Full Duplicate Candidate Pruning for Frequent Connected Subgraph Mining

Andrés Gago-Alonso[1,2] Jesús A. Carrasco-Ochoa[2,*]
José E. Medina-Pagola[1]
José Fco. Martínez-Trinidad[2]

[1] Advanced Technologies Application Center
7a ♯ 21812, Siboney, Playa, CP: 12200, Havana, Cuba
{agago,jmedina}@cenatav.co.cu

[2] Computer Science Department
National Institute of Astrophysics, Optics and Electronics
Luis Enrique Erro ♯ 1, Sta. María Tonantzintla, CP: 72840, Puebla, México
{ariel,fmartine}@inaoep.mx

April 27, 2010

Abstract

Support calculation and duplicate detection are the most challenging and unavoidable subtasks in frequent connected subgraph (FCS) mining. The most successful FCS mining algorithms have focused on optimizing these subtasks since the existing solutions for both subtasks have high computational complexity. In this paper, we propose two novel properties that allow removing all duplicate candidates before support calculation. Besides, we introduce a fast support calculation strategy based on embedding structures. Both properties and the new embedding structure are used for designing two new algorithms: gdFil for mining all FCSs; and gdClosed for mining all closed FCSs. The experimental results show that our proposed algorithms get the best performance in comparison with other well known algorithms.

[*]Corresponding author

1 Introduction

Frequent connected subgraph (FCS) mining in labeled-graph collections is an active research topic in data mining, with wide applications. FCS mining is the process of finding connected subgraphs that frequently occur in a collection of labeled graphs. Examples of labeled-graph collections can be found in diverse sources: chemical compound databases, XML documents, citation networks, biological networks, and so forth. As a consequence, several FCS mining algorithms have been proposed [10]. The present work is focused on algorithms for mining the complete set of FCSs, and the closed FCSs, which are a compact representation allowing the reconstruction of the whole set of FCSs including their frequencies.

SUBDUE [11] was one of the first proposals for solving the frequent subgraph mining problem. This approach is based on minimum description length and background knowledge. Another proposal for frequent subgraph mining in chemical compounds datasets was developed using inductive logic programming [5]. However, the first algorithm for finding all frequent (connected or unconnected) subgraphs in a collection of labeled graphs was AGM [15]. This algorithm was followed by FSG [16] and AcGM [14] algorithms for mining all frequent connected subgraphs. Both algorithms are based on the original Apriori algorithm [1] for mining frequent itemsets.

Later, pattern growth based algorithms such as gSpan [21], CloseGraph [23], MoFa [3], Moss-MoFa [4], FFSM [13], Gaston [17], and gRed [8] were developed. Previous comparative studies have shown that pattern growth based algorithms have better performance than Apriori based ones [18, 20]. Therefore, in this paper we will focus on pattern growth based algorithms.

Duplicate detection and support calculation are the two hardest subtasks in FCS mining. A duplicate candidate is a subgraph that has already been considered in a previous step, but it appears again during the search. The problem of duplicate candidates is faced by representing the subgraphs with an unique code called canonical form (CF). Candidate enumeration strategies are commonly defined using these representations, trying to avoid non-canonical forms by performing CF tests which have very high computational complexity [2]. The DFS code (Depth First Search code) is an example of a promising kind of canonical form for FCS mining [21, 18]. On the other side, support calculation requires the expensive subgraph isomorphism tests in FCS mining. Embedding structures used by MoFa, FFSM, and Gaston avoid this kind of tests, precalculating and storing subgraph isomorphism embeddings.

On the other hand, mining all frequent connected subgraphs may generate an exponential number of frequent patterns. Closed frequent connected subgraphs have been used to reduce the number of patterns resulting from mining [23, 4, 9]. Moreover, this set of patterns allows us to reconstruct the whole set of all frequent connected subgraphs. Pruning techniques developed in closed FCS mining also increase the efficiency of graph mining algorithms. The most important algorithms for closed FCS mining are CloseGraph [23] and Moss-MoFa [4].

This work is an extension of [7], where we presented preliminary results. In such conference paper, a new algorithm for FCS mining, called gdFil, was introduced. A cut property of the DFS code was used in gdFil, for detecting all duplicate candidates before support calculation. Moreover, support calculation in gdFil was faced using a new kind of embedding structure called DFSE (Depth First Search Embedding). In the present work, we further extend the published results in a substantive way, including a more detailed explanation about cut properties, gdFil, and the DFSE structure. The proof of the cut property is also included in this paper jointly with another property called cut distribution property, which also supports the proposed duplicate detection strategy. The proposed properties and the DFSE structure can be used to mine closed FCSs; therefore, an extension of gdFil, called gdClosed, for closed FCS mining is also introduced. The proposed algorithms gdFil and gdClosed are compared against other reported algorithms over real world and synthetic datasets.

The basic outline of this paper is as follows. Section 2 provides some basic concepts; it also contains the related work. The cut properties and the DFSE structure are introduced, discussed, and proved in section 3; this section also introduces the gdFil and gdClosed algorithms. The experimental results in real world and synthetic datasets are presented in section 4. Finally, conclusions of the research and some ideas about future directions are exposed in section 5.

2 Background

In this section, we start providing the background and notation used in the following sections. Next, some definitions and properties of the DFS code are presented. Finally, the early termination pruning for closed FCS mining is also explained.

2.1 Basic Concepts

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 *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* (each edge is a set with exactly two vertices), L is a set of *labels* and $l : V \cup E \to L$ is a *labeling function* for assigning labels to vertices and edges.

Let $G_1 = \langle V_1, E_1, L_1, l_1 \rangle$ and $G_2 = \langle V_2, E_2, L_2, l_2 \rangle$ 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, and we use the notation $G_1 \subseteq G_2$, if $V_1 \subseteq V_2$ and $E_1 \subseteq E_2$. In this case, we say that G_2 is a *supergraph* of G_1.

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 and G_2 if $f : V_1 \to V_2$ is a bijective function and:

- $\forall v \in V_1, l_1(v) = l_2(f(v))$;
- $\forall \{u, v\} \in E_1, \{f(u), f(v)\} \in E_2 \wedge l_1(\{u, v\}) = l_2(\{f(u), f(v)\})$.

When there is an isomorphism between G_1 and G_2, we say that G_1 and G_2 are *isomorphic*. One way to approach the isomorphism test is using canonical forms (CF) for representing graphs [2].

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. One way to approach the subgraph isomorphism test in graph mining is using embedding structures [13, 17].

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 $P \subseteq V$ 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 $v_1 = v_k$ 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 connected graphs. 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.

Let $T = \langle V_T, E_T, L_T, l_T\rangle$ be a rooted tree with root v_0 and let $v, u \in V_T$. 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 $D = \{G_1, G_2, \dots, G_{|D|}\}$ be a collection of labeled graphs and let δ be a predefined threshold of frequency. The *support* of a graph g in D is defined as the number of graphs $G_i \in D$ such that there is a subgraph isomorphism from g to G_i. 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 all the connected subgraphs that frequently occur in a collection of graphs.

A graph g is *closed* in a collection D if there is no supergraph of g with the same support of g. *Closed FCS mining* consists in finding all closed frequent connected subgraphs in a collection of graphs.

2.2 DFS code

The DFS code is a kind of canonical graph representation proposed by Yan and Han [21]. This code was used in gSpan [21], CloseGraph [23], and gRed [8] for representing frequent graph candidates.

A *DFS tree* T is constructed when a DFS traversal in a graph $G = \langle V, E, L, l \rangle$ is performed. Each DFS traversal (DFS tree) defines a unique order among all the vertices; therefore, we can number each vertex according to this DFS order. Thus, each edge can be represented by a 5-tuple, $(i, j, l_i, l_{(i,j)}, l_j)$ where i and j are the numbers (subindices) of the vertices (v_i and v_j), l_i and l_j are the labels of these vertices respectively, and $l_{(i,j)}$ is the label of the edge connecting v_i and v_j. If $i < j$ it is a forward edge; otherwise it is a backward edge. In short, the order relation $e_1 \prec_e e_2$ holds if e_1 appears before e_2 in a DFS traversal. When e_1 and e_2 have same source and destination vertices, they are compared lexicographically using the order between labels $\prec_l$ (the last three components in each 5-tuple).

A *DFS code* is a sequence of edges built from a DFS tree sorting the edges according to $\prec_e$. The order $\prec_e$ can be also extended to a *lexicographic order* ($\prec_s$) between two DFS codes. The *minimum DFS code* is defined as the minimum sequence among all DFS codes of the same graph according to $\prec_s$ [21].

Suppose that $s = e_1, e_2, \ldots, e_m$ is a minimum DFS code. An edge e is a *rightmost path extension* of s if e connects the rightmost vertex with another vertex in the rightmost path (backward extension); or it introduces a new vertex connected from a vertex of the rightmost path (forward extension). In such cases, the DFS code $s' = s \diamond e$ is the code obtained extending s by e; s' is called a *child* of s and s is called a *parent* of s'.

Some properties of the DFS code were studied and used for reducing the number of candidates in gRed [8]. These properties are summarized as follows.

The set $RE(s)$ of all rightmost path extensions of s can be partitioned into several sets $RE(s) = B_0(s) \cup \ldots \cup B_{n-1}(s) \cup F_0(s) \cup \ldots \cup F_{n-1}(s)$, where $B_i(s)$ denotes the set that contains the backward extensions to the destination vertex v_i, and $F_i(s)$ is the set of forward extensions from vertex v_i.

For example, Fig. 1(a) shows the DFS tree of the code

$$\begin{aligned} s &= (0,1,A,-,B)(1,2,B,-,C)(2,0,C,-,A)(2,3,C,-,C)(0,4,A,-,B) \\ &\quad (4,5,B,-,C)(4,6,B,-,C); \end{aligned}$$

the rightmost path of s is (v_0, v_4, v_6). The fourth component of each edge tuple (that is the edge label) is set to "$-$" for indicating an undefined or identical labels. In Fig. 1(b), the backward extension sets $B_0(s) = \{(6,0,C,-,A)\}$ and $B_4(s) = \{(6,4,C,-,B)\}$ are shown; and in Fig. 1(c), the forward extension sets $F_0(s) = \{(0,7,A,-,C)\}$, $F_4(s) = \{(4,7,B,-,D),(4,7,B,-,C)\}$, and $F_6(s) = \{(6,7,C,-,C)\}$ are shown. Thus, the extension set $RE(s)$ can be partitioned into $B_0(s) \cup B_4(s) \cup F_0(s) \cup F_4(s) \cup F_6(s)$.

For each vertex v_i in the rightmost path, $i \neq n-1$, the edge f_i denotes the forward edge from vertex v_i lying in the rightmost path. The notation e^{-1} is used to refer to the reverse edge of e.

The first two properties proposed in [8] were named *non-minimality conditions* and the last one *reuse condition*.

1. **Forward non-minimality condition:** If $e \in F_i(s)$ with $i \neq n-1$ and $e \prec_l f_i$, then $s' = s \diamond e$ is a non-minimum DFS code.
2. **Backward non-minimality condition:** If $e \in B_i(s)$ and $e^{-1} \prec_l f_i$, then $s' = s \diamond e$ is a non-minimum DFS code.
3. **Reuse condition:** Let E be one of the sets $F_i(s)$ or $B_i(s)$. If $e, e' \in E$, then, the following statements are true:
   (a) if $s \diamond e$ is a minimum DFS code and $e \preceq_l e'$, then $s \diamond e'$ is a minimum DFS code;
   (b) if $s \diamond e$ is a non-minimum DFS code and $e' \preceq_l e$, then $s \diamond e'$ is a non-minimum DFS code.

These properties will be used in this paper to propose two new properties, which allow removing all duplicate candidates before support calculation (full candidate pruning).

2.3 Early termination pruning for mining closed patterns

The CloseGraph algorithm [23] combines the use of DFS codes with the early termination pruning for achieving better runtimes in closed FCS mining.

Let g be a graph represented by its minimum DFS code s and let e be an extension of g which could be a rightmost path extension of s or not. Let $D = \{G_1, G_2, \ldots, G_{|D|}\}$ be a collection of labeled graphs. For each $G_i \in D$, $\varphi(g, G_i)$ denotes the number of subgraph isomorphisms from g to G_i. Thus, the *occurrence* of g in D is defined as $\mathcal{I}(g, D) = \sum_{i=1}^{|D|} \varphi(g, G_i)$.

The graph $g \diamond_x e$ denotes the resulting graph after adding the edge e to g. The operator $\diamond_x$ is used to indicate that e can be rightmost path extension or not. For each $G_i \in D$, $\phi(g, e, G_i)$ denotes the number of subgraphs of G_i that are isomorphic to g and can be extended using e. Let $\mathcal{L}(g, e, D)$ denote the sum $\sum_{i=1}^{|D|} \phi(g, e, G_i)$.

The extension e is called an *equivalent extension* of g in D if $\mathcal{L}(g, e, D) = \mathcal{I}(g, D)$. This definition implies that every closed child of g in D must contain $g \diamond_x e$. Therefore, only the children of $g \diamond_x e$ should be considered for mining closed patterns, and the children of g must be pruned from the search space (this pruning is called *early termination pruning*). Nevertheless, this statement is not true for all the cases, there are situations where the early termination pruning can not be applied. This fact is known as a *failure of early termination.*

A failure of early termination can be described as follows. Let e be an equivalent extension of g in D and let h be a supergraph of g such that $g \diamond_x e \not\subset h$. The graph h is called a *crossing situation* of e for g if e is not an *equivalent extension* of h. Crossing situations can cause the lack of patterns in graph mining, so they must be tested during the mining. Early termination fails when the equivalent extension e of g has at least a crossing situation.

The CloseGraph algorithm uses the early termination pruning in an efficient way for mining closed patterns [23]. One of the algorithms proposed in this paper uses this pruning for closed FCS mining in combination with a novel candidate pruning, also proposed in this paper.

3 Frequent connected subgraph mining

In this section, we introduce two novel properties of the DFS code, which are useful to remove all duplicate candidates in FCS mining, before support calculation. The *cut property* defines boundaries between useful and duplicated candidates. Moreover, these boundaries can be efficiently detected using the *cut distribution property.* A new embedding structure called DFSE will also be introduced. This structure is based on an edge sequence growing pattern strategy according to DFS codes.

Based on the novel properties and the DFSE structure, two new FCS mining algorithms are proposed: gdFil for mining all FCSs; and gdClosed for mining all closed FCSs. The DFSE structure is used, in both algorithms, to speed up the subgraph isomorphism tests, which allows getting a better efficiency in the enumeration process and support calculation. The enumeration strategy of gdFil and gdClosed is based on the DFS code; that is, candidates are represented by means of a sequence of edges. DFSE stores embeddings of edges unlike the structure used by Gaston, which uses embeddings of vertices.

3.1 The cut properties of the DFS codes

In this section, we present the *cut* and *cut distribution* properties, which are used in the gdFil and gdClosed algorithms. The proofs and some explanations about these properties are also included.

The *cut* property proposed in this paper is derived from the reuse condition, which is applied to forward and backward extensions as follows:

Theorem 1 (Cut property) *Let s be a minimum DFS code, and H be one of the sets $F_i(s)$ or $B_i(s)$ (the forward or backward extension sets for a vertex v_i respectively), and suppose that H is sorted in ascending order according to $\prec_e$. If there are extensions in H that produce non-minimum DFS codes then they are at the beginning of H. Besides, if there are extensions that produce minimum DFS codes they are at the end of H.*

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

Thus, each set H ($F_i(s)$ or $B_i(s)$) is divided into two partitions $H = \hat{H} \cup \check{H}$, the duplicate partition $\hat{H}$ (extensions that produce non-minimum DFS codes) and the useful partition $\check{H}$ (extensions that produce minimum DFS codes). The extensions from the duplicate partition are called *duplicate extensions* and the extensions from the useful partition are called *useful extensions*.

This property states that there could be a *non-canonical cut* $\hat{h} \in \hat{H}$ being the last in the duplicate partition, according to $\prec_e$. Besides, in the same way, there could be a *canonical cut* $\check{h} \in \check{H}$ being the first in the useful partition. It is important to highlight that these partitions could be empty; therefore, one of those cut elements may not exist. When one of these cut elements does not exist, we use the notation $\hat{h} = \epsilon$ or $\check{h} = \epsilon$ according to the case. If both elements exist, the non-canonical cut is immediately before the canonical cut.

The cut property was used in gdFil for removing all duplicates before support calculation. Additionally, this property is used in combination with early termination pruning for removing all duplicates in gdClosed.

The following theorem explains an important property about cut elements. This property is called *cut distribution property*, since it states some relationships of cuts between different sets H' and H.

Theorem 2 (Cut distribution property) *Let s be a minimum DFS code, and let H be one of the sets $F_i(s)$ or $B_i(s)$. Suppose that $H' \subseteq H$ is a subset of*

H. Let $\hat{h}$ and $\check{h}$ be the cuts of H, and let $\hat{h}'$ and $\check{h}'$ be the cuts of H'. Then the following statements are true:

1. *If $\hat{h}' \neq \epsilon$, then $\hat{h}' \preceq_e \hat{h}$.*

2. *If $\check{h}' \neq \epsilon$, then $\check{h} \preceq_e \check{h}'$.*

Proof. First, we will prove the first statement. Suppose that $\hat{h}' \neq \epsilon$. Since $H' \subseteq H$, we have $\hat{h}' \in H$. Specifically, we have $\hat{h}' \in \hat{H}$ since $\hat{h}'$ is a non--minimum DFS code. Therefore, we conclude that $\hat{h}' \prec_e \hat{h}$, since $\hat{h}$ is the last element in the duplicate partition $\hat{H}$.

Next, we will prove the second statement. Suppose that $\check{h}' \neq \epsilon$. Since $H' \subseteq H$, we have $\check{h}' \in H$. Specifically, we have $\check{h}' \in \check{H}$ since $\check{h}'$ is a minimum DFS code. Therefore, we conclude that $\check{h} \prec_e \check{h}'$, since $\check{h}$ is the first element in the useful partition $\check{H}$. □

In FCS mining, the whole set of all rightmost path extensions $RE(s)$ is built incrementally. The cut distribution property allows calculating cuts of $RE(s)$ in such incremental process.

3.2 The DFSE structure

In this section, we introduce the DFSE structure, which is used in the gdFil and gdClosed algorithms for improving support calculation. We include a graphical example and some explanations about the embedding structure.

Let $D = \{G_0, G_1, \ldots, G_{N-1}\}$ be a graph collection with N graphs which are represented using adjacency lists. Let e be an edge belonging to a DFS code, let $G \in D$ be a graph in the collection, and let $e_k \in E(G)$ be an edge of G where k $(0 \leq k < |E(G)|)$ is the edge identifier. We denote by $e.l_1$, $e.l_e$, and $e.l_2$ the labels of the first vertex, the edge and the second vertex of e respectively. We say that e and e_k are the same edge according to the labels (denoted as $e =_l e_k$)

if $(e.l_1, e.l_e, e.l_2) = (e_k.l_1, e_k.l_e, e_k.l_2)$ or $(e.l_1, e.l_e, e.l_2) = (e_k.l_2, e_k.l_e, e_k.l_1)$. In the first case, we say that $sign(e, e_k) = +1$, and in the second case $sign(e, e_k) = -1$.

If e is a non-symmetric edge ($e.l_1 \neq e.l_2$), the embedding list of e regarding G denoted as $L(e, G)$ is defined as follows:

$$L(e, G) \quad = \quad \{(e_k, \epsilon, \lambda) | e_k \in E(G), e =_l e_k \text{ and } sign(e, e_k) = \epsilon\}, \qquad (1)$$

where λ represents a null pointer. The third component of each element of $L(e, G)$ will be used for those codes with more than one edge. If e is a symmetric edge ($e.l_1 = e.l_2$), the function $sign(e, e_k)$ can take both values ($+1$ or -1). Thus, we consider in $L(e, G)$ any value $(e_k, \pm 1, \lambda)$ for each $e_k \in E(G)$ such that $e =_l e_k$.

Let s be a minimum DFS code. If s is a code with only one edge e the embedding list of s regarding G is denoted as $L(s, G) = L(e, G)$.

As it is shown in (2), the embedding list $L(s \diamond e, G)$ of a child of s is built from $L(s, G)$, during the execution of the `gdFilExtensions` procedure (see Fig. 2). The third component in the embedding tuples is a pointer to an embedding tuple of the parent. The notations $\tau.k, \tau.\epsilon$ and $\tau.p$ are used to refer to the edge identifier, the sign and the parent of the tuple τ, respectively.

$$\begin{aligned} L(s \diamond e, G) \quad = \quad & \{(e_k, \epsilon, p) | e_k \in E(G), e =_l e_k, sign(e, e_k) = \epsilon \text{ and} \\ & \text{is a pointer to a tuple in } L(s, G)\}. \end{aligned} \qquad (2)$$

Examples of embedding lists are illustrated in Fig. 3. In order to simplify the explanation of Fig. 3, we assume that each undirected edge, in the graphs of the collection, is represented according to the lexicographic order of vertex labels. For example in the graph of Fig. 3(B), the edge e_2 is stored as (v_1, v_0). In Fig. 3(C), the embedding list of s_0 regarding G_0 is $L(s_0, G_0) =$

$\{(e_2, -1, \lambda), (e_1, +1, \lambda)\}$. The embedding list of s_1 regarding G_0 is $L(s_1, G_0) = \{(e_4, +1, p_1), (e_5, +1, p_2)\}$, where p_1 and p_2 are pointers to $(e_2, -1, \lambda)$ in $L(s_0, G_0)$ since the edges e_4 and e_5 are extensions of e_2 in G_0.

Using the embedding list definition, we introduce the DFSE structure of a DFS code s regarding the collection D as:

$$ES(s) = \{(G, L) | G \in D, L = L(s, G) \text{ and } L \neq \emptyset\}. \tag{3}$$

For example in Fig. 3, the DFSE structure of s_i is ES_i for $i = 0, 1$, for simplicity only the graph G_0 is illustrated.

The support of s in D can be calculated as the number of elements in $ES(s)$. Moreover, the use of embedding lists avoids all exhaustive isomorphism tests since the whole embedding of a candidate graph can be obtained traversing the third component (a pointer to its parent tuple) of each embedding tuple (see lines 2–8 of `gdFilExtensions`).

3.3 The gdFil algorithm

It is known that non-minimality and reuse conditions do not allows removing all duplicate candidates [8]. Therefore, some canonical form (CF) tests are required for pruning all duplicates. CF tests are commonly preformed after pruning non-frequent candidates since these tests have high computational complexity [7, 8, 21].

Unlike previous algorithms, which only eliminate some duplicate candidates, gdFil uses the cut properties, introduced in section 3.1, for eliminating all duplicate candidates before support calculation. Moreover, gdFil uses the DFSE structure, introduced in section 3.2, for speeding up support calculation by avoiding all subgraph isomorphism tests.

Fig. 4 outlines the pseudo-code of the main procedure of gdFil. This procedure is quite similar to the one in gSpan [21] and gRed [8], but it includes the initialization of the DFSE structure. Then gdFil algorithm starts by removing all non-frequent vertices and edges. Next, for each frequent edge its embedding list is initialized using (1) and (3). Later, the `gdFilMining` procedure (see Fig. 5) is invoked for each frequent edge. At the end of each iteration, the used edge is dropped from the collection; that is, it will not be used anymore as a possible extension in the next iterations.

The `gdFilMining` procedure recursively generates all candidate graphs (DFS codes) that hold s, while the generated DFS codes are frequent. During an execution of this procedure, not all $RE(s)$, the set of extensions of s (see section 2.2 for clarifying this notation), is stored. In fact, only useful (non duplicate) candidates are stored in ME. However, the cuts of $RE(s)$ (see section 3.1 for clarifying these concepts) are calculated and used for removing such duplicates. These cuts are called *global cuts* since they are referring to the cuts of the whole set $RE(s)$.

In line 3, global cuts are initialized in ϵ, for indicating the non-existence of such cuts when $RE(s) = \emptyset$. It is important to notice that the extensions of s are calculated during the loop at lines 4–14.

In line 4, we can see that all the embeddings of s are traversed according to the DFSE structure $ES(s)$. For each embedding (G, L) of s, gdFil calculates the rightmost path extension set EX of s in G, initializing $EX = \emptyset$ at line 5. The `gdFilExtensions` procedure, explained in section 3.2, is used for calculating EX (see line 6). Next, each set EX is purged by removing all duplicates (lines 7–9 and 11). The set ME, which does not contain duplicate candidates, is updated by adding the elements of EX (line 13).

In line 7, some duplicate candidates are filtered applying some optimizations, which were presented for the first time in the gSpan algorithm [22]. For example, gdFil restricts extensions to edges lexicographically higher than the first edge in the code. Moreover, each backward extension with destination vertex v_j, of a minimum DFS code s, should be non smaller than any forward edge from v_j in s. These optimizations are performed during the candidate enumeration, before the duplicate detection process.

In line 8, the non-minimality conditions are applied for removing some duplicates from EX. The usefulness of this step was shown in the gRed algorithm [22]. These conditions can be checked in time $O(1)$ for each candidate, and they allow eliminating several duplicate candidates.

In line 9, the current values of the global cuts are used for removing some duplicates from EX. We use the expression "some duplicates" because the current values of global cuts are estimations of the correct value. For example in the first iteration of the loop at lines 4–14, this line has no effect since global cuts have the initial value ϵ. However, in the next iterations, these estimated values converge to the correct values of global cuts. Therefore, the amount of duplicates detected in line 9 increases during the above mentioned loop.

In line 10, the cuts of EX are calculated by performing CF tests using binary search. Thus, we can reduce the number of canonical form tests, needed for detecting such cuts. These cuts are called *local cuts* since they are referring to the cuts of a subset of $RE(s)$. Therefore, the cut distribution property (see section 3.1) can be used for estimating bounds for global cuts. Next, local cuts are used for removing all duplicate candidates from EX (see line 11).

In line 12, global cuts are updated from local cuts according to the cut distribution property. The first statement of this property ensures that non-canonical local cuts are lower bounds, according to $\prec_e$, of non-canonical global cuts. Therefore, the estimated values of non-canonical global cuts can be updated by selecting the maximum value between the previously estimated value and the local cut. On the other hand, the second statement of the cut distribution property ensures that canonical local cuts are upper bounds of canonical global cuts. Thus, the estimated values of global cuts can be updated by selecting the minimum value between the previously estimated value and the local cut.

In summary, the correct values of global cuts are surely achieved in the last iteration of the loop at lines 4–14. Nevertheless, correct values could also be achieved, for example, at the first iteration. Therefore, the number of CF tests required for removing all duplicates could be reduced substantially. This duplicate elimination strategy is called by us *full candidate pruning*, and it makes gdFil the first FCS miner that removes all duplicate candidates before support calculation.

3.4 The gdClosed algorithm

In this section, we introduce the gdClose algorithm, which uses the cut properties in combination with the early termination pruning for mining all closed FCSs. The gdClosed algorithm differs from gdFil in the `gdFilMining` procedure, which must be modified for conducting the search space traversal toward the closed patterns. The new mining procedure is called `gdCloseMining` (see Fig. 7).

The cut and cut distribution properties, used in gdFil, are also used in gdClosed (see lines 7–9 and 11 of Fig. 7) for removing all duplicate candidates before early termination pruning and support calculation.

Crossing situations (see section 2.3) are checked in line 15 of Fig. 7. Thus, only closed patterns are considered in the final results (see line 16 of Fig. 7). Every useful extension of s is traversed during the mining, until an equivalent extension without crossing situations appears. In line 21 of Fig. 7, the numbers

$\mathcal{I}(g, D)$ and $\mathcal{L}(g, e, D)$ (see section 2.3) are checked for detecting equivalent extensions. Moreover, the failure of early termination is also checked. Thus, early termination pruning can be applied, and the incoming extensions of s can be pruned.

The calculation of the number of embeddings ($\mathcal{I}(g, D)$ and $\mathcal{L}(g, e, D)$) of each extension (not only rightmost path extensions) of s is required for checking early termination in lines 15 and 21 of Fig. 7. This number can be efficiently calculated by introducing little modifications in the `gdFilExtensions` procedure; the modified procedure is called `gdClosedExtensions` (see Fig. 8).

In line 10 of Fig. 8, we can see that all the extensions of s are considered, according to the early termination pruning. For each extension e_k, its occurrence $\mathcal{I}(g \diamond_x e_k, D)$ is updated, increasing its value when it appears (see line 12 of Fig. 8). Thus, the correct value of $\mathcal{I}(g \diamond_x e_k, D)$ is achieved at the end of the `gdClosedExtensions` procedure. Moreover, all extensions from the same embedding G' are stored in the set EA', which is initialized in line 9 of Fig. 8.

In line 22 of Fig. 8, the set EA' is traversed for calculating the other embedding number $\mathcal{L}(g, e_k, D)$. It is important to notice that this number can be increased only once in each embedding of s.

As we can see, $\mathcal{I}(g, D)$ and $\mathcal{L}(g, e, D)$ can be efficiently calculated inside of the `gdClosedExtensions` procedure. This is one of the main advantages of using early termination pruning together with the DFSE structure for closed FCS mining.

4 Experimental results

For our experiments, we took gSpan and Gaston from the ParMol [20] Java framework which is distributed under GNU license. The gRed and gdFil algorithms were implemented by us, using the same graph managing libraries proposed in ParMol. We chose these libraries and Java as programming language because the ParMol framework contains implementations of gSpan and Gaston that allows us to compare the algorithms regarding their conceptual improvements. That is, the ParMol implementations are not biased by programming styles and compiler optimizations, as it would happen if we use the original implementation of the authors, which would lead to an unfair comparison.

On the other hand, we also compare gdClosed against the reported closed FCS miners, CloseGraph and Moss-MoFa. These algorithms were also implemented and compared using the above mentioned graph managing.

In this experimentation, we use real world and synthetic datasets for algorithm performance evaluation. All the experiments were done using an Intel Core 2 Duo PC at 2.2 GHz with 4 GB of RAM running 64-bit Debian GNU/Linux. The IBM Java Virtual Machine (JVM) Version 6 was used to run the algorithms. The maximum heap memory space of JVM was assigned in 3.2GB. Thus, we remove the influence of swap operations during the execution.

4.1 Tests on real world datasets

The biochemical data collections, specifically the molecular datasets, constitute one of the main application field for graph mining. Therefore, this kind of collections has been commonly used to evaluate the performance of the algorithms for FCS mining. The real world collections used in our experiments are described in Table 1 where P_D is the number of graph in the collection, P_T is the average size of graphs (in terms of the number of edges), P_V is the number of vertex labels, and P_E is the number of edge labels.

The PTE collection is the smallest dataset (according to the number of graphs) used in this work; it contains only 337 graphs representing molecules used in the predictive toxicologic evaluation challenge [19]. In spite of its small size, PTE has a big amount of frequent connected subgraphs; for example, it has 136981 frequent connected subgraphs using the 2% of the collection size as support threshold.

In this work, we also used two medium size collections CAN2DA99[1] and HIV[2]. The CAN2DA99 collection contains the graph representation of 32557 molecules discovered in carcinogenic tumors; whereas the HIV collection contains the graph representation of 42689 molecular structures of the human immunodeficiency virus. The biggest collection used in our experiments was NCI[3], which contains the graph representation of molecules from several sources. All of these graph collections have been commonly used for performance evaluations [20].

In our experiments, we used low support thresholds to evaluate the performance of the algorithms. These thresholds are very important in data mining applications [6, 12]. For example, there are some applications like classification and clustering where frequent complex graph structures are needed [12], and these complex structures only can be found with low support thresholds. Additionally, high thresholds are commonly fulfilled by connected subgraphs with small size regarding the number of vertices, edges, or cycles. For example, Table 2 shows the number and size of frequent connected subgraphs found in the PTE collection using high and low support thresholds. High values of these thresholds only produce a few small patterns unlike low values which produce more patterns with larger size. Moreover, almost all recent algorithms achieve short runtimes for high support thresholds, therefore it is more important to propose fast algorithms for low support thresholds. Finally, it is important to highlight that, like in previous comparative studies [20, 18], in this work the support thresholds are defined as a percentage of the collection size.

[1]http://dtp.nci.nih.gov/docs/cancer/cancer_data.html
[2]http://dtp.nci.nih.gov/docs/aids/aids_data.html
[3]http://cactus.nci.nih.gov/ncidb2/download.html

In this experimentation, we compare the algorithms for finding all FCSs and those for finding all closed FCSs. Table 3 shows the number of FCSs and closed FCSs in the PTE collection. As we can see, the number of FCSs is much greater than the number of closed FCSs. For example, in PTE using the 2% of the collection size as support threshold, we found 136981 FCSs, but only 3741 of them were closed.

The gdFil, gRed, gSpan, and Gaston algorithms (for mining all FCSs) were compared regarding their runtimes on the four molecular collections varying the support threshold (see Fig. 9). Runtime rises for Gaston with the lowest support thresholds and for NCI it was unable to complete the execution for support thresholds smaller than 4% due to high memory requirements. Gaston needed much more memory than the other tested algorithms, since it uses embedding structures for storing useful and duplicate candidates. However, in the smallest collection (PTE), the best results were achieved by Gaston. In CAN2DA99, HIV, and NCI collections, the best runtimes were obtained by gdFil because it does not store any duplicate candidate and it achieves fast isomorphism tests using the DFSE structure. It is known that much of the time consumption in gSpan and gRed is spent in subgraph isomorphism tests during the candidate enumeration process. The gdFil algorithm got the best runtimes since it does a full duplicate candidate pruning, and additionally it achieves fast subgraph isomorphism tests using the DFSE structure.

On the other hand, the gdClosed, CloseGraph, and Moss-MoFa algorithms (for closed FCS mining) were also compared in the molecular collections (see Fig. 10). As we can see, the best runtimes in this experiment were obtained by gdClosed showing that the combination of the full candidate pruning with the early termination pruning allows to gdClosed having a good performance.

4.2 Tests on synthetic datasets

As we can see in the previous section, molecular collections used for performance evaluation have similar attributes (see Table 1). Therefore, performance evaluation using molecular collections does not allow a deep study of the algorithm performance.

Synthetic graph collections are also commonly used for performance evaluations because these collections allow studying the performance of the algorithms, depending on different attributes of the dataset, for example, the number of edge labels or the size of the graphs according to the number of edges.

In our experiments, we use the synthetic graph generator proposed by Kuramochi and Karypis [16], which has been used in several comparative studies for graph mining [21, 23]. This generator allows us to build graph collections varying the parameters: number of graphs in the collection (P_D), average size of graphs (P_T) in terms of the number of edges, number of vertex labels (P_V), number of edge labels (P_E), the number of potentially frequent subgraphs (P_L), and the average size of potentially frequent subgraphs (P_i).

Previously reported comparative studies in graph mining have fixed $P_L = 200$ for building synthetic graph collections [16, 21, 23]. The values for the parameter P_i are commonly taken in the interval $3 \leq P_i \leq 15$. Specifically, values among $3 \leq P_i \leq 6$ have been used for building collections whose frequent subgraphs are small regarding the average number of edges, while values in $10 \leq P_i \leq 15$ have been used for generating collections with big frequent subgraphs. For our experiments, we use $P_L \in \{100, 200, 400, 800\}$ and $P_i \in \{6, 8, 10, 12, 14\}$ for building synthetic graph collections.

The other parameters, were chosen around the mean value of the tested real world collections. Table 4 shows all the tested values for these parameters around their mean values, which appear underlined in each case. In each test, only one parameter is varied, and the other parameters are fixed to the mean values.

The gdFil, gRed, gSpan, and Gaston algorithms (for mining all FCSs) were compared in synthetic graph collections, using the aforementioned setup (see Fig. 11). As we can see, the best runtimes were achieved by gdFil in almost all the cases. However, there are some cases that should be explained. For small values of P_V and P_E, for example $P_V = 10$ or $P_E = 1$, Gaston had better runtimes than gdFil. The last happened because the usefulness of the cut and cut distribution properties in gdFil is very limited when the number of labels in vertices or edges is small. In these cases, the runtime of gdFil tends to be similar to the runtime of gSpan.

On the other hand, a comparison among the gdClosed, CloseGraph, and Moss-MoFa algorithms (for closed FCS mining) were presented in Table 12. This comparison was made using the same parameters setup showed in Table 4. The best runtimes were obtained by gdClosed in almost all the tests, achieving the best results when P_V and P_E are greather than 40 and 4 respectively.

In summary, we can conclude that small values of P_V and P_E significatively affect the performance of gdFil and gdClosed. However, both algorithms are a good option for processing collections with $P_V \geq 40$ and $P_E \geq 4$. For the other parameters, our algorithms have in general a good performance.

5 Conclusions

In this paper, two novel properties of the DFS code, which allows us to define boundaries between duplicate and useful candidates, were introduced and proved. These properties are useful to remove all duplicate candidates, before support calculation. Besides, a new kind of embedding structure (DFSE), which allows us to reduce the cost of subgraph isomorphism tests, was also introduced.

Based on the proposed properties and the new embedding structure, two new FCS mining algorithms were introduced. The first one (gdFil) for finding the whole set of frequent connected subgraphs, whereas the other one (gdClosed) used the full candidate pruning in combination with the early termination pruning for closed FCS mining. The pruning of all duplicate candidates (full candidate pruning) using the cut and cut distribution properties during candidate enumeration allows reducing the cost of subgraph isomorphism tests without storing duplicate candidates in the DFSE structure.

The experimental results over real world and synthetic datasets show that the full candidate pruning allows to our proposed algorithms getting the better performance than other well known algorithms. Specially, the improvement achieved for our proposals can be appreciated for small support thresholds over big datasets with many vertex and edge labels.

As future work, we are going to develop new ways for taking advantage of the cut properties and the DFSE structure for finding other frequent graph based patterns like maximal and approximate.

References

[1] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules. In *Proceedings of the 1994 International Conference on Very Large Data Bases (VLDB'94)*, pages 487–499, Santiago, Chile, 1994.

[2] C. Borgelt. Canonical Forms for Frequent Graph Mining. In *Proceedings of the 30th Annual Conference of the Gesellschaft für Klassifikation e.V.*, pages 8–10, Freie Universitat Berlin, 2006.

[3] C. Borgelt and M.R. Berthold. Mining Molecular Fragments: Finding Relevant Substructures of Molecules. In *Proceedings of the 2002 International*

Conference on Data Mining (ICDM'02), pages 211–218, Maebashi, Japan, 2002.

[4] C. Borgelt, T. Meinl, and M.R. Berthold. Advanced Pruning Strategies to Speed Up Mining Closed Molecular Fragments. *In Proceedings of the 2004 IEEE International Conference on Systems, Man and Cybernetics*, 5:4535–4570, 2004.

[5] L. Dehaspe, H. Toivonen, and R. King. Finding frequent substructures in chemical compounds. In *Proceedings of the 1998 International Conference on Knowledge Discovery and Data Mining (KDD'98)*, pages 30–36, New York, NY, 1998.

[6] W. Fan, K. Zhang, H. Cheng, J. Gao, X. Yan, J. Han, P. Yu, and O. Verscheure. Direct Mining of Discriminative and Essential Frequent Patterns via Model-based Search Tree. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 230–238, Nevada, USA, 2008.

[7] A. Gago-Alonso, J. A. Carrasco-Ochoa, J. E. Medina-Pagola, and J. F. Martínez-Trinidad. Duplicate Candidate Elimination and Fast Support Calculation for Frequent Subgraph Mining. In *Proceedings of 10th International Conference on Intelligent Data Engineering and Automated Learning (IDEAL'09)*, pages 292–299, Burgos, Spain, 2009.

[8] A. Gago-Alonso, J. E. Medina-Pagola, J. A. Carrasco-Ochoa, and J. F. Martínez-Trinidad. Mining Frequent Connected Subgrahps Reducing the Number of Candidates. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD'08)*, pages 365–376, Antwerp, Belgium, 2008.

[9] R. Geng, W. Xu, and X. Dong. Efficient Mining of Interesting Weighted Patterns from Directed Graph Traversals. *Integrated Computer-Aided Engineering*, 16(1):21–49, 2009.

[10] J. Han, H. Cheng, D. Xin, and X. Yan. Frequent Pattern Mining: Current Status and Future Directions. *Data Mining and Knowledge Discovery, 10th Anniversary Issue*, 15(1):55–86, 2007.

[11] L.B. Holder, D.J. Cook, and S. Djoko. Substructure discovery in the subdue system. In *Proceedings of the AAAI Workshop on Knowledge Discovery in Databases (KDD'94)*, pages 169–180, Seattle, WA, 1994.

[12] M.S. Hossain and R.A. Angryk. GDClust: A Graph-based Document Clustering Technique. In *Proceedings of the 7th IEEE International Conference on Data Mining Workshops*, pages 417–422, Omaha, NE, 2007.

[13] J. Huan, W. Wang, and J. Prins. Efficient Mining of Frequent Subgraph in the Presence of Isomorphism. In *Proceedings of the 2003 International Conference on Data Mining (ICDM'03)*, pages 549–552, Melbourne, FL, 2003.

[14] A. Inokuchi, T. Washio, Nishimura K., and H. Motoda. A Fast Algorithm for Mining Frequent Connected Subgraphs. Technical Report RT0448, IBM Research, Tokyo Research Laboratory, 2002.

[15] A. Inokuchi, T. Washio, and H. Motoda. An Apriori based Algorithm for Mining Frequent Substructures from Graph Data. In *Proceedings of the 2000 European Symposium on the Principle of Data Mining and Knowledge Discovery (PKDD'00)*, pages 13–23, Lyon, France, 2000.

[16] M. Kuramochi and G. Karypis. Frequent Subgraph Discovery. In *Proceedings of the 2001 International Conference on Data Mining (ICDM'01)*, pages 313–320, San Jose, CA, 2001.

[17] S. Nijssen and J. Kok. A Quickstart in Frequent Structure Mining can Make a Difference. In *Proceedings of the 2004 ACM SIGKDD International Conference on Kowledge Discovery in Databases (KDD'04)*, pages 647–352, Seattle, WA, 2004.

[18] S. Nijssen and J. Kok. Frequent Subgraph Miners: Runtimes Don't Say Everything. In *Proceedings Mining and Learning with Graphs (MLG'06), workshop held with ECML-PKDD'06*, pages 173–180, Berlin, Germany, 2006.

[19] A. Srinivasan, R.D. King, S.H. Muggleton, and M. Sternberg. The Predictive Toxicologic Evaluation Challenge. In *Proceedings of the 15th International Conference on Artificial Intelligence (IJCAI'97), Morgan-Kaufmann*, pages 1–6, 1997.

[20] M. Wörlein, T. Meinl, I. Fischer, and M. Philippsen. A Quantitative Comparison of the Subgraph Miners Mofa, gSpan, FFSM, and Gaston. In *Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'05)*, pages 392–403, Porto, Portugal, 2005.

[21] X. Yan and J. Han. gSpan: Graph-Based Substructure Pattern Mining. In *Proceedings of the 2002 International Conference on Data Mining (ICDM'02)*, pages 721–724, Maebashi, Japan, 2002.

[22] X. Yan and J. Han. gSpan: Graph-based Substructure Pattern Mining. *UIUC Technical Report, UIUCDCS-R-2002-2296*, 2002.

[23] X. Yan and J. Han. CloseGraph: Mining Closed Frequent Graph Patterns. *In Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC*, pages 286–295, 2003.

Table 1: Real world datasets used in the experiments.

	P_D	P_T	P_V	P_E
PTE	340	27	66	4
CAN2DA99	32 557	28	69	4
HIV	42 689	28	63	3
NCI	237 771	22	78	4

Table 2: Number and size of graphs found for high and low thresholds in the PTE collection.

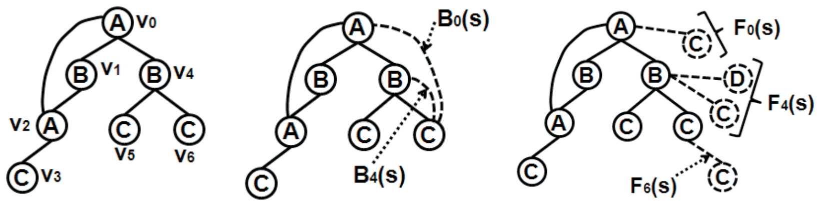
High support thresholds			Low support thresholds		
threshold	Number of FCSs	Size of the bigest FCS	threshold	Number of FCSs	Size of bigest FCS
30%	75	9	3%	18 146	22
40%	62	9	4%	5 955	15
50%	37	7	5%	3 627	14

Table 3: Number of patterns mined by gdFil and gdClosed in the PTE collection varying support thresholds.

Support threshold	gdFil	gdClosed
2%	136 981	3 741
3%	18 146	1 928
4%	5 955	1 284
5%	3 627	991
6%	2 138	739
8%	1 786	634

Table 4: Parameters settings for experimentation in synthetic graph collections.

Parameter	Values
P_D	$\{500, 5K, \underline{50K}, 500K\}$
P_T	$\{5, 15, \underline{25}, 35, 45\}$
P_V	$\{10, 40, \underline{70}, 100, 130\}$
P_E	$\{1, 2, \underline{4}, 6, 8\}$
P_L	$\{100, \underline{200}, 400, 800\}$
P_i	$\{6,8,\underline{10},12,14\}$



(a) A DFS tree. (b) Backward extensions. (c) Forward extensions.

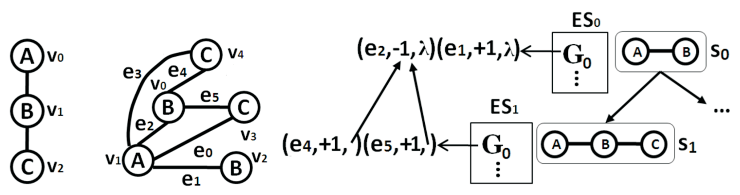
Fig. 1: Example of partitions in the rightmost path extensions.

Procedure `gdFilExtensions`(s, G, L, EX)

Input: s - DFS code, G - a graph of the collection D,
L - the embedding list $L(s, G)$
Output: EX - the extension set of s in G

```
foreach embedding tuple $\tau \in L$ do
    Mapping $\leftarrow \{(\tau.k, \tau.\epsilon)\}$;
    $\tau_c \leftarrow \tau$;
    while $\tau_c.p \neq \lambda$ do
        $\tau_c \leftarrow \tau_c.p$;
        Mapping $\leftarrow$ Mapping $\cup \{(\tau_c.k, \tau_c.\epsilon)\}$;
    end
    Let $G' \subseteq G$ be the subgraph which is contained in Mapping being
    isomorphic to $s$;
    forall edge $e_k \in E(G) \setminus E(G')$ such that $e_k$ is rightmost extension of $s$
    do
        if $e_k$ is compatible with its computational representation in $G$
        then $\epsilon \rightarrow +1$;
        else $\epsilon \rightarrow -1$;
        $L(s \diamond e_k, G) \leftarrow L(s \diamond e_k, G) \cup \{(e_k, \epsilon, \tau)\}$;
        $EX \leftarrow EX \cup \{e_k\}$;
    end
end
```

Fig. 2: Pseudo-code of the `gdFilExtensions` procedure.



(a) s_1. (b) G_0. (c) Embeddings of s_1 and all of its ancestors.

Fig. 3: Example of the DFSE structure of s_1 in a collection containing G_0.

```
Procedure gdFil($D$, $\delta$, $S$)
Input: $D$ - graph collection, $\delta$ - support threshold
Output: $S$ - mining results
1  Remove infrequent vertices and edges;
2  $S \leftarrow$ all frequent vertices;
3  $S^1 \leftarrow$ all frequent 1-edge codes;
4  forall code $s \in S^1$ do
5      Initialize the DFSE structure $ES(s)$;
6      gdFilMining($D$,$s$,$\delta$,$S$);
7      $D \leftarrow D \setminus s$;
8      if $|D| < \delta$ then break;
9  end
```

Fig. 4: Pseudo-code of the main procedure of gdFil.

```
Procedure gdFilMining($D$, $s$, $\delta$, $S$)
Input: $D$ - graph collection, $s$ - a minimum DFS code, $\delta$ - support
       threshold
Output: $S$ - mining results
1  $S \leftarrow S \cup \{s\}$;
2  $ME \leftarrow \emptyset$;
3  Initialize global cuts in $\epsilon$, indicating the non-existence of cuts for
   $ME = \emptyset$;
4  forall pair $(G, L) \in ES(s)$ do
5      $EX \leftarrow \emptyset$;
6      gdFilExtensions($s, G, L, EX$);
7      Remove from $EX$ those extensions that are filtered by gSpan basic
       optimizations;
8      Remove from $EX$ those extensions that are filtered by
       non-minimality conditions;
9      Remove from $EX$ those extensions that are filtered by current global
       cuts;
10     Calculate local cuts of $EX$ performing canonical form tests;
11     Remove from $EX$ the extensions in the duplicate partitions of $EX$;
12     Update global cuts from local cuts;
13     $ME \leftarrow ME \cup EX$;
14 end
15 Remove from $ME$ non-frequent extensions, according to $\delta$;
16 forall extension $e \in ME$ do gdFilMining($D$, $s \diamond e$, $\delta$, $S$);
```

Fig. 5: Pseudo-code of the search space traversal in gdFil.

```
Procedure gdClosed($D$, $\delta$, $S$)
Input: $D$ - graph collection, $\delta$ - support threshold
Output: $S$ - mining results
1 Remove infrequent vertices and edges;
2 $S \leftarrow$ all frequent vertices;
3 $S^1 \leftarrow$ all frequent 1-edge codes;
4 forall code $s \in S^1$ do
5     Initialize the DFSE structure $ES(s)$;
6     gdClosedMining($D$,$s$,$\delta$,$S$);
7     $D \leftarrow D \setminus s$;
8     if $|D| < \delta$ then break;
9 end
```

Fig. 6: Pseudo-code of the main procedure of gdClosed.

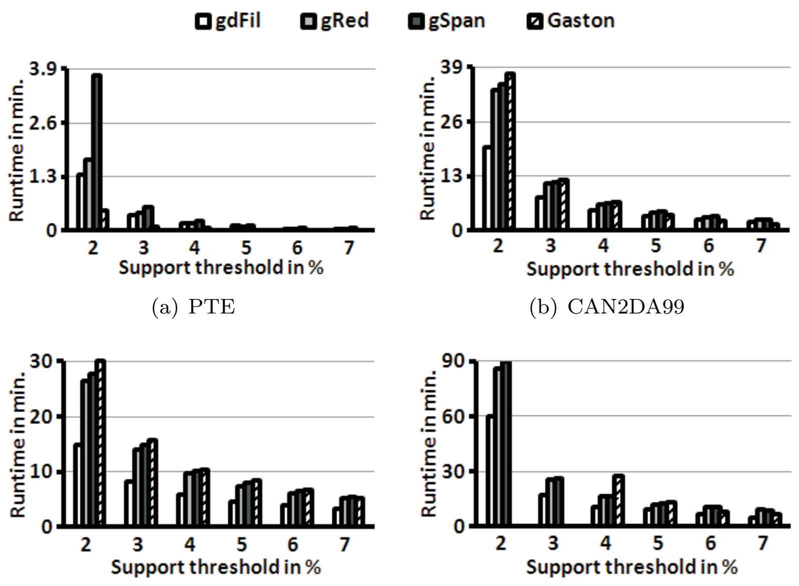
```
Procedure gdClosedMining($D$, $s$, $\delta$, $S$)
Input: $D$ - graph collection, $s$ - a minimum DFS code, $\delta$ - support
       threshold
Output: $S$ - mining results
1  $S \leftarrow S \cup \{s\}$;
2  $ME \leftarrow \emptyset$;
3  Initialize global cuts in $\epsilon$, indicating the non-existence of cuts for
   $ME = \emptyset$;
4  forall pair $(G, L) \in ES(s)$ do
5      $EX \leftarrow \emptyset$;
6      gdClosedExtensions($s, G, L, EX$);
7      Remove from $EX$ those extensions that are filtered by gSpan basic
       optimizations;
8      Remove from $EX$ those extensions that are filtered by
       non-minimality conditions;
9      Remove from $EX$ those extensions that are filtered by current global
       cuts;
10     Calculate local cuts of $EX$ performing canonical form tests;
11     Remove from $EX$ the extensions in the duplicate partitions of $EX$;
12     Update global cuts from local cuts;
13     $ME \leftarrow ME \cup EX$;
14 end
15 Detect the extensions of $s$ that have crossing situations in $s$;
16 if $s$ is closed then $S \leftarrow S \cup \{s\}$;
17 Remove from $ME$ non-frequent extensions, according to $\delta$;
18 forall extension $e \in ME$ do
19     gdClosedMining($D$, $s \diamond e$, $\delta$, $S$);
20     Let $g$ be the graph represented by the DFS code $s$;
21     if $\mathcal{L}(g, e, D) = \mathcal{I}(g, D)$ and early termination does not fail in $e$ then
       return;
22 end
```

Fig. 7: Pseudo-code of the search space traversal in gdClosed.

Procedure `gdClosedExtensions`(s, G, L, EX, EA)

```
Input: $s$ - DFS code, $G$ - a graph of the collection $D$,
$L$ - the embedding list $L(s, G)$
Output: $EX$ - the extension set of $s$ in $G$
foreach embedding tuple $\tau \in L$ do
    Mapping $\leftarrow \{(\tau.k, \tau.\epsilon)\}$;
    $\tau_c \leftarrow \tau$;
    while $\tau_c.p \neq \lambda$ do
        $\tau_c \leftarrow \tau_c.p$;
        Mapping $\leftarrow$ Mapping $\cup \{(\tau_c.k, \tau_c.\epsilon)\}$;
    end
    Let $G' \subseteq G$ be the subgraph which is contained in Mapping being
    isomorphic to $s$;
    $EA' \leftarrow \emptyset$;
    forall edge $e_k \in E(G) \setminus E(G')$ such that $e_k$ is an extension of $s$ do
        Let $g$ be the graph represented by the DFS code $s$;
        $\mathcal{I}(g \diamond_x e_k, D) \leftarrow \mathcal{I}(g \diamond_x e_k, D) + 1$;
        $EA' \leftarrow EA' \cup \{e_k\}$;
        if $e_k$ is rightmost extension of $s$ then
            if $e_k$ is compatible with its computational representation in $G$
            then $\epsilon \rightarrow +1$;
            else $\epsilon \rightarrow -1$;
            $L(s \diamond e_k, G) \leftarrow L(s \diamond e_k, G) \cup \{(e_k, \epsilon, \tau)\}$;
            $EX \leftarrow EX \cup \{e_k\}$;
        end
    end
    $EA \leftarrow EA \cup EA'$;
    forall $e_k \in EA'$ do $\mathcal{L}(g, e_k, D) \leftarrow \mathcal{L}(g, e_k, D) + 1$;
end
```

Fig. 8: Pseudo-code of the `gdClosedExtensions` procedure.



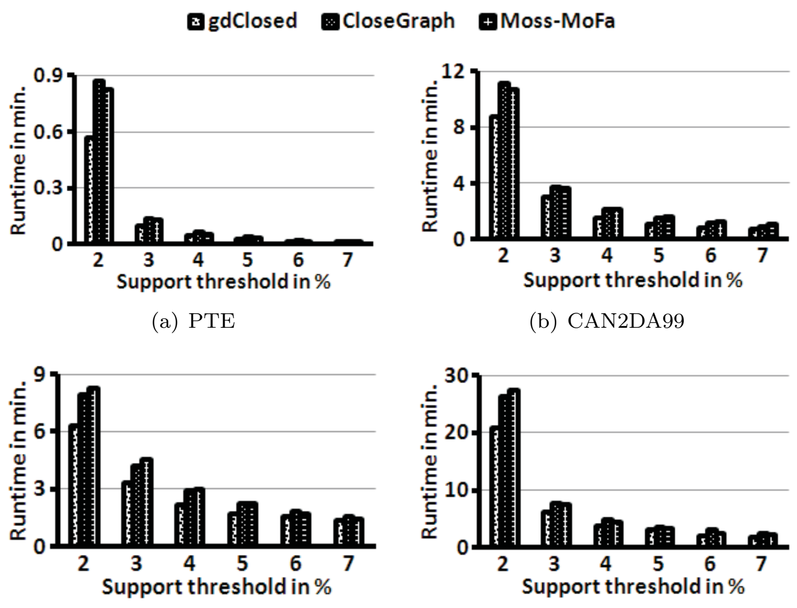
(a) PTE

(b) CAN2DA99

(c) HIV

(d) NCI

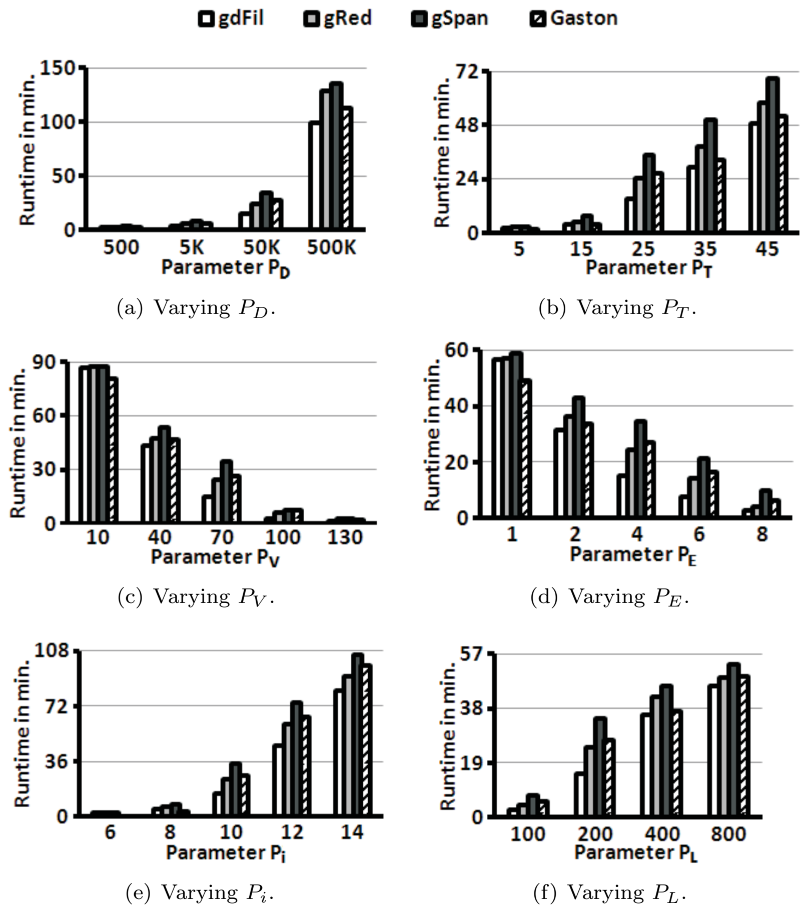
Fig. 9: Performance evaluation of gdFil, gRed, gSpan, and Gaston, in real world datasets.



(a) PTE (b) CAN2DA99

(c) HIV (d) NCI

Fig. 10: Performance evaluation of gdClosed, CloseGraph, and Moss-MoFa, in real world datasets.



(a) Varying P_D.

(b) Varying P_T.

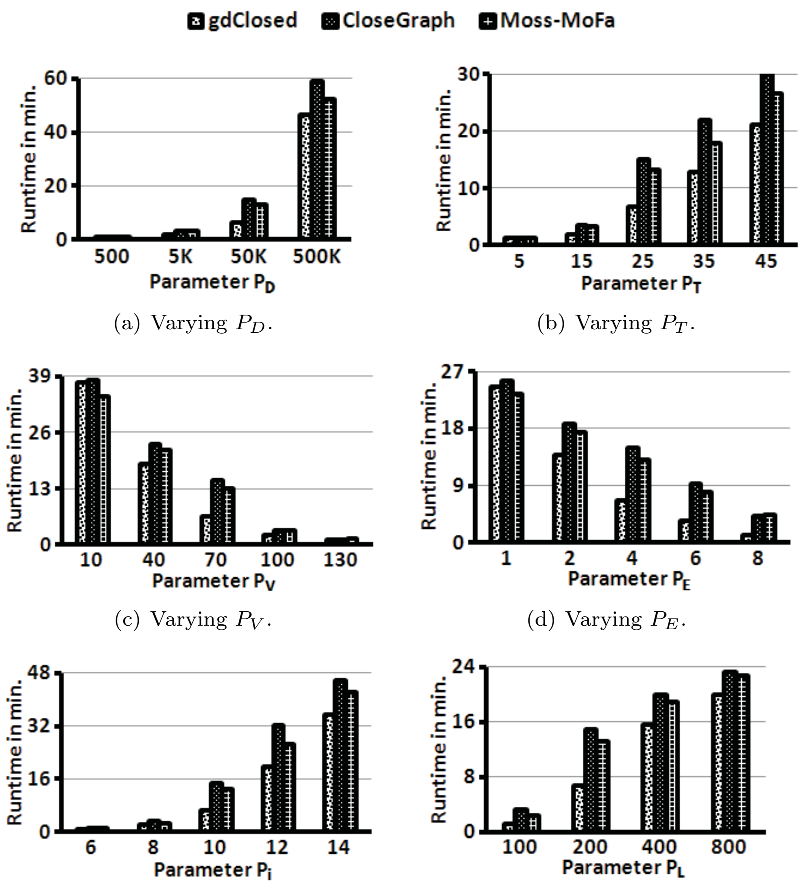
(c) Varying P_V.

(d) Varying P_E.

(e) Varying P_i.

(f) Varying P_L.

Fig. 11: Performance evaluation of gdFil, gRed, gSpan, and Gaston, in synthetic datasets.



(a) Varying P_D.

(b) Varying P_T.

(c) Varying P_V.

(d) Varying P_E.

(e) Varying P_i.

(f) Varying P_L.

Fig. 12: Performance evaluation of gdClosed, CloseGraph, and Moss-MoFa, in synthetic datasets.