RNPS No. 2143
ISSN 2072-6260
Versión Digital

SERIE GRIS

REPORTE TÉCNICO
Minería de Datos

Minería de subgrafos frecuentes utilizando cotejo inexacto de grafos

Niusvel Acosta Mendoza,
Andrés Gago Alonso y
José E. Medina Pagola

RT_017 **junio 2011**

CENATAV
Centro de Aplicaciones de Tecnologías de Avanzada
MINISTERIO DE LA INDUSTRIA BÁSICA

7ma. No. 21812 e/218 y 222,
Rpto. Siboney, Playa;
La Habana.
Cuba. C.P. 12200
www.cenatav.co.cu

RNPS No. 2143
ISSN 2072-6260
Versión Digital

CENATAV
Centro de Aplicaciones de Tecnologías de Avanzada
MINISTERIO DE LA INDUSTRIA BÁSICA

REPORTE TÉCNICO
Minería de Datos

SERIE GRIS

Minería de subgrafos frecuentes utilizando cotejo inexacto de grafos

Niusvel Acosta Mendoza,
Andrés Gago Alonso y
José E. Medina Pagola

RT_017 **junio 2011**

7ma. No. 21812 e/218 y 222,
Rpto. Siboney, Playa;
La Habana.
Cuba. C.P. 12200
www.cenatav.co.cu

Tabla de contenido

Lista de Algoritmos

Minería de subgrafos frecuentes utilizando cotejo inexacto de grafos

Estado del Arte

Niusvel Acosta-Mendoza, Andrés Gago-Alonso, y José E. Medina-Pagola

Dpto. Minería de Datos, Centro de Aplicaciones de Tecnologías de Avanzada (CENATAV),
La Habana, Cuba
{nacosta,agago,jmedina}@cenatav.co.cu

RT_017, Serie Gris, CENATAV
Aceptado: 5 de abril de 2011

Resumen. Hoy día es posible apreciar un lento avance en el desarrollo de algoritmos para la minería de subgrafos frecuentes (SF) con cotejo inexacto de grafos. La mayoría de los algoritmos reportados utilizan cotejo exacto para minar SF. Por otro lado, la necesidad del cotejo inexacto en este tipo de minería ha sido planteada en diversas aplicaciones por algunos autores.
Al analizar algunos de los algoritmos de minería de SF que utilizan cotejo inexacto, se pudo apreciar que presentan problemas de eficiencia computacional. De este modo, el procesamiento de colecciones de gran tamaño sigue siendo un problema vigente.
En esta investigación se propone abordar el problema de la minería de SF utilizando cotejo inexacto sobre colecciones de grafos.

Palabras clave: minería de grafos, cotejo inexacto, subgrafos frecuentes, subgrafos aproximados frecuentes, minería de subgrafos frecuentes.

Abstract. Today it is possible to appreciate a slow progress in the development of algorithms for mining frequent subgraphs with inexact graph matching. In one hand, most of the reported algorithms for mining frequent subgraphs use exact matching. On the other hand, the necessity of inexact matching in this kind of mining in many other application has been stated by some other authors.
When analyzing some of the frequent subgraph mining algorithms that use inexact matching, it was observed that they present problems of computational efficiency. In this way, the processing of high size collection is still a present problem.
In this investigation we propose to deal with the problem of frequent subgraphs mining using inexact matching about graphs collections.

Keywords: graph mining, inexact matching, frequent subgraphs, approximate frequent subgraphs, frequent subgraph mining.

1 Introducción

En los últimos años se ha detectado la necesidad de convertir grandes volúmenes de datos en información útil. Como consecuencia se han ido desarrollando técnicas y métodos que permiten procesar este cúmulo de datos o repositorios. Ejemplo de tales técnicas lo constituye el descubrimiento de patrones frecuentes [1].

Muchos repositorios de datos están o pueden ser representados mediante colecciones de grafos (enfoque transaccional de la minería) [2], las cuales se pueden clasificar, entre otros aspectos, por las estructuras reales que representan o los tipos de grafos que contienen. Algunas de estas colecciones han sido utilizadas para el análisis de estructuras bioquímicas como: las redes PPI (de sus siglas en inglés, *Protein-Protein Interactions networks*) que varían su tamaño según los organismos que representen, los cuales son modelados por grafos no dirigidos con pesos en las aristas en algunos casos [3] y en otros casos se le agregan etiquetas únicas en los vértices [4]; KEGG (de sus siglas en inglés, *Kyoto Encyclopedia of Genes and Genomes*) [5] está conformada por grafos no dirigidos y etiquetados; Saccharomyces Cerevisiae y Drosophila Melanogaster compuestas por grafos no dirigidos y etiquetados sólo en los vértices [6]. Por otro lado están las bases de datos utilizadas para el análisis de vínculos y redes sociales como: IMDb (de sus siglas en inglés, *Internet Movie Database*) [7] conformada por grafos etiquetados representando interacciones entre artistas y películas; En [8] se utilizan las colecciones: NYSE (de sus siglas en inglés, *New York Stock Exchange*), e-mails e interacciones sociales entre cebras. Estas tres últimas representan redes temporales conformadas por grafos dirigidos sin etiquetas en las aristas y con etiquetas únicas en los vértices.

El descubrimiento de patrones frecuentes, en especial la detección de subgrafos frecuentes (SF) en colecciones de grafos, es un importante problema en la minería de grafos. Esto se debe al crecimiento de las soluciones a muchos de los problemas existentes en los diferentes dominios de la ciencia [9],[8],[10],[5]. Por este motivo, varios autores han trabajado en el desarrollo de algoritmos eficientes para el descubrimiento de SF [11],[2],[12].

Los trabajos reportados en la literatura para la minería SF muestran el avance logrado en esta temática. Sin embargo, aún queda mucho por hacer para lograr mejores desempeños computacionales en tiempo y espacio. Esto se debe a la existencia de algoritmos con gran complejidad computacional [6],[13] y otros que necesitan de mayor capacidad de almacenamiento para mantener sus estructuras internas [14].

Evaluar la similitud estructural de los grafos es una de las tareas más costosa en tiempo en la minería de SF, conocida como cotejo de grafo (en inglés, *graph matching*) [15]. Algunas de las formas de cotejo de grafos son NP-Completo (ejemplo, homeomorfismo [16]), otras son NP-Duro (ejemplo, sub-isomorfismo [2]) y otras como el caso del isomorfismo no se ha podido determinar si es P o NP [15]. Existen dos formas de abordar el cotejo de grafos: cotejo exacto y cotejo inexacto.

El cotejo exacto consiste en determinar si las etiquetas y la estructura de dos grafos son idénticas y ha sido utilizada con éxito en muchas aplicaciones [2],[7],[17],[18]. Sin embargo, existen aplicaiones donde esta forma exacta de describir las correspondencias no es aplicable con éxito [13]. Es por eso que se ha hecho necesario evaluar la similitud entre grafos admitiendo algunas diferencias estructurales, o sea, mediante técnicas de cotejo inexacto. El cotejo inexacto consiste en encontrar la mejor adecuación entre vértices o aristas de dos grafos para determinar su similitud admitiendo diferencias entre estructuras y etiquetas. Sobre esta base, se ha planteado la necesidad de realizar la minería de SF utilizando cotejo inexacto de grafos [7],[19],[20],[21].

Varios autores defienden la idea de que se podrían encontrar SF de mayor interés para las aplicaciones y los usuarios. Hossain y Angryk recomendaron la detección de los SF utilizando cotejo inexacto en tareas de agrupamiento de documentos [19]. Este tipo de patrones similares frecuentes también ha sido recomendado en tareas del procesamiento de datos químicos [21], en tareas de análisis de vínculos [7] y se ha presentado como un problema abierto en tareas de procesamiento de datos moleculares y redes sociales [20]. En respuesta a esta necesidad, en los últimos años el cotejo inexacto ha sido exitosamente aplicados en diferentes dominios de la ciencia como: análisis de estructuras bioquímicas [3],[5],[6],[14], análisis de imágenes y escenas, análisis de circuitos [9], análisis de vínculos y redes sociales [8],[22],[10],[23],[24].

En trabajos como [25], se han definido las siguientes características principales de las redes sociales: (1) Redes incompletas y (2) Límites borrosos, ya sea por la manipulación de la información obtenida o por la falta de la misma, y (3) Redes dinámicas, dado a la pérdida de sus miembros o la adición o modificación de tipos de relaciones. Estas características limitan el uso de cotejo exacto y favorecen al uso más idóneo del cotejo inexacto.

El siguiente ejemplo muestra las diferencias entre el uso de los dos tipos de cotejo. Se tiene la colección de grafos D de la figura Fig. 1, compuesta por tres grafos y se define como umbral de mínima frecuencia $\delta = 2$.

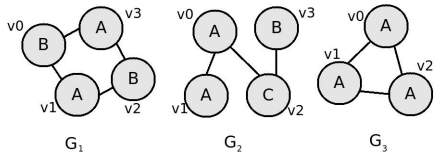

Fig. 1. Colección de grafos D.

Cuando se realiza la minería de SF sobre la colección D utilizando cotejo exacto, solamente se obtienen los SF mostrados en la Fig. 2. Puede apreciarse que los patrones encontrados son pequeños.

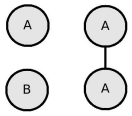

Fig. 2. Subgrafos frecuentes en la colección D utilizando cotejo exacto y $\delta = 2$.

Por otro lado, al utilizar técnicas de cotejo inexacto para realizar la minería en la colección D, es posible obtener otros subgrafos que pueden ser considerados frecuentes como, por ejemplo, los mostrados en la Fig. 3.

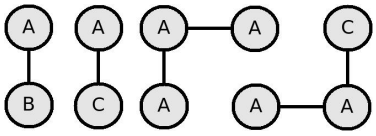

Fig. 3. Subgrafos de la colección D que pueden ser considerados frecuentes al utilizar cotejo inexacto.

La función de cotejo utilizada en el ejemplo anterior calcula la similitud entre grafos que presenten diferencias estructurales y entre etiquetas. Por lo que, en esta figura se muestran otros subgrafos de mayor tamaño identificados sobre la colección D, que no se encontrarían si se utilizara cotejo exacto.

Por las razones anteriormente expuestas mediante este trabajo se presentará un estado del arte sobre los algoritmos reportados para la minería de SF utilizando cotejo inexacto de grafos.

2 Marco teórico

En esta sección se presentan los conceptos básicos necesarios para definir el problema de la minería de SF y para entender el resto del documento. Se describen algunas propiedades presentes en algoritmos del estado del arte, así como la forma canónica utilizada para representar los grafos, y finalmente se dan las características fundamentales de las estructuras de datos utilizadas para indizar la colección de grafos durante la minería.

2.1 Conceptos preliminares

La mayoría de los algoritmos de minería de grafos reportados en la literatura están diseñados para procesar colecciones de grafos simples[1] y no dirigidos, ya que existen numerosas aplicaciones que pueden ser modeladas con este tipo de grafos. En gran parte de estas aplicaciones, la minería de grafos se ha realizado utilizando técnicas de cotejo exacto [2]. No obstante, dada la necesidad planteada por varios autores de realizar la minería de SF utilizando cotejo inexacto de grafos, se han desarrollado trabajos con el objetivo de resolver algunos de estos problemas mediante este tipo de técnicas de cotejo de grafos [7],[19],[20],[21].

De manera similar se utilizan los grafos simples y no dirigidos en este trabajo, por lo que en lo adelante cuando se hable de grafo se suponen todas estas características y en otro caso se especificará explícitamente. En la literatura se pueden encontrar varias definiciones de este tipo de grafo. Sin embargo, se considera que la definición dada a continuación es suficientemente flexible para una gran variedad de aplicaciones.

Definición 1 (Grafo etiquetado). *Dados los alfabetos de etiquetas de vértices y aristas L_V y L_E, respectivamente, un grafo etiquetado y no dirigido se define por $G = (V, E, L, l)$, donde:*

- *V es un conjunto finito de vértices,*
- *$E \subset \{(u,v) | u,v \in V, u \neq v\}$ es el conjunto de aristas (se suele decir que la arista (u,v) conecta a los vértices u y v),*
- *L es el alfabeto de etiquetas y*
- *$l : V \cup E \rightarrow L$ es la función que asigna etiquetas a los vértices y aristas del grafo.*

El alfabeto de etiquetas es un conjunto discreto de símbolos $L = \{s_1, s_2, \ldots, s_k\}$. El *grafo vacío* se define por $G = (\emptyset, \emptyset, \emptyset, l) = \emptyset$.

Con el objetivo de enmarcar el dominio de todos los posibles grafos etiquetados se denotará este conjunto por el símbolo Ω, donde $\emptyset \in \Omega$.

Dos vértices $u, v \in V$ son *vecinos* o *adyacentes* si $(u,v) \in E$. Dos aristas $(u_1, u_2) \neq (v_1, v_2)$, tal que $(u_1, u_2), (v_1, v_2) \in E$, son adyacentes si tienen uno de sus vértices en común. De esta forma, se define $N(u) = \{v_0, v_1, \ldots, v_{n-1}\}$ como el conjunto de vértices vecinos a $u \in V$, tal que $N(u) \subset V$. De manera similar para las aristas.

Un *grafo es completo* si con n vértices contiene C_2^n aristas; es decir, cada uno de sus vértices es adyacente a cada uno de los vértices restantes.

Definición 2 (Subgrafo y supergrafo). *Sean $G_1, G_2 \in \Omega$, $G_1 = (V_1, E_1, L_1, l_1)$ y $G_2 = (V_2, E_2, L_2, l_2)$, que comparten el mismo conjunto de etiquetas $L_1 = L_2$ y la misma función etiquetadora $l_1 = l_2$. Se dice que G_1 es subgrafo de G_2 si $V_1 \subseteq V_2, E_1 \subseteq E_2$. En este caso se utiliza la notación $G_1 \subseteq G_2$ y se dice que G_2 es un supergrafo de G_1.*

Definición 3 (Extensión de un grafo). *Sean $G_1, G_2 \in \Omega$, $G_1 = (V_1, E_1, L_1, l_1)$ y $G_2 = (V_2, E_2, L_2, l_2)$. Denotemos u y v dos vértices tales que $u, v \in V_2$, $u \in V_1$ y $(u,v) \notin E_1$. Supongamos que:*

- *$V_2 = V_1 \cup \{v\}$,*
- *$E_2 = E_1 \cup \{(u,v)\}$,*
- *$L_2 = L_1 \cup \{l_2(v), l_2((u,v))\}$,*
- *$l_2 = V_2 \cup E_2 \rightarrow L_2$.*

En tal caso, se dice que (u,v) es una extensión de G_1. G_2 es un hijo de G_1, y G_1 es el padre de G_2. Además, se puede usar la notación $G_2 = G_1 \diamond (u,v)$.

[1] Un grafo es simple cuando no presenta lazos ni dobles aristas entre vértices

Las extensiones (u,v) de G_1 donde $v \in V_1$ suelen ser llamadas extenciones cerradas. Mientras que en el otro caso, $v \notin V_1$ se dice que (u,v) es una extensión por vértice. En lo sucesivo, cuando se hable de extensión cerrada se utilizará la expresión $G_1 \diamond_c (u,v)$ y cuando se hable de extensión por vértice se utilizará $G_1 \diamond_v (u,v)$. La figura 4 ilustra un ejemplo de cada tipo de extensión.

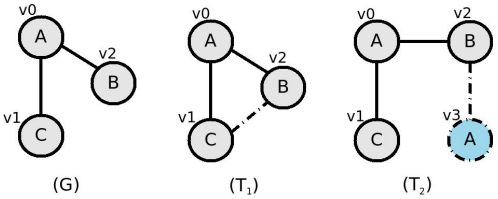

Fig. 4. Ejemplo de extensión donde $T_1 = G \diamond_c (v_1, v_2)$ y $T_2 = G \diamond_v (v_2, v_3)$.

La gran mayoría de los algoritmos para la minería de SF reportados en la literatura que utilizan técnicas de cotejo inexacto están basados en crecimiento de patrones. Estos algoritmos obtienen los candidatos mediante las extensiones de los SF obtenidos en el proceso de minado.

2.1.1 Cotejo exacto de grafos

Como se explicó anteriormente, el cotejo exacto de grafos consiste en determinar si dos grafos son idénticos en término de estructura y etiquetas. El concepto de la igualdad entre grafos se conoce como isomorfismo (ver definición 4).

Definición 4 (Isomorfismo). *Se dice que f es un isomorfismo entre $G_1, G_2 \in \Omega$, $G_1 = (V_1, E_1, L_1, l_1)$ y $G_2 = (V_2, E_2, L_2, l_2)$, si $f : V_1 \to V_2$ es una función biyectiva tal que*

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

En otras palabras, dos grafos son isomorfos si existe una correspondencia biunívoca entre V_1 y V_2 que mantiene las aristas y todas las etiquetas. En la figura 5 se ilustra un ejemplo de isomorfismo.

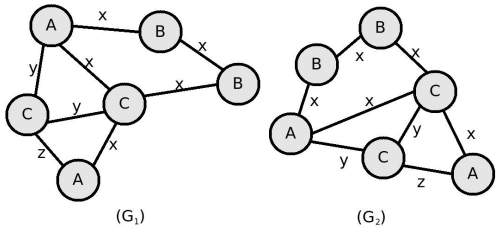

Fig. 5. Ejemplo de isomorfismo donde el grafo G_1 es isomorfo con el G_2.

Definición 5 (Sub-isomorfismo o isomorfismo de subgrafos). *Sean $G_2, G_3 \in \Omega$. Se dice que existe un sub-isomorfismo de G_3 a G_2 si G_3 es isomorfo a un subgrafo de G_2, denotado como $G_3 \subset_s G_2$ (ver figura 6).*

Definición 6 (Frecuencia o soporte). *Sea una colección de grafos $D \subset \Omega$, $D = \{G_0, G_1, \ldots, G_{m-1}\}$ y $G \in \Omega$, la frecuencia de aparición o soporte de G en D, que se denotará como $sup(G,D)$, se define como la cantidad de grafos $G_i \in D$, tal que $G \subset_s G_i$.*

Se dice que un grafo G es frecuente en D, si $sup(G,D) \geq \delta$, dado un umbral de mínima frecuencia δ. De esta forma se define la minería de SF sobre una colección de grafos D como el problema de encontrar todos los SF en D tales que $sup(G,D) \geq \delta$.

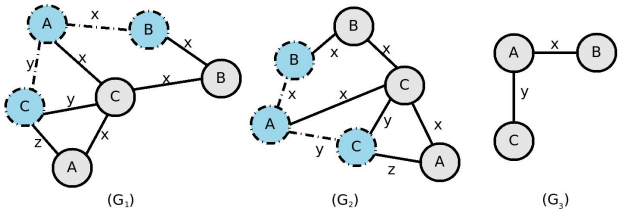

Fig. 6. Ejemplo de sub-isomorfismo entre G_3 y G_1 y entre G_3 y G_2.

2.1.2 Cotejo inexacto de grafos

Como se mencionó anteriormente, existen aplicaciones donde el cotejo inexacto de grafos es requerido para la modelación de determinados problemas en estas. Esto se debe a la existencia de diferencias entre etiquetas o en la estructura de la información representada por los grafos.

En este epígrafe se presentará un resumen de las conceptos preliminares de cotejo inexacto de grafos. En la sección 2.2 se presentarán los conceptos generales de las técnicas de cotejo inexacto.

Definición 7 (Función de similitud). *Dado el conjunto de todos los posibles grafos etiquetados Ω, la similitud entre elementos de Ω será definida como una función $sim : \Omega \times \Omega \to \mathbb{R}_+$. Donde 0 significa que son muy diferentes y mientras mayor sea el valor más parecidos son los elementos.*

Definición 8 (Ocurrencia aproximada). *Sean $G_1 \in \Omega$, $G_2 \in \Omega$ dos grafos etiquetados, $sim : \Omega \times \Omega \to \mathbb{R}_+$ una función de similitud, y τ un umbral de mínima similitud. Se dice que $g \subseteq G_2$ es una ocurrencia de G_1 en G_2 si $sim(G_1, g) \geq \tau$.*

De esta manera se define el *conjunto de ocurrencias* de G_1 en G_2, denotado por $O(G_1, G_2)$, como:

$$O(G_1, G_2) = \{g_i | sim(G_1, g_i) \geq \tau, g_i \subseteq G_2\}. \tag{1}$$

Ejemplo 1. Dados $G_1, G_2, g \in \Omega$, un umbral de mínima similitud $\tau = 0{,}5$ y conociendo que $g_1, g_2 \subseteq G_2$, $sim(G_1, g_1) = 1{,}0$ y $sim(G_1, g_2) = 0{,}6$, el conjunto $O(G_1, G_2) = \{g_1, g_2\}$, donde $g_1 = (\{v_0, v_1\}, \{(v_0, v_1)\}, \{A, \emptyset\}, l)$ y $g_2 = (\{v_0, v_2\}, \{(v_0, v_2)\}, \{A, C, \emptyset\}, l)$ (ver figura 7).

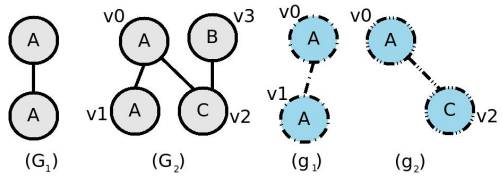

Fig. 7. Ejemplo de ocurrencias de un grafo G_1 en otro G_2, dado un umbral de mínima similitud $\tau = 0{,}5$.

Definición 9 (Soporte aproximado). *Dada una colección de grafos $D \subset \Omega$, $D = \{G_0, G_1, \ldots, G_{m-1}\}$, un grafo $T \in \Omega$, el conjunto $O(T, G_i)$, el soporte aproximado del grafo T en D se define como:*

$$supApp(T, D) = |\{G_i \in D | O(T, G_i) \neq \emptyset\}|. \tag{2}$$

Definición 10 (Subgrafo aproximado frecuente (SAF)). *Dada una colección de grafos $D \subset \Omega$, $D = \{G_0, G_1, \ldots, G_{m-1}\}$, un umbral de mínima frecuencia δ, se dice que un grafo $T \in \Omega$ es aproximado frecuente en D si $supApp(T, D) \geq \delta$.*

La minería de subgrafos frecuentes con el uso de cotejo inexacto entre grafos sobre una colección D se define como el problema de encontrar todos los SAF en D.

Por ejemplo, si se utiliza $\tau = 0{,}50$ como umbral de mínima similitud, $\delta = 2$ como umbral de mínima frecuencia en la colección D ilustrada en la figura 1 se obtendrán cinco SAF (ver figura 8), conociendo que $sim(SF_5, g) = 0{,}6$, donde $g = (\{v_0, v_1, v_2\}, \{(v_0, v_1), (v_0, v_2)\}, \{A, C, \emptyset\}, l)$ y $g \subseteq G_2$. Nótese, que el conjunto soporte del subgrafo SF_5 es $supApp(SF_5, D) = |\{G_2, G_3\}|$ y $supApp(SF_5, D) = 2$.

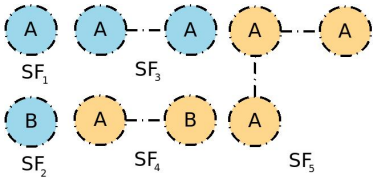

Fig. 8. Los SF de la colección D mostrada en la figura 1 utilizando $\delta = 2$ como umbral de mínima frecuencia y $\tau = 0{,}5$ como umbral de mínima similitud.

2.2 Ejemplos de técnicas de cotejo inexacto

En este epígrafe se presentarán los conceptos generales de las técnicas de cotejo inexacto de grafos que se han usado en la minería de SAF. En la sección 3 se presentará el uso de estas técnicas dentro de la minería de SAF.

2.2.1 Distancia de edición de grafos

Una técnica utilizada para calcular la similitud entre dos grafos es la *Distancia de Edición de Grafos* (DEG) [11]. En la literatura es posible encontrar varias definiciones de la DEG [26],[27],[28],[29]. A continuación se presentará una generalización de la DEG basada en la definida por Kaspar Riesen *et al.* [11].

La definición de *operación de edición* y *camino de edición* son requeridas para formalizar de la DEG como función de similitud. Esto se debe a que la DEG se define sobre la base de estas definiciones.

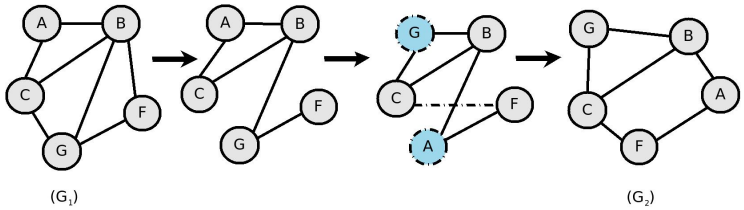

Fig. 9. Este posible camino de edición entre el grafo G_1 y el grafo G_2 consiste en la eliminación de dos aristas, la inserción de una arista y la sustitución de dos vértices.

Definición 11 (Operación de edición). *Se conoce como el conjunto de operación de edición estándar sobre un grafo G a eliminar, reetiquetar o sustituir, e insertar un vértice o una arista en G. Existen también otras operaciones de edición como la fusión o división de vértices o aristas [30].*

En lo adelante cuando se hable de operación de edición se estará refiriendo a una operación estándar. En caso de que no se trate de una operación estandar se especificará explícitamente.

Definición 12 (Camino de edición). *Sean $G_1, G_2 \in \Omega$, un camino de edición de G_1 a G_2, denotado por CE, es un conjunto de operaciones de edición $CE = \{e_{1_1}, \ldots, e_{1_k}\}$ (ver figura 9).*

Existen un conjunto de caminos de edición de G_1 a G_2, denotado por $\Upsilon(G_1,G_2) = \{CE_1,\ldots,CE_m\}$. El costo de un camino de edición, denotado por $Cost(CE)$, se calcula en función de los valores de costo de sus operaciones de edición. Estos valores de costo de las operaciones de edición son definidos según el tipo de aplicación. En la sección 3.1 se presenta un ejemplo de función de costo para un camino de edición.

Definición 13 (Distancia de edición de grafo (DEG)). *Sean $G_1,G_2 \in \Omega$, la DEG entre G_1 y G_2, denotada por $DEG(G_1,G_2)$, se obtiene mediante el $CE_i \in \Upsilon(G_1,G_2)$ de menor costo.*

$$DEG(G_1,G_2) = Min(\{Cost(CE_i)|CE_i \in \Upsilon(G_1,G_2)\}) \ , \tag{3}$$

donde mientras más pequeña sea la DEG resultante el costo de edición será menor, siendo muy similares los grafos en término de estructura y etiquetas.

Existe otra manera de calcular la DEG conocida como *distancia de inserción* [5]. El camino de edición mediante el cual se obtiene la distancia de inserción consiste en eliminando todos los vértices y aristas de G_2 que no aparecen en G_1 y se insertan los vértices y aristas en G_2 que aparecen en G_1 pero no aparecen en G_2.

Definición 14 (Costos distancia de inserción). *Dados $G_1,G_2 \in \Omega$, $G_1 = (V_1,E_1,L_1,l_1)$, $G_2 = (V_2,E_2,L_2,l_2)$, se define el costo de la distancia de inserción de G_1 a G_2 en función de los valores de insC y delC, tales que:*

- $insC = iC * (Size(G_1) - I(G_1,G_2))$,
- $delC = dC * (Size(G_2) - I(G_1,G_2))$,

donde iC y dC denotan los valores de costo de una operación de inserción y eliminación respectivamente, $Size(G_1) = |V_1| + |E_1|$, $I(G_1,G_2) = |(V_1 \cap V_2)| + |(E_1 \cap E_2)|$.

La complejidad computacional del cálculo de la DEG es exponencial en términos del número de vértices del grafo [31],[32]. Esto se debe a que es necesario explorar todos los posibles caminos de edición en un grafo para determinar el de menor costo. No obstante, algunos autores han aplicado diferentes heurísticas que permiten no procesar todos los posibles caminos de edición [5].

2.2.2 β arista isomorfismo

Otro ejemplo de función de similitud es definida en términos de β *arista isomorfismo* (ver definición 15). Mediante esta función se calcula la similitud entre dos grafos tratando la falta de aristas entre grafos, manteniendo exactos los vértices [24].

Definición 15 (β arista isomorfismo). *Se dice que f es un β arista isomorfismo entre $G_1,G_2 \in \Omega$, $G_1 = (V_1,E_1,L_1,l_1)$ y $G_2 = (V_2,E_2,L_2,l_2)$, denotado por $G_1 \approx_\beta G_2$, si $f : V_1 \to V_2$ es una función biyectiva tal que*

- $\forall u \in V_1, l_1(u) = l_2(f(u))$,
- $\forall (u,v) \in E_1 \ \exists (f(u),f(v)) \in E_2$, *tales que existan como máximo β aristas (u,v) donde $l_1((u,v)) \neq l_2((f(u),f(v)))$,*
- *si $l_1((u,v)) \neq l_2((f(u),f(v)))$, entonces $(u,v) \notin E_1$.*

Definición 16 (β arista sub-isomorfismo). *Dados $G_1,G_2 \in \Omega$, $G_1 = (V_1,E_1,L_1,l_1)$ y $G_2 = (V_2,E_2,L_2,l_2)$, se dice que G_1 es β arista sub-isomorfo a G_2, denotado por $G_1 \subseteq_\beta G_2$, si y sólo si $\exists g \subset_s G_2$ tal que $g \approx_\beta G_1$.*

En la figura 10 se muestra un ejemplo de β arista sub-isomorfismo con $\beta = 3$ para una mejor comprensión de la definición anterior.

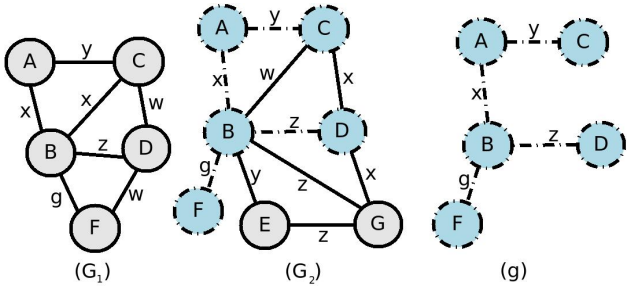

Fig. 10. Ejemplo con $\beta = 3$ donde $G_1 \subseteq_\beta G_2$, ya que G_2 contiene un subgrafo g tal que $g \approx_\beta G_1$.

2.2.3 Homeomorfismo con vértice/arista disjuntas

La siguiente función de similitud se define en términos de *homeomorfismo con vértice/arista disjuntas* (ver definición 22). Mediante esta calcula la similitud entre dos grafos basada en sus topologías menor [6],[16].

Las siguientes definiciones son requeridas para lograr una definición formal de la función homeomorfismo con vértices/aristas disjuntas. Esto se debe a que esta similitud se presenta en función de la *topología menor* de los grafos involucrados y esta última se presenta en función de los *caminos independientes*.

Definición 17 (Camino). *Sea $G \in \Omega$, $G = (V,E,L,l)$. Se dice que $P = (v_1, v_2, \ldots, v_k)$ es un camino en G si:*

- $\forall v_i \in V | v_i \in P$,
- $\forall (v_i, v_{i+1}) \in E | v_i, v_{i+1} \in P$.

En tal caso se dice que v_1 y v_k están conectados por P y se conocen como los extremos de P.

Se dice que un camino P es un *camino simple* si todos sus vértices son distintos. De esta forma se denota como $PS(G) = \{P_1, \ldots, P_m\}$ al conjunto de todos los caminos simples existentes en G.

Definición 18 (Camino independiente). *Sea $G \in \Omega$, $G = (V,E,L,l)$ y un camino $P = (v_1, v_2, \ldots, v_k)$. Se dice que P es un camino independiente si las intersecciones con otros caminos $P_i \subseteq V$ son solamente en sus extremos.*

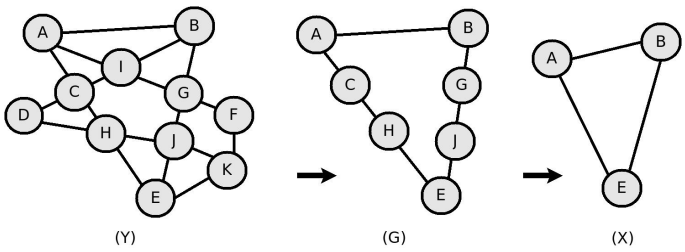

Fig. 11. Una topología menor obtenida mediante la contracción de todas las aristas de G hasta obtener los caminos independientes entre sus extremos.

Definición 19 (Subdivisión y contracción de arista). *Se conoce como subdivisión de un grafo $G = (V,E,L,l)$ a la sustitución de la arista $(v_1,v_2) \in E$ por un par de aristas (v_1,z), (z,v_2), donde $V = V \cup \{z\}$, $E = E \cup \{(v_1,z),(z,v_2)\} \setminus \{(v_1,v_2)\}$. La operación contraria a una subdivisión es conocida como contracción de arista.*

Definición 20 (Sub-homeomorfismo). *Se dice que existe un sub-homeomorfismo entre G_1 y G_2 si existe un isomorfismo entre alguna subdivisión de G_1 y una subdivisión de G_2. En caso de existir este sub-homeomorfismo se denotará por $G_1 \subset_h G_2$.*

Definición 21 (Topología menor). *Dado $G \in \Omega$, se genera una topología menor de G al contraer los caminos independientes de uno de sus subgrafos hasta transformarlos en aristas.*

Ejemplo 2 (Topología menor). Dados $X, G, Y \in \Omega$ (ver la figura 11), X es una topología menor de Y, puesto que X es generado por la transformación de los caminos independientes de $G \subseteq Y$.

Definición 22 (Sub-homeomorfismo con vértice/arista disjuntas). *Dados $G_1, G_2 \in \Omega$, $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$, se dice que G_1 es una topología menor y un sub-homeomorfismo con vértices/aristas disjuntas de G_2 si:*

- $G_1 \subset_h G_2$,
- $\exists f : E_1 \rightarrow PS(G_2)$,
- $\forall P_i \in PS(G_2)$ *que mapearon son independientes.*

2.2.4 Isomorfismo

El isomorfismo (ver definición 4) ha sido utilizado como función de similitud entre grafos para la minería de SAF [4],[33]. Esto se debe a que a los algoritmos reportados obtienen la aproximación entre los grafos mediante el uso de funciones heurísticas para el cálculo de la frecuencia aproximada (ver sección 3.4). En función de este soporte es que se identifican los posibles SAF. Esta función se apoya en la información adicional que presentan los grafos acerca de la probabilidad de que existan sus vértices y aristas.

2.2.5 Basadas en las probabilidades de sustitución

Existe una función de similitud entre dos grafos que se define en término de *sub-isomorfismo aproximado* [14] y una función de disimilitud definida en término de *grado de aproximación* [3]. Estas se basan en las probabilidades de que las etiquetas puedan ser sustituidas por otras. Este enfoque permite distinguir las sustituciones de las etiquetas de los vértices y aristas de los grafos, característica que no presentan las funciones de similitud mencionadas anteriormente.

La distinción de las sustituciones entre etiquetas se ha realizado mediante el uso de una función de probabilidad por cada etiqueta sobre todo el dominio de estas. Esta función de probabilidad da como resultado cuán probable sería que una etiqueta u pudiera tomar el valor de otra v.

Definición 23 (Función de probabilidad). *Una función de probabilidad de una variable aleatoria discreta*[2] *u, definida por $p(v)$, es una función tal que, al sustituir v por un valor de la variable, el valor que toma la función es la probabilidad de que la variable u asuma el valor de v. Para ello la función debe cumplir las siguientes propiedades:*

- $\forall v, p(v) \geq 0$,
- $\sum_v p(v) = 1$,
- $p(v) = P(u = v)$

[2] variable discreta: Es la variable que presenta separaciones o interrupciones en la escala de valores que puede tomar. Estas separaciones o interrupciones indican la ausencia de valores entre los distintos valores específicos que la variable pueda asumir.

Definición 24 (Matriz de compatibilidad). *Una matriz de compatibilidad $M = (m_{i,j})$ es una matriz $n \times n$ indizada por el conjunto de símbolos de etiquetas L ($n = |L|$). Una celda $m_{i,j} = P(l(i) = l(j)) = p_i(l(j)) | \sum_{j=1}^{n} p_i(l(j)) = 1$ en M es la probabilidad de que una etiqueta i pueda ser reemplazada por otra etiqueta j.*

Se dice que una matriz de compatibilidad M es estable si es diagonal predominante (i.e. $m_{i,i} > m_{i,j}, \forall j \neq i$). Este tipo de matriz especifica que una etiqueta tiene mayor probabilidad de ser reemplazada por ella misma que por otra etiqueta. Estas matrices no se diferencian por el tipo de grafo que se defina en el problema a tratar.

La función conocida como *sub-isomorfismo aproximado* toma como base la información presente en las matrices de compatibilidad para calcular la similitud entre dos grafos.

Definición 25 (Sub-isomorfismo aproximado). *Dados $G_1, G_2 \in \Omega$, $G_1 = \{V_1, E_1, L_1, l_1\}$ y $G_2 = \{V_2, E_2, L_2, l_2\}$, y un umbral de mínima similitud τ, se dice que el grafo G_1 es sub-isomorfo aproximado a G_2, denotado por $G_1 \subseteq_a G_2$ si existe una función biyectiva $f : V_1 \to V_2$, tal que:*

- $\prod_{u \in V_1} M_{l_1(u), l_2(f(u))} \geq \tau$,
- $\forall u, v \in V_1, (u,v) \in E \Leftrightarrow (f(u), f(v)) \in E_2$,
- $\forall (u,v) \in E_1, l_1(u,v) = l_2(f(u), f(v))$.

La función de disimilitud conocida como *grado de aproximación* se calcula teniendo en cuenta la aproximación de los vértices y aristas de los grafos (ver definición 28). En esta se utiliza una lista por cada $u \in V'$ de los vértices que lo pueden sustituir, denotada por $matchable(u) \subseteq V$, donde $G = (V,E)$, $g = (V', E') \in \Omega$ y $g \subseteq G$.

Definición 26 (Penalización de vértices). *Para un vértice $u \in V'$ y una lista $matchable(u)$ se calcula la disimilitud entre los vértices de dos grafos de la siguiente manera.*

$$dis\_sim_v(u, f(u)) = \begin{cases} < \infty & si\ f(u) \in matchable(u) \\ \infty & en\ otro\ caso \end{cases} \tag{4}$$

De esta forma se garantiza que una aproximación ocurre sólo dentro del conjunto existente en $matchable(u)$.

Definición 27 (Penalización de aristas). *Para una arista $(u,v) \in E'$ y sus imágenes $(f(u), f(v)) \in E$ se calcula la distancia entre estas de la siguiente manera.*

$$dis\_sim_e(u,v) = |dist[u,v] - dist[f(u), f(v)]| \tag{5}$$

Existe una alternativa para los casos en que no se conoce el peso de las aristas o sus etiquetas, simplemente se verifica que sus vértices son vecinos y de esta manera se pueden conformar los elementos de 5:

$$dist[u,v] = \begin{cases} 1 & si\ existe\ la\ arista \\ 2 & en\ otro\ caso \end{cases} \tag{6}$$

Definición 28 (Grado de aproximación). *Dados $G, g \in \Omega$, $G = (V,E)$, $g = (V', E')$ y $g \subseteq G$ se calcula el grado de la aproximación de la siguiente manera.*

$$approx(g \to^f G) = \sum_{u \in V'} dis\_sim_v(u, f(u)) + \sum_{u,v \in V'} dis\_sim_e(u,v) \tag{7}$$

2.3 Formas canónicas para grafos

Las formas canónicas de los grafos son representaciones únicas de cada grafo. Estas representaciones se construyen mediante una secuencia de símbolos que describen la topología del grafo y el valor de sus etiquetas. Las formas canónicas son obtenidas al seleccionar la secuencia que cumpla ciertas características entre todas las posibles secuencias que representan al grafo, según criterios de orden lexicográfico entre los símbolos. Las formas canónicas más utilizadas por los algoritmos para la minería de SF son: la codificación DFS (por sus siglas en inglés *Depth First Search*), la codificación BFS (por sus siglas en inglés *Breadth First Search*) y la codificación CAM (por sus siglas en inglés *Canonical Adjacency Matrix*).

En la minería de grafos no es trivial asegurar que el mismo subgrafo es verificado sólo una vez ya que puede ser obtenido por extensiones en diversas vías, adicionando los mismos vértices y aristas en diferentes órdenes. La detección de duplicados puede hacerse manteniendo una estructura de datos que almacene los grafos ya analizados y aplicando múltiples comprobaciones sobre dicha estructura. Esta variante garantiza validez en los resultados pero puede ser muy costosa en tiempo de ejecución y consumo de memoria. Es por eso que el uso de las formas canónicas en el recorrido del espacio de búsqueda se ha convertido en algo tradicional para los algoritmos que realizan minería de SF [2].

La codificación CAM es una de las formas canónicas utilizadas por algunos de los algoritmos reportados en el estado del arte que hacen uso de estas [14],[24]. Esta codificación se presentará en este epígrafe con el objetivo de una mejor comprensión de dichos algoritmos introducidos en la sección 3.

Al representar la estructura de un grafo etiquetado usando *matrices de adyacencia* (ver definición 29) es posible definir una secuencia única para dicho grafo conocida como codificación CAM.

Definición 29 (Matriz de adyacencia). *Dado un grafo etiquetado $G = (V,E,L,l)$ tal que $|V| = n$ y una permutación de los vértices en $V =< v_0, v_1, \ldots, v_{n-1} >$, la matriz de adyacencia de G respecto a dicha permutación es $X = (x_{i,j})_{i,j=0}^{n-1}$:*

$$x_{i,j} = \begin{cases} l(v_i) & si\ i = j \\ l(e) & si\ e = \{v_i, v_j\} \in E \\ 0 & si\ \{v_i, v_j\} \notin E \end{cases} \tag{8}$$

Dado que cada entrada diagonal representa un vértice en el grafo y cada permutación del conjunto de vértices corresponde a una matriz de adyacencia diferente, se obtienen $n!$ matrices diferentes para un grafo de n vértices. Caracterizar un grafo etiquetado requiere de una representación unívoca de dicho grafo y de todos aquellos que son isomorfos con éste. Es por eso que se define la *matriz de adyacencia canónica* partiendo del código de la matriz. Una manera de definir el código de G usando X, lo cual no indica que sea la única, es concatenando las filas de la submatriz triangular inferior (la matriz triangular inferior incluyendo la diagonal) [17] ver (9).

$$codeCAM(X) = x_{0,0}x_{1,0}x_{1,1}\ldots x_{n-1,n-1} \tag{9}$$

Cuando dos códigos representan al mismo grafo siempre tienen la misma cantidad de símbolos componiendo la secuencia. Es por eso que estos códigos se pueden comparar lexicográficamente teniendo en cuenta el orden de las etiquetas. De esta forma, es posible definir la CAM de un grafo como la matriz que da lugar al código lexicográficamente mayor entre los posibles códigos de G. En la figura 12 se muestra un ejemplo de matrices de adyacencia generadas mediante diferentes recorridos sobre un grafo G, donde la primera matriz es CAM y el resto de las matrices no son CAM. Es por eso que solo el primer código representa el $codeCAM(G)$.

Por las características que definen estas secuencias, cuando dos códigos son iguales significa que fueron construidos partiendo de matrices de adyacencia iguales. Por tanto, cuando dos códigos CAM de dos grafos son iguales, entonces se está en presencia de dos grafos isomorfos. Sin embargo, el comprobar que un grafo está representado por su forma canónica puede ser muy costoso en tiempo. Esto se debe a que se requiere enumerar exhaustivamente todas las posibles secuencias para seleccionar de estas la canónica. Estas pruebas exhaustivas se conocen como pruebas de formas canónicas.

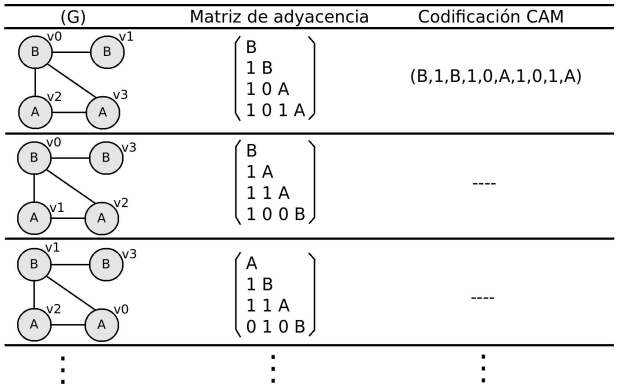

Fig. 12. Ejemplo de matrices de adyacencia generadas mediante diferentes recorridos sobre un grafo G y el $codeCAM(G)$.

2.4 Estructuras de datos para indizar la colección de grafos

Tanto en los algoritmos que utilizan técnicas de cotejo exacto como en los que utilizan técnicas de cotejo inexacto el conteo de frecuencia es uno de los procesos fundamentales para llevar a cabo la minería de SF. Al usar técnicas de cotejo inexacto, las pruebas de aproximación estructural entre grafos hacen que este proceso sea más crítico que al usar técnicas de cotejo exacto. Con el propósito de acelerar algunos procesos dentro de la búsqueda como el conteo de frecuencia, se hace necesario el uso de estructuras de datos que indicen la colección de grafos para este tipo de minería. De esta manera se evita un engorroso proceso de búsqueda exhaustiva para cada candidato G en la colección de grafos D.

Las estructuras de datos utilizadas por los algoritmos reportados para indizar la colección de grafos en la minería de SF pueden clasificarse en dos tipos: las listas de identificadores y las estructuras de correspondencias.

2.4.1 Listas de identificadores

La lista de identificadores de un grafo candidato T sobre la colección de grafos $D = \{G_1, \ldots, G_n\}$ es una lista de los índices i de los grafos G_i que contienen al menos una ocurrencia de T.

$$LI(T,D) = \{i | \exists g \in O(T,G_i), G_i \in D, g \subset_s G_i\} \tag{10}$$

Esta variante surge por la necesidad de conocer en cuáles grafos G_i de la colección se encuentra el patrón T para en ellos realizar la búsqueda de los candidatos $T \diamond (u,v)$. De esta forma se disminuye el número de grafos a procesar, ya que solamente se procesan los grafos que contengan una o varias ocurrencias de T.

Para mantener las listas no se requiere de gran poder de cómputo, ya que la lista del nuevo candidato se obtiene mediante la lista del candidato que le da origen. Para almacenar dichas listas no se requiere de gran capacidad de almacenamiento comparado con las *estructuras de correspondencias* (ver subsección 13) y disminuye su tamaño mientras mayor se va haciendo el candidato generado. Sin embargo, no es la

más recomendable a usar ya que no elimina las pruebas de sub-isomorfismo por cada grafo candidato en el grafo de la colección. Estas pruebas de sub-isomorfismo son más costosas cuando se utilizan técnicas de cotejo inexacto que con el uso de técnicas de cotejo exacto en este tipo de minería. Los algoritmos RNGV [5], Monkey [24] usan este tipo de estructura.

El algoritmo 1 se encarga de obtener los hijos de T que ocurren en $G_i \in D$ usando $LI(T,D)$ y actualizar o crear su correspondiente lista de identificadores (ver línea 6). Este algoritmo comienza con la obtención del conjunto de ocurrencias aproximadas de T en G_i mediante las costosas pruebas de sub-isomorfismo inexacto en este caso (ver línea 1). El cálculo de estas listas de identificadores se puede realizar durante el proceso de detección de los candidatos.

Algoritmo 1: $BCandidatos\_LI(T, G_i, \tau, C)$

Input: $T = (V,E)$: Un grafo candidato, G_i : Un grafo, τ : Umbral de mínima similitud.
Output: C : Los hijos de T que ocurren en G_i.

```
1 O(T,G_i) ← {Conjunto de ocurrencias de T en G_i};
2 foreach (u,v) ∈ G_i, donde (u,v) ∈ N(o_j), o_j ∈ O(T,G_i) do
3     foreach w ∈ V', donde (u,w) ∉ E y V' es el conjunto de todos los vértices con etiquetas diferentes de D do
4         if Sim(T⋄(u,w), o_j⋄(u,v)) ≥ τ then
5             C ← C ∪ {T⋄(u,w)};
6             LI(T⋄(u,w),D) ← LI(T⋄(u,w),D) ∪ {i};
```

2.4.2 Estructuras de correspondencias

Las estructuras de correspondencias son creadas con el objetivo de mantener todas las ocurrencias de un candidato en cada grafo contenido en D en la minería de SF. Estas estructuras requieren muchos más espacio en memoria que las listas de identificadores pero eliminan pruebas de sub-isomorfismo de manera exhaustiva de los grafos que contienen al candidato. Este tipo de estructuras han sido usadas en los algoritmos SUBDUE [34], gApprox [3] y APGM [14].

Cada algoritmo en particular define sus estructuras de correspondencias según sus características específicas como las condiciones específicas del recorrido del espacio de búsqueda o la información necesaria a mantener por las ocurrencias. Algunos algoritmos para la minería presentan estructuras que solamente mantienen las ocurrencias de los vértices [14] y en otros casos se mantienen las ocurrencias de las aristas utilizando códigos DFS como las estructuras DFSE (por sus siglas en inglés *Depth First Search Embedding*) [2].

Para el uso de las estructuras de correspondencias, los algoritmos parten de los SF que tienen un vértice $T = (\{v\}, \emptyset, \{l(v)\}, l)$. Sea $G_i \in D$ el i-ésimo grafo de la colección que contiene al menos una ocurrencia de T, una correspondencia de T en $G_i = (V_i, E_i, L_i, l_i)$ se define como (j, λ) tal que λ[3] es una referencia vacía y j es el índice de un vértice de G_i que tiene asignada la etiqueta $l(v_j)$. La lista de correspondencias de T en G_i se define según (12), dada una función de similitud para las etiquetas de los vértices, denotada por $sim(l(u), l(v))$ y un umbral de mínima similitud τ.

$$LC(T, G_i) = \{(j, \lambda) | G_i \in D, v_j \in V_i, sim(l_i(v_j), l(v)) \geq \tau\} \tag{11}$$

Las listas de correspondencias para grafos de mayor tamaño se definen de manera recurrente. Sea T un SF y $G_i \in D$, $LC(T \diamond (u,v), G_i)$ se obtiene usando $LC(T, G_i)$ de la siguiente manera:

$$LC(T \diamond (u,v), G_i) = \{(k, \lambda) | G_i \in D, \lambda = LC(T, G_i), v_k \in V_i, sim(l_i(v_j), l(v)) \geq \tau\} \tag{12}$$

[3] La componente λ será una referencia vacía solamente para las correspondencias de grafos de un vértice

El algoritmo 2 se encarga de obtener los hijos de T que ocurren en $G_i \in D$ usando $LC(T,G_i)$ y construye las listas de correspondencias de los nuevos candidatos durante la búsqueda. Estas listas obtenidas son listas de pares ordenados (j,λ) donde j es un identificador de vértices o aristas en el grafo $G_i \in D$, y λ es un elemento de $LC(T,G_i)$. La lista $LC(T,G_i)$ fue creada con anterioridad usando (11) o mediante llamadas anteriores de este procedimiento usando (12). Las pruebas de sub-isomorfismo entre T y sus ocurrencias se obtienen recorriendo la lista $LC(T,G_i)$ a través de λ (ver líneas 2-8).

Algoritmo 2: $BCandidatos\_LC(T,LC(T,G_i),G_i,\tau,C)$

Input: $T=(V,E)$: Un grafo candidato, $LC(T,G_i)$: Lista de correspondencias de T en G_i, G_i : Un grafo, τ : Umbral de mínima similitud.
Output: C : Los hijos de T que ocurren en G_i.

```
1  foreach ptr = (t,λ) ∈ LC(T,G_i) do
2      while λ no apunte a null do
3          if t es un identificador de vértice then
4              E' ← E' ∪ {(u,v_t)}; //Donde u ∈ V
5              V' ← V' ∪ {v_t};
6          else
7              E' ← E' ∪ {(u,v)_t}
8          (t,λ) ← El par apuntado por λ;
9      V' ← V ∪ {v_t};
10     g = (V',E');//Donde g ⊆_s G_i
11     foreach (u,v_j)_k ∈ G_i, donde (u,v_j)_k ∈ N(g) do
12         foreach w ∈ V', donde (u,w) ∉ E y V' es el conjunto de todos los vértices con etiquetas diferentes de D do
13             if Sim(T⋄(u,w), g⋄(u,v_j)_k) ≥ τ then
14                 C ← C ∪ {T⋄(u,w)};
15                 if w ∉ V then
16                     LC(T⋄(u,w),G_i) ← LC(T⋄(u,w),G_i) ∪ {(j,ptr)};
17                 else
18                     LC(T⋄(u,w),G_i) ← LC(T⋄(u,w),G_i) ∪ {(k,ptr)};
```

La estructura de correspondencias de un grafo T respecto a la colección D es una estructura de datos que contiene todas las listas de correspondencias de $T \diamond (u,v)$ respecto a cada uno de los grafos $G_i \in D$:

$$EC(T \diamond (u,v),D) = \{LC(T \diamond (u,v),G_i) | G_i \in D, LC(T \diamond (u,v),G_i) \neq \emptyset\} \tag{13}$$

2.5 Síntesis y conclusiones

En esta sección se han definido formalmente los conceptos necesarios para una mejor comprensión del resto del documento. Se presentaron los conceptos básicos de la teoría de grafos que caracterizan el problema de la minería de SF. Se definió un lenguaje común que será utilizado en el resto del documento. También se presentaron ejemplos de técnicas de cotejo inexacto utilizadas como funciones de similitud entre grafos en la minería de SF.

Por otro lado, se mensionaron las diferentes formas canónicas para la representación de grafos etiquetados y se describió una de estas conocida como código CAM. Además, fueron presentados los detalles de las estructuras comunmente utilizadas para indizar la colección de grafos.

3 Estado del arte sobre el uso de cotejo inexacto en la minería de subgrafos frecuentes

En esta sección se presentan los algoritmos reportados para la minería de SAF. Estos algoritmos han sido clasificado según la técnica de cotejo que utilizan y la forma de obtener los subgrafos candidatos.

Generar los subgrafos candidatos de una manera efectiva incide en el buen desempeño de los algoritmos para este tipo de minería. Estos algoritmos pertenecen a dos grupos, los basados en *crecimiento de patrones* [35] y los basados en *Apriori* [36]. En el primer grupo se recorre en profundidad o amplitud el espacio de búsqueda para, generar nuevos candidatos a partir de las apariciones de un patrón en la colección. En el segundo grupo siempre se realiza una búsqueda en amplitud.

Los algoritmos 3 y 4 describen de manera general la forma de generar candidatos basados en la operación de extensión (crecimiento de patrones). En estos algoritmos las extensiones, ya sean cerradas o por un vértice, son mediante las aristas. Esto se debe a que por cada SAF se determinan qué aristas pudieran ser añadidas al SAF según la colección de grafos para la obtención de los nuevos candidatos. Para invocar estos algoritmo por primera vez es necesario obtener los SAF de tamaño 1 en un procedimiento aparte.

Algoritmo 3: $DExten(T, D, \delta, \tau, F)$

Input: T : Un SAF, D : Una colección de grafos, δ : Umbral de mínima frecuencia, τ : Umbral de mínima similitud.
Output: F : Conjunto de todos los SAF.

```
1 foreach G_i ∈ D do
2     foreach (u,v) ∈ G_i, donde (u,v) ∈ N(o_j), o_j ∈ O(T,G_i) do
3         score ← Sim(T ⋄ (u,v), o_j ⋄ (u,v));
4         if score ≥ τ then C ← C ∪ {T ⋄ (u,v)};
5 foreach G ∈ C do
6     if supApp(G,D) ≥ δ AND G ∉ F then
7         F ← F ∪ {G};
8         DExten(G,D,δ,τ,F);
```

En el algoritmo 3 se obtienen todos los SAF recorriendo el espacio de búsqueda en profundidad. Este permite ahorrar notablemente los recursos de memoria. Esto se debe a que es necesario mantener la información adicional de cada SAF solo de la rama activa. Esta información adicional es la estructura de datos utilizada para indizar la colección de grafos (ver sección 2.4).

El algoritmo 4 recorre el espacio de búsqueda en amplitud. En este recorrido no se pasa a la obtención de los SAF de tamaño $k+1$ mientras exista algún posible SAF de tamaño k por detectar. Por esta razón es necesario mantener la información de todos los SAF hasta el final del minado.

Algoritmo 4: $BExten(S, D, \delta, \tau, F)$

Input: S : Conjunto de SAF, D : Una colección de grafos, δ : Umbral de mínima frecuencia, τ : Umbral de mínima similitud.
Output: F : Conjunto de todos los SAF.

```
1 foreach T ∈ S do
2     foreach G_i ∈ D do
3         foreach (u,v) ∈ G_i, donde (u,v) ∈ N(o_j), o_j ∈ O(T,G_i) do
4             score ← Sim(T ⋄ (u,v), o_j ⋄ (u,v));
5             if score ≥ τ then C ← C ∪ {T ⋄ (u,v)};
6     foreach G ∈ C do if supApp(G,D) < δ then Se elimina G de C;
7 if C ≠ ∅ then
8     F ← F ∪ C;
9     BExten(C,D,δ,τ,F);
```

Por otro lado están los algoritmos Apriori que obtienen subgrafos candidatos mezclando los SAF mediante operaciones de *fusión*. De esta forma van obteniendo los SAF de mayor tamaño. En la minería de SAF, estos algoritmos tienen dos grandes dificultades relacionadas con la operación de fusión: es computacionalmente compleja en tiempo y genera muchos candidatos [2].

El algoritmo 5 muestra una descripción general de los algoritmos basados en Apriori. El tamaño de los nuevos SAF identificados en cada llamado recursivo se incrementa en uno respecto al llamado anterior. Para invocar este algoritmo por primera vez es debe obtener los SAF de tamaño 1 en un procedimiento aparte al igual que los algoritmos 3 y 4.

Algoritmo 5: $Apriori(S, D, \delta, \tau, F)$

Input: S : Conjunto de SAF, D : Una colección de grafos, δ : Umbral de mínima frecuencia, τ : Umbral de mínima similitud.
Output: F : Conjunto de todos los SAF.

```
1 foreach T_i ∈ S do
2     foreach T_j ∈ S do
3         foreach G ∈ Fusion(T_i, T_j) do if supApp(G, D) ≥ δ then C ← C ∪ {G};
4 if C ≠ ∅ then
5     F ← F ∪ C;
6     Apriori(C, D, δ, τ, F);
```

En el resto de esta sección se presentan conceptos ligados a problemas específicos unidos con los algoritmos que los usan. También se exponen varios comentarios y algunos problemas detectados que influyen de forma directa en la eficiencia y eficacia de dichos algoritmos.

3.1 Algoritmos basados en DEG

Como se planteó en la sección 2.1.2, esta técnica calcula el camino de edición de menor costo de un grafo a otro. El costo de un camino de edición es $Cost(CE, iC, sC, dC) = \sum_{i=1}^{n} c(e_i)$ en los algoritmos reportados que usan DEG [11],[37]; donde n es la cantidad de operaciones de edición, $c(e_i)$ es el costo de una operación de edición, e_i es su i-ésima operación de edición, iC, sC y dC son los costos de las operaciones de inserción, sustitución y eliminación respectivamente de vértices o aristas.

Ejemplo 3. Dados los valores asignados como ejemplo a los costos de las operaciones de edición sobre los vértices y aristas como, (1) por cada inserción, (0) por cada eliminación, (1) por cada reetiquetado o sustitución, se obtiene un costo total de $Cost(CE_1, iC, sC, dC) = 3$ para el camino de edición ilustrado en la figura 9. Sin embargo, al realizar las operaciones de edición mostradas en la Fig. 13 sobre los mismos grafos, se obtiene como costo resultante $Cost(CE_2, iC, sC, dC) = 2$. Suponiendo que solamente existen estos dos caminos de edición de G_1 a G_2 se obtiene $DEG(G_1, G_2, iC, sC, dC) = CE_2$, ya que su costo es el menor de los dos caminos de edición.

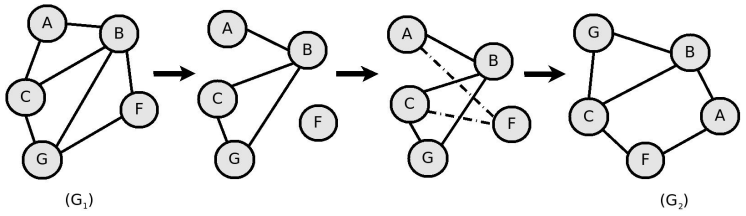

Fig. 13. Esta posible distancia de edición de grafo entre el grafo G_1 y el grafo G_2 consiste en la eliminación de tres aristas y la inserción de dos aristas.

Un pseudo-código genérico para calcular la DEG entre dos grafos presentado por Cook y Holder [37] se muestra en el algoritmo 6. Este se basa en mantener un conjunto de caminos de edición parciales para ser procesados y terminar tan pronto como un camino de edición completo[4] con un costo mínimo esté disponible. En las líneas 3 y 15 se obtiene y se elimina el candidato con costo mínimo del conjunto de caminos de edición parciales para ser procesados. En las líneas 6-14 se añaden nuevas operaciones de edición dando lugar a nuevos caminos de edición candidatos. En caso de que la arista $(u,v)_{k+1} \in E_1$ no haya sido procesada se insertan todas las operaciones de sustitución de esta arista a todas las aristas de G_2 en los caminos de edición candidatos, así como la eliminación de dicha arista (ver líneas 7-11). En caso de que todas las aristas de G_1 hayan sido procesadas, entonces se insertan todas las aristas restantes de G_2 generando los caminos candidatos parciales (ver líneas 12-14). Finalmente se obtiene como resultado el camino de edición con el mínimo costo de G_1 a G_2 y su costo.

Algoritmo 6: $DEG(G_1, G_2, iC, sC, dC, CE, Costo)$

Input: $G_1 = (V_1, E_1)$: Un grafo, $G_2 = (V_2, E_2)$: Un grafo, iC : Costo de la operación de inserción de arista o vértice, sC : Costo de la operación de sustitución de arista o vértice, dC : Costo de la operación de eliminación de arista o vértice.

Output: CE : Camino de edición que transforma $G1$ en $G2$ con mínimo costo, $Costo$: Costo de CE.

1 **foreach** $(u,v)_i \in E_2$ **do**
2 $\quad P \leftarrow P \cup \{(u,v)_1 \rightarrow (u,v)_i | (u,v)_1 \in E_1, (u,v)_1 \neq (u,v)_i\}$; //Operación de sustitución
3 $CE \leftarrow$`RemoveElementMinCost (`P, iC, sC, dC`)` ;
4 $Costo =$`Cost (`CE, iC, sC, dC`)` ;
5 **while** *CE no sea completo* **do**
6 $\quad$ **if** $k < |E_1|$ *tal que* $CE = \{(u,v)_1 \rightarrow (u,v)_{i_1}, \ldots, (u,v)_k \rightarrow (u,v)_{i_k}\}$ **then**
7 $\quad\quad$ **foreach** $(u,v)_i \in E_2 \setminus \{(u,v)_{i_1}, \ldots, (u,v)_{i_k}\}$ **do**
8 $\quad\quad\quad CE \leftarrow CE \cup \{(u,v)_{k+1} \rightarrow (u,v)_i | (u,v)_{k+1} \in E_1, (u,v)_{k+1} \neq (u,v)_i\}$;
9 $\quad\quad\quad P \leftarrow P \cup CE$;
10 $\quad\quad CE \leftarrow CE \cup \{(u,v)_{k+1} \rightarrow \varepsilon | (u,v)_{k+1} \in E_1\}$;
11 $\quad\quad P \leftarrow P \cup CE$;
12 $\quad$ **else**
13 $\quad\quad CE \leftarrow CE \cup \{\varepsilon \rightarrow (u,v)_i | (u,v)_i \in E_2 \setminus \{(u,v)_{i_1}, \ldots, (u,v)_{i_k}\}\}$;
14 $\quad\quad P \leftarrow P \cup CE$;
15 $\quad CE \leftarrow$`RemoveElementMinCost (`P, iC, sC, dC`)` ;
16 $\quad Costo =$`Cost (`CE, iC, sC, dC`)` ;

En general, la DEG es una de las técnicas de cotejo inexacto más aplicables entre las reportadas, ya que se ha utilizado en diferentes dominios de la ciencia. Una aplicación es en el análisis de imágenes, donde se pueden encontrar subestructuras correspondientes a objetos similares frecuentes en el entorno, o la detección de casos similares de una subestructura previamente descubierta en un nuevo entorno [9]. También es aplicable en el análisis de escenas y circuitos [13], de documentos [38], clasificación de emails y documentos [39],[40], en el ámbito de la química [41] y la biología molecular [5],[34].

3.1.1 SUBDUE

Este algoritmo, presentado por Cook y Holder [9] ha sido considerado el primero de todos los trabajos para la minería de SAF, con varias aplicaciones [42],[41],[43],[44]. SUBDUE, basándose en el principio de longitud mínima de descripción (MDL, de sus siglas en inglés, *Minimum Description Length*) [45] comprime los datos originales del grafo de entrada. Con el uso de un limitado cotejo inexacto de grafos [46] encuentra las instancias similares a una subestructura y la medida de su similitud. En este algoritmo el

[4] Un camino de edición es completo cuando contiene todas las posibles operaciones de edición de G_1 a G_2

usuario puede asignar los costos de las operaciones de edición que se le realizan a los subgrafos candidatos y definir el tamaño máximo de los subgrafos a obtener [13]. Mediante la determinación de los costos de las operaciones de edición el usuario puede influir en el cotejo a favor o en contra de determinados tipos de operaciones.

Este algoritmo se ha usado para procesar un solo grafo etiquetado y no dirigido, pero se plantea que puede utilizarse para procesar colecciones de grafos [34]. Sin embargo, no se han encontrado trabajos que aborden la variante del procesamiento de colecciones de grafos en SUBDUE y sin estos, no se conoce la forma de calcular la frecuencia del candidato. Una modificación de SUBDUE para trabajar sobre una colección D, pudiera ser definiendo a D como un gran grafo no conexo, entonces solamente se modificaría el cálculo del soporte enfocado a los subgrafos conexos.

SUBDUE utiliza MDL para evaluar las subestructuras candidatas manteniendo las que mejor comprimen el grafo de entrada, en lugar de calcular su frecuencia. Esto es posible debido a que mientras más ocurrencias existan de la subestructura en el grafo, más comprimido quedará y de esta forma se mantienen las subestructuras más frecuentes. El uso de MDL cuando se procese una colección sería mantener las subestructuras que mejor compriman la colección. Sin embargo, pudiera existir una subestructura S_i que tenga una o varias ocurrencias en un solo grafo de D pero su tamaño es lo suficientemente grande como para comprimir D mejor que otra subestructura S_j de menor tamaño que presenta ocurrencias en varios grafos de D. De esta forma se desechan candidatos como S_j para mantener subestructuras como S_i.

La complejidad computacional del cálculo de la DEG es exponencial en términos del número de vértices del grafo [31],[32]. Esto se debe a que es necesario explorar todos los posibles caminos de edición en un grafo para determinar el de menor costo. No obstante, algunos autores han aplicado diferentes heurísticas que permiten no procesar todos los posibles caminos de edición [5].

En SUBDUE se identifica el camino de edición de menor costo explorando todos los posibles caminos entre los grafos. Esto hace que este algoritmos tenga una complejidad computacional alta. En función del camino de menor costo CE y su costo resultante según los costos de las operaciones de edición dados por el usuario en el llamado al procedimiento $DEG(G_1, G_2, iC, sC, dC, CE, Costo)$ 6 se define la similitud entre grafos. Siendo $G_1 = (V_1, E_1, L_1, l_1)$ y $G_2 = (V_2, E_2, L_2, l_2)$, el tamaño del mayor grafo $Max(Size(G_1), Size(G_2))$ y el costo del camino de edición CE definido por $Costo$, se puede calcular $S(G_1, G_2, Costo)$ según (14). Cuando los dos grafos son muy parecidos $S(G_1, G_2, Costo) = 1$ y mientras más diferentes sean más se acerca a cero el valor de la similitud.

$$S(G_1, G_2, Costo) = 1 - \frac{Costo}{Max(Size(G_1), Size(G_2))} \tag{14}$$

En SUBDUE se presenta el esquema de compresión conocido como el principio MDL descrito por Rinssanen [45]. Este principio se basa en el número de bits necesarios para representar la subestructura. La mejor subestructura es la que reduce al mínimo el grafo de entrada G, $MDL = I(S) + I(G|S)$, donde S es la subestructura descubierta, G es el grafo de entrada, $I(S)$ es el número de bits necesarios para codificar a S y $I(G|S)$ es el número de bits necesarios para codificar a G después de haber sido comprimido usando S. Una vez que se encuentra una subestructura S, esta se utiliza para comprimir los datos de G mediante la sustitución de sus instancias por un puntero ptr a la nueva subestructura detectada. Por tanto, al ser codificado un subgrafo S de un grafo G, se reduce el espacio en memoria de G y se dice que ha sido comprimido usando S. El número de bits necesarios para representar G se define en (15), dada la subestructura detectada S, el número de instancias n encontradas de esta en G y el puntero ptr a la misma.

$$I(G|S) = I(G) - n(I(S) + I(ptr)) \tag{15}$$

Para evaluar lo bien que la subestructura encontrada S comprime el grafo de entrada G se hace como en (16). Tanto el grafo de entrada como la subestructura encontrada pueden ser codificadas usando el siguiente esquema.

$$Compression = 1 - \frac{I(S) + I(G|S)}{I(G)} \tag{16}$$

Cuando la compresión es mayor que cero, la representación de G usando S es usada en lugar de la representación original, ya que requiere menos bits. De esta manera se va comprimiendo el grafo de entrada G hasta lograr la mejor compresión posible de G según las subestructuras detectadas. Esta compresión permite minimizar, en cada iteración, el espacio de memoria utilizado para mantener a G.

En el algoritmo 7 se muestra el punto de partida del algoritmo de los tres pseudo-códigos que componen el algoritmo SUBDUE. Estos pseudo-códigos conforman una variante de SUBDUE presentada por N. S. Ketkat *et al.* 2005 [34], pero adaptada para el procesamiento de colecciones de grafos. El algoritmo *Subdue* comienza generando una subestructura por cada etiqueta de vértices con sus ocurrencias en la colección. A partir de dichas subestructuras se comienza el minado de SAF. Con el llamado al procedimiento *subdue_BExten* en la línea 3 se obtienen las extensiones en todas las posibles formas de cada subestructura identificada. Al finalizar el minado se retorna un subconjunto de subgrafos frecuentes.

Algoritmo 7: $Subdue(D, Beam, MB, MS, Limit, iC, sC, dC, \tau)$

Input: D : Una colección de grafos, *Beam* : Umbral máximo del espacio de búsqueda, MB : Umbral de la cantidad máxima de los subgrafos a encontrar, MS : Umbral del tamaño máximo de los subgrafos a encontrar, *Limit* : Número de estructuras a incluir en la búsqueda, iC : Costo de la operación de inserción de arista o vértice, sC : Costo de la operación de sustitución de arista o vértice, dC : Costo de la operación de inserción de eliminación de arista o vértice, τ : Umbral de mínima similitud.

Output: F : Conjunto de los SAF, tal que $|F| \leq MB$.

1 $S \leftarrow$ {Subestructuras generadas a partir de las etiquetas de los vértices};
2 Inicializar para cada subestructura $s \in S$ la lista de correspondencias $EC(s, D)$;
3 $subdue\_BExten(S, D, Beam, MB, MS, Limit, iC, sC, dC, \tau, 0, F)$;
4 **return** F;

Algoritmo 8: $subdue\_BExten(S, D, Beam, MB, MS, Limit, iC, sC, dC, \tau, Pro, F)$

Input: S : Conjunto de SAF, D : Una colección de grafos, *Beam* : Umbral máximo del espacio de búsqueda, MB : Umbral de la cantidad máxima de los subgrafos a encontrar, MS : Umbral del tamaño máximo de los subgrafos a encontrar, *Limit* : Número de estructuras a incluir en la búsqueda, iC : Costo de la operación de inserción de arista o vértice, sC : Costo de la operación de sustitución de arista o vértice, dC : Costo de la operación de eliminación de arista o vértice, τ : Umbral de mínima similitud, *Pro* : Cantidad de subgrafos procesados.

Output: F : Conjunto de los SAF, tal que $|F| \leq MB$.

1 **while** $S \neq \emptyset$ *AND* $Pro \leq Limit$ **do**
2 Almacenar en T el primer elemento de S; Eliminar de S el primer elemento;
3 **foreach** $(i, \lambda) \in EC(T, D)$ **do** $subdue\_BCandidatos\_LC(T, LC(T, G_i), G_i, \tau, iC, sC, dC, C)$;
4 **foreach** $G \in C$ **do**
5 **if** `Size(G)` $\geq MS$ **then** Eliminar G de C;
6 **else** Evaluar G y se reordena en C según su valor MDL de forma ascendente;
7 Eliminar los últimos elementos de C, tal que $|C| \leq Beam$;
8 Insertar T en F ordenado por su valor MDL de forma ascendente, manteniendo $|F| \leq MB$;
9 $Pro++$;
10 **if** $C \neq \emptyset$ AND $Pro \leq Limit$ **then** $subdue\_BExten(C, D, Beam, MB, MS, Limit, iC, sC, dC, \tau, Pro, F)$;

Los detalles del algoritmo *subdue_BExten* se presentan en el algoritmo 8. Este algoritmo es el encargado de realizar el crecimiento de patrones (ver analogía con el algoritmo 4). Los candidatos son obtenidos mediante el llamado al procedimiento *subdue_BCandidatos_LC* en la línea 3. No todos estos candidatos

son almacenados para proximos procesamientos, solo los que su tamaño no sobrepase el valor del umbral del tamaño máximo de los subgrafos a encontrar *MS* (ver línea 5). Por lo tanto, se desechan posibles SAF, permitiendo la pérdida de información y la obtención de un conjunto solución incompleto. De estos candidatos almacenados solamente se mantendrán para la próxima iteración los *Beam* que mejor compriman la colección de grafos D (ver línea 7). Este proceso se realizará hasta que se hayan identificado como máximo la cantidad límite de subgrafos definida por el usuario denotada por *Limit* (ver líneas 10). De esta forma no se obtendrán todos los subgrafos que mejor comprimen a D.

Los detalles del algoritmo de búsqueda de los subgrafos candidatos con el uso de las listas de correspondencias son mostrados en el algoritmo 9 (ver analogía con el algoritmo 2). De los candidatos obtenidos solamente se mantendrán los que cumplan con el umbral de mínima similitud τ y su valor de costo no sobrepase el tamaño del mayor subgrafo involucrado en la similitud (ver línea 7). El cálculo de dicha similitud se realiza mediante (14) basado en la DEG (ver líneas 5-8).

Algoritmo 9: *subdue_BCandidatos_LC*$(T, LC(T,G_i), G_i, \tau, iC, sC, dC, C)$

Input: $T=(V,E)$: Un grafo candidato, $LC(T,G_i)$: Lista de correspondencias de T en G_i, G_i : Un grafo, τ : Umbral de mínima similitud, *iC* : Costo de la operación de inserción de arista o vértice, *sC* : Costo de la operación de sustitución de arista o vértice, *dC* : Costo de la operación de eliminación de arista o vértice.

Output: C : Los hijos de T que ocurren en G_i.

```
1  foreach ptr = (t, λ) ∈ LC(T, G_i) do
2      Obtener la ocurrencia aproximada g de T en G_i mediante las pruebas de sub-isomorfismo recorriendo LC(T, G_i) a
       través de λ;
3      foreach (u, v_j)_k ∈ G_i, donde (u, v_j)_k ∈ N(g) do
4          foreach w ∈ V', donde (u, w) ∉ E y V' es el conjunto de todos los vértices con etiquetas diferentes de D do
5              DEG (T ◇ (u, w), g ◇ (u, v_j)_k, iC, sC, dC, CE, Costo) ;
6              if CE ≠ ∅ then
7                  if S(T ◇ (u, v), g ◇ (u, v_j)_k, Costo) ≥ τ AND Costo ≤ Size(T ◇ (u, w)) then
8                      C ← C ∪ {T ◇ (u, w)};
9                      Crear o actualizar LC(T ◇ (u, w), G_i) con el par ordenado (j, ptr) si es una extensión por un
                       vértice o (k, ptr) en caso de que sea cerrada;
```

Normalmente, cuando la longitud de descripción de la extensión de una subestructura comienza a aumentar, una futura extensión de esta otra no obtendrá una longitud más pequeña. Por tal motivo, este algoritmo hace uso de un mecanismo de poda que elimina las extensiones de las subestructuras del conjunto a mantener cuando su valor MDL se incrementa.

3.1.2 RNGV

El algoritmo RNGV presentado por Song y Chen [5] para la minería de SAF, que encuentra todos los SAF maximales[5] en una colección de grafos etiquetados y dirigidos. Al igual que SUBDUE, la base de este algoritmo es el uso de DEG. Sin embargo, en este trabajo se defiende la idea de que el objetivo del cotejo inexacto de grafos es encontrar eficientemente si dos grafos cotejan de manera inexacta y no encontrar la mejor forma para que estos cotejen. De esta manera no es necesario el procesamiento de todos los posibles caminos de edición, sino que solamente se procesan un número de caminos hasta que se encuentra uno que permita el cotejo de los grafos bajo el umbral de mínima similitud sin importar si es el más cercano a lo óptimo. Esta variante hace que este algoritmo marque una diferencia en tiempos de procesamiento respecto a SUBDUE. Para lograr esto se combina la distacia de inserción y la DEG.

[5] Es maximal ya que no está contenido en ningún otro SAF

En RNGV se utilizan métodos heurísticos que reducen considerablemente el tiempo de procesamiento al calcular el soporte del cotejo inexacto. Estos métodos se conocen como mecanismo de comprobación rápida y mecanismo que limita el tamaño del espacio de búsqueda.

El mecanismo de comprobación rápida (ver definición 30) es utilizado para saber si el subgrafo T coteja de forma inexacta con algún subgrafo del grafo G_i sin tener que encontrar un camino de edición definitivo. Por esta razón se utiliza una combinación de la distancia de inserción y la DEG. Las operaciones de edición de vértices y aristas en este algoritmo presentan un costo fijo de (1) para cada inserción y sustitución, (0) para cada eliminación. De esta manera el costo de la distancia de inserción según la definición 14 sería:

$$I\_dist(G_1, G_2) = Size(G_1) - I(G_1, G_2). \tag{17}$$

Definición 30 (Mecanismo de comprobación). *Un grafo G_1 coteja de manera inexacta con G_2 si después de un camino de edición $CE = \{ce_1, ce_2, \ldots, ce_k\}$ con un costo $c = Cost(CE)$ se obtiene G_3 y dado un umbral τ, se tiene que:*

$$I\_dist(G_1, G_3) + c \leq \tau * Size(G_1). \tag{18}$$

Este mecanismo comprueba si G_1 y G_2 cotejan de forma inexacta según (18), donde G_3 es el grafo resultante cuando se le realizan las operaciones de edición $CE_i = \{e_{i_1}, \ldots, e_{i_k}\}$ al grafo G_2. El camino de edición CE_i forma parte del conjunto de caminos de edición $\Upsilon(G_2, G_3) = \{CE_1, \ldots, CE_n\}$ obtenidos con el uso de DEG en un nivel determinado, en este caso el nivel es k.

El mecanismo que limita el tamaño del espacio de búsqueda permite mantener solamente algunos caminos de edición como candidatos para la siguiente ronda de búsqueda. De esta manera no se realizan las extensiones de todos los caminos sino solamente de los que potencialmente pudieran dar la solución.

Definición 31 (Mecanismo de límite de búsqueda). *Dados los subgrafos G_1, G_2 y un factor del límite de búsqueda $f|(0 < f \leq 1)$, se obtiene G_3 mediante un camino de edición $CE = \{e_1, \ldots, e_k\}$ con un costo $c = Cost(CE)$ y se determina que G_3 y CE se mantienen para la próxima ronda si y sólo si, se cumple que:*

$$I\_dist(G_1, G_3) + c \leq f * Size(G_1). \tag{19}$$

Este mecanismo desecha del conjunto de caminos los $CE_i \in \Upsilon(G_2, G_3)$ que no cumplen con (19), de esta manera se reduce la cantidad de caminos de edición de $\Upsilon(G_2, G_3)$ obtenidos con el uso de DEG en un nivel determinado, en este caso el nivel es k. Esta reducción permite que pasen a la obtención del siguiente nivel solamente los CE_i que pudiesen contribuir a la obtención del resultado esperado.

Algoritmo 10: $rngv\_Cmp(G_1, G_2, \tau, f, CE, \Upsilon)$

Input: G_1 : Un subgrafo, G_2 : Un grafo, τ : Umbral de tolerancia del cotejo, f : Factor del límite de búsqueda, CE : Un camino de edición.
Output: Υ : Conjunto de caminos de edición, Verdadero si coteja de forma inexacta, Falso en caso contrario.

```
1 Obtener G3 al aplicarle CE a G2;
2 c ← Cost (CE, 1, 1, 0) ; result ← Falso;
3 if I_dist(G1, G3) + c ≤ τ * Size(G1) then result ← Verdadero;
4 else if I_dist(G1, G3) + c ≤ f * Size(G1) then Υ ← Υ ∪ {CE};
5 return result;
```

El pseudo-código mostrado en el algoritmo 10 realiza la comprobación del candidato con el uso de (18) en la línea 3 y retorna verdadero en caso satisfactorio, en caso contrario, mantiene el camino de edición si cumple con (19) (ver líneas 4) y retorna falso. Estas comprobaciones se realiza después de haber obtenido G_3 como resultado de G_2 al aplicarle un camino de edición CE (ver línea 1).

El algoritmo RNGV 11 es el punto de partida para dar comienzo a la búsqueda de SAF. Este algoritmo parte de todas las subestructuras compuestas por las aristas que tengan al menos $(n*\delta)$ ocurrencias en la colección de grafos D (ver línea 1). En este algoritmo las listas de identificadores se crean en forma de matriz indizada por los vértices y aristas en las filas y por los identificadores de los grafos $G_i \in D$ en las columnas. Cuando la celda $m_{i,j} = 1$ indica que la arista o vértice i tiene una aparición en el grafo j de la colección, en caso contrario $m_{i,j} = 0$. Se realiza el llamado al procedimiento encargado de extender de forma recursiva los SAF (ver línea 3). Finalmente, como resultado se obtienen en F todos los SAF maximales en D.

Algoritmo 11: $RNGV(D,\delta,\tau,f)$

Input: D : Colección de grafos, δ : Umbral de soporte, τ : Umbral de mínima similitud, f : Factor del límite de búsqueda.
Output: F : Conjunto de SAF maximales.

1 $C \leftarrow \{$Aristas frecuentes en $D\}$;
2 Inicializar para cada subestructura de $T \in C$ la lista de identificadores $LI(T,D)$;
3 **foreach** $T \in C$ **do** $rngv\_DExten(T,D,\delta,\tau,f,F)$;
4 **return** F;

En el algoritmo 12 se muestran los detalles del algoritmo encargado de realizar el crecimiento de patrones (ver analogía con el algoritmo 3). En caso de que exista al menos una extensión de T que resulte frecuente, entonces se marcará la existencia de una extensión satisfactoria de T y se continúa con la extensión del SAF actual. Finalmente sólo se adicionan a F los SAF que no posean ninguna extensión frecuente (ver líneas 6).

Algoritmo 12: $rngv\_DExten(T,D,\delta,\tau,f,F)$

Input: T : Un subgrafo frecuente, D : Colección de grafos, δ : Umbral de soporte, τ : Umbral de mínima similitud, f : Factor del límite de búsqueda.
Output: F : Conjunto de SAF maximales.

1 **foreach** $G_i \in D$ **do** $rngv\_BCandidatos\_LI(T,G_i,\tau,f,C)$;
2 **foreach** $G \in C$ **do**
3 **if** `supApp` $(G,D) \geq (n*\delta)$ *AND* $G \notin F$ **then**
4 Marcar como extensión satisfactoria;
5 $rngv\_DExten(G,D,\delta,\tau,f,F)$;
6 **if** La extensión no fue satisfactoria **then** $F \leftarrow F \cup \{T\}$;

El algoritmo 13 se encarga de generar los candidatos hijos de un SAF con el uso de las listas de identificadores (ver analogía con el algoritmo 1). Mediante el llamado al procedimiento $CotejoRNGV$ en la línea 4 se comprueba si el candidato se almacena en C para verificar su soporte.

Algoritmo 13: $rngv\_BCandidatos\_LI(T,G_i,\tau,f,C)$

Input: $T=(V,E)$: Un grafo candidato, G_i : Un grafo, τ : Umbral de mínima similitud, f : Factor del límite de búsqueda.
Output: C : Los hijos de T que ocurren en G_i.

1 $O(T,G_i) \leftarrow \{$Conjunto de ocurrencias de T en $G_i\}$;
2 **foreach** $(u,v) \in G_i$, *donde* $(u,v) \in N(o_j)$, $o_j \in O(T,G_i)$ **do**
3 **foreach** $w \in V'$, *donde* $(u,w) \notin E$ *y* V' *es el conjunto de vértices con etiquetas diferentes de* D **do**
4 **if** `CotejoRNGV` $(T \diamond (u,w), G_i, \tau, f) = Verdadero$ **then**
5 $C \leftarrow C \cup \{T \diamond (u,w)\}$;
6 $LI(T \diamond (u,w),D) \leftarrow LI(T \diamond (u,w),D) \cup \{i\}$;

El algoritmo 14 se encarga de comprobar si un subgrafo candidato G_1 coteja de forma inexacta con algún subgrafo de G_2. Primero se comprueba si cotejan mediante la distancia de inserción de G_1 a G_2 (ver

líneas 1-2). En caso contrario se le comienzan a aplicar operaciones de edición a G_2 hasta que el llamado al procedimiento *rngv_Cmp* retorne verdadero o mientras que el conjunto de caminos de edición obtenido por DEG bajo el mecanismo de límite de búsqueda Υ no esté vacío (ver líneas 3-19). En las líneas 4-9 se generan los caminos de edición del nivel 1 y en las líneas 11-19 se generan para los niveles superiores a 1. Este algoritmo finaliza la búsqueda de caminos de edición cuando se obtiene un candidato que coteja de forma inexacta con G_1 o cuando no existan más caminos de edición candidatos en Υ.

Algoritmo 14: $CotejoRNGV(G_1, G_2, \tau, f)$

Input: $G_1 = (V_1, E_1)$: Un subgrafo, $G_2 = (V_2, E_2)$: Un grafo, τ : Umbral de mínima similitud, f : factor del límite de búsqueda.
Output: Verdadero si coteja de forma inexacta, Falso en caso contrario.

1 $result \leftarrow Falso$;
2 **if** $I\_dist(G_1, G_2) \leq \tau * Size(G_1)$ **then** $result \leftarrow Verdadero$;
3 **else**
4 **foreach** $(u,v)_{1_i} \in E_1$ **do**
5 $CE \leftarrow \{(u,v)_{2_1} \rightarrow (u,v)_{1_i} | (u,v)_{2_1} \in E_2, (u,v)_{2_1} \neq (u,v)_{1_i}\}$; //Operación de sustitución
6 **if** $rngv\_Cmp(G_1, G_2, \tau, f, CE, \Upsilon) = Verdadero$ **then** $result \leftarrow Verdadero$; Finalizar el foreach;
7 **if** $result = Falso$ **then**
8 $CE \leftarrow \{(u,v)_{2_1} \rightarrow \varepsilon | (u,v)_{2_1} \in E_2\}$; //Operación de eliminación
9 **if** $rngv\_Cmp(G_1, G_2, \tau, f, CE, \Upsilon) = Verdadero$ **then** $result \leftarrow Verdadero$;
10 **if** $result = Falso\ AND\ \Upsilon \neq \emptyset$ **then**
11 Extraer el primer elemento de Υ; Guardar el elemento extraido en T y en CE;
12 **while** $k < |E_2|$ *tal que* $CE = \{(u,v)_{2_1} \rightarrow (u,v)_{1_{i_1}}, \ldots, (u,v)_{2_k} \rightarrow v_{1_{i_k}}\}$ **do**
13 **foreach** $(u,v)_{1_i} \in E_1 \setminus \{(u,v)_{1_{i_1}}, \ldots, (u,v)_{1_{i_k}}\}$ **do**
14 $CE \leftarrow T \cup \{(u,v)_{2_{k+1}} \rightarrow (u,v)_{1_i} | (u,v)_{2_{k+1}} \in E_2, (u,v)_{2_{k+1}} \neq (u,v)_{1_i}\}$;
15 **if** $rngv\_Cmp(G_1, G_2, \tau, f, CE, \Upsilon) = Verdadero$ **then** $result \leftarrow Verdadero$; Finalizar los dos ciclos (foreach y while);
16 $CE \leftarrow T \cup \{(u,v)_{2_1} \rightarrow \varepsilon | (u,v)_{2_1} \in E_2\}$; //Operación de eliminación
17 **if** $rngv\_Cmp(G_1, G_2, \tau, f, CE, \Upsilon) = Verdadero$ **then** $result \leftarrow Verdadero$; Finalizar el while;
18 **else if** $\Upsilon = \emptyset$ **then** Finalizar el while;
19 Extraer el primer elemento de Υ; Guarda el elemento extraido en T y en CE;
20 **return** *result*;

En RNGV se obtienen los caminos de edición *CE* que permiten identificar si un grafo coteja de forma inexacta con otro. De esta forma identifican todos los subgrafos candidatos en el recorrido del espacio de búsqueda. Sin embargo, no utilizan la información que brindan los SAF, como *CE* que lo identificó como candidato en su momento, para la identificación de sus hijos. Siempre se busca el *CE* que le pertenece al subgrafo partiendo de $\emptyset$. De esta forma se generan una y otra vez los mismos caminos de edición y se desecha una vía rápida para identificar el cotejo entre grafos ya que no necesitan el mejor cotejo como SUBDUE.

3.1.3 Discusión

En los algoritmos basados en DEG se detectaron los siguientes problemas:

1. Para la obtención de los caminos de edición de todos los subgrafos candidatos no se tienen en cuenta los caminos identificados para los subgrafos que les dieron origen y son confeccionados comenzando desde $\emptyset$, lo cual afecta la eficiencia de estos algoritmos.
2. SUBDUE presenta alta complejidad computacional que atenta contra su eficiencia.
3. En SUBDUE el usuario define el tamaño máximo y la cantidad máxima de los subgrafos a obtener provocando pérdida de la información de posibles SAF resultantes.

4. En SUBDUE se desechan candidatos con un número considerable de ocurrencias para mantener subestructuras de mayor tamaño que no necesariamente son más frecuentes que los anteriores, provocando pérdida de información.

3.2 Algoritmos basados en β arista sub-isomorfismo

Como se mencionó en la sección 2.1.2, esta técnica de cotejo calcula la similitud entre grafos manteniendo los vértices y permite una cantidad máxima (β) de diferencias entre las aristas (ver definición 15). Estas diferencias son solamente ante la ausencia de aristas, lo que hace que el cotejo inexacto sea parcial. En la figura 14 se muestra un ejemplo de β arista isomorfismo con $\beta = 2$, observe que el grafo T es no conexo.

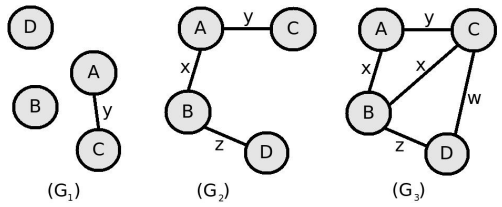

Fig. 14. Ejemplo con $\beta = 2$ donde el grafo $T \approx_\beta G$ y $G \approx_\beta Q$, ya que en ambos casos se tienen a lo sumo β aristas diferentes, cuyas etiquetas son igual a $\emptyset$ en el grafo T en el primer caso y en el segundo caso en el grafo G.

En la literatura se reportó un algoritmo conocido como Monkey que utiliza β arista isomorfismo como función de similitud entre grafos. Este fue presentado por S. Zhang *et al.* [24] para la minería de SAF en una colección de grafos D etiquetados y no dirigidos. Este algoritmo transforma los grafos de la colección en completos creando aristas virtuales con la etiqueta $\emptyset$ entre los vértices que no estén unidos mediante alguna arista. De esta forma simplemente se sustituyen las aristas que poseen la etiqueta $\emptyset$ por las que contienen valor en la etiqueta.

En Monkey se determina si el subgrafo T es aproximado frecuente según la definición 32, con el uso del conjunto de grafos $G_i \in D$ de los cuales T es β arista sub-isomorfo, denotado por $sup_\beta(D,T,\beta) = \{G_i \in D | T \subseteq_\beta G_i\}$. Sin embargo, para que T se incluya en los resultados no debe tener ninguna arista que aparezca menos de $(\delta - \alpha)$ veces en $sup_\beta(D,T,\beta)$, donde δ es el umbral de mínima frecuencia y α es un número entero definido por el usuario.

Definición 32 (β arista δ frecuente). *Dada la colección de grafos D, un umbral de mínima frecuencia* δ*, un número entero* α *y un umbral de máximas diferencias entre aristas* β*, se dice que el grafo* $T = (V,E)$ *es* β *arista* δ *frecuente si cumple con las siguientes características.*

- $|sup_\beta(D,T,\beta)| \geq \delta$*, y*
- *Por cada arista* $(u,v) \in E, |S| \geq \delta - \alpha$*, donde* $S \subseteq sup_\beta(D,T,\beta)$ *y cada grafo* $G_i \in S$ *cumple con la siguiente condición.* G_i *contiene al menos un subgrafo* $Q = (V',E'), Q \approx_\beta T$ *y* $|O((u,v),Q))| = |O((u,v),T)|$.

En el algoritmo 15 se muestran los detalles del procedimiento encargado de generar los candidatos hijos de un SAF T en un grafo $G_i \in D$ (ver analogía con el algoritmo 1). El cálculo de la similitud entre los subgrafos candidatos y sus ocurrencias aproximadas se realiza mediante el uso de β arista isomorfismo. Los candidatos son almacenados en dos tablas hash, una para los que son árboles denotada por HT y otra para los que son grafos denotada por HNt. Para HT se utilizan los códigos DFS como llaves mientras que en HNt se utilizan los códigos CAM. Estas formas canónicas son obtenidas mediante costosas pruebas de formas canónicas y se le realizan a todos los candidatos generados por el crecimiento de patrones.

El algoritmo 16 muestra el pseudo-código de Monkey, que comienza a partir de los subgrafos aproximados frecuentes compuestos por una arista (ver línea 1). Para cada arista (u,v) frecuente, el procedimiento *monkey_DExten* se encarga de encontrar todos los SAF que la contengan. Luego, la arista (u,v) se elimina de cada uno de los grafos de la colección; por tanto esta arista no será utilizada en las iteraciones siguientes.

Algoritmo 15: $monkey\_BCandidatos\_LI(T, G_i, \beta, C)$

Input: $T = (V,E)$: Un grafo candidato, G_i : Un grafo, β : Cantidad máxima de aristas diferentes.
Output: C : Los hijos de T que ocurren en G_i.

1 $O(T, G_i) \leftarrow$ {Conjunto de ocurrencias de T en G_i};
2 **foreach** $(u,v) \in G_i$, *donde* $(u,v) \in N(o_j)$, $o_j \in O(T, G_i)$ **do**
3 **foreach** $w \in V'$, *donde* $(u,w) \notin E$ *y* V' *es el conjunto de todos los vértices con etiquetas diferentes de* D **do**
4 **if** $o_j \diamond (u,v) \approx_\beta T \diamond (u,w)$ **then**
5 **if** $T \diamond (u,w)$ es un árbol **then** $HT \leftarrow HT \cup \{T \diamond (u,w)\}$; // Función hash
6 **else** $HNt \leftarrow HNt \cup \{T \diamond (u,w)\}$; // Función hash para los candidatos que son grafos
7 $C \leftarrow C \cup \{T \diamond (u,w)\}$;
8 $LI(T \diamond (u,w), D) \leftarrow LI(T \diamond (u,w), D) \cup \{i\}$;

Algoritmo 16: $Monkey(D, \beta, \delta, \alpha)$

Input: D : Colección de grafos, β : Cantidad máxima de aristas diferentes, δ : Umbral mínimo de soporte, α : Un número entero factor del umbral de soporte de cada arista.
Output: F : Conjunto de subgrafos β arista δ frecuentes en D.

1 $F \leftarrow C \leftarrow$ {Aristas aproximadas frecuentes en D};
2 Eliminar de D todas las aristas que no son frecuentes;
3 **foreach** $T \in C$ **do**
4 $monkey\_DExten(T, D, \beta, \delta, \alpha, F)$;
5 Eliminar las aristas iguales a T de cada uno de los grafos de D;
6 **if** $|D| < \delta$ **then** Finalizar el foreach;
7 **return** F;

El algoritmo 17 muestra los detalles del procedimiento encargado de realizar el crecimiento de patrones (ver analogía con el algoritmo 3). Mediante el llamado al procedimiento *monkey_BCandidatos_LI* por cada grafo de la colección se genera el conjunto de subgrafos candidatos C.

Algoritmo 17: $monkey\_DExten(T, D, \beta, \delta, \alpha, F)$

Input: D : Colección de grafos, T : Un grafo candidato, $O(T,D)$: Conjunto de ocurrencias de T en D, β : Cantidad máxima de aristas diferentes, δ : Umbral mínimo de soporte, α : Número entero.
Output: F : Conjunto de subgrafos β arista δ frecuentes en D.

1 **foreach** $G_i \in D$ **do** $monkey\_BCandidatos\_LI(T, G_i, \beta, C)$;
2 Eliminar de C los candidatos que no son β arista δ frecuentes en D;
3 **foreach** $G \in C$ **do**
4 **if** $G \notin F$ **then**
5 $F \leftarrow F \cup \{G\}$;
6 $monkey\_DExten(G, D, \delta, \tau, F)$;

Los algoritmos en general que utilicen β arista sub-isomorfismo solamente realizarán un cotejo inexacto donde esté presente la adición de aristas sobre la colección de grafos. En el caso de Monkey, se detectaron además los siguientes problemas:

1. En Monkey se realizan las costosas pruebas de formas canónicas a todos los candidatos generados por el crecimiento de patrones, lo que reduce la eficiencia de dicho algoritmo.

2. En Monkey se realizan las costosas pruebas de sub-isomorfismo debido al uso de listas de identificadores para indizar la colección, afectando la eficiencia de dicho algoritmo.

3.3 Algoritmos basados en homeomorfismo con vértice/arista disjuntas

El algoritmo conocido como CSMiner fue presentado por Y. Xiao *et al.* para la minería basado en la conservación de las topologías de los grafos [6],[16]. La función de similitud utilizada en este algoritmo para la minería de SAF se conoce como *homeomorfismo con vértice/arista disjuntas* (ver definición 22). Ese algoritmo se basa en el paradigma de los algoritmos Apriori y de esta manera obtiene todos los subgrafos que son isomorfos a una *(l,h)-Topología menor* (ver definición 33) de cada grafo $G_i \in D \subset \Omega$. La definición del problema de la minería de subgrafos que conservan la topología se identifica con la minería de subgrafos frecuentes tradicional. Por lo que se ha considerado este algoritmo entre los algoritmos desarrollados para la minería de SAF. Sin embargo, CSMiner tiene una característica que lo distingue de los demás algoritmos para este tipo de minería y es que la frecuencia de los subgrafos candidatos tiene que es de 100 % para ser identificados como frecuentes.

Definición 33 ((l,h)-Topología menor). *Sean $G,Y,X \in \Omega$, $G \subseteq Y$, X una topología menor de Y (ver figura 11) y G es una subdivisión de X, si G se obtiene al reemplazar todas las aristas de X por caminos independientes con una longitud de l a h, entonces G es* (l,h)-subdivisión *de X y T es una (l,h)-Topología menor de Y.*

En otras palabras, determinar si un grafo T es una (l,h)-Topología menor de G es equivalente a encontrar un par de funciones de mapeo (f_n, f_e) entre estos grafos, donde f_n mapea los vértices de T con los de G que tengan las mismas etiquetas y f_e mapea las aristas de T con los correspondientes caminos de G. La solución de esta determinación se describe como la tupla (NM, EPM), donde NM es el conjunto de pares de vértices que cotejan y EPM es el conjunto de pares *arista-camino* que cotejan.

Algoritmo 18: $NodeMappingSearch(state, M, R, E, V)$

Input: M : Matriz del mapeo entre los vértices, R : Matriz de los caminos independientes, V Conjunto de vértices, E : Conjuntos de aristas.
Output: $state$: Estado del cotejo, VERDADERO si se encontró un mapeo completo, FALSO en caso contrario.

```
1  if |NM_state| = |V| then result ← VERDADERO;
2  else if Existe al menos una fila de M que todos los valores sean 0 then result ← FALSO;
3  else
4      found ← FALSO;
5      while found = FALSO AND Existan pares de vértices válidos por mapear do
6          m ← El siguiente par de vértices válidos (v_i, v_j);
7          state_2 ← state;
8          NM_state ← NM_state ∪ {m}; Actualizar M y R;
9          E' = {(v_i, u) | u ∈ NM_state ∩ V, u ∈ N(v_i)};
10         if E' = ∅ then found ← NodeMappingSearch(state, M, R, E, V);
11         else found ← EdgePathMappingSearch(state, M, R, E, E');
12         if found = VERDADERO then state ← state_2;
13 return found;
```

El algoritmo 19 muestra el pseudo-código encargado de verificar si entre un subgrafo candidato T y $G \in D$ existe una relación de (l,h)-Topología menor. Luego de inicializar las estructuras de datos básicas para este algoritmo en las líneas 1-2, comienza el proceso de cotejo de vértices partiendo de un estado inicial (vacío) mediante el llamado al procedimiento $NodeMappingSearch$ (ver línea 3). Cuando un completo mapeo de vértices haya sido obtenido comienza el proceso de cotejo del espacio de búsqueda

de aristas-caminos mediante el llamado al procedimiento *EdgePathMappingSearch* (ver líneas 4). Finalmente, retorna verdadero si T es una (l,h)-Topología menor de G y falso en caso contrario. Estos últimos son representados mediante los algoritmos 18 y 20.

Algoritmo 19: $ndSHD(T,G,l,h)$

Input: $T=(V,E,L,l), G$: Dos grafos etiquetados, l : Longitud mínima de un camino, h : Longitud máxima de un camino.

Output: VERDADERO si T es una (l,h)-Topología menor de G, FALSO en caso contrario.

```
1 Construir una matriz M que represente el mapeo entre los vértices de T y G;
2 Construir una matriz R que represente los caminos independientes;
3 result ← NodeMappingSearch(state, M, R, V);
4 if state es un cotejo válido then result ← EdgePathMappingSearch(state, M, R, E, E);
5 return result;
```

Como se pudo observar en el procedimiento anterior (*ndSHD*), la eficiencia del algoritmo CSMiner se ve afectada ya que se recorre el espacio de búsqueda en dos niveles por cada candidato T. Un primer nivel para realizar el cotejo de los pares de vértices y un segundo nivel para realizar el cotejo de los pares arista-camino.

Algoritmo 20: $EdgePathMappingSearch(state, M, R, E_1, E_2)$

Input: M : Matriz del mapeo entre los vértices, R : Matriz de los caminos independientes, E_1, E_2 Dos conjuntos de aristas.

Output: $state$: Estado del cotejo, VERDADERO si se encontró un mapeo completo, FALSO en caso contrario.

```
1  if |EPM_state| = |E1| then result ← VERDADERO;
2  else if Si no existe un camino entre cualquier par de vértices rama en R then result ← FALSO;
3  else
4      found ← FALSO;
5      while found = FALSO AND Existan pares arista-camino válidos por mapear do
6          m ← El siguiente par arista-camino válido;
7          state2 ← state;
8          EPM_state ← EPM_state ∪ {m}; Actualizar M y R;
9          found ← EdgePathMappingSearch(state, M, R, E1, E2);
10         if found = VERDADERO then found ← NodeMappingSearch(state, M, R, V, E1);
11         else state ← state2;
12 return found;
```

En CSMiner se emplean tres operadores de fusión para generar los subgrafos candidatos: un operador de fusión hacia delante y dos hacia atrás. Con el objetivo de identificar cuál operador utilizar se organizan los vértices de los subgrafos de forma ascendente según sus índices y los subgrafos de la misma forma según el orden lexicográfico basado en los índices de los vértices. A cada subgrafo patrón $T \in F^k$ se le asigna un identificador *PID* y una secuencia de bit $b = b_1, \ldots, b_{|F^1|}$ donde $b_i = 1$ si la i-ésima arista de F^1 ocurre en T, $b_i = 0$ en caso contrario. De esta forma se asocian para cada subgrafo patrón un par ordenado (PID_1, PID_2) que indican los patrones que le dieron origen. Esto se realiza con el objetivo de identificar si dos patrones son fusionables hacia delante o hacia atrás mediante la definición 34 dada a continuación.

Definición 34 (Operadores de fusión). *Dado un conjunto de SAF de tamaño k ordenados de forma ascendente $C^k \subset \Omega$, para dos grafos $T_1, T_2 \in C^k$ se define la fusión de la siguiente manera:*

- *Fusión hacia delante, denotada por $T_1 \rhd T_2$, si $T_1.PID_2 = T_2.PID_1$.*
- *Fusión hacia atrás 1, denotada por $T_1 \lhd_1 T_2$, si $T_1.PID_1 = T_2.PID_1$.*
- *Fusión hacia atrás 2, denotada por $T_1 \lhd_2 T_2$, si $T_1.PID_2 = T_2.PID_2$.*

El algoritmo CSMiner comienza la minería a partir de todas las aristas que ocurran en todos los $G_i \in D$, luego realiza el llamado al procedimiento 21 para finalmente obtener como resultado el conjunto de todos los SAF en la colección D. El algoritmo 21 es el encargado de obtener los candidatos a partir de los SAF de mayor tamaño como algoritmo Apriori en cuestión (ver analogía con el algoritmo 5). Las pruebas de fusión se realizan solamente comparando los identificadores del par de patrones que dan origen al candidato actual (ver líneas 3-5). Para las pruebas de duplicidad $P \notin C$ (ver línea 6) se realizan mediante un AND(&) lógico entre las secuencias de bits de P y $T_i \in C$, i.e. si $P.b \& T_i.b = P.b$ entonces P es duplicado de T_i. En este algoritmo se utiliza el conjunto de subgrafos infrecuentes para filtrar el conjunto de subgrafos candidatos. Este procedimiento de poda se describe en la línea 7.

Algoritmo 21: $Apriori\_CSMiner(S, D, l, h, F)$

Input: S : Subconjunto de SAF, D : Una colección de grafos, l : Longitud mínima de un camino, h : Longitud máxima de un camino.
Output: F : Conjunto de todos los SAF.

```
1  foreach T_i ∈ S do
2      foreach T_j ∈ S do
3          if T_1.PID_2 = T_2.PID_1 then P ← T_1 ▷ T_2;
4          else if T_1.PID_1 = T_2.PID_1 then P ← T_1 ◁_1 T_2;
5          else if T_1.PID_2 = T_2.PID_2 then P ← T_1 ◁_2 T_2;
6          if P ≠ ∅ AND P ∉ C then
7              if Ningún subgrafo de la lista de infrecuentes es subpatrón de P then C ← C ∪ {P};
8  foreach T ∈ C do
9      foreach G_i ∈ D do if ndSHD(T, G_i, l, h) = FALSE then Se elimina T de C;
10 if C ≠ ∅ then F ← F ∪ C; Apriori_CSMiner(C, D, δ, τ, F);
```

Al algoritmo CSMiner se le detectaros los siguientes problemas:

1. Se recorre el espacio de búsqueda en dos ocaciones por cada candidato, una para el cotejo entre los vértices del candidato T y los del grafo de la colección $G_i \in D$ y otra para el cotejo entre las aristas de T y los caminos de G_i. Esto influye de forma negativa en la eficiencia de este algoritmo.
2. Se realizan las costosas pruebas de sub-isomorfismo ya que no se almacenan las ocurrencias de los candidatos en la colección, sino que solamente se mantienen los identificadores de los grafos donde ocurren, afectando la eficiencia de dicho algoritmo.

3.4 Algoritmos basados en isomorfismo con el uso de heurísticas para el cálculo de la frecuencia aproximada

En la literatura se reportó un algoritmo conocido como MUSE (de sus siglas en inglés, *Mining Uncertain Subgraph pattErns*) presentado por Z. Zuo *et al.* [4],[33]. Este algoritmo fue desarrollado para la minería de SAF basado en isomorfismo como función de similitud entre grafos. La inexactitud en este algoritmo se obtiene mediante el uso de una heurística presentada con el nombre de *soporte esperado* (ver definición 35) que determina si un subgrafo candidato puede ser frecuente.

MUSE trabaja sobre colecciones de grafos conexos, no dirigidos y etiquetados, con un valor agregado a las aristas que representa la probabilidad de su existencia definido por el intervalo $(0,1]$. Estos grafos se conocen como *grafos inciertos* y a diferencia de los grafos presentados en la sección 2, un grafo incierto G tiene asociado un conjunto de grafos etiquetados conocidos como *grafos implicados* denotados por $I(G)$. Estos grafos implicados son grafos con una probabilidad muy cercana a 1 de que sus aristas existan. A cada grafo incierto $G = (V, E, L, l, P)$ le corresponde una cantidad de grafos implicados en fun-

ción de su cantidad de aristas, es decir $|I(G)| = 2^{|E|}$. Esto implica que una colección de grafos inciertos $D = \{\{G_1, \ldots, G_n\} | G_i = (V_i, E_i, L_i, l_i, P_i)\}$ tiene una colección de grafos implicados denotada por $I(D)$ donde $|I(D)| = \prod_{i=1}^{n} 2^{|E_i|}$. Esto indica que se tendrán dos colecciones para ser procesadas partiendo de la colección de grafos inciertos de entrada, donde el tamaño de la colección $I(D)$ crece exponencialmente por la cantidad de aristas existentes en cada grafo de D. Nótese que la complejidad computacional de MUSE aumentará considerablemente mientras mayor sea la colección D o más densos sean sus grafos.

La probabilidad de que un grafo incierto $G = \{V, E, L, l, P\}$ implique a un grafo $I = \{V^I, E^I, L^I, l^I\}$ se obtiene de la siguiente manera:

$$P(G \Rightarrow I) = \prod_{(u,v) \in E^I} P((u,v)) \prod_{(u,v)' \in E \setminus E^I} (1 - P((u,v)')). \tag{20}$$

En MUSE se dice que T es un *subgrafo patrón* en D si es isomorfo a al menos un grafo implicado I de la colección de grafos implicados $I(D)$. Para la identificación de este tipo de subgrafo se debe realizar la minería de SF sobre $I(D)$. Esta identificación de subgrafo patrón es utilizada en (21) para calcular la probabilidad de que T ocurra en $G_i \in D$, denotado por $T \sqsubseteq_U G_i$, si T es un subgrafo patrón en G_i, donde $\psi(I,T) = 1$ si T es patrón y $\psi(I,T) = 0$ en caso contrario.

$$P(T \sqsubseteq_U G_i) = \sum_{I \in I(G_i)} P(T \Rightarrow I) \psi(I,T). \tag{21}$$

Algoritmo 22: $approxExpSup(T, D, \delta, \varepsilon, \alpha, X)$

Input: T : Un subgrafo patrón, D : Colección de grafos inciertos, δ : Umbral de mínima frecuencia, ε : Umbral de tolerancia de error relativo, α : Número real, $X = \{X_1, \ldots, X_n\}$: Conjunto de ocurrencias exactas de T en D, X_i : Conjunto de ocurrencias exactas de T en $G_i \in D$.

Output: l, u : cota inferior y superior respectivamente de la aproximación del soporte esperado de T en D.

foreach $G_i \in D$ **do**
  if $\frac{2^{|X_i|-5}}{|X_i|} \geq \frac{ln(2/\delta)}{(\varepsilon * \delta)^2}$ **then**
    Construir la fórmula DNF $F' = C_1 \vee \ldots \vee C_n$;
    $\varepsilon' \leftarrow \varepsilon \frac{\delta}{2}$;
    $N \leftarrow \frac{4n}{\varepsilon'^2} ln \frac{2}{\alpha}$;
    $Z \leftarrow Pr(C_1) + \ldots + Pr(C_n)$;
    for 1 *to* N **do**
      $k \in \{1, \ldots, n\}$; Escoger de forma aleatoria una variable π que satisfaga C_k;
      $Y \leftarrow Y + Pr(\pi)$;
      if π no satisface a C_j para todo $1 \leq j \leq k$ **then** $X \leftarrow X + Pr(\pi)$;
    $l' \leftarrow (XZ/Y - \varepsilon')$; $u' \leftarrow (XZ/Y + \varepsilon')$;
  else
    Construir la fórmula DNF F' basado en X_i;
    $l' \leftarrow u' \leftarrow Pr(F')$;
  $l \leftarrow l + l'$; $u \leftarrow u + u'$;
return $[l/|D|, u/|D|]$;

Definición 35 (Soporte esperado). *Sea $T \in \Omega$ un subgrafo patrón en D, $D \subset \Omega$, se calcula el soporte esperado de T en D de la siguiente forma:*

$$esup(T,D) = \frac{1}{|D|} \sum_{i=1}^{|D|} P(T \sqsubseteq_U G_i). \tag{22}$$

Se dice que un subgrafo candidato T es SAF en una colección de grafos inciertos D si su soporte esperado $esup(T,D) \geq \delta$, donde δ es el umbral de mínima frecuencia. De esta forma se realiza la minería de SF sobre los subgrafos patrones en D. Para obtener el conjunto de SAF se incorpora un umbral de tolerancia de error relativo $\varepsilon \in [0,1]$, donde T es SAF si $esup(T,D) \geq (1-\varepsilon)\delta$.

El algoritmo 22 muestra el pseudo-código que calcula el soporte esperado de un subgrafo patrón T. Lo primero es determinar si se buscan las ocurrencias exactas de T en $G_i \in D$ o se calculan los intervalos aproximados de la probabilidad de que exista una ocurrencia $P(T \sqsubseteq_U G_i)$. $[l',u']$ son los intervalos aproximados resultantes de $P(T \sqsubseteq_U G_i)$. Finalmente, se retornan las cotas inferiores y superiores del soporte esperado de T en D para una posterior comprobación de la frecuencia aproximada de T.

En MUSE se presenta una forma de calcular $P(T \sqsubseteq_U G_i)$ basado en las ocurrencias de T en G_i. La técnica fundamental de este enfoque es transformar el problema de calcular $P(T \sqsubseteq_U G_i)$ en el problema del conteo DNF (de sus siglas en inglés *Disjuntive Normal Form*) [47]. Este último problema del conteo se construye mediante tres pasos: (1) Por cada arista $(u,v)_j$ en una ocurrencia se crea una variable booleana x_j. La probabilidad $Pr(x_j)$ que se le asigne verdadero es igual a la existencia de la arista $P((u,v)_j)$. (2) Por cada ocurrencia T_i de $T = (V,E,L,l)$ se construye una cláusula $C_i = x_{i_1} \wedge \ldots \wedge x_{i_{|E|}}$, donde x_{i_j} es una variable booleana creada por la arista $(u,v)_{i_j}$ de la i-ésima ocurrencia de T en el paso 1. (3) La fórmula F' resultante de DNF es la disjunción de todas las conjunciones de las cláusulas construidas para todas las ocurrencias en el paso 2, i.e. $F' = (x_{1_1} \wedge \ldots \wedge x_{1_{|E|}}) \vee \ldots \vee (x_{n_1} \wedge \ldots \wedge x_{n_{|E|}})$.

El algoritmo 23 muestra el pseudo-código inicial del algoritmo MUSE. Lo primero es encontrar todos los subgrafos patrones de una arista en D. Estos subgrafos son el punto de partida para recorrer los grafos de D mediante la llamada al procedimiento *muse_DExten* para la confección del conjunto de SAF resultantes del minado.

Algoritmo 23: $MUSE(D,\delta,\varepsilon,\alpha)$

Input: D : Colección de grafos inciertos, δ : Umbral de mínima frecuencia, ε : Umbral de tolerancia de error relativo, α : Número real.
Output: F : Conjunto de SAF en D.

```
1 C ← {Todos los subgrafos patrones con una arista};
2 foreach T ∈ C do muse_DExten(T, D, δ, ε, α, F);
3 return F;
```

Algoritmo 24: $muse\_DExten(T,D,\delta,\varepsilon,\alpha,F)$

Input: T : Un subgrafo patrón, D : Colección de grafos inciertos, δ : Umbral de mínima frecuencia, ε : Umbral de tolerancia de error relativo, α : Número real.
Output: F : Conjunto de SAF.

```
1 foreach G_i ∈ D do X ← {Todas las ocurrencias exactas de T en G_i};
2 [l, u] ← approxExpSup(T, D, δ, ε, α, X);
3 if l ≥ (1 − ε)δ AND u ≥ δ then
4 |   F ← F ∪ T;
5 |   C ← {Todas las extensiones de T que son subgrafos patrones en D};
6 foreach T' ∈ C do
7 |   if El prefijo del código DFS de T' es igual a T then muse_DExten(T', D, δ, ε, α, F);
```

En el algoritmo 24 se muestran los detalles del procedimiento encargado de realizar el crecimiento de patrones (ver analogía con el algoritmo 3). En la línea 2 se obtienen los cotas inferior l y superior u de la aproximación del soporte esperado de T en D mediante el llamado al procedimiento *approxExpSup*. En la línea 3 se define como SAF a T si la cota inferior de la aproximación del soporte esperado $l \geq (1-\varepsilon)\delta$ y su cota superior $u \geq \delta$. En caso de que T sea SAF, entonces se agrega en el conjunto de SAF resultantes

F (ver línea 4) y se buscan todas sus extensiones que sean patrones en D (ver línea 5). Esto último implica mantener un conjunto de subgrafos que son patrones en D e identificar cada extensión $T \diamond (u,v)$ en este conjunto. El esquema de codificación DFS de [48] es usado por lo que en las líneas 7 se extienden los subgrafos que el prefijo de su código DFS sea igual al de T. De esta forma se garantiza que solamente se extenderán los patrones por su subgrafo DFS canónico. Sin embargo se realizan las pruebas de formas canónicas para todos los candidatos identificados.

Los problemas detectados en los algoritmo que utilizan isomorfismo como función de similitud en la minería de SAF sobre grafos inciertos son los siguientes:

1. Aumenta considerablemente la complejidad computacional cuando la colección de grafos inciertos de entrada D va aumentando, debido a que el tamaño de la colección de grafos implicados de D aumenta de forma exponencial en términos de la cantidad de aristas existentes en cada grafo de D.
2. En MUSE se realizan las costosas pruebas de formas canónicas para todos los candidatos generados por el crecimiento de patrones, lo que reduce la eficiencia de dicho algoritmo.

3.5 Algoritmos basados en las probabilidades de sustitución

Los algoritmos basados en las probabilidades de sustitución son los primeros, hasta lo que se conoce, que especifican cuáles vértices, aristas o etiquetas pueden sustituir a otra. En los trabajos reportados se defiende la idea de que no siempre un vértice o una arista puede ser sustituida por cualquier otra. Sobre esta base se detectan todos los SAF con la utilización de funciones heurísticas que permiten calcular la similitud entre los subgrafos y sus ocurrencias aproximadas. A continuación se presentan algunos algoritmos reportados que realizan este tipo de minería.

3.5.1 gApprox

C. Chen *et al.* [3] presentan un algoritmo para la minería de SAF en un gran grafo G. Este algoritmo se conoce como gApprox. En este se utiliza el grado de aproximación (ver definición 28) como función de similitud entre grafos etiquetados y no dirigidos. Cada etiqueta o peso de las aristas se interpreta como la distancia de la relación entre dos entidades. Estas entidades se representan mediante los vértices. En gApprox se divide a G en subgrafos $g_i = (V_i, E_i)$ donde exista una ocurrencia aproximada del vértice aproximado frecuente $u \in V_i$ en cada uno de ellos. Sobre estas divisiones se realiza la minería de SAF. El hecho de dividir el grafo de entrada permite adaptar el algoritmo para que trabaje sobre colecciones de grafos. Esto es posible debido a que se puede asumir que la colección $D = G$ y que los subgrafos $g_i \in G$ son los grafos de la colección $G_i = g_i$ tal que $G_i \in D$.

gApprox utiliza un cotejo entre vértices para cualquier tipo de etiquetas que tengan, no ocurriendo de esta forma para el cotejo entre aristas. Esto se debe a que la penalización de las aristas utilizada en el grafo de aproximación (ver definición 27) solo permite la inexactitud cuando las etiquetas son numéricas o tienen definidas las distancias entre ellas.

En gApprox se obtiene el soporte aproximado del subgrafo g por la cantidad máxima de subgrafos que cotejan y no comparten ningún vértice en común. De esta manera se realiza el conteo del soporte excluyendo el solapamiento (ver definición 36) entre ocurrencias al procesar un solo grafo. El cálculo de este soporte varía ligeramente al adaptar este algoritmo para el trabajo sobre colecciones de grafos. Esta variación no afecta la concepción de esta definición, simplemente descarta la identificación del solapamiento y quedaría como la definición 9. Finalmente, un grafo T es SAF en la colección D si $supApp(T,D) \geq \delta$.

Definición 36 (Solapamiento). *Sean $G = (V,E)$ y $G' = (V',E')$ dos grafos, se está en presencia de un solapamiento entre estos si $V \cap V' \neq \emptyset$ o $E \cap E' \neq \emptyset$.*

Para realizar la extensión de los SAF se utiliza la combinación de los procedimientos 25 y 26. En el primero de estos se recorre el espacio de búsqueda en profundidad. En este, lo primero es construir una lista de los vértices vecinos de T en G_i, denotada por V_{exten} y se marcan como visitados (ver líneas 2-4). Luego de ordenar V_{exten} de forma ascendente se realiza el llamado al segundo procedimiento $Horizontal\_Exten$, el cual se encarga de recorrer el espacio de búsqueda en amplitud basado en V_{exten} (ver líneas 5-6). Finalmente se actualiza el conjunto de SAF F con los candidatos que tengan una frecuencia mayor o igual que δ (ver líneas 8).

Algoritmo 25: $Vertical\_Exten(T, D, \Delta, \delta, matchable, F, C)$

Input: T : Un SAF, D : Colección de grafos, Δ : Umbral de máxima disimilitud, δ : Umbral de mínima frecuencia, $matchable$: Listas por cada vértice de los vértices que pueden sustituirlo, C : Conjunto de subgrafos candidatos.
Output: F : Conjunto de SAF.

```
1 foreach G_i ∈ D do
2     foreach (u,v) ∈ G_i, donde (u,v) ∈ N(o_j), o_j ∈ O(T,G_i), v no estén marcados como visitados do
3         V_exten = V_exten ∪ {v};
4         Marcar v como visitado;
5     Ordenar el conjunto V_exten de forma ascendente según sus índices;
6     Horisontal_Exten(T, G_i, Δ, matchable, V_exten, C);
7     Desmarcar todos los vértices de V_exten;
8 foreach T' ∈ C do if supApp(T', D) ≥ δ then F ← F ∪ {T'};
```

Algoritmo 26: $Horizontal\_Exten(T, G, \Delta, matchable, V_{exten}, C)$

Input: T : Un SAF, G : Un grafo etiquetado, Δ : Umbral de máxima disimilitud, $matchable$: Listas por cada vértice de los vértices que pueden sustituirlo, V_{exten} : Conjunto de vértices que son vecinos a T.
Output: C : Conjunto de subgrafos candidatos.

```
1  foreach v ∈ V_exten do
2      if approx(T ◇ v →^m G) ≤ Δ then
3          C ← C ∪ {T ◇ v};
4          if upperBound(EC(T ◇ v, D)) ≥ supApp(T ◇ v, D) then
5              V'_exten = V_exten \ {v};
6              Horizontal_Exten(T ◇ v, G, Δ, matchable, V'_exten, F);
7              Vertical_Exten(T ◇ v, D, Δ, matchable, F);
8      foreach u ∈ matchable(v) do
9          if approx(T ◇ u →^m G) ≤ Δ then
10             C ← C ∪ {T ◇ u};
11             if upperBound(EC(T ◇ u, D)) ≥ supApp(T ◇ u, D) then
12                 V'_exten = V_exten \ {u};
13                 Horizontal_Exten(T ◇ v, G, Δ, matchable, V'_exten, C);
14                 Vertical_Exten(T ◇ v, D, Δ, matchable, C);
```

El algoritmo 26 es el encargado de recorrer el espacio de búsquea en amplitud con el objetivo de identificar los subgrafos candidatos. Cuando un candidato cumple con el umbral de máxima disimilitud respecto a su ocurrencia aproximada se almacena en el conjunto de canidatos y se verifica si es extensible (ver líneas 2-4). Este verificación se realiza mediante el llamado al procedimiento $upperBound$ y la comparación de su resultado con el soporte del candidato. En caso de que este soporte sea mayor o igual que la cota superior de ocurrencias retornada por $upperBound$ se pasa a extender el candidato, no ocurriendo así en caso contrario. En el caso satisfactorio se quita de la lista de vértices V_{exten} el vértice por el cual se realizó la extensión y se continúa el recorrido en amplitud del espacio de búsqueda mediante el llamado recursivo del algoritmo en cuestión (ver líneas 5-6). Luego de terminado el recorrido en amplitud se realiza

la llamada al procedimiento *Vertical_Exten* en la línea 7 para avanzar en profundidad el recorrido. Este proceso se realiza también para las extensiones basadas en la lista *matchable* en las líneas 8-14.

Algoritmo 27: $upperBound(EC(T,D))$

Input: T : Un subgrafo candidato, D : Colección de grafos, $EC(T,D)$: Estructura de correspondencias de T en D.
Output: *supBound* : Un número entero.

1 Se construye una lista de las ocurrencias M_T a partir de $EC(T,D)$;
2 **while** $M_T \neq \emptyset$ **do**
3 Sea v el que aparece en el mayor número de conjuntos de vértices en M_T;
4 $supBound = supBound + 1$;
5 **foreach** $m \in M_T$ **do if** m contiene a v **then** Eliminar m de M_T;

En gApprox se utiliza una función heurística para identificar la factibilidad de realizar la extensión de algún candidato (ver algoritmo 27). Esta función se basa en la cota superior de ocurrencias disjuntas para garantizar que se extiendan solamente los candidatos con alta probabilidad de que sean SAF. El cálculo de la cota superior de ocurrencias disjuntas puede ser explicado mediante el siguiente ejemplo. Asumiendo que cada ocurrencia de un patrón T en D es un conjunto de vértices como: $\{v_1, v_2\}$, $\{v_1, v_3\}$, $\{v_1, v_4\}$ y $\{v_2, v_5\}$, entonces se selecciona v_1 ya que está contenido en 3 ocurrencias, que es la mayoría entre los 5 vértices. Al mantener iterando este proceso, se considera el conjunto que queda después de que se eliminaran todas las ocurrencias que contenían a v_1. Finalmente el número total de ocurrencias disjuntas que pueden ser seleccionadas es a lo sumo $1+1=2$. De esta forma se reduce el número de candidatos innecesarios que se generan al recorrer el espacio de búsqueda.

El algoritmo 28 muestra el pseudo-código punto de partida del algoritmo gApprox. Este comienza identificando los vértices frecuentes en D a partir de los cuales comenzará a recorrer el espacio de búsqueda con la llamada al procedimiento *Vertical_Exten* en la línea 5. Cuando se hayan terminado cada recorrido del espacio de búsqueda se elimina de la colección el vértice a partir del cual se realizó este proceso (ver línea 6). En la línea 7 se desmarcan los vértices que quedaron en la colección. Luego de terminar la detección de todos los posibles SAF se retorna ese conjunto como resultado de este algoritmo (ver línea 8).

Algoritmo 28: $gApprox(D, \Delta, matchable)$

Input: D : Colección de grafos, Δ : Umbral de máxima disimilitud, *matchable* : Listas por cada vértice de los vértices que pueden sustituirlo.
Output: F : Conjunto de SAF.

1 $C \leftarrow$ {Vértices que son SAF en D};
2 Inicializar la lista de correspondencia $EC(T,D)$ para cada $T \in C$;
3 **foreach** $T \in C$ **do**
4 Marcar el vértice T como visitado;
5 $Vertical\_Exten(T, D, \Delta, matchable, F, \emptyset)$;
6 Eliminar el vértice T de la colección;
7 Desmarcar todos los vértices de la colección;
8 **return** F;

En el ejemplo 4 se demuestra que gApprox no identifica todos los posibles SAF en la colección de grafos D ilustrada en la figura 15. Esto influye en la disminusión de la eficiencia de este algoritmo. Esto se debe precisamente al intentar disminuir la generación de duplicados al eliminar de la colección el vértice del cual ya se identificaron todos sus supergrafos.

Ejemplo 4. Dada una colección de grafos etiquetados D (ver figura 15), donde $matchable(A) = \{B\}$ y los vértices B, C no tienen posibles sustituciones y $\Delta = 1$. En la figura 16 se muestran los posibles SAF de la colección D. El conjunto de SAF $\{SAF_4, SAF_5, SAF_6, SAF_7, SAF_8\}$ son identificados partiendo del SAF_1.

De estos, los SAF_7 y SAF_8 tienen el valor del grado de aproximación igual que Δ ya que adicionó una arista en la ocurrencias de G_1 y G_2.

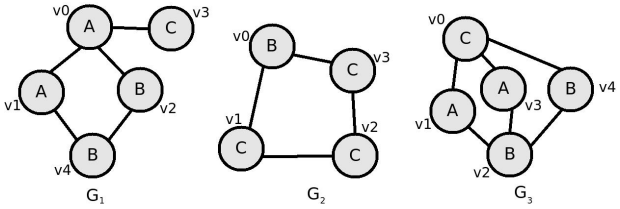

Fig. 15. Una colección de grafos etiquetados $D = \{G_1, G_2, G_3\}$.

Luego de eliminar todas las ocurrencias del SAF_1 de la colección D se pasa a la detección de los SAF a partir del SAF_2. Este SAF contiene algunas ocurrencias que ya no existen en D debido a la eliminación de las ocurrencias del SAF_1 como: v_0 y v_1 de G_1, v_1 y v_3 de G_3. Esto no permite la detección de los SAF $\{SAF_{12}, SAF_{13}\}$ que se obtienen utilizando estas ocurrencias utilizando la lista $matchable(A)$. Por lo tanto, se obtienen solamente el conjunto de SAF $\{SAF_9, SAF_{10}, SAF_{11}\}$.

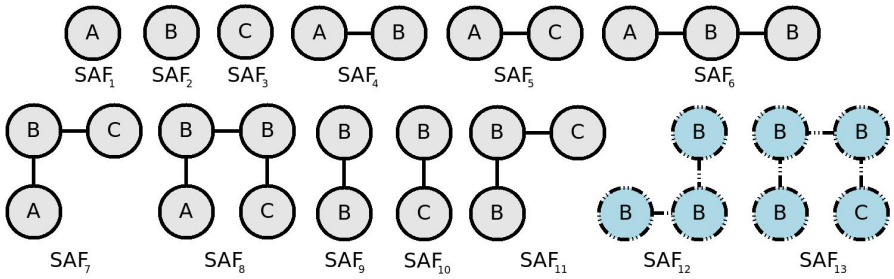

Fig. 16. Conjunto de posibles SAF identificados en la colección de grafos mostrada en la figura 15, de los cuales SAF_{12} y SAF_{13} no son detectados por gApprox. gApprox se ejecuta con $\Delta = 1$ y $matchable(A) = \{B\}$ sobre D.

3.5.2 APGM

Y. Jia *et al.* [14] presentan un algoritmo conocido como APGM para la minería de SAF en colecciones de grafos etiquetados y no dirigidos. Este algoritmo, hasta lo conocido, es el primero en incorporar las matrices de compatibilidad (ver definición 24) en este tipo de minería. En este algoritmo se utilizan solamente matrices estables en el cálculo de la similitud entre etiquetas. Estas matrices estarán indizadas solamente por las etiquetas de los vértices. Esto se debe a que en este algoritmo se utiliza el sub-isomorfismo aproximado para el cálculo de la similitud entre los grafos (ver definición 25) con la particularidad de que la inexactitud es tratada solamente en las etiquetas de los vértices manteniendo exactas las aristas.

En APGM se calcula la puntuación del sub-isomorfismo aproximado de la ocurrencia de $T = (V, E, L, l)$ en $G_i = (V_i, E_i, L_i, l_i)$, tal que $G_i \in D$, denotado por $S_f(T, G_i)$, según (23). Esta puntuación no es más que el producto normalizado de las probabilidades de que una etiqueta sea sustituida por otra.

$$S_f(T, G_i) = \prod \frac{M_{l(u),l_i(f(u))}}{M_{l(u),l(u)}}. \tag{23}$$

Para un par de grafos existe más de una forma de asignación de los vértices de un grafo a otro. Esto permite que se puedan obtenerse diferentes puntuaciones de sub-isomorfismo aproximado. Por tal motivo, se define la puntuación de cotejo aproximado entre dos grafos, denotado como $S(T, G_i)$, por la mayor puntuación de sub-isomorfismo aproximado, que representa la mejor aproximación entre los dos grafos (ver 24).

$$S(T, G_i) = max_f\{S_f(T, G_i)\}. \tag{24}$$

Por otro lado, se determina que el subgrafo candidato es aproximado frecuente si y solo si el valor de su soporte es al menos igual al del umbral de mínimo soporte, el cual tiene un valor $(0 < \delta \leq 1)$. Este soporte se obtiene a través de la siguiente ecuación.

$$supApprox(T,D) = \sum_{G_i \in D, T \subseteq_a G_i} S(T,G_i)/|D|. \tag{25}$$

Se dice que un subgrafo es SAF si su soporte es mayor o igual que el umbral de mínimo soporte $(supApprox(T,D) \geq \delta)$.

Definición 37 (Subgrafo aproximado). *Dado un grafo etiquetado $T = (V,E,L,l)$, una de las ocurrencias $o_j = u_1,\ldots,u_k$ de T en un grafo $G_i = (V_i,E_i,L_i,l_i)$, un vértice $v \in N(o_j)$, y la etiqueta de un vértice lv. El subgrafo candidato es un grafo $X = (V',E',L',l')$ denotado por $G|_{T,o_j,v,lv}$ tal que:*

- $V' = \{u_1,\ldots,u_k\} \cup \{v\}$,
- $E' = V' \times V' \cap E$,
- $L' = L$,
- $\forall u \in o_j : l'(u) = l(u)$,
- $l'(v) = lv$ *y*
- $\forall u,v \in o_j : l'((u,v)) = l_i((u,v))$.

Mediante la definición anterior se confeccionan los subgrafos candidatos. Con el uso de esta se obtiene la extensión, en un vértice, de un SAF y las aristas que unen a dicho vértice con los ya existentes en su ocurrencia. Estas aristas son obtenidas del grafo original al que pertenece la ocurrencia en cuestión. Sin embargo, mediante esta definición se generan solamente los subgrafos candidatos cerrados, lo cual no permite la identificación de un conjunto de SAF que deberían ser retornados por el algoritmo (ver ejemplo 5).

Ejemplo 5. En la figura 17 se muestra un SAF T y una de sus ocurrencias $o_j = (v_0,v_1)$ en el grafo $G = (V,E)$. El vértice v_2 es un vecino de o_j ya que $\exists u \in o_j | (u,v_2) \in E$. Dada la etiqueta de un vértice $lv =$ "3", se obtiene un subgrafo candidato $X = G|_{T,o_j,v_2,lv}$ mostrado en la misma figura. El nuevo subgrafo candidato X tiene una ocurrencia $g = (v_0,v_1,v_2)$ en G y la puntuación del sub-isomorfismo aproximado de X en G mediante g es $S(X,G) = \frac{M_{0,0}}{M_{0,0}} * \frac{M_{0,2}}{M_{0,0}} * \frac{M_{1,1}}{M_{1,1}} = \frac{M_{0,2}}{M_{0,0}} = 0{,}60$. Sin embargo, los subgrafos candidatos T_1 y T_2 tienen la misma puntuación de sub-isomorfismo aproximado que X, por lo que si X es SAF entonces los subgrafos T_1 y T_2 también lo son.

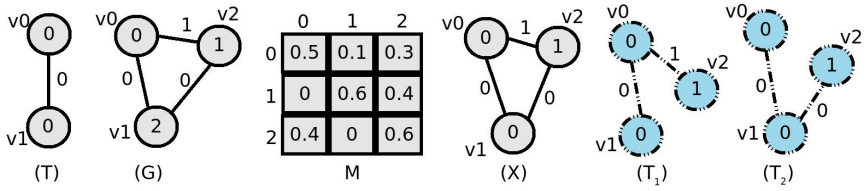

Fig. 17. Un SAF T, un grafo G, un subgrafo candidato X de G y dos posibles candidatos T_1 y T_2.

El algoritmo 29 muestra el pseudo-código inicial del algoritmo APGM. Lo primero es encontrar todos los vértices aproximados frecuentes en D y almacenarlos como parte de los SAF resultantes. Estos vértices son el punto de partida para recorrer los grafos de D mediante la llamada al procedimiento *apgm_DExten* para la confección del conjunto de SAF resultantes del minado.

El algoritmo 30 es el encargado de extender cada SAF en un vértice (ver analogía con el algoritmo 3). Para ello se utiliza las listas de correspondencias por cada SAF y se identifican los candidatos mediante el llamado al procedimiento *apgm_BCandidatos_LC* en la línea 1. Este procedimiento da como resultado un

conjunto de subgrafos candidatos de los cuales se mantienen los que resulten SAF. A cada uno de estos se le realiza la operación de extensión en busca de SAF mayores y de esta forma se confecciona el conjunto de todos los SAF de manera recursiva (ver líneas 2-4).

El algoritmo 31 muestra los detalles del procedimiento encargado de obtener los subgrafos candidatos hijos de T en G_i (ver analogía con el algoritmo 2). Para ello se realiza el llamado al procedimiento *approximateLabelSet* en la línea 4 con el objetivo de obtener todas las posibles etiquetas del vértice vecino a la ocurrencia de T que puede formar una extensión de la misma. Al encontrar estas posibles etiquetas se generan los grafos candidatos para cada una de estas etiquetas con el uso de la definición 37 en la línea 6. Todas las ocurrencias de los candidatos se van almacenando en una tabla hash H con el código CAM como llave (ver línea 9). Este código CAM es calculado como lo propusieron J. Huan *et al.* [17]. En este pseudo-código se puede observar que para cada candidato identificado mediante el crecimiento de patrones se realizan las costosas pruebas de formas canónicas. La mayoría de estas son innecesarias e influyen negativamente en el eficiente desempeño del algoritmo APGM en general.

Algoritmo 29: $APGM(D, M, \tau, \delta)$

Input: D : Colección de grafos, M : Matriz de compatibilidad, τ : Umbral de mínimo isomorfismo, δ : Umbral de mínimo soporte.
Output: F : Conjunto de SAF.

1 $F \leftarrow C \leftarrow \{$Vértices aproximados frecuentes$\}$;
2 Inicializar para cada SAF $T \in C$ la lista de correspondencias $EC(T, D)$;
3 **foreach** $T \in C$ **do** $apgm\_DExten(T, D, M, \tau, \delta, F)$;
4 **return** F;

Algoritmo 30: $apgm\_DExten(T, D, M, \tau, \delta, F)$

Input: T : Un SAF, D : Colección de grafos, M : Matriz de compatibilidad, τ : Umbral de mínimo isomorfismo, δ : Umbral de mínimo soporte.
Output: F : Conjunto de SAF.

1 **foreach** $(i, \lambda) \in EC(T, D)$ **do** $apgm\_BCandidatos\_LC(T, LC(T, G_i), G_i, M, \tau, C)$;
2 Borrar los subgrafos no frecuentes $T \in C$;
3 $F \leftarrow F \cup C$;
4 **foreach** $T \in C$ **do** $apgm\_DExten(T, D, M, \tau, \delta, F)$;

Algoritmo 31: $apgm\_BCandidatos\_LC(T, LC(T, G_i), G_i, M, \tau, C)$

Input: T : Un grafo candidato, $LC(T, G_i)$: Lista de correspondencias de T en G_i, $G_i = (V_i, E_i)$: Un grafo, M : Matriz de compatibilidad, τ : Umbral de mínima similitud.
Output: C : Los hijos de T que ocurren en G_i.

1 **foreach** $ptr = (t, \lambda) \in LC(T, G_i)$ **do**
2 Obtener la ocurrencia aproximada g de T en G_i mediante las pruebas de sub-isomorfismo recorriendo $LC(T, G_i)$ a través de λ;
3 **foreach** $v_j \in V_i$, *donde* $v_j \in N(g)$ **do**
4 $CL \leftarrow$ `approximateLabelSet` (T, G_i, M, g, v_j) ; //Conjunto de etiquetas candidatas
5 **foreach** $b \in CL$ **do**
6 $X \leftarrow G|_{T,g,v_j,b}$; // Subgrafo confeccionado mediante la definición 37
7 $C \leftarrow C \cup \{X\}$;
8 $LC(X, G_i) \leftarrow LC(X, G_i) \cup \{(j, ptr)\}$;
9 $H(X) = H(X) \cup (g, v_j)$;

En el algoritmo 32 se muestra el pseudo-código del procedimiento *approximateLabelSet*. En este procedimiento se encuentran todas las posibles etiquetas de los vértices de G que pueden formar una extensión de T al agregarle el vértice v a la ocurrencia g de T en G. Para lograrlo, primero se toma la etiqueta

del vértice v en el grafo G y se almacena en b_0 para el cálculo posterior del isomorfismo aproximado. Por cada etiqueta se comprueba la existencia de un sub-isomorfismo aproximado con la ocurrencia con el uso de la matriz de compatibilidad. Esta comprobación se realiza mediante la comparación del resultado de la función heurística que representa la ecuación 24 y el umbral de mínimo isomorfismo. Cuando esta comprobación es positiva se almacena la etiqueta del vértice como posible extensión con la ocurrencia $g \in G$ (ver línea 3).

Algoritmo 32: $approximateLabelSet(T, G, M, g, v, \tau)$

Input: T : Grafo candidato, $G = (V, E, L, l)$: Grafo de la colección, M : Matriz de compatibilidad, g : Ocurrencia de T en G, v : Vértice vecino a la ocurrencia de T en G, τ : Umbral de mínimo isomorfismo.
Output: CL : Conjunto de etiquetas candidatas.

1 $b_0 \leftarrow l(v)$;
2 **foreach** $j \in L_V$ **do**
3 $\quad$ **if** $S(g,T) * \frac{M_{j,b_0}}{M_{j,j}} \geq \tau$ **then** $CL \leftarrow CL \cup j$;
4 **return** CL;

Al analizar el pseudo-código del procedimiento $approximateLabelSet$ fue detectada una dificultad que atenta contra la obtención de los subgrafos candidatos. Según lo planteado en el ciclo *foreach*, se toman todas las posibles etiquetas de un vértice según el conjunto de etiquetas del grafo $G_i \in D$ para la generación de futuros candidatos. De esta forma no se detectan algunos SAF ya que una o varias etiquetas pudieran existir solamente en uno o unos pocos grafos de la colección. Aunque la similitud de estas etiquetas con respecto a otras esté entre las más altas de la matriz de compatibilidad solamente se obtendrán cuando se estén procesando los G_i que las contienen. Esto se debe a que los candidatos que contengan esta o estas etiquetas solamente presentarán ocurrencias en los G_i donde estos existen. Un ejemplo de una colección donde APGM no detecta todos los SAF se muestra en la figura 18. Los grafos G_1, G_2 y G_3 conforman la colección de grafos D y los subgrafos T_1, T_2 y T_3 son SAF que no serán detectados con $\delta = 0{,}55$ y $\tau = 0{,}4$.

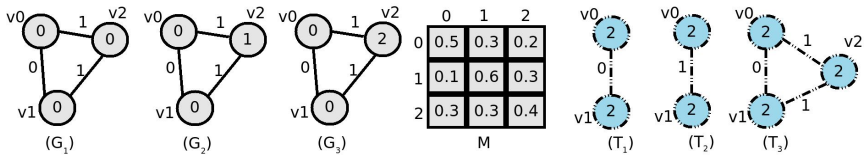

Fig. 18. Ejemplo de una colección conformada por los tres grafos de la izquierda donde APGM no detecta algunos SAF como son los subgrafos T_1, T_2 y T_3.

3.5.3 Discusión

Los algoritmos que utilizan las probabilidades de sustitución en el cálculo de la similitud tienen las siguientes dificultades:

1. No se detectan todos los posibles SAF de la colección de grafos, lo que afecta la eficacia de dicho algoritmo.
2. En APGM no se incluyen a las aristas de forma inexacta en la obtensión de la similitud entre grafos.
3. En APGM se realizan las pruebas de formas canónicas para todos los candidatos generados por el crecimiento de patrones, lo cual influye de forma negativa en la eficiencia de este.

3.6 Síntesis y conclusiones

En esta sección se analizaron 7 algoritmos para la minería de SAF reportados en la literatura relacionados con el tema de investigación del presente trabajo. Estos fueron clasificados según la técnica de cotejo utilizada como función de similitud entre grafos y la estrategia de generación de candidatos que se utilizó. A modo de resumen, la tabla 1 muestra las características generales de los algoritmos reportados en el estado del arte de este trabajo.

Los algoritmos presentados en este trabajo tienen otras características que no fueron ilustradas en la tabla anterior pero que influyen de forma directa sobre la eficiencia y eficacia de estos:

- La mayor parte de estos aceleran el cálculo de la frecuencia utilizando estructuras de datos para indizar la colección de grafos.
- En algunos algoritmos se logra la detección de duplicados mediante pruebas de formas canónicas. En otros, se van eliminando los vértices de los cuales se han identificado todos sus supergrafos y de esta forma se reduce el número de duplicados.
- En algunos algoritmos de almacenan los subgrafos candidatos en tablas hash para consultar de manera eficiente los subgrafos obtenidos con anterioridad.

Tabla 1. Resumen de las características de los algoritmos de minería de SAF.

Algoritmo	**Funsión de similid**	**Estrategia de generación de candidatos**	**Estructura de datos para indizar la colección de grafos**	**Resultado de la minería**
SUBDUE	Distancia de edición de grafos	Incluyen características de crecimiento de patrones	Listas de correspondencias	Una cantidad definida por el usuario de los SAF
RNGV	Distancia de edición de grafos	Crecimiento de patrones	Listas de identificadores	Los SAF maximales
Monkey	β arista isomorfismo	Crecimiento de patrones	Listas de identificadores	Todos los SAF
CSMiner	Homeomorfismo con vértice/arista disjuntas	Apriori	Matrices indizadas por los vértices de los grafos de la colección	Todos los SAF
MUSE	Isomorfismo	Crecimiento de patrones	Fórmulas DNF	Todos los SAF
gApprox	Grado de aproximación	Crecimiento de patrones	Listas de correspondencias	Algunos SAF
APGM	Isomorfismo aproximado	Crecimiento de patrones	Listas de correspondencias	Algunos SAF cerrados

Partiendo del análisis realizado a lo largo de esta sección se ha detectado que los algoritmos reportados en este trabajo presentan diferentes problemas en general, tales como:

1. Solo dos algoritmos de los siete reportados (gApprox, APGM) tienen en cuenta las diferencias estructurales y entre las etiquetas de los vértices.
2. Ninguno de los algoritmos reportados en el estado del arte de este trabajo tienen en cuenta las diferencias de las etiquetas de las aristas cuando estas etiquetas no son numéricas.
3. Algunos algoritmos (SUBDUE, gApprox, APGM) no obtienen todos los posibles SAF debido al uso de restricciones por parte del usuario de los tamaños y cantidades a obtener, o simplemente para ganar en eficiencia.
4. Algunos algoritmos (SUBDUE, RNGV) no tienen en cuenta los caminos de edición identificados para los SAF en la obtención de los caminos de sus hijios.
5. En algunos algoritmos se realizan las costosas pruebas de formas canónicas para todos los candidatos generados por el crecimiento de patrones.
6. En algunos algoritmos se realizan las costosas pruebas de sub-isomorfismo debido al uso de listas de identificadores para indzar la colección de grafos.

4 Conclusiones

Para lograr mejores resultados en diferentes tareas es necesario desarrollar algoritmos eficientes para la minería de SAF, tal y como fue reportado por varios autores en tareas de: procesamiento de datos químicos [21], agrupamiento de documentos [19], análisis de vínculos [7]. Los problemas presentes en los siete algoritmos reportados en la literatura para este tipo de minería atentan contra su buen desempeño. Teniendo en cuenta este conjunto de problemas se pudo llegar a las siguientes conclusiones:

- No se puede realizar la minería de SAF basada en la sustitución de etiquetas no numéricas de vértices y aristas utilizando los algoritmos reportados.
- No se logra un equilibrio entre el uso de los recursos de memoria y la realización de las pruebas de sub-isomorfismo en los algoritmos reportados.

Partiendo del análisis realizado a lo largo de este trabajo se puede concluir que los algoritmos presentados no satisfacen todas las necesidades de los problemas existentes en este tipo de minería que afectan a la eficiencia y eficacia de dichos algoritmos. Por lo tanto, el desarrollo de algoritmos eficientes para la minería de SAF se considera una línea de investigación abierta.

Referencias bibliográficas

1. Han, J., Cheng, H., Xin, D., Yan, X.: Frequent pattern mining: Current status and future directions. In: Data Mining and Knowledge Discovery,10th Anniversary Issue. (November 2007) 15(1):55–86
2. Gago-Alonso, A.: Minería de subgrafos conexos frecuentes en colecciones de grafos etiquetados. PhD thesis, Instituto Nacional de Astrofísica, Óptica y Electrónica, Tonantzintla, Puebla, México (Enero 2010)
3. Chen, C., Yan, X., Zhu, F., Han, J.: gapprox: Mining frequent approximate patterns from a massive network. In: International Conference on Data Mining (ICDM'07). (2007) 445–450
4. Zou, Z., Li, J., Gao, H., Zhang, S.: Frequent subgraph pattern mining on uncertain graph data. In: CIKM'09: Proceeding of the 18th ACM conference on Information and knowledge management, New York, NY, USA, ACM (2009) 583–592
5. Song, Y., Chen, S.S.: Item sets based graph mining algorithm and application in genetic regulatory networks. Data Mining, IEEE International Conference on Volume, Issue (2006) 337–340
6. Xiao, Y., Wang, W., Wu, W.: Mining conserved topological structures from large protein-protein interaction networks, Hiroshima, Japan, DEWS2007 (18th IEICE data engineering workshop / 5th DBSJ annual meeting) (2007)
7. Ketkar, N., Holder, L., Cook, D.: Mining in the proximity of subgraphs. Analysis and Group Detection KDD Workshop on Link Analysis: Dynamics and Statics of Large Networks (August 2006)
8. Lahiri, M., Berger-Wolf, T.Y.: Structure prediction in temporal networks using frequent subgraphs. In: Computational Intelligence and Data Mining, CIDM'07. (2007) 35–42
9. Cook, D.J., Holder, L.B.: Substructure discovery using minimum description length and background knowledge. Journal of Artificial Intelligence Research **1** (1994) 231–255
10. Mukherjee, M., Holder, L.B.: Graph-based data mining on social networks. Workshop on Link Analysis and Group Detection (in conj. with the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining) (2004)
11. Eichinger, F., Böhm, K.: Software-Bug Localization with Graph Mining. In Aggarwal, C.C., Wang, H., eds.: Managing and Mining Graph Data. Volume 40 of Advances in Database Systems. Springer-Verlag New York (2010)
12. Nijssen, S., Kok, J.N.: A quickstart in frequent structure mining can make a difference. In: KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, New York, NY, USA, ACM (2004) 647–652
13. Holder, L.B., Cook, D.J., Bunke, H.: Fuzzy substructure discovery. In: ML92: Proceedings of the ninth international workshop on Machine learning, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (1992) 218–223
14. Jia, Y., Huan, J., Buhr, V., Zhang, J., Carayannopoulos, L.: Towards comprehensive structural motif mining for better fold annotation in the "twilight zone" of sequence dissimilarity. BMC Bioinformatics **10**(S-1) (2009)
15. Conte, D., Foggia, P., Sansone, C., Vento, M.: Thirty years of graph matching in pattern recognition. IJPRAI **18**(3) (2004) 265–298
16. Xiao, Y., Wu, W., Wang, W., He, Z.: Efficient algorithms for node disjoint subgraph homeomorphism determination. In: Proceedings of the 13th international conference on Database systems for advanced applications. DASFAA'08, Berlin, Heidelberg, Springer-Verlag (2008) 452–460

17. Huan, J., Wang, W., Prins, J.: Efficient mining of frequent subgraphs in the presence of isomorphism. In: Proceedings of the 3rd IEEE International Conference on Data Mining. (2003) 549–552
18. Kuramochi, M., Karypis, G.: An efficient algorithm for discovering frequent subgraphs. Technical report, IEEE Transactions on Knowledge and Data Engineering (2002)
19. Hossain, M.S., Angryk, R.A.: Gdclust: A graph-based document clustering technique. In: ICDMW '07: Proceedings of the Seventh IEEE International Conference on Data Mining Workshops, Washington, DC, USA, IEEE Computer Society (2007) 417–422
20. Koyutürk, M., Grama, A., Szpankowski, W.: An efficient algorithm for detecting frequent subgraphs in biological networks. In: Bioinformatics. (2004) 200–207
21. Borgelt, C.: Mining molecular fragments: Finding relevant substructures of molecules. In: Proc. of 2002 IEEE International Conference on Data Mining (ICDM, IEEE Press (2002) 51–58
22. Eberle, W., Holder, L.B.: Mining for insider threats in business transactions and processes. In: Computational Intelligence and Data Mining (CIDM'09). (2009) 163–170
23. Cook, D.J., Manocha, N., Holder, L.B.: Using a graph-based data mining system to perform web search. International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI) **17**(5) (2003) 705–720
24. Zhang, S., Yang, J., Cheedella, V.: Monkey: Approximate graph mining based on spanning trees. In: IEEE 23rd International Conference on Data Engineering 2007, Los Alamitos, CA, USA (2007) 1247–1249
25. Fellman, P.V.: Modeling terrorist networks-complex systems at the mid-range. In: Downloaded from the internet. (November 2008)
26. Sanfeliu, A., Fu, K.S.: A distance measure between attributed relational graphs for pattern recognition. In: IEEE Transactions on Systems, Man, and Cybernetics (Part B). (1983) 13(3):353–363
27. Messmer, B.T., Bunke, H.: A new algorithm for error-tolerant subgraph isomorphism detection. In: IEEE Transactions on Pattern Analysis and Machine Intelligence. (1998) 20(5):493–504
28. Neuhaus, M., Bunke, H.: A probabilistic approach to learning costs for graph edit distance. In: J. Kittler, M. Petrou, and M. Nixon, eds. Proceedings 17th International Conference on Pattern Recognition, Cambridge, United Kingdom. (2004) Vol. 3, pp. 389–393
29. Neuhaus, M., Bunke, H.: Automatic learning of cost functions for graph edit distance. Information Sciences **177**(1) (2007) 239–247
30. Ambauen, R., Fischer, S., Bunke, H.: Graph edit distance with node splitting and merging, and its application to diatom identification. In: Proceedings of the 4th IAPR international conference on Graph based representations in pattern recognition. GbRPR'03, Berlin, Heidelberg, Springer-Verlag (2003) 95–106
31. ZhipingZeng, Tung, A.K.H., Wang, J., JianhuaFeng, Zhou, L.: Comparing stars: On approximating graph edit distance. PVLDB **2**(1) (2009) 25–36
32. Neuhaus, M., Riesen, K., Bunke, H.: Fast suboptimal algorithms for the computation of graph edit distance. In Yeung, D.Y., Kwok, J.T., Fred, A.L.N., Roli, F., de Ridder, D., eds.: SSPR/SPR. Volume 4109 of Lecture Notes in Computer Science., Springer (2006) 163–172
33. Zou, Z., Li, J., Gao, H., Zhang, S.: Mining frequent subgraph patterns from uncertain graph data. IEEE Trans. on Knowl. and Data Eng. **22**(9) (2010) 1203–1218
34. Ketkar, N.S.: Subdue: compression-based frequent pattern discovery in graph data. In: OSDM'05: Proceedings of the 1st international workshop on open source data mining, ACM Press (2005) 71–76
35. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In Proceedings of the 2000 ACM-SIGMOD International Conference on Management of Data (SIGMOD'2000), Dallas, TX (2000) 1–12
36. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In Proceedings of the 1994 International Conference on Very Large Data Bases (VLDB'94), Santiago, Chile (1994) 487–499
37. Cook, D.J., Holder, L.B.: Mining Graph Data. John Wiley & Sons (2006)
38. Hamza, H., Belaïd, Y., Belaïd, A., Chaudhuri, B.B.: An end-to-end administrative document analysis system. In: Document Analysis Systems '08: Proceedings of the 2008 The Eighth IAPR International Workshop on Document Analysis Systems, Washington, DC, USA, IEEE Computer Society (2008) 175–182
39. Aery, M., Chakravarthy, S.: emailsift: Email classification based on structure and content. In: ICDM. (2005) 18–25
40. Venkatachalam, A.: M-infosift: A graph-based approach for multiclass document classification (2007)
41. Chittimoori, R.N., Holder, L.B., Cook, D.J.: Applying the subdue substructure discovery system to the chemical toxicity domain. FLAIRS Conference (1999)
42. Aery, M.: Infosift: Adapting graph mining techniques for document classification (2004)
43. Cook, D.J., Holder, L.B., Su, S., Maglothin, R., Jonyer, I.: Structural mining of molecular biology data. In: IEEE Engineering in Medicine and Biology, special issue on Advances in Genomics. Volume 2. (2001) 67–74
44. Galal, G., Cook, D.J., Holder, L.B.: Improving scalability in a knowledge discovery system by exploiting parallelism. In: Proceedings of the Third International Conference on Knowledge Discovery and Data Mining. (1997) 171–174

45. Rissanen, J.: Stochastic Complexity in Statistical Inquiry Theory. World Scientific Publishing Co., Inc., River Edge, NJ, USA (1989)
46. Bunke, H., Allermann, G.: Inexact graph matching for structural pattern recognition. Pattern Recognition Letters **1**(4) (1983) 245–253
47. Valiant, L.G.: The complexity of computing the permanent. Theor. Comput. Sci. **8** (1979) 189–201
48. Yan, X., Huan, J.: gspan: Graph-based substructure pattern mining. In: Proc. Int'l Conf. Data Mining. (2002)

Anexos

Notaciones

Ω	Conjunto de todos los posibles grafos etiquetados y $\emptyset \in \Omega$
τ	Umbral de mínimo isomorfismo
δ	Umbral de mínima frecuencia
β	Umbral de cantidad máxima diferencias entre aristas de dos grafos
Δ	Umbral de máxima disimilitud
u, v, w	Vértices de un grafo
$(u,v), (u,w)$	Aristas de un grafo
$N(u)$	Conjunto de vértices vecinos al vértice u
$N(u,v)$	Conjunto de aristas vecinas a la arista (u,v)
G_i	Grafos etiquetados
$\langle V, E, L, l \rangle$	Las cuatro componentes de un grafo etiquetado
V	Conjunto de vértices de un grafo
E	Conjunto de aristas de un grafo
L	Conjunto de etiquetas de un grafo
l	Función que asigna etiquetas a los vértices y aristas de un grafo
i, j, k	Sub-índices (números enteros no negativos)
$A \subseteq B$	A es subconjunto de B
$G_1 \subseteq_s G_2$	G_1 es subgrafo de G_2
$G_1 \subset_s G_2$	G_1 es sub-isomorfo de G_2
D	Colección de grafos
G, T, G_i	Grafos
$LI(T,D)$	Lista de identificadores i de los grafos $G_i \in D$ que contienen al menos una ocurrencia de T
$LC(T,G_i)$	Lista de correspondencias de T en $G_i \in D$
S	Una subestructura
G^k	Un subgrafo con k aristas
C_k	Conjunto de subgrafos candidatos con k aristas
iC	Costo de la operación de inserción de vértice o arista
sC	Costo de la operación de sustitución de vértice o arista
dC	Costo de la operación de eliminación de vértice o arista
CE_i	Camino de edición $\{e_1, \ldots, e_k\}$
$\Upsilon(G_1, G_2)$	Conjunto de caminos de edición de G_1 a G_2
$sup(G,D)$	Soporte del grafo G en la colección de grafos D
$sim(G_1, G_2)$	Función de similitud entre los grafos G_1 y G_2
$supApp(G,D)$	Soporte del grafo aproximado G en la colección de grafos D
$O(T,G)$	Conjunto de ocurrencias de T en G
o_j	j-ésima ocurrencia de un grafo en otro $o_j \in O(T,G)$
F	Conjunto de Subgrafos Frecuentes
$sup_\beta(D,G,\beta)$	conjunto de grafos $G_i \in D$ de los cuales G es β arista sub-isomorfo
$G \diamond \{u,v\}$	Extensión del grafo G mediante una arista
$G_1 \approx_\beta G_2$	G_1 es β arista isomorfo a G_2
$G_1 \subseteq_\beta G_2$	G_1 es β arista sub-isomorfo a G_2

Acrónimos

DFS	*Depth First Search*
BFS	*Breadth First Search*
CAM	*Canonical Adjacency Matrix*
SF	*Subgrafo Frecuente*
SAF	*Subgrafo Aproximado Frecuente*
DEG	*Distancia de edición de grafos*
MDL	*Minimum Description Length*

RT_017, junio 2011
Aprobado por el Consejo Científico CENATAV

Editor: Lic. Lucía González Bayona
Diseño de Portada: Di. Alejandro Pérez Abraham
RNPS No. 2143
ISSN 2072-6260
Indicaciones para los Autores:
Seguir la plantilla que aparece en www.cenatav.co.cu
C E N A T A V
7ma. No. 21812 e/218 y 222, Rpto. Siboney, Playa;
La Habana. Cuba. C.P. 12200
Impreso en Cuba