	90	0.005s	0.059s	42	63	0.003s
D1kV30E40	0.081s	70	73	0.005s	0.039s	41	44	0.003s
D1kV40E40	0.082s	80	74	0.005s	0.036s	48	43	0.003s
D1kV50E40	0.080s	90	75	0.005s	0.035s	56	42	0.002s

(b) Varying $|E|$ from 10 to 50 with $|D| = 1000$ and $|V| = 20$.

Collection	**allEdges**				**onlyMulti**			
	$M2Simple1$	$\|V\|$	$\|E\|$	$S2Multi$	$M2Simple2$	$\|V\|$	$\|E\|$	$S2Multi$
D1kV20E10	0.015s	30	18	0.001s	0.005s	22	10	0.001s
D1kV20E20	0.026s	40	36	0.003s	0.018s	26	22	0.002s
D1kV20E30	0.050s	50	54	0.003s	0.026s	30	35	0.002s
D1kV20E40	0.082s	60	79	0.005s	0.059s	42	63	0.003s
D1kV20E50	0.088s	70	90	0.007s	0.063s	48	73	0.004s

(c) Varying $|D|$ from 1000 to 5000 with $|V| = 20$ and $|E| = 40$.

Collection	**allEdges**				**onlyMulti**			
	$M2Simple1$	$\|V\|$	$\|E\|$	$S2Multi$	$M2Simple2$	$\|V\|$	$\|E\|$	$S2Multi$
D1kV20E40	0.082s	60	79	0.005s	0.059s	42	63	0.003s
D2kV20E40	0.183s	60	79	0.004s	0.139s	42	63	0.003s
D3kV20E40	0.262s	60	79	0.004s	0.160s	42	63	0.003s
D4kV20E40	0.318s	60	79	0.004s	0.198s	42	63	0.003s
D5kV20E40	0.415s	60	79	0.004s	0.247	42	63	0.003s

According to the results presented in Table 4.1, the runtime of $M2Simple1$ and $M2Simple2$ grows with the increment of $|D|$ and $|E|$, while with the variations in $|V|$ it has not a significant increment since the transformation is done over the edges exclusively. As can be seen, $M2Simple1$ adds more vertices and edges than the $M2Simple2$ (see the third, fourth, seventh and eighth columns of Table 4.1), which allows for the mining of more FASs, then allEdges requires more time over the same collections than onlyMulti, for returning the mined FASs to multi-graphs.

4.4 Summary and Conclusions

In this chapter, two new methods for FAS mining in multi-graph collections, by transforming multi-graphs into simple-graphs and vice versa, have been proposed. The first step of these methods is to transform a multi-graph collection into a simple-graph collection, then over this simple-graph collection a traditional FASs miner is applied and finally the mined FASs are transformed into multi-graphs. The performance of the proposed transformation methods was evaluated over different synthetic multi-graph collections.

From our experiments, we can conclude that onlyMulti is able to mine FASs from multi-graph collections faster than allEdges, this was expected because onlyMulti only computes a subset of the FASs identified by the allEdges. This fact is important in order to reduce the cost of the FAS mining step. On the other hand, as we can see in our experiments, the higher the number of edges the higher runtime is required by both methods for mining FASs over multi-graph collections. This happens because the performance of the FAS miner is mainly affected by the number of FASs found. Besides, when the number of multi-graphs increases a higher runtime is required, since the methods must process a higher number of graphs.

CHAPTER 5

MINING PATTERNS DIRECTLY FROM MULTI-GRAPH COLLECTIONS

With the aim of mining all Frequent Approximate Subgraph (FAS) without graph transformations, in this chapter, we propose algorithms for mining FASs directly from multi-graph collections.

For comparing graphs, current frequent subgraph miners use isomorphism tests based on canonical forms[1] of graphs (Yan and Han, 2002; Kuramochi and Karypis, 2002; Huan et al., 2003; Yan and Han, 2003; Gago-Alonso et al., 2010b; Jia et al., 2009, 2011; Acosta-Mendoza et al., 2012a; Muñoz-Briseño et al., 2016; Alam et al., 2017). The canonical form based on adjacency matrices is the most used by FAS miners (Huan et al., 2003; Gago-Alonso et al., 2010b; Jia et al., 2009, 2011; Acosta-Mendoza et al., 2012a); however, this canonical form was designed for dealing with simple-graphs. Thus, in this chapter, we introduce an extension of this canonical form for dealing with multi-graphs. Then, based on this extended Canonical Adjacency Matrix (CAM), we propose a new algorithm, called MgVEAM, for directly mining FASs in multi-graph collections.

On the other hand, exact graph mining algorithms based on the Depth-First Search (DFS) canonical form have reported better performance than those algorithms based on CAM (Yan and Han, 2002; Gago-Alonso et al., 2010a; Gago-Alonso, 2010; Alam et al., 2017). This fact is because some prunings are introduced for reducing the search space of the mining process based on DFS trees, which can be included only when the DFS canonical form is used. However, like the CAM, the DFS canonical form was designed for working with simple-graphs.

[1] A canonical form is a unique representation for isomorphic graphs. Thus, two isomorphic graphs have the same canonical form.

Then, we also extend the DFS canonical form for representing isomorphic multi-graphs, and using this extension, we propose another algorithm, called AMgMiner, for directly mining FASs in multi-graph collections, which is faster than MgVEAM; however, MgVEAM requires less memory for mining all FASs.

5.1 Algorithm based on Canonical Adjacency Matrices

Many FAS miners use the canonical form based on Canonical Adjacency Matrices (CAM) (Huan et al., 2003; Jia et al., 2009; Gago-Alonso et al., 2010b; Jia et al., 2011; Acosta-Mendoza et al., 2012a). In this section, we introduce an algorithm, called *MgVEAM*, for directly mining FASs from multi-graph collections, based on the CAM code[2]. MgVEAM performs graph comparisons through CAM code comparisons, transforming graph isomorphism tests into string (code) comparisons. However, as the traditional CAM code was designed for working with simple-graphs, first we extend this code for representing isomorphic multi-graphs.

5.1.1 Canonical Adjacency Matrix for Multi-Graph Mining

The adjacency matrix introduced in this section allows us to represent multi-graphs, including the information of the corresponding multi-edges and loops into each cell. In this case, each cell of the adjacency matrix contains information of its corresponding edges, or vertices and loops (when the cell is on the diagonal). In definition 5.1, the adjacency matrix for a multi-graph is introduced.

Definition 5.1 (Adjacency matrix for a labeled multi-graph)**.** *Let* $G' = \{V_{G'}, E_{G'}, \phi_{G'}, I_{G'}, J_{G'}\}$ *be a labeled multi-graph, let* $v_i, v_j \in V_{G'}$ *two different vertices of* G', *let* $LL(v_i)$ *an ordered list of loop labels, where* $LL(v_i) = \langle J_{G'}(e_1), \ldots, J_{G'}(e_{|LL(v_i)|})\rangle$, *such that* $e_k \in E_{G'}, \phi_{G'}(e_k) = \{v_i\}$, *and let* $EL(v_i, v_j)$ *be an ordered list of edge labels, where* $EL(v_i, v_j) =$

[2] CAM code is a canonical string obtained from an adjacency matrix that uniquely represents the graph. This string, known as canonical code, unequivocally describes the label values and graph structure of all isomorphic graphs.

$\langle J_{G'}(e_1), J_{G'}(e_2), \ldots, J_{G'}(e_{|EL(v_i,v_j)|})\rangle$*, such that* $e_k \in E_{G'}, \phi_{G'}(e_k) = \{v_i, v_j\}$*. The adjacency matrix* $M = (m_{i,j})_{|V_{G'}|\times|V_{G'}|}$ *for* G' *is defined by:*

$$m_{i,j} = \begin{cases} (I_{G'}(v_i), LL(v_i)) & if\ i = j, LL(v_i) \neq \emptyset \\ I_{G'}(v_i) & if\ i = j, LL(v_i) = \emptyset \\ EL(v_i, v_j) & if\ i \neq j, EL(v_i, v_j) \neq \emptyset \\ - & otherwise \end{cases} \quad (5.1)$$

where the symbol "$-$" is used for representing edge absence for non-connected vertices.

Notice that, since the adjacency matrix of an undirected multi-graph is symmetric, only the upper or lower triangular adjacency matrices are needed for computing the CAM code of a multi-graph.

In Definition 5.1, the multi-graph matrix dimensions are the same as for the traditional (simple-graph) adjacency matrix dimensions (Cormen et al., 2001; Huan et al., 2003; Gago-Alonso et al., 2010b; Diestel, 2012), but loops and multi-edges information is included. Besides, in a multi-graph CAM, the cells of the matrix cannot be treated as simple labels anymore as in traditional adjacency matrices, but the cells must be treated as vertex labels and loop label lists (for the cells in the diagonal), and edge label lists.

Figure 5.1 illustrates an example of the adjacency matrices for a multi-graph G' according to Definition 5.1. In this figure, we can see that the order of the multi-edge and loop label lists into each cell of the matrix generates different adjacency matrices. To avoid this problem, we maintain these label lists sorted in descending order based on a labeled lexicographic order. Then, by performing all row permutations following a descendant lexicographic order, the CAM (see Figure 5.1(c)) of a multi-graph can be computed. Finally, the cells of the CAM are traversed, from the top to the bottom and from left to left, for building the CAM code by concatenating the values of the visited cells.

Due to the high computational complexity required for obtaining the CAM code of a

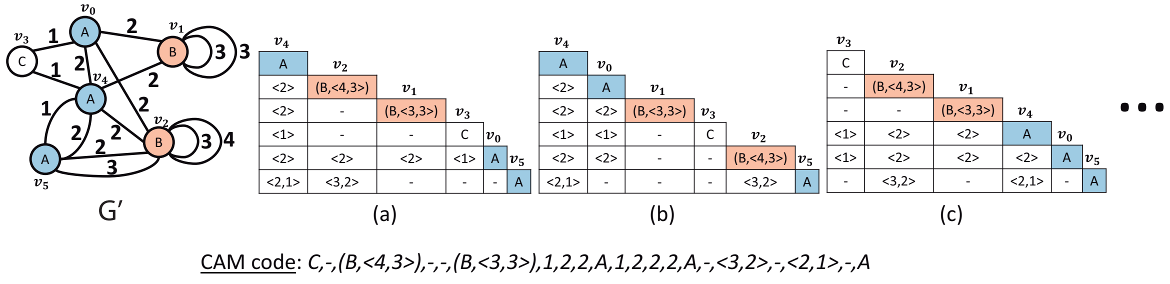


Figure 5.1: Example of a multi-graph G' and three of its adjacency matrices according to different vertex ordering; the CAM code of G' is obtained from the matrix (c).

multi-graph ($|V_{G'}|!$), this approach is not suitable for medium-large graphs (Read and Corneil, 1977; Kuramochi and Karypis, 2002; Diestel, 2012; Gago-Alonso et al., 2010b). Nevertheless, in practice, some concepts as *vertex invariants* (Read and Corneil, 1977; Kuramochi and Karypis, 2002), can be used for reducing the cost of computing the CAM code (Dinari and Naderi, 2016; Hlaing and Oo, 2016). Vertex invariants are properties useful for keeping the same vertex ordering in different isomorphism mappings. Vertex invariants can be used for partitioning the vertices of a graph into equivalence classes such that all the vertices assigned to the same partition have the same value for a vertex invariant. Vertex invariants are also known as isomorphism-invariant properties (Read and Corneil, 1977; Kuramochi and Karypis, 2002; Dinari and Naderi, 2016; Hlaing and Oo, 2016).

In our case, we use vertex degrees and labels as vertex invariants for computing the CAM of a multi-graph. First, the label list of each cell of the adjacency matrix is sorted following a descendant lexicographic order. Next, we assign each cell of the diagonal to a partition defined by the vertex labels. Then, we apply a second partitioning process, where we assign each element of a partition to a sub-partition taking into account the vertex degree. Later, we sort the partitions according to the descendant lexicographical order of vertex labels and, as a second criterion, in descendant order according to vertex degrees. In this way, following these two ordering criteria, we perform row permutations required for obtaining the CAM, as well as the CAM code, of a multi-graph.

An example for computing the CAM and the CAM code for a multi-graph G' is il-

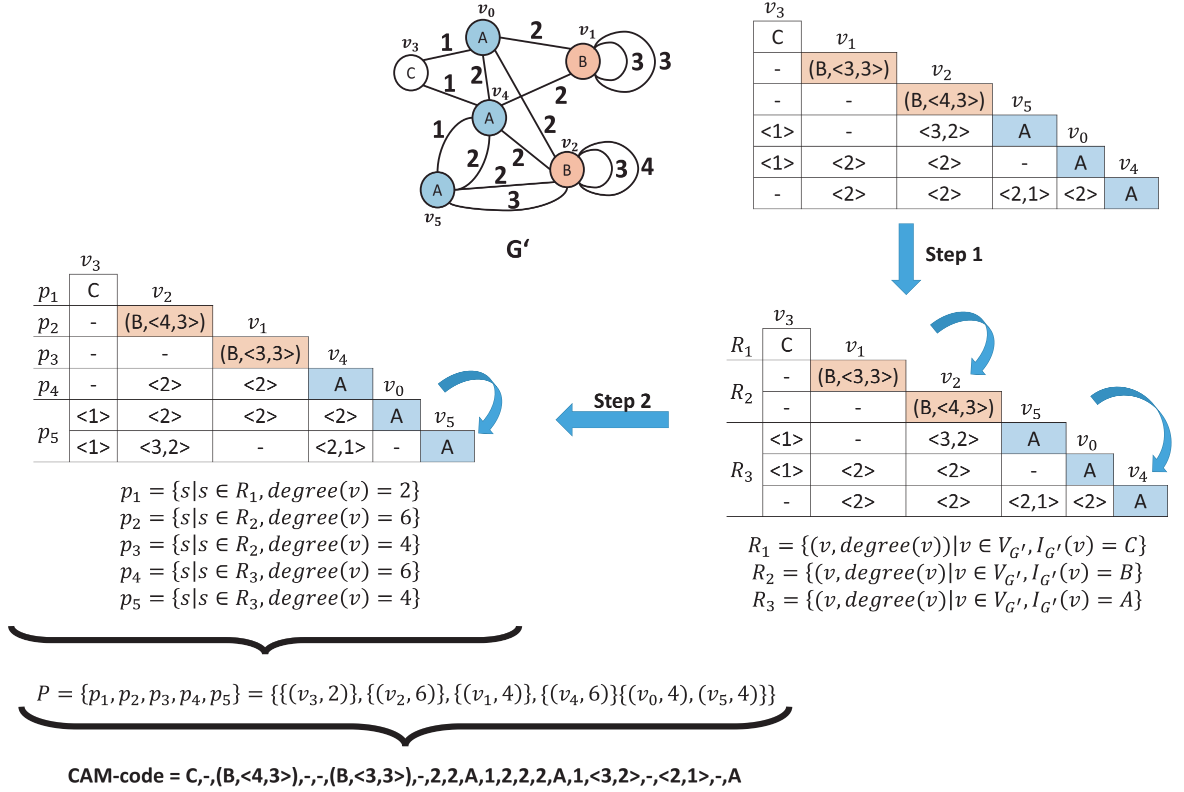


Figure 5.2: Example of the CAM code building process, following the idea proposed in (Kuramochi and Karypis, 2002), over an adjacency matrix of a multi-graph G'. In this example, the first adjacency matrix of G' is obtained by ascendantly ordering the vertices according to their labels. Then, through two steps, some cell permutations are performed for obtaining the CAM of G'. Finally, the CAM code is build concatenating the values of the matrix rows from top to the bottom and from left to right.

lustrated in Figure 5.2. Starting with an adjacency matrix of G', the first partitioning step (see step 1 of Figure 5.2) consists in sorting the rows of the matrix taking into account the vertex labels of G' following a descending order. In this step, the partitions R_1, R_2 and R_3 are obtained. Later, in the second step, the rows of each partition are reordered taking into account the loop labels (treating as an empty set the absence of loops), and the vertex degrees as a second criterion. In this way, we obtain a more specific partition set (i.e., the partitions p_1, p_2, p_3, p_4 and p_5 of Figure 5.2). The CAM code is computed concatenating the values of the matrix rows from top to bottom and left to right. Notice that, in this example, the diagonal of the CAM is partitioned according to the vertex labels and degrees resulting in the five partitions $\{C, 2\}$, $\{B, 6\}$, $\{B, 4\}$, $\{A, 6\}$ and $\{A, 4\}$ with $1, 1, 1, 1$ and 2 elements

respectively. In this case, the computational cost for computing the CAM of G' is reduced from $6! = 720$ permutations to only $2! = 2$ because given a partition set $P = \{P_1, \ldots, P_{|P|}\}$, the number of permutations required for computing the CAM is $\prod_{j=1}^{|P|} |P_j|!$, which is smaller than $|V_{G'}|!$ if $|P| \neq 1$. Finally, following the partition set P, the CAM code of G' is:

CAM-code = C,-,(B<4,3>),-,-,(B,<3,3>),1,2,2,A,1,2,2,2,A,-,<3,2>,-,<2,1>,-,A.

The algorithm for computing the CAM code, following the exposed ideas, is shown in Algorithm 5.1.

Algorithm 5.1: $ComputeCAM(G, e', CAM_G)$

Input: G : A candidate multi-subgraph, e' : The last edge added into G.
Output: CAM_G : The CAM code for the candidate multi-subgraph G.

1. $P \leftarrow$ *Partition list of* $v \in V_G$ *such that: for each* $P_k \in P$; $v_i, v_j \in P_k$, *if they have the same label and degree, with* $i \neq j$;
2. P is lexicographically sorted in a descending order;
3. **foreach** $P_k = \{p_1, \ldots, p_{|P_k|}\} \in P$ **do**
4. $l = |P_k|$;
5. l_k is the label for the clusters P_k;
6.   **while** $l \neq 1$ **do**
7. $newl = 1$;
8.   **foreach** $i = \{2, \ldots, |P_k|\}$ **do**
9. $X = (x_{(1,1)}, \ldots, x_{(|P_1|,1)}, \ldots, x_{(1,k)}, \ldots, x_{(i-1,k)})$ is computed based on Definition 5.1, being $x_{(j,w)}$ the descendant sorted multi-edge label list between the vertex in p_{i-1} and the vertex in $p_j \in P_w \in P$;
10. $Y = (y_{(1,1)}, \ldots, y_{(|P_1|,1)}, \ldots, y_{(1,k)}, \ldots, y_{(i-2,k)}, y_{(i,k)})$ is computed based on Definition 5.1, being $y_{(j,w)}$ the descendant sorted multi-edge label list between the vertex in p_i and the vertex in $p_j \in P_w \in P$;
11. Xl and Yl are the loop label list of vertices in p_{i-1} and p_i, respectively;
12.   **if** $X < Y$ ***or*** $(X = Y$ ***and*** $Xl < Yl)$, *following a lexicographical order* **then**
13.   `Swap`(p_{i-1}, p_i);
14. $newl = i$;
15. $l = newl$;
16. The adjacency matrix M of G is built sorting its cells following the lexicographical order of P;
17. The CAM code is obtained from M and it is stored into CAM_G;

5.1.2 The MgVEAM Algorithm

The algorithm for directly mining FASs from multi-graph collections proposed in this section (called MgVEAM: **M**ulti-**g**raph **V**ertex and **E**dge **A**pproximate **M**iner) is an extension of the VEAM algorithm (Acosta-Mendoza et al., 2012a,b) which follows a pattern growth approach for traversing the search space. It is important to highlight that, those FAS mining algorithms using this approach, have reported the best results, in terms of efficiency. For representing labeled multi-graphs, we will use CAM code applying the multi-graph adjacency matrices according to Definition 5.1. As we have explained in Section 5.1.1, this representation allows us to build the CAM code. Comparisons between CAM codes, which consist in string comparisons, are used for efficiently performing isomorphism tests during the mining process. Additionally, by using two substitution matrices, one for edge labels and another one for vertex labels, approximate matching for vertices and edges keeping the graph structure can be handled.

MgVEAM starts mining all frequent approximate single-vertex subgraphs; then, these subgraphs are recursively extended, following a DFS approach, by adding one edge at a time. Unlike VEAM, in this recursive process, all extensions of each FAS (i.e., candidate multi-graphs) are computed by first adding all the loops, then, all the simple-edges and multi-edges following the proposed descendant lexicographic order. For computing the CAM code of each candidate, MgVEAM follows the idea proposed in Section 5.1.1. MgVEAM performs isomorphism tests over each computed candidate, but this process is speeded-up by comparing the extended CAM codes. In this way, duplicate candidates are eliminated. Then, based on the downward closure property, only those FAS candidates are extended.

In Algorithm 5.2, MgVEAM starts computing all FASs corresponding to single-vertex graphs from a given multi-graph collection D'. Then, following a depth-first search approach, each frequent single-vertex is extended by adding an edge at a time through a recursive function, called *SearchMgVEAM* (see line 5 of Algorithm 5.2). When all single-vertex patterns have been extended, the set of all FASs in the collection D' is returned.

Algorithm 5.2: $MgVEAM(D', \tau, minsup, F)$

Input: D' : Multi-graph collection, τ : Similarity threshold, $minsup$: Support threshold.
Output: F : Frequent approximate subgraphs in D'.

```
1 C ←The frequent approximate single-vertex graph set in D' based on Definition 2.6;
2 F ← C;
3 foreach T ∈ C do
4     D'_T ←The occurrence set of T in D' based on Definition 3.3;
5     SearchMgVEAM(T, D'_T, D', τ, minsup, F);
```

The *SearchMgVEAM* function, given in Algorithm 5.3, recursively performs the extension of all frequent subgraphs. A candidate set for a given frequent subgraph is obtained by the *GenCandidateMgVEAM* function invoked in the line 1 of Algorithm 5.3. Then, only those candidates that satisfy the support constraint and which have not been identified in previous steps (i.e., their CAM codes are not in the set F) are stored and extended by performing recursive calls to the *SearchMgVEAM* function.

Algorithm 5.3: $SearchMgVEAM(G, D'_G, D', \tau, minsup, F)$

Input: G : FAS, D'_G : Occurrence set of the pattern G, D' : Multi-graph collection, τ : Similarity threshold, $minsup$: Support threshold.
Output: F : Frequent approximate subgraphs in D'.

```
1 C ← GenCandidateMgVEAM(G, D'_G, τ, ∅);
2 foreach T ∈ C do
3     if appSupp(T, D') ≥ minsup and T ∉ F then
4         F ← F ∪ {T};
5         D'_T ←The occurrence set of T in D' based on Definition 3.3;
6         SearchMgVEAM(T, D'_T, D', τ, minsup, F);
```

The aim of the *GenCandidateMgVEAM*, shown in Algorithm 5.4, is to compute all candidate extensions of a given frequent graph T. In this candidate generation, all extensions of T and its occurrences in the multi-graph collection D'_T are searched. Later, the CAM code for these subgraphs, which satisfy the similarity constraint using Definition 3.2, are computed by the *ComputeCAM* function (described in Section 5.1.1) invoked in line 3 of Algorithm 5.4. Finally, each pattern G, its corresponding CAM code and its similarity value are stored as an output candidate in C.

Algorithm 5.4: $GenCandidateMgVEAM(T, D'_T, \tau, C)$

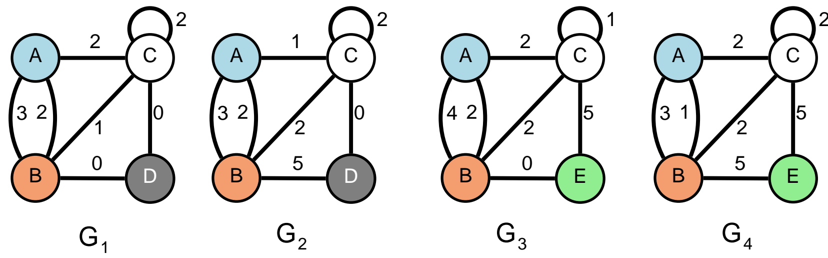
Input: T : FAS, D'_T : Occurrence set of the pattern T, τ : Similarity threshold.
Output: C : Pattern candidate set.

```
1 O ← {(G, T_e)|G is a child of T and T_e is an occurrence of G in G_k ∈ D'_T};
2 foreach (G, T_e) ∈ O do
3     ComputeCAM(G,e',CAM_G) ;                    // according to Algorithm 5.1
4     sim(G, T_e) is inserted into C using CAM_G as identifier;
```

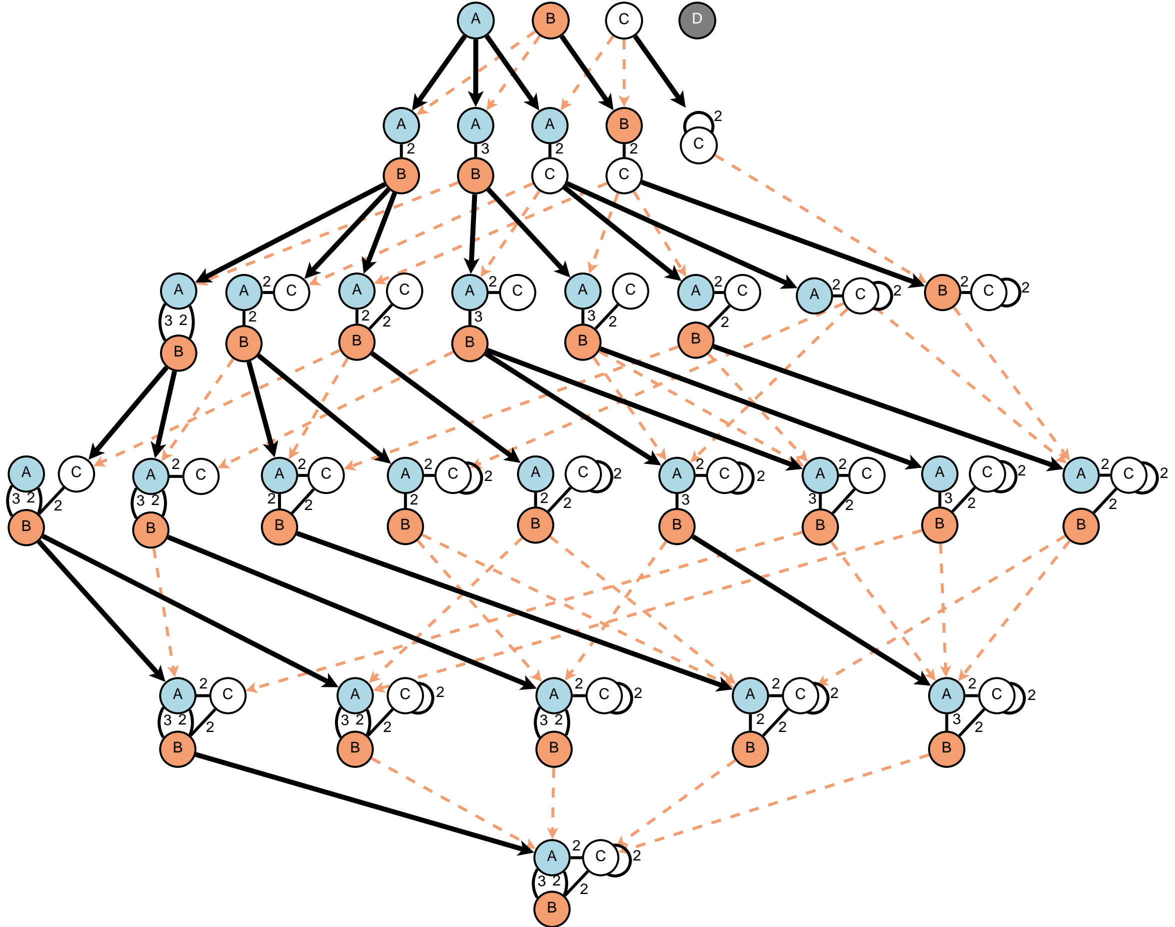
In Figure 5.3, we show an example of how MgVEAM traverses the search space for identifying all FASs on a multi-graph collection D'. In this example, when MgVEAM is applied on the multi-graph collection of Figure 5.3(a), the 32 subgraphs of Figure 5.3(b) are identified as FASs in D'. As it is shown in this figure, starting with the frequent approximate vertices, MgVEAM recursively extends them by an edge at time; traversing the search space until the frequency threshold is not fulfilled. After each extension, there could be two cases: (1) identifying a new FAS (continuous dark arrows), and (2) identifying a FAS which was previously found (dashed arrows), and therefore it does not need to be extended again. In this way, MgVEAM finds 39 duplicate candidates (the number of dashed arrows).

Since the exact complexity of MgVEAM depends not only on the size of the multi-graph collection or the average size of the multi-graphs, but also on the internal distribution of edges and the similarities among subgraphs, we analyze the computational complexity of MgVEAM for the worst case, where each multi-graph G_i in the collection D' has the same size; i.e., each multi-graph has n vertices and m edges, every $G_i \in D'$ is completely connected, each vertex label can be replaced by any other vertex label, and each edge label can be substituted by any other edge label, where l_V and l_E are the number of vertex and edge labels respectively.

The complexity of MgVEAM is analyzed separately for the algorithms above described. First, MgVEAM (Algorithm 5.2) traverses all vertices of the collection D' for finding those frequent single-vertex graphs, which is $O(n)$ for each multi-graph of D'. Then, all FASs are extended following a DFS strategy; obtaining the candidate set by calling *GenCandidateMgVEAM* (see Algorithm 5.4). The process, for obtaining the candidate set, extends a



(a) A multi-graph collection $D' = \{G_1, G_2, G_3, G_4\}$.



(b) Traversing of the search space by MgVEAM. In this example, the continuous dark arrows show the path followed by MgVEAM and the dashed arrows show the identified duplicate candidates, which are not extended again.

Figure 5.3: Example of the mining process of MgVEAM on a multi-graph collection $D' = \{G_1, G_2, G_3, G_4\}$; supposing that $minsup = 3/4$, $\tau = 0.55$ and the labels D, 2 and 3 can substitute the labels E, 1 and 4, respectively, where $L_V = \{A, B, C, D, E\}$ and $L_E = \{0, 1, 2, 3, 4, 5\}$.

pattern by an edge taking into account all possible vertex and edge labels, which is, in the worst case, $O(ml_V l_E)$. Next, the CAM code of each candidate is computed by calling the

$ComputeCAM$ function, which in the worst case is $O(n!)$ because it performs all permutations of the vertices of a candidate FASs for computing its canonical form. Thus, the complexity of the $GenCandidateMgVEAM$ function is $O(ml_V l_E n!)$. The $SearchMgVEAM$ function (see Algorithm 5.3) traverses all the edges that allow growing a FAS to a new candidate pattern recursively adding an edge at each time (without taking into account those previously added edges) and just those candidates fulfilling the similarity threshold τ are stored and extended; thus, at most, m extensions are made for the first extension of the pattern, $m-1$ for the second one, $m-2$ for the third one, and so on, resulting $m!$ extensions in this recursive function. Then, considering the complexity of the $GenCandidateMgVEAM$ function, the comparisons performed for growing the patterns, and the number of times that $SearchMgVEAM$ calls itself, we can conclude that the complexity of the $SearchMgVEAM$ function is $O(m!(ml_V l_E n!))$. Therefore, since all the analyzed steps are carried out for each frequent single-vertex over each multi-graph of D', the complexity of these steps into MgVEAM is $O(dn) + O(dn(m!ml_V l_E n!))$, where d is the number of multi-graphs of D', n and m are the number of vertices and edges, respectively, and, l_V and l_E are the number of vertex and edge labels, respectively. Thus, the complexity of the MgVEAM algorithm, in the worst case, is $O(dmm!l_V l_E nn!)$.

5.2 Algorithm based on Depth-First Search canonical forms

In the exact context, the most efficient mining algorithms reported are based on the Depth-First Search (DFS) canonical form (Yan and Han, 2002, 2003; Zhu et al., 2007; Gago-Alonso et al., 2010a; Alam et al., 2017). However, the DFS canonical form was designed for dealing with simple-graphs. For this reason, in this section, we propose an extension of the DFS canonical form for dealing with multi-graphs, and using the extended DFS canonical form, we introduce a FAS mining algorithm, called *AMgMiner*, for directly mining all FASs from multi-graph collections.

5.2.1 Depth-First Search Canonical Form for Multi-Graph Mining

Depth-First Search (DFS) can be used for traversing a graph, representing it as a DFS tree[3]. This strategy has been applied in the simple-graph mining context, improving the efficiency of mining algorithms (Yan and Han, 2002, 2003; Zhu et al., 2007; Gago-Alonso et al., 2010a; Alam et al., 2017). Thus, following the idea proposed in (Yan and Han, 2003), we develop an extension of the DFS canonical form for working with multi-graphs.

The traditional DFS is constructed as follows: starting from a vertex chosen at random, following a single route from the current vertex, we visit all the adjacent edges until a full traverse is formed. Then, taking into account this vertex visiting order, a tree (DFS tree) can be built. For constructing this DFS tree, each new edge, which extends the previous DFS tree, is known as a forward or a backward extension (edge) into the DFS tree. An edge is a forward extension if it introduces a new vertex into the DFS tree, otherwise it is a backward extension. This is formalized in Definition 5.2.

Definition 5.2 (Forward, backward extensions and child)**.** *Let G_1 and G_2 be two labeled multi-graphs, where G_1 is a subgraph of G_2, the edge $e \in E_{G_2}$ is an extension of G_1, denoted as $G_2 = G_1 \diamond e$, if $E_{G_2} = E_{G_1} \cup \{e\}$, $V_{G_1} \cap \phi_{G_2}(e) \neq \emptyset$, $V_{G_2} = V_{G_1} \cup \phi_{G_2}(e)$. The edge e is a backward extension if $\phi_{G_2}(e) \subseteq V_{G_1}$, otherwise e is a forward extension (it extends the vertex set of G_1). If e is an extension of G_1 and $G_2 = G_1 \diamond e$, then we will refer to G_2 as a child of G_1.*

Since in a multi-graph there are several ways for traversing a graph according to a vertex visiting order, a multi-graph can produce different DFS trees. An example of this fact can be seen in Figure 5.4, where four DFS trees for a given multi-graph G' are shown.

The starting vertex v_0 in a DFS tree is known as the root, and the last visited vertex v_n is known as the *rightmost vertex* (Yan and Han, 2002, 2003). The vertices of G' are visited from v_0 to v_n for building its corresponding DFS tree. The dark edges in Figures 5.4(a)-(d) show four DFS trees for G'. Notice that, in these figures, all edges in the DFS trees of G'

[3] A DFS-tree is a tree generated by a DFS traverse.

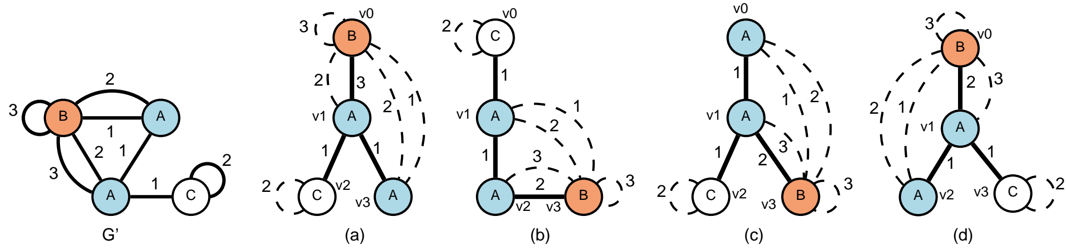


Figure 5.4: Examples of some DFS trees for a multi-graph G' by using different root vertices.

(dark edges) are *forward extensions* according to the visited order. The *backward extensions* (dotted edges) represent all edges that are not in the DFS trees, loops are treated as a special case of backward edges. Commonly, edges are treated as an ordered pair (v_i, v_j) according to the DFS traverse order. Thus, an edge $e \in E_{G'}$, with $\phi_T(e) = \{v_i, v_j\}$, being i and j vertex indexes according to the visiting order of T, is a forward edge for a DFS tree T of G' if $i < j$, and it is a backward edge if $i > j$, or $\phi_T(e) = \{v_i\}$ (i.e., e is a loop).

A *rightmost path* of a given multi-graph is defined as the path from v_0 to v_n. The rightmost path is used for efficiently extending a DFS tree (Yan and Han, 2002, 2003; Gago-Alonso et al., 2010a). In our example, (v_0, v_1, v_3) is the rightmost path in Figures 5.4(a), (c) and (d), while (v_0, v_1, v_2, v_3) is the rightmost path of Figure 5.4(b).

Based on DFS, a FAS can be extended from every possible vertex, generating a large number of candidate patterns, but several of these candidates are duplicated. A duplicate graph is a pattern which was founded in previous steps (i.e., when a previous FAS was extended), but it is obtained again when the current FAS is extended. Thus, a better way for extending candidate FASs, known as *rightmost extension*, was proposed by Yan and Han (2002, 2003). This extension method consists in: given a DFS tree T for a FAS, a new edge can be added if it is a backward extension (see Definition 5.2) for the rightmost vertex v_n, connecting v_n to any vertex on the rightmost path; or if it is a forward extension (see Definition 5.2) for any vertex $v_i, i \leq n$ in the rightmost path, connecting v_i to a new vertex $v \notin T$. An example of this process is shown in Figure 5.5, where the rightmost extensions of a DFS tree T are 5.5(a)-(h) (the dark vertices represent the rightmost path). As we can

see, in Figures 5.5(a)-(e), the rightmost vertex of T is extended. Notice that the rightmost extension restricts the extensions of a given DFS tree, but cannot restricts the possible order for traversing a graph; then, the number of generated duplicate graphs is reduced but they are not eliminated.

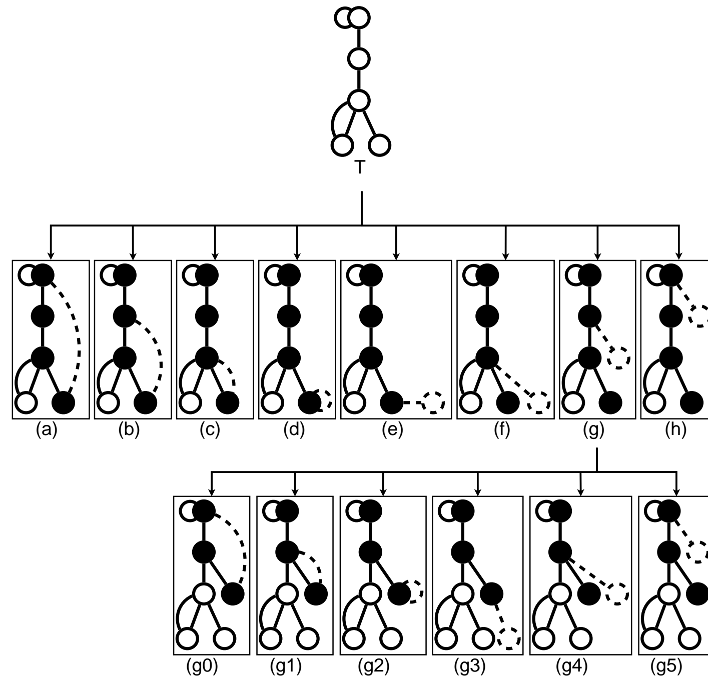


Figure 5.5: Example of extensions of a FAS T based on the rightmost path. In this example, the FAS T and its child (g) only can be extended by an edge from one of the dark vertices, which are the rightmost path extensions.

For dealing with multi-graphs, loops are included as a special case of backward extensions, this makes an important difference regarding the canonical form proposed in (Yan and Han, 2002, 2003). Taking loops into account implies that the order for building DFS trees is different to the way proposed in (Yan and Han, 2002, 2003). For building a DFS tree, our proposal selects the first vertex as root for the DFS tree according to a label lexicographic order. Next, all backward extensions of this vertex are added. After, a forward extension is added. Then, adding recursively all possible non-visited backward extensions before adding a new non-visited forward extension. However, since a vertex can have several backward and forward extensions for multi-graphs, an order according to the graph structure is defined as

follows.

Definition 5.3 (Linear order according to a DFS tree). *Let T be a DFS tree of a graph and let $e_1, e_2 \in E_T$ be two edges, where $\phi_T(e_1) = \{v_{i_1}, v_{j_1}\}$ and $\phi_T(e_2) = \{v_{i_2}, v_{j_2}\}$. The linear order, $\prec_T$, defined as: $e_1 \prec_T e_2$ holds iff one of the following statements is true:*

- *e_1 and e_2, are forward edges and $j_1 < j_2$ or $j_1 = j_2 \wedge i_1 > i_2$.*
- *e_1 and e_2, are backward edges and $i_1 < i_2$ or $i_1 = i_2 \wedge j_1 < j_2$.*
- *e_1 is a backward edge, e_2 is a forward edge and $i_1 \leq j_2$.*
- *e_1 is a forward edge, e_2 is a backward edge and $j_1 \leq i_2$.*

Using the linear order of Definition 5.3, the number of DFS trees for a multi-graph is reduced; however, it does not guarantee finding the canonical DFS tree of a multi-graph, because several DFS trees can be identified for the same multi-graph.

A DFS tree can be expressed as a sequence of edges, which is known as DFS code. In this case, each edge $e \in E_{G'}$ is represented by the 5-tuple $e = (i, j, l_i, l_e, l_j)$, where $\phi_{G'}(e) = \{v_i, v_j\}$ (or $\phi_{G'}(e) = \{v_i\}$ for a loop where $j = i$), $l_i = I_{G'}(v_i)$, $l_e = J_{G'}(e)$ and $l_j = I_{G'}(v_j)$. Then, a sequence of edge tuples, following a route of the DFS tree, is a DFS code of a multi-graph. An example of DFS codes is presented in Table 5.1, where the DFS codes of the multi-graph G' of Figure 5.4 are illustrated. In this table, the DFS codes (a)-(d) correspond to the DFS trees of Figure 5.4(a)-(d), respectively.

Table 5.1: DFS codes obtained from the DFS trees shown in Figures 5.4(a)-(d).

Id	**DFS code**
(a)	(0,0,B,3,B)(0,1,B,2,A)(1,0,A,3,B)(1,2,A,1,C)(2,2,C,2,C)(1,3,A,1,A)(3,0,A,2,B)(3,0,A,1,B)
(b)	(0,0,C,2,C)(0,1,C,1,A)(1,2,A,1,A)(2,3,A,2,B)(3,1,B,1,A)(3,1,B,2,A)(3,2,B,3,A)(3,3,B,3,B)
(c)	(0,1,A,1,A)(1,2,A,1,C)(2,2,C,2,C)(1,3,A,2,B)(3,0,B,1,A)(3,0,B,2,A)(3,1,B,3,A)(3,3,B,3,B)
(d)	(0,0,B,3,B)(0,1,B,2,A)(1,0,A,3,B)(1,2,A,1,A)(2,0,A,1,B)(2,0,A,2,B)(1,3,A,1,C)(3,3,C,2,C)

In order to represent isomorphic multi-graphs in an unequivocal way, we propose a DFS code based on the minimum DFS order. This DFS code is similar to the one proposed

in (Yan and Han, 2002) but, in our case, information about loops and multi-edges is included. Therefore, with the aim of identifying the canonical DFS code (minimum DFS code) from all possible DFS codes of a given multi-graph, a lexicographic order among DFS codes is needed. In this order, vertex and edge labels are used for removing ambiguities from two edges with the same vertex index order. The lexicographic order is defined as follows.

Definition 5.4 (DFS lexicographic order)**.** *Let Z be the set of all possible DFS codes of a labeled multi-graph G', and let $C_1 = \langle a_0 \dots a_m \rangle$ and $C_2 = \langle b_0 \dots b_n \rangle$ be two DFS codes in G', $C_1 \leq C_2$ iff one of the following conditions is true.*

- $\exists t, 0 \leq t \leq min(m,n), a_k = b_k$ *for all* $k < t$*, and* $a_t \prec_T b_t$*.*
- $a_k = b_k$ *for all* $0 \leq k \leq m$*, and* $m \leq n$*.*

Where the relation $a_t \leq b_t$ takes into account a label lexicographic order and a linear order in the vertex and edge indexes (see Definition 5.3).

By using Definition 5.4, the *minimum DFS code* of a given multi-graph can be obtained. This minimum DFS code can be used as a canonical form for all isomorphic multi-graphs. Supposing that the DFS codes presented in Table 5.1 are all possible codes for the multi-graph G' of Figure 5.4, when we apply Definition 5.4 over these codes, the DFS code (d), which corresponds to Figure 5.4(d), is the canonical DFS code (minimum DFS code according to the label lexicographic order), for all multi-graphs that are isomorphic to G'.

5.2.2 The AMgMiner Algorithm

In this section, we introduce a new algorithm for directly mining frequent subgraphs (allowing approximate matching) from multi-graph collections (called AMgMiner: **A**pproximate **M**ulti-graph **Miner**), which is based on the extended DFS canonical form introduced in Section 5.2.1. In AMgMiner, for traversing the search space, a pattern growth approach is followed, and by using the extended DFS code, isomorphism tests can be efficiently performed.

AMgMiner performs approximate matching between multi-graphs by means of the similarity function (see Definition 2.5), and likewise MgVEAM, we use two substitution matrices for handling approximate matching for vertices and edges keeping the graph structure.

AMgMiner starts computing the frequent approximate single-vertex and single-edge subgraphs; next, the single-edges subgraphs are recursively extended by adding an edge at each time. AMgMiner, following a DFS approach, first adds all the loops, and after all the simple-edges and multi-edges, taking into account the proposed lexicographic order. Then, following the idea proposed in Section 5.2.1, the extended canonical DFS code is computed. By using this canonical code, the isomorphism tests (which is an NP-Hard problem) are transformed into DFS code comparisons, where isomorphic multi-graphs have the same canonical code. In this way, these comparisons are used for speeding up isomorphism tests between multi-graph candidates. AMgMiner is based on downward closure property, therefore, recursive extensions of a FAS G is performed while the support threshold is fulfilled and G is in canonical form. By extending those patterns that are in canonical form allows considerably reducing the generation of duplicate candidates, and the unnecessary isomorphism tests among them, allowing speeding up the mining process.

AMgMiner is detailed in Algorithm 5.5. Once AMgMiner has computed the frequent approximate single-vertex and single-edge subgraphs from a given multi-graph collection D' (see lines 1 and 2), each frequent approximate single-edge graph is recursively extended by using the function *SearchAMgMiner* (see lines $4-6$).

Algorithm 5.5: $AMgMiner(D', \tau, minsup, F)$

Input: D' : Multi-graph collection, τ : Similarity threshold, $minsup$: Support threshold.
Output: F : Frequent approximate subgraphs in D'.

```
1  F ←Frequent approximate single-vertex graph set in D' ;   // based on Definition 2.6
2  C ←Frequent approximate single-edge graph set in D' ;     // based on Definition 2.6
3  F ← F ∪ C;
4  foreach T ∈ C do
5  |   D'_T ←Occurrence set of T in D' ;                     // based on Definition 3.3
6  |_  SearchAMgMiner(T, DFScode(T), D'_T, D', τ, minsup, F);
```

Algorithm 5.6: $SearchAMgMiner(G, c, D'_G, D', \tau, minsup, F)$

Input: G : FAS, c : DFS code of G, D'_G : Occurrence set of the pattern G, D' : Multi-graph collection, τ : Similarity threshold, $minsup$: Support threshold.
Output: F : Frequent approximate subgraphs in D'.

```
1 C ← GenCandidateAMgMiner(c, D'_c, D', τ);
2 foreach T ∈ C do
3     if appSupp(T, D') ≥ minsup and T ∉ F then
4         F ← F ∪ {T};
5         D'_T ←The occurrence set of T in D' ;          // based on Definition 3.3
6         SearchAMgMiner(T, DFScode(T), D'_T, D', τ, minsup, F);
```

In Algorithm 5.6, the recursive function for extending FASs is shown. For a given FAS, a candidate set is identified through the function *GenCandidateAMgMiner* (see line 1 of Algorithm 5.6). Then, only the frequent candidates which were not identified in previous steps are stored into the output FAS set and the *SearchAMgMiner* algorithm is recursively applied over them (see lines $3-6$).

Algorithm 5.7 extends a FAS with DFS code c by adding a single-edge if and only if c is in canonical form (see line 1). This pruning was proposed in (Yan and Han, 2002) for processing only those patterns that have been not obtained in previous steps, because when a pattern is not in canonical form means that this pattern is a duplicate candidate. In this way, the number of duplicate candidates is reduced. Also, for reducing the number of candidate FASs and for increasing the efficiency of our algorithm, for each FAS, the rightmost (backward and forward) extensions are computed only over the rightmost path following the idea discussed in Section 5.2.1. Later, for each rightmost extension, the corresponding FAS set (those patterns fulfilling the similarity threshold τ) is computed, identifying those candidates that are approximate to the current pattern with a similarity greater than or equal to τ (see lines $6-8$ of Algorithm 5.7). For efficiently accessing to those patterns, both frequency and occurrences of each pattern are stored into a hash table using the DFS code of the pattern as key (see the line 8). Next, with the aim of first analyzing the backward extensions (according to Section 5.2.1), the new candidate patterns are sorted based on the lexicographical order according to Definition 5.4 (see line 9 of Algorithm 5.7).

Algorithm 5.7: $GenCandidateAMgMiner(c, D'_c, D', \tau, C)$

Input: c : DFS code, D'_c : Occurrence set of the pattern with DFS code c, D' : Multi-graph collection, τ : Similarity threshold, $minsup$: Support threshold.
Output: C : Pattern candidate set.

```
1 if isMin(c) = true then
2     H ← An empty hash table;
3     C ← ∅;
4     RE ← Rightmost extensions of each occurrence in D'_c;
5     foreach G_r ∈ RE do
6         foreach G, such that Θ_(f,g)(G, G_r) ≥ τ do
7             C ← C ∪ {The DFS code of G};
8             Θ_(f,g)(G, G_r) and G_r are inserted into H[{The DFS code of G}];
9     C is sorted based on the lexicographical order ;   // according to Definition 5.4
```

In order to avoid as many duplicate candidates as possible, as it is explained in Section 5.2.1, we apply the pruning of non-minimum DFS codes, because, as was demonstrated in (Yan and Han, 2002), a non-minimum DFS code implies that the pattern is a duplicate candidate. Then, as it can be seen in line 1 of Algorithm 5.7, the Boolean function *isMin* is called for determining if the given DFS code is minimum or not. This is performed using the DFS canonical form for multi-graphs introduced in Section 5.2.1. We determine (through the *isMin* function) if there is a DFS code smaller than the current DFS code, according to Definition 5.4. In such case, the current one is non-canonical and the candidate extension process is stopped. The cost of checking if a code is non-minimum is less than computing the whole canonical DFS code, because in this last case, exhaustive edge permutations are required.

The function $isMin$ is shown in Algorithm 5.8, which is based on the *compare* function shown in Algorithm 5.9. The main loop of isMin performs the pruning of non-minimum DFS codes (see lines $3-10$); thus, each edge is visited for building DFS code tuples and each new tuple is compared with the corresponding one in c (see lines $6-9$). Each tuple contains the edge information of the proposed extension of the DFS code, as we described in Section 5.2.1. This comparison is performed by calling the recursive function *compare* (see line 8) until the smallest DFS code is found or all the edges have been compared.

Algorithm 5.8: *isMin(c)*

Input: c : DFS code.
Output: $result$: true if c is the canonical DFS code; otherwise false.

```
1  result ← true;
2  V ← the veritices of c;
3  foreach v ∈ V and result = true do
4  |  IE ← incident edge set of v sorted ;           // according to Definition 5.3
5  |  V ← V \ {v};
6  |  foreach e ∈ IE and result = true do
7  |  |  e and its vertices are marked as visited;
8  |  |  result ← compare(c,0,e,V);
9  |  |  e and its vertices are marked as non-visited;
10 |  V ← V ∪ {v};
```

Algorithm 5.9: *compare(c,index,e,V)*

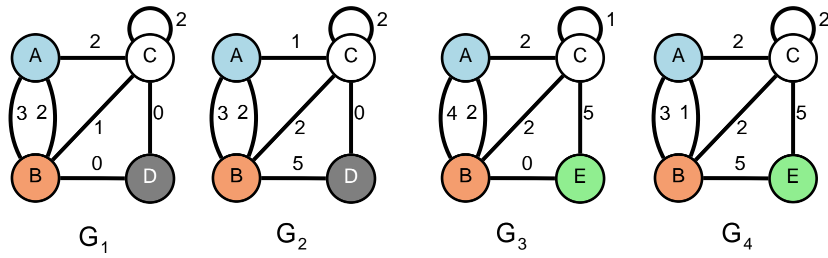
Input: c : DFS code, $index$: Index for c to be analyzed, e : Edge, V : Vertex set of c.
Output: $result$: false if there is a code less than c; otherwise true.

```
1  k ← the index corresponding to the current vertex into the tuple to be built;
2  if e is a loop then dfs ← (k, k, l_v, l_e, l_v); // where v is the vertex of e
3  else dfs ← (k, id(w), l_v, l_e, l_w); // where v and w are the vertices of e
4  result ← true;
5  if dfs < c[index] according to Definition 5.4 then
6  |  result ← false;
7  else if dfs = c[index] according to Definition 5.4 then
8  |  index ← index + 1;
9  |  foreach v ∈ V and result = true do
10 |  |  IE ← non-visited incident edge set of v sorted according to Definition 5.3;
11 |  |  V ← V \ {v};
12 |  |  foreach e ∈ IE and result = true do
13 |  |  |  e and its vertices are marked as visited;
14 |  |  |  result ← compare(c,index,e,V);
15 |  |  |  e and its vertices are marked as non-visited;
16 |  |  V ← V ∪ {v};
```

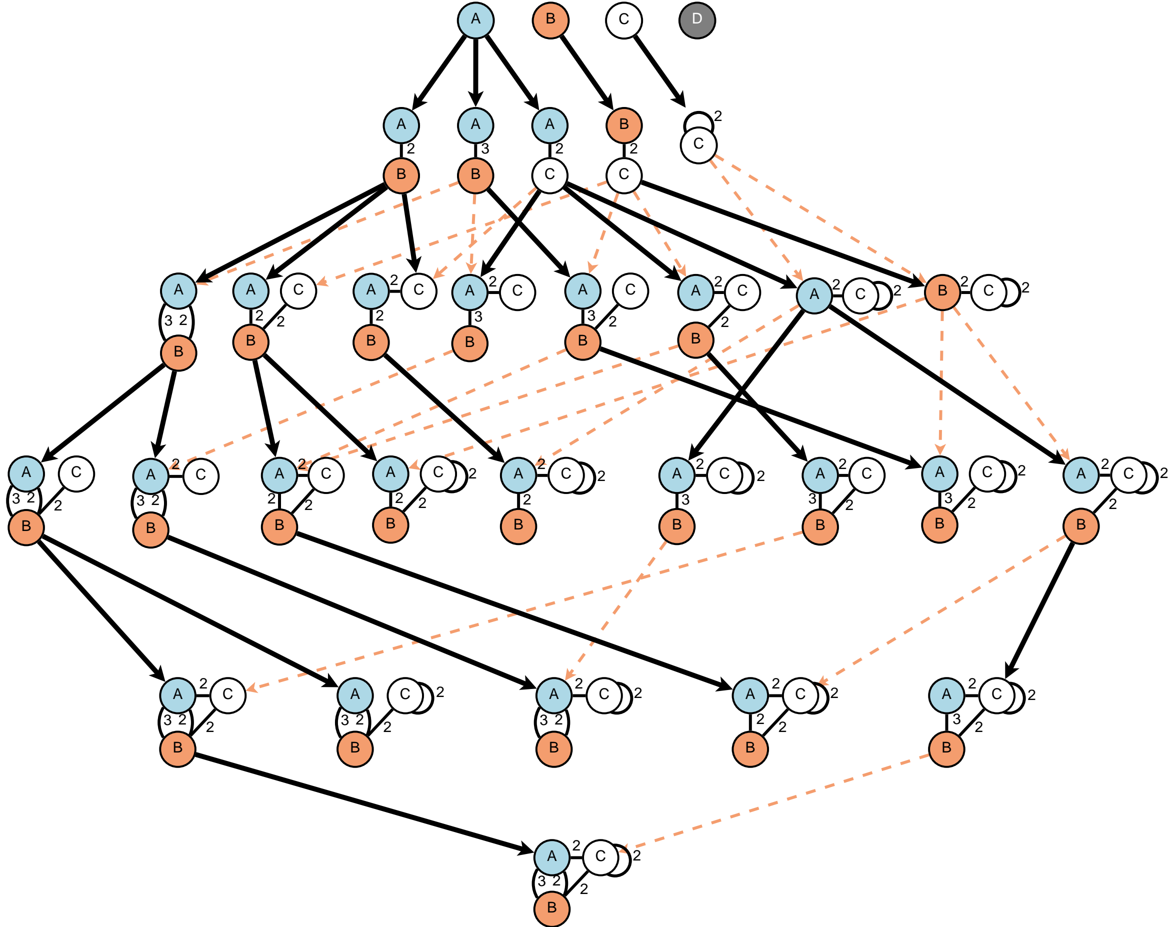
In Algorithm 5.9 (*compare*), the recursive verification of non-minimum DFS codes is performed. A comparison between tuples of two DFS codes is the key of this function. We build a tuple, taking into account the ordering introduced in Section 5.2.1, using the last visited edge of the input DFS code c. This tuple is built using the information of the loops, simple-edges and multi-edges as described in Section 5.2.1 (see lines $2-3$). Then, the new tuple is compared with the tuple corresponding to the last visited edge of c (see lines 5 and 7). If these two tuples are equal, then the next tuple is built and the function compare is recursively called (see lines $7-16$).

In Figure 5.6, we show an example of how AMgMiner traverses the search space for identifying all FASs on a multi-graph collection D'. In this example, when MgMiner is applied over D', the 32 subgraphs of Figure 5.6(b) are identified as FASs. As it is illustrated in this figure, starting with the frequent approximate vertices, AMgMiner recursively extends them by an edge at time; traversing the search space until the frequency threshold is not fulfilled. After each extension, there could be two cases: (1) identifying a new FAS (continuous dark arrows), and (2) identifying a FAS which was previously found (dashed arrows), and therefore it does not need to be extended again. In this way, AMgMiner finds 19 duplicate candidates (dashed arrows). Comparing this amount of duplicate candidates with those found by MgVEAM (see Figure 5.3(b)), we can see that AMgMiner traverses the search space in a more efficient way than MgVEAM. This improve is obtained because AMgMiner only extends those candidates that are in canonical form and the extensions are performed over the right-most path in the DFS tree representation. While MgVEAM does not have any pruning for decreasing the duplicate candidate computation. However, as we will show in our experiments, MgVEAM uses less memory than AMgMiner for mining all FASs, because AMgMiner needs additional memory to efficiently apply the search space pruning.

Finally, we analyze the computational complexity of AMgMiner for the worst case, where each multi-graph G_i in the collection D' has the same size; i.e., each multi-graph has n vertices and m edges, every $G_i \in D'$ is completely connected, each vertex label can be replaced by any other vertex label, and each edge label can be substituted by any other edge



(a) A multi-graph collection $D' = \{G_1, G_2, G_3, G_4\}$.



(b) Traversing of the search space by AMgMiner. In this example, the continuous dark arrows show the path followed by AMgMiner and the dashed arrows show the identified duplicate candidates, which are not extended again.

Figure 5.6: Example of the mining process of AMgMiner on a multi-graph collection $D' = \{G_1, G_2, G_3, G_4\}$; supposing that $minsup = 3/4$, $\tau = 0.55$ and the labels D, 2 and 3 can substitute the labels E, 1 and 4, respectively, where $L_V = \{A, B, C, D, E\}$ and $L_E = \{0, 1, 2, 3, 4, 5\}$.

label, where l_V and l_E are the number of vertex and edge labels respectively.

The complexity of AMgMiner is obtained by analyzing the algorithms above explained.

First, AMgMiner (Algorithm 5.5) traverses all vertices and edges of the collection D' for finding those frequent single-vertex and single-edge graphs, which are $O(n)$ and $O(m)$, respectively. Next, by applying our proposed DFS strategy (see Algorithm 5.6), all FASs are recursively obtained from the candidate pattern set returned by the *GenCandidateAMgMiner* function (see Algorithm 5.7). Also, this search process traverses all the edges that allow recursively growing a FAS to a new candidate pattern by adding an edge at each time (excluding those already traversed edges) and only those candidates fulfilling the similarity threshold τ are stored and extended; thus, $m-1$ extensions are made for the first extension of the FAS, $m-2$ for the second one, and so on, resulting in $O((m-1)!)$. Additionally, the *GenCandidateAMgMiner* function, first verifies if a pattern is in canonical form by mean of the *isMin* function, which in the worst case is $O(m!)$ because it performs all permutations of the vertices of a candidate FASs for computing its canonical form. Then *GenCandidateAmgMiner* extends only those patterns in canonical form by an edge, taking into account all possible vertex and edge labels which is $O(ml_V l_E)$. Thus, the complexity of *GenCandidateAMgMiner* is $O(m!) + O(ml_V l_E)$. Then, considering the *GenCandidateAMgMiner* function complexity, the comparisons performed for growing the patterns, and the number of times that *SearchAMgMiner* calls itself, we can conclude that the complexity of the *SearchAMgMiner* function is $O((m-1)!m!) + O((m-1)!ml_V l_E)$. Therefore, since this process is carried out for each frequent single-edge over each multi-graph of D', the complexity of these steps into AMgMiner is $O(dn)+O(dm)+O(d((m-1)!m!))+O(d(m-1)!ml_V l_E)$, where d is the number of multi-graphs in D', n and m are the number of vertices and edges, respectively, and, l_V and l_E are the number of vertex and edge labels, respectively. Thus, the complexity of the AMgMiner algorithm, in the worst case, is $O(dn) + O(d(m-1)!m!) + O(dm!l_V l_E)$.

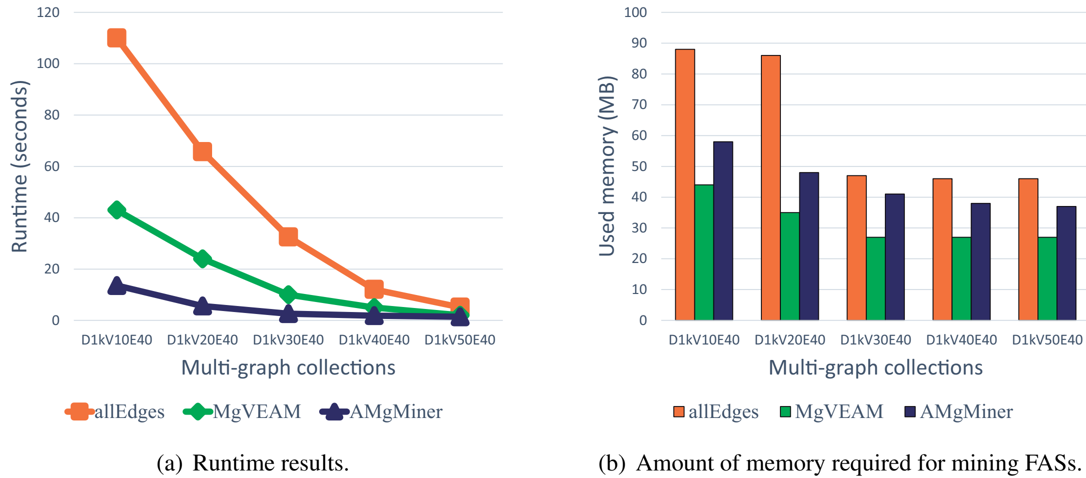
5.3 Experiments and Results

In this section, we show the performance of MgVEAM and AMgMiner over synthetic multi-graph collections, which were generated varying the graph characteristics (i.e., number

of multi-graph, edges and vertices). In this experiment, we contrast the performance of MgVEAM and AMgMiner against the performance of the allEdges method based on graph transformations (introduced in Section 4.1). Notice that we do not compare against the onlyMulti method (introduced in Section 4.2) because it does not mine all FASs and this chapter is focused on algorithms for mining all FASs. Also, we did not compare against any state-of-the-art algorithms because there is no algorithm able to mine FAS on multi-graph collections. Then, for evaluating the performance of MgVEAM and AMgMiner in real-world scenarios, we apply our algorithms over real-world multi-graph collections.

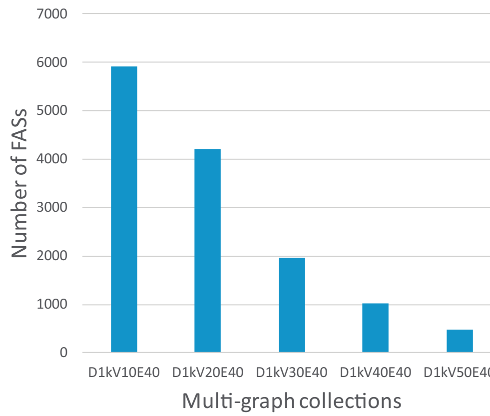
For our first experiment, we use the synthetic multi-graph collections used in Section 4.3, which were randomly generated using the PyGen graph emulation library. For building these collections, we first fix the size of the collection $|D| = 1000$ and the number of edges $|E| = 40$, varying the number of vertices $|V|$ from 10 to 50, with increments of 10. Next, we fix $|V| = 20$, maintaining $|D| = 1000$ and varying $|E|$ from 10 to 50, with increments of 10. Finally, we vary $|D|$ from 1000 to 5000, with increments of 1000, fixing $|V| = 20$ and $|E| = 40$. Additionally, for this experiment, two real-world multi-graph collections (*PROT-DB* obtained from protein classification and *WEB-DB* obtained from web document classification) were taken from the IAM graph database repository (Riesen and Bunke, 2008). All our experiments were carried out on a personal computer with an Intel(R) Core(TM) i5-3317U CPU @ 1.70 GHz with 4 GB of RAM. All the algorithms were implemented in ANSI-C and executed on Microsoft Windows 10.

In Figures 5.7, 5.8 and 5.9, we show the performance of MgVEAM and AMgMiner (which directly mine FASs from multi-graph collections), as well as allEdges (which is based on graph transformations) over synthetic multi-graph collections. These figures are split in three sub-figures: (a) runtime, (b) memory required for mining FASs, and (c) the number of identified FASs. It is important to highlight that MgVEAM, AMgMiner and allEdges mine the same number of patterns (i.e., all FASs). All the results reported in these sub-tables were achieved with the similarity threshold $\tau = 0.55$ and support threshold $minsup = 0.02$, as it is explained in Section 4.3 for these collections.



(a) Runtime results.

(b) Amount of memory required for mining FASs.

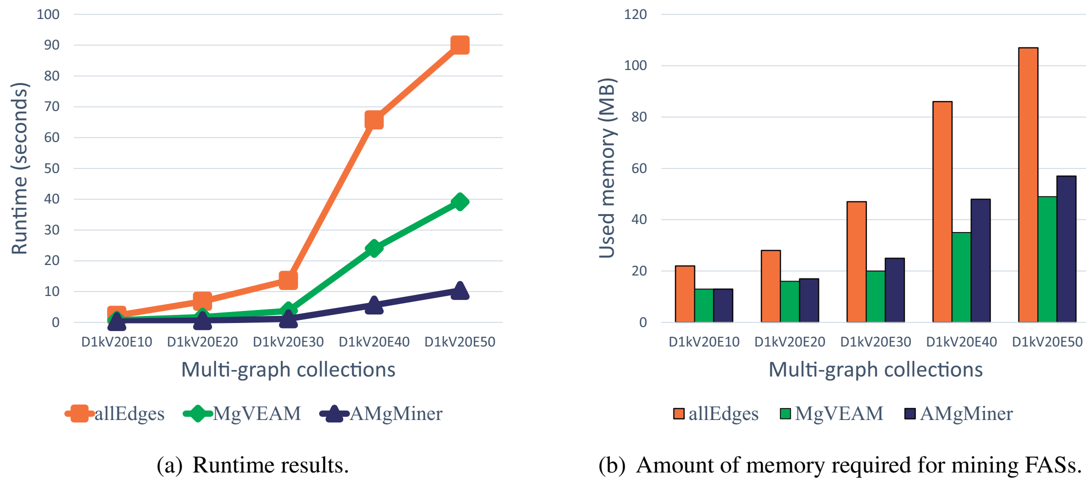


(c) Number of FASs identified on each multi-graph collection.

Figure 5.7: Performance of allEdges, MgVEAM and AMgMiner, using the similarity threshold $\tau = 0.55$ and the support threshold $minsup = 0.02$, over synthetic multi-graph collections obtained varying the number of vertices from $|V| = 10$ to $|V| = 50$ with increments of 10, fixing the number of multi-graphs $|D| = 1000$ and the number of edges $|E| = 40$.

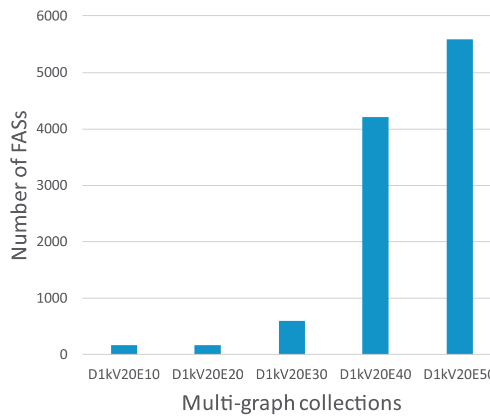
As we can see in Figure 5.7(a), the runtime of our proposed algorithms decreases when the number of vertices ($|V|$) grows, this happens because the greater the values of $|V|$ the less dense the multi-graphs in the collection (since the number of edges is fixed to 40); and therefore, fewer frequent subgraphs are mined (see Figure 5.7(c)). Also, due to the decrement of the number of FASs, less memory is required for the mining process (see Figure 5.7(b)).

In Figure 5.8, we show the performance of our proposed algorithms over multi-graph



(a) Runtime results.

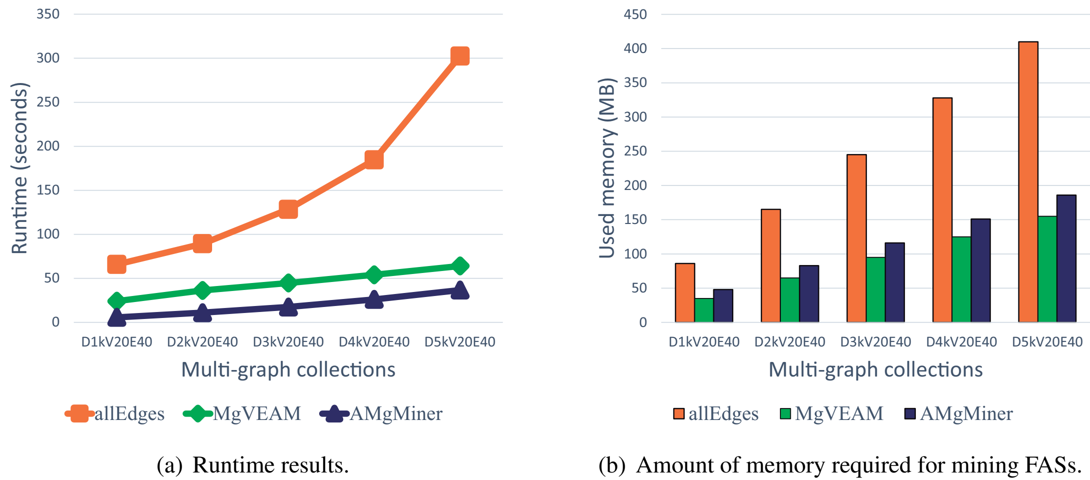
(b) Amount of memory required for mining FASs.



(c) Number of FASs identified on each multi-graph collection.

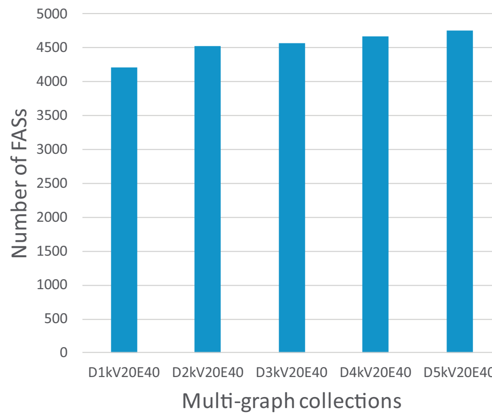
Figure 5.8: Performance of allEdges, MgVEAM and AMgMiner, using the similarity threshold $\tau = 0.55$ and the support threshold $minsup = 0.02$, over synthetic multi-graph collections obtained varying the number of edges from $|E| = 10$ to $|E| = 50$ with increments of 10, fixing the number of multi-graphs $|D| = 1000$ and the number of vertices $|V| = 20$.

collections obtained varying only the number of edges. In this figure, the performance of our algorithms is inverse to that one shown in Figure 5.7 because when the number of edges grows longer runtime and more memory are required for mining FASs. In fact, the greater the number of edges, the more dense the multi-graphs; and consequently, more frequent subgraphs are mined (see Figure 5.8(c)) and greater memory is required during the mining process (see Figure 5.8(b)).



(a) Runtime results.

(b) Amount of memory required for mining FASs.



(c) Number of FASs identified on each multi-graph collection.

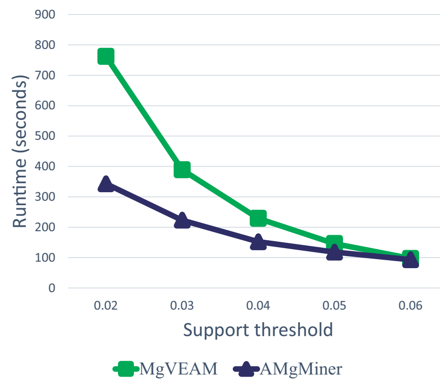
Figure 5.9: Performance of allEdges, MgVEAM and AMgMiner, using the similarity threshold $\tau = 0.55$ and the support threshold $minsup = 0.02$, over synthetic multi-graph collections obtained varying the number of multi-graphs from $|D| = 1000$ to $|D| = 5000$ with increments of 1000, fixing the number of multi-graphs $|V| = 20$ and the number of edges $|E| = 40$.

In Figure 5.9, we show the performance of our proposed algorithms over multi-graph collections obtained varying only the number of multi-graphs. In this figure, we can observe that the number of multi-graphs of the collection also affects the performance of our proposed algorithms. When the size of the collection grows, both runtime and memory increase. In fact, the greater the number of multi-graphs, the larger the number of occurrences for each FAS, and this increases the search space and the required memory.

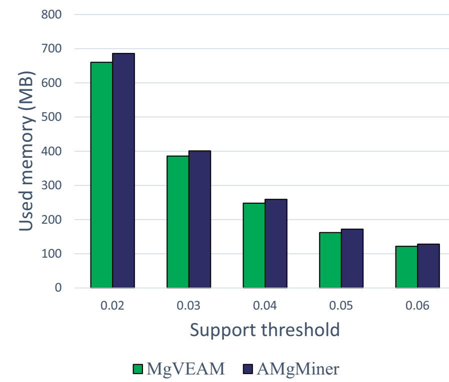
The results presented in Figures 5.7, 5.8 and 5.9 show that, in all cases, MgVEAM and AMgMiner achieve smaller runtimes, and they require less memory for storing the FASs, than the allEdges method. In these synthetic multi-graph collections, on average MgVEAM and AMgMiner are 3 and 8 times faster, respectively, tan allEdges. Besides, MgVEAM and AMgMiner use 48% and 60%, respectively, of the memory required by allEdges for mining all FASs. Moreover, on average, AMgMiner is 3 times faster than MgVEAM; while MgVEAM only requires 80% of the memory required by AMgMiner for mining all FASs. This is because AMgMiner uses some prunings for reducing the search space exploration during the mining process, but these prunings require additional memory; while MgVEAM only uses the downward closure to prune the search space.

In order to evaluate and show the performance of our proposals in real-world scenarios, we apply MgVEAM and AMgMiner over two real-world multi-graph collections (PROT-DB and WEB-DB). For showing the performance of our algorithm over both PROT-DB and WEB-DB multi-graph collections with the same amount of multi-graphs and due to PROT-DB contains only 600 multi-graphs, we take 600 multi-graphs from WEB-DB. In WEB-DB, the 600 multi-graphs have 9395 vertex labels and 64 edge labels, the average size of the multi-graphs is 65 vertices and 51 edges, and an average of 43 multi-edges per graph. In PROT-DB, the 600 multi-graphs have 3 vertex labels and 986 edge labels, the average size of the multi-graphs is 33 vertices and 73 edges, and an average of 69 multi-edges per graph.

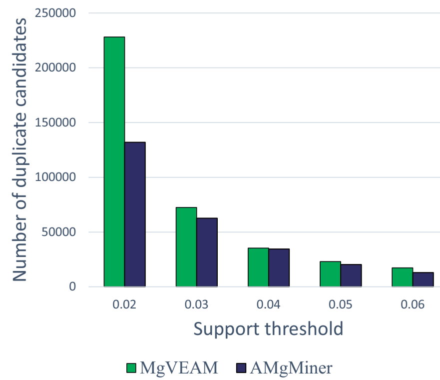
In Figures 5.10 and 5.11, we show the performance of MgVEAM and AMgMiner over PROT-DB and WEB-DB, respectively. Each figure is split into five sub-figures: (a) runtime; (b) memory required for mining FASs; (c) number of duplicated candidates; (d) amount of canonical form tests performed during the mining process; and (e) the number of FASs identified by our proposals. These results were achieved by MgVEAM and AMgMiner over the specified multi-graph collection using different support threshold values (i.e., values from 0.02 to 0.06 with increments of 0.01). These support threshold values were chosen for allowing to perform all the experiments in a reasonable time. As similarity threshold (τ), we used the average of the values in the substitution matrices of each collection (i.e., τ = 0.55). For



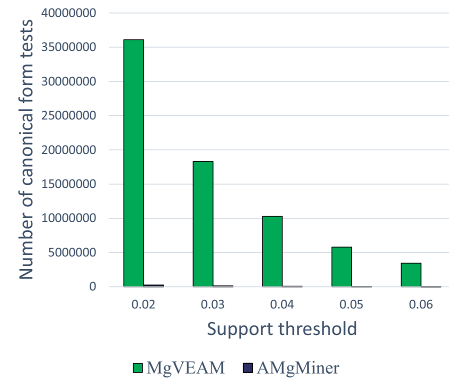
(a) Runtime.



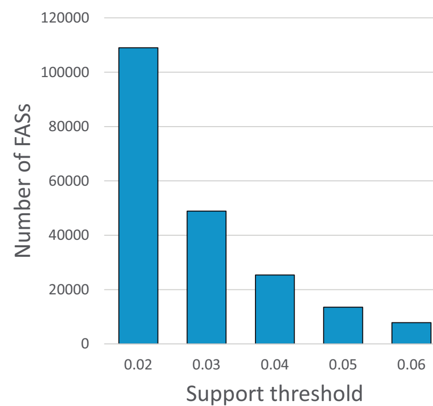
(b) Amount of memory required for mining FASs.



(c) Number of duplicate candidates.

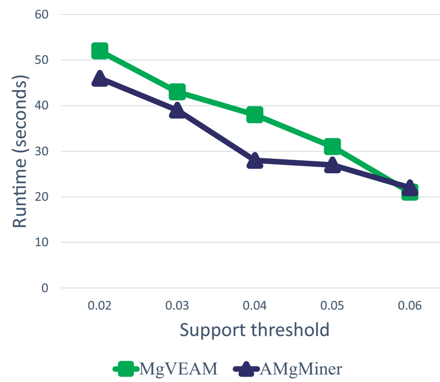


(d) Number of canonical form tests.

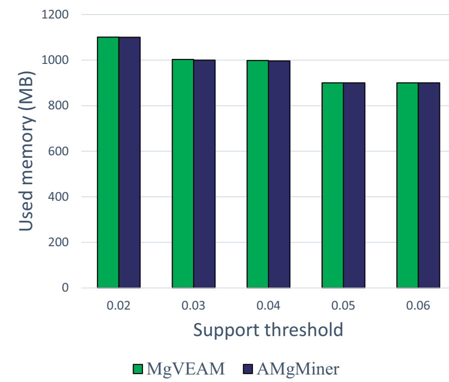


(e) Number of FASs identified on each multi-graph collection.

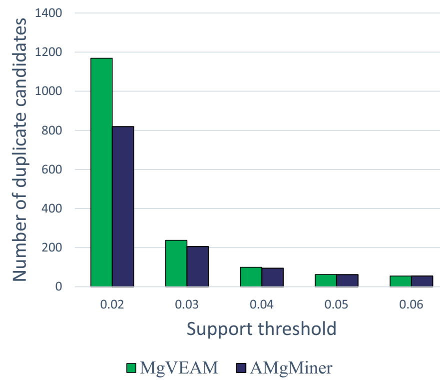
Figure 5.10: Performance of MgVEAM and AMgMiner, using the similarity threshold $\tau = 0.55$ with different values for the support threshold $minsup$, over PROT-DB.



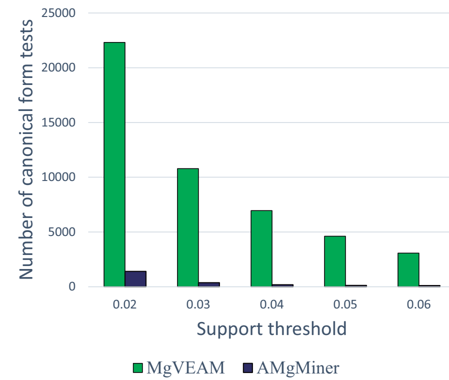
(a) Runtime.



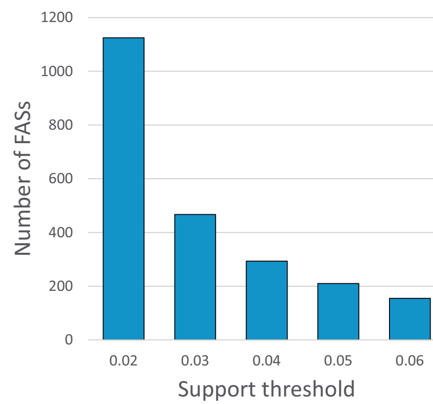
(b) Amount of memory required for mining FASs.



(c) Number of duplicate candidates.



(d) Number of canonical form tests.



(e) Number of FASs identified on each multi-graph collection.

Figure 5.11: Performance of MgVEAM and AMgMiner, using the similarity threshold $\tau = 0.55$ with different values for the support threshold $minsup$, over WEB-DB.

building the substitution matrices, since in these collections all edge labels are numerical, we manually assigned greater substitution probabilities to nearer values; on the other hand, since vertex labels are categorical, only substitutions by equality were allowed.

As we can see from Figures 5.10(a-b) and 5.11(a-b), the higher the support threshold the less runtime and memory are required by MgVEAM and AMgMiner for mining FASs over PROT-DB and WEB-DB. This happens because the performance of our proposed FAS miners is mainly affected by the number of duplicate candidates (see Figures 5.10(c) and 5.11(c)) mined from the collection and the number of canonical form tests performed (see Figures 5.10(d) and 5.11(d)), and when the support threshold increases less candidates fulfill the support restriction and therefore, less FASs are found (see Figures 5.10(e) and 5.11(e)), and less canonical form tests are performed. For this reason, the runtime required by MgVEAM and AMgMiner over WEB-DB is smaller than from PROT-DB.

In Figures 5.10(c-d) and 5.11(c-d), the usefulness of the prunings included into the AMgMiner algorithm is illustrated. In AMgMiner, only those patterns that are in canonical form are grown and the growth is performed over the right-most vertex; then, the number of candidates to process is reduced. This reduction leads to a decrement in the number of duplicate candidates, as well as the number of canonical form tests performed during the mining process. Therefore, as we can see in Figures 5.10(c-e) and 5.11(c-e), on average, AMgMiner identified 10% and 19% of duplicate candidates less than MgVEAM over WEB-DB and PROT-DB, respectively. In this way, on average, AMgMiner performs only 4% and 1% of the canonical form tests performed by MgVEAM during the mining process over WEB-DB and PROT-DB, respectively. These reductions lead to shorter runtimes for mining the same number of FASs.

Finally, according to the results shown in Figures 5.10 and 5.11, AMgMiner had better performance than MgVEAM (being AMgMiner 2 times faster than MgVEAM) over these real-world multi-graphs collections, but MgVEAM uses less memory than AMgMiner for mining all FASs (a memory reduction of 3%).

5.4 Summary and Conclusions

In this Chapter, the canonical adjacency matrix (CAM) and the depth-first search (DFS) canonical code have been extended for representing isomorphic multi-graphs. These extensions can be used for developing new frequent multi-graph mining algorithms. More specifically, these extended canonical forms were used for introducing two new algorithms (MgVEAM and AMgMiner). The proposed algorithms directly mine FASs from multi-graph collections, allowing approximate matching between edge and vertex labels but keeping the graph structure.

The performance of MgVEAM and AMgMiner, in terms of runtime and memory required for mining FASs, over synthetic multi-graph collections was compared against the allEdges method proposed in Section 4.1. According to our experiments, MgVEAM and AMgMiner are clearly faster, for mining FASs from muti-graph collections, in terms of runtime and memory, than allEdges. However, it is important to highlight that in allEdges, any traditional FAS miner can be applied and this allows mining different kinds of patterns on multi-graph collections.

On the other hand, AMgMiner performs less canonical form tests and identifies a smaller number of duplicate candidates than MgVEAM; while MgVEAM requires less memory than AMgMiner for mining FASs. Then, MgVEAM is the option to consider when the memory is a critical element; otherwise AMgMiner is the best option for mining FASs.

Appendix A shows some experiments on using the FASs mined by the algorithms proposed in this chapter.

Chapter 6

Mining Representative Patterns

In several real-world applications, it is common that a large number of frequent approximate subgraphs (FASs) is mined when a FAS miner is applied (Jia et al., 2011; Flores-Garrido et al., 2014; Acosta-Mendoza et al., 2016b; Emmert-Streib et al., 2016), making difficult the further use of them. For this reason, several researchers have focused on mining only representative (maximal, closed or clique, among others) subgraphs from the whole set of FASs (Yan and Han, 2003; Cavique et al., 2009; Flores-Garrido et al., 2014; Xu et al., 2014; Lartillot, 2015; Li and Wang, 2015; Liu and Gribskov, 2015; Caiyan and Ling, 2016; Chalupa, 2016; Chen et al., 2016; El Islem Karabadji et al., 2016; Hahn et al., 2016; Hao et al., 2016; Segundo et al., 2016; Salma, 2016; Shu-Jing et al., 2016; Unil and Gangin, 2016; Yi-Cheng et al., 2016; Demetrovics et al., 2017; Lu et al., 2017; Wu et al., 2017).

There are two alternatives for mining representative FASs: (1) directly mining only the representative FASs from a graph collection, and (2) mining all FASs and, in a post-processing step, extracting those representative FASs from the whole set. However, in the second alternative, a process for filtering out non-representative patterns is required, which adds additional computational cost to the graph mining process. Therefore, in this research, we followed the first alternative.

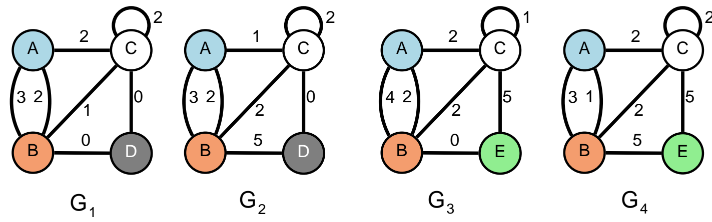
In this chapter we introduce two new algorithms for mining representative FASs (i.e., maximal, closed and clique) from multi-graph collections. Finally, through different experiments, we will show the performance of the proposed algorithms over synthetic and real-world multi-graph collections.

6.1 Maximal and Closed FASs

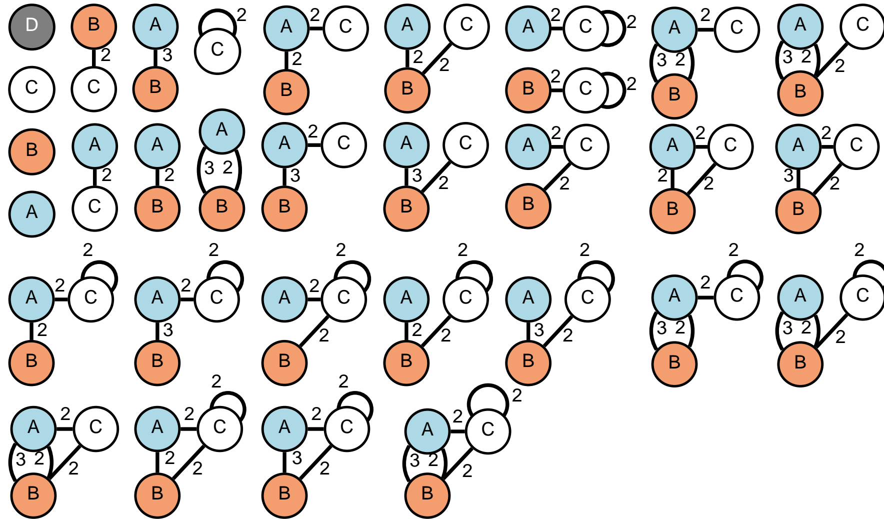
A well-known technique to obtain a set of representative FASs is by mining only maximal patterns. A maximal FAS (Huan et al., 2004; Thomas et al., 2010; Kimelfeld and Kolaitis, 2013; Flores-Garrido et al., 2014; Liu and Gribskov, 2015; Salma, 2016) is a FAS that is not sub-isomorphic to another FAS. An example of how the set of FASs can be reduced by preserving only the maximal ones is shown in Figure 6.1. Supposing that we mine FASs over the multi-graph collection $D' = \{G_1, G_2, G_3, G_4\}$ of Figure 6.1(a) using $minsup = 3/4$ and $\tau = 0.55$, Figures 6.1(b) and 6.1(c) illustrate the whole set of FASs and the maximal ones, respectively. As we can notice, the number of FASs is considerably reduced, from 32 to 2, when only the maximal FASs are mined in D'.

From the maximal FASs it is possible to recompute the whole set of FASs because all of them are summarized into the maximal ones. However, from the maximal FASs, the information about the support of the non-maximal FASs cannot be retrieved. To face this problem, in several applications, closed patterns are used (Yan and Han, 2003; Yan et al., 2003; Cheng et al., 2006; Borgelt and Meinl, 2009; Takigawa and Mamitsuka, 2011; Lartillot, 2015; Demetrovics et al., 2017). A closed frequent subgraph is a pattern that is not sub-isomorphic to another frequent subgraph with the same frequency (Yan and Han, 2003; Song and Chen, 2006; Borgelt and Meinl, 2009; Takigawa and Mamitsuka, 2011; Salma, 2016; Demetrovics et al., 2017). Thus, from the closed frequent subgraphs, which is a superset of the maximal ones, it is possible to recompute the whole set of frequent subgraphs including the information about their support. For this reason, the closed patterns are commonly used for reducing the size of a pattern set. It is important to highlight that, commonly the number of closed patterns is larger than the number of maximal ones, but they are less than the number of whole frequent patterns.

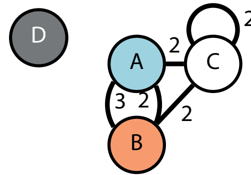
An important detail to take into account is that, in our approximate context, when we apply the traditional closed condition over the FASs, usually the whole set of FASs is obtained, since in the approximate graph mining approach rarely two patterns have exactly the same



(a) A multi-graph collection $D' = \{G_1, G_2, G_3, G_4\}$.



(b) The whole FAS set of D'.



(c) Maximal FASs of D'.

Figure 6.1: Example of all FASs and the maximal FASs mined form a multi-graph collection $D' = \{G_1, G_2, G_3, G_4\}$; supposing that $minsup = 3/4$, $\tau = 0.55$ and the labels D, 2 and 3 can substitute the labels E, 1 and 4, respectively, where $L_V = \{A, B, C, D, E\}$ and $L_E = \{0, 1, 2, 3, 4, 5\}$.

frequency. An example of this fact can be seen in Figure 6.2, where we show the FASs mined from the multi-graph collection D' of Figure 6.1, including the frequency, denoted by s, of each mined FAS. As it can be seen, if we apply the traditional closed condition over these patterns, the whole set of FASs is kept as closed patterns because there is not a FAS with the same frequency of any of its sub-isomorphic FASs. Thus, another solution, called δ-tolerance or generalized closed patterns (Cheng et al., 2006, 2008; Boley et al., 2009; Bringmann et al.,

2011; Takigawa and Mamitsuka, 2011; Gay et al., 2012), was proposed for allowing a relaxation for the strict closed condition. Then, following this idea, in Definition 6.1, we introduce the generalized closed FAS.

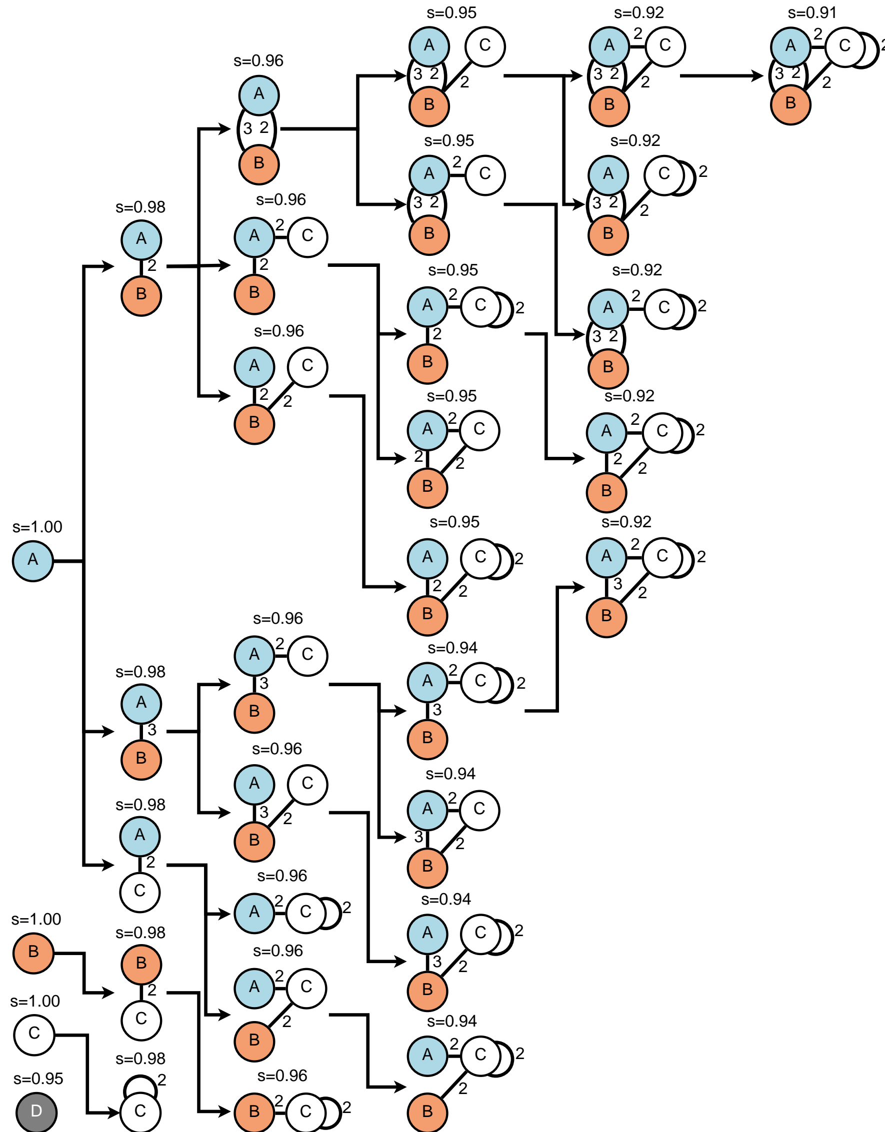


Figure 6.2: The set of all FASs, and each support value s, mined from the multi-graph collection of Figure 6.1(a); supposing that $minsup = 3/4$, $\tau = 0.55$ and the labels D, 2 and 3 can replace the labels E, 1 and 4, respectively, where $L_V = \{A, B, C, D, E\}$ and $L_E = \{0, 1, 2, 3, 4, 5\}$.

Definition 6.1 (Generalized closed frequent approximate subgraph). *Given a multi-graph collection D', a minimum support threshold $minsup$, a similarity threshold τ, a closed threshold $\delta \in [0, 1]$, and a frequent multi-graph G in D', G is a generalized closed FAS iff there is no FAS G' such that G is*

subgraph of G' and $appSup(G', D') \geq (1 - \delta)appSupp(G, D')$.

As it can be seen from Definition 6.1, when $\delta = 0$ the generalized closed condition becomes into the traditional closed condition. Moreover, when $\delta = 1$ the generalized closed condition becomes into the maximal condition because $appSup(G', D')$ is always greater than or equal to 0 and the only way that G would be a generalized closed FAS if it does not have frequent super-graphs (i.e., it is a maximal FAS). Using this generalized closed condition, when $0 < \delta < 1$, we can obtain more patterns than using the maximal condition, but less than using the traditional closed one; because the generalized closed condition allows filtering sub-patterns with small frequency differences. Then, if we have an algorithm for computing the generalized closed FASs from a multi-graph collection, we can also use it for mining both maximal and traditional closed FASs. Therefore, in this section, we propose an algorithm for mining generalized closed FASs in multi-graph collections.

6.1.1 The GenCloMgVEAM Algorithm

For mining generalized closed FASs, we extend our proposed MgVEAM algorithm. The main idea of our extension, called *GenCloMgVEAM* (**Gen**eralized **Clo**sed **M**ulti-**g**raph **V**ertex and **E**dge **A**pproximate **M**iner), consists in, following the approximate graph mining process of MgVEAM (see Section 5.1.2) but storing only those FASs that fulfill the generalized closed condition (Definition 6.1). This condition can be verified during the mining process of MgVEAM because when a FAS G is recursively extended by adding an edge, if all the frequent extensions of G fulfill the generalized closed condition regarding G, then G is stored as a generalized closed FAS; otherwise G is discarded.

Including the above mentioned idea into a frequent subgraph algorithm it is possible to obtain algorithms for mining generalized closed FASs without additional cost for the original mining process. However, likewise our proposed AMgMiner algorithm (see Section 5.2), some reported algorithms introduce prunings in the search space and in the pattern growth process

for speeding up the mining process. In this case, the generalized closed condition verification incorporates additional cost. For this reason, we decide to extend MgVEAM instead of AMgMiner.

GenCloMgVEAM is outlined in Algorithm 6.1, where the frequent approximate single-vertex set is firstly identified in a collection of multi-graphs D'. Then, iterating among this vertex set, GenCloMgVEAM extends each FAS T by calling the function *SearchGenCloMgVEAM*; traversing only those multi-graphs $G_i \in D'$ that contain at least one occurrence of T.

In Algorithm 6.2, the process performed by the *SearchGenCloMgVEAM* function is detailed. This function, recursively performs the extension of FASs. The candidate set is obtained by calling the *GenCandidateMgVEAM* function; extending a FAS by all possible extensions (by adding one edge at each time). Then, only those candidates that fulfill the support constraint and which have not been identified in previous steps are recursively extended by the *SearchGenCloMgVEAM* function. During this process, only those patterns that are identified as generalized closed FASs are kept as output (see lines $3-5$ of Algorithm 6.2).

Algorithm 6.1: $GenCloMgVEAM(D', \tau, minsup, \delta, F)$

Input: D' : Multi-graph collection, τ : Similarity threshold, $minsup$: Support threshold, δ : Closed threshold.
Output: F : Generalized closed frequent approximate subgraphs in D'.

```
1 C ←The frequent approximate single-vertex graph set in D' ;   // based on Definition 2.6
2 F ← C; NF ← ∅;
3 foreach T ∈ C do
4 |  D'_T ←The subgraphs of D' in which T is approximate sub-isomorphic to T;
5 |  SearchGenCloMgVEAM(T, D'_T, D', τ, minsup, δ, F, NF);
```

The goal of the *GenCandidateMgVEAM* function, as it was described in Section 5.1.2, is to compute all candidate extensions of a given frequent multi-graph G'. In *GenCandidateMgVEAM*, all extensions of G' and their occurrences in the multi-graph collection $D'_{G'}$ are searched; computing and storing their corresponding CAM codes and their similarity values.

Algorithm 6.2: $SearchGenCloMgVEAM(G', D'_{G'}, D', \tau, minsup, \delta, F, NF)$

Input: G' : FAS, $D'_{G'}$: Occurrence set of G', D' : Multi-graph collection, τ : Similarity threshold, $minsup$: Support threshold, δ : Closed threshold, NF : Multi-graph set.
Output: F : Generalized closed FASs in D'.

```
1 C ← GenCandidateMgVEAM(G', D'_G', D', τ) ;            // Algorithm 5.4
2 foreach T ∈ C do
3     if appSupp(T, D') ≥ minsup then
4         if appSup(T, D') ≥ [1 − δ]appSupp(G', D') then
5             F ← F \ {G'}; NF ← NF ∪ {G'} ;            // based on Definition 6.1
6         if T ∉ F and T ∉ NF then
7             F ← F ∪ {T};
8             D'_T ←The subgraphs of D' which are approximate sub-isomorphic to T;
9             SearchGenCloMgVEAM(T, D'_T, D', τ, minsup, δ, F, NF);
```

It is important to highlight that, in GenCloMgVEAM, the order of the pattern identification does not alter the final result. This is because we always extend an edge from all possible extensions of each FAS. This growing process allows finding the FASs from all possible traverses, then, all FASs are mined from any starting vertex. In this way, the support values of the extended FASs and their children are known in the growing process; allowing verifying the generalized closed condition without including additional computational cost into the mining process. Then, the final result of GenCloMgVEAM only depends of the pattern frequencies and it does not depend of the pattern identification order.

Our *GenCloMgVEAM* algorithm examines the same set of candidate as MgVEAM; besides, the generalized closed condition verification, which was included for keeping only generalized closed FASs, does not introduces additional costs to the mining process because it is a comparison between two pre-calculated support values. For these reasons, the computational complexity of MgVEAM is kept in GenCloMgVEAM, which is $O(dmm!l_V l_E nn!)$; in the worst case, for a multi-graph collection containing d multi-graphs with n vertices and m edges, where l_V and l_E are the number of vertex and edge labels, respectively.

6.2 Clique FASs

Another kind of patterns that has been commonly used for representing the whole set of the mined FASs is the clique FASs. In real-world applications such as biochemical compounds analysis and communities detection, clique patterns have been used for representing the mined patterns (Huan et al., 2006; Cavique et al., 2009; Jia et al., 2009; Tsourakakis, 2014; Xu et al., 2014; Chalupa, 2016; Chen et al., 2016; Hahn et al., 2016; Segundo et al., 2016; Lu et al., 2017). As in Definition 6.2, a clique subgraph is a pattern where every pair of vertices are connected by an edge (Huan et al., 2006; Cavique et al., 2009; Jia et al., 2009; Jungnickel, 2012; Tsourakakis, 2014; Xu et al., 2014; Deo, 2017; Rahman, 2017; Lu et al., 2017).

Definition 6.2 (Clique frequent approximate subgraph)**.** *Let G' be a labeled multi-graph, and let D' be a multi-graph collection, G' is a clique frequent approximate subgraph in D' iff it is a FAS in D' and each vertex $v \in V_{G'}$ is connected to any other vertex $w \in V_{G'}$ by at least one edge.*

In Figure 6.3, we show the clique patterns identified in the multi-graph collection D' of Figure 6.1(a). As it can be seen, the number of patterns is reduced from 32 to 18 when only the clique patterns are kept.

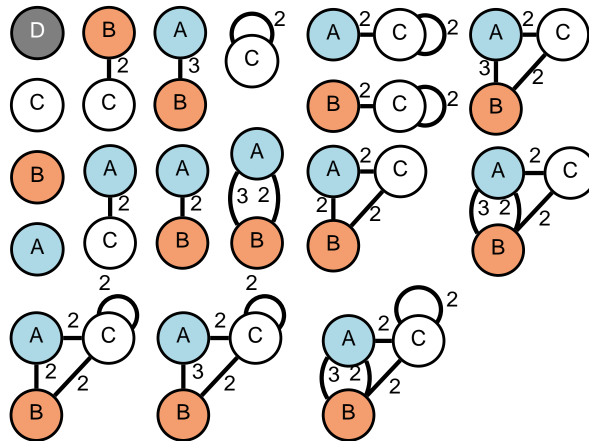


Figure 6.3: Example of clique FASs identified from the multi-graph collection $D' = \{G_1, G_2, G_3, G_4\}$ of Figure 6.1; supposing that $minsup = 3/4$, $\tau = 0.55$ and the labels D, 2 and 3 can substitute the labels E, 1 and 4, respectively, where $L_V = \{A, B, C, D, E\}$ and $L_E = \{0, 1, 2, 3, 4, 5\}$.

6.2.1 The CliqueAMgMiner algorithm

Our proposed algorithm for mining clique FASs from multi-graph collections, called *CliqueAMgMiner* (**Clique** **A**pproximate **M**ulti-**g**raph **Miner**), is an extension of our AMgMiner algorithm introduced in Section 5.2.2. CliqueAMgMiner follows the procedures of AMgMiner for mining all the FASs but storing only those patterns that are clique, according to Definition 6.2. It is important to highlight that we extend AMgMiner instead of MgVEAM because AMgMiner achieved the best runtime performance in our experiments in Section 5.3. Besides, the clique condition verification introduces the same cost into AMgMiner and MgVEAM because, in both algorithms, the clique verification consists in traversing the edge set of each FASs.

The idea of CliqueAMgMiner consists in identifying the clique FASs directly on a multi-graph collection starting with the frequent approximate single-vertex and single-edge subgraphs. Then, each frequent approximate single edge is recursively extended by adding a single-edge at a time, following a DFS approach, while the support threshold is fulfilled. In CliqueAMgMiner, only clique patterns are stored in the output FAS set.

CliqueAMgMiner is shown in Algorithm 6.3, where, once all the frequent approximate single-edge graphs have been computed from a multi-graph collection, each pattern of C (frequent approximate single-edge subgraph) is recursively extended in the *SearchClique* function.

Algorithm 6.3: $CliqueAMgMiner(D', \tau, minsup, F)$

Input: D' : Multi-graph collection, τ : Similarity threshold, $minsup$: Support threshold.
Output: F : Clique frequent approximate subgraphs in D'.

```
1 F ←The frequent approximate single-vertex graph set in D' ;      // based on Definition 2.6
2 C ←The frequent approximate single-edge graph set in D' ;        // based on Definition 2.6
3 F ← F ∪ C; NF ← ∅;
4 foreach T ∈ C do
5     D'_T ←The subgraphs of D' which are approximate sub-isomorphic to T;
6     SearchClique(T, DFScode(T), D'_T, D', τ, minsup, F, NF);
```

In Algorithm 6.4, the recursive function for extending patterns is shown. This function,

following a DFS approach, recursively extends each pattern by adding a single edge at a time while the support threshold is fulfilled. The recursion starts computing the candidate set by calling the *GenCandidateAMgMiner* function, which, as we explained in Section 5.2.2, extends a FAS by all possible extensions (by adding one edge). Then, only those frequent candidates, which have not been identified in previous steps, are extended; performing a recursive call to the *SearchClique* function. Only those patterns that fulfill the clique condition (which is verified through the *isClique* function) are kept in the output FAS set (see lines $4-5$ of Algorithm 6.4). It is important to highlight that the clique verification is performed only over those FASs that were obtained by a backward extension (see the line 4 of Algorithm 6.4) because forward extensions always add a new vertex to the pattern and there is no way that this vertex is connected to all the other vertices in the pattern. In this way, we avoid to perform some unnecessary clique verifications.

Algorithm 6.4: $SearchClique(G, c, D'_G, D', \tau, minsup, F, NF)$

Input: G : FAS, c : DFS code of G, D'_G : Occurrence set of the pattern G, D' : Multi-graph collection, τ : Similarity threshold, $minsup$: Support threshold, NF : Multi-graph set.
Output: F : Clique frequent approximate subgraphs in D'.

```
1 C ← GenCandidateAMgMiner(c, D'_c, D', τ) ;                    // Algorithm 5.7
2 foreach T ∈ C where T = G ◇ e do
3     if appSupp(T, D') ≥ minsup and T ∉ F and T ∉ NF then
4         if e is a backward extension and isClique(T) = true then
5             F ← F ∪ {T};
6         else NF ← NF ∪ {T};
7         D'_T ←The subgraphs of D' which are approximate sub-isomorphic to T;
8         SearchClique(T, DFScode(T), D'_T, D', τ, minsup, F, NF);
```

It is important to highlight that, in the growing process of CliqueAMgMiner, only two kinds of extensions can be performed: (1) forward and (2) backward extensions (see Definition 5.2). Through a forward extension we cannot be able to obtain a clique candidate; however, if this candidate is frequent, then it is stored as a FAS for future growth. On the other hand, through a backward extension we can obtain a clique regardless of whether the extended FAS is clique or not. Then, even if the FASs are extended by an edge and a clique is not obtained, the pattern growth process continues on all FASs (whether they are clique

or not). In the pattern growth process, only the clique FAS are stored in the output set and this process only stops when the support threshold is not fulfilled.

For analyzing the complexity of CliqueAMgMiner, we must take into account that, in the *SearchClique* function, the clique condition is verified by traversing all edges in a pattern, which is, in the worst case, $O(m)$. Then, since the complexity of the *SearchAMgMiner* function is $O((m-1)!m!)+O((m-1)!ml_V l_E)$ (see Section 5.2.2), the complexity of the *SearchClique* function is $O(m(m-1)!m!)+O(m(m-1)!ml_V l_E))$. Thus, the complexity of CliqueAMgMiner is, in the worst case, $O(dn)+O(dm)+O(dm(m-1)!m!)+O(dm(m-1)!ml_V l_E))$, resulting $O(dn)+O(d(m!)^2)+O(dm!ml_V l_E))$; for a multi-graph collection containing d multi-graphs with n vertices and m edges, where l_V and l_E are the number of vertex and edge labels, respectively.

6.3 Experiments and Results

For evaluating the performance of GenCloMgVEAM and CliqueAMgMiner, we carried out experiments over synthetic and real-world multi-graph collections. Besides, for showing that the generalized closed and clique conditions do not add a high cost to the mining process, we compare the performance of GenCloMgVEAM and CliqueAMgMiner with the performance of MgVEAM and AMgMiner (proposed in the sections 5.1.2 and 5.2.2, respectively). Moreover, since we are aiming to reduce the number of FASs, we contrast the number of representative FASs regarding all FASs (mined by MgVEAM and AMgMiner).

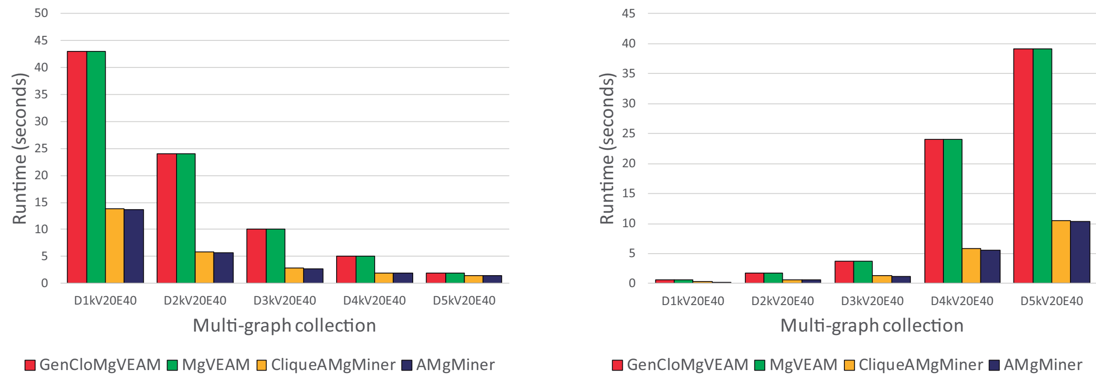
The synthetic multi-graphs used for our first experiment are the same collections of the sections 4.3 and 5.3, which were generated using the PyGen graph emulation library following three strategies: first, the size of the collection $|D| = 1000$ and the number of edges $|E| = 40$ were fixed, and the number of vertices $|V|$ was varied from 10 to 50, with increments of 10; second, we fixed $|V| = 20$ and $|D| = 1000$, varying $|E|$ from 10 to 50, with increments of 10; and finally, $|D|$ was varied from 1000 to 5000, with increments of 1000, but fixing $|V| = 20$

and $|E| = 40$. On the other hand, for our second experiment, we used the same two multi-graph collections, *PROT-DB* and *WEB-DB*, of Section 5.3. WEB-DB has 600 multi-graphs, 9395 vertex labels and 64 edge labels, the average size of the multi-graphs is 65 vertices and 51 edges, with an average of 43 multi-edges per graph. PROT-DB has 600 multi-graphs, 3 vertex labels and 986 edge labels, the average size of the multi-graphs is 33 vertices and 73 edges, with an average of 69 multi-edges per graph. All our experiments were carried out on a personal computer with an Intel(R) Core(TM) i5-3317U CPU @ 1.70 GHz with 4 GB of RAM. All the algorithms were implemented in ANSI-C and executed on Microsoft Windows 10.

In Figure 6.4, we show the runtimes of GenCloMgVEAM and CliqueAMgMiner, for mining representative FASs from different synthetic multi-graph collections. In this figure, the runtimes of MgVEAM and AMgMiner are also shown for contrasting the performance of these algorithms against the performance of our representative FAS miners. This figure is split in three sub-figures according to the characteristics of the collections: (a) fixing $|D|$ and $|E|$, varying $|V|$; (b) fixing $|D|$ and $|V|$, varying $|E|$; and (c) fixing $|V|$ and $|E|$, varying $|D|$. All the results reported in these sub-figures were achieved with a similarity threshold $\tau = 0.55$ and a support threshold $minsup = 0.02$, as it is explained in Section 4.3 for these collections.

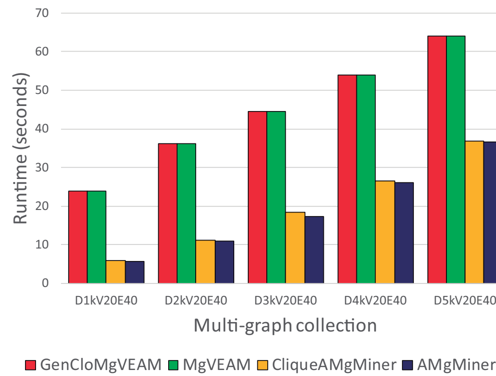
As we can see in Figure 6.4, when the number of vertices grows, less runtime is required for mining FASs (see Figure 6.4(a)), this happens because, since the number of edges is fixed to 40, the greater the number of vertices the less dense the multi-graphs in the collection; and therefore, as we will show in Figure 6.5, fewer FASs are mined. When the number of edges and the size of the collection grow, a longer runtime is required for mining FASs (see figures 6.4(b-c)). In fact, the greater the number of edges, the more dense the multi-graphs, and consequently, more FASs are mined; while the greater the number of multi-graphs, the larger the number of occurrences for each FAS, and thus, the larger the search space.

The results reported in Figure 6.4 show that GenCloMgVEAM and MgVEAM have the



(a) Multi-graph collections obtained by varying $|V|$, keeping $|D| = 1000$ and $|E| = 40$.

(b) Multi-graph collections obtained by varying $|E|$, keeping $|D| = 1000$ and $|V| = 20$.



(c) Multi-graph collections obtained by varying $|D|$, keeping $|V| = 20$ and $|E| = 40$.

Figure 6.4: Runtime, in seconds, achieved by AMgMiner, MgVEAM, GenCloMgVEAM and CliqueAMgMiner with different closed threshold δ values, a support threshold $minsup = 0.02$ and a similarity threshold $\tau = 0.55$ over synthetic multi-graph collections.

same performance, in terms of runtime; while CliqueAMgMiner requires slightly more runtime than AMgMiner. In this way, as we expected, the generalized closed condition verification introduced into MgVEAM does not affect the GenCloMgVEAM performance, and despite the computational cost added by the clique condition verification into AMgMiner, the runtime is not highly increased.

The size of the representative FAS subset mined by GenCloMgVEAM and CliqueAMgMiner over the synthetic multi-graph collections is shown in Figures 6.5, 6.6 and 6.7. In Figure 6.5, the used multi-graph collections were obtained by fixing $|D| = 1000$ and $|E| = 40$,

varying $|V|$ from 10 to 50 with increments of 10. In Figure 6.6, the used multi-graph collections were obtained by fixing $|D| = 1000$ and $|V| = 20$, varying $|E|$ from 10 to 50 with increments of 10; while the collections used in Figure 6.6 were obtained by setting $|V| = 20$ and $|E| = 40$, varying $|D|$ from 1000 to 5000 with increments of 1000. It is important to highlight that, all FASs are the patterns mined by AMgMiner, MgVEAM and allEdges, the generalized closed FASs were mined by GenCloMgVEAM with different closed threshold (δ) values, and the clique FASs were mined by CliqueAMgMiner. Besides, when $\delta = 0$ and $\delta = 1$, the patterns mined are the traditional closed FASs and the maximal FASs, respectively.

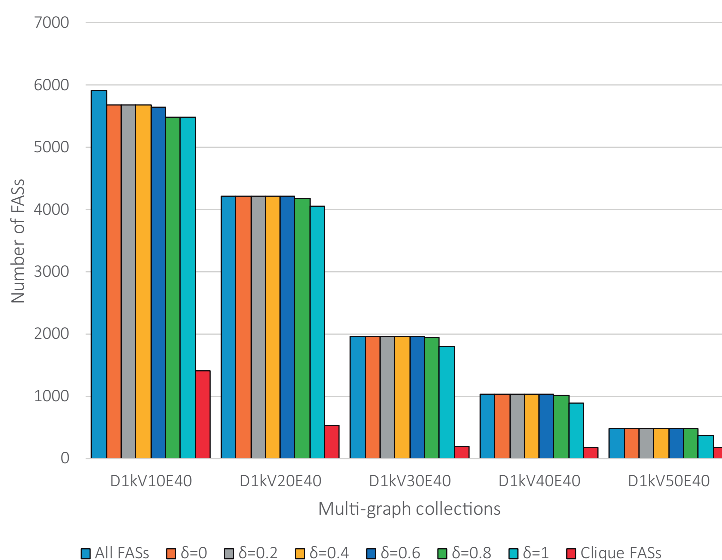


Figure 6.5: Number of FASs identified by AMgMiner ("All FASs"), as well as GenCloMgVEAM ("Generalized closed FASs") and CliqueAMgMiner ("Clique FASs") with different closed threshold δ values, a support threshold $minsup = 0.02$ and a similarity threshold $\tau = 0.55$ over synthetic multi-graph collections obtained by varying $|V|$, keeping $|D| = 1000$ and $|E| = 40$.

From Figures 6.5, 6.6 and 6.7, it can be noticed that, on average, by mining clique FASs we achieved a remarkable reduction of 72% of the whole set of FASs. On the other hand, by mining generalized closed FASs we obtained a reduction of 7%; this happens because the mined patterns in these collections are very small (FASs with two or three edges). Therefore, most of these FASs are already maximal or closed.

In our second experiment, we apply GenCloMgVEAM, CliqueAMgMiner, AMgMiner

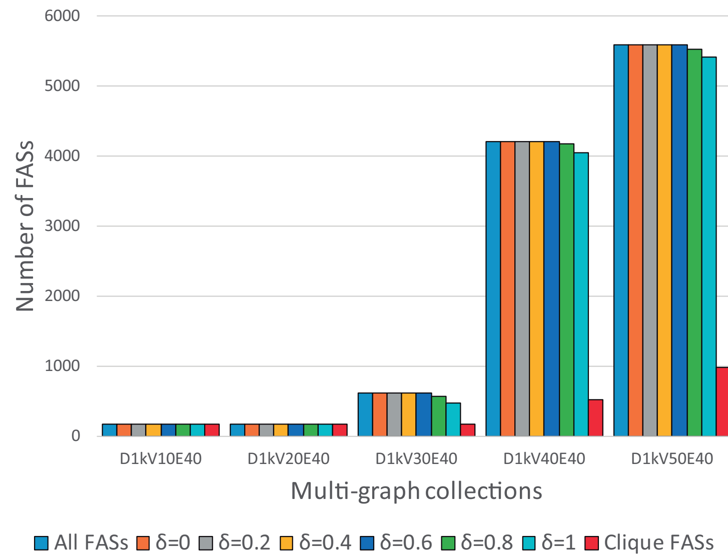


Figure 6.6: Number of FASs identified by AMgMiner ("All FASs"), as well as GenCloMgVEAM ("Generalized closed FASs") and CliqueAMgMiner ("Clique FASs") with different closed threshold δ values, a support threshold $minsup = 0.02$ and a similarity threshold $\tau = 0.55$ over synthetic multi-graph collections obtained by varying $|E|$, keeping $|D| = 1000$ and $|V| = 20$.

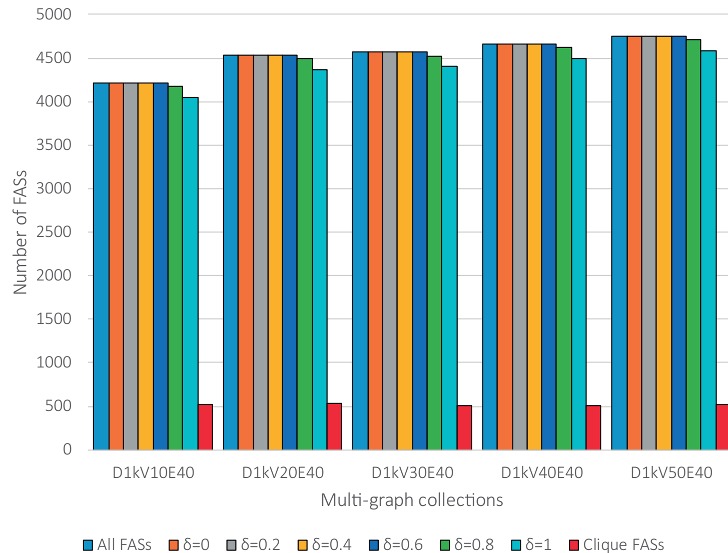
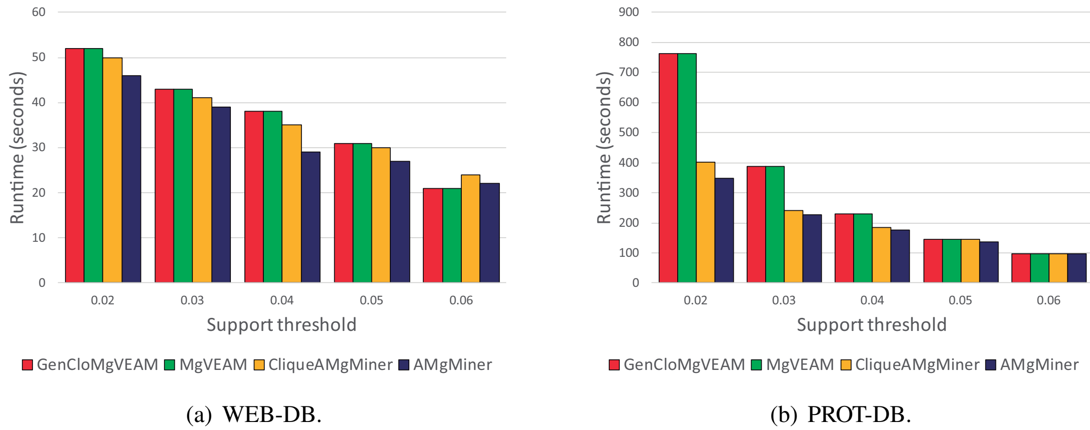


Figure 6.7: Number of FASs identified by AMgMiner ("All FASs"), as well as GenCloMgVEAM ("Generalized closed FASs") and CliqueAMgMiner ("Clique FASs") with different closed threshold δ values, a support threshold $minsup = 0.02$ and a similarity threshold $\tau = 0.55$ over synthetic multi-graph collections obtained by varying $|D|$, keeping $|V| = 20$ and $|E| = 40$.

and MgVEAM over WEB-DB and PROT-DB using different closed threshold values (from $\delta = 0$ to 1 with increments of 0.2) for GenCloMgVEAM and the same values for the support (from $minsup = 0.02$ to 0.06 with increments of 0.01) and similarity ($\tau = 0.55$) thresholds described in Section 5.3. In Figure 6.8, the runtimes of GenCloMgVEAM, CliqueAMgMiner, AMgMiner and MgVEAM are shown.

Figure 6.8 is split into two sub-figures by showing the runtime required for each algorithms over (a) WEB-DB and (b) PROT-DB. All the results reported in these sub-figures were achieved with the similarity threshold $\tau = 0.55$, as explained in Section 5.3.



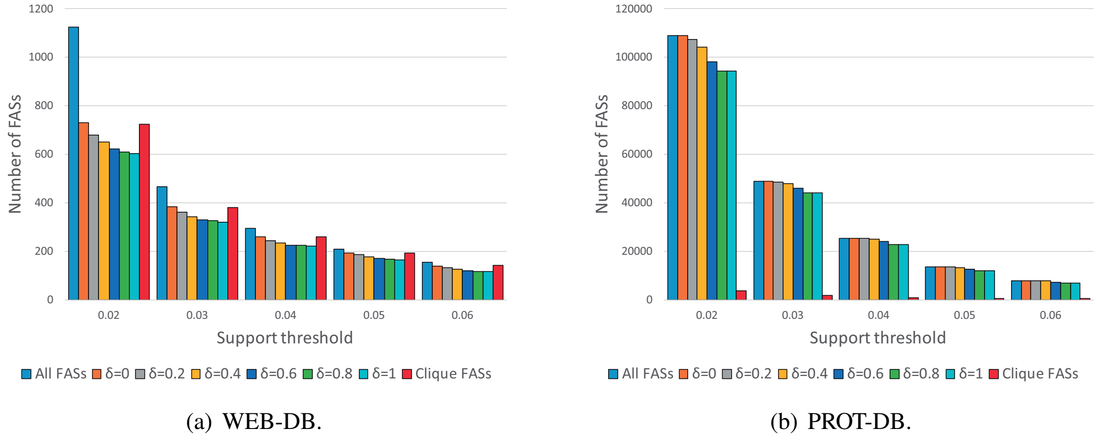
(a) WEB-DB. (b) PROT-DB.

Figure 6.8: Performance, in terms of runtime (seconds), achieved by AMgMiner, MgVEAM, GenCloMgVEAM and CliqueAMgMiner with different support threshold $minsup$ values and a similarity threshold $\tau = 0.55$ over two real-world multi-graph collections.

As we can see from Figure 6.8, as in the experiments over synthetic multi-graph collections, GenCloMgVEAM and MgVEAM achieved the same performance, in terms of runtime; while CliqueAMgMiner requires slightly more runtime than AMgMiner. This experiment confirms that the generalized closed condition does not affect the graph mining performance of GenCloMgVEAM, and that the complexity introduced by the clique condition verification is low.

On the other hand, in Figure 6.9, we show the size of the representative subset of FASs mined by GenCloMgVEAM and CliqueAMgMiner over WEB-DB and PROT-DB. This Figure contains two sub-figures for showing the results achieved by the algorithms over: (a) WEB-

DB, and (b) PROT-DB. In these sub-figures, the number of FASs mined by AMgMiner (i.e., all FASs), the number of generalized closed FASs mined by GenCloMgVEAM with different closed threshold (δ) values, and the number of clique FAS mined by CliqueAMgMiner are shown.



(a) WEB-DB.

(b) PROT-DB.

Figure 6.9: Number of FASs identified by AMgMiner (denoted as "All FASs"), CliqueAMgMiner (denoted as "Clique FASs") and GenCloMgVEAM (denoted as "Generalized closed FASs") with different closed threshold δ values, different support threshold $minsup$ values and a similarity threshold $\tau = 0.55$ over two real-world multi-graph collections.

In Figure 6.9(a), over WEB-DB, it can be seen that, unlike in the experiment over the synthetic multi-graph collections, the number of representative FASs obtained by GenCloMgVEAM and CliqueAMgMiner is clearly smaller than the whole set of FASs. In this case, on average, GenCloMgVEAM achieves a FAS reduction of 30% over WEB-DB; while by mining the clique FASs we obtain a reduction of 16%. In this multi-graph collection, the identified FASs are highly connected, resulting into a large number of clique FASs, and consequently, CliqueAMgMiner does not reduce too much the FAS set. On the other hand, as we can see in Table 6.9(b), in most cases, GenCloMgVEAM mines all FASs when the traditional closed patterns are searched (i.e., when $\delta = 0$). In this way, we can conclude that, for reducing the size of the FAS set, the traditional closed FASs are less useful than the generalized closed ones mined by GenCloMgVEAM. Furthermore, over PROT-DB, the highest reduction was obtained by mining the clique FASs, which, on average, is 95%; while

by mining the generalized closed FASs we obtained a reduction of 12%. It is important to highly that the patterns identified in PROT-DB were less connected than the ones mined in WEB-DB. Therefore, with the clique FASs we achieved a high reduction of the set of FASs.

6.4 Summary and Conclusions

In this Chapter, we introduced two algorithms, GenCloMgVEAM and CliqueAMgMiner, for mining representative frequent approximate subgraphs (FASs) from multi-graph collections. GenCloMgVEAM is an extension of MgVEAM (proposed in Section 5.1.2) for mining generalized closed FASs by including the generalize closed condition into the mining process. GenCloMgVEAM is able to mine traditional closed or maximal FASs, by setting the closed threshold δ to 0 or 1, respectively. CliqueAMgMiner is an extension of our proposed AMgMiner algorithm for mining clique FASs.

The performance of GenCloMgVEAM and CliqueAMgMiner, in terms of runtime and the number of representative FASs, over synthetic and real-world multi-graph collections was evaluated. From our results we can conclude that, by using GenCloMgVEAM and CliqueAMgMiner, the execution time required for mining all FASs is kept similar to MgVEAM and AMgMiner, respectively, but reducing the size of the FAS set. As it can be seen in Section 6.3, the generalized closed FASs are the best option for mining a smaller subset of FASs when the multi-graphs in the collection, and therefore the mined FASs are highly connected; while the clique FASs are the best option when the multi-graphs of the collection and therefore the FASs are less connected. Moreover, as it was expected, in most cases, the traditional closed patterns did not reduce the number of FASs as the generalized closed patterns.

In Appendix A, we present an example of how to use the FASs mined by the algorithms proposed in this chapter, comparing the results achieved by using representative FASs with the results obtained by using all FASs.

Chapter 7

Conclusions and Future Work

Frequent Approximate Subgraph (FAS) mining has been successfully addressed for simple-graphs; however, although multi-graphs having been used for representing entities in several applications (Cazabet et al., 2015; Goonetilleke et al., 2015; Hulianytsky and Pavlenko, 2015; Setak et al., 2015; Terroso-Saez et al., 2015; Wang et al., 2015; Wei et al., 2015; Youssef et al., 2015; Verma and Bharadwaj, 2017), before this Ph.D. research, there were no algorithm for mining FASs from multi-graph collections.

For solving the lack of multi-graph FAS mining algorithms, in this Ph.D. research, we first proposed a method (called allEdges) based on graph transformations, which allows the application algorithms designed for mining all FASs from simple-graph collections to mine multi-graph FASs. After, looking for speeding up the mining process, we proposed another method (called onlyMulti), also based on graph transformations, which is faster than allEdges, but it does not mine all FASs from a multi-graph collection. Although using our proposed methods allows the application of any traditional FAS miner (e.g. APGM (Jia et al., 2011), RAM (Zhang and Yang, 2008), REAFUM (Li and Wang, 2015)), in both cases, the graph transformation process increases the size of the graphs in the collections, which also increases the computational cost of the mining process. For this reason, we focused on directly mining FASs from multi-graph collections (i.e., without transformations). To this end, since the canonical form based on adjacency matrices (CAM canonical form) has been widely used for mining FASs, we decided to use it for developing a new algorithm. However, the CAM canonical form was designed for simple-graphs, therefore, in this work, we extended this canonical form to allow representing isomorphic multi-graphs. After, using the extended version of the CAM canonical form, we proposed the MgVEAM algorithm for directly mining all FASs from multi-graph collections. Despite, the CAM canonical form having been widely

used in FAS mining, in the context of mining exact graph patterns, those algorithms based on the depth-first search canonical form (DFS canonical form) have reported better performance than those based on the CAM canonical form. Thus, we also extended the DFS canonical form for representing isomorphic multi-graphs and, based on this extension, we proposed the AMgMiner algorithm for directly mining all FASs from multi-graph collections. AMgMiner is faster than MgVEAM, but MgVEAM requires less memory than AMgMiner for mining all multi-graph FASs. It is important to highlight that with our proposed FAS miners (allEdges, onlyMulti, MgVEAM and AMgMiner) the first specific objective of this Ph.D. research about proposing algorithms for FAS mining in multi-graph collections, was achieved.

On the other hand, when a FAS miner is applied, it is common to obtain a large number of patterns. Therefore, in order to reduce the number of mined patterns, we proposed the GenCloMgVEAM algorithm, which is an extension of MgVEAM for mining generalized closed FASs. For developing GenCloMgVEAM, we decided to extend MgVEAM instead of AMgMiner because verifying if a pattern is a generalized closed FAS does not add a high computational cost to the MgVEAM mining process, as it would do to the AMgMiner's (as it was explained in Chapter 6). GenCloMgVEAM is able to mine both maximal and traditional closed FASs from multi-graph collections by fixing the closed threshold (δ) to 1 or 0, respectively. In this way, we fulfilled the second and third specific objectives of this research, which respectively consist in proposing algorithms for mining maximal and closed FASs from multi-graph collections.

Finally, we proposed an algorithm (called CliqueAMgMiner), which is an extension of AMgMiner, for mining clique FASs from multi-graphs collections. For developing CliqueAMg-Miner, we decided to extend AMgMiner because it is faster than MgVEAM and verifying the clique condition adds the same computational cost to both MgVEAM and AMgMiner. With CliqueAMgMiner we fulfill the fourth specific objective of this research about proposing an algorithm for computing clique FASs from multi-graph collections.

With all the algorithms (allEdges, onlyMulti, MgVEAM, AMgMiner, GenCloMgVEAM

and CliqueAMgMiner) proposed in this research, the general aim about proposing algorithms for mining representative FASs from multi-graph collections, was successfully achieved.

In the following sections, we present the conclusions, contributions and publications derived from this Ph.D. research. Finally, we discuss some future research directions.

7.1 Conclusions

Regarding our proposed methods based on graph transformations for mining multi-graph FASs, we conclude that:

- It is possible to mine multi-graph FASs from multi-graph collections with simple-graph FAS miners by means of graphs transformations.
- onlyMulti is faster than allEdges but the first one does not mine all multi-graph FASs, as the last one does.
- Both allEdges and onlyMulti allow applying other simple-graph FAS miners for mining FASs over multi-graph collections.
- Both allEdges and onlyMulti can be successfully used in real-world applications where the number of edges is less than 50 and the number of multi-graphs in the collection is not greater than 5000.

From our algorithms (MgVEAM and AMgMiner) proposed for directly mining multi-graph FASs from multi-graph collections, we conclude that:

- It is possible to mine all multi-graph FASs directly from multi-graph collections.
- MgVEAM and AMgMiner mine all FASs from multi-graph collections more efficiently, in terms of runtime and memory, than our methods based on graph transformations.

- AMgMiner is faster than MgVEAM, but MgVEAM requires less memory than AMgMiner for mining all multi-graph FASs.
- The extended CAM and DFS canonical forms can be used for reducing the number of comparisons at developing new algorithms for mining FASs from multi-graphs.
- In real-world applications where the multi-graph collections are highly connected, MgVEAM could be the best option, because the CAM canonical form requires less memory than the DFS one to represent isomorphic multi-graphs; otherwise, AMgMiner is the best option.
- Both MgVEAM and AMgMiner can be successfully used in real-world applications where the number of edges is less than 80 and the number of multi-graphs is slightly greater than 5000.

Regarding our proposed algorithms for mining representative FASs from multi-graph collections, we conclude that:

- It is possible to mine closed, maximal and clique FASs directly from multi-graph collections.
- GenCloMgVEAM requires the same runtime as MgVEAM, showing that verifying if a pattern is a generalized closed FAS adds a very little computational cost.
- CliqueAMgMiner requires more time than AMgMiner, but the additional time is not too long because verifying if a pattern is a clique FAS does not add a very high computational cost.
- Both GenCloMgVEAM and CliqueAMgMiner, as MgVEAM and AMgMiner, can be successfully used in real-world applications where the number of edges is less than 80 and the number of multi-graphs is slightly greater than 5000.
- It is recommended to use GenCloMgVEAM for reducing the number of FASs when the multi-graph of the collection are highly connected.

- CliqueAMgMiner is the best option when the collection has lowly connected multi-graphs.

7.2 Contributions

The contributions of this Ph.D. research are:

- The allEdges method, based on graph transformations, for mining all multi-graph FASs on multi-graph collections (Acosta-Mendoza et al., 2015c).
- The onlyMulti method, based on graph transformations, for mining some FASs on multi-graph collections (Acosta-Mendoza et al., 2015a).
- An extension of the CAM canonical form for representing isomorphic multi-graphs (Acosta-Mendoza et al., 2017a).
- An extension of the DFS canonical form for representing isomorphic multi-graphs (Acosta-Mendoza et al., 2016a).
- The MgVEAM algorithm, based on the extended CAM canonical form, for directly mining all FASs from multi-graph collections (Acosta-Mendoza et al., 2017a).
- The AMgMiner algorithm, based on the extended DFS canonical form, for directly mining all FASs from multi-graph collections (Acosta-Mendoza et al., 2016a).
- The GenCloMgVEAM algorithm for mining generalized closed FASs from multi-graph collections (Acosta-Mendoza et al., 2017b).
- The CliqueAMgMiner algorithm for mining clique FASs from multi-graph collections (we are working on a paper for reporting this result).

7.3 Publications

The contributions of this Ph.D. research were published in the following papers:

JCR Journals:

- N. Acosta-Mendoza *et al.* **Extension of Canonical Adjacency Matrices for Frequent Approximate Subgraph Mining on Multi-graph Collections**. *International Journal of Pattern Recognition and Artificial Intelligence*, 32(8): 1-25, 1750025, 2017 (Acosta-Mendoza et al., 2017a).
- N. Acosta-Mendoza *et al.* **A New Algorithm for Approximate Pattern Mining in Multi-graph Collections**. *Knowledge-Based Systems*, 109: 198-207, 2016 (Acosta-Mendoza et al., 2016a).

Conference proceedings:

- N. Acosta-Mendoza *et al.* **Mining Generalized Closed Patterns from Multi-graph Collections**. Accepted in: *22nd Iberoamerican Congress of Pattern Recognition (CIARP'2017)*, LNCS 10657, p. 1-9, 2017 (Acosta-Mendoza et al., 2017b).
- N. Acosta-Mendoza *et al.* **A New Method Based on Graph Transformation for FAS Mining in Multi-graph Collections**. *7th Mexican Conference on Pattern Recognition (MCPR'2015)*, LNCS 9116, p. 13-22, 2015 (Acosta-Mendoza et al., 2015c).

Technical Reports and other publications:

- N. Acosta-Mendoza *et al.* **Minería de subgrafos frecuentes aproximados cerrados en colecciones de multi-grafos**. *Technical Report RT_038*, Advanced Technologies Application Center (CENATAV), p. 1-23, 2017 (Acosta-Mendoza et al., 2017c).
- N. Acosta-Mendoza *et al.* **Representative Frequent Approximate Subgraph Mining in Multi-Graph Collections**. *Technical Report CCC-15-001*, Instituto Na-

cional de Astrofísica, Óptica y Electrónica (INAOE), p. 1-41, 2015 (Acosta-Mendoza et al., 2015b).

- N. Acosta-Mendoza *et al.* **Minería de subgrafos frecuentes aproximados para multi-grafos basada en transformaciones**. *XIII National Congress on Pattern Recognition of Cuba (RECPAT'2015)*, p. 1-8, 2015 (Acosta-Mendoza et al., 2015a).

- N. Acosta-Mendoza *et al.* **Representative Pattern Mining in Graph Collections**. *Research in Computing Science*, 71: 3-12, 2014 (Acosta-Mendoza et al., 2014).

7.4 Future Work

An immediate line for future research is speeding up the FAS mining process, where some kind of vectorial or parallel processing could be used for creating more efficient FASs mining algorithms. In some real-world applications such as social network analysis, multi-graph representations take into account edge directions, resulting in directed multi-graphs. Therefore, extending our proposed algorithms for mining FASs on directed multi-graphs collections is another possible future research line. Furthermore, in these social network applications, it is common to represent the problem by using a single multi-graph, then mining representative FASs from a single multi-graph is another task that is worth to be studied.

Bibliography

Abdelhamid, E., Abdelaziz, I., Kalnis, P., Khayyat, Z., and Jamour, F. (2016). ScaleMine: Scalable Parallel Frequent Subgraph Mining in a Single Large Graph. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '16, pages 61:1–61:12, Piscataway, NJ, USA. IEEE Press.

Acosta-Mendoza, N. (2013). Clasificación de imágenes basada en subconjunto de subgrafos frecuentes aproximados. Master's thesis, The National Institute of Astrophysics, Optics and Electronics of Mexico (INAOE).

Acosta-Mendoza, N., Alonso, A. G., Medina-Pagola, J., Carrasco-Ochoa, J., and Martínez-Trinidad, J. (2015a). Minería de subgrafos frecuentes aproximados para multi-grafos basada en transformaciones. In *XIII Congreso Nacional de Reconocimiento de Patrones (RECPAT)*, pages 1–8, Santiago de Cuba, Cuba.

Acosta-Mendoza, N., Carrasco-Ochoa, J., Gago-Alonso, A., Martínez-Trinidad, J., and Medina-Pagola, J. (2015b). Representative Frequent Approximate Subgraph Mining in Multi-Graph Collections. Technical Report CCC-15-001, Instituto Nacional de Astrofísica, Óptica y Electrónica, Mexico.

Acosta-Mendoza, N., Carrasco-Ochoa, J., Martínez-Trinidad, J., Gago-Alonso, A., and Medina-Pagola, J. (2014). Representative Pattern Mining in Graph Collections. *Research in Computing Science*, 71:3–12.

Acosta-Mendoza, N., Carrasco-Ochoa, J., Martínez-Trinidad, J., Gago-Alonso, A., and Medina-Pagola, J. (2015c). A New Method Based on Graph Transformation for FAS Mining in Multi-graph Collections. In *The 7th Mexican Conference on Pattern Recognition (MCPR'2015), Pattern Recognition*, volume LNCS 9116, pages 13–22. Springer.

Acosta-Mendoza, N., Gago-Alonso, A., Carrasco-Ochoa, J., Martínez-Trinidad, J., and Medina-Pagola, J. (2013). Feature Space Reduction for Graph-Based Image Classification. In *Proceedings of the 18th Iberoamerican Congress on Pattern Recognition (CIARP'13)*, volume Part I, LNCS 8258, pages 246–253, Havana, Cuba. Springer-Verlag Berlin Heidelberg.

Acosta-Mendoza, N., Gago-Alonso, A., Carrasco-Ochoa, J., Martínez-Trinidad, J., and Medina-Pagola, J. (2016a). A New Algorithm for Approximate Pattern Mining in Multi-graph Collections. *Knowledge-Based Systems*, 109:198–207.

Acosta-Mendoza, N., Gago-Alonso, A., Carrasco-Ochoa, J., Martínez-Trinidad, J., and Medina-Pagola, J. (2016b). Improving Graph-Based Image Classification by using Emerging Patterns as Attributes. *Engineering Applications of Artificial Intelligence*, 50:215–225.

Acosta-Mendoza, N., Gago-Alonso, A., Carrasco-Ochoa, J., Martínez-Trinidad, J., and Medina-Pagola, J. (2017a). Extension of Canonical Adjacency Matrices for Frequent Approximate Subgraph Mining on Multi-graph Collections. *International Journal on Pattern Recognition and Artificial Intelligence*, 31(7):25.

Acosta-Mendoza, N., Gago-Alonso, A., Carrasco-Ochoa, J., Martínez-Trinidad, J., and Medina-Pagola, J. (2017b). Mining Generalized Closed Frequent Approximate Subgraphs from Multi-graph Collections. In *Proceedings of the 22th Iberoamerican Congress on Pattern Recognition (CIARP'17)*, volume LNCS 10657, pages 1–9, Valparaiso, Chile. Springer International Publishing AG, part of Springer Nature.

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

Acosta-Mendoza, N., Gago-Alonso, A., and Medina-Pagola, J. (2012b). On speeding up frequent approximate subgraph mining. In *Proceedings of the 17th Iberoamerican Congress on Pattern Recognition (CIARP'12)*, volume LNCS 7441, pages 316–323. Buenos Aires, Argentina, Springer-Verlag Berlin Heidelberg.

Acosta-Mendoza, N., Gago-Alonso, A., Medina-Pagola, J., Carrasco-Ochoa, J., and Martínez-Trinidad, J. (2017c). Minería de subgrafos frecuentes aproximados cerrados en colecciones de multi-grafo. Technical Report *RT*_038, Serie Gris, CENATAV - DATYS, Havana, Cuba.

Acosta-Mendoza, N., Morales-González, A., Gago-Alonso, A., García-Reyes, E., and Medina-Pagola, J. (2012c). Classification using frequent approximate subgraphs. In *Proceedings of the 17th Iberoamerican Congress on Pattern Recognition (CIARP'12)*, volume LNCS 7441, pages 292–299. Buenos Aires, Argentina, Springer-Verlag Berlin Heidelberg.

Alam, T., Zahin, S., Samiullah, M., and Ahmed, C. (2017). *An Efficient Approach for Mining Frequent Subgraphs*, pages 486–492. Springer International Publishing, Cham.

Aoun, N., M.M., and Amar, C. (2014). Bag of sub-graphs for video event recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP, Florence, Italy,*, pages 1547–1551.

Appel, A. and Moyano, L. (2017). Link and Graph Mining in the Big Data Era. *Handbook of Big Data Technologies*, pages 583–616. Springer International Publishing.

Boley, M., Horváth, T., and Wrobel, S. (2009). Efficient discovery of interesting patterns based on strong closedness. *Statistical Analysis and Data Mining*, 2(5-6):346–360.

Borgelt, C. (2002). Mining molecular fragments: Finding relevant substructures of molecules. In *Proc. IEEE International Conference on Data Mining (ICDM)*, pages 51–58, Maebashi City, Japan. IEEE Press.

Borgelt, C. and Meinl, T. (2009). Full perfect extension pruning for frequent subgraph mining. *Mining Complex Data / Zighed, Djamel A. et al. (ed.). - Berlin : Springer, 2009. - (Studies in Computational Intelligence*, 165:189–205.

Bringmann, B., Nijssen, S., and Zimmermann, A. (2011). Pattern-Based Classification: A Unifying Perspective. *arXiv preprint arXiv:1111.6191*, page 15.

Brun, L. and Kropatsch, W. (2000). Introduction to Combinatorial Pyramids. In *Digital and Image Geometry*, volume 2243 of *LNCS*, pages 108–128. Springer.

Caiyan, D. and Ling, C. (2016). An algorithm for mining frequent closed itemsets with density from data streams. *International Journal of Computational Science and Engineering*, 12(2–3):146–154.

Cavique, L., Mendes, A., and Santos, J. (2009). An Algorithm to Discover the k-Clique Cover in Networks. In Lopes, L., Lau, N., Mariano, P., and Rocha, L., editors, *Progress in Artificial Intelligence: 14th Portuguese Conference on Artificial Intelligence, EPIA 2009, Aveiro, Portugal, October 12-15, 2009. Proceedings*, LNCS 5816, pages 363–373, Berlin, Heidelberg. Springer Berlin Heidelberg.

Cazabet, R., Takeda, H., and Hamasaki, M. (2015). Characterizing the nature of interactions for cooperative creation in online social networks. *Social Network Analysis and Mining*, 5(1):1–17.

Chalupa, D. (2016). On Combinatorial Optimisation in Analysis of Protein-Protein Interaction and Protein Folding Networks. In *Applications of Evolutionary Computation (1)*, volume 9597 of *LNCS*, pages 91–105. Springer.

Chen, C., Lin, C., Yan, X., and Han, J. (2008). On effective presentation of graph patterns: a structural representative approach. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management, CIKM 2008, Napa Valley, California, USA, October 26-30, 2008*, pages 299–308.

Chen, C., Yan, X., Zhu, F., and Han, J. (2007). gApprox: Mining Frequent Approximate Patterns from a Massive Network. In *International Conference on Data Mining (ICDM'07)*, pages 445–450.

Chen, Q., Fang, C., Wang, Z., Suo, B., Li, Z., and Ives, Z. (2016). Parallelizing Maximal Clique Enumeration Over Graph Data. In *Database Systems for Advanced Applications - 21st International Conference, DASFAA 2016, Dallas, TX, USA, April 16-19, 2016, Proceedings, Part II*, pages 249–264.

Chen, X., Zhang, C., Liu, F., and Guo, J. (2012). Algorithm research of top-down mining maximal frequent subgraph based on tree structure. In *Snac, P., Ott, M., Seneviratne, A. (Eds.), Wireless Communications and Applications.*, volume 72, pages 401–411. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer Berlin Heidelberg.

Cheng, J., Ke, Y., and Ng, W. (2006). δ-Tolerance Closed Frequent Itemsets. In *6th International Conference on Data Mining (ICDM'06)*, pages 139–148, Hong Kong. IEEE.

Cheng, J., Ke, Y., and Ng, W. (2008). Effective elimination of redundant association rules. *Data mining and knowledge discovery*, 16(2):221–249.

Conte, D., Foggia, P., Sansone, C., and Vento, M. (2004). Thirty years of graph matching in pattern recognition. *IJPRAI*, 18(3):265–298.

Cook, D. and Holder, L. (1994). Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research*, 1:231–255.

Cormen, T., Leiserson, C., Rivest, R., and Stein, C. (2001). *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts Lonfon, England, 2th edition.

Demetrovics, J., Quang, H., Anh, N., and Thi, V. (2017). An optimization of closed frequent subgraph mining algorithm. *Cybernetics and Information Technologies*, 17(1):1–13.

Deo, N. (2017). *Graph Theory with Applications to Engineering and Computer Science*. Dover Publications.

Deore, V., Kamble, P., Bendkule, R., Dhatrak, M., and Jadhav, S. (2017). Document Recommendation using Boosting Based Multi-graph Classification: A Review. *International Research Journal of Engineering and Technology (IRJET)*, 04(02):962–964.

Diestel, R. (2012). *Graph Theory*. Graduate texts in mathematics 173, Springer, Berlin, electronic, 4th edition.

Dinari, H. and Naderi, H. (2016). A method for improving graph queries processing using positional inverted index (p.i.i) idea in search engines and parallelization techniques. *Journal of Central South University*, 23(1):150–159.

El Islem Karabadji, N., Aridhi, S., and Seridi, H. (2016). *A Closed Frequent Subgraph Mining Algorithm in Unique Edge Label Graphs*, pages 43–57. Springer International Publishing, Cham.

Elseidy, M., Abdelhamid, E., Skiadopoulos, S., and Kalnis, P. (2014). GRAMI: Frequent Subgraph and Pattern Mining in a Single Large Graph. *PVLDB*, 7(7):517–528.

Emmert-Streib, F., Dehmer, M., and Shi, Y. (2016). Fifty years of graph matching, network alignment and network comparison. *Information Sciences*, pages 1–22.

Finkel, R. and Bentley, J. (1974). Quad Trees: A Data Structure for Retrieval on Composite Keys. *Acta Informatica*, 4:1–9.

Flores-Garrido, M., Carrasco-Ochoa, J., and Martínez-Trinidad, J. (2014). Mining maximal frequent patterns in a single graph using inexact matching. *Knowledge-Based Systems*, 66:166–177.

Flores-Garrido, M., Carrasco-Ochoa, J., and Martínez-Trinidad, J. (2015). AGraP: an algorithm for mining frequent patterns in a single graph using inexact matching. *Knowledge and Information Systems*, 42(2):1–22.

Gago-Alonso, A. (2010). *Minería de subgrafos conexos frecuentes en colecciones de grafos etiquetados*. PhD thesis, Instituto Nacional de Astrofísica, Óptica y Electrónica, Tonantzintla, Puebla, México.

Gago-Alonso, A. (2015). Connected Permutations of Vertices for Canonical Form Detection in Graph Mining. *Revista Cubana de Ciencias Informáticas*, 9:57–71.

Gago-Alonso, A., Carrasco-Ochoa, J. A., Medina-Pagola, J., and Martínez-Trinidad, J. (2010a). Full Duplicate Candidate Pruning for Frequent Connected Subgraph Mining. *Integrated Computer-Aided Engineering*, 17:211–225.

Gago-Alonso, A., Medina-Pagola, J., Carrasco-Ochoa, J., and Trinidad, J. M. (2008). Mining Frequent Connected Subgraphs Reducing the Number of Candidates. In *Machine Learning and Knowledge Discovery in Databases, European Conference, (ECML/PKDD)*, volume 5211 of *Lecture Notes in Computer Science*, pages 365–376. Springer.

Gago-Alonso, A., Puentes-Luberta, A., Carrasco-Ochoa, J., Medina-Pagola, J., and Martínez-Trinidad, J. (2010b). A new algorithm for mining frequent connected subgraphs based on adjacency matrices. *Intelligent Data Analysis*, 14:385–403.

Gao, L., Pan, H., Han, Q., Xie, X., Zhang, Z., Zhai, X., and Li, P. (2015). Finding Frequent Approximate Subgraphs in Medical Image Database. In *IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 1004–1007. IEEE.

Gao, X., Xiao, B., Tao, D., and Li, X. (2010). A survey of graph edit distance. *Pattern Analysis and Applications*, 13(1):113–129.

Gay, D., Selmaoui-Folcher, N., and Boulicaut, J. (2012). Application-independent feature construction based on almost-closedness properties. *Knowledge and information systems*, 30(1):87–111.

González, J., Holder, L., and Cook, D. (2001). Graph-Based Concept Learning. In *Proceedings of the Fourteenth International Florida Artificial Intelligence Research Society Conference*, pages 377–381, Key West, Florida, USA. AAAI Press.

Goonetilleke, O., Sathe, S., Sellis, T., and Zhang, X. (2015). Microblogging Queries on Graph Databases: An Introspection. In *Third International Workshop on Graph Data Management Experiences and Systems, (GRADES), Melbourne, VIC, Australia*, volume 5, pages 1–6.

Hahn, K., Massopust, P., and Prigarin, S. (2016). A new method to measure complexity in binary or weighted networks and applications to functional connectivity in the human brain. *BMC Bioinformatics*, 17(87):1–18.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. (2009). The WEKA Data Mining Software: An Update. *Special Interest Group on Knowledge Discovery and Data Mining (SIGKDD) Explorations*, 11:10–18.

Han, M., Zang, W., and Li, J. (2010). RANKING: An efficient K-maximal frequent pattern mining algorithm on uncertain graph database. *J Comput*, 33:1387–1395.

Hao, F., Park, D., Li, S., and Lee, H. (2016). Mining λ-Maximal Cliques from a Fuzzy Graph. *Sustainability*, 8(553):1–16.

Herrera-Semenets, V. and Gago-Alonso, A. (2017). A novel rule generator for intrusion detection based on frequent subgraph mining. *INGENIARE - Revista Chilena de Ingeniera*, 25(2):226 – 234.

Hlaing, Y. and Oo, K. (2016). A Graph Representative Structure for Detecting Automorphic Graphs. In *T.T. Zin et al. (eds.), Genetic and Evolutionary Computing: Proceedings of the Ninth International Conference on Genetic and Evolutionary Computing*, volume 1 of *LNCS*, pages 189–197. Springer International Publishing Switzerland.

Holder, L., Cook, D., and Bunke, H. (1992). Fuzzy substructure discovery. In *ML92: Proceedings of the ninth international workshop on Machine learning*, pages 218–223, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Hu, J., He, L., and Mao, Y. (2015). Research of improved mining frequent subgraph patterns in uncertain graph databases. *Computer Engineering and Applications*, 51:112–116.

Huan, J., Bandyopadhyay, D., Snoeyink, J., Prins, J., Tropsha, A., and Wang, W. (2006). Distance-based identification of spatial motifs in proteins using constrained frequent subgraph mining. In *Proceedings of the IEEE Computational Systems Bioinformatics (CSB)*, pages 227–238.

Huan, J., Wang, W., and Prins, J. (2003). Efficient mining of frequent subgraphs in the presence of isomorphism. In *The 3rd IEEE International Conference on Data Mining*, pages 549–552, FL. Melbourne.

Huan, J., Wang, W., Prins, J., and Yang, J. (2004). Spin: Mining maximal frequent subgraphs from graph databases. In *Proceedings of the tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Minin.*, pages 581–586. ACM.

Hulianytsky, L. and Pavlenko, A. (2015). Ant Colony Optimization for Time Dependent Shortest Path Problem in Directed Multigraph. *International Journal Information Content and Processing*, 2(1):50–61.

Inokuchi, A., Washio, T., Nishimura, K., and Motoda, H. (2002). A fast algorithm for mining frequent connected subgraphs. In *Technical Report RT0448, In IBM Research*, Tokyo Research Laboratory.

Jabeur, L., Tamine, L., and Boughanem, M. (2012). Active microbloggers: Identifying influencers, leaders and discussers in microblogging networks. In *String Processing and Information Retrieval – 19th International Symposium, Cartagena de Indias, Colombia*, volume 7608 of *LNCS*, pages 111–117.

Jia, J., Yu, N., Rui, X., and Li, M. (2008). Multi-Graph Similarity Reinforcement for Image Annotation Refinement. In *International Conference on Image Processing*, pages 993–996. December 29.

Jia, Y., Huan, J., Buhr, V., Zhang, J., and Carayannopoulos, L. (2009). Towards comprehensive structural motif mining for better fold annotation in the "twilight zone" of sequence dissimilarity. *BMC Bioinformatics*, 10(S-1):1–14.

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

Jiang, C., Coenen, F., and Zito, M. (2013). A survey of frequent subgraph mining algorithms. *Knowledge Engineering Review*, 28(1):75–105.

Jungnickel, D. (2012). *Graphs, Networks and Algorithms*. Algorithms and Computation in Mathematics. Springer Berlin Heidelberg.

Ketkar, N. S. (2005). Subdue: compression-based frequent pattern discovery in graph data. In *OSDM'05: Proceedings of the 1st international workshop on open source data mining*, pages 71–76. ACM Press.

Kimelfeld, B. and Kolaitis, P. (2013). The complexity of mining maximal frequent subgraphs. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013*, pages 13–24. ACM.

Kropatsch, W., Haxhimusa, Y., Pizlo, Z., and Langs, G. (2005). Vision pyramids that do not grow too high. *Pattern Recognition Letters*, 26:319–337.

Kuramochi, M. and Karypis, G. (2002). An efficient algorithm for discovering frequent subgraphs. Technical report, IEEE Transactions on Knowledge and Data Engineering.

Lartillot, O. (2015). Automated Motivic Analysis: An Exhaustive Approach Based on Closed and Cyclic Pattern Mining in Multidimensional Parametric Spaces. *Computational Music Analysis*, Part V:273–302.

Li, J., Zou, Z., and Gao, H. (2012). Mining frequent subgraphs over uncertain graph databases under probabilistic semantics. *VLDB J.*, 21(6):753–777.

Li, R. and Wang, W. (2015). REAFUM: Representative Approximate Frequent Subgraph Mining. In *SIAM International Conference on Data Mining*, pages 757–765, Vancouver, BC, Canada. SIAM.

Liu, M. and Gribskov, M. (2015). MMC-Marging: Identification of Maximum Frequent Subgraphs By Metropolis Monte Carlos Sampling. In *IEEE International Conference on Big Data*, pages 849–856. IEEE.

Liu, Y., Wang, Y., and Shang, X. (2014). An uncertain graph classification algorithm based on discriminative sub-graphs. *Journal of Shanxi Normal University (Natural Science Edition)*, 42:16–19.

Lu, C., Yu, J., Wei, H., and Zhang, Y. (2017). Finding the Maximum Clique in Massive Graphs. *Proceedings of the VLDB Endowment*, 10(11):1538–1549.

Manzo, M., Pellino, S., Petrosino, A., and Rozza, A. (2015). A novel graph embedding framework for object recognition. In *ECCV 2014 Workshops*, volume Part IV, LNCS 8928, pages 341–352. Springer International Publishing Switzerland.

Morales-González, A., Acosta-Mendoza, N., Gago-Alonso, A., García-Reyes, E., and Medina-Pagola, J. (2014). A new proposal for graph-based image classification using frequent approximate subgraphs. *Pattern Recognition*, 47(1):169–177.

Morales-González, A. and García-Reyes, E. B. (2010). Assessing the Role of Spatial Relations for the Object Recognition Task. In *The 15th Iberoamerican Congress on Pattern Recognition (CIARP'10)*, volume 6419 of *Lecture Notes in Computer Science*, pages 549–556. Springer, Heidelberg.

Morales-González, A. and García-Reyes, E. B. (2013). Simple object recognition based on spatial relations and visual features represented using irregular pyramids. *Multimedia tools and applications*, 63(3):875–897.

Moussaoui, M., Zaghdoud, M., and Akaichi, J. (2016). POSGRAMI: Possibilistic Frequent Subgraph Mining in a Single Large Graph. In Carvalho, P., Lesot, M., Kaymak, U., Vieira, S., Bouchon-Meunier, B., and Yager, R., editors, *Information Processing and Management of Uncertainty in Knowledge-Based Systems - 16th International Conference, IPMU 2016, Eindhoven, The Netherlands, June 20-24, 2016, Proceedings, Part I*, pages 549–561, Cham. Springer International Publishing.

Muñoz-Briseño, A., Lara-Alvarez, G., Gago-Alonso, A., and Hernández-Palancar, J. (2016). A Novel Geometric Graph Miner and its Applications. *Pattern Recognition Letters*, (84):208–214.

Nijssen, S. and Kok, J. (2004). A Quickstart in Frequent Structure Mining can make a Difference. In *The 10th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 647–652, New York, NY, USA. ACM.

O'Hara, S. and Draper, B. (2011). Introduction to the Bag of Features Paradigm for Image Classification and Retrieval. *Computing Research Repository (CoRR)*, abs/1101.3354.

Papalexakis, E., Akoglu, L., and Ienco, D. (2013). Do more Views of a Graph help? Community Detection and Clustering in Multi-Graphs. In *16th International Conference on Information Fusion, IEEE, Istanbul, Turkey*, pages 899–905.

Petermann, A., Junghanns, M., and Rahm, E. (2017). DIMSpan - Transactional Frequent Subgraph Mining with Distributed In-Memory Dataflow Systems. *Cornell University Library*, pages 1–17.

Rahman, M. (2017). *Basic Graph Theory*. Undergraduate Topics in Computer Science. Springer International Publishing.

Ramraj, T. and Prabhakar, R. (2015). Frequent subgraph mining algorithms - a survey. *Procedia Computer Science*, 47:197–204.

Read, R. and Corneil, D. (1977). The graph isomorph disease. *Journal of Graph Theory*, 1:339–363.

Riesen, K. and Bunke, H. (2008). IAM Graph Database Repository for Graph Based Pattern Recognition and Machine Learning. *Structural, Syntactic, and Statistical Pattern Recognition, Joint IAPR International Workshop, SSPR and SPR 2008*, pages 208–297.

Rousseau, F., Kiagias, E., and Vazirgiannis, M. (2015). Text categorization as a graph classification problem. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing*, volume 1, pages 1702–1712. Beijing, China.

Salma, M. (2016). An Efficient Algorithm for mining Frequent Pattern Growth without candidate Generation. *International Journal of Emerging Trends in Technology and Sciences*, 06(03):480–487.

Santhi, S. and Padmaja, P. (2015). A Survey of Frequent Subgraph Mining algorithms for Uncertain Graph Data. *International Research Journal of Engineering and Technology (IRJET)*, 2(2):688–696.

Segundo, P., Lopez, A., and Pardalos, P. (2016). A new exact maximum clique algorithm for large and massive sparse graphs. *Computers and Operations Research*, 66:81–94.

Senthilkumaran, B. and Thangadurai, K. (2017). A Comparative Study of Discovering Frequent Subgraphs - Approaches and Techniques. *International Journal of Computer Engineering In Research Trends*, 4(1):41–45.

Setak, M., Habibi, M., Karimi, H., and Abedzadeh, M. (2015). A time-dependent vehicle routing problem in multigraph with FIFO property. *Journal of Manufacturing Systems*, 35:37–45.

Shi, B. and Weninger, T. (2016). Discriminative predicate path mining for fact checking in knowledge graphs. *Knowledge-Based Systems, DOI: 10.1016/j.knosys.2016.04.015*.

Shu-Jing, L., Yi-Chung, C., Li-Don, Y., and Jungpin, W. (2016). Discovering Long Maximal Frequent Pattern. In *8th International Conference on Advanced Computational Intelligence*, pages 136–142, Chiang Mai, Thailand. IEEE.

Song, Y. and Chen, S. (2006). Item sets based graph mining algorithm and application in genetic regulatory networks. *Data Mining, IEEE International Conference on Volume, Issue*, pages 337–340.

Stikic, M., Larlus, D., and Schiele, B. (2009). Multi-graph Based Semi-supervised Learning for Activity Recognition. *International Symposium on Wearable Computers*, pages 85–92.

Takigawa, I. and Mamitsuka, H. (2011). Efficiently Mining δ-tolerance Closed Frequent Subgraphs. *Machine Learning*, 82(2):95–121.

Terroso-Saez, F., Valdés-Vela, M., and Skarmeta-Gómez, A. (2015). Online Urban Mobility Detection Based on Velocity Features. In *17th International Conference of Big Data Analytics and Knowledge Discovery, Valencia, Spain*, volume 9263 of *LNCS*, pages 351–362.

Thomas, L., Valluri, S., and Karlapalem, K. (2009). ISG: Itemset based subgraph mining. *Technical Report*, IIIT, Hyderabad.

Thomas, L., Valluri, S., and Karlapalem, K. (2010). Margin: Maximal frequent subgraph mining. *ACM Trans. Knowl. Discov. Data*, 4:10:1–10:42.

Tsourakakis, C. (2014). A Novel Approach to Finding Near-Cliques: The Triangle-Densest Subgraph Problem. *CoRR*, abs1405.1477.

Unil, Y. and Gangin, L. (2016). Incremental mining of weighted maximal frequent itemsets from dynamic databases. *Expert Systems With Applications*, 54:304–327.

Verma, A. and Bharadwaj, K. (2017). Identifying community structure in a multi-relational network employing non-negative tensor factorization and GA k-means clustering. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 7(1):1–22.

Wang, C., Wang, W., Pei, J., Zhu, Y., and Chi, B. (2004). Scalable Mining of Large Disk-based Graph Databases. In *Proc. of the 2004 ACM SIGKDD of International Conference on Knowledge Discovery in databases (KDD)*, pages 316–325. Seattle, WA.

Wang, H., Ma, W., Shi, H., and Xia, C. (2015). An Interval Algebra-based Modeling and Routing Method in Bus Delay Tolerant Network. *KSII Transactions on Internet and Information Systems*, 9(4):1376–1391.

Wang, K., Xie, X., Jin, H., Yuan, P., Lu, F., and Ke, X. (2016). Frequent Subgraph Mining in Graph Databases Based on MapReduce. In Wang, G., Han, Y., and Martínez, G., editors, *Advances in Services Computing - 10th Asia-Pacific Services Computing Conference, APSCC 2016, Zhangjiajie, China, November 16-18, 2016, Proceedings*, pages 464–476, Cham. Springer International Publishing.

Wang, M., Hua, X., Yuan, X., Song, Y., and Dai, L. (2007a). Multi-Graph Semi-Supervised Learning for Video Semantic Feature Extraction. *International Conference on Multimedia and Expo*, pages 1978–1981.

Wang, M., Hua, X., Yuan, X., Song, Y., and Dai, L. (2007b). Optimizing Multi-Graph Learning: Towards A Unified Video Annotation Scheme. In *ACM Multimedia*, pages 862–871. September 23.

Wang, W. and Li, J. (2013). MUSIC: An Efficient Algorithm of Mining Frequent Subgraph Patterns in Uncertain Graph Databases. *Intelligent Computer and Applications*, 3:20–23.

Wei, D., Liu, H., and Qin, Y. (2015). Modeling cascade dynamics of railway networks under inclement weather. *Transportation Research Part E*, 80:95–122.

Wu, D., Ren, J., and Sheng, L. (2017). Uncertain maximal frequent subgraph mining algorithm based on adjacency matrix and weight. *International Journal of Machine Learning and Cybernetics*, pages 1–11.

Wu, J., Zhu, X., Zhang, C., and Cai, Z. (2013). Multi-instance Multi-graph Dual Embedding Learning. In *13th International Conference on Data Mining*, pages 827–836. January 1.

Wu, J., Zhu, X., Zhang, C., and Yu, P. (2014). Bag Constrained Structure Pattern Mining for Multi-Graph Classification. *IEEE Transactions on Knowledge and Data Engineering*, 26(10):2382–2396.

Xiao, Y., Wang, W., and Wu, W. (2007). Mining conserved topological structures from large protein-protein interaction networks. In *Proceedings of the 18th IEICE data engineering workshop / 5th DBSJ annual meeting*, pages 1–8, Hiroshima, Japan. DEWS'2007.

Xiao, Y., Wu, W., Wang, W., and He, Z. (2008). Efficient Algorithms for Node Disjoint Subgraph Homeomorphism Determination. In *Proceedings of the 13th international conference on Database systems for advanced applications*, pages 452–460, New Delhi, India. Springer-Verlag, Berlin, Heidelberg.

Xu, Y., Cheng, J., Fu, A. W.-C., and Bu, Y. (2014). Distributed Maximal Clique Computation. In *2014 IEEE International Congress on Big Data*, pages 160–167. IEEE.

Yan, X. and Han, J. (2002). gSpan: Graph-Based Substructure Pattern Mining. In *International Conference on Data Mining*, Japan. Maebashi.

Yan, X. and Han, J. (2003). ClosedGraph: Mining Closed Frequent Graph Patterns. In *Proc. of the 9th ACM SIGKDD of International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 286–295. Washington, DC.

Yan, X., Han, J., and Afshar, R. (2003). CloSpan: Mining closed sequential patterns in large datasets. In *3rd SIAM International Conference on Data Mining*, pages 166–177, San Franciso, USA. SIAM.

Yi-Cheng, C., Weng, J. T.-Y., and Lin, H. (2016). A novel algorithm for mining closed temporal patterns from interval-based data. *Knowledge and Information Systems*, 46(1):151–183.

Youssef, R., Kacem, A., Sevestre-Ghalila, S., and Chappard, C. (2015). Graph Structuring of Skeleton Object for Its HighLevel Exploitation. In *Image Analysis and Recognition - 12th International Conference, ICIAR 2015, Niagara Falls, ON, Canada, July 22-24, 2015, Proceedings*, volume 9164 of *LNCS*, pages 419–426. Springer.

Zhang, S. and Yang, J. (2008). RAM: Randomized Approximate Graph Mining. In *The 20th International Conference on Scientific and Statistical Database Management*, pages 187–203, China. Hong Kong.

Zhang, S., Yang, J., and Cheedella, V. (2007). Monkey: Approximate Graph Mining Based on Spanning Trees. In *International Conference on Data Engineering*, pages 1247–1249, Los Alamitos, CA, USA. IEEE ICDE.

Zhu, F., Yan, X., Han, J., and Yu, P. (2007). gPrune: A Constraint Pushing Framework for Graph Pattern Mining. In *Advances in Knowledge Discovery and Data Mining, 11th Pacific–Asia Conference, PAKDD 2007, Nanjing, China, May 22–25, Proceedings*, volume 4426 of *LNCS*, pages 388–400. Springer.

Zou, Z., Li, J., Gao, H., and Zhang, S. (2010a). Finding top-k maximal cliques in an uncertain graph. In *IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pages 649–652.

Zou, Z., Li, J., Gao, H., and Zhang, S. (2010b). Mining frequent subgraph patterns from uncertain graph data. *IEEE Trans. on Knowl. and Data Eng.*, 22(9):1203–1218.

APPENDIX A

A.1 Using Multi-Graph FASs

Several researchers have used multi-graphs for representing images on image classification problems (Brun and Kropatsch, 2000; Kropatsch et al., 2005; Morales-González and García-Reyes, 2010; Acosta-Mendoza et al., 2012a; Morales-González and García-Reyes, 2013; Morales-González et al., 2014). For this reason, we decide to show the use of multi-graph FASs for image classification. It is important to highlight that this appendix was added just to show how the multi-graph FASs mined by our proposals can be used in a specific context.

A.1.1 Image Classification based on FASs

For our experiments, we use the graph-based image classification method proposed in (Acosta-Mendoza et al., 2012a), which is based on FASs, but we will use multi-graphs instead of simple-graphs. Given a set of images, each image is represented as a multi-graph and we apply the multi-graph FASs mining algorithms proposed in this thesis over a training set. Then, the mined multi-graph FASs are used as attributes for building a vectorial representation of the images. Using this vectorial representation, a traditional classifier is built. Thus, each new image is represented as a vector by using the FASs obtained from the training set; and it is classified by the trained classifier. In Figure A.1, we show the workflow of the image classification method used for our experiments.

For representing an image as a multi-graph, we used a quad-tree approach (Finkel and Bentley, 1974). We represent each image as it was proposed in (Acosta-Mendoza et al., 2012a), but using more than one relation among vertices. First, we obtain a quad-tree from the image by recursively dividing it in quadrants; stopping when a uniform color or a predefined number (4 for our experiments) of levels is reached (see Figure A.2(a-b)). Then, each leaf of the quad-tree is represented as a vertex of a multi-graph; where the most frequent color, in the corresponding quadrant, is used as label (see Figure A.2(c)). Vertices corresponding to adjacent quadrants are connected by two edges labeled as follows. For one edge, the smallest angle formed between the line that connects the centers of the two adjacent quadrants and the horizontal axis is used as label. An example of this angle is shown in Figure A.3, where we

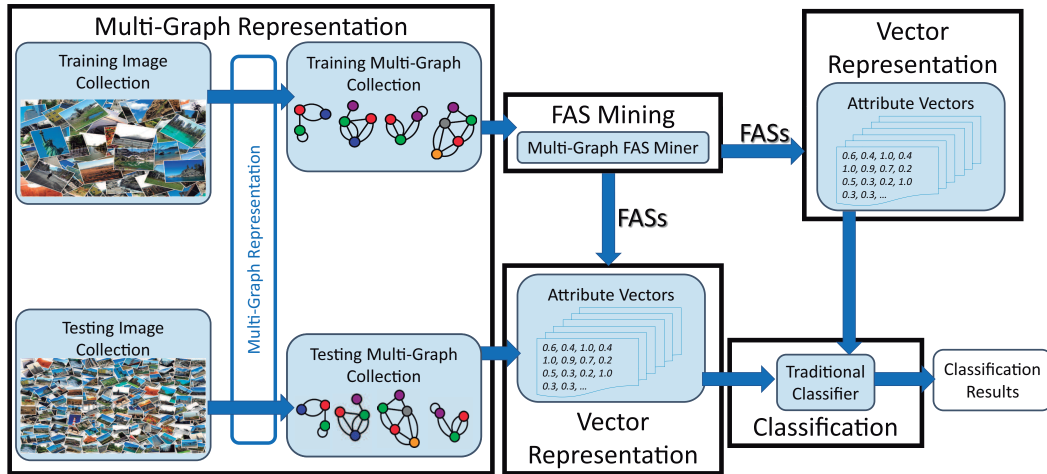


Figure A.1: Workflow of the image classification method based on multi-graph FASs.

have two angles (α and β) between the horizontal line and the line connecting the quadrants. In this example, α is selected as the edge label, since $\alpha < \beta$. For the other edge, the distance between the centers of the two adjacent quadrants is used as label; d in Figure A.3(b). Since there could be a large number of different values for these edge labels, the angle and distance values were discretized into 24 equal bins for each one, as it was suggested in (Acosta-Mendoza et al., 2012a).

Once all training images have been represented as multi-graphs, our proposed algorithms are used for mining FASs. Then, following the idea of the bag of words model (O'Hara and Draper, 2011), the mined FASs are used as words, and each image is represented as a vector, where each element contains the approximate frequency of a FAS into the image. Next, using this vectorial representation, a traditional classifier is trained. After, for classifying a new image, firstly it is represented as a multi graph. Then, through the patterns mined from the training set, the new image is represented; sorting the patterns as in the training set. This representation allows to obtain a vectorial representation for the new image similar to that built for the training set. Finally, a supervised classifier decides the class of the image.

A.1.2 Experiments and Results

Following the classification method described in Section A.1.1, we perform two experiments. In the first experiment, we show how multi-graph FASs can be used for image classification. In the second experiment, we show the use of the representative multi-graph FASs mined by our algorithms. All our experiments were carried out on a personal computer with an Intel(R) Core(TM) i5-3317U CPU @ 1.70 GHz with 4 GB of RAM. All the algorithms were implemented in ANSI-C and executed on Microsoft Windows 10.

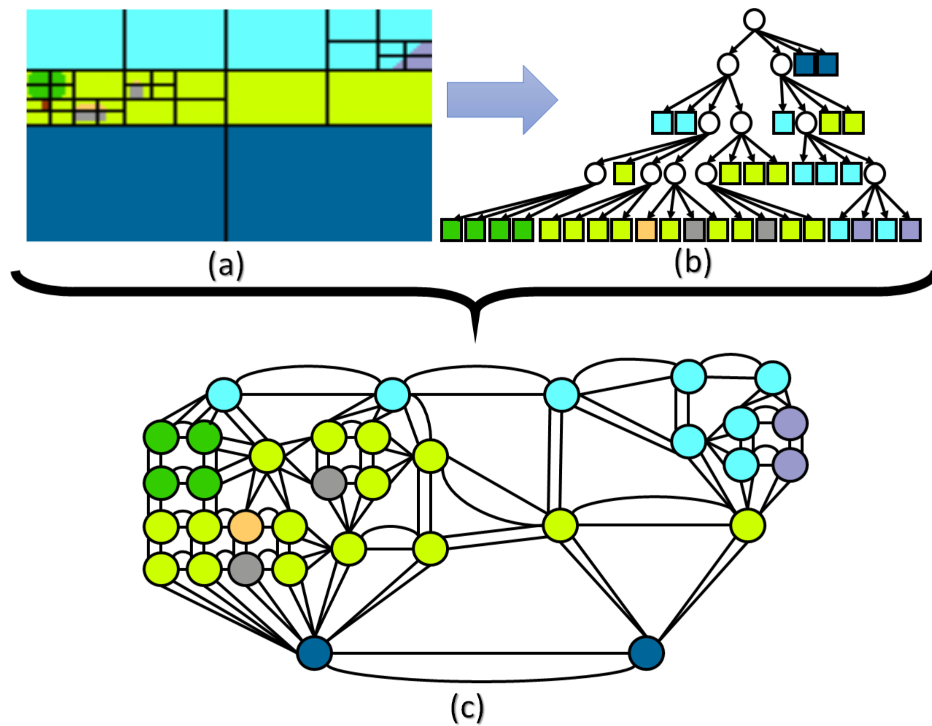


Figure A.2: Example of the multi-graph representation for an image.

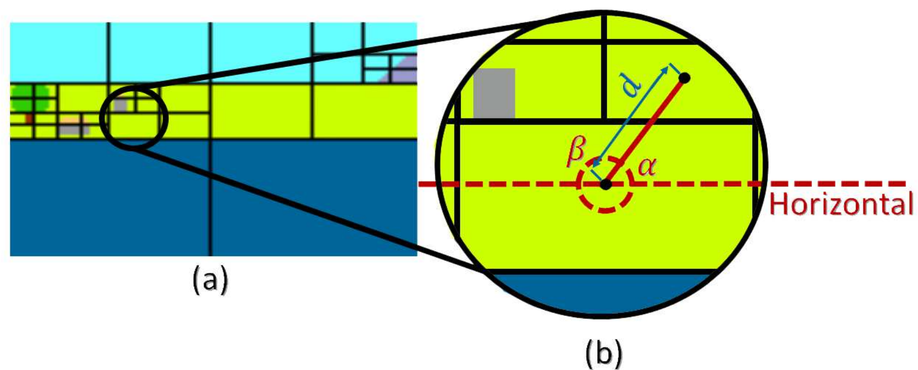


Figure A.3: Example of the angles formed between the line that connects the centers of the two adjacent quadrants and the horizontal axis in the multi-graph representation for an image.

A.1.2.1 Graph Collections

In our experiments, an image dataset generated with the Random image generator of Coenen[1] is used (Coenen image dataset). Coenen image dataset contains 6000 images distributed in two classes, which were randomly divided into two sub-sets: one for training with 3500 images

[1] www.csc.liv.ac.uk/∼frans/KDD/Software/ImageGenerator/imageGenerator.html

(70%) and another for testing with 1800 images (30%).

Following the graph representation approach described in Section 7.4, we represent the Coenen image dataset as three graph collections. The first one, named *Coenen-Multi*, contains the obtained multi-graphs. The other two collections contain simple-graphs obtained by using only the edges labeled with angles (*Coenen-Angle*), and only the edges labeled with distances (*Coenen-Distance*).

In Table 1, the characteristics of the graph collections used in our experiment are presented. In this table, the first column shows the collection identifier. The other five columns show the number of graphs in the collection, the number of vertex labels, the number of edge labels, the average number of vertices per graph, and the average number of edges per graph, respectively.

Table 1: Graph collections used in our experiments.

Collection	$\lvert D\rvert$	$\lvert L_V\rvert$	$\lvert L_E\rvert$	**Average** $\lvert V\rvert$	**Average** $\lvert E\rvert$
Coenen-Angle	6000	21	24	9	13
Coenen-Distance	6000	21	24	9	13
Coenen-Multi	6000	21	48	9	26

A.1.2.2 Multi-Graph FASs vs. Simple-Graph FASs

In this experiment, we compare the use of multi-graphs versus the use of simple-graphs, where AMgMiner was applied over Coenen-Multi, and VEAM over Coenen-Angle and Coenen-Distance. Then, the classification results obtained by using multi-graph FASs are compared with the ones obtained by using simple-graph FASs. For this experiment, two well-known classifiers were used: SVM with a polynomial kernel and J48graft; taken from Weka v3.6 (Hall et al., 2009) using the default parameters.

In Table 2, we show the classification results achieved by using as attributes simple-graph FASs mined over Coenen-Angle and Coenen-Distance, and multi-graph FASs mined over Coenen-Multi. In this experiment, we use the two substitution matrices and the similarity threshold ($\tau = 0.4$) suggested in (Acosta-Mendoza et al., 2012a) for this image collection.

Table 2 is composed by three sub-tables with the Accuracy, F-measure and Area under ROC results, respectively. The highest results per row are highlighted in bold. The first column of each sub-table shows the values used for the support threshold (0.3, 0.4 or 0.5). The other three consecutive columns show the classification results in the collection specified on the top of these columns. Each one of these columns shows the classification (accuracy, F-measure or area under ROC) results obtained using SVM or J48graft.

As we can see, in Table 2, an improvement in the classification results is achieved when multi-graph FASs are used. This improvement shows that our proposal is able to provide good results for image classification, and also reveals the usefulness of multi-graph representations.

Table 2: Classification results (%) achieved using simple-graph and multi-graph FASs.

(a) Accuracy.

Support Threshold	Coenen-Angle SVM	Coenen-Angle J48graft	Coenen-Distance SVM	Coenen-Distance J48graft	Coenen-Multi SVM	Coenen-Multi J48graft
0.5	97.67	97.67	96.33	96.33	**98.00**	**98.00**
0.4	98.00	98.00	97.67	97.67	**98.67**	**98.67**
0.3	98.00	98.33	97.67	98.00	**99.00**	**99.00**
Average	97.89	98.00	97.22	97.33	**98.56**	**98.56**

(b) F-measure.

Support Threshold	Coenen-Angle SVM	Coenen-Angle J48graft	Coenen-Distance SVM	Coenen-Distance J48graft	Coenen-Multi SVM	Coenen-Multi J48graft
0.5	97.70	97.70	96.30	96.30	**98.00**	**98.00**
0.4	98.00	98.00	97.70	97.70	**98.70**	**98.70**
0.3	98.00	98.30	97.70	98.00	**99.00**	**99.00**
Average	97.90	98.00	97.23	97.33	**98.57**	**98.57**

(c) Area under ROC.

Support Threshold	Coenen-Angle SVM	Coenen-Angle J48graft	Coenen-Distance SVM	Coenen-Distance J48graft	Coenen-Multi SVM	Coenen-Multi J48graft
0.5	97.70	97.70	96.40	96.40	**98.00**	**98.00**
0.4	98.00	98.00	97.70	97.70	98.60	**99.30**
0.3	98.00	98.30	97.70	98.00	99.00	**99.40**
Average	97.90	98.00	97.27	97.37	98.53	**98.90**

A.1.2.3 Representative FASs vs. All FASs

In our second experiment, by using the same similarity threshold of our first experiment ($\tau = 0.4$), we contrast the results achieved over Coenen-Multi using representative FASs (applying GenCloMgVEAM and CliqueAMgMiner) against the results obtained using all FASs (applying AMgMiner). First, in Table 3, we show the number of patterns identified in Coenen-Multi, and in the tables 4 and 5, we show the results obtained by SVM and J48graft, respectively.

Table 3: Number of FASs identified by AMgMiner ("All FASs"), GenCloMgVEAM ("Generalized closed FASs") and CliqueMgVEAM ("Clique FASs") with $\tau = 0.4$, as well as different closed threshold δ and support threshold values over Coenen-Multi.

Support Threshold	All FASs	Generalized Closed FASs $\delta = 0$	$\delta = 0.1$	$\delta = 0.2$	$\delta = 0.3$	$\delta = 0.4$	$\delta = 1$	Clique FASs
0.5	12	12	11	8	8	8	8	12
0.4	37	25	21	18	17	11	11	35
0.3	349	154	86	71	68	62	62	59

In Table 3, the first column shows the support threshold values used in the mining process. The second column shows the number of FASs mined by AMgMiner (i.e., all FASs), the other six consecutive columns show the number of generalized closed FASs mined by GenCloMgVEAM with different closed threshold values ($\delta = 0$, 0.1, 0.2, 0.3, 0.4 and 1), and the last column shows the number of clique FASs mined by CliqueAMgMiner. It is important to highlight that, over Coenen-Multi, for $\delta > 0.4$ GenCloMgVEAM mines the same patterns

as those mined using $\delta = 1$ (i.e., the maximal FASs). Thus, in Table 3, we show only the number of patterns mined by GenCloMgVEAM with $0 \leq \delta \leq 0.4$ and $\delta = 1$.

As it can be seen in Table 3, when our representative FAS mining algorithms are applied, we obtain up to 16%, 29% and 66% of the total number of patterns for $minsup = 0.3$, 0.4 and 0.5, respectively.

Table 4: Classification results (%) achieved over Coenen-Multi using SVM with all FASs and representative FASs.

(a) Accuracy.

Support Threshold	All FASs	Generalized Closed FASs $\delta = 0$	$\delta = 0.1$	$\delta = 0.2$	$\delta = 0.3$	$\delta = 0.4$	$\delta = 1$	Clique FASs
0.5	**98.00**	**98.00**	**98.00**	96.33	96.33	96.33	96.33	**98.00**
0.4	**98.67**	**98.67**	98.00	98.00	98.00	98.00	98.00	**98.67**
0.3	99.00	**99.33**	99.00	**99.33**	98.67	98.67	98.67	98.67
Average	98.56	**98.67**	98.33	97.89	97.67	97.67	97.67	98.45

(b) F-measure.

Support Threshold	All FASs	Generalized Closed FASs $\delta = 0$	$\delta = 0.1$	$\delta = 0.2$	$\delta = 0.3$	$\delta = 0.4$	$\delta = 1$	Clique FASs
0.5	**98.00**	**98.00**	**98.70**	96.30	96.30	96.30	96.30	**98.00**
0.4	**98.70**	**98.70**	98.00	98.00	98.00	98.00	98.00	**98.70**
0.3	99.00	**99.33**	99.00	**99.33**	98.70	98.70	98.70	98.70
Average	98.57	**98.68**	98.57	97.88	97.67	97.67	97.67	98.47

(c) Area under ROC.

Support Threshold	All FASs	Generalized Closed FASs $\delta = 0$	$\delta = 0.1$	$\delta = 0.2$	$\delta = 0.3$	$\delta = 0.4$	$\delta = 1$	Clique FASs
0.5	**98.00**	**98.00**	**98.00**	96.40	96.40	96.40	96.40	**98.00**
0.4	**98.60**	**98.60**	98.00	98.00	98.00	98.00	98.00	**98.60**
0.3	99.00	**99.33**	99.00	**99.33**	98.60	98.60	98.60	98.70
Average	98.53	**98.64**	98.33	97.91	97.67	97.67	97.67	98.43

The tables 4 and 5 are composed by three sub-tables for showing: (a) accuracy, (b) F-meature, and (c) area under ROC results achieved on Coenen-Multi. Each sub-table contains nine columns, where the first column shows the support threshold values used in the experiment; the second one shows the classification results achieved by using all FASs as attributes for image classification. The next six columns show the results achieved by using the generalized closed FASs mined by GenCloMgVEAM with different closed threshold values, and the last column shows the results obtained by using the clique FASs mined by CliqueAMgMiner.

In the tables 4 and 5, we can see that similar classification results are obtained; however, through the representative FASs we reduce the number of FASs.

A.1.3 Summary and Conclusions

In this appendix, an example of how the multi-graph FASs mined by our proposed algorithms can be used for image classification was shown. From this example, we can see that, as we expected, representing images as multi-graphs and using multi-graph FASs for building

Table 5: Classification results (%) achieved over Coenen-Multi using J48graft with all FASs and representative FASs.

(a) Accuracy.

Support Threshold	All FASs	Generalized Closed FASs $\delta = 0$	$\delta = 0.1$	$\delta = 0.2$	$\delta = 0.3$	$\delta = 0.4$	$\delta = 1$	Clique FASs
0.5	**98.00**	**98.00**	97.33	97.33	97.33	97.33	97.33	**98.00**
0.4	**98.67**	98.00	98.00	98.00	98.00	97.67	97.67	**98.67**
0.3	99.00	99.00	**99.33**	**99.33**	98.00	98.00	98.00	99.00
Average	**98.56**	98.33	98.22	98.22	97.78	97.67	97.67	**98.56**

(b) F-measure.

Support Threshold	All FASs	Generalized Closed FASs $\delta = 0$	$\delta = 0.1$	$\delta = 0.2$	$\delta = 0.3$	$\delta = 0.4$	$\delta = 1$	Clique FASs
0.5	**98.00**	**98.00**	97.33	97.33	97.33	97.33	97.33	**98.00**
0.4	**98.70**	98.00	98.00	98.00	98.00	97.70	97.70	**98.70**
0.3	99.00	99.00	**99.30**	**99.30**	98.00	98.00	98.00	99.00
Average	**98.57**	98.33	98.21	98.21	97.78	97.68	97.68	98.56

(c) Area under ROC.

Support Threshold	All FASs	Generalized Closed FASs $\delta = 0$	$\delta = 0.1$	$\delta = 0.2$	$\delta = 0.3$	$\delta = 0.4$	$\delta = 1$	Clique FASs
0.5	**98.00**	**98.00**	97.33	97.33	97.33	97.33	97.33	**98.00**
0.4	**99.30**	**99.30**	98.70	98.70	98.00	97.80	97.80	**99.30**
0.3	**99.40**	**99.40**	**99.40**	**99.40**	98.70	98.70	98.70	**99.40**
Average	**98.90**	**98.90**	98.48	98.48	98.01	97.91	97.91	**98.90**

a vectorial representation allows improving the accuracy of some image classification tasks; regarding representing images as simple-graphs and using simple-graph FASs.

We also showed that the representative FASs mined by GenCloMgVEAM and CliqueAMgMiner allow reducing the number of patterns maintaining similar classification results. This reduction is important because, in this way, the dimensionality of the vectorial representation is reduced. Therefore, the performance of applying traditional classifiers could be improved.

APPENDIX B

B.1 Graph Transformation Correctness

In this appendix we demonstrate that our proposed allEdges method, based on graph transformations, mines all Frequent Approximate Subgraphs (FASs) from a multi-graph collection by using a simple-graph FAS miner. For introducing this demonstration, we need to define some preliminary concepts. Therefore, we provide the concepts of induced subgraph, the operator sum between two graphs, multi-graph simplification and graph generalization. These concepts allow us simplifying the demonstrations introduced in this appendix.

Definition B.1 (Induced subgraphs) *Let $G = (V, E, \phi, I, J)$ be a graph and $V' \subseteq V$ and $E' \subseteq E$. The* v-induced subgraph *of G regarding V' is defined as the subgraph of G denoted by $G[V'] = (V', E_1, \phi_1, I_1, J_1)$, where $E_1 = \{e \in E | \phi(e) \subseteq V'\}$. In a similar way, the* e-induced subgraph *of G regarding E' is defined as the subgraph of G denoted by $G[E'] = (V_2, E', \phi_2, I_2, J_2)$, where $V_2 = \bigcup_{e \in E'} \phi(e)$.*

Definition B.2 (The operator $\oplus$) *Let $G_1 = (V_1, E_1, \phi_1, I_1, J_1)$ and $G_2 = (V_2, E_2, \phi_2, I_2, J_2)$ be two graphs, where for each $v \in V_1 \bigcap V_2$, $I_1(v) = I_2(v)$, and for each $e \in E_1 \bigcap E_2$, $\phi_1(e) = \phi_2(e)$ and $J_1(e) = J_2(e)$. Thus, the* sum *of G_1 and G_2 is a supergraph of G_1 and G_2 denoted by $G_1 \oplus G_2 = (V_3, E_3, \phi_3, I_3, J_3)$, where $V_3 = V_1 \bigcup V_2$; $E_3 = E_1 \bigcup E_2$; for each $v \in V_1$ $I_3(v) = I_1(v)$ and for each $v \in V_2$, $I_3(v) = I_2(v)$; for each $e \in E_1$, $\phi_3(e) = \phi_1(e)$ and $J_3(e) = J_1(e)$, and for each $e \in E_2$, $\phi_3(e) = \phi_2(e)$ and $J_3(e) = J_2(e)$. We will use the notation $\bigoplus_i G_i$ for denoting the successive sum of several graphs G_i.*

The transformation of a multi-graph into a simple-graph used for allEdges can be formally defined as in Definition B.3.

Definition B.3 (Multi-graph simplification) *Let $G = (V, E, \phi, I, J)$ be a connected multi-graph, and let k and p be two different vertex labels that will be used to represent loops and multi-edges, respectively. The* multi-graph simplification *of G is a graph defined as:*

$$G' = \bigoplus_{e \in E} G'_e, \tag{1}$$

where the graph G'_e is defined starting from $G[\{e\}]$ as follows:

- *If e is a simple-edge, $G'_e = G[\{e\}]$.*

- *If e is a multi-edge and $\phi(e) = \{u, v\}$, the graph G'_e is defined as $G'_e = (V_1, E_1, \phi_1, I_1, J_1)$, where $V_1 = \{u, v, w\}$, $E_1 = \{e_1, e_2\}$, $E_1 \cap E = \emptyset$, $\phi_1(e_1) = \{u, w\}$, $\phi_1(e_2) = \{w, v\}$, I_1 is a restriction of I to V_1 with $I_1(w) = p$, $J_1(e_1) = J_1(e_2) = J(e)$, and $w \notin V$ is a new vertex.*

- *If e is a loop and $\phi(e) = \{v\}$, the graph G'_e is defined as $G'_e = (V_2, E_2, \phi_2, I_2, J_2)$, where $V_2 = \{v, w\}$, $E_2 = \{e'\}$, $e' \notin E$, $\phi_2(e') = \{v, w\}$, I_2 is a restriction of I to V_2 with $I_2(w) = k$, $J_2(e') = J(e)$, and $w \notin V$ is a new vertex.*

Notice that for building simplifications only vertices with label k and p are added, as well as the simple-edges connecting such vertices.

Finally, we need to transform back the simple-graph FASs to a multi-graph context. This transformation can be formally defined as in Definition B.4.

Definition B.4 (Graph generalization) *Let $G' = (V', E', \phi', I', J')$ be a returnable graph and let k and p be the special labels used in Definitions B.3 and 4.1. Let V'_p be the set of all of $v \in V'$ such that $I'(v) = p$, and let V'_k be the set of all of $v \in V'$ such that $I'(v) = k$. Thus, the* generalization *of G' is a graph defined as:*

$$G = G'[V' \setminus (V'_p \cup V'_k)] \oplus \bigoplus_{w \in V'_p \cup V'_k} G_w, \tag{2}$$

where $G'[V' \setminus (V'_p \cup V'_k)]$ is a v-induced subgraph of G' (see Definition B.1), and the graph G_w is defined as follows:

- *If $I'(w) = p$, by the first condition of returnable graph (see Definition 4.1) there are exactly two incidents edges e_1 and e_2, such that $\phi'(e_1) = \{u, w\}$ and $\phi'(e_2) = \{w, v\}$, then G_w is defined as $G_w = (V_1, E_1, \phi_1, I_1, J_1)$, where $V_1 = \{u, v\}$, $E_1 = \{e\}$, $e \notin E'$, $\phi_1(e) = \{u, v\}$, I_1 is a restriction of I' to V_1, and $J_1(e) = J'(e_1) = J'(e_2)$.*

- *If $I'(w) = k$, by the second condition of returnable graph (see Definition 4.1) there is exactly one incident edge e_2, such that $\phi'(e_2) = \{v, w\}$, then G_w is defined as $G_w = (V_2, E_2, \phi_2, I_2, J_2)$, where $V_2 = \{v\}$, $E_2 = \{e\}$, $e \notin E'$, $\phi_2(e) = \{v\}$, I_2 is a restriction of I' to V_2, and $J_2(e) = J'(e_2)$.*

Notice that, for building generalizations, only vertices with label k or p are removed, together with the simple-edges connecting such vertices.

Once introduced the previous defined preliminary concepts, we demonstrate the correctness of allEdges.

First, we will demonstrate that the multi-graph patterns identified by our proposal are FASs of the input multi-graph collection D. Next, supposing that the set F' contains all simple-graph FASs mined from the simple-graph collection D', we will demonstrate that the set F with all multi-graph FASs of the input multi-graph collection can be obtained from F'

by applying Definition B.4. The set of non-returnable patterns are removed from the solution, maintaining only those patterns which are returnable FASs.

For verifying the problem discussed in this appendix, we first demonstrate, through Theorem 4, that given two multi-graphs G_1 and G_2, if $G_2 \subseteq_s G_1$ then its simplifications (generalizations) fulfill that $G'_2 \subseteq_s G'_1$. To this end, we first introduce some theorems as support for Theorem 4. First, sufficient conditions for the isomorphism between graphs is provided by Theorem 1.

Theorem 1. *Let $G_1 = (V_1, E_1, \phi_1, I_1, J_1)$ and $G_2 = (V_2, E_2, \phi_2, I_2, J_2)$ be two graphs, such that there is a bijective function $f : V_1 \to V_2$ fulfilling the first condition of Definition 2.3. Moreover, let us to suppose that the set E_i can be partitioned into two subsets, $E_i = E_{i,1} \cup E_{i,2}$, for each $i \in \{1, 2\}$, such that there is an isomorphism (f_j, g_j) between $G_1\{E_{1,j}\}$ and $G_2\{E_{2,j}\}$, being f_j a restriction of f, for each $j \in \{1, 2\}$. Then, there is a function g such that (f, g) is an isomorphism between G_1 and G_2.*

Proof. Let us define $g : E_1 \to E_2$ as the following function $g(e) = g_j(e)$, for each $e \in E_{1,j}$; $j \in \{1, 2\}$. Let e_2 and $e_1 \in E_1$ be two edges, where $\phi_1(e_1) = \{u, v\}$ and $e_2 = g(e_1)$. It is easy to see that the second condition of Definition 2.3 is fulfilled. In fact, for each $j \in \{1, 2\}$, it is verified that if $e_1 \in E_{1,j}$ then: $e_2 = g(e_1) = g_j(e_1) \in E_{2,j} \subseteq E_2$; and $\phi_2(e_2) = \{f_j(u), f_j(v)\} = \{f(u), f(v)\}$, since f_j is a restriction of f; and, $J_1(e_1) = J_{1,j}(e_1) = J_{2,j}(e_2) = J_2(e_2)$, being $J_{1,j}(e_1)$ and $J_{2,j}(e_2)$ the edge labeling function in $G_1\{E_{1,j}\}$ and $G_2\{E_{2,j}\}$, respectively. Thus, the second condition of Definition 2.3 is ensured. □

With Theorem 2 we establish that the simplifications or generalizations of two isomorphic graphs are isomorphic too.

Theorem 2. *Let $G_1 = (V_1, E_1, \phi_1, I_1, J_1)$ and $G_2 = (V_2, E_2, \phi_2, I_2, J_2)$ be two isomorphic connected (returnable) graphs. Then, the graphs, $G'_1 = (V'_1, E'_1, \phi'_1, I'_1, J'_1)$ and $G'_2 = (V'_2, E'_2, \phi'_2, I'_2, J'_2)$, obtained from the simplifications (generalizations) of G_1 and G_2 are also isomorphic.*

Proof. Let (f_1, g_1) be the isomorphism between G_1 and G_2. This theorem has two parts: (1) G'_1 and G'_2 are simplifications, according to Definition B.3, of G_1 and G_2, respectively, and (2) G'_1 and G'_2 are generalizations, according to Definition B.4, of G_1 and G_2, respectively.

(1) Let $e_1 \in E_1$ be an edge, then, applying Definitions B.3 and B.4, we have three cases:

1. If e_1 is a simple-edge, then there are three edges $e'_1 \in E'_1$, $e_2 \in E_2$, and $e'_2 \in E'_2$, such that $\phi_1(e_1) = \phi'_1(e'_1) = \{u, v\}$, $\phi_2(e_2) = \phi'_2(e'_2) = \{f_1(u), f_1(v)\}$, and $e_2 = g_1(e_1)$.

2. If e_1 is a loop, then there is only one vertex $u \in \phi_1(e_1)$ and there are three edges $e'_1 \in E'_1$, $e'_2 \in E'_2$, and $e_2 \in E_2$, such that $\phi_2(e_2) = \{f_1(u)\}$, $e_2 = g_1(e_1)$, $\phi'_1(e'_1) = \{u, w\}$, $\phi'_2(e'_2) = \{f_1(u), f_1(w)\}$, and $I'_1(w) = I'_2(w) = k$.

3. If e_1 is a multi-edge, then $\exists e \in E_1, \phi_1(e) = \phi_1(e_1)$. There are five edges $e_2 \in E_2$, $e'_1, e'_2 \in E'_1$, $e'_3, e'_4 \in E'_2$, and there is a vertex $w \in V'_1 \cap V'_2$, such that $e_2 = g_1(e_1)$, $J_1(e_1) = J_2(e_2) =$

$J'_1(e'_1) = J'_1(e'_2) = J'_2(e'_3) = J'_2(e'_4)$, $\phi_1(e_1) = \{u, v\}$, $u \neq v$, $\phi_2(e_2) = \{f_1(u), f_1(v)\}$, $\phi'_1(e'_1) = \{u, w\}$, $\phi'_1(e'_2) = \{w, v\}$, $v \neq w \neq u$, $I'_1(w) = p$, $\phi'_2(e'_3) = \{f_1(u), f_1(w)\}$, $\phi'_2(e'_4) = \{f_1(w), f_1(v)\}$, and $I'_1(w) = I'_2(w)$.

For each $e_1 \in E_1$ it is possible to unambiguously pick up $e_2 \in E_2$, such that $G_1[\{e_1\}]$ and $G_2[\{e_2\}]$ are isomorphic. Thus, a bijective function $g_1 : E_1 \mapsto E_2^*$ can be defined, where $g_1(e_1) = e_2$, for each $e_1 \in E_1$, and $E_2^* = \{e_2 \in E_2 | \exists e_1 \in E_1 : g_1(e_1) = e_2\}$.

Taking into account that the substructures of G'_1 and G'_2 represent the substructures of G_1 and G_2 (according to Definitions B.3 and B.4), respectively, and there is an isomorphism between the substructures of G_1 and G_2, then, applying the theorem 1, there is an isomorphism between the substructures of G'_1 and G'_2, and therefore there is an isomorphism between G'_1 and G'_2.

(2) This can be demonstrated in a similar way, using Definitions B.3 and B.4 in a reverse way. □

Theorem 3 introduces a reversing property, which means that if we have a simplification (generalization) of a graph G_1, denoted by G, and we perform the inverse process, i.e. a generalization (simplification), over G, then the result is isomorphic to G_1.

Theorem 3. *Let $G = (V, E, \phi, I, J)$ be the simplification (generalization) of the connected graph $G_1 = (V_1, E_1, \phi_1, I_1, J_1)$, and let k and p be the special labels; then G_1 is isomorphic to the generalization (simplification) of G.*

Proof. This theorem has two parts: (1) G is a simplification of G_1, and (2) G is a generalization of G_1.

(1) Let $G_2 = (V_2, E_2, \phi_2, I_2, J_2)$ be the generalization of G. Applying Definitions B.3 and B.4, and taking into account that for building G only vertices with label k or p were added, without changing any other label, it is clear that $V_1 = V_2$ and $I_1 \equiv I_2$.

Now, let $f : V_1 \mapsto V_2$, such that $f(v) = v$, be the identity function, which is consequently bijective. In this sense, for each $u \in V_1$, it is fulfilled that $f(u) = u \in V_1 = V_2$ and $I_1(u) = I_1(f(u)) = I_2(f(u))$, since $I_1 \equiv I_2$.

Next, applying the aforementioned definitions, it is easy to see that for each $e_1 \in E_1$ there is $e_2 \in E_2$, such that $\phi_1(e_1) = \phi_2(e_2)$, $J_1(e_1) = J_2(e_2)$, $I_1(v) = I_2(v)$, for $v \in \phi_1(e_1)$. This fact can be stated by considering the three cases for e_1:

1. If e_1 is a simple-edge, with $\phi_1(e_1) = \{u, v\}$, then there are two edges $e \in E$ and $e_2 \in E_2$, such that $\phi_1(e_1) = \phi(e) = \phi_2(e_2)$, $J_1(e_1) = J(e) = J_2(e_2)$, $I_1(u) = I(u) = I_2(u)$, and $I_1(v) = I(v) = I_2(v)$. In this case, the transformation process does not produce any change according to Definitions B.3 and B.4.
2. If e_1 is a multi-edge, with $\phi_1(e_1) = \{u, v\}$, then, according to Definition B.3, there are two simple-edges $e \in E$ and $e' \in E$, such that $\phi(e) = \{u, w\}$, $\phi(e') = \{w, v\}$, being $w \in V$

and $I(w) = p$, $J_1(e_1) = J(e) = J(e')$, $I_1(u) = I(u)$, and $I_1(v) = I(v)$. Thus, according to Definition B.4 there is also an edge $e_2 \in E_2$, such that $\phi_2(e_2) = \{u, v\}$, $J_2(e_2) = J(e) = J(e')$, $I_2(u) = I(u)$, and $I_2(v) = I(v)$.

3. If e_1 is a loop, with $\phi_1(e_1) = \{v\}$, then, according to Definition B.3, there is a simple-edge $e \in E$, such that $\phi(e) = \{v, w\}$, being $w \in V$ and $I(w) = k$, $J_1(e_1) = J(e)$, and $I_1(v) = I(v)$. Thus, according to Definition B.4, there is also an edge $e_2 \in V_2$, such that $\phi_2(e_2) = \{v\}$, $J_2(e_2) = J(e)$, and $I_2(v) = I(v)$.

In summary, for each $e_1 \in E_1$ ($e_2 \in E_2$) there is $e_2 \in E_2$ ($e_1 \in E_1$), such that $\phi_1(e_1) = \phi_2(e_2)$, $J_1(e_1) = J_2(e_2)$, $I_1(v) = I_2(v)$, for each $v \in \phi_1(e_1)$ ($v \in \phi_2(e_2)$). Thus, we have an isomorphism (f_{e_1}, g_{e_1}) between $G_1[\{e_1\}]$ and $G_2[\{e_2\}]$, where f_{e_1} is a restriction of f and $g_{e_1}(e_1) = e_2$.

Taking into account that $G_i = \bigoplus_{e \in E_i} G_i[\{e\}]$, being G_i connected, for each $i \in \{1, 2\}$, and repeatedly applying Theorem 1, it is concluded that G_1 and G_2 are isomorphic. Thus, the part (1) of this theorem was proved.

(2) This can be also demonstrated in a similar way, using Definitions B.3 and B.4 in a reverse way. □

Finally, using Theorems 1, 2 and 3, we can introduce and prove Theorem 4 as follows:

Theorem 4. *Let $G'_1 = (V'_1, E'_1, \phi'_1, I'_1, J'_1)$ and $G'_2 = (V'_2, E'_2, \phi'_2, I'_2, J'_2)$ be two multi-graphs, and let $G_1 = (V_1, E_1, \phi_1, I_1, J_1)$ and $G_2 = (V_2, E_2, \phi_2, I_2, J_2)$ be two simple-graphs, such that G_1 and G_2 are simplifications of G'_1 and G'_2, respectively; then $G'_2 \subseteq_s G'_1 \Leftrightarrow G_2 \subseteq_s G_1$.*

Proof. For proving the direct sense ($\Rightarrow$) of the theorem, it is assumed that G'_2 is sub-isomorphic to G'_1. Let (f'_2, g'_2) be the isomorphism between G'_2 and a subgraph of G'_1. Let $e_2 \in E_2$ be an edge, then, according to Definitions B.3 and B.4, we have three cases:

1. If $\forall v \in \phi_2(e_2), I_2(v) \notin \{k, p\}$, then e_2 is a simple-edge and there are three edges $e'_2 \in E'_2$, $e'_1 \in E'_1$, and $e_1 \in E_1$, such that $\phi_2(e_2) = \phi'_2(e'_2) = \{u, v\}$, $\phi'_1(e'_1) = \phi_1(e_1) = \{f'_2(u), f'_2(v)\}$, $e'_1 = g'_2(e'_2)$.

2. If $\exists w \in \phi_2(e_2), I_2(w) = p$, then there is one vertex $u \in \phi_2(e_2)$, $u \neq w$, and $\phi_2(e_2) = \{u, w\}$, and there is one edge $e \in E_2$, $e \neq e_2$ and $\phi_2(e) = \{w, v\}$, $v \neq u$. There are four edges $e'_2 \in E'_2$, $e'_1 \in E'_1$, and $e_1, e_3 \in E_1$, such that $\phi'_2(e'_2) = \{u, v\}$, $\phi'_1(e'_1) = \{f'_2(u), f'_2(v)\}$, $\phi_1(e_1) = \{f'_2(u), w\}$, $\phi_1(e_3) = \{w, f'_2(v)\}$, $I_1(w) = p$, and $e'_1 = g'_2(e'_2)$.

3. If $\exists w \in \phi_2(e_2), I_2(w) = k$, then there is one vertex $u \in \phi_2(e_2)$, $u \neq w$, and $\phi_2(e_2) = \{u, w\}$. There are three edges $e'_2 \in E'_2$, $e'_1 \in E'_1$, and $e_1 \in E_1$, such that $\phi'_2(e'_2) = \{u\}$, $\phi'_1(e'_1) = \{f'_2(u)\}$, $\phi_1(e_1) = \{f'_2(u), w\}$, $I_1(w) = k$, and $e'_1 = g'_2(e'_2)$.

In summary, for each $e_2 \in E_2$ it is possible unambiguously to finding $e_1 \in E_1$, such that $G_1[\{e_1\}]$ and $G_2[\{e_2\}]$ are isomorphic. Thus, a bijective function $g_2 : E_2 \mapsto E_1^*$ can be defined, where $g_2(e_2) = e_1$, for each $e_2 \in E_2$, and $E_1^* = \{e_1 \in E_1 | \exists e_2 \in E_2 : g_2(e_2) = e_1\}$.

Using Theorem 1, it is clear that $G_2 = \bigoplus_{e \in E_2} G_2[\{e\}]$ is isomorphic to $G_3 = \bigoplus_{e \in E_1^*} G_1[\{e\}]$, with $G_3 \subseteq G_1$. Therefore, the direct sense of the theorem has been proved.

The inverse sense ($\Leftarrow$) of the theorem can also be easily proved, it is enough to follow the definition of simplification in a reverse way as it was done for the direct sense. □

Theorem 4 demonstrates the sub-isomorphic correspondence between two graphs and its generalizations (simplifications). Next, a correspondence between the support of a graph and its generalization (simplification) is formalized in Corollary 1.

Corollary 1. *Let $D = \{G_1, \ldots, G_N\}$ be a collection of N simple-graphs, let $D' = \{G'_1, \ldots, G'_N\}$ be a collection of N multi-graphs, where G_i is the simplification of G'_i, for each $1 \leq i \leq N$. Let G be the simplification of a multi-graph G'. Then, $supp(G, D) = supp(G', D')$.*

Corollary 1 is directly derived from Theorem 4, since G is sub-isomorphic to $G_i \Leftrightarrow G'$ is sub-isomorphic to G'_i, for each $1 \leq i \leq N$.

Corollary 2. *Let D and D' be the collections of simple-graphs and multi-graphs, respectively used in Corollary 1. Let G be the simplification of a multi-graph G'. Then, G is a FAS in $D \Leftrightarrow G'$ is a returnable FAS in D'.*

Corollary 2 is derived directly from Corollary 1, since $supp(G, D) = supp(G', D')$.

Theorem 5. *Let $D' = \{G_1, G_2, \ldots, G_{|D'|}\}$ be a multi-graph collection, if allEdges is applied over D' using an algorithm for mining all simple-graph FASs; then, all multi-graph FASs are mined from D'.*

Proof. The proof is immediate from Theorem 4 and Corollaries 1 and 2. □