UNIVERSIDAD DE CONCEPCIÓN
Facultad de Ingeniería
Departamento de Ingeniería Informática y Ciencias de la Computación

Descubrimiento eficiente de subgrafos densos en superposición y su aplicación para la compresión de grafos

Informe de Memoria de Título para optar al título de
Ingeniero Civil Informático

Carlos Felipe Mella Ibáñez

Profesor patrocinante:
Cecilia Hernández Rivas

Miembros de la comisión:
Lilian Salinas Ayala
Diego Seco Naveiras

Junio 2016

Sumario

La exploración de algoritmos eficientes para descubrir patrones en grafos es un tema que ha recibido un renovado interés en los últimos años, debido al rápido crecimiento que se ha dado en grafos vistos en ámbitos como las redes sociales, la web, estudio de interacciones entre proteínas y muchos otros. El procesar estos grafos muy grandes es un desafío: los altos requerimientos de memoria principal pueden hacer infactible la ejecución directa de algoritmos de búsqueda de patrones, mientras que la alta complejidad computacional de algunos de estos algoritmos puede volver impráctica su aplicación. Para tratar las limitaciones de espacio y procesamiento, la comunidad científica ha propuesto, entre otras técnicas, el uso de representaciones compactas para grafos que soportan la ejecución de algunas operaciones básicas de consulta directamente sobre la estructura comprimida; a partir de estas operaciones básicas es posible implementar algoritmos clásicos de grafos. Esto permite relajar los requerimientos de memoria principal para su ejecución.

En particular, en grafos de la web y redes sociales, Hernández y Navarro propusieron una estructura compacta para grafos basada en subgrafos densos que incluye una combinación de cliques y bicliques, que son descubiertos utilizando heurísticas eficientes. Estos subgrafos no tienen superposición de aristas entre ellos.

Esta memoria de título presenta una propuesta para expandir las herramientas de descubrimiento de Hernández y Navarro para incluir subgrafos con superposición entre ellos. Los resultados obtenidos usando grafos reales de la web y redes sociales muestran un incremento del número, tamaño promedio y cobertura en el grafo de los subgrafos densos con respecto a los resultados obtenidos con las herramientas de Hernández y Navarro.

El segundo objetivo principal de este trabajo es el diseño e implementación de una nueva representación compacta para grafos basada en subgrafos densos con superposición. Dos representaciones compactas fueron creadas: una construida a partir de los cliques maximales de un grafo no dirigido, y una segunda para grafos dirigidos y no dirigidos a partir de sus subgrafos densos maximales. En los resultados obtenidos destaca la obtención de una mejor tasa de compresión que las técnicas de referencia actuales con el grafo social no dirigido dblp-2011.

Índice general

Índice de tablas

Índice de figuras

Índice de algoritmos

Capítulo 1

Introducción

La exploración de algoritmos eficientes para descubrir patrones en grafos es un tema que ha recibido un renovado interés en los últimos años, debido al rápido crecimiento que se ha dado en grafos vistos en ámbitos como las redes sociales, la web, estudio de interacciones entre proteínas y muchos otros. Los tamaños de estos grafos están creciendo a una razón sin precedentes: por ejemplo, el número de sitios indexados por los principales motores de búsqueda en la web se estima actualmente en al menos 4,6 miles de millones.[1] Lograr descubrir patrones en estos grafos enormes tiene múltiples aplicaciones, como es la identificación de comunidades y actores relevantes en redes sociales, o el descubrimiento de patrones de *link spam* en la web [3], relevante para mejorar los algoritmos de *ranking* utilizados por los motores de búsqueda en la web, como es *PageRank* [8] de Google.

El procesar estos grafos muy grandes para descubrir patrones es un desafío: los altos requerimientos de memoria principal pueden hacer infactible la ejecución directa de algoritmos de búsqueda de patrones, mientras que la alta complejidad computacional de algunos de estos algoritmos puede volver impráctica su aplicación.

Para tratar las limitaciones de espacio y procesamiento, la comunidad científica ha propuesto, entre otras técnicas, el uso de representaciones compactas para grafos que soportan la ejecución de algunas operaciones básicas de consulta directamente sobre la estructura comprimida [2, 5, 10, 13, 21]; a partir de estas operaciones básicas es posible implementar algoritmos clásicos de grafos. Esto permite relajar los requerimientos de memoria principal para su ejecución.

En particular, en grafos de la web y redes sociales, las representaciones compactas existentes aprovechan propiedades como la similitud entre las listas de adyacencia, la localidad de referencia y la asignación de identificadores a los vértices del grafo según la presencia de *clusters* [5] u otras maneras de recorrer el grafo [2]. Otras estrategias aprovechan que la matriz de adyacencia es dispersa [10]. Finalmente, hay propuestas que buscan subcomponentes densos —subgrafos donde el número de aristas no está lejos del máximo posible— a partir de estructuras tales como cliques maximales, quasi-cliques y otros patrones densos [13, 33, 21, 25]. En particular, Hernández y Navarro [21] proponen una estructura compacta para grafos basada en subgrafos densos que incluye una combinación de cliques y bicliques, que son descubiertos utilizando heurísticas eficientes. Estos subgrafos densos tienen la característica que no se superponen, es decir, no comparten aristas entre sí.

La propuesta de este trabajo es extender el descubrimiento de subgrafos densos presentada en [21] para incluir subgrafos densos superpuestos. Esto incluye su diseño e implementación, así como la

[1] http://www.worldwidewebsize.com/, consultado el 6 de noviembre de 2015.

evaluación de la eficiencia de los algoritmos de descubrimiento y la calidad de los resultados. Se desea que tal implementación sea estructurada como una biblioteca de *software* que permita un desarrollo rápido para evaluar posibles trabajos derivados o aplicaciones en otros contextos de estudio. Finalmente el trabajo contempla el diseño, implementación y evaluación de una estructura comprimida que busca aprovechar la alta superposición entre estos patrones descubiertos.

1.1. Objetivos

Este trabajo contempla 3 objetivos principales:

1. **Diseñar, implementar y evaluar un algoritmo de minería eficiente para el descubrimiento de subgrafos densos superpuestos en grafos grandes de la web y redes sociales.**

   Se busca extender el planteamiento de Hernández y Navarro [21] para incluir subgrafos densos en superposición. Para la evaluación experimental se definirán y ejecutarán pruebas que permitan caracterizar los subgrafos obtenidos y compararlos con la implementación previa, incluyendo, entre otras métricas, el total de subgrafos densos encontrados; el total de vértices y aristas cubiertos en el grafo; y métricas específicas que proporcionen una idea de la superposición entre los subgrafos. Para la evaluación con conjuntos de datos reales se utilizarán grafos públicamente accesibles, como los proporcionados por *The Laboratory for Web Algorithmics* (LAW)[2] [4, 6].

2. **Estructurar las herramientas de descubrimiento de subgrafos densos superpuestos como una biblioteca de *software* que permita su aplicación en grafos provenientes de otros contextos.**

   Con esto se busca facilitar al máximo el desarrollo rápido de aplicaciones que utilicen las herramientas de descubrimiento de subgrafos densos, ya sea tanto para la exploración y estudio de nuevas ideas que permitan extender la misma biblioteca de software, como también para la utilización de estas técnicas en otros contextos similares donde el descubrimiento de patrones densos superpuestos en grafos sea de interés, como lo es, por ejemplo, la detección de complejos proteicos en redes de interacciones entre proteínas [1, 17, 29, 32].

3. **Diseñar, implementar y evaluar una nueva estructura de datos compacta para el almacenamiento de grafos con un alto número de subgrafos densos superpuestos.**

   Dados un conjunto de subgrafos densos superpuestos y el remanente del grafo del que se extrajo tal conjunto, se diseñará una estructura de datos compacta que pueda aprovechar tanto la superposición entre subgrafos densos como otras posibles características particulares de los subgrafos densos descubiertos. Para su implementación se usarán representaciones de mapas de bits y secuencias soportadas por herramientas de software públicamente disponibles para estructuras de datos compactas, como es la *Succinct Data Structure Library* (SDSL)[3] [19].

   Su evaluación incluye el consumo de espacio de las nuevas estructuras, tiempo de ejecución necesario para su creación y tiempo de ejecución de consultas sobre la estructura compacta.

[2] http://law.di.unimi.it/datasets.php
[3] https://github.com/simongog/sdsl-lite

1.2. Estructura del informe

Este informe de memoria de título está estructurado de la siguiente forma: el Capítulo 2 contiene breves definiciones de algunos conceptos matemáticos que son importantes para comprender los capítulos siguientes, además de exponer ciertas convenciones para la notación utilizada. El Capítulo 3 comienza con una descripción general de la propuesta existente hecha por [21] y la presentación breve de algunas de sus limitaciones que terminaron motivando la realización de este trabajo. El resto del capítulo presenta los fundamentos matemáticos y algoritmos creados como parte de esta nueva propuesta para el descubrimiento de subgrafos densos superpuestos.

En el Capítulo 4 se presentan los fundamentos y algoritmos creados para una representación compacta de grafos, que busca aprovechar la presencia de cliques o subgrafos densos altamente superpuestos entre sí.

En el Capítulo 5 se presentan los resultados de diversos experimentos realizados utilizando grafos reales de la web y redes sociales reales para la evaluación de las herramientas creadas para el descubrimiento y compresión. Finalmente, en el Capítulo 6 se resumen las conclusiones de este trabajo.

Dos apéndices son incluidos en este informe: el Apéndice A contiene una descripción más técnica acerca de los requerimientos y estructuración de las herramientas del proyecto en una biblioteca de software, facilitando el prototipado rápido de nuevas aplicaciones que hacen uso de las herramientas de descubrimiento, y cómo se buscó asegurar su correctitud. En el Apéndice B se listan las presentaciones dictadas relacionadas y nuevos proyectos surgidos a partir de este trabajo durante su desarrollo.

Capítulo 2

Definiciones y convenciones

En este capítulo se definen de manera breve diversos conceptos básicos que son reiteradamente referidos a lo largo de este trabajo, junto con ciertas convenciones escogidas para su notación.

Para algunos términos importantes, su primera aparición será seguida de inmediato por el término correspondiente en inglés utilizado en el código fuente y documentación de las aplicaciones de *software* desarrolladas en este proyecto.

2.1. Orden total y conjunto totalmente ordenado

Dado un conjunto no vacío S, la relación binaria $R \subseteq S \times S$ es una *relación de orden* si cumple las propiedades de reflexividad, antisimetría y transitividad. Si, además, todos los elementos de S son comparables entre sí —es decir, $\forall x, y \in S : xRy \vee yRx$— entonces R es una *relación de orden total*, o simplemente un *orden total*, y se dice que el par (S, R) es un *conjunto totalmente ordenado*.

Dos elementos x, y de un conjunto totalmente ordenado (S, R) son *consecutivos* si $xRy \ \wedge \ (\nexists z \in S : xRz \wedge zRy)$.

2.2. Grafo

Un *grafo* $G = (V, E)$ está compuesto por un conjunto finito V de *vértices* y un conjunto $E \subseteq V \times V$ de *aristas*. Dado G, la expresión $V(G)$ es frecuentemente usada para referirse al conjunto de sus vértices y $E(G)$ al conjunto de sus aristas. El *orden* de un grafo corresponde al total de sus vértices $|V(G)|$, mientras que el *tamaño* de un grafo corresponde al total de sus aristas.

Dos vértices $v_1, v_2 \in V(G)$ son llamados *adyacentes* o *vecinos* si $(v_1, v_2) \in E(G)$. En este caso v_2 es llamado el *vecino directo* (*outlink* en inglés) de v_1, mientras que v_1 es el *vecino inverso* (*inlink*) de v_2. El *grado de salida* (*outdegree*) de un vértice v corresponde al total de sus vecinos directos, mientras que el *grado de entrada* (*indegree*) de v es el total de sus vecinos inversos.

La *lista de adyacencia* de un vértice $v \in V(G)$ es una colección ordenada con todos los vecinos directos de v en G. La expresión $adjlist_\phi(v)$ denota a la lista de adyacencia de v ordenada usando un orden total $\phi \subseteq V \times V$. Dado que el grado de entrada de un vértice se corresponde al número de

apariciones de éste en todas las listas de adyacencia, el término *frecuencia de aparición*, o simplemente *frecuencia*, es utilizado frecuentemente como sinónimo de grado de entrada.

Un *bucle* (*self-loop*) es toda arista con la forma (v, v).

Nótese que toda arista conlleva un sentido: $(v_1, v_2) \in E(G) \nRightarrow (v_2, v_1) \in E(G)$. *Grafo dirigido* es otra forma utilizada en este documento para referirse a un grafo general G. Un grafo referido como *no dirigido* satisface $\forall v_1, v_2 \in V(G) : (v_1, v_2) \in E(G) \iff (v_2, v_1) \in E(G)$.

Rutas y ciclos

Una *ruta* en G corresponde a una secuencia ordenada de vértices $(v_1, v_2, \ldots, v_n)$ tal que $(v_i, v_{i+1}) \in E(G)$, con $i = 1, \ldots, n-1$. El *largo* de una ruta corresponde al valor n. Un *ciclo* es una ruta que comienza y termina en un mismo vértice. Un *grafo acíclico* es un grafo que no contiene ciclos.

Cliques y bicliques

Un *clique* es un grafo donde todos los vértices son adyacentes entre sí, es decir, $\forall v_1, v_2 \in V(G) : (v_1, v_2) \in E(G)$. En ocasiones, un clique es presentado simplemente listando sus vértices.

Un *grafo bipartito* G es un grafo cuyos vértices pueden ser repartidos en dos subconjuntos disjuntos V_1 y V_2, tal que no existen aristas entre vértices pertenecientes al mismo subconjunto: $\forall v_1, v_2 \in V_1 : (v_1, v_2) \notin E(G)$ y $\forall v_3, v_4 \in V_2 : (v_3, v_4) \notin E(G)$. Un *grafo bipartito completo*, también conocido como *biclique*, es un grafo bipartito donde todo vértice en V_1 es adyacente a todo vértice en V_2.

Grafo ponderado

Un *grafo con pesos*, o *grafo ponderado*, es un grafo de la forma $G = (V, E, w)$, donde w es una función $w : E \to \mathbb{R}$ que asigna un valor —o «peso»— a cada arista del grafo.

2.2.1. Grafos de la web y redes sociales

Un *grafo de la web* contiene una fracción de la estructura de hipervínculos de la web existente en algún momento determinado. Es modelado como un grafo dirigido G donde $V(G)$ es un conjunto de páginas web, cada una de ellas identificada por su *URL*, y cada arista $(v_1, v_2) \in E(G)$ corresponde a un hipervínculo existente en la página web v_1 que apunta a la página v_2.

Un *grafo de red social* representa un tipo de relación existente entre entidades participantes de alguna red o una plataforma social —como son *Facebook* o *Twitter*— en algún momento determinado. $V(G)$ es el conjunto total de participantes, y cada arista $(v_1, v_2) \in E(G)$ refleja una relación entre ambas entidades. En algunos casos esta relación puede corresponder a una relación simétrica, como, por ejemplo, la relación de amistad en Facebook.

2.3. DAG

Un *DAG* (del inglés *Directed Acyclic Graph*) es un grafo dirigido sin ciclos. Al tratar con un DAG, se utiliza una terminología distinta para algunos de sus componentes y propiedades básicas de grafo: sus vértices son referidos como *nodos* y sus aristas como *arcos*, por lo que un DAG D es usualmente designado con el par (N, A) en vez de (V, E), y las expresiones $N(D)$ y $A(D)$ son usadas para referirse a sus nodos y aristas. Dos nodos adyacentes son referidos como *padre* e *hijo*, respectivamente. Un *nodo raíz* es un nodo que no tiene padre alguno, y un *nodo hoja* es uno que no tiene hijos.

Un *ordenamiento topológico* de un DAG D es un ordenamiento total $(N(D), \preceq)$ de los nodos de D tal que $\forall (u_1, u_2) \in A(D) : n_1 \preceq n_2$. Dado un DAG, siempre es posible encontrar al menos un ordenamiento topológico [15].

2.4. Subgrafo denso

Un *subgrafo* de un grafo $G = (V, E)$ es un grafo $G' = (V', E')$ con $V' \subseteq V$ y $E' \subseteq E$.

El término *grafo denso* designa un grafo con una alta concentración de aristas en comparación con su número de vértices. En los capítulos siguientes, el término *subgrafo denso* se refiere siempre a un subgrafo denso representado a través del patrón bipartito dado por la Definición 3.1. Las expresiones $S(H)$ y $C(H)$ son usadas frecuentemente para referirse a los componentes S y C de un subgrafo denso H.

Se dice que dos subgrafos densos son *superpuestos*, o que se encuentran *en superposición*, si comparten al menos una arista.

Dados los subgrafos densos $H = (S, C)$ y $H' = (S', C')$, la operación de *inclusión* se define como: $H \subseteq H' \iff S \subseteq S' \wedge C \subseteq C'$. Se dice que dos subgrafos densos H y H' son *maximales entre sí*, si $H \nsubseteq H' \wedge H' \nsubseteq H$. En un llamado *conjunto $\mathcal{H}$ de subgrafos densos maximales entre sí*, $\nexists H, H' \in \mathcal{H} : H \subseteq H'$.

2.5. Grafo de intersección

Dado un conjunto no vacío $F = \{c_1, c_2, \ldots, c_n\}$ cuyos elementos son a su vez conjuntos —llamado una *familia* de conjuntos—, el *grafo de intersección* I de F es un grafo con $V(I) = F$ y $(c_i, c_j) \in E(I) \iff i \neq j \wedge c_i \cap c_j \neq \emptyset$.

Dado que I puede ser inequívocamente construido a partir de F, en ocasiones un grafo de intersección es expresado como un conjunto simplemente listando su familia de origen.

Capítulo 3

Descubrimiento de subgrafos densos superpuestos

En este capítulo se presentan los modelos y algoritmos desarrollados para el descubrimiento —también referido como minería (*mining* en inglés)— de subgrafos densos en grafos de la web y redes sociales, creados a partir de la propuesta de Hernández y Navarro [21].

3.1. Propuesta de Hernández y Navarro para el descubrimiento de subgrafos densos

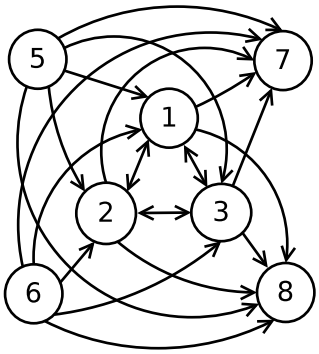
Hernández y Navarro proponen [21] una técnica escalable para el descubrimiento de patrones en grafos enormes de la web y redes sociales. Está compuesta por un algoritmo inicial de *clustering* [13], que identifica vértices con listas de adyacencia suficientemente similares y agrupa tales listas en *clusters*, seguido por un algoritmo de minería eficiente que descubre patrones de subgrafos densos a partir de cada cluster. Tales patrones de subgrafos incluyen cliques, bicliques y otros subgrafos densos más generales, pero todos ellos son descubiertos a través de un mismo patrón de grafo bipartito completo, definido como sigue:

Definición 3.1. *Un subgrafo denso $H = (S, C)$ de un grafo G es un grafo $G' = (S \cup C, S \times C)$, con $S, C \in V(G)$. Los conjuntos S y C son llamados* sources *y* centers*, respectivamente.*

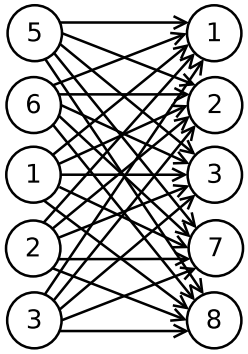
Esta definición incluye cliques (cuando $S = C$) y bicliques ($S \cap C = \emptyset$). La Figura 3.1 muestra cómo un grafo denso general puede representarse con este patrón bipartito.

3.1.1. Etapa de *clustering*

La etapa inicial de clustering es un proceso no determinístico, que usa el concepto de *shingles* [12] para representar cada lista de adyacencia con P huellas digitales (o valores *hash*), donde listas más similares entre sí tienen una mayor probabilidad de obtener las mismas huellas digitales. Estas huellas son almacenadas en una matriz de $|V(G)|$ filas y P columnas. El algoritmo, recorriendo la matriz de manera iterativa por columnas, agrupa en cada iteración i las filas con idénticas huellas digitales presentes en las primeras i columnas, y, si tal número de filas resulta ser menor a un valor de corte



(a) Representación tradicional.



(b) Representación bipartita.

Figura 3.1: Distintas representaciones de un mismo grafo denso.

preestablecido, las extrae y coloca en un cluster. Las filas remanentes después de haber evaluado la totalidad de las columnas son agrupadas en un cluster adicional.

En la Figura 3.2, los pasos 1 y 2 ejemplifican el proceso de clustering para el grafo dado, que concluye con dos clusters.

3.1.2. Etapa de descubrimiento

En esta etapa un algoritmo de minería eficiente de subgrafos densos se aplica sobre cada uno de los clusters de listas de adyacencia determinados en la etapa anterior. La etapa comienza con el ordenamiento de los elementos de las listas de adyacencia según la frecuencia de cada vértice en el cluster —el número de apariciones en las listas de adyacencia— de manera decreciente; vértices con igual frecuencia son ordenados por su identificador númerico de manera creciente. Los vértices con frequencia igual a 1 son removidos. El paso 3A de la Figura 3.2 muestra el ordenamiento resultante de las listas de adyacencia para el cluster 1, con los vértices 5 y 6 removidos.

Las listas de adyacencia ordenadas de cada cluster son entonces insertadas en un llamado árbol de prefijos (un *prefix tree*, conocido también en la literatura técnica como *trie*). Este trie tiene una estructura de árbol similar al obtenido por las técnicas de clustering propuestas por [28]. Las listas de adyacencia con un vértice inicial distinto a la raíz del trie en construcción son ignoradas, si bien el hecho de tener las listas ordenadas por frecuencia de aparición maximiza el número de listas con un mismo elemento inicial.

Todo nodo u en el trie tiene como etiqueta el identificador de un vértice, y tiene implícitamente asociado una secuencia $l(u)$ de etiquetas: las de aquellos nodos en la ruta desde la raíz hasta el propio nodo. Cada nodo también almacena un conjunto denominado *vertexSet* que contiene todo vértice del grafo cuya lista de adyacencia en el cluster comienza con la secuencia $l(u)$.

Es posible identificar un subgrafo denso $H = (S, C)$ a partir de cada nodo u en el trie resultante, tomando como S el conjunto *vertexSet* del nodo, y C el conjunto de los nodos listados en $l(u)$. El número total de arcos cubiertos por este subgrafo denso es igual a $|S| \cdot |C|$. El algoritmo de minería extrae entonces de cada trie los subgrafos densos con un mayor número de arcos.

Nótese que un nodo u no puede estar en S y C a la vez si el bucle (u, u) no está presente en el grafo. Dado que estas aristas suelen ser poco frecuentes —o aún inexistentes— en grafos de redes sociales, la

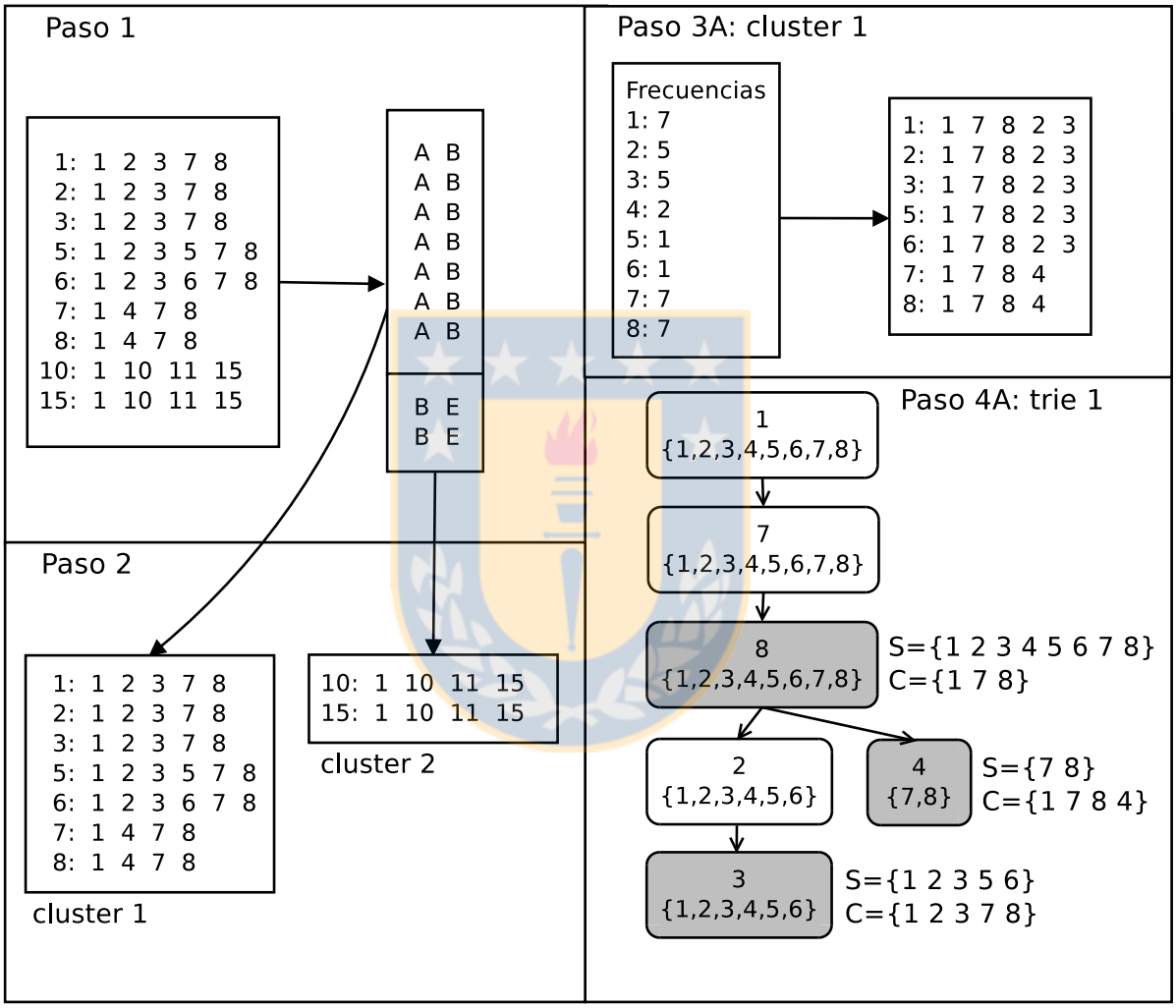


Figura 3.2: Un ejemplo del proceso de *clustering* y descubrimiento de subgrafos densos a partir de un grafo dado, según la propuesta de Hernández y Navarro.

propuesta de Hernández y Navarro realiza un procesamiento previo del grafo de entrada, donde bucles para todo vértice son añadidos al grafo, y un *bitmap* interno es generado para dejar registrado estos bucles insertados artificialmente. Esto permite al algoritmo de minería ser más sensible a la detección de subgrafos densos distintos de bicliques.

El paso 4 de la Figura 3.2 muestra el árbol de prefijos construido a partir del cluster 1 de la misma figura, acompañado por los subgrafos densos identificados a partir de 3 nodos distintos.

3.1.3. Limitaciones del algoritmo de minería

El algoritmo de minería por frecuencia propuesto por Hernández y Navarro sobre un trie cuenta con ciertas limitaciones:

1. Por la estructura misma del árbol de prefijos, donde más de un nodo puede corresponder a un mismo vértice del grafo original, el algoritmo propuesto no puede descubrir todo subgrafo denso. La Figura 3.3 muestra un grafo que incluye dos cliques altamente superpuestos: un clique entre los vértices $\{1, 2, 3, 4\}$ y un segundo clique en $\{2, 3, 4, 5\}$, además de los subgrafos densos $(\{1, 2, 3, 4, 5\}, \{2, 3, 4\})$ y $(\{2, 3, 4\}, \{1, 2, 3, 4, 5\})$. En este caso no será posible descubrir el segundo clique mencionado, independiente del *clustering* aplicado a las listas de adyacencia dadas.

2. Es posible que, al interior de un cluster, no todas las listas de adyacencia ordenadas tengan un mismo elemento inicial. Dado que se construye un árbol —con una única raíz— sobre el cual se realiza la búsqueda de subgrafos densos, las listas de adyacencia con un vértice inicial distinto al aplicado en primer lugar son, en la práctica, descartadas. El tener ordenadas las listas por frecuencia es entonces una necesidad para minimizar el número de listas a descartar.

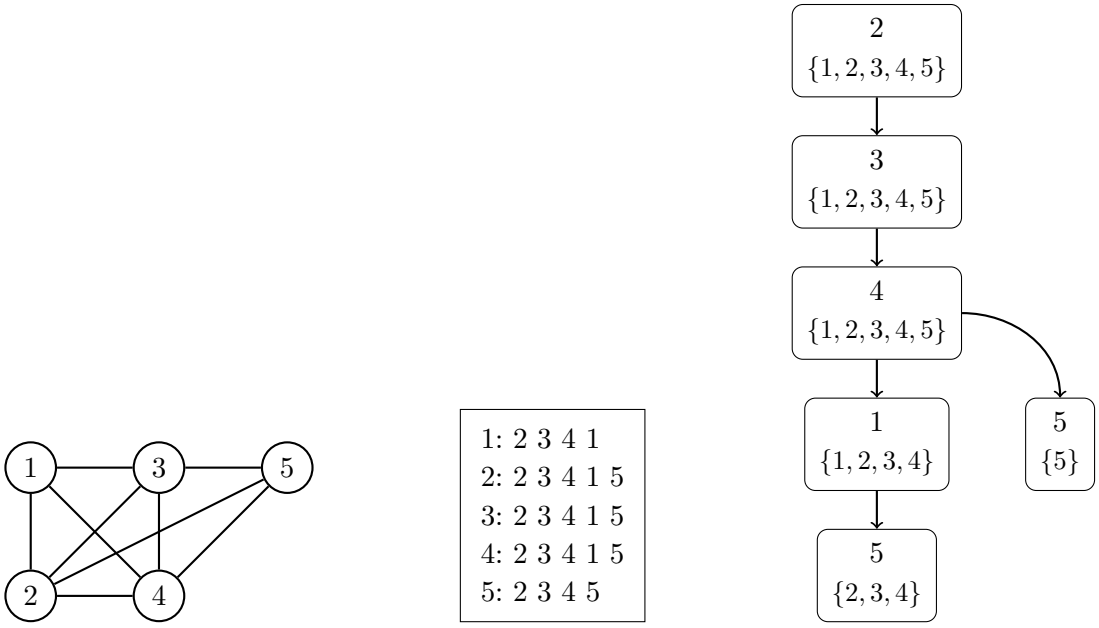
   Aún si tales listas descartadas fueran, por ejemplo, insertadas en árboles de prefijos adicionales, la capacidad global del algoritmo para encontrar subgrafos densos se ve afectada en estos casos.

3.2. Fundamentos para una nueva propuesta para el descubrimiento de subgrafos densos

El esquema general que se propone en este trabajo para expandir las técnicas propuestas por Hernández y Navarro [21] es conservar el algoritmo de clustering escalable, pero reemplazar el trie sobre el cual se ejecuta el algoritmo de minería por otra nueva estructura de datos que permita evitar las limitaciones presentadas en la sección anterior. De manera consecuente, un nuevo algoritmo eficiente de minería es también propuesto, que es compatible con esta nueva estructura de datos, y que permite el descubrimiento de subgrafos densos superpuestos con el mismo patrón bipartito dado por la Definición 3.1.

3.2.1. DAG

Es posible generalizar el árbol de prefijos usado por Hernández y Navarro, descrito en la Subsección 3.1.2, en un grafo dirigido acíclico (referido a partir de ahora como DAG):



(a) Grafo.

(b) Listas de adyacencia ordenadas por frecuencia.

(c) Arbol de prefijos.

Figura 3.3: Un ejemplo que muestra la imposibilidad del algoritmo de minería de Hernández y Navarro para detectar todos los subgrafos densos existentes: en este caso, no le es posible descubrir a partir del árbol de prefijos el clique existente entre los vértices $\{2, 3, 4, 5\}$.

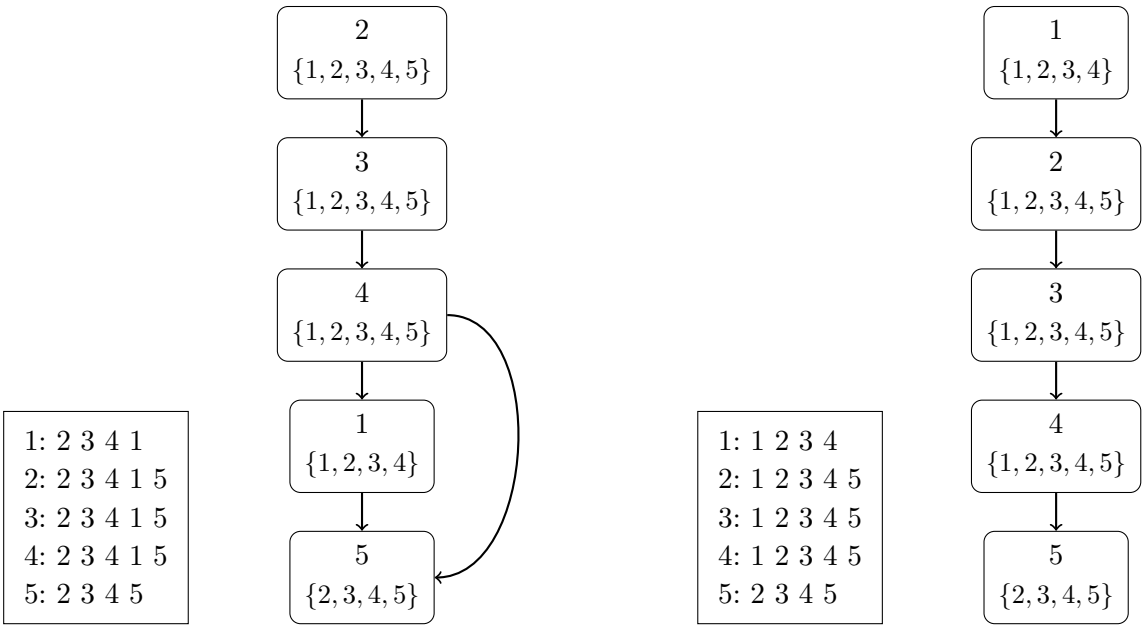
Definición 3.2. *Dado un grafo $G = (V, E)$, un conjunto $V' \subseteq V$ y un orden total $\phi \subseteq V \times V$, es posible construir un grafo dirigido acíclico $D = (N, A)$, haciendo:*

- $N = \bigcup_{v' \in V'} \{v : v \in adjlist_\phi(v')\}$.
- $A = \{(u_1, u_2) : u_1, u_2 \in N \wedge (\exists v' \in V' : u_1 \text{ y } u_2 \text{ son elementos consecutivos en } adjlist_\phi(v'))\}$.

Este DAG propuesto almacena en sus nodos los vértices en las listas de adyacencia de un conjunto $V' \subseteq V(G)$ de vértices. El uso en la definición de un conjunto V' permite la construcción de un DAG a partir tanto del propio grafo G (haciendo $V' = V$), como también a partir de un *cluster* de listas de adyacencia generado por el proceso de clustering descrito anteriormente en la Subsección 3.1.1. Los arcos del DAG corresponden a pares de vértices consecutivos en las listas de adyacencia del grafo (o cluster): una lista de adyacencia corresponde entonces a una ruta de igual largo en el DAG —ruta que, a diferencia de un trie, no necesariamente comienza en un nodo raíz—.

Dado que la Definición 3.2 no excluye vértice ni lista de adyacencia alguna para el DAG resultante, es posible ahora incluir efectivamente listas de adyacencia con distintos elementos iniciales. Esto permite eliminar el requerimiento de contar con las listas de adyacencia ordenadas específicamente por frecuencia de aparición, reemplazándolo por contar con un ordenamiento total cualquiera, representado por ϕ. Es el uso de este orden total ϕ lo que asegura que el grafo dirigido construido es realmente acíclico: de haber un ciclo en el DAG, de la forma $(u_1, u_2, \ldots, u_n, u_1)$, con $n \geq 2$, esto implicaría por definición que $u_1 \leq u_2 \leq \cdots \leq u_n \leq u_1$, lo que sólo es posible tomando $n = 1$, haciendo con ello imposible la existencia de un ciclo.

La Figura 3.4 muestra los distintos DAG resultantes a partir de dos distintos ordenamientos para las listas de adyacencia de un mismo grafo: el ordenamiento por frecuencia ya conocido y otro ordenamiento



(a) Listas de adyacencia ordenadas por frecuencia. (b) DAG. (c) Listas de adyacencia ordenadas por id. (d) DAG.

Figura 3.4: Diferentes DAG construidos a partir de dos ordenamientos distintos para las listas de adyacencia del grafo de la Figura 3.3. Nótese que el segundo DAG tiene una única raíz aún cuando las listas de adyacencia cuentan con diferentes elementos iniciales.

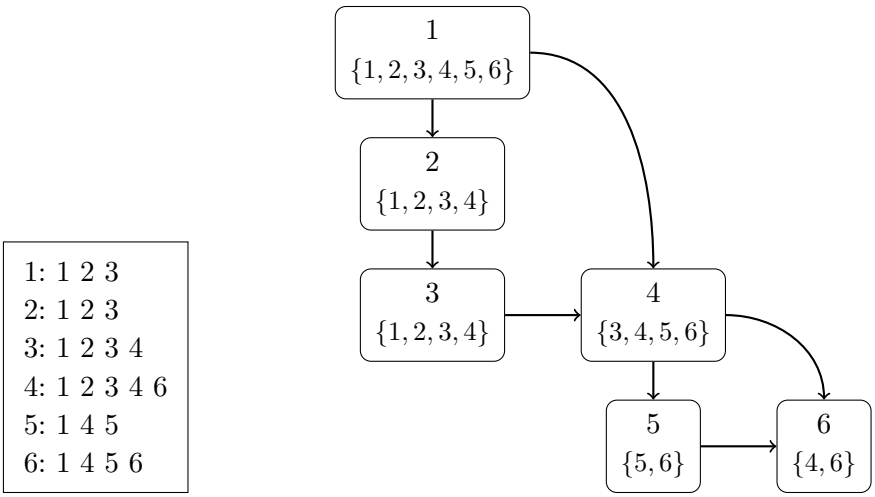
simple según el identificador de cada vértice de manera creciente. Nótese que el ordenamiento por frecuencia debe tener bien definido cómo comparar vértices de igual frecuencia —en este caso, se comparan los identificadores numéricos de los vértices—, de lo contrario no correspondería en realidad a un orden total. Ordenamientos distintos al por frecuencia tienen una menor probabilidad de ubicar un mismo elemento inicial en las listas de adyacencia, pero esto no implica necesariamente que el DAG resultante contendrá múltiples raíces, como ejemplifica el segundo DAG de la figura. En general, cada raíz resultante en un DAG corresponde a un vértice encontrado como elemento inicial en alguna lista de adyacencia, siempre y cuando este vértice no se encuentre también en una posición distinta en alguna de las otras listas.

El conjunto *vertexSet*

Análoga a la propiedad *vertexSet* de cada nodo en un trie, la propiedad *vertexSet* de cada nodo en un DAG es definida como sigue:

Definición 3.3. *Dado un DAG $D = (N, A)$ construido a partir de un grafo $G = (V, E)$, un conjunto $V' \subseteq V$ y un orden total $\phi \subseteq V \times V$, entonces $\forall u \in N, vertexSet(u) = \{v \in V' : (v, u) \in E\}$.*

El conjunto *vertexSet* de un nodo $u \in N(D)$ contiene entonces todos los vértices en V' cuyas listas de adyacencia contienen a u. Nótese que este *vertexSet* corresponde a la unión de los *vertexSets* de los distintos nodos etiquetados con el mismo vértice en el trie correspondiente, como puede verse para el vértice 5 en el trie de la Figura 3.3 y los DAG de la Figura 3.4.



(a) Listas de adyacencia ordenadas por frecuencia.

(b) DAG correspondiente.

Figura 3.5: En un DAG, ya no basta con considerar el conjunto *vertexSet* de un nodo, y las etiquetas de los nodos de una ruta desde una raíz para determinar siempre un subgrafo denso real, tal como sucedía en el árbol de prefijos propuesto por Hernández y Navarro. Aún más, ciertas rutas ya no se corresponden a listas de adyacencia reales, como la ruta $(1, 2, 3, 4, 5, 6)$.

3.2.2. Descubrimiento de subgrafos densos a partir de un DAG

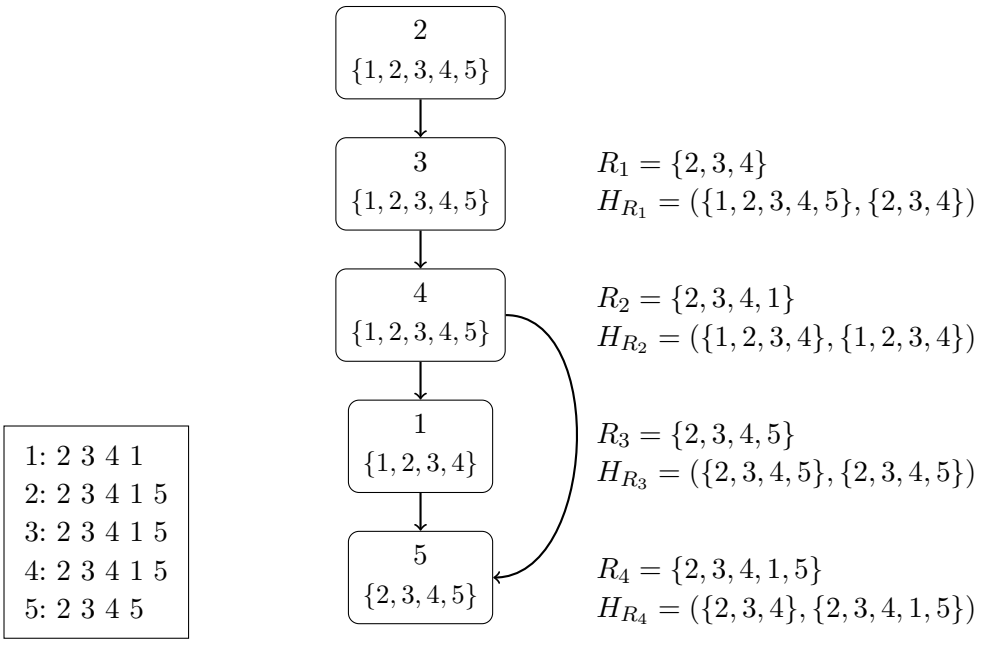
La mayor flexibilidad del DAG propuesto en la sección anterior con respecto al trie descrito por Hernández y Navarro viene con un costo: la identificación de un subgrafo denso bipartito a partir de un nodo ya no resulta ser directo como en un trie —como se describe en la Subsección 3.1.2—, debido principalmente a la existencia de múltiples rutas desde una raíz hasta el propio nodo: en la Figura 3.5, para el nodo 4 —con $\{3, 4, 5, 6\}$ como *vertexSet*— y la ruta $(1, 2, 3, 4)$, entonces $(\{3, 4, 5, 6\}, \{1, 2, 3, 4\})$ no corresponde a un subgrafo denso real.

El siguiente lema proporciona una base para la identificación de subgrafos densos en un DAG:

Lema 3.1. *Dado un DAG $D = (N, A)$, una ruta $P = (u_1, u_2, \ldots, u_n)$ sobre el DAG D, y un conjunto $R \subseteq P$, entonces es posible identificar un subgrafo denso bipartito $H_R = (S, C)$, haciendo $S = \bigcap_{u \in R} vertexSet(u)$ y $C = R$.*

Este lema declara que los vértices cuyas listas de adyacencia originales contienen a los elementos en R pueden ser determinados a partir de la intersección de los respectivos *vertexSets*. La Figura 3.6 contiene el primer DAG de la Figura 3.4 pero acompañado ahora por 4 posibles valores para R y sus respectivos subgrafos densos.

Dado que cada lista de adyacencia original se corresponde a una ruta en el DAG, la limitación dada por el Lema 3.1 para los elementos de R —de ser todos nodos presentes en una misma ruta— busca descartar una gran cantidad de casos donde sólo es posible obtener un subgrafo denso con su componente S vacío. Sin embargo, estos casos son aún posibles. La Figura 3.5 muestra un DAG que contiene una ruta $(1, 2, 3, 4, 5, 6)$ que no se corresponde con ninguna lista de adyacencia original, dado que se forma por la combinación de las rutas $(1, 2, 3, 4, 6)$ y $(1, 4, 5, 6)$, ambas correspondientes a listas de adyacencia reales. Tomando $R = \{1, 2, 3, 4, 5, 6\}$ se obtiene el subgrafo denso vacío $(\{\}, \{1, 2, 3, 4, 5, 6\})$.



(a) Listas de adyacencia ordenadas por frecuencia.

(b) DAG correspondiente y 4 subgrafos densos.

Figura 3.6: A partir del DAG construido para el grafo de la Figura 3.3, los 4 conjuntos R_i dados permiten el descubrimiento de los 4 subgrafos densos maximales entre sí existentes en el grafo.

3.3. Descubrimiento eficiente de subgrafos densos a partir de un DAG

El Lema 3.1 permite el descubrimiento de subgrafos densos a partir de un DAG; sin embargo, aún limitados a una ruta P en particular, el número de posibles valores distintos para el conjunto R es exponencial con respecto al número de nodos en la ruta. Por esto es que en esta sección se propone un heurística eficiente para el descubrimiento de subgrafos densos, que utiliza las propiedades establecidas por el lema dado. El Algoritmo 3.1 presenta el esquema general básico: se busca un único subgrafo denso a partir de cada nodo que contenga a éste en su componente C, obteniéndose entonces $|N(D)|$ subgrafos en total. Éstos son comparados y filtrados para terminar con un conjunto de subgrafos densos maximales entre sí. Esta última operación es realizada lo antes posible, apenas cada nuevo subgrafo denso es descubierto, buscando con esto reducir el número total de comparaciones.

Para el descubrimiento de un subgrafo denso a partir de cada nodo u, se busca en primer lugar determinar una ruta $P_u = (u_1, u_2, \ldots, u_n)$ tal que u_1 es un nodo raíz y $u_n = u$. Un mecanismo general que aquí se propone para determinar rutas candidatas de manera eficiente es el uso de un llamado *viajero inverso* sobre un DAG, definido como sigue:

Definición 3.4. *Dado un DAG $D = (N, A)$, un viajero inverso para D es toda función parcial $t : N \to N$, tal que $t(u)$ es padre de u en D.*

Un viajero inverso permite recorrer los nodos de un DAG —o al menos una fracción de ellos— moviéndose desde cada nodo a alguno de sus padres hasta alcanzar una raíz. Entonces, dado u, los nodos componentes de P_u pueden ser determinados a partir de la aplicación repetida de t sobre u: $u \to t(u) \to (t \circ t)(u) \to \cdots \to \varepsilon$.

Algoritmo 3.1 Esquema general básico para el descubrimiento de subgrafos densos maximales entre sí a partir de un DAG.

Entrada: Un DAG D.
Salida: Un conjunto $DSGs$ con a lo más $|N(D)|$ subgrafos densos maximales entre sí.

```
1:  función BASICMINEDENSESUBGRAPHS(D)
2:      DSGs ← ∅
3:      para todo node ∈ N(D) hacer
4:          nodeDsg ← GETDENSESUBGRAPHFROM(D, node)
5:          si nodeDsg es maximal con respecto a DSGs entonces
6:              DSGs ← DSGs − {dsg : dsg ∈ DSGs ∧ dsg ⊂ nodeDsg}
7:              DSGs ← DSGs ∪ {nodeDsg}
8:          fin si
9:      fin para
10:     devolver DSGs
11: fin función
```

Teniendo escogida una ruta P_u, se busca ahora determinar un conjunto $R_u \subseteq P_u$, con $u \in R_u$, tal que el subgrafo denso resultante sea entonces el de mayor tamaño posible o, generalizando, sea el que devuelva el mayor valor dado por una función objetivo f_{obj}, definida como:

Definición 3.5. *Una función objetivo para subgrafos densos es toda función $f_{obj} : \mathcal{H} \to \mathbb{N}_0$, donde $\mathcal{H}$ es el universo de subgrafos densos con la forma $H = (S, C)$ según la Definición 3.1.*

Para determinar qué nodos son efectivamente incluidos en R_u, se comienza con $R_u = \{u\}$ y el respectivo subgrafo denso base $(vertexSet(u), u)$. Entonces se recorre la ruta P_u en sentido inverso, y para cada nodo encontrado se evalúa si el incluirlo en el subgrafo denso formado hasta entonces mejora el valor dado por f_{obj}.

El Algoritmo 3.2, con la definición de la función GETDENSESUBGRAPHFROM referenciada en el Algoritmo 3.1, resume todo este proceder para la búsqueda de un subgrafo denso a partir de cada nodo. Aquí se asume que las funciones GETPREVIOUSNODEINPATH y ISBETTERDENSESUBGRAPH implementan internamente un viajero inverso y una función objetivo, respectivamente, que han sido establecidos con anterioridad.

La personalización del Algoritmo 3.2 a través de funciones objetivo y viajeros inversos permite adaptar el algoritmo de descubrimiento para favorecer ciertas características de los subgrafos densos obtenidos, como también explorar con facilidad nuevas ideas para mejorar la calidad de los resultados. Las Tablas 3.1 y 3.2 listan las distintas definiciones para f_{obj} y t soportadas por las herramientas de descubrimiento actualmente implementadas.

Algoritmo 3.2 Descubrimiento de un subgrafo denso a partir de un nodo dado en un DAG.

Entrada: Un DAG D, un nodo $node \in N(D)$.
Salida: Un subgrafo denso $bestDsg = (S, C)$, con $node \in C$.

```
1:  función GetDenseSubGraphFrom(D, node)
2:      bestDsg ← (node.vertexSet, {node})
3:      nextNode ← GetPreviousNodeInPath(D, node)
4:      si nextNode ≠ NULL entonces
5:          candidateDsg ← (bestDsg.S ∩ nextNode.vertexSet, bestDsg.C ∪ nextNode)
6:          si IsBetterDenseSubGraph(bestDsg, candidateDsg) entonces
7:              bestDsg ← candidateDsg
8:          fin si
9:          nextNode ← GetNextNodeInPath(D, nextNode)
10:     fin si
11:     devolver bestDsg
12: fin función
```

Tabla 3.1: Funciones objetivo implementadas para el descubrimiento de subgrafos densos.

Identificador	Definición	Observaciones
F0-CLIQ	$H(S,C) \mapsto \begin{cases} \lvert C\rvert & \text{si } C \subseteq S \\ 0 & \text{si no} \end{cases}$	Favorece el descubrimiento de cliques; no descubre bicliques.
F1-ARCS	$H(S,C) \mapsto \lvert S\rvert \cdot \lvert C\rvert$	-
F2-INTERSECT	$H(S,C) \mapsto \lvert S \cap C\rvert$	No descubre bicliques.

Tabla 3.2: Viajeros inversos implementados para el descubrimiento de subgrafos densos.

Identificador	Definición (entiéndase que p es padre de u)
T0-DEEPEST	$u \mapsto p$, tal que $maxDepth(p) = maxDepth(u) - 1$
T1-SHARING	$u \mapsto p$, tal que $\lvert u.vertexSet \cap p.vertexSet\rvert$ es maximal

3.3.1. Descubrimiento de un ordenamiento topológico en un DAG

Con el objetivo de soportar de manera eficiente el viajero inverso *T0-DEEPEST* mencionado en la Tabla 3.2, todo nodo en un DAG cuenta con una nueva propiedad auxiliar llamada *maxDepth*, definida recursivamente como sigue:

Definición 3.6. *Dado un DAG $D = (N, A)$, entonces $\forall u \in N$:*

$$maxDepth(u) = \begin{cases} 1 & \text{si } u \text{ es un nodo raíz} \\ MAX(maxDepth(p)) + 1, \text{con } (p,u) \in A & \text{en el caso contrario} \end{cases}$$

La propiedad *maxDepth* de cada nodo u corresponde básicamente al largo de la ruta más extensa existente desde algún nodo raíz hasta u, y la recursividad permite determinar su valor a partir de los valores de *maxDepth* de los padres de u.

En cualquier algoritmo para inicializar los valores de *maxDepth* para los nodos de un DAG —o, en general, cualquier nueva propiedad de un nodo definida recursivamente a partir de sus padres—, es posible reemplazar esta recursividad por un proceso iterativo si de antemano se cuenta con un ordenamiento topológico para $N(D)$, que permite listar los nodos asegurando que ningún nodo será visitado antes que alguno de sus padres. Implementaciones generales eficientes para encontrar un ordenamiento topológico en un DAG tienen un tiempo de ejecución del orden de $O(|N| + |A|)$ [15]. Sin embargo, el siguiente teorema proporciona una conveniente alternativa:

Teorema 3.1. *Dado un DAG $D = (N, A)$ construido a partir de un grafo $G = (V, E)$, un conjunto $V' \subseteq V$ y un orden total $\phi \subseteq V \times V$, entonces un ordenamiento topológico posible para D es el listado lineal de los elementos de N según el orden total ϕ.*

Los algoritmos de ordenamiento disponibles en la literatura típicamente tienen un tiempo de ejecución del orden de $O(|N| \cdot \log |N|)$. Sin embargo, la implementación desarrollada en este trabajo utiliza internamente un árbol binario de búsqueda como una estructura auxiliar para recuperar rápidamente cualquier nodo de un DAG, con ϕ como clave de comparación de los nodos. Este árbol es generado durante el proceso de construcción del DAG para reducir su tiempo de ejecución; entonces, una vez asumido el costo de construcción, un recorrido completo del árbol en profundidad *inorder* proporciona un ordenamiento topológico con un tiempo de ejecución adicional en $O(|N|)$ [15].

3.3.2. Impacto del descubrimiento propuesto en el proceso de *clustering*

Se mencionó en la Sección 3.2 el uso del mismo proceso de *clustering* utilizado por Hernández y Navarro para el particionamiento inicial del grafo de entrada, previo al descubrimiento de subgrafos densos. Dado que el proceso de descubrimiento propuesto ya no exige tener las listas de adyacencia ordenadas por frecuencia, es posible entonces evitar completamente el tener que ordenar las listas si éstas ya cuentan con un ordenamiento anterior dado por su fuente de origen. Esto permite reducir significativamente las necesidades de tiempo de ejecución para la etapa de *clustering*, de considerarse necesario. Los grafos de la web y redes sociales utilizados para los experimentos —listados en la Subsección 5.1.1— se encuentran efectivamente disponibles en más de una presentación posible.

3.3.3. Paralelismo

El Algoritmo 3.1 dado anteriormente para el descubrimiento de subgrafos densos ofrece oportunidades para ser implementado usando programación en paralelo, permitiendo con esto mejorar los tiempos de ejecución en, por ejemplo, equipos con procesadores multinúcleo, al aprovechar de mejor manera los recursos de *hardware* disponibles. El Algoritmo 3.3 presenta una primera aproximación para tal implementación en paralelo con memoria compartida: la idea básica es asignar cada ejecución de la función GetDenseSubGraphFrom a una distinta unidad de procesamiento. Dado que la variable $allDSGs$ resulta ser un recurso compartido, se debe proporcionar exclusión mutua para su manipulación.

Algoritmo 3.3 Esquema general básico para el descubrimiento en paralelo de subgrafos densos maximales entre sí a partir de un DAG.

Entrada: Un DAG D.
Salida: Un conjunto $DSGs$ con a lo más $|N(D)|$ subgrafos densos maximales entre sí.

```
1:  función basicParallelMineDenseSubGraphs(D)
2:      allDSGs ← ∅
3:      para todo node ∈ Inverted-Topological-Sorting(D) hacer en paralelo
4:          nodeDsg ← getDenseSubGraphFrom(D, node)
5:              < Comienzo de sección crítica >
6:          allDSGs ← nodeDsg
7:              < Fin de sección crítica >
8:      fin para
9:      DSGs ← ∅
10:     para todo nodeDsg ∈ allDSGs hacer
11:         si nodeDsg es maximal con respecto a DSGs entonces
12:             DSGs ← DSGs − {dsg : dsg ∈ DSGs ∧ dsg ⊂ nodeDsg}
13:             DSGs ← DSGs ∪ {nodeDsg}
14:         fin si
15:     fin para
16:     devolver DSGs
17: fin función
```

Nótese que el tiempo de ejecución para la función GetDenseSubGraphFrom es dependiente del valor de la propiedad *maxDepth* del nodo dado; nodos con un mayor *maxDepth* requerirán un mayor tiempo. Por esto, para cualquier implementación del algoritmo propuesto se recomienda el uso de un planificador que asigne las tareas hacia las distintas unidades de procesamiento de manera dinámica, asignando a cada unidad un único requerimiento a la vez, y sólo cuando ésta finalize pueda asignarle una nueva tarea entre aquellas pendientes. En particular, dado que se cuenta con un ordenamiento topológico para el DAG, en el algoritmo se propone iterar sobre los nodos del DAG según el ordenamiento topológico recorrido en sentido contrario, de manera de favorecer —mas no asegurar— el procesamiento temprano de nodos con un alto valor para *maxDepth*.

Detalles de la implementación en paralelo realizada para este trabajo son dados junto con los resultados de algunas pruebas relacionadas presentadas en la Subsección 5.1.4.

3.3.4. Descubrimiento eficiente de cliques a partir de un DAG

Es posible adaptar los algoritmos propuestos anteriormente con el objetivo de descubrir sólo cliques maximales entre sí: el Algoritmo 3.4 presenta un esquema basado en Algoritmo 3.1, donde se reutiliza getDenseSubGraphFrom para la determinación de subgrafos densos a partir de cada nodo, en conjunción con funciones objetivo específicas que favorecen el descubrimiento de subgrafos densos con alta intersección entre sus componentes S y C. Entre aquellas presentadas en la Tabla 3.1, tanto *F0-CLIQ* como *F2-INTERSECT* son aptas para ello; nótese que para *F0-CLIQ*, el componente C de cada subgrafo denso extraído ya corresponde a un clique. Para cada subgrafo denso encontrado se puede extraer un clique a partir de $S \cap C$, para finalizar aplicando el filtro ya conocido pero ahora para los cliques no maximales.

Algoritmo 3.4 Esquema general básico para el descubrimiento de cliques maximales entre sí a partir de un DAG.

Entrada: Un DAG D.
Salida: Un conjunto $CLIQs$ con a lo más $|N(D)|$ cliques maximales entre sí.

```
1:  función basicMineCliques(D)
2:      CLIQs ← ∅
3:      para todo node ∈ N(D) hacer
4:          nodeDsg ← getDenseSubGraphFrom(D, node)
5:          nodeClique ← nodeDsg.C ∩ nodeDsg.S
6:          si nodeClique es maximal con respecto a CLIQs entonces
7:              CLIQs ← CLIQs − {clique : clique ∈ CLIQs ∧ clique ⊂ nodeClique}
8:              CLIQs ← CLIQs ∪ {nodeClique}
9:          fin si
10:     fin para
11:     devolver CLIQs
12: fin función
```

Capítulo 4

Compresión de grafos a partir de sus componentes densos

En este capítulo se presentan los fundamentos y algoritmos de una representación compacta para grafos a partir de sus componentes densos. Estos componentes densos son determinados a partir de cliques o subgrafos densos que cuentan con una alta superposición de vértices entre sí. Este último problema se formula conceptualmente en base a encontrar una partición en grafos de intersección, y se presenta una heurística probabilística eficiente basada en *minwise hashing* para estimar tales componentes densos sin la necesidad de construir los respectivos grafos de intersección.

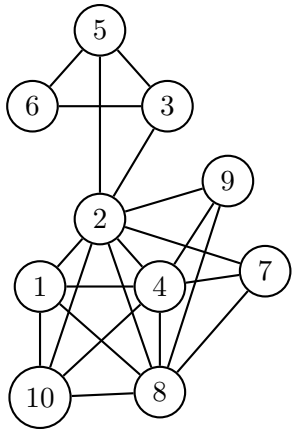
4.1. Determinación de los componentes densos de un grafo en base a grafos de intersección

4.1.1. Enumerando componentes densos a partir de cliques altamente superpuestos

Sea $K = \{c_1, c_2, \ldots, c_n\}$ un conjunto de cliques maximales entre sí presentes en un grafo no dirigido $G_u = (V, E)$ y que cubre la totalidad de las aristas de G_u —conocido como *edge clique cover*—. Esto último es posible al incluir cliques de orden 2, que permiten cubrir aristas no dirigidas individuales.

Se busca ahora enumerar todos los «componentes densos» (subgrafos densos en su acepción más general, que no necesariamente satisfacen la Definición 3.1) de G_u a partir de K. Para esto, es posible aprovechar la superposición de vértices entre pares de cliques y construir lo que se llamará un grafo de cliques. Éste es básicamente un grafo ponderado de intersección, donde cada vértice en este grafo corresponde a un clique maximal, y hay una arista entre dos vértices de este nuevo grafo si la intersección entre los vértices de los respectivos cliques es no vacía. Tal arista tiene asociado un peso, un número real proporcional a la similitud entre ambos cliques. Formalmente, se tiene:

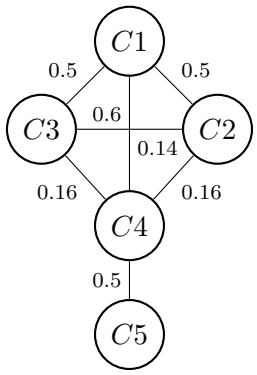
Definición 4.1. *Dado un grafo no dirigido $G_u = (V, E)$ y un conjunto $K = \{c_1, c_2, \ldots, c_n\}$ de cliques maximales entre sí que cubren las aristas de tal grafo, se define el grafo de cliques de G_u como el grafo ponderado $CG(V, E, w)$, donde $V = K$, $(c, c') \in E \iff c \neq c' \wedge V(c) \cap V(c') \neq \emptyset$, y $w((c, c')) = J(V(c), V(c'))$.*



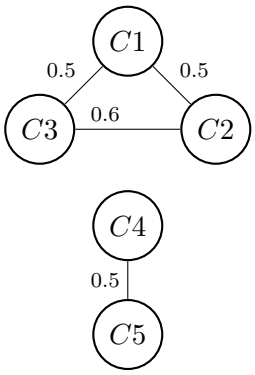
(a) Grafo no dirigido.

$C1 = \{1, 2, 4, 8, 10\}$
$C2 = \{2, 4, 8, 9\}$
$C3 = \{2, 4, 7, 8\}$
$C4 = \{2, 3, 5\}$
$C5 = \{3, 5, 6\}$

(b) Cliques maximales.



(c) Grafo de cliques.



(d) Particionamiento del grafo de cliques.

Figura 4.1: Un posible particionamiento de un grafo no dirigido en sus componentes más densos, utilizando un grafo de intersección construido a partir de los cliques maximales del grafo.

Esta definición hace uso de una función J que corresponde al coeficiente de similitud de Jaccard entre dos conjuntos dados:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{4.1}$$

La necesidad de este coeficiente de similitud será explicado en la siguiente sección.

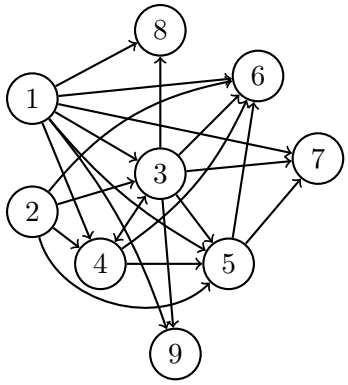
Con todo esto, es posible enumerar los componentes densos de G_u al determinar una partición sobre el respectivo grafo de cliques, de tal manera que los elementos componentes de la partición maximicen o cumplan una función de densidad dada:

Definición 4.2. *Dado un grafo de cliques $CG = (V, E, w)$ y una función objetivo de densidad f_d, el conjunto $\mathcal{CP} = \{OC_1, OC_2, \ldots, OC_m\}$, con $m \geq 1$, es una partición de CG al satisfacer $\bigcup_{1 \leq i \leq m} OC_i = CG$, $V(OC_i) \cap V(OC_j) = \emptyset$ para $i \neq j$, y OC_i es un grafo denso según f_d.*

La Figura 4.1 presenta un grafo de cliques construido a partir de todos los cliques maximales presentes en el grafo dado. También es dado un posible particionamiento sobre el grafo de cliques, permitiendo entonces representar el grafo dado a través de dos componentes densos: los subgrafos inducidos por los conjuntos de vértices $\{1, 2, 4, 7, 8, 9, 10\}$ —que son los vértices presentes en los cliques $C1$, $C2$ y $C3$— y $\{2, 3, 5, 6\}$.

4.1.2. Enumerando componentes densos a partir de subgrafos densos altamente superpuestos

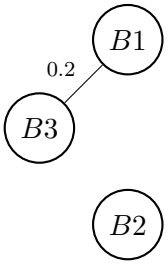
Sea $Q = \{b_1, b_2, \ldots, b_n\}$ un conjunto de subgrafos densos maximales entre sí y que satisfacen la Definición 3.1, presentes todos en un grafo $G = (V, E)$. Al considerar subgrafos densos de tamaño 1 —correspondientes a una única arista de G—, este conjunto puede formar también una cobertura de aristas en G, aún si G no es un grafo no dirigido.



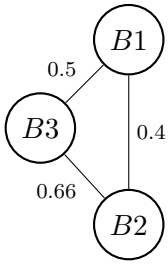
(a) Grafo.

$B1 = (\{1, 2, 3, 4\}, \{3, 4, 5, 6\})$
$B2 = (\{1, 3, 5\}, \{6, 7\})$
$B3 = (\{1, 3\}, \{8, 9\})$

(b) Subgrafos densos maximales.



(c) Grafo-C de subgrafos densos.



(d) Grafo-S de subgrafos densos.

Figura 4.2: Dos distintos tipos de grafo de intersección definidos a partir de los subgrafos densos maximales de un grafo dado.

De manera análoga a lo planteado en la subsección anterior, es posible utilizar este conjunto Q para enumerar todos los componentes densos de G, aprovechando ahora la superposición de vértices entre un componente de los subgrafos densos $b_i = (S_i, C_i)$, lo que conlleva a la definición de dos distintos grafos ponderados de intersección:

Definición 4.3. *Dado un grafo $G = (V, E)$ y un conjunto $Q = \{b_1, b_2, \ldots, b_n\}$ de subgrafos densos entre sí en G, donde $b_i = (S_i, C_i)$ cumple el patrón bipartito dado por la Definición 3.1 y Q cubre las aristas de G, se define el grafo-C de subgrafos densos de G como el grafo ponderado $BG^C(V, E, w)$, donde $V = Q$, $(b, b') \in E \iff b \neq b' \wedge C(b) \cap C(b') \neq \emptyset$, y $w((b, b')) = J(C(b), C(b'))$.*

Análogamente, el grafo-S de subgrafos densos de G es el grafo ponderado $BG^S(V, E, w)$, donde $V = Q$, $(b, b') \in E \iff b \neq b' \wedge S(b) \cap S(b') \neq \emptyset$, y $w((b, b')) = J(S(b), S(b'))$.

La Figura 4.2 presenta los dos grafos de subgrafos densos posibles de construir a partir de una colección de subgrafos densos maximales en el grafo mostrado.

Entonces es posible enumerar los componentes densos de G determinando una partición en cualquiera de sus dos grafos de subgrafos densos:

Definición 4.4. *Dado un grafo de subgrafos densos $BG = (V, E, w)$ y una función objetivo de densidad f_d, el conjunto $\mathcal{BP} = \{OB_1, OB_2, \ldots, OB_m\}$, con $m \geq 1$, es una partición de BG al satisfacer $\bigcup_{1 \leq i \leq m} OB_i = BG$, $V(OB_i) \cap V(OB_j) = \emptyset$ para $i \neq j$, y OB_i es un grafo denso según f_d.*

4.2. Estimación eficiente de los componentes densos de un grafo

Dado que la construcción de grafos de intersección puede resultar costoso en términos computacionales, y encontrar una partición óptima de estos grafos es un problema NP-completo [18], se propone como alternativa para estimar los componentes densos de un grafo el uso de una heurística probabilística eficiente basada en *minwise hashing* [11], que permite estimar similitud entre conjuntos.

4.2.1. *Minwise hashing*

La similitud entre dos conjuntos puede ser calculada usando el coeficiente de similitud de Jaccard dado por (4.1). Una alternativa es utilizar *minwise hashing*. Ésta ha sido utilizada como una técnica general para estimar similitud entre conjuntos en aplicaciones como la detección de *spam* [31] y compresión de grafos sociales y de la web por Hernández y Navarro [22]. Puede describirse como sigue: asúmase una permutación aleatoria $\pi : \Omega \to \Omega$, entonces $J(A, B)$ puede ser estimado por la probabilidad de haber una colisión:

$$J(A, B) = Pr(\ min(\pi(A)) = min(\pi(B))\ ) \tag{4.2}$$

Al considerar k permutaciones aleatorias $\pi_1, \pi_2, \ldots, \pi_k$ para cada conjunto, se tiene un llamado *sketch* [27]:

$$S_A = (\ min(\pi_1(A)), min(\pi_2(A)), \ldots, min(\pi_k(A))\ )$$
$$S_B = (\ min(\pi_1(B)), min(\pi_2(B)), \ldots, min(\pi_k(B))\ )$$
$$J_k(A, B) = \frac{|S_A \cap S_B|}{k} \tag{4.3}$$

Dado que el cómputo de permutaciones puede resultar computacionalmente costoso, es posible utilizar en su lugar funciones *hash* universales.

4.2.2. Estimación eficiente de una partición de un grafo de intersección

Asúmase que se tiene un grafo no dirigido G_u y un conjunto $K = \{c_1, c_2, \ldots, c_n\}$ de cliques maximales entre sí que cubren las aristas de G_u. Para estimar directamente una partición $\mathcal{CP} = \{OC_1, OC_2, \ldots, OC_m\}$ del grafo de cliques de G_u y K sin realmente construir tal grafo, primero se representa cada clique $c_i \in K$ por k valores *minwise hash* calculados a partir de $V(c_i)$. Entonces, cada componente denso OC_i corresponde a los cliques en K con los mismos k valores hash asociados: cliques que, con una alta probabilidad, comparten vértices.

De manera similar, dado un conjunto Q de subgrafos densos maximales entre sí que cubren las aristas de un grafo G, es posible estimar directamente una partición $\mathcal{BP} = \{OB_1, OB_2, \ldots, OB_m\}$ de un grafo de subgrafos densos, estimando cada OB_i a partir de los subgrafos densos en Q representados por un mismo conjunto de valores hash. Estos valores hash son ahora calculados a partir de cada componente S de los subgrafos o cada componente C.

En general, la probabilidad de que dos conjuntos A y B tengan asociados un mismo valor minwise hash es de $J(A, B)$. Ahora es posible justificar el uso del coeficiente de Jaccard para ponderar las aristas de los grafos de intersección dados por las Definiciones 4.1 y 4.3: el peso de cada arista refleja la probabilidad de que esos dos vértices en el grafo de intersección —correspondientes dos cliques o dos subgrafos densos— tengan asociados el mismo conjunto de valores minwise hash. Dado $CC \subseteq V(CG)$, la probabilidad de que el subgrafo inducido por CC en el grafo de intersección corresponda por sí mismo a un clique es:

$$Pr(CC \text{ es un clique}) = \prod_{c,c' \in CC} w((c, c')) \tag{4.4}$$

Entonces, la probabilidad de que $OC_i \in \mathcal{CP}$ es:

$$\begin{aligned} Pr(OC_i \in \mathcal{CP}) &= Pr(OC_i \text{ es un clique}) \cdot \prod_{CC \supset OC_i} Pr(CC \text{ no es un clique}) \\ &= Pr(OC_i \text{ es un clique}) \cdot \prod_{CC \supset OC_i} (1 - Pr(CC \text{ es un clique})) \\ &= \prod_{c,c' \in V(OC_i)} w((c,c')) \cdot \prod_{CC \supset OC_i} \left(1 - \prod_{c,c' \in CC} w((c,c'))\right) \end{aligned} \tag{4.5}$$

Este resultado proporciona un candidato para la función de densidad f_d en las definiciones de particiones de grafos de intersección dadas en la sección anterior —Definiciones 4.2 y 4.4—.

4.3. Representación compacta de grafos usando componentes densos

Esta sección presenta una representación compacta general para grafos, compatible con una partición de grafos de cliques o grafos de subgrafos densos. Esta representación general utiliza, para ambos casos, dos secuencias de símbolos y dos mapas de bits.

4.3.1. Representación compacta basada en cliques

Sea $\mathcal{CP} = \{OC_1, OC_2, \ldots, OC_m\}$ una estimación de una partición de un grafo de cliques. Se propone construir una representación compacta de $\mathcal{CP}$ a partir de dos secuencias de símbolos X e Y, junto con dos mapas de bits $B1$ y $B2$. Cada una de estas cuatro secuencias es construida por la concatenación de las respectivas subsecuencias X_r, Y_r, $B1_r$ y $B2_r$ de cada $OC_r \in \mathcal{CP}$:

$$\begin{aligned} X_r &= \bigcup_{c_i \in V(OC_r)} V(c_i) \\ Y_r &= |V(OC_r)| \\ B1_r &= 10^{|X_r|} \\ B2_r &\in \{0,1\}^{X_r} \times \{0,1\}^{Y_r} : \\ B2_{r(x,c)} &= \begin{cases} 1 & \text{si } x \in V(c), x \in X_r, c \in V(OC_r) \\ 0 & \text{en caso contrario} \end{cases} \end{aligned}$$

X_r almacena los vértices presentes en los cliques que a su vez se corresponden a los vértices de OC_r, mientras que Y_r almacena el total de tales cliques. $B1_r$ registra el largo de X_r, y $B2_r$ registra la pertenencia de cada símbolo de X_r en cada uno de los cliques. La Figura 4.3 muestra la representación compacta del grafo mostrado en la Figura 4.1 a partir de la partición del grafo de cliques dada: puede verse, por ejemplo, a partir de los 3 primeros bits de la secuencia $B2$, que el vértice 1 en X pertenece a un solo clique: evaluando los bits número 1, 4, 7, 10, 13, 16 y 19 de $B2$ para los 7 primeros símbolos de X puede determinarse que tal clique es efectivamente $C1$.

$C1 = \{1, 2, 4, 8, 10\}$
$C2 = \{2, 4, 8, 9\}$
$C3 = \{2, 4, 7, 8\}$

$C4 = \{2, 3, 5\}$
$C5 = \{3, 5, 6\}$

(a) Partición de cliques maximales.

X	1 2 4 7 8 9 10	2 3 5 6
B1	1 0 0 0 0 0 0 0	1 0 0 0 0
B2	100 111 111 001 111 010 100	10 11 11 01
Y	3	2

(b) Representación compacta.

Figura 4.3: Representación compacta para la partición del grafo de cliques dado en la Figura 4.1.

Las secuencias X, Y, $B1$ y $B2$ pueden ser implementadas, en general, utilizando estructuras de datos compactas que permiten las operaciones de consulta básicas *rank*, *select* y *access*: $rank_S(a, i)$ corresponde al total de ocurrencias del símbolo a encontradas hasta la posición i de la secuencia S; $select_S(a, i)$ retorna la posición en S de la ocurrencia número i de a; y $access_S(i)$ retorna el símbolo en la posición i de S. A partir de estas primitivas básicas es posible definir algoritmos más generales de consulta directamente sobre la representación compacta.

El Algoritmo 4.1 presenta un algoritmo de consulta de los vecinos de un vértice u en un grafo no dirigido a partir de su representación compacta. Éste comienza determinando el total de ocurrencias de u en X. Entonces se determina a qué componente denso pertenece cada ocurrencia particular de u, y entonces se determina el intervalo en $B2$ a examinar y el total de cliques de ese componente. La determinación de los cliques donde u participa, a partir de los cuales sus vecinos pueden ser listados, es realizada por la función auxiliar CLIQUES definida en el Algoritmo 4.3. El Algoritmo 4.2 inicializa la estructura de datos auxiliar I_{B2} que almacena, para cada componente OC_i, la posición en el mapa de bits $B2$ del primer elemento de tal componente.

Algoritmo 4.1 Determinación de los vecinos de un vértice en un grafo no dirigido a partir directamente de su representación compacta construida vía grafos de cliques.

Entrada: Una representación compacta $\{X, Y, B1, B2\}$ de un grafo G_u, y un vértice $u \in V(G_u)$
Salida: Un conjunto *neighbors* con los vecinos de u

```
1:  función GetNeighbors(X, Y, B1, B2, u)
2:      neighbors ← ∅
3:      I_B2 ← Indexes(X, Y, B1, B2)
4:      occurrences ← rank_X(u, |X|)
5:      para r = 1 a occurrences hacer
6:          xpos ← select_X(u, r)
7:          component ← select_B1(0, xpos) − xpos
8:          xs ← select_B1(1, component)
9:          (bs, be) ← (I_B2[component], I_B2[component + 1])
10:         ye ← access_Y(component)
11:         C_i ← Cliques(X, B2, xs, bs, be, ye)
12:         neighbors ← neighbors ∪ edges(C_i, u)
13:     fin para
14:     devolver neighbors
15: fin función
```

Algoritmo 4.2 Método auxiliar del Algoritmo 4.1 para la indexación del mapa de bits $B2$.

Salida: Índice I_{B2}

función INDEXES($X, Y, B1, B2$)
  $I_{B2}[1] \leftarrow 0$
  $ct \leftarrow rank_{B1}(1, |B1|)$
  $sX_{r-1} \leftarrow 0$
  $lj_{r-1} \leftarrow 0$
  para $r = 2$ **a** ct **hacer**
    $Y_r \leftarrow access_Y(r-1)$
    $|X_r| \leftarrow select_{B1}(1, r) - sX_{r-1}$
    $sX_{r-1} \leftarrow sX_{r-1} + |X_r|$
    $j_r \leftarrow Y_r|X_r| + lj_{r-1}$
    $lj_{r-1} \leftarrow j_r$
    $I_{B2}[r] \leftarrow j_r$
  fin para
  devolver I_{B2}
fin función

Algoritmo 4.3 Método auxiliar del Algoritmo 4.1 para determinar los cliques a los que pertenece un vértice.

Salida: Un conjunto de cliques C_i

función CLIQUES($X, B2, xs, bs, be, ye$)
  $C_i \leftarrow \emptyset$
  para $j = bs$ **a** be **hacer**
    $i \leftarrow (j - bs)\,\%ye$
    si $(access_{B2}(j) = 1)$ **entonces**
      $v \leftarrow access_X(xs)$
      $C_i \leftarrow C_i \cup \{v\}$
    fin si
    si $(i = 0 \wedge j - bs \neq 0)$ **entonces**
      $xs \leftarrow xs + 1$
    fin si
  fin para
  devolver C_i
fin función

4.3.2. Representación compacta basada en subgrafos densos

Sea ahora $\mathcal{BP} = \{OB_1, OB_2, \ldots, OB_m\}$ una estimación de una partición de un grafo-C o un grafo-S de subgrafos densos. $\mathcal{BP}$ puede también ser representado de manera compacta por dos secuencias de símbolos X e Y, y los mapas de bits $B1$ y $B2$. Estas cuatro secuencias se construyen de manera similar al caso de grafos de cliques, concatenando las respectivas subsecuencias de cada componente denso $OB_r \in \mathcal{BP}$. Para definir éstas, se necesitan los siguientes conjuntos auxiliares:

$$SS_r = \bigcup_{b_i \in V(OB_r)} S(b_i), \qquad CC_r = \bigcup_{b_i \in V(OB_r)} C(b_i)$$

$$L_r = SS_r - CC_r, \qquad M_r = SS_r \cap CC_r, \qquad R_r = CC_r - SS_r$$

Los conjuntos disjuntos L_r, M_r y R_r agrupan los vértices existentes en los subgrafos densos $b_i \in V(OB_r)$ de acuerdo a su presencia en sólo componentes S (para L_r), sólo componentes C (R_r), o en ambos (M_r). Las subsecuencias X_r, Y_r, $B1_r$ y $B2_r$ son definidas formalmente como sigue:

$$
\begin{aligned}
X_r &= L_r : M_r : R_r \\
Y_r &= |V(OB_r)| \\
B1_r &= 10^{|L_r|}10^{|M_r|}10^{|R_r|} \\
B2_r &= B2_r^L : B2_r^M : B2_r^R \\
B2_r^L, B2_r^R &\in \{0,1\}^{X_r} \times \{0,1\}^{Y_r} : \\
B2^L_{r_{(x,c)}} &= \begin{cases} 1 & x \in S(b), x \in X_r, b \in OB_r \\ 0 & \text{en caso contrario} \end{cases} \\
B2^R_{r_{(x,c)}} &= \begin{cases} 1 & x \in C(b), x \in X_r, b \in OB_r \\ 0 & \text{en caso contrario} \end{cases} \\
X_r^S &= \{x^S : x \in X_r\} \\
X_r^C &= \{x^C : x \in X_r\} \\
B2_r^M &\in \{0,1\}^{X_r^S \cup X_r^C} \times \{0,1\}^{Y_r} : \\
B2^M_{r_{(x,c)}} &= \begin{cases} b2^S(x,c), x \in X_r^S \\ b2^C(x,c), x \in X_r^C \end{cases} \\
b2^S(x,c) &= \begin{cases} 1 & x \in S(b), x \in OB_r \\ 0 & \text{en caso contrario} \end{cases} \\
b2^C(x,c) &= \begin{cases} 1 & x \in C(b), x \in OB_r \\ 0 & \text{en caso contrario} \end{cases}
\end{aligned}
$$

X_r almacena los vértices presentes en los subgrafos densos de OB_r, agrupados según su pertenencia a L_r, M_r o R_r; Y_r almacena el total de subgrafos densos; el mapa de bits $B1_r$ marca la separación en X_r entre los vértices de L_r, M_r y R_r; finalmente $B2_r$ marca la pertenencia de cada símbolo de X_r en cada uno de los subgrafos densos.

La Figura 4.4 muestra la representación compacta del grafo mostrado en la Figura 4.2 a partir de la partición del grafo-S de subgrafos densos dada. Por simplicidad se consideró que el particionamiento del grafo de intersección determinó un único componente denso.

Un algoritmo de consulta para los vecinos directos de un vértice u a partir de esta representación compacta, si bien no es dado en este documento, tiene una estructura similar a la del Algoritmo 4.1,

$B1 = (\{1,2,3,4\},\{3,4,5,6\})$
$B2 = (\{1,3,5\},\{6,7\})$
$B3 = (\{1,3\},\{8,9\})$

(a) Partición de subgrafos densos maximales.

X	1 2	3 4 5	6 7 8 9
B1	1 0 0	1 0 0 0	1 0 0 0 0
B2	111 100	111 100 100 100 010 100	110 010 001 001
Y	3		

(b) Representación compacta.

Figura 4.4: Representación compacta para el grafo-S de subgrafos densos dado en la Figura 4.2.

comenzando por determinar el número de ocurrencias de u en X, analizando $B1$ para detectar a qué componente OB_r pertenece cada ocurrencia de u, y con ello determinar el número de subgrafos densos en tal componente. Ahora, sin embargo, corresponde realizar un más extenso análisis de $B2_r$ para determinar a qué componente S o C de cada uno de los subgrafos densos pertenece u, esto a partir de la información de pertenencia de u en los conjuntos L_r, M_r y R_r. Los métodos auxiliares son también comparables a los ya dados.

Capítulo 5

Resultados

En este capítulo se presentan los resultados generales de diversos experimentos realizados para evaluar las herramientas de minería y de compresión descritas en los Capítulos 3 y 4.

Todos los experimientos se ejecutaron en un equipo con un procesador Intel Xeon E5620 @ 2.40 GHz con 8 núcleos físicos, 64 GB en memoria RAM y 256 KB en memoria caché L2.

5.1. Evaluación de las herramientas de descubrimiento de subgrafos densos

5.1.1. *Datasets*

Para la evaluación de las herramientas de descubrimiento de subgrafos densos se utilizaron grafos reales y públicamente accesibles. Se recurrió a *The Laboratory for Web Algorithmics* (LAW)[1] para obtener una selección variada de grafos de la web y sociales. La Tabla 5.1 caracteriza los grafos seleccionados. Cuatro de estos grafos fueron también utilizados por Hernández y Navarro para evaluar su propia propuesta [21]. Entre los grafos sociales seleccionados, dblp-2011 es el único grafo no dirigido.

5.1.2. Metodología de evaluación

En primer lugar se presentan los resultados de los experimentos realizados para evaluar el impacto de varias de las opciones de configuración soportadas por las herramientas de descubrimiento. Estas pruebas son limitadas a los grafos específicos mencionados en cada caso. En segundo lugar se presentan los resultados para todos los grafos considerados y listados en la Tabla 5.1, usando ahora las opciones de configuración más efectivas para cada grafo, comparando estos resultados con las herramientas de descubrimiento de Hernández y Navarro.

El procedimiento general para evaluar cada grafo consiste en, dado un tamaño mínimo *minSize* para los subgrafos densos a considerar, ejecutar los algoritmos de *clustering* y descubrimiento en repetidas ocasiones. Al finalizar cada iteración, los arcos presentes en los subgrafos densos descubiertos son extraídos del grafo dado, y este nuevo grafo resultante, de menor tamaño (llamado a partir de ahora

[1] http://law.di.unimi.it/datasets.php

Tabla 5.1: Grafos reales de redes sociales y de la web utilizados en los experimentos para evaluar las herramientas de descubrimiento.

(a) Grafos de redes sociales

Grafo	$\|V(G)\|$	$\|E(G)\|$
enron	69.244	276.143
dblp-2011	986.324	6.707.236
ljournal-2008	5.363.260	79.023.142

(b) Grafos de la web

Grafo	$\|V(G)\|$	$\|E(G)\|$
cnr-2000	325.557	3.216.152
eu-2005	862.664	19.235.140
indochina-2004	7.414.866	194.109.311

grafo remanente), es usado como entrada en la siguiente iteración. El procedimiento continúa mientras el número de subgrafos densos descubiertos en cada iteración se mantenga por encima de un valor *threshold* dado, y entonces todo este procedimiento es reiniciado usando el grafo remanente anterior y un menor valor para *minSize*. El uso de valores decrecientes para *minSize* busca reducir la posibilidad de que la extracción temprana de un subgrafo pequeño coarte la detección posterior de un subgrafo de tamaño mayor, del cual el primero resultara formar parte.

Los valores utilizados para las variables *minSizes* y *threshold* son los siguientes:

- *minSizes* es «500,100,50,30,15,6» para el descubrimiento de subgrafos densos en general, pero es «400,100,64,36,16» para el descubrimiento limitado a cliques. Nótese que estos últimos valores corresponden a cliques de un orden igual a 20, 10, 8, 6 y 4, respectivamente.
- *threshold* es «10» para los grafos enron, dblp-2011, cnr-2000 y eu-2005; y «500» para indochina-2004 y ljournal-2008.

El procedimiento finaliza entregando la colección de todos los subgrafos densos descubiertos y el grafo remanente final. Los subgrafos densos son todos maximales entre sí.

Caracterización de resultados

Los resultados obtenidos a partir de los experimentos son presentados agrupados en tres categorías: ejecución, calidad de los subgrafos densos y cobertura de tales subgrafos en el grafo:

1. Los valores bajo la categoría *Ejecución* caracterizan el procedimiento descrito en la subsección anterior: estos valores son *iteraciones* y *tiempo*. *Tiempo* corresponde al tiempo real (*clock time*) requerido por el proceso completo, medido con la aplicación `time` en Linux. Si bien esto incluye el tiempo usado por el sistema operativo para la gestión del proceso mismo (*kernel time*), se verificó en todos los casos que resultó ser órdenes de magnitud menor al tiempo total real. La decisión de usar el tiempo real fue debido al uso de paralelismo —descrito más adelante en esta sección—. Con respecto a la configuración de las ejecuciones en paralelo, se hizo uso de 14 de los 16 procesadores virtuales disponibles en el equipo de pruebas.
2. Los subgrafos densos descubiertos son caracterizados en términos de su número total, tamaño promedio (número de arcos) y tamaño del mayor subgrafo denso encontrado. Cabe mencionar que al comparar los subgrafos densos de dos ejecuciones distintas, una mayor cantidad de subgrafos densos con un tamaño promedio mucho menor puede estar reflejando el descubrimiento frecuente

de un subgrafo denso real a través de múltiples subgrafos densos superpuestos menores. El caso de *F2-INTERSECT* con respecto a *F1-ARCS* para el dataset enron en la Tabla 5.2 ejemplifica esta situación.

3. Los valores bajo la categoría *Cobertura* reflejan cuántos vértices y aristas únicos del grafo están presentes en los subgrafos densos obtenidos, valores expresados como un porcentaje del total.

5.1.3. Resultados

Comparación de funciones objetivo

La Tabla 5.2 compara las funciones objetivo listadas en la Tabla 3.1 para el descubrimiento de subgrafos densos en general. El viajero inverso *T0-DEEPEST* es utilizado en todos estos casos. Se puede ver que *F1-ARCS* resulta en general ser tanto la opción más eficiente, al comparar de manera conjunta el número de iteraciones y tiempo total, como la más eficaz, al evaluar la calidad de los subgrafos densos descubiertos y la cobertura alcanzada en el grafo.

Comparación de viajeros inversos

La Tabla 5.3 compara los viajeros inversos listados en la Tabla 3.2, para los grafos y funciones objetivo mencionados. Los datos de los casos usando *T0-DEEPEST* son los mismos ya expuestos en la Tabla 5.2. *T1-SHARING* resulta ser para cnr-2000 la opción más eficaz; para el resto de los casos los resultados son mixtos o favorables para *T0-DEEPEST*.

Evaluación general con respecto a propuesta de Hernández y Navarro

La Tabla 5.4 presenta los resultados para todos los datasets, esta vez utilizando la mejor configuracion determinada a partir de los experimentos previos. Para los 3 grafos de mayor tamaño (ljournal-2008, eu-2005 e indochina-2004), tal mejor configuración fue extrapolada a partir de los resultados con los datasets menores: se utilizó la función objetivo *F1-ARCS* para todos los casos, y el viajero inverso *T0-DEEPEST* para todos los casos con excepción de cnr-2000, para el cual se usó *T1-SHARING*.

También se presentan en la misma tabla los resultados obtenidos con las herramientas de descubrimiento de Hernández y Navarro. Dado que su implementación no soporta ejecución en paralelo, los tiempos dados corresponden a su ejecución secuencial. Tales herramientas no cuentan con opciones para personalizar la etapa de descubrimiento, si bien el hecho que su descubrimiento prioriza directamente los subgrafos densos con un mayor tamaño lo hace efectivamente comparable con *F1-ARCS*.

Los resultados muestran en todos los casos un incremento cercano al 100 % en el número total de subgrafos densos descubiertos con respecto a Hernández y Navarro, acompañado en la mayoría de los casos por un incremento en el tamaño promedio de tales subgrafos. El total de iteraciones requerido para obtener los subgrafos densos es menor, lo que refleja una mayor eficiencia por iteración del proceso de descubrimiento, si bien los tiempos de ejecución total son similares o peores que Hernández y Navarro, aún correspondiendo a ejecuciones en paralelo, debido básicamente a la mayor complejidad de los algoritmos propuestos. En todos los casos se mejoró la cobertura de vértices y aristas en el grafo en al menos unos pocos puntos porcentuales. Este leve incremento en cobertura, en conjunción con el significativo incremento del número y tamaño de los subgrafos densos, refleja un incremento en la superposición entre los subgrafos densos descubiertos.

Tabla 5.2: Capacidad de descubrimiento de subgrafos densos para las distintas funciones objetivo definidas, en 3 datasets reales de la web y redes sociales.

(a) Resultados generales.

Función objetivo	Ejecución		Subgrafos densos			Cobertura de G	
	Iteraciones	Tiempo	Total	Tamaño promedio	Tamaño máximo	Vértices (%)	Aristas (%)
Grafo enron							
F0-CLIQ	104	7m16s	4.700	14,77	216	5,78	16,60
F1-ARCS	41	2m25s	7.914	66,29	2.236	26,66	56,41
F2-INTERSECT	82	3m38s	13.118	12,22	216	11,18	28,55
Grafo dblp-2011							
F0-CLIQ	555	130m02s	437.997	15,66	14.161	78,62	73,15
F1-ARCS	446	95m47s	481.676	15,40	14.042	81,75	81,28
F2-INTERSECT	534	122m18s	476.196	15,98	13.689	79,08	73,42
Grafo cnr-2000							
F0-CLIQ	235	146m49s	47.041	48,40	76.750	47,31	49,20
F1-ARCS	119	6m26s	42.650	112,65	285.882	63,42	86,91
F2-INTERSECT	266	53m10s	76.007	51,73	136.360	50,39	62,59

(b) Desglose de los tipos de subgrafos densos descubiertos.

Función objetivo	Cliques	($C \subset S$)	Bicliques	Otros	Total
Grafo enron					
F0-CLIQ	240	4.460	—	—	4.700
F1-ARCS	1	76	4.225	3.612	7.914
F2-INTERSECT	65	1.912	—	11.141	13.118
Grafo dblp-2011					
F0-CLIQ	330.251	107.746	—	—	437.997
F1-ARCS	128.181	47.516	52.309	253.670	481.676
F2-INTERSECT	256.695	100.906	—	118.595	476.196
Grafo cnr-2000					
F0-CLIQ	18.582	28.459	—	—	47.041
F1-ARCS	1.973	5.863	12.372	22.442	42.650
F2-INTERSECT	9.541	19.592	—	46.874	76.007

Tabla 5.3: Capacidad de descubrimiento de subgrafos densos para los distintos viajeros inversos definidos, en 3 datasets reales de la web y redes sociales.

	Ejecución		Subgrafos densos			Cobertura de G	
Viajero inverso	Iteraciones	Tiempo	Total	Tamaño promedio	Tamaño máximo	Vértices (%)	Aristas (%)
enron, con F1-ARCS							
T0-DEEPEST	41	2m25s	7.914	66,29	2.236	26,66	56,41
T1-SHARING	41	0m39s	9.402	41,27	1.600	26,46	55,38
enron, con F2-INTERSECT							
T0-DEEPEST	82	3m38s	13.118	12,22	216	11,18	28,55
T1-SHARING	67	1m07s	11.268	13,63	208	11,29	29,11
dblp-2011, con F1-ARCS							
T0-DEEPEST	446	95m47s	481.676	15,40	14.042	81,75	81,28
T1-SHARING	503	103m34s	493.694	15,30	14.042	81,69	81,03
dblp-2011, con F2-INTERSECT							
T0-DEEPEST	534	122m18s	476.196	15,98	13.689	79,08	73,42
T1-SHARING	583	126m50s	484.569	15,87	13.924	79,52	74,31
cnr-2000, con F1-ARCS							
T0-DEEPEST	119	6m26s	42.650	112,65	285.882	63,42	86,91
T1-SHARING	133	7m20s	44.425	120,78	259.530	63,41	86,91
cnr-2000, con F2-INTERSECT							
T0-DEEPEST	266	53m10s	76.007	51,73	136.360	50,39	62,59
T1-SHARING	276	32m49s	73.218	61,56	141.528	51,81	69,85

Tabla 5.4: Comparación de la capacidad de descubrimiento de subgrafos densos para diversos grafos, entre la propuesta de este trabajo y la propuesta de Hernández y Navarro.

(a) Descubrimiento usando las herramientas propuestas, usando las opciones más favorables determinadas en los anteriores experimentos.

Grafo	Ejecución		Subgrafos densos			Cobertura de G	
	Iteraciones	Tiempo	Total	Tamaño promedio	Tamaño máximo	Vértices (%)	Aristas (%)
Grafos de redes sociales							
enron	41	2m25s	7.914	66,29	2.236	26,66	56,41
dblp-2011	427	79m53s	480.426	15,48	14.161	81,73	81,13
ljournal-2008	469	1497m25s	5.109.045	31,22	159.598	63,46	67,53
Grafos de la web							
cnr-2000	133	7m20s	44.425	120,78	259.530	63,41	86,91
eu-2005	268	46m59s	207.728	165,47	310.680	88,09	93,67
indochina-2004	91	474m15s	1.301.052	1921,81	26.943.308	76,02	95,60

(b) Descubrimiento usando las herramientas de Hernández y Navarro.

Grafo	Ejecución		Subgrafos densos			Cobertura de G	
	Iteraciones	Tiempo[a]	Total	Tamaño promedio	Tamaño máximo	Vértices (%)	Aristas (%)
Grafos de redes sociales							
enron	47	0m19s	3.786	35,60	2.056	25,29	48,40
dblp-2011	599	115m11	264.853	18,72	13.566	70,12	65,49
ljournal-2008	535	1286m40s	2.270.291	20,19	160.370	56,04	56,58
Grafos de la web							
cnr-2000	159	5m45s	22.465	119,48	347.362	59,59	84,58
eu-2005	381	53m43s	135.923	128,85	233.820	87,49	91,90
indochina-2004	112	132m13s	670.711	272,12	47.650.253	73,05	94,47

[a]Tiempo de ejecución secuencial, dado que paralelismo no es soportado.

Tabla 5.5: Capacidad de descubrimiento de cliques superpuestos usando la función objetivo *F0-CLIQ*, en 4 datasets reales de la web y redes sociales.

Grafo	Ejecución		Cliques			Cobertura de G	
	Iteraciones	Tiempo	Total	Tamaño promedio	Tamaño máximo	Vértices (%)	Aristas (%)
Grafos de redes sociales							
enron	15	1m22s	334	22,67	121	1,09	1,80
dblp-2011	383	108m25s	178.075	25,99	13.924	54,64	53,62
Grafos de la web							
cnr-2000	380	513m42s	70.964	10,93	6.561	24,84	16,88
eu-2005	143	828m55s	314.557	16,38	142.884	38,66	20,92

Tabla 5.6: Comparación de la capacidad de descubrimiento de cliques superpuestos de orden mayor o igual a 4 usando la función objetivo *F0-CLIQ*, con respecto a los realmente existentes en el grafo social no dirigido dblp-2011.

	Cliques			Cobertura de G	
	Total	Tamaño promedio	Tamaño máximo	Vértices (%)	Aristas (%)
F0-CLIQ	178.075	25,99	13.924	54,64	53,62
Todos los existentes	619.943	20,36	14.161	83,73	94,44

Resultados generales para cliques

La Subsección 3.3.4 presenta una propuesta para limitar las herramientas de descubrimiento a cliques maximales entre sí. En este trabajo se implementó tal propuesta con la función objetivo *F0-CLIQ*. La Tabla 5.5 presenta los mejores resultados obtenidos para los grafos mencionados. Los resultados obtenidos muestran una limitada cobertura en el grafo, que se explica principalmente por la limitada capacidad de descubrimiento propia de *F0-CLIQ* (que también se observa en el descubrimiento de subgrafos densos). El descubrimiento de cliques usando la función objetivo *F2-INTERSECT* debería proporcionar un mejor desempeño, viendo sus resultados para subgrafos densos, pero esto finalmente no fue implementado.

La Tabla 5.6 compara estos resultados contra la totalidad de los cliques maximales existentes. Estos fueron obtenidos con la herramienta de Makino y Uno —descrita en la Subsección 5.2.2— que permite listar todos los cliques maximales en grafos no dirigidos. Los datos dados son limitados al dataset dblp-2011, que es el único grafo no dirigido entre los evaluados, y a cliques de orden igual o mayor a 4. Puede observarse la diferencia significativa en la cobertura del grafo obtenida y el total de subgrafos densos.

5.1.4. Escalabilidad del descubrimiento en paralelo

La Subsección 3.3.3 describe los fundamentos para una implementación de los algoritmos de descubrimiento usando paralelismo con memoria compartida. Tal implementación paralela se hizo con la especificación *OpenMP*,[2] que permite añadir paralelismo a código fuente ya existente en C, C++ o Fortran sin necesidad de reescribirlo significativamente [16]. Este soporte puede ser opcionalmente habilitado en las aplicaciones finales creadas, usando la configuración de compilación `parallel` listada en la Tabla A.2.

Para evaluar la ganancia en tiempo de ejecución con respecto a la implementación secuencial, se utilizó el grafo web cnr-2000 y sus mejores opciones de configuración ya descritas en la subsección anterior. Se midió en primer lugar el tiempo real total de ejecución para la implementación secuencial sin paralelismo, seguido por la medición de los tiempos reales totales para la implementación en paralelo, utilizando distintos números de procesadores disponibles. El cociente entre el tiempo de ejecución secuencial T_{sec} y el tiempo de ejecución en paralelo T_{par} de cada caso proporciona el factor de mejora obtenido:

$$A_{obtenido} = \frac{T_{sec}}{T_{par}}$$

Este valor puede ser comparado con el factor de mejora teórico máximo que es posible de alcanzar, determinado a partir de la ley de Amdahl:

$$A_{teórico\_máximo} = \frac{1}{(1 - frac) + \frac{frac}{N}}$$

Donde N es el número de unidades de procesamiento y $frac$ corresponde a la fracción del tiempo de ejecución secuencial que es paralelizable: nótese que sólo la etapa de descubrimiento es efectivamente paralelizada, mientras que la etapa de *clustering* y la construcción de los DAG están implementadas de manera estrictamente secuencial. Para determinar el valor de $frac$, se utilizó la herramienta de análisis de rendimiento `gprof`[3] (a través de la configuración de compilación `profiling` listada en la Tabla A.2), que, para el caso indicado de cnr-2000, proporcionó un valor $frac \approx 0{,}85$.

La Figura 5.1 presenta los factores de mejora obtenidos para cnr-2000 haciendo uso de distintos números de procesadores, junto con los factores de mejora máximos teóricos. Hay una serie de factores que pueden explicar las diferencias entre ambos conjuntos de valores: desde un punto de vista algorítmico, el principal factor es el hecho que las tareas asignadas a las distintas unidades de procesamiento tienen distintos tiempos de ejecución. Por otra parte, la presencia de patrones frecuentes que degradan el rendimiento de aplicaciones en paralelo con memoria compartida, tal como es el *false sharing* [7], no fue investigado.

[2] http://openmp.org/wp/about-openmp/
[3] https://sourceware.org/binutils/

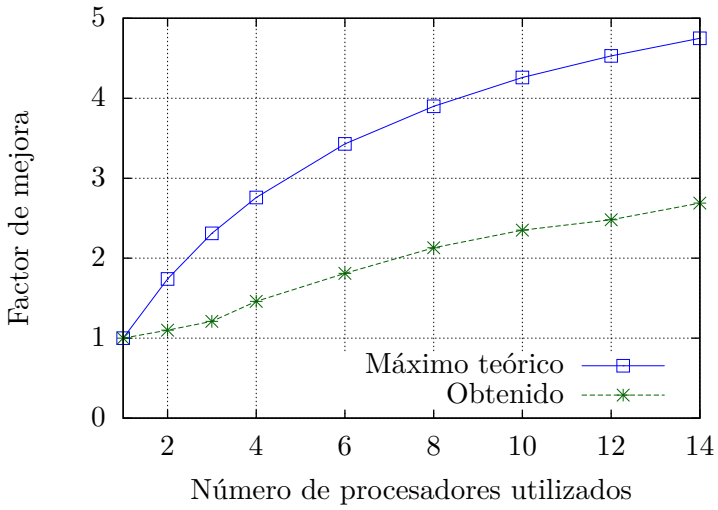


Figura 5.1: Factores de mejora obtenidos en el grafo cnr-2000 al usar la implementación en paralelo del descubrimiento de subgrafos densos.

5.2. Evaluación de las herramientas de compresión a partir de cliques y de subgrafos densos

5.2.1. *Datasets*

Acerca de los grafos reales usados ahora para la evaluación de las herramientas de compresión, para el caso de compresión a partir de subgrafos densos se utilizaron los mismos grafos obtenidos de LAW usados en las pruebas de descubrimiento y listados en la Tabla 5.1.

Para el caso de compresión a partir de grafos de cliques, se consideró una selección menor de grafos no dirigidos de redes sociales obtenidos tanto de LAW como del proyecto SNAP de la Universidad de Stanford[4]. Nótese que grafos web no son elegibles por no corresponder conceptualmente a grafos no dirigidos. Además se incluye una colección de pequeños grafos biólogicos correspondientes a redes de interacciones entre proteínas [29] disponibles desde la web de *PaccanaroLab*,[5] grafos que fueron utilizados en un contexto de trabajo relacionado —ver el Apéndice B— y que, siendo todos grafos no dirigidos, se incluyeron para complementar estas pruebas. La Tabla 5.7 caracteriza estos dos conjuntos de grafos.

5.2.2. Metodología de evaluación

Los modelos propuestos para la representación compacta de grafos requieren en primer lugar la determinación previa de un conjunto K de cliques o un conjunto Q de subgrafos densos, todos ellos maximales entre sí, que forman una cobertura de aristas sobre el grafo dado. El cómo se determinan estos conjuntos en la práctica varía para ambos casos.

Para la determinación de una cobertura de subgrafos densos se utilizaron las herramientas de descubrimiento de este trabajo. Para todos los grafos se usó la mejor configuración determinada en los

[4] http://snap.stanford.edu/data/
[5] http://www.paccanarolab.org/clusterone/

Tabla 5.7: Grafos no dirigidos de redes sociales y biológicos utilizados en los experimentos de compresión a partir de grafos de cliques.

(a) Grafos de redes sociales

Grafo	$\|V(G)\|$	$\|E(G)\|$
dblp-2010	326.186	1.615.400
comm-amazon	334.864	1.851.744
dblp-2011	986.324	6.707.236

(b) Grafos de redes biológicas

Grafo	$\|V(G)\|$	$\|E(G)\|$
collins2007	1.622	18.148
krogan2006-core	2.708	14.246
krogan2006-extended	3.672	28.634
gavin2006	1.855	15.338
biogrid	5.640	119.496

experimentos de descubrimiento. Dado que es posible concluir la extracción de subgrafos densos con un grafo remanente no vacío, este grafo es transformado en un conjunto adicional de subgrafos densos, de manera de asegurar la cobertura total del grafo: cada vértice u y su respectiva lista de adyacencia en el grafo remanente son expresados como un subgrafo denso, donde S contiene a u y C a los elementos de la lista.

Para la determinación de una cobertura de cliques, dado los limitados resultados obtenidos por las herramientas de descubrimiento de cliques propuestas en este trabajo, se optó por no utilizar estos datos y recurrir, en cambio, a la propuesta de Makino y Uno para emunerar todos los cliques maximales existentes en grafos no dirigidos densos o dispersos [24]. Tal algoritmo se ejecuta, para el caso de enumerar cliques maximales, con un tiempo de retardo entre salidas acotado por $O(|V|+|E|)$. La herramienta MACE[6] proporciona una implementación de tal algoritmo.

Con respecto al procedimiento de compresión en sí, las secuencias de símbolos y mapas de bits descritos fueron implementados con la *Succinct Data Structure Library* (SDSL)[7], usando *wavelet matrices* para las secuencias de símbolos, y mapas de bits comprimidos con RRR [30].

Para comparar el desempeño de las herramientas desarrolladas, se consideraron algunas de las técnicas de referencia en la actualidad para grafos web y redes sociales: *WebGraph* en su versión 3.5.1, que usa LLP para el ordenamiento de los vértices [6, 5]; MP_k en su implementación mejorada por Claude y Ladra [14] a partir de la propuesta original de Maserrat y Pei [25]; k^2-*tree* con los ordenamientos LLP y BFS [9, 10]; y la propuesta de Apostolico y Drovandi [2] para la compresión a partir de BFS, usando la versión 0.2.1 de su implementación.

Caracterización de resultados

El principal resultado caracterizado en las comparativas dadas en esta sección es la capacidad —o tasa— de compresión de cada propuesta, medido por el número de bits promedio usado por arista (referido como *bpe*); éste corresponde al total de bits usados para representar el grafo dividido por el total de sus aristas. Dada la naturaleza probabilística del proceso de compresión, todos los valores de bpe dados para la propuesta de este trabajo corresponden en realidad al promedio obtenido a partir de al menos 3 ejecuciones.

[6] http://research.nii.ac.jp/~uno/code/mace.html
[7] https://github.com/simongog/sdsl-lite

También se presentan, bajo la categoría *Recuperación*, los tiempos de ejecución obtenidos para la recuperación secuencial del grafo original a partir de la representación compacta. El valor dado corresponde al tiempo total promediado por arista.

5.2.3. Resultados

Evaluación de la compresión a partir de grafos de subgrafos densos

La Tabla 5.8 muestra la capacidad de compresión obtenida en grafos web y de redes sociales, a partir del particionamiento de grafos-S y de grafos-C de subgrafos densos y usando cuatro distintos valores k de minwise hash. Los resultados muestran, en general, la obtención de mayores tasas de compresión al usar mayores valores para k: estas mejoras son leves o nulas con grafos-S, pero son significativas usando grafos-C para todos los casos con la excepción del grafo dblp-2011. Las mejores tasas de compresión en grafos sociales se obtuvieron usando grafos-C, mientras que para 2 de los 3 grafos web los mejores resultados fueron obtenidos a partir de grafos-S. Tales mejores resultados son comparados en la Tabla 5.9 con las técnicas de referencia que soportan directamente consultas de vecinos directos e inversos. Los resultados obtenidos con MP_k son dados sólo para los grafos sociales, que es con los cuales esta técnica proporciona resultados más competitivos.

En general no se obtuvieron mejores resultados con respecto a las alternativas presentadas. Nótese que estos resultados involucran indirectamente la eficacia de las herramientas de descubrimiento usadas para la determinación de los datos de entrada de las herramientas de compactación; mejoras en las primeras pueden tener un impacto en los resultados obtenidos por las segundas.

La Tabla 5.10 compara ahora los tiempos de ejecución promedio obtenidos para la recuperación secuencial de cada grafo. No se obtuvieron resultados competitivos, por lo que muestran sólo los datos obtenidos con *WebGraph*.

La comparación de los tiempos de ejecución para la consulta aleatoria de vecinos directos también entregó resultados significativamente peores que las referencias actuales; tales valores no son presentados en este documento.

Tabla 5.8: Capacidad de compresión (medida en bits por arista) obtenida en grafos web y de redes sociales a partir de particiones $\mathcal{BP}$ de grafos de subgrafos densos, usando distintos k valores *minwise hash*. Se ha resaltado las mejores tasas de compresión obtenidas para cada grafo.

	Particionamiento $\mathcal{BP}^S$				Particionamiento $\mathcal{BP}^C$			
Grafo	$k=1$	$k=2$	$k=3$	$k=4$	$k=1$	$k=2$	$k=3$	$k=4$
Grafos de la web								
cnr-2000	6,07	5,95	6,01	5,98	6,67	6,18	6,00	**5,77**
eu-2005	4,62	**4,47**	4,56	4,53	6,36	5,89	4,83	4,84
indochina-2004	3,08	3,03	3,05	**2,96**	4,29	3,48	3,30	3,26
Grafos de redes sociales								
enron	14,20	13,64	13,30	13,20	15,09	13,02	12,80	**12,76**
dblp-2011	13,52	13,57	13,53	13,48	**12,33**	12,57	12,75	12,74
ljournal-2008	18,76	18,28	18,24	18,28	19,82	17,42	**16,94**	17,51

Tabla 5.9: Comparación de las tasas de compresión obtenidas en grafos web y de redes sociales por propuestas que soportan directamente la consulta de vecinos directos e inversos.

	Particionamiento $\mathcal{BP}$		MP_k		k^2-tree	
Grafo	Parámetros	*bpe*	Parámetros	*bpe*	Parámetros	*bpe*
Grafos de la web						
cnr-2000	$\mathcal{BP}^C, k=4$	5,77			-	-
eu-2005	$\mathcal{BP}^S, k=2$	4,47			$4, 2, 5, BFS$	3,22
indochina-2004	$\mathcal{BP}^S, k=4$	2,96			$4, 2, 5, BFS$	1,23
Grafos de redes sociales						
enron	$\mathcal{BP}^C, k=4$	12,76	$k=4$	17,02	$4, 2, 5, LLP$	10,31
dblp-2011	$\mathcal{BP}^C, k=1$	12,33	$k=5$	8,48	$4, 2, 5, LLP$	9,83
ljournal-2008	$\mathcal{BP}^C, k=3$	16,94	$k=5$	13,35	$4, 2, 5, LLP$	13,63

Tabla 5.10: Comparación de los tiempos de ejecución promedio por arista para la recuperación secuencial de grafos web y de redes sociales.

	Particionamiento $\mathcal{BP}$		WebGraph	
Grafo	Parámetros	Recuperación (μs)	Parámetros	Recuperación (μs)
Grafos de la web				
cnr-2000	$\mathcal{BP}^C, k=4$	26,03	$M=3, W=8$	0,169
eu-2005	$\mathcal{BP}^S, k=2$	41,72	$M=3, W=8$	0,070
indochina-2004	$\mathcal{BP}^S, k=4$	27,00	$M=3, W=8$	0,023
Grafos de redes sociales				
enron	$\mathcal{BP}^C, k=4$	17,03	$M=3, W=8$	0.531
dblp-2011	$\mathcal{BP}^C, k=1$	11,70	$M=3, W=8$	0,126
ljournal-2008	$\mathcal{BP}^C, k=3$	20,08	$M=3, W=8$	0,057

Tabla 5.11: Capacidad de compresión (medida en bits por arista) obtenida en grafos no dirigidos de redes sociales y biológicos a partir de particiones $\mathcal{CP}$ de grafos de cliques, usando distintos k valores *minwise hash*.

(a) Grafos de redes sociales

	Particionamiento $\mathcal{CP}$		
Grafo	$k=1$	$k=2$	$k=3$
dblp-2010	6,82	7,36	7,51
comm-amazon	10,74	12,56	13,57
dblp-2011	7,35	8,09	8,42

(b) Grafos biológicos

	Particionamiento $\mathcal{CP}$		
Grafo	$k=1$	$k=2$	$k=3$
collins2007	10,41	10,50	10,83
krogan2006-core	8,83	9,58	9,91
krogan2006-extended	9,88	10,40	10,90
gavin2006	5,55	6,25	6,55
biogrid	26,36	19,96	21,47

Tabla 5.12: Comparación de las tasas de compresión obtenidas en grafos no dirigidos de redes sociales y biológicos por propuestas que soportan directamente la consulta de vecinos directos e inversos. Se ha resaltado las mejores tasas de compresión obtenidas para cada grafo.

	Particionamiento $\mathcal{CP}$		MP_k		k^2-tree		BFS	
Grafo	Parámetros	*bpe*	Parámetros	*bpe*	Parámetros	*bpe*	Parámetros	*bpe*
Grafos de redes sociales								
dblp-2010	$k=1$	6,82	$k=4$	8,88	*LLP*	7,35	$L=100$	**6,52**
com-amazon	$k=1$	10,74	$k=4$	9,56	*BFS*	10,68	$L=100$	**8,12**
dblp-2011	$k=1$	**7,35**	$k=5$	8,48	*LLP*	9,83	$L=100$	9,39
Grafos biológicos								
collins2007	$k=1$	10,41	$k=5$	4,81	*BFS*	6,23	$L=100$	**3,20**
krogan2006-core	$k=1$	8,83	$k=2$	7,89	*BFS*	14,10	$L=100$	**7,13**
krogan2006-extended	$k=1$	9,88	$k=2$	**7,73**	*BFS*	14,57	$L=100$	8,80
gavin2006	$k=1$	5,55	$k=5$	5,72	*BFS*	10,14	$L=100$	**5,09**
biogrid	$k=2$	19,96	$k=3$	**6,68**	*BFS*	13,28	$L=100$	8,70

Evaluación de la compresión a partir de grafos de cliques

La Tabla 5.11 muestra la capacidad de compresión obtenida en los grafos no dirigidos de redes sociales y biológicos, esta vez a partir de componentes densos basados en cliques maximales. Al contrario de los resultados anteriores, se observa que incrementar el número de valores hash usados proporciona cada vez peores resultados de compresión en todos los casos excepto el grafo biogrid. La Tabla 5.12 compara los resultados obtenidos con las técnicas de referencia dadas. Se logró obtener resultados competitivos en algunos casos con respecto a las alternativas presentadas, destacándose el caso de dblp-2011, con el que se obtuvo la mejor tasa de compresión entre las técnicas comparadas.

La Tabla 5.13 compara los tiempos de ejecución promedio obtenidos para la recuperación secuencial de cada grafo. No se obtuvieron resultados competitivos.

Tabla 5.13: Comparación de los tiempos de ejecución promedio por arista para la recuperación secuencial de grafos no dirigidos de redes sociales y biológicos.

Grafo	Particionamiento $\mathcal{CP}$		WebGraph + LLP	
	Parámetros	Recuperación (μs)	Parámetros	Recuperación (μs)
Grafos sociales				
dblp-2010	$k = 1$	5,31	$M = 3, W = 8$	0,323
com-amazon	$k = 1$	8,88	$M = 3, W = 8$	0,214
dblp-2011	$k = 1$	6,89	$M = 3, W = 8$	0,126
Grafos biológicos				
collins2007	$k = 1$	54,55	$M = 3, W = 8$	1,507
krogan2006-core	$k = 1$	6,31	$M = 3, W = 8$	2,158
krogan2006-extended	$k = 1$	9,08	$M = 3, W = 8$	1,422
gavin2006	$k = 1$	5,21	$M = 3, W = 8$	1,793
biogrid	$k = 2$	105,55	$M = 3, W = 8$	0,779

Capítulo 6

Conclusiones y trabajo futuro

6.1. Conclusiones

El primer objetivo establecido para este trabajo fue expandir la propuesta de Hernández y Navarro [21] para el descubrimiento de subgrafos densos en grafos de la web y redes sociales. Dado el gran tamaño de estos grafos y la alta complejidad de los algoritmos clásicos para resolver este tipo de problema, el uso de heurísticas eficientes resulta ser una necesidad. Se propuso entonces una nueva heurística con la que se busca el descubrimiento de subgrafos densos con superposición entre ellos. Ésta fue implementada mediante una aplicación desarrollada en C++ y estructurada como una biblioteca de *software* que, por una parte, permite la personalización de varias características de los subgrafos a buscar, y por otra facilitó el desarrollo rápido de una nueva aplicación para investigar la predicción de complejos proteicos a partir de redes de interacciones entre proteínas, área donde la búsqueda de este tipo de patrones densos también es de interés.

Los resultados obtenidos con datasets reales de la web y redes sociales muestran un incremento cercano al 100 % en el número total de los subgrafos densos descubiertos con respecto a la propuesta de Hernández y Navarro, acompañado en la mayoría de los casos por un incremento en el tamaño promedio de tales subgrafos. También hubo un incremento en la superposición entre los subgrafos densos descubiertos. Debido a la mayor complejidad algorítmica de la nueva heurística, los tiempos de ejecución son también significativamente más altos, por lo cual se adaptó la heurística de descubrimiento para su ejecución en paralelo usando memoria compartida. Esta implementación en paralelo proporcionó tiempos reales de ejecución competitivos con respecto a la implementación secuencial de Hernández y Navarro.

La determinación de componentes densos en un grafo fue la motivación para el último objetivo de este trabajo: el diseño e implementación de una nueva representación compacta para grafos basada en subgrafos densos con superposición. Dos representaciones compactas fueron creadas: una construida a partir de los cliques maximales de un grafo no dirigido, y una segunda para grafos dirigidos y no dirigidos a partir de sus subgrafos densos maximales. Ambas cuentan con una estructura similar, basada en 2 secuencias de símbolos y dos mapas de bits, y fueron implementadas usando una biblioteca de software de referencia para estructuras compactas.

En los resultados obtenidos con grafos reales no dirigidos destaca la obtención, con algunos grafos sociales, de tasas de compresión competitivas con varias de las técnicas de referencia actuales, destacándose el caso del grafo social dblp-2011, con el que se obtuvo la mejor tasa de compresión vista. Esto,

sin embargo, está acompañado por significativamente peores tiempos de consulta para la recuperación secuencial del grafo y acceso aleatorio.

6.2. Trabajo futuro

Entre las líneas de investigación futuras se puede mencionar en primer lugar el análisis de distintos ordenamientos para las listas de adyacencia de los grafos de entrada, de manera de maximizar ciertas características particulares de los DAG construidos: por ejemplo, un ordenamiento que específicamente minimiza el número de pares de vértices consecutivos en las listas asegura que el DAG respectivo cuenta con el mínimo número de arcos posible.

En relación a las limitaciones existentes del proceso de minería propuesto, resulta que no es posible detectar en su totalidad un subgrafo denso de gran tamaño cuyos arcos resultaron repartidos entre 2 o más clusters de listas de adyacencia. Esto puede ser abordado parcialmente con la definición de un nuevo proceso de *clustering* que permita superposición entre los clusters, como permitiría, por ejemplo, la aplicación de la técnica de *min-hashing* por bandas [23].

Con respecto al algoritmo de minería en sí, se seguirá investigando nuevos viajeros inversos y nuevas funciones objetivo que se ajusten a las necesidades específicas de futuras aplicaciones. De manera paralela a esto, se puede investigar el diseño de nuevas estructuras de datos alternativas a los viajeros inversos propuestos, así como generalizaciones del algoritmo de minería que permitan, por ejemplo, incrementar el tamaño de la vecindad inspeccionada de cada nodo en el DAG al momento de definir y recorrer rutas de minería. Todos estos casos pueden también sacar provecho de nueva información adicional disponible a partir de las aristas del grafo de entrada, soportando con esto, por ejemplo, el procesamiento de grafos ponderados.

Con respecto a las representaciones compactas para grafos propuestas, una línea de investigación futura es la definición de una representación compacta mixta que permita la compresión de ciertos componentes del grafo de entrada de manera diferente y específica a sus características particulares. Un ejemplo de esto sería el usar una representación distinta para el grafo remanente resultante al finalizar la extracción de los subgrafos densos; otra posibilidad sería el agrupar los componentes densos estimados que resultan contener un único clique, y entonces representarlos usando sólo las secuencias X y $B1$.

Apéndice A

Biblioteca de *software*

Durante el desarrollo de esta memoria de título, surgió en más de una ocasión la curiosidad por ver cómo se comportaban las herramientas para el descubrimiento de subgrafos densos, aún en desarrollo, en la resolución de problemas similares en otros contextos, para ayudar a evaluar la factibilidad de posibles futuros proyectos derivados de este trabajo.

Este capítulo presenta parte del trabajo técnico realizado para estructurar el código fuente escrito en una biblioteca de software que permitió utilizarla como una base para el desarrollo rápido de aplicaciones. Sirve también como una guía introductoria para la utilización de tales herramientas.

A.1. Requerimientos

Una biblioteca de *software* es un conjunto de recursos, estructurados de una manera modular y documentada, que permite su reutilización para el desarrollo de nuevas aplicaciones u otras bibliotecas de software más generales. Los módulos de la biblioteca deben exponer una interfaz bien definida, mientras al mismo tiempo permiten a sus usuarios (usualmente otros desarrolladores de software) abstraerse de su implementación y detalles internos.

Los principales objetivos definidos para este proyecto en relación a su estructuración como una biblioteca de software son:

1. Facilidad para evaluar y extender las herramientas en desarrollo, dada la necesidad durante las etapas iniciales de realizar trabajo exploratorio para buscar posibles nuevos enfoques favorables para las soluciones buscadas.

2. Facilidad para crear y añadir nuevas aplicaciones al proyecto que hagan uso de uno o más de las herramientas disponibles.

3. Una administración automatizada de las dependencias de compilación y construcción entre los distintos archivos fuentes y archivos objetos del proyecto, de manera que, de cumplirse ciertas condiciones mínimas, no se requiera modificar las herramientas de compilación y construcción a la hora de implementar cambios como los descritos en los dos puntos anteriores.

4. Compatibilidad con un entorno GNU/Linux estándar, y, en lo posible, minimizar las dependencias de bibliotecas o herramientas externas a la hora de construir o ejecutar las aplicaciones.

No es un objetivo deseado el integrar de manera sencilla las herramientas de descubrimiento en aplicaciones ya existentes, con su propias herramientas de compilación y desarrollo, por lo que, en la práctica, las herramientas de compilación escritas no soportan, por ejemplo, la construcción de archivos objeto y bibliotecas de enlace dinámico.

A.2. Estructura del proyecto

El proyecto de software desarrollado está compuesto básicamente por los siguientes componentes:

- La biblioteca de software, de nombre *odsg*: una colección de módulos independientes entre sí que soportan, entre otras cosas, la creación de grafos a partir de archivos de entrada y su posterior descripción, la creación de uno o múltiples DAG a partir de un grafo dado, y el descubrimiento y posterior caracterización de subgrafos densos a partir de uno o más DAG.
- Una colección de archivos fuentes que implementan múltiples aplicaciones, independientes entre sí, que hace uso de uno o más módulos de la biblioteca.
- Las herramientas de compilación y construcción del proyecto.
- Herramientas complementarias de diversa índole, tal como, por ejemplo, una aplicación para ejecutar las pruebas unitarias descritas a continuación en la Subsección A.3.2.

El código fuente de la biblioteca de software y las aplicaciones dependientes está escrito completamente en el lenguaje de programación C++, según el estándar ISO de 1998/2003. Está estructurado bajo un enfoque de programación orientado a objetos, por lo que los referidos hasta ahora como módulos de la biblioteca son en la práctica un conjunto de clases. Las herramientas de compilación y construcción fueron escritas utilizando archivos *Makefiles* compatibles con GNU Make.[1]

A.2.1. Compilación y construcción

El principal mecanismo implementado para el cumplimiento del requerimiento 3 es la clasificación automática de los archivos fuentes del proyecto a partir de su ubicación en una jerarquía de directorios predeterminada. La Tabla A.1 describe tal jerarquía. Así, por ejemplo, la creación de una nueva aplicación pasa simplemente por crear un nuevo archivo fuente en el directorio `tools/` que incluya referencias a las clases de interés disponibles en `src/`.

Las herramientas de compilación y construcción también soportan diferentes configuraciones de compilación y construcción para las aplicaciones. Cada una de éstas tiene definidas distintas opciones de compilación, y distintos archivos objeto y ejecutables binarios son generados. La Tabla A.2 lista las configuraciones soportadas. Configuraciones «basadas» en otras incluyen las características de estas otras; así, `parallel` cuenta también con todas las optimizaciones de compilación habilitadas.

[1] https://www.gnu.org/software/make/

Tabla A.1: Directorios en el proyecto que son reconocidos de manera automática y tratados de manera especial por las herramientas de compilación y construcción.

Directorio	Descripción de archivos que contiene
`bin/`	Ejecutables binarios generados por las herramientas de construcción.
`include/`	Archivos de cabeceras de bibliotecas de software desarrolladas por terceros.
`obj/`	Archivos objetos resultantes del proceso de compilación.
`src/`	Archivos fuentes pertenecientes a la biblioteca de software.
`tests/`	Archivos fuentes de las aplicaciones de pruebas unitarias para la biblioteca.
`tools/`	Archivos fuentes de aplicaciones dependientes de la biblioteca.

Tabla A.2: Configuraciones de compilación y construcción disponibles para el proyecto (biblioteca de software y aplicaciones dependientes). Para cada configuración se generan distintos archivos objeto y ejecutables binarios.

Configuración	Basada en	Descripción
`dev`	-	Macros de aserciones habilitadas; mínimas optimizaciones.
`debugging`	`dev`	Opciones de depuración habilitadas.
`release`	-	Macros de aserciones deshabilitadas; máximas optimizaciones.
`profiling`	`release`	Opciones para medir el rendimiento de las aplicaciones habilitadas usando `gprof`.
`parallel`	`release`	Ejecución en paralelo habilitada (sólo con `gcc`).

A.3. Correctitud

La correctitud es un factor clave para medir la calidad de un producto de software, y se define como la habilidad de tal producto para desempeñar sus tareas tal como ha sido declarado en su especificación [26]. Para el caso de una biblioteca de software, asumir su correctitud a la hora de desarrollar una aplicación u otra biblioteca dependiente de ésta es, en muchos casos, la única técnica realista posible de aplicar en la práctica, ya que permite la separación de preocupaciones y limita el número de problemas a considerar durante cada una de las distintas etapas del desarrollo [26].

En general, es muy difícil llegar a asegurar la correctitud de un producto de software. En caso de no poder realizarse una verificación estricta y formal, múltiples mecanismos pueden utilizarse para, en conjunto, alcanzar un significativo nivel de confianza en la correctitud del software [26]. A continuación se describen 2 de ellos utilizados para la biblioteca *odsg*.

A.3.1. Aserciones

Una aserción es un predicado insertado en el código fuente de un programa, que se espera que sea siempre cierto en tal punto del programa durante su ejecución [26]. Al añadirlas mientras se escribe el código fuente respectivo, son una poderosa ayuda para producir software que es de partida correcto, en contraste con el enfoque más tradicional de escribir y luego depurar repetidamente [26]. Estas pueden ser activadas o desactivadas de manera global, permitiendo removerlas en la versión publicada del programa. Aserciones son también una pieza importante para la documentación del código fuente, y

pueden ser utilizadas por herramientas como generadores automáticos de documentación, o analizadores estáticos de código fuente (como es *Clang analyzer*[2]) que buscan detectar errores en la lógica de un programa en las etapas de compilación y construcción.

En el lenguaje de programación C++ (y C), aserciones son usualmente implementadas vía la macro `assert` que es parte de la biblioteca estándar de tales lenguajes. Su ejecución es dependiente de otra macro `NDEBUG`, cuyo valor puede definirse condicionalmente dependiendo del objetivo seleccionado para la compilación y construcción del software.

Aserciones se utilizaron profusamente en la biblioteca *odsg*. Diversas propiedades matemáticas del DAG descrito en la Subsección 3.2.1 fueron incorporadas como aserciones. La Tabla A.2 lista los configuraciones de compilación que activan o desactivan las macros de aserciones.

A.3.2. Pruebas unitarias

Una prueba unitaria es una pieza de código fuente escrita para probar de manera automatizada el correcto funcionamiento de un módulo o componente individual de un producto de software [26]. Con la biblioteca odsg, se utilizó el *framework* multiparadigma para pruebas automatizadas *Catch*[3]. Las pruebas unitarias escritas fueron agrupadas en una aplicación separada de la biblioteca misma. Puede ser compilada y ejecutada de manera integrada junto con el resto de la aplicación, o de manera aislada.

Cada prueba unitaria consta de 1 o más aserciones. Un total de 30 métodos no triviales de clases de la biblioteca *odsg* fueron cubiertos por las pruebas unitarias escritas, con un total de 187 aserciones. El tiempo de ejecución total de las pruebas unitarias no supera el medio segundo en un equipo con recursos de hardware modestos, permitiendo su ejecución repetida y automatizada durante las etapas de compilación y construcción.

Las pruebas unitarias sólo cubren el correcto comportamiento individual de los componentes de software escritos. Para verificar la correcta integración entre estos, de una manera más cercada al uso real que se espera dar a la biblioteca de software, aplicaciones específicas fueron también escritas.

[2] http://clang-analyzer.llvm.org/scan-build.html
[3] https://github.com/philsquared/Catch

Apéndice B

Proyectos y charlas derivadas de esta memoria de título

B.1. Charlas nacionales realizadas

- Charla: «Predicción de complejos proteicos a partir de redes de interacciones entre proteínas vía subgrafos densos superpuestos»; II Workshop CeBiB «Bioinformatics and mathematical modelling in biotechnological, molecular genetics and ecophysiological applications», Santiago, Chile. Julio de 2015.
- Charla: «Detection of PPI networks by dense subgraphs»; III Workshop CeBIB «Genomics, metabolic engineering and bioinformatics in biotechnological applications», Santa Cruz, Chile. Diciembre de 2015.

B.2. Proyectos de memoria de título

- Proyecto de memoria de título para optar al título de Ingeniero Civil Informático: «Predicción automática de complejos de proteínas basado en grafos con pesos y análisis biológico», por Jaime Araya, semestre 2015-2. Universidad de Concepción, Chile.

B.3. Publicaciones en curso

- Publicación de investigación: «Protein complex detection via overlapping dense subgraphs», por Cecilia Hernández, Carlos Mella, Gonzalo Navarro, Alvaro Olivera-Nappa y Jaime Araya. A la fecha en proceso de revisión por pares en la revista *BMC Bioinformatics*.[1]
- Publicación de investigación: «Detecting and compressing graphs using intersection graph partitions», por Cecilia Hernández, Carlos Mella, Lilian Salinas y Gonzalo Navarro. A la fecha en proceso de evaluación para su incorporación en el simposio internacional SPIRE 2016.[2]

[1] https://bmcbioinformatics.biomedcentral.com/
[2] https://sites.google.com/site/spire2016jp/

Bibliografía

[1] Balázs Adamcsek, Gergely Palla, Illés J Farkas, Imre Derényi, and Tamás Vicsek. Cfinder: locating cliques and overlapping modules in biological networks. *Bioinformatics*, 22(8):1021–1023, 2006.

[2] Alberto Apostolico and Guido Drovandi. Graph compression by BFS. *Algorithms*, 2(3):1031–1044, 2009.

[3] Luca Becchetti, Carlos Castillo, Debora Donato, Ricardo A. Baeza-Yates, and Stefano Leonardi. Link analysis for Web spam detection. *ACM Transactions on the Web (TWEB)*, 2(1), 2008.

[4] Paolo Boldi, Bruno Codenotti, Massimo Santini, and Sebastiano Vigna. Ubicrawler: A scalable fully distributed web crawler. *Software: Practice & Experience*, 34(8):711–726, 2004.

[5] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. Layered label propagation: a multiresolution coordinate-free ordering for compressing social networks. In *World Wide Web (WWW)*, pages 587–596, 2011.

[6] Paolo Boldi and Sebastiano Vigna. The Webgraph framework I: compression techniques. In *World Wide Web (WWW)*, pages 595–602, 2004.

[7] William J. Bolosky and Michael L. Scott. False sharing and its effect on shared memory performance. In *USENIX Systems on USENIX Experiences with Distributed and Multiprocessor Systems - Volume 4*, Sedms'93, pages 3–3, Berkeley, CA, USA, 1993. USENIX Association.

[8] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks*, 30(1-7):107–117, 1998.

[9] Nieves R. Brisaboa, Susana Ladra, and Gonzalo Navarro. k2-trees for compact Web graph representation. In *International Symposium on String Processing and Information Retrieval (SPIRE)*, pages 18–30, 2009.

[10] Nieves R. Brisaboa, Susana Ladra, and Gonzalo Navarro. Compact representation of Web graphs with extended functionality. *Inf. Syst.*, 39:152–174, 2014.

[11] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-wise independent permutations. *J. Comput. Syst. Sci.*, 60(3):630–659, 2000.

[12] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the Web. *Computer Networks*, 29(8-13):1157–1166, 1997.

[13] Gregory Buehrer and Kumar Chellapilla. A scalable pattern mining approach to Web graph compression with communities. In *Web Search and Data Mining (WSDM)*, pages 95–106, 2008.

[14] Francisco Claude and Susana Ladra. Practical representations for Web and social graphs. In *ACM International Conference on Information and Knowledge Management (CIKM)*, pages 1185–1190, 2011.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2 edition, 2001.

[16] Leonardo Dagum and Ramesh Menon. Openmp: An industry-standard api for shared-memory programming. *IEEE Comput. Sci. Eng.*, 5(1):46–55, January 1998.

[17] Antonio Del Sol, Hirotomo Fujihashi, and Paul O'Meara. Topology of small-world networks of protein–protein complex structures. *Bioinformatics*, 21(8):1311–1315, 2005.

[18] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified np-complete problems. In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, STOC '74, pages 47–63, New York, NY, USA, 1974. ACM.

[19] Simon Gog, Timo Beller, Alistair Moffat, and Matthias Petri. From theory to practice: Plug and play with succinct data structures. In *13th International Symposium on Experimental Algorithms, (SEA 2014)*, pages 326–337, 2014.

[20] Jonathan L. Gross and Jay Yellen. *Graph Theory and Its Applications, Second Edition (Discrete Mathematics and Its Applications)*. Chapman & Hall/CRC, 2005.

[21] Cecilia Hernández and Gonzalo Navarro. Compressed representation of Web and social networks via dense subgraphs. In *International Symposium on String Processing and Information Retrieval (SPIRE)*, pages 264–276, 2012.

[22] Cecilia Hernández and Gonzalo Navarro. Compressed representations for web and social graphs. *Knowl. Inf. Syst.*, 40(2):279–313, 2014.

[23] Jure Leskovec, Anand Rajaraman, and Jeffrey D. Ullman. *Mining of Massive Datasets, 2nd Ed.* Cambridge University Press, 2014.

[24] Kazuhisa Makino and Takeaki Uno. *Algorithm Theory - SWAT 2004: 9th Scandinavian Workshop on Algorithm Theory, Humlebæk, Denmark, July 8-10, 2004. Proceedings*, chapter New Algorithms for Enumerating All Maximal Cliques, pages 260–272. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[25] Hossein Maserrat and Jian Pei. Neighbor query friendly compression of social networks. In *Special Interest Group On Knowledge Discovery and Data Mining (SIGKDD)*, pages 533–542, 2010.

[26] Bertrand Meyer. *Object-oriented Software Construction (2Nd Ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1997.

[27] Michael Mitzenmacher, Rasmus Pagh, and Ninh Pham. Efficient estimation for high similarities using odd sketches. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, pages 109–118, New York, NY, USA, 2014. ACM.

[28] Katharina Morik, Andreas Kaspari, Michael Wurst, and Marcin Skirzynski. Multi-objective frequent termset clustering. *Knowl. Inf. Syst.*, 30(3):715–738, 2012.

[29] Tamás Nepusz, Haiyuan Yu, and Alberto Paccanaro. Detecting overlapping protein complexes in protein-protein interaction networks. *Nature methods*, 9(5):471–472, 2012.

[30] Rajeev Raman, Venkatesh Raman, and S. Srinivasa Rao. Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In *Symposium on Discrete Algorithms (SODA)*, pages 233–242, 2002.

[31] Bin Shao, Haixun Wang, and Yanghua Xiao. Managing and mining large graphs: systems and implementations. In *Special Interest Group On Management of Data (SIGMOD)*, pages 589–592, 2012.

[32] Xiwei Tang, Jianxin Wang, Min Li, Yiming He, and Yi Pan. A novel algorithm for detecting protein complexes with the breadth first search. *BioMed research international*, 2014, 2014.

[33] Takeaki Uno. An efficient algorithm for solving pseudo clique enumeration problem. *Algorithmica*, 56(1):3–16, January 2010.