Minería de subgrafos conexos frecuentes eliminando todos los candidatos duplicados

Andrés Gago-Alonso, Jesús Ariel Carrasco-Ochoa,
José Eladio Medina-Pagola, José Fco. Martínez-Trinidad
Coordinaciones de Ciencias Computacionales
agago@ccc.inaoep.mx, ariel@inaoep.mx,
jmedina@cenatav.co.cu, fmartine@inaoep.mx

RESUMEN

La minería de subgrafos conexos frecuentes (SCF) en colecciones de grafos etiquetados se ha convertido en una importante línea de investigación, dentro de la minería de datos, por sus amplias aplicaciones. La identificación de los duplicados y el conteo de frecuencia siguen siendo dos tareas que demandan de muchos recursos computacionales en la minería de SCF. En este trabajo se presenta un nuevo algoritmo para la minería de SCF que sea enfocado a realizar estas dos tareas de una manera más eficiente que los métodos existentes. El desempeño de este nuevo algoritmo es comparado con otros métodos reportados en el estado del arte.

I INTRODUCCIÓN

Hoy en día, debido a los rápidos avances científicos y tecnológicos, se han incrementado notablemente las capacidades de creación, almacenamiento y distribución de grandes volúmenes de datos. Esta situación ha generado la necesidad de nuevas herramientas para transformar estos datos en información o conocimiento útil para la toma de decisiones. Ejemplos de tales técnicas son las relacionadas con subgrafos conexos frecuentes (SCF) en colecciones de grafos etiquetados [1].

La minería de SCF consiste en encontrar todos los subgrafos conexos que ocurren frecuentemente en una colección de grafos.

Los primeros algoritmos para la minería de todos los SCF fueron FSG [3] y AcGM [2]. Desde entonces se han ido creando nuevos algoritmos cada vez más eficientes para encontrar todos los SCF [1]. Entre estos se puede mencionar a gSpan [6] y Gaston [4] los cuales han mostrado los mejores desempeños en estudios comparativos [5]. La identificación de los duplicados y el conteo de frecuencia siguen siendo dos tareas que atentan contra la eficiencia de estos algoritmos [1,5].

Para acelerar la detección de duplicados se han utilizado las formas canónicas. En particular, los códigos DFS han sido presentados como una solución prometedora para este problema [1,5].

En este artículo se presentan cinco nuevas propiedades de la codificación DFS que permiten acelerar la detección de duplicados en la minería de SCF. Además, se introduce un nuevo algoritmo llamado gdFil que está enfocado a acelerar el tiempo dedicado a dichas dos tareas (utilizando las nuevas propiedades).

La organización de este artículo es la siguiente. Primero, se introducen los conceptos básicos relacionados con los códigos DFS (sección II). Luego, se presentan las nuevas propiedades de los códigos DFS (sección III). El algoritmo gdFil es presentado en la sección IV. La sección V muestra los resultados experimentales. Finalmente, se plantean las conclusiones de este trabajo así como los posibles trabajos futuros.

II CONCEPTOS BÁSICOS

Un árbol DFS T puede ser construido cuando se realiza un recorrido en profundidad en un grafo $G = \langle V, E, L, l \rangle$. Este grafo G puede tener varios árboles DFS porque casi siempre existe más de un recorrido en profundidad. Cada árbol define un orden entre todos los vértices del grafo; por tanto, se puede enumerar cada vértice de acuerdo a este orden DFS. La raíz de T será enumerada con índice 0 ya que es el primer vértice en el orden DFS. El último vértice en dicho orden corresponde al *vértice más a la derecha* de T. El camino directo desde la raíz hasta el vértice más a la derecha es llamado la *rama más a la derecha*.

Cada arista de G es representada por una 5-tupla, $(i, j, l_i, l_{(i,j)}, l_j)$ donde i y j son los índices de los vértices (origen y destino respectivamente), l_i y l_j son las etiquetas de tales vértices, y $l_{(i,j)}$ es la etiqueta de la arista. Si $i < j$ entonces se está en presencia de una arista hacia adelante; de otro modo es una arista hacia atrás.

Una relación de orden total $\prec_e$ entre dos tuplas e_1 y e_2 puede ser definida de la siguiente manera. Se cumple que $e_1 \prec_e e_2$ si e_1 aparece antes que e_2 en un recorrido DFS. Cuando tuplas e_1 y e_2 tiene los

mismo vértices de origen y destino, entonces son comparadas respecto al orden lexicográfico de las etiquetas (las últimas tres componentes de las tuplas).

Un código DFS de G es una secuencia de las tuplas que conforman las aristas de G ordenadas según $\prec_e$. El orden $\prec_e$ puede ser extendido a un *orden lexicográfico* ($\prec_s$) entre códigos DFS. El código DFS mínimo de G es definido como el menor código DFS, respecto a $\prec_s$, entre todos los posibles códigos de G.

Para comprobar que un código DFS es mínimo o no se requiere de una prueba exhaustiva que poseer una alta complejidad computacional en tiempo. Estas pruebas son conocidas como *pruebas CF* (por sus siglas en inglés *Canonical Form test*). Todos los algoritmos de minería de SCF realizan pruebas CF para todos los candidatos (mecanismo para eliminar los duplicados). En la próxima sección se introducen nuevas propiedades de los códigos DFS que permiten reducir el número de pruebas CF.

Sea s un código DFS mínimo y e una arista que no pertenece a s. Se dice que e es una *extensión más a la derecha* de s si e conecta al vértice más a la derecha con otro vértice en la rama más a derecha de s (*extensión hacia atrás*); o e introduce un nuevo vértice conectado desde algún vértice de la rama más a derecha de s (*extensión hacia adelante*). En tales casos, el código $s' = s \diamond e$ denotará al código obtenido luego de extender a s con e; se dice que s' es un hijo de s o que s es el padre de s'.

Los códigos DFS fueron introducidos por primera vez en [6].

III NUEVAS PROPIEDADES DE LOS CÓDIGOS DFS

En esta sección se presentan nuevas propiedades de los códigos DFS que, más adelante, serán utilizadas para reducir el costo de detectar los candidatos duplicados en la minería de SCF.

Sea s un código DFS mínimo. Se denotará mediante $RE(s)$ el conjunto de extensiones más a la derecha desde s. Este conjunto puede ser particionado en varios subconjuntos:

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

donde $B_i(s)$ contiene las extensiones hacia atrás (tuplas) que tienen vértice destino con índice i, y $F_i(s)$ es el conjunto de las extensiones hacia adelante desde el vértice con índice i. Para cada vértice v_i en la rama más a la derecha ($i, i \neq n-1$, es el índice de v_i en el recorrido DFS) la arista f_i representa la arista hacia adelante que nace en v_i y que forma parte de la rama más a la derecha. La notación e^{-1} será utilizada para referirse a la tupla e pero en el sentido inverso al que se le impuso en el código s.

Sea v_i un vértice en la rama más a la derecha de s con índice i. Entonces, se cumplen las siguientes propiedades:

P1: Si e $\in F_i(s)$, $i \neq n-1$, y $e \prec_e f_i$, entonces s $\diamond$ e es un código DFS no-mínimo.

P2: Si e $\in B_i(s)$, y $e^{-1} \prec_e f_i$, entonces s $\diamond$ e es un código DFS no-mínimo.

P3: Sea $E_i(s)$ uno de los conjuntos $B_i(s)$ o $F_i(s)$ y sea $e_1, e_2 \in E_i(s)$ tales que $e_1 \prec_e e_2$. Si $s \diamond e_1$ es un código DFS mínimo, entonces $s \diamond e_2$ es un código DFS mínimo. Si $s \diamond e_2$ es un código DFS no-mínimo, entonces $s \diamond e_1$ es un código DFS no-mínimo.

Las propiedades P1 y P2 constituyen condiciones suficientes de no-minimalidad para algunos hijos de s. P3, por su parte, es una propiedad de reutilización ya que permite deducir la minimalidad o no de un hijo de s a partir de la minimalidad o no de otro hijo de s.

Si se asume que $E_i(s)$ está ordenado según el orden $\prec_e$, entonces de la propiedad P3 se deduce lo siguiente:

P4: Si existen extensiones en $E_i(s)$ que producen códigos DFS no-mínimos entonces tales extensiones se encuentran al principio de $E_i(s)$.

P5: Si existen extensiones en $E_i(s)$ que producen códigos DFS mínimos entonces tales extensiones se encuentran al final de $E_i(s)$.

De las propiedades P4 y P5 se deduce que cada conjunto $E_i(s)$, puede ser particionado en dos subconjuntos $E_i(s) = \hat{E}_i(s) \cup \check{E}_i(s)$ tal que todas las extensiones en $\hat{E}_i(s)$ producen códigos DFS no-mínimos (partición duplicada), todas las extensiones en $\check{E}_i(s)$ producen códigos DFS mínimos (partición útil) y las extensiones de $\hat{E}_i(s)$ son anteriores a las de $\check{E}_i(s)$ según la relación de orden $\prec_e$.

El mayor elemento (según $\prec_e$) del conjunto $\hat{E}_i(s)$ es llamado *corte no-canónico* de $E_i(s)$ y el menor elemento de $\check{E}_i(s)$ es llamado *corte canónico* de $E_i(s)$. Nótese que alguno de estos cortes pudiera no existir ya que los conjuntos $\hat{E}_i(s)$ y $\check{E}_i(s)$ pudieran ser vacíos.

Sea $H_i(s)$ un subconjunto propio de $E_i(s)$. Se denotará mediante $\hat{e}_i(s)$ y $\check{e}_i(s)$ los cortes no-

canónico y canónico de $E_i(s)$ respectivamente. De igual manera, $\hat{h}_i(s)$ y $\check{h}_i(s)$ denotan los cortes de $H_i(s)$. Entonces dichos cortes están dispuestos en el siguiente orden:

$$\hat{h}_i(s) \leq \hat{e}_i(s) < \check{e}_i(s) \leq \check{h}_i(s) \ . \qquad (2)$$

En este caso, se dice que $\hat{e}_i(s)$ y $\check{e}_i(s)$ son los *cortes globales* de $E_i(s)$ y que $\hat{h}_i(s)$ y $\check{h}_i(s)$ son *cortes locales* de $E_i(s)$. Los cortes globales son únicos para cada conjunto $E_i(s)$; sin embargo $E_i(s)$ puede tener varios cortes locales.

IV ALGORITMO GDFIL

En esta sección de introduce un nuevo algoritmo llamado gfFil (por sus siglas en Inglés *graph duplicate-filtering miner*) que utiliza las propiedades de la sección anterior para eliminar todos los duplicados durante la enumeración de candidatos.

La idea central de gdFil consiste en ir calculando los cortes (canónicos y no canónicos) mientras se enumeran los candidatos.

El Algoritmo 1 muestra una descripción general gdFil. Primero, se eliminan todos los vértices y aristas no frecuentes y así tener una representación simplificada de la colección de grafos (línea 1). Luego, se recorre el espacio de búsqueda a partir de cada una de las aristas frecuentes (línea 4). Para cada arista frecuente e, el procedimiento `buscar` es el encargado de encontrar todos los SCF frecuentes que contienen a e.

El procedimiento `buscar` se encarga de determinar las extensiones más a la derecha desde el SCF representado por el código DFS s (determinar los nuevos candidatos). Para evitar que surjan candidatos ilógicos (que no ocurran en D), se recorren cada uno de los grafos $G \in D$ tales que s es subgrafo de G (línea 8) y se determinan las extensiones e tales que el candidato $s \diamond e$ es un subgrafo de G (línea 9).

Para eliminar los candidatos duplicados, el procedimiento `buscar` va calculando los cortes canónicos y no-canónicos mientras se enumeran los candidatos. Primero, los cortes globales son inicializados, los no-canónicos toman valor $-\infty$ y los canónicos en $+\infty$ para garantizar que se cumpla (2) aun cuando no se han calculado cortes locales. Luego, los cortes locales son calculados (línea 12) usando cada conjunto C_r y se utilizan para obtener aproximaciones a los cortes globales (línea 13).

Las aproximaciones de los cortes globales son utilizadas para eliminar candidatos duplicados sin necesidad de realizar pruebas CF (línea 11). En la primera iteración del ciclo (líneas 8–15), el filtrado por los cortes globales (línea 11) no tiene ningún efecto ya que los cortes globales están inicializados en $\pm\infty$. En las siguientes iteraciones de dicho ciclo, algunos candidatos serán filtrados según las aproximaciones a los cortes globales que cada vez serán más cercanos al valor correcto.

Las pruebas CF solamente son requeridas cuando existan candidatos que no sean filtrados por los cortes globales. En ese caso, el resultado de dichas pruebas es usado para calcular los cortes locales (línea 12).

Algoritmo 1. `gdFil`(D, δ, S)

Input: D – colección de grafos, δ – umbral de frecuencia mínima.
Output: S – resultado de la minería

1 Eliminar vértices y aristas no frecuentes en D;
2 $S \leftarrow$conjunto con todos los vértices frecuentes;
3 $E \leftarrow$conjunto con todos las aristas (códigos DFS de tamaño 1) frecuentes;
4 **forall** $e \in E$ **do** `buscar`(D, e, δ, S);

Procedimiento `buscar` (D, s, δ, S);

5 $S \leftarrow S \cup \{s\}$;
6 $C \leftarrow \emptyset$;
7 Inicializar **cortes globales** de C los no-canónicos en $-\infty$ y los canónicos en $+\infty$;
8 **forall** $G \in D$ tal que s es un subgrafo de G **do**
9 $C_r \leftarrow$ conjunto de extensiones e tales que $s \diamond e$ es un subgrafo de G;
10 Eliminar de C_r los elementos que son filtrados por las propiedades P1 y P2;
11 Eliminar de C_r los elementos que son filtrados por los **cortes globales** de C;
12 Calcular los cortes de C_r (**cortes locales** de C) mediante las búsquedas binarias y eliminar de C_r los duplicados que quedan;
13 Actualizar los **cortes globales** a partir de los **cortes locales** de C;
14 $C \leftarrow C \cup C_r$;
15 **end**
16 Eliminar de C las extensiones no frecuentes según δ;
17 **forall** $e \in C$ **do** `buscar` $(D, s \diamond e, \delta, S)$;

El valor correcto de los cortes globales sólo se alcanzará con seguridad al final de estas iteraciones; no obstante, pudiera ocurrir que desde la primera iteración ya estos cortes alcancen su valor definitivo. Es de esperar que en las últimas iteraciones no sea necesario realizar ninguna prueba CF para localizar nuevos cortes locales.

Al terminar estas iteraciones el conjunto C contiene solamente candidatos útiles (no duplicados) solamente es necesario eliminar aquellos que no son frecuentes (línea 16). Luego, se procede a continuar la búsqueda de manera recursiva para encontrar los SCF que contienen a s. Esta búsqueda termina cuando s los candidatos sean todos duplicados o no frecuentes.

En gdFil se utilizan estructuras de correspondencias, análogas a las utilizadas en otros algoritmos de minería de SCF [4], para mantener indexados los grafos $G \in D$ tales que s es subgrafo de G. De este modo se acelera el paso de una iteración a otra en la línea 8.

A diferencia de los algoritmos reportados, gdFil solamente calcula dichas estructuras de correspondencias para los candidatos útiles (no duplicados).

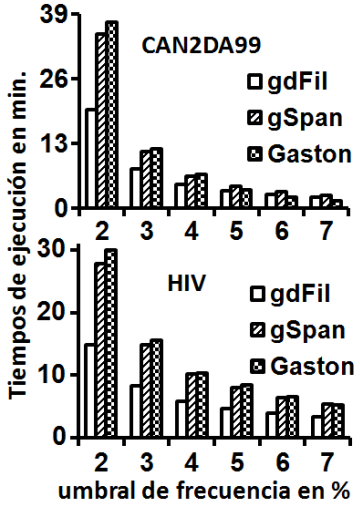

Fig. 1 Tiempos de ejecución usando las colecciones CAN2DA99 y HIV y variando el umbral de frecuencia.

V RESULTADOS EXPERIMENTALES

En esta sección, nuestro algoritmo es comparado respecto a los dos algoritmos que más se han citado recientemente (gSpan y Gaston). Estos últimos fueron implementados en el paquete ParMol con el fin de realizar un estudio comparativo [5].

El algoritmo gdFil fue implementado también en Java, utilizando las mismas bibliotecas para el manejo de grafos del paquete ParMol.

Todos los experimentos fueron ejecutados en una PC con procesador Intel Core 2 Duo de 2.2 GHz con 4 GB de RAM usando Linux-Debian a 64-bits. Se utilizó la máquina virtual de Java de IBM Versión 6.

Para evaluar el desempeño de estos algoritmos se utilizaron las colecciones de grafos CAN2DA99[i] y HIV[ii].

La Fig. 1 muestra el desempeño de los algoritmos para diferentes valores del umbral de frecuencia. En los casos evaluados, gdFil obtuvo los menores tiempos de ejecución.

VI CONCLUSIONES

En este trabajo se presentaron cinco nuevas propiedades de los códigos DFS las cuales pueden ser usadas para acelerar el proceso de detección de duplicados en la minería de SCF. Estas propiedades permiten definir fronteras (cortes) entre los candidatos útiles y los duplicados durante el recorrido del espacio de búsqueda.

Un nuevo algoritmo (gdFil) para la minería de SCF fue diseñado utilizando estas nuevas propiedades. Este algoritmo fue comparado respecto a dos de los mejores algoritmos reportados en la literatura. La experimentación mostró que nuestro algoritmo logra el mejor desempeño.

Como trabajo futuro, se propone utilizar estas nuevas propiedades para acelerar la búsqueda en la minería de SCF cerrados y maximales.

VII AGRADECIMIENTOS

El autor le agradece al CONACyT el apoyo otorgado a través de la Beca para estudios de Doctorado # 228063.

VIII REFERENCIAS

[1] Han, J., *et al.*, "Frequent Pattern Mining: Current Status and Future Directions", Data Mining and Knowledge Discovery, Vol.15, No.1, 2007, pp. 55–86.

[2] Inokuchi, A., *et al.*, "A Fast Algorithm for Mining Frequent Connected Subgraphs", Technical Report RT0448, Tokyo Research Laboratory, 2002.

[3] Kuramochi, M., and Karypis, G.: "Frequent Subgraph Discovery", Proc. of ICDM 2001, pp. 313–320.

[4] Nijssen, S., Kok, J., "A Quickstart in Frequent Structure Mining can Make a Difference". Proc. of KDD 2004, pp. 647–352.

[5] Wörlein, M., *et al.*, "A Quantitative Comparison of the Subgraph Miners Mofa, gSpan, FFSM, and Gaston", Proc. of PKDD 2005, pp. 392–403.

[6] Yan, X., Han, J., "gSpan: Graph-Based Substructure Pattern Mining", Proc. of ICDM 2002, pp. 721–724.

[i] http://dtp.nci.nih.gov/docs/cancer/cancer_data.html

[ii] http://dtp.nci.nih.gov/docs/aids/aids_data.html