Reducing the Number of Canonical Form Tests for Frequent Subgraph Mining

Andrés Gago Alonso[1], Jesús A. Carrasco Ochoa[2], José E. Medina Pagola[1], and José F. Martínez Trinidad[2]

[1] Data Mining Department, Advanced Technologies Application Center, La Habana, Cuba
{agago, jmedina}@cenatav.co.cu
[2] Computer Science Department, National Institute of Astrophysics, Optics and Electronics, Santa María de Tonantzintla, Puebla, México
{ariel, fmartine}@inaoep.mx

Abstract. Frequent connected subgraph (FCS) mining is an interesting problem with wide applications in real life. Most of the FCS mining algorithms have been focused on detecting duplicate candidates using canonical form tests. Canonical form tests have high computational complexity, and therefore, they affect the efficiency of graph miners. In this paper, we introduce novel properties to reduce the number of canonical form tests in FCS mining. Based on these properties, a new algorithm for FCS mining called gRed is presented. The experimentation on real world datasets shows the impact of the proposed properties on the efficiency of gRed reducing the number of canonical form tests regarding gSpan. Besides, the performance of our algorithm is compared against gSpan and other state-of-the-art algorithms.

Keywords. Data mining, frequent patterns, graph mining, frequent subgraph.

Reduciendo el número de pruebas de forma canónica para la minería de subgrafos frecuentes

Resumen. La minería de subgrafos conexos frecuentes es un problema interesante con amplias aplicaciones en la vida práctica. La mayor parte de los algoritmos para este tipo de minería detectan los candidatos duplicados utilizando pruebas de forma canónica. Este tipo de pruebas tienen una alta complejidad computacional, lo cual afecta el desempeño de los algoritmos de minería de grafos. En este artículo se proponen nuevas propiedades para reducir el número de pruebas de forma canónica en este tipo de minería. Basado en estas propiedades, se propone un nuevo algoritmo llamado gRed. Los resultados experimentales en colecciones de datos reales muestran el impacto de las nuevas propiedades en la eficiencia de gRed, reduciendo el número de pruebas de forma canónicas con respecto a gSpan. Además, el desempeño de gRed es comparado respecto gSpan y otros algoritmos reportados en el estado del arte.

Palabras clave. Minería de datos, patrones frecuentes, minería de grafos, subgrafos frecuentes.

1 Introduction

Graph mining is becoming increasingly important since advances in collecting and storing data have produced an explosive growth in the amount of available structured data. This situation has boosted the necessity of new tools to transform this big amount of complex data into useful information or knowledge for decision makers. The development of such tools requires techniques that usually need long time and have high memory requirements. Frequent connected subgraph (FCS) mining is an example of these techniques.

FCS mining is the process of finding connected subgraphs that frequently occur in a collection of graphs. Recently, this topic has been an interesting theme in data mining with wide applications, including mining substructures from chemical compound databases [3], XML documents [9], biological networks [14], and so forth [7]. Labeled graphs can be used to model relations among data in the aforementioned applications because labels can represent attributes of entities and relations among themselves [10]. As a consequence, several algorithms have been proposed for FCS mining in collections of labeled graphs.

The first algorithm for finding all frequent subgraphs (connected or unconnected) in a collection of labeled graphs was AGM [12]. This algorithm was followed by FSG [15] and AcGM [12] algorithms, for mining all frequent connected subgraphs (FCSs). These algorithms are similar to the original Apriori algorithm [1] for mining frequent itemsets.

To avoid overheads of the earlier algorithms, new pattern growth based algorithms such as gSpan [24, 25], MoFa [3], FFSM [11], and Gaston [17] were developed. These algorithms were compared in a common framework [22]. In this experimentation, the four algorithms were competitive among themselves, although Gaston and MoFa were the fastest and slowest algorithms respectively, in almost all tests. On the other hand, gSpan was the best algorithm regarding its memory requirements since the embedding structures, used by MoFa, FFSM, and Gaston for frequency calculation and candidate enumeration, could be a problem if not enough memory is available or if the memory throughput is not high enough.

The emergence of duplicate candidates during the enumeration process is one of the major problems in all recent approaches. Duplicate candidates are treated by representing the subgraphs with a unique code called canonical form. The DFS code (Depth First Search code) is an example of promising kind of canonical form for FCS mining [17]. Candidate enumeration strategies are commonly defined by means of these codes, trying to avoid non-canonical forms by performing canonical form tests which has very high computational complexity [2].

In this paper, we introduce non-minimality conditions, a reuse condition, and a cut property for DFS codes, which are useful for reducing the number of canonical form tests in FCS mining. The non-minimality conditions allow knowing the results of some canonical form tests in constant time. These conditions do not remove all the duplicate candidates; therefore, canonical form tests are required for non-filtered candidates. However, the reuse condition helps to reduce the number of such expensive tests by reusing previous test results for predicting new results without performing a test. The cut property provides an efficient way for taking advantage of the reuse condition by defining boundaries between canonical and non-canonical forms in the candidate space. Additionally, this paper introduces a new algorithm called gRed (graph Candidate Reduction Miner) based on these properties. Our algorithm uses the non-minimality conditions to reduce the number of candidate graphs and applies the cut property for finding the boundaries between useful and duplicate candidates, in an efficient way.

Preliminary results of this research were introduced in a previous conference paper [6]. In this conference paper, a version of gRed, which is referenced in our research as gRed-v1, is presented. The version gRed-v1 does not exploit the reuse conditions for locating the above mentioned boundaries.

The basic outline of this paper is as follows. Section 2 provides some basic concepts. Section 3 contains the related work. The novel DFS code properties are introduced, discussed, and proved in Section 4, including the details of the gRed algorithm. In Section 5, the experimental results are presented. Finally, conclusions of the research and some ideas about future directions are exposed in Section 6.

2 Basic Concepts

In this section, the background and notation used in the next sections are provided. This work is focused on simple undirected labeled graphs. Henceforth, when we refer to graphs we assume this kind of graph. The formal definition of this type of graph is as follows.

A simple undirected labeled graph is a 4-tuple, $G = \langle V, E, L, l \rangle$, where V is a set whose elements are called vertices, $E \subset \{\{u, v\} | u, v \in V\}$ is a set whose elements are called edges, L is the set of labels and $l : V \cup E \rightarrow L$ is a labeling function for assigning labels to vertices and edges.

Let $G_1 = \langle V_1, E_1, L, l \rangle$ and $G_2 = \langle V_2, E_2, L, l \rangle$ be two graphs having the same set of labels L and the same function l. We say that G_1 is a subgraph of G_2 if $V_1 \subseteq V_2$ and $E_1 \subseteq E_2$. In this case, we use the notation $G_1 \subseteq G_2$.

In graph mining over collections of labeled graphs, the frequency of the candidates is calculated using subgraph isomorphism tests. We

say that f is an isomorphism between $G_1 = \langle V_1, E_1, L_1, l_1 \rangle$ and $G_2 = \langle V_2, E_2, L_2, l_2 \rangle$ if $f : V_1 \rightarrow V_2$ is a bijective function such that f preserves vertex labels (that is $\forall v \in V_1, l_1(v) = l_2(f(v))$), f preserves edges and edges labels (that is $\forall \{u, v\} \in E_1,\ \{f(u), f(v)\} \in E_2$ and $l_1(\{u, v\}) = l_2(\{f(u), f(v)\})$.

A subgraph isomorphism from G_1 to G_2 is an isomorphism from G_1 to a subgraph of G_2. In this case we will say that G_2 holds G_1.

We say that $P = (v_1, v_2, \ldots, v_k)$ is a path in a graph $G = \langle V, E, L, l \rangle$, if $v_i \in V$ for all i, $1 \leq i \leq k$ and for each pair of consecutive vertices v_i and v_{i+1}, $\{v_i, v_{i+1}\} \in E$. In this case we say that v_1 and v_k are connected by P. If $v_1 = v_k$ and $k > 1$, we say that P is a cycle. A graph G is connected if each pair of vertices in V is connected by a path.

Trees are a special kind of graph. A tree is a connected graph without cycles. A tree is called a rooted tree if one vertex has been selected as root; in this case, the edges have a natural orientation starting from the root. In graph theory, a tree is a graph in which any two vertices are connected by exactly one path [5].

Let $T = \langle V_T, E_T, L_T, l_T \rangle$ be a rooted tree with root v_0 and let $v, u \in V_T$ be two vertices in T. We say that v is the parent of u if the unique path from v_0 to u passes through v and $\{v, u\} \in E_T$; in this case, we also say that u is a child of v.

Let $D = \{G_1, G_2, \ldots, G_{|D|}\}$ be a collection of labeled graphs and let δ be a predefined support threshold. The support of a graph g in D is defined as the number of graphs $G_i \in D$ such that G_i holds g. We use the notation $\sigma(g, D)$ to refer to the support of g in the collection D. A graph g occurs frequently in the collection D if $\sigma(g, D) \geq \delta$. Frequent connected subgraph (FCS) mining is the process of finding connected subgraphs that occur frequently in a collection of graphs.

3 Related Work

Algorithms for FCS mining have been classified, according to the candidate enumeration strategy, in two classes: the Apriori based algorithms and the pattern growth based algorithms [7]. Previous comparative studies have shown that the second class of algorithms has better performance than the first ones [17, 22].

One of the major problems in all pattern growth based algorithms is the emergence of duplicate candidates during the enumeration process. A duplicate candidate is one that has already been considered in a previous step and appears again during the search. In pattern growth based algorithms, duplicate candidates are treated using unique sequential graph representations called canonical forms or unique codes. The candidate enumeration strategies are always defined by means of these codes trying to avoid non-canonical forms. A non-canonical form represents a duplicate candidate since its corresponding canonical form should be already considered in a previous step.

None of the existing candidate enumeration strategies remove all non-canonical forms. Therefore, a canonical form test in each candidate is required. Unfortunately, a canonical form test is equivalent to the isomorphism problem (that is a NP-Complete problem); therefore, its computational complexity is very high [2].

3.1 Brief of Pattern Growth Based Algorithms

The most commonly cited pattern growth based algorithms for FCS mining are gSpan [24],[25], MoFa [3], FFSM [11], and Gaston [16]. In addition, there are other works on this paradigm but all of them are based on at least one of the aforementioned four algorithms.

The gSpan algorithm was the first pattern growth based algorithm for FCS mining [24]. It introduced a promising canonical form for labeled graph representation called DFS code. A DFS code is built during a depth first search traversal of a graph.

The gSpan scheme has been used as a starting point for designing more efficient and better adapted algorithms. For example, a data structure called ADI was used for processing big collections of graphs [20]. An algorithm called Edgar improved the gSpan implementation introducing code optimization techniques and a data structure for storing the candidate embeddings [21]. In the FSP algorithm, novel

properties of the search space are used to improve the graph and subgraph isomorphism tests [8].

MoFa uses canonical forms based on breadth first search graph traversals [2, 3]. This algorithm provides several functionalities for applications in molecular data collections; nonetheless, it has shown a poor behavior in previous comparative studies [22].

FFSM is a hybrid algorithm combining ideas of Apriori and pattern growth based algorithms [11]. A special kind of canonical forms based on adjacency matrices is used for representing graph candidates. Recently, a new algorithm based on FFSM, called FSMA, was presented [23]. FSMA uses incidence matrices instead adjacency matrices; thus, it reaches some improvements in the canonical form tests.

Gaston is one of the most efficient algorithms for FCS mining [16]. Several previous results, obtained for sequence and tree mining, were exploited to improve the search of paths and trees in Gaston. Next, frequent paths and trees are used to generate graphs with cycles. Nevertheless, generating graph with cycles does not reach the efficiency of generating paths and trees.

In conclusion, pattern growth based algorithms have shown great advances in the last years. However, almost all algorithms require exhaustive canonical form tests to detect duplicates in each candidate; only Gaston improves duplicate detection for path and tree mining. Therefore, duplicate detection is still a challenging problem. In this paper, we propose novel properties of the DFS code to reduce the number of canonical form tests in FCS mining.

3.2 DFS Codes

A labeled graph can be represented by a unique sequence of edges called minimum DFS code. This kind of canonical representation is based on DFS graph traversals and it was introduced in gSpan [24]. This section includes some concepts about DFS codes that are required for understanding our work.

Let $G = \langle V, E, L, l \rangle$ be a connected graph and suppose that a DFS traversal in G is performed. A DFS tree $T = \langle V_T, E_T, L_T, l_T \rangle$ of G is the rooted tree built as follow: the starting vertex in the traversal is the root of T, T is a spanning tree of G ($V_T = V$) and T contains the edges of G that were used for the DFS traversal ($E_T \subseteq E$).

A connected graph $G = \langle V, E, L, l \rangle$ can have many different DFS trees because there is more than one DFS traversal. Each DFS tree T defines a unique order among all the vertices in V. Therefore, each vertex could be numbered according to this DFS order.

Assuming $n = |V|$, the root of T is numbered with index 0 and the last vertex in the DFS traversal is numbered with index $n - 1$. The last vertex is also called rightmost vertex of T. The rightmost path of T is defined as the straight path from the root to the rightmost vertex.

Each edge $e = \{u, v\} \in E$ is coded as a tuple according to the DFS tree T. Suppose that the vertices u and v have indices i and j, respectively, according to the DFS order. Let $l_i = l(u)$, $l_j = l(v)$ and $l_{(i,j)} = l_{(j,i)} = l(e)$ be the labels of u, v and e, respectively. Without loss of generality, assume that $i < j$. The tuple of e regarding T is calculated as in (1).

$$\tau(e, T) = \begin{cases} (i, j, l_i, l_{(i,j)}, l_j), e \in E_T; \\ (j, i, l_j, l_{(j,i)}, l_i), e \notin E_T. \end{cases} \quad (1)$$

Thus, each edge $e \in E$ can be coded as a tuple, $\tau(e, T) \in K_n^2 \times L^3$, where $K_n = \{0, 1, \ldots, n - 1\}$ and L is the set of labels of G.A linear order $\prec_e$ among the tuples of the set $K_n^2 \times L^3$ could be defined as follows. If $e_1 = (i_1, j_1, a_1, b_1, c_1)$ and $e_2 = (i_2, j_2, a_2, b_2, c_2)$, $e_1 \prec_e e_2$ if and only if one of the following statements is true:

- $i_1 < j_1 \wedge i_2 < j_2 \wedge (j_1 < j_2 \vee (j_1 = j_2 \wedge i_1 > i_2))$,
- $i_1 \geq j_1 \wedge i_2 \geq j_2 \wedge (i_1 < i_2 \vee (i_1 = i_2 \wedge j_1 < j_2))$,
- $i_1 \geq j_1 \wedge i_2 < j_2 \wedge i_1 < j_2$,
- $i_1 < j_1 \wedge i_2 \geq j_2 \wedge j_1 \leq i_2$,
- $i_1 = i_2 \wedge j_1 = j_2 \wedge e_1 \prec_l e_2$.

The lexicographic order $\prec_l$ is used to compare the tuples e_1 and e_2 regarding the last three components in each tuple. This order is determined comparing the third component as first priority, next the fourth component, and finally the fifth one.

The DFS code of the graph $G = \langle V, E, L, l \rangle$ regarding the DFS tree T is a tuple sequence constructed using $\prec_e$. All the tuples obtained from the edges in E are sorted using $\prec_e$ to build this sequence. Thus, a graph G can be coded as a sequence of tuples, denoted as code(G, T), using one of its DFS trees.

A new order $\prec_s$ among tuple sequences can be built using $\prec_e$. Let $s_1 = (a_1, a_2, \ldots, a_m)$ and $s_2 = (b_1, b_2, \ldots, b_n)$ be two DFS codes (or two tuple sequences), where $a_i, b_j \in K_n^2 \times L^3$ for $1 \leq i \leq m$ and $1 \leq j \leq n$; $s_1 \prec_s s_2$ if one of the following conditions, (2) or (3), is true.

$$\exists t, \forall k < t, a_k = b_k, \text{ and } a_t \prec_e b_t; \quad (2)$$

$$m < n \text{ and } \forall k \leq m, a_k = b_k. \quad (3)$$

The order $\prec_s$ is called DFS lexicographic order and it is used to define a unique DFS code for representing each graph. The minimum DFS code of a graph G is defined as the minimum tuple sequence according to $\prec_s$ among all DFS codes of G.

3.3 Summary of gSpan

Let $s = (a_1, a_2, \ldots, a_m)$ be a minimum DFS code and let $s_0 = (a_1, a_2, \ldots, a_m, b)$ be a DFS code, where $a_i \in K_n^2 \times L^3$ for $1 \leq i \leq m$ and $b \in K_n^2 \times L^3$. The code s_0 is a child of s if the tuple b connects the rightmost vertex of s with another vertex in the rightmost path of s (backward extension), or it introduces a new vertex connected from a vertex of the rightmost path of s (forward extension). In this case, s is called the parent of s_0 and the tuple b is called a rightmost path extension of s and it is denoted as $s_0 = s \diamond b$.

The search space in gSpan is then defined as a rooted tree consisting of nodes representing DFS codes and the relation between parent and child node complies with the aforementioned parent/child relationship. The root of the search space is the degenerated DFS code having zero tuples. The DFS traversal over the search space follows the DFS lexicographic order among all DFS codes.

In general, gSpan works as follows. First, all frequent and minimum DFS codes with only one edge are identified. Next, the search space is traversed in a depth first search order, and for each frequent and minimum DFS code found during the traversal, all its occurrences in the graph collection are located for computing the support of its candidate extensions. Whenever an extension turns out to be non-frequent or non-minimum, it does not need to be considered for further extension, therefore it can be pruned.

In gSpan, the duplicate candidates are the non-minimum DFS codes. A canonical form test, for a DFS code s, verifies if s is the minimum DFS code of the corresponding graph. Instead of calculating the minimum DFS code of s from all the possible DFS codes, picking up the smallest one and comparing it against s, gSpan follows a heuristic search designed using the DFS lexicographic order. Whenever a prefix of a DFS code is generated and it is less than s, then s is non-minimum and the test concludes.

Some canonical form tests are avoided in gSpan using a pre-pruning stage: each backward extension with destination vertex v_j, of a minimum DFS code s, should be no smaller than any forward edge from v_j in s [25]. This pruning is performed during the candidate enumeration (before the duplicate detection process).

In order to compute the support for each candidate code, gSpan stores a TID list. A TID list (Transaction ID list) contains the identifier of each graph in the collection holding the corresponding subgraph. TID lists are created during the enumeration process and they are used for determining all possible children of s, through subgraph isomorphism tests that allow to find all the embeddings. The length of each TID list is the support of its corresponding candidate.

4 Frequent Connected Subgraph Mining

In this section, we introduce some novel properties of the DFS code, which are useful for reducing the number of duplicate candidates as well as the number of canonical form tests in FCS mining. This paper shows the results of a first work that proposes reusing previous test results for improving duplicate candidate detection.

Moreover, we define boundaries between useful and duplicate candidates, which can be efficiently detected using the cut property. Finally, a new FCS mining algorithm called gRed is proposed showing the usefulness of these properties.

4.1 Novel Properties of DFS Codes

Suppose that $s = (e_1, e_2, \dots, e_m)$ is a minimum DFS code representing a labeled graph $G = \langle V, E, L, l \rangle$, where $|V| = n$ and $|E| = m$. Let $\mathrm{RE}(s)$ be the set of tuples representing the rightmost path extensions of s that emerge when gSpan traverses a graph collection D. This set can be partitioned into several subsets

$$\mathrm{RE}(s) = B_0(s) \cup \dots \cup B_{n-1}(s) \cup F_0(s) \cup \dots \cup F_{n-1}(s),$$

where $B_i(s)$ contains the backward rightmost path extensions of s with destination at vertex v_i (the vertex with index i), and $F_i(s)$ is the forward extension set from vertex v_i. For each vertex v_i in the rightmost path, $i \neq n-1$, the tuple f_i represents the forward edge from v_i to its consecutive vertex in the rightmost path. The notation e^{-1} is a tuple representing the reverse edge of the tuple e; that is, if $e = (i, j, a, b, c)$ then $e^{-1} = (j, i, c, b, a)$ for each $e \in K_n^2 \times L^3$.
For example, Fig. 1 (A) shows the DFS tree of the code

$$s = (0,1,A,-,B)(1,2,B,-,C)(2,0,C,-,A)(2,3,C,-,C)(0,4,A,-,B)(4,5,B,-,C)(4,6,B,-,C);$$

the rightmost path of s is (v_0, v_4, v_6). The fourth component of each tuple (that is the edge label) is set to "−" for indicating an undefined or identical labels. In Fig. 1 (B), the backward extension sets

$$B_0(s) = \{(6,0,C,-,A)\},$$
$$B_4(s) = \{(6,4,C,-,B)\},$$

are shown; and in Fig. 1 (C), the forward extension sets

$$F_0(s) = \{(0,7,A,-,C)\},$$
$$F_4(s) = \{(4,7,B,-,D),(4,7,\mathrm{B},-,\mathrm{C})\},$$
$$F_6(s) = \{(6,7,C,-,B),(6,7,C,-,C)\},$$

are shown.

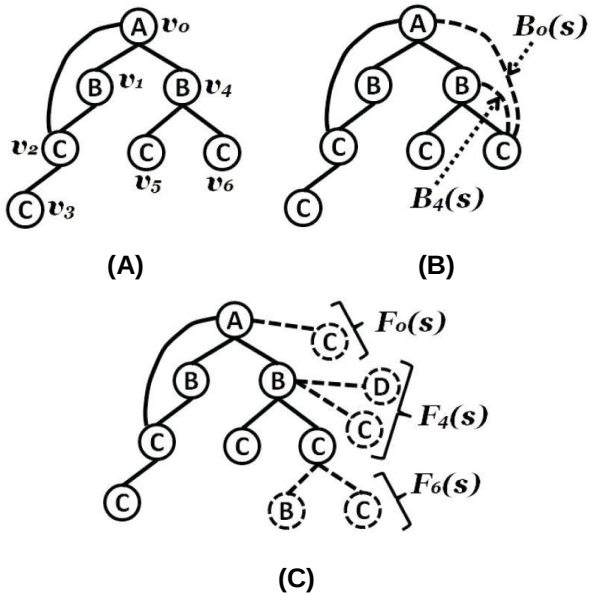

Fig. 1 Example of a DFS tree (A), its partitions of the backward extensions (B) and forward extensions (C)

The following two theorems are called non-minimality conditions for the children of s. These theorems allow knowing the result of some canonical form tests without performing the exhaustive procedure.

Theorem 1. Let s be a minimum DFS code, let v_i be a vertex in the rightmost path of s and suppose that $i \neq n-1$. If $e \in F_i(s)$ and $e \prec_l f_i$, then $s_0 = s \diamond e$ is a non-minimum DFS code.

Proof. Let h be an integer number such that $e_h = f_i$. The tuples e_h and e start from v_i and they represent forward edges in s_0. Therefore, we can perform a DFS traversal visiting first e and then e_h or vice versa. If e is visited immediately before e_h, the resulting DFS code has the following format

$$r_0 = (e_0, \ldots, e_{h-1}, e, b_h, \ldots, b_m),$$

where b_j is e_j with another subindices for each $j \geq h$. The codes s_0 and r_0 have the same prefix $e_0, \ldots, e_{h-1}$ and we are considering that $e \prec_l e_h = f_i$; therefore, $r_0 \prec_s s_0$ by the condition (2) of the DFS lexicographic order. Therefore, we conclude that s_0 is a non-minimum DFS code.

A similar result for backward extensions is showed in the following theorem.

Theorem 2. Let s be a minimum DFS code, let v_i be a vertex in the rightmost path of s. If $e \in B_i(s)$ and $e^{-1} \prec_l f_i$, then $s_0 = s \diamond e$ is a non-minimum DFS code.

Proof. As in the proof of Theorem 1, let h be the integer number such that $e_h = f_i$. We can perform a DFS traversal visiting first e^{-1} and then e_h or vice versa. If e^{-1} is visited immediately before e_h, the resulting DFS code has the following format

$$r_0 = (e_0, \ldots, e_{h-1}, e^{-1}, b_h, \ldots, b_m),$$

The codes s_0 and r_0 have the same prefix $e_0, \ldots, e_{h-1}$ and we assume that $e^{-1} \prec_l e_h = f_i$; therefore, $r_0 \prec_s s_0$. Thus, we conclude that s_0 is a non-minimum DFS code.

The statement of Theorem 2 is quite similar to the pre-pruning stage of gSpan (see Section 3.3). However, it is important to notice that Theorem 2 only proposes to compare e regarding f_i unlike gSpan where each backward extension e should be compared against any forward edge from v_i.

Let $\overline{\text{RE}}(s)$ be the set obtained from $\text{RE}(s)$ by removing the extensions whose generated DFS codes, according to Theorems 1 and 2, are non-minimum. In $\overline{\text{RE}}(s)$ could exist other extensions whose generated DFS codes are non-minimum. The following theorem provides a method to reuse previous calculations for predicting the results of other canonical form tests; therefore, it is called reuse condition.

Theorem 3. Let s be a minimum DFS code, let v_i be a vertex in the rightmost path of s and let E be one of the sets $F_i(s)$ or $B_i(s)$. If $e, b \in E$; then, the following statements are true

1. if $s \diamond e$ is a minimum DFS code and $e \preccurlyeq_l b$, then $s \diamond b$ is a minimum DFS code;
2. if $s \diamond e$ is a non-minimum DFS code and $b \preccurlyeq_l e$, then $s \diamond b$ is a non-minimum DFS code.

Proof. Let us prove each case separately.

In the first statement, we have that $s \diamond e$ is a minimum DFS code and $e \preccurlyeq_l b$. Suppose that $s \diamond b$ is a non-minimum DFS code, then there is at least one code $s_1 = (a_1, a_2, \ldots, a_{m+1})$ such that $s_1 \prec_s s \diamond b$. Using the definition of the DFS lexicographic order $\prec_s$, there is an integer t, $0 \leq t \leq m$, such that $a_k = e_k$ for all $k < t$, and $a_t \prec_e e_t$. As it can be noticed, $t < m + 1$ because s is a minimum DFS code. Thus, by the condition (2), $s_1 \prec_s s$.

Since e and b start from the same vertex, we can replace the edge representing b in s_1 by the edge e. Assume r_1 is the code obtained when replacing b by e in s_1; this is a valid code for the graph coded by $s \diamond e$ and we have $r_1 \prec_s s_1 \prec_s s$. Using the condition (3), $r_1 \prec_s s \prec_s s \diamond e$. Then, $s \diamond e$ is a non-minimal child of s, representing a contradiction. Therefore, the initial assumption ($s \diamond b$ is a non-minimum DFS code) must be false. Thus, we conclude the proof of the first statement.

In the second statement, we have that $s \diamond e$ is a non-minimum DFS code and $b \preccurlyeq_l e$. Then, there is at least one code $s_1 = (a_1, a_2, \ldots, a_{m+1})$ such that $s_1 \prec_s s \diamond b$. Let t be the integer such that $0 \leq t \leq m$, $a_k = e_k$ for all $k < t$, and $a_t \prec_e e_t$. Thus, by the condition (2), we have $s_1 \prec_s s$ (it does not contradict the fact that s is a minimum DFS code since s_1 and s represent different graphs). Since e and b start from the same vertex, we can replace the edge representing e in s_1 by the edge b. The resulting code (assume it is r_1) is a valid DFS code for the graph coded by $s \diamond b$ and we have $r_1 \prec_s s_1 \prec_s s \prec_s s \diamond b$. Therefore, $s \diamond b$ is a non-minimum DFS code.

The tuples in the set $\overline{\text{RE}}(s)$ can be sorted according to $\prec_e$; thus, the subsets in the aforementioned partition can be placed in ascending order as

$$\bar{B}_0(s), \ldots \bar{B}_{n-1}(s), \bar{F}_0(s), \ldots, \bar{F}_{n-1}(s),$$

where each $\bar{B}_i(s)$ (or $\bar{F}_i(s)$) is obtained from $B_i(s)$(or $F_i(s)$) by removing the extensions whose generated DFS codes, according to Theorems 1

and 2, are non-minimum. The tuples inside $\bar{B}_i(s)$ (or $\bar{F}_i(s)$) are sorted in ascending order using the order in labels $\prec_l$. Henceforth, this arrangement in $\overline{\text{RE}}(s)$ is assumed.

Based on Theorem 3, the boundaries between the extensions producing minimum and non-minimum DFS codes can be located. The following result called cut property shows such allocation.

Theorem 4. Let s be a minimum DFS code, let v_i be a vertex in the rightmost path of s and let E be one of the sets $B_i(s)$ or $F_i(s)$. Suppose that E is sorted in ascending order according to $\prec_l$. If there are extensions in E that produce non-minimum DFS codes then they are at the beginning of E. Besides, if there are extensions that produce minimum DFS codes they are at the end of E.

Proof. If all extensions in E produce non-minimum DFS codes or all of them produce minimum DFS codes then the corollary is true. Therefore, suppose that there are both kinds of extensions in E. Let $\hat{e}$, $\check{e} \in E$ be two extensions such that $\hat{e}$ produces a non-minimum code and $\check{e}$ produces a minimum code. It is easy to see that $\hat{e} \prec_l \check{e}$, because in the opposite case $s \diamond \check{e}$ should be a non-minimum code (see Theorem 3), contradicting the hypothesis. Therefore, we obtain that any extension in E producing a non-minimum code is before any other one producing a minimum code.

The cut property states that each set $B_i(s)$ can be partitioned into two disjoint subsets, $B_i(s) = \hat{B}_i(s) \cup \check{B}_i(s)$, where the extensions in $\hat{B}_i(s)$ produce non-minimum codes, the extensions in $\check{B}_i(s)$ produce minimum codes and the tuples in $\hat{B}_i(s)$ are before of the ones in $\check{B}_i(s)$ regarding $\prec_l$. In a similar way, each set $F_i(s)$ can be partitioned as $F_i(s) = \hat{F}_i(s) \cup \check{F}_i(s)$.

Thus, the set $\text{ME}(s) \subseteq \overline{\text{RE}}(s)$ of all extensions producing minimum codes is represented by means of the second subsets of these partitions (4).

$$\text{ME}(s) = \check{B}_0(s) \cup \ldots \cup \check{B}_{n-1}(s) \cup \check{F}_0(s) \cup \ldots \cup \check{F}_{n-1}(s). \quad (4)$$

These proposed properties are important for FCS mining when candidates are represented by DFS codes since the extensions producing non-minimum codes are eliminated because they represent duplicate candidates. The following sections show how these results could be used in FCS mining.

4.2 The gRed Algorithm

The properties introduced in Section 4.1 can be used to improve the mining process of gSpan and all algorithms based on it. In this section, a new algorithm called gRed based on these properties is introduced. The pattern growth strategy of gRed is shown in Algorithm 1 and it is explained in this section.

```
Procedure gRed-Growth(s, D, δ, S)
Input: s - a minimum DFS code (representing a frequent subgraph), D - collection of graphs, δ - support threshold.
Output: S - mining results
1  S ← S ∪ {s};
2  RE(s) ← The set of tuples e such that the extension s ◇ e takes place in D and e is not filtered by non-minimality conditions (pre-filtering);
3  Remove from RE(s) the infrequent extensions;
4  ME(s) ← The set of all extensions producing minimum codes which is calculated directly from RE(s) using the cut property and binary searches (post-filtering);
5  foreach extension e ∈ ME(s) do
6    gRed-Growth(s ◇ e, D, δ, S);
7  end
```

Algorithm 1. Pseudo-code of the gRed pattern growth approach

Let s be a minimum DFS code representing a FCS in a graph collection D. Suppose that $|\text{RE}(s)| = N$, then obtaining the set of all extensions producing minimum DFS codes, $\text{ME}(s)$, through the procedure proposed by the

gSpan scheme would require N exhaustive canonical form tests, since each candidate must be checked. Henceforward, it will be shown that the number of such tests could be reduced.

Taking into account that the non-minimality conditions can be checked in constant time, they could be executed inside the pattern growth process. Thus, the candidates that need to be tested are obtained directly from the extension set $\overline{\text{RE}}(s)$. This process is called pre-filtering (see line 2 of Algorithm 1). It is important to remember that $\left|\overline{\text{RE}}(s)\right| = \bar{N} \leq N$.

The cut property is used to obtain $\text{ME}(s)$ directly from $\overline{\text{RE}}(s)$ having a procedure more efficient than the one used in the gSpan scheme. This procedure called post-filtering (see line 4 of Algorithm 1) will be described as follow.

Suppose that $|\bar{B}_0(s)| = b_0$,..., $|\bar{B}_{n-1}(s)| = b_{n-1}$, $|\bar{F}_0(s)| = f_0$,..., $|\bar{F}_{n-1}(s)| = f_{n-1}$, then

$$\sum_i b_i + \sum_i f_i = \bar{N}\,, \tag{5}$$

where $\sum_i$ sums over the vertices in the rightmost path of s.

The gSpan scheme sorts the extensions in $\text{RE}(s)$ according to $\prec_e$. This sorting is maintained during the pattern growth process in line 2 of Algorithm 1. This fact is not exploited in the gSpan scheme. Theorem 5 shows a novel approach to reduce the number of canonical form tests (see the proof of this theorem).

Theorem 5. Let s be a minimum DFS code, let v_i be a vertex in the rightmost path of s and let E be one of the sets $\bar{B}_i(s)$ or $\bar{F}_i(s)$ and suppose that $|E| > 0$. Then, the number of exhaustive canonical form tests required to separate E into $\hat{E} \cup \check{E}$ (notations clarified in Section 4.1) is at most $\log_2(|E|) + 1$.

Proof. If $|E| = 1$, only one test is required and the theorem is true.

Let $N(f)$ be the number of tests required for separating a set E with f elements ($|E| = f$).

Let e be the median of E regarding the order $\prec_l$ and let $E_< = \{b \in F \mid b \prec_l e\}$ and $E_> = \{b \in F \mid e \prec_l b\}$ be two sets containing the other elements in E. A canonical form test is performed for $s \diamond e$.

First case: Suppose that $s \diamond e$ is a minimum DFS code. Then, the first statement of Theorem 3 ensures that the DFS codes obtained from the extensions of $E_>$.

Second case: Suppose that $s \diamond e$ is a non-minimum DFS code. Then, the second statement of Theorem 3 guarantees that the DFS codes obtained from the extensions of $E_<$ are also non-minimum codes.

In both cases, more tests to separate the remaining sets $E_<$ or $E_>$ respectively are required. The cardinality of each remaining sets is always less or equal than $\lfloor f/2 \rfloor$.

In brief, a problem with size f is reduced to the same problem but with size less or equal than $\lfloor f/2 \rfloor$. It is important to notice that for this reduction only one test is performed. Therefore, the inequality $N(f) = N(\lfloor f/2 \rfloor) + 1$ is true and $N(1) = 1$. This recurrent inequality can be easily solved using the known algorithm analysis tools [4]. Thus, the proof is concluded.

Let K be the number of exhaustive canonical form tests to obtain $\text{ME}(s)$ from $\overline{RE}(s)$. Using Theorem 5, we conclude that

$$K \leq \sum_i (\log_2(b_i) + 1) + \sum_i (\log_2(f_i) + 1)\,, \tag{6}$$

where $\sum_i$ sums over the vertices in the rightmost path of s. Let r be the number of vertices in the rightmost path of s. Since $\log_2(x)$ is a convex function [18] in the interval $[1, +\infty)$, the inequality (6) is transformed in

$$K \leq 2r \log_2\left(\frac{\sum_i b_i + \sum_i f_i}{2r}\right) + 2r, \tag{7}$$

The argument of $\log_2(\cdot)$ in (7) can be simplified using (5); Finally, we obtain

$$K \leq 2r \log_2\left(\frac{\bar{N}}{2r}\right) + 2r, \tag{8}$$

The theoretical result showed in (8) states an upper bound of the number of canonical form tests required to obtain useful extensions of s. This bound depends on two parameters the number of extensions $\bar{N}$ (this number depends on the graph collection features such as density of edges, number of labels, etc.) and the length of the rightmost path of s. As it can be noticed, the parameter $\bar{N}$ is affected by a $\log_2(\cdot)$ function; therefore, the proposed strategy reduces the

dependence of the graph collection features regarding the gSpan scheme.

The procedure gRed-Growth of Algorithm 1 recursively generates all candidate codes (graphs) starting from a DFS code s representing a FCS. The pre-filtering and post-filtering processes are used to reduce the number of canonical form tests. In the proof of Theorem 5, we can appreciate the process for performing a binary search. For each partition subset $\bar{B}_i(s)$ (or $\bar{F}_i(s)$) of $\overline{RE}(s)$, a binary search is performed for filtering duplicate candidates (post-filtering). The post-filtering stage is the main difference of gRed and the previously published version gRed-v1 [6], since gRed-v1 does not use binary search for removing duplicates.

```
Procedure gRed(D, δ, S)
Input: D - collection of graphs, δ - support threshold.
Output: S - mining results
1  Remove infrequent vertices and edges from D;
2  S ← the set of all frequent vertices in D;
3  S¹ ← the set of all frequent edges in D (DFS codes with only one edge);
4  foreach DFS code s ∈ S¹ do
5    Initialize the TID list s.L by the graphs which contains the edge of s;
6    gRed-Growth(s, D, δ, S);
7    D ← D\s;
8    if |D| < δ then break;
9  End
```

Algorithm 2. Pseudo-code of gRed

Completing the description of gRed, the procedure gRed-Growth is invoked from the main procedure (see Algorithm 2). This main procedure starts by removing all infrequent vertices and edges. Next, the procedure gRed-Growth is invoked for each frequent edge (DFS code with size 1) for traversing the search space in a depth-first way. After a frequent edge has been processed, the edge is dropped from the graphs in the collection D; thus, it will not be used as possible extension in the next iterations.

It is important to highlight that the pre-filtering and post-filtering processes introduced by gRed over the gSpan scheme can be also introduced in any other algorithm or implementation for FCS mining using DFS codes.

5 Experimental Results

In order to evaluate the usefulness of the proposed properties to reduce the number of canonical form tests, we compared gRed against gSpan and the already published version of gRed called gRed-v1 [6]. The algorithms considered as improvements of the gSpan scheme (for example ADI [20], Edgar [21], and FSP [8]) are not included in the comparison, because they use DFS codes; therefore, the pre-filtering and post-filtering processes introduced in gRed could be also adapted in such algorithms to reduce the number of canonical form tests. The usefulness of the novel properties introduced in this paper could be illustrated without including these algorithms in the comparison.

Additionally, we include a comparison of gRed against gSpan, MoFa, FFSM, and Gaston regarding their runtimes and memory requirements. These algorithms are the most commonly referenced and the most successful in previous comparative studies [16,22]. They were implemented in the common Java framework which is distributed under GNU license [22]. The implementation of gRed was developed in this framework.

All the experiments were done using an Intel Core 2 Duo PC at 2.2 GHz with 2 GB of RAM. The SUN Java Virtual Machine (JVM) 1.5.0 was used to run the algorithms.

5.1 Collections of Graphs

The biochemical data collections, specifically the molecular datasets, constitute one of the main application fields for graph mining. Therefore, this kind of collections has been commonly us d to evaluate the performance of FCS mining algorithms.

The PTE collection is the smallest dataset (according with the number of graphs) considered in this work; it contains only 337 graphs representing molecules used in the predictive toxicological evaluation challenge [19]. In spite of its small size, PTE has a big amount of FCSs; for example, it has 136 981 FCSs using 2% of the collection size as support threshold.

In this work, we use two medium size collections CAN2DA99 (http://dtp.nci.nih.gov/docs/cancer/cancer _data.html) and HIV (http://dtp.nci.nih.gov/docs/aids/ aids_data.html). CAN2DA99 contains the graph representation of 32 557 molecules discovered in carcinogenic tumors and HIV includes the description of 42 689 molecular structures of the human immunodeficiency virus. NCI (http://cactus.nci.nih.gov/ncidb2/download. html) is the biggest dataset (237 771 graphs) used in our experiments. This dataset contains molecules from several sources.

5.2 Experiments

In our experiments we used low support thresholds to evaluate the performance of the algorithms, because these thresholds are very important in data mining applications [7]. Moreover, there are some applications like classification and clustering where frequent complex graph structures are important, and these complex structures can only be found with low support thresholds [10]. Additionally, high thresholds are commonly fulfilled by connected subgraphs with small size regarding the number of vertices, edges, or cycles. Moreover, almost all recent algorithms achieve good execution times for high support thresholds; and the differences among algorithm performances are more distinguishable for low support thresholds.

The experiments in this paper are conceived for evaluating the usefulness of the pre-filtering and post-filtering stages. However, the usefulness of non-minimality was presented in the previously published version of gRed-v1 [6]. In this sense, the number of duplicates in all cases was significantly reduced by the pre-filtering stage; for example, in PTE using 2% of the collection size as support threshold, this stage reduced almost 60% of the duplicates regarding gSpan.

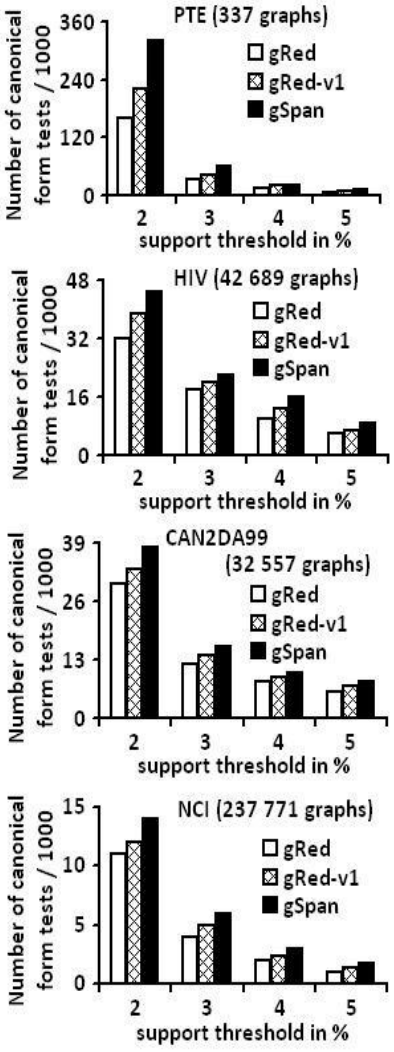

Fig. 2 The number of canonical form tests performed by gRed and gSpan in datasets PTE, CAN2DA99, HIV, and NCI varying the support threshold

The first experiment of this paper is conceived for evaluating the usefulness the cut property in the post-filtering stage of gRed. The algorithms gSpan, gRed-v1, and gRed use DFS codes to represent graph candidates during the mining process, unlike MoFa, FFSM and Gaston which

use different approaches. Therefore, in this experiment, the algorithms gRed, gSpan, and gRed-v1 were compared regarding the number of exhaustive canonical form tests (see Fig. 2). As it can be seen, the number of such expensive tests was reduced by gRed in all cases.

Additionally, we included a performance comparison involving gRed, gRed-v1, gSpan, MoFa, FFSM and Gaston. This comparison includes the evaluation of runtimes. The runtime for the algorithms was recorded varying the support threshold in the four datasets (see Fig. 3). In these experiments, Gaston was unable to complete the execution for some low supports threshold in CAN2DA99, HIV, and NCI due to memory requirements. For the same reason, FFSM and MoFa were unable to process NCI for the evaluated support thresholds. Moreover, the runtime of MoFa was truncated in several times for highlighting the scores of the others algorithms.

As we can see, gRed beats gSpan and gRed-v1 in all tests. It is known that much of runtime of gSpan and gRed-v1 is spent by subgraph isomorphism tests during the candidate enumeration process [6]. Since gRed also uses this kind of tests, they had similar behavior. However, gRed showed better performance in PTE, since the number of canonical form tests was reduced considerably, through the use of the properties introduced in Section 4.1 (see Fig. 2).

Gaston was unable to complete the execution for low support thresholds (less than 3% in CAN2DA99, less than 5% in HIV, and less than 6% in NCI) due to its high memory requirements. However, in the smallest collection (PTE), the best runtimes were achieved by Gaston. The worst runtimes were achieved by FFSM and MoFa while the best runtimes on the large collections were obtained by gRed and gSpan for the evaluated support thresholds.

6 Conclusions

In this paper, two non-minimality conditions, a reuse condition and a cut property of DFS codes, which are useful for graph mining, were introduced. The non-minimality conditions allow the reduction of the number of candidates. The

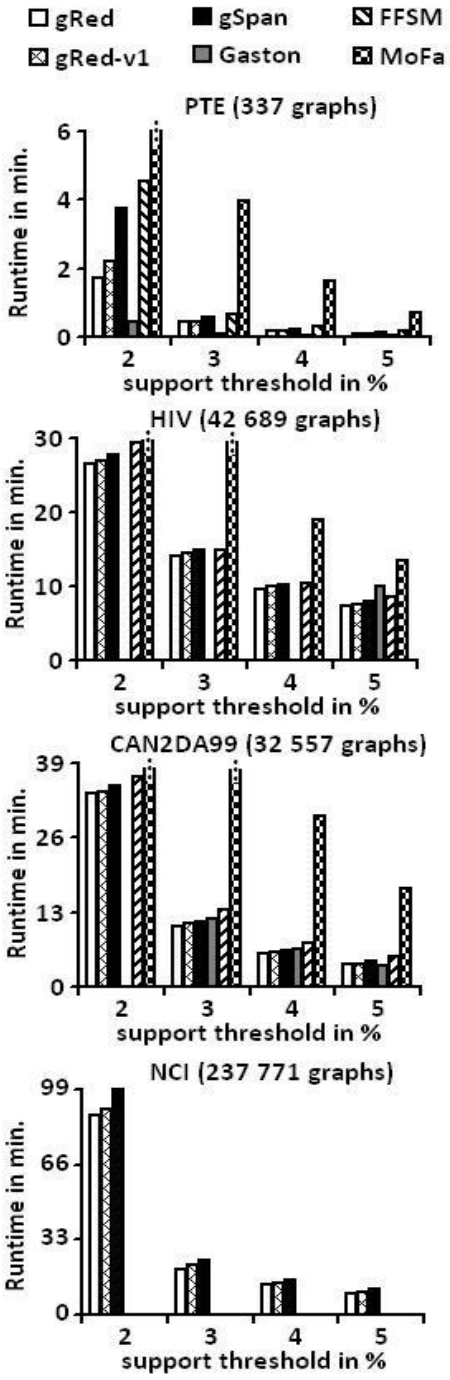

Fig. 3. Runtime with datasets PTE, CAN2DA99, HIV, and NCI varying the support threshold

reuse condition enables to reduce the number of canonical form tests by reusing previous test results. Besides, the reuse condition allows defining boundaries between canonical and non-canonical forms in the candidate space by means of the cut property.

Additionally, a new algorithm (gRed) for FCS mining using the proposed properties was introduced. Theoretical analysis and experimental results show the good performance of our proposal.

We compared gRed against gSpan and other commonly referenced algorithms. The experimentation showed that our proposal overcomes gSpan in all tests reducing significantly the number of canonical form tests. Moreover, gRed achieved better runtimes than the other tested algorithms when graph collections were large. The usefulness of the novel DFS code properties for graph mining was corroborated, showing that these properties allow reducing the number of duplicate candidates, as well as the number of canonical form tests.

It is important to highlight that this paper shows the results of a first work that proposes reuse conditions to improve duplicate candidate detection in graph mining.

In this research, we have shown that DFS codes have not been sufficiently studied and new properties can be found to improve the mining process. Our proposal showed that the time spent in duplicate candidate detection can be reduced during the mining process.

As future work, we are going to develop hybrid approaches combining gRed with fast evaluation strategies for reducing the cost of isomorphism tests during the mining process in order to reach better runtimes.

References

1. **Agrawal, R. & Srikant, R. (1994).** Fast Algorithms for Mining Association Rules. In J.B. Bocca, M. Jarke & C. Zaniolo (Eds.), 20th *International Conference on Very Large Data Bases*, Santiago de Chile, Chile, 487–499.
2. **Borgelt, C. (2006).** Canonical Forms for Frequent Graph Mining. In R. Decker & H.J. Lenz (Eds.), *30th Annual Conference of the Gesellschaft für Klassifikation*, Berlin, Germany, 337–349.
3. **Borgelt, C. & Berthold, M.R. (2002).** Mining Molecular Fragments: Finding Relevant Substructures of Molecules. IEEE *International Conference on Data Mining*, Maebashi, Japan, 51-58.
4. **Cormen, T.H., Leiserson, C.E., Rivest, R.L. & Stein, C. (2001).** Introduction to Algorithm (Second edition). Cambridge, Mass.: MIT Press.
5. **Diestel, R. (2005).** Graph Theory (Third edition). Berlin: Springer.
6. **Gago, A., Medina, J. E., Carrasco-Ochoa, J. A. & Martínez-Trinidad, J.F. (2008).** Mining Frequent Connected Subgraphs Reducing the Number of Candidates. *Machine Learning and Principles and Knowledge Discovery in Databases*. *Lecture Notes in Computer Science*, 5211, 365–376.
7. **Han, J., Cheng, H., Xin, D. & Yan, X. (2007).** Frequent Pattern Mining: Current Status and Future Directions. *Data Mining and Knowledge Discovery*, 15(1), 55–86.
8. **Han, S., Wee, K.N. & Yu, Y. (2007).** FSP: Frequent Substructure Pattern Mining. *6th International Conference on Information, Communications and Signal Processing*, Singapore, 1–5.
9. **Hernández, J.I. (2009).** Reactive Scheduling of DAG Applications on Heterogeneous and Dynamic Distributed Computing Systems, Abstract of PhD Thesis. *Computacion y Sistemas*, 13(2), 221–237.
10. **Hossain, M.S. & Angryk, R A. (2007).** GDClust: A Graph-based Document Clustering Technique. *7th IEEE International Conference on Data Mining Workshops*, Nebraska, USA, 417–422.
11. **Huan, J., Wang, W. & Prins, J. (2003).** Efficient Mining of Frequent Subgraph in the Presence of Isomorphism. *Third IEEE International Conference on Data Mining*, Florida, USA, 549–552.
12. **Inokuchi, A., Washio, T. & Motoda, H. (2000).** An Apriori based Algorithm for Mining Frequent Substructures from Graph Data. 4th *European Conference on Principles of Data Mining and Knowledge Discovery*, Lyon, France, 13–23.
13. **Inokuchi, A., Washio, T., Nishimura, K. & Motoda, H (2002).** *A Fast Algorithm for Mining Frequent Connected Subgraphs (RT0448)*. Japan: IBM Research.
14. **Koyuturk, M., Grama, A. & Szpankowski, W. (2004).** An Efficient Algorithm for Detecting Frequent Subgraphs in Biological Networks. *Bioinformatics*, 20(1), 200-207.
15. **Kuramochi, M. & Karypis, G. (2001).** Frequent Subgraph Discovery. *IEEE International*

Conference on Data Mining, California, USA, 313–320.

16. **Nijssen, S. & Kok, J.N. (2004).** A Quickstart in Frequent Structure Mining can Make a Difference. *Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Washington, USA, 647–652.
17. **Nijssen, S. & Kok, J. (2006).** Frequent Subgraph Miners: Runtimes Don't Say Everything. In T. Gartner, G. C. Garriga & T. Meinl, (Eds.), *Fourth Workshop on Mining and Learning with Graphs*, Berlin, Germany, 173–180.
18. **Rudin, W. (1976).** Principles of Mathematical Analysis (3rd edition). New York: McGraw-Hill.
19. **Srinivasan, A., King, R.D., Muggleton, S.H. & Sternberg, M.J.E. (1997).** The Predictive Toxicology Evaluation Challenge. *15th International Joint Conference on Artificial Intelligence,* Nagoya, Japan, 1, 4–9.
20. **Wang, C., Wang, W., Pei, J., Zhu, Y. & Shi, B. (2004).** Scalable Mining of Large Disk-based Graph Databases. *Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Washington, USA, 316–325.
21. **Worlein, M., Dreweke, A., Meinl, T., Fischer, I. & Philippsen, M. (2006).** Edgar: the Embedding-based Graph Miner. T. Gartner, G. C. Garriga & T. Meinl, (Eds.), *4th International Workshop on Mining and Learning with Graphs*, Berlin, Germany, 221–228.
22. **Worlein, M., Meinl, T., Fischer, I. & Philippsen, M. (2005).** A Quantitative Comparison of the Subgraph Miners Mofa, gSpan, FFSM, and Gaston. *Knowledge Discovery in Databases: PKDD 2005. Lecture Notes in Computer Science*, 3721, 392–403.
23. **Wu, J. & Chen, L. (2008).** Mining Frequent Subgraph by Incidence Matrix Normalization. *Journal of Computers*, 3(10), 109–115.
24. **Yan, X. & Han, J. (2002).** gSpan: Graph-Based Substructure Pattern Mining. *IEEE International Conference on Data Mining (ICDM 2002)*, Maebashi, Japan, 721–724.
25. **Yan, X. & Han, J. (2002).** *gSpan: Graph-Based Substructure Pattern Mining (UIUCDCS-R-2002-2296)*. Illinois, USA: University of Illinois at Urbana-Champaign.

Andrés Gago Alonso received his B.S. degree in Computer Science from the Havana University in 2004. He holds a M.Sc. degree in Mathematics from the same university in 2007. He completed his Ph.D. Degree in Computational Sciences at the National Institute of Astrophysics, Optics and Electronics (INAOE) in January 2010. His research interests include but are not restricted to knowledge discovery and data mining in graph-based content. Currently, he works as a fulltime researcher in the Advanced Technologies Application Centre (CENATAV), Cuba.

Jesús Ariel Carrasco Ochoa received his Ph.D. in Computer Science from the Center for Computing Research of the National Polytechnic Institute (CIC-IPN), Mexico, in 2001. He works as a fulltime researcher at Computer Science Department of the National Institute for Astrophysics, Optics and Electronics (INAOE) of Mexico. His current research interests include Logical Combinatorial Patter Recognition, Data Mining, Feature and Prototype Selection, Document Analysis, Fast Nearest Neighbor Classifiers and Clustering.

José Eladio Medina Pagola received his B.S. in Cybernetic Mathematics from the Havana University in 1977 and his Ph.D. from the Higher Polytechnic Institute "José A. Echeverría" (ISPJAE) in 1996. His research interests include but not restricted to knowledge discovery and data mining, association rules, clustering, computational linguistic, information retrieval and text mining. He is currently a Senior Researcher and Research Deputy Director of the Advanced Technologies Application Centre (CENATAV), Cuba.

José Francisco Martinez Trinidad received his B.S. degree in Computer Science from Physics and Mathematics School of the Autonomous University of Puebla (BUAP), Mexico in 1995, his M.Sc. degree in Computer Science from the faculty of Computers Science of the Autonomous University of Puebla, Mexico in 1997 and his Ph.D. degree in the Center for Computing Research of the National Polytechnic Institute (CIC, IPN), Mexico in 2000. Professor Martinez-Trinidad edited/authored six books and over one hundred journal and conference papers, on subjects related to Pattern Recognition.

Article received on 12/03/2010; accepted 05/04/2011.