BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA
Facultad de Ciencias de la Computación

Sistema de Minería de Datos basado en Grafos Etiquetados para encontrar Subgrafos Conexos Frecuentes a partir de Representaciones Canónicas

Tesis presentada como requisito para obtener el título de:
Licenciado en Ciencias de la Computación

presenta

Gerardo Velázquez Solis

Asesores: Dr. Ivan Olmos Pineda
Dr. Jesús Antonio González Bernal
Puebla, Puebla, 2 de septiembre de 2014

Tesis de la Licenciatura en Ciencias de la Computación

Sistema de Minería de Datos basado en Grafos Etiquetados para encontrar Subgrafos Conexos Frecuentes a partir de Representaciones Canónicas

Gerardo Velázquez Solis

2014

Índice general

Dedicado a
mi familia.

Agradecimientos

El trabajo de tesis que ahora leen ha sido posible no solo por mi esfuerzo, sino también gracias al apoyo de personas que estuvieron a mi lado en el proceso.

Antes que alguien más, quiero agradecer a mis padres, Rosa Guadalupe Solis Hernández y Gerardo Velázquez Palafox, quienes me apoyaron y alentaron a continuar a cada instante. Sé que por momentos hubo dudas sobre el trabajo y si tendría fin, pero siempre me tuvieron confianza. Ahora les presento ésta tesis y espero que haya valido la pena la espera.

A mi hermana Mónica, gracias por aguantarme y por recordarme que no importa que tan cansados o desalineados nos veamos, la recompensa que espera por el trabajo será buena si lo que hacemos es con gusto. A mi hermana Carina, que me enseñó los frutos que puede dar el trabajo con convicción y luchar por nuestros sueños y no conformarse con lo que está solo al alcance.

Gracias a mis asesores, los Doctores Ivan Olmos Pineda y Jesús Antonio González Bernal. Sin su apoyo, consejos y sobre todo paciencia, la elaboración de éste trabajo hubiera sido difícil. Sobre todo gracias al Dr. Ivan por asesorarme éstos dos últimos años y mostrarme el poder de las técnicas aplicadas en la presente investigación.

A mis sinodales, los Doctores Manuel Martín Ortiz y David Eduardo Pinto Avendaño, gracias por dedicar parte de su tiempo leyendo mi tesis y haber aceptado ser parte de mi jurado.

Quiero agradecer especialmente a Saúl, compañero de equipo en proyectos extraescolares y muy buen amigo, que me hizo recomendaciones y ayudó con la implementación del sistema; ahora la palabra *auto* tiene tantos significados.

A Rodrigo, muchas gracias por todos los atajos y facilidades que me mostró al usar un sistema GNU/Linux. También gracias por las sugerencias y ayuda al ver que la necesitaba.

En general, gracias a los compañeros del Laboratorio de Software Libre por permitirme ocupar un lugar con ustedes aún después de finalizar mis estudios de Licenciatura. Gracias por otorgarme un espacio y dejar que mi proyecto de tesis se desarrollara en el laboratorio.

Sé que faltan personas por agradecer el apoyo que he recibido, pero también es cierto que es poco lo antes mencionado para expresar lo agradecido que estoy con quienes sí aparecen en estos agradecimientos.

Por último en mis agradecimientos, pero que estuvo en primera fila a cada momento, gracias Alejandra. Gracias por tus palabras de aliento, tu apoyo incondicional, gracias por abrirme los ojos e impulsarme a seguir adelante en todo, gracias por tu confianza y estar sentada junto a mí mientras duró éste proyecto, gracias por que sé que seguirás en todos los proyectos que vengan. Gracias princesa.

Gerardo Velázquez Solis

Resumen

En éste trabajo se presenta un sistema que implementa un algoritmo para la identificación de subgrafos frecuentes. La base del algoritmo es la construcción de listas canónicas de códigos que representan a los grafos. Se establece un orden entre los códigos, utilizando tres criterios básicos: grado de adyacencia de los vértices en los grafos, frecuencia de aparición de la combinación de las etiquetas vértice - arco - vértice (código L_{VEV}), así como el orden lexicográfico de las etiquetas de los vértices y arcos. La construcción del código se basa en la técnica de exploración primero el mejor, así como un proceso iterativo para generar las diferentes secuencias basadas en las reglas que pueden representar al grafo. Para probar el algoritmo, se generó una serie de grafos de control con diferentes tipos de topologías, con la finalidad de probar la capacidad de detección de subgrafos frecuentes del algoritmo propuesto. Los resultados obtenidos muestran que la propuesta funciona correctamente, siendo una alternativa viable para la identificación de estructuras comunes en tareas de minería de datos basada en grafos.

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

El manejo de información siempre ha sido necesario para poder tener conocimiento de diversas características de un trabajo, experimento, ensayo, etc. En las últimas décadas los volúmenes de información son más y más grandes debido al rápido crecimiento y desarrollo en muchos ámbitos, gracias al también creciente avance en tecnología. Por tales motivos las técnicas para la creación, almacenamiento, distribución y procesamiento de información han tenido también que evolucionar para poder encontrar y generar conocimiento para toma de decisiones.

Dentro de las técnicas que últimamente se han explorado con gran interés para la representación y análisis de datos se encuentran las basadas en grafos. Entre las variantes que se han gestado se encuentra la Minería de Datos basada en grafos (MDBG), cuyo interés es el descubrimiento de estructuras comunes entre un conjunto de grafos, los cuales representan patrones de los datos originales.

Este tipo de técnicas se convirtió en todo un reto computacional y se han desarrollado un gran número de algoritmos para encontrar todos los subgrafos frecuentes de una colección de grafos. Esto se debe a la amplia gama de aplicaciones que tiene, que van de la biología, química, estructuras web, análisis de redes sociales, la astrofísica y hasta la geología e incluso lucha contra el terrorismo [8]; así como estas hay muchas más áreas en las que se pueden aplicar diversas técnicas de subgrafos frecuentes.

Dentro de la MDBG se encuentra la tarea de Minería de Subgrafos Frecuentes (SF), la cual consiste en encontrar todos los subgrafos comunes en un par de grafos de entrada (o bien en un conjunto de grafos de entrada). La tarea de encontrar Subgrafos Conexos Frecuentes (SCF) junto con las de localización de duplicados y pruebas de subisomorfismos, son bastante complejas y consumen gran cantidad de recursos computacionales, debido a las múltiples representaciones que puede tener un mismo patrón. Los sistemas que mejores

resultados reportan son aquellos que trabajan basados en representaciones canónicas, que buscan reducir las diferentes formas en las que se puede representar un mismo grafo.

Al hablar de algoritmos que trabajan con representación por grafos, tenemos principalmente dos técnicas, Con Generación de Candidatos y Sin Generación de Candidatos. Además, en un proceso de MDBG se debe tomar en cuenta si se trabaja con grafos etiquetados que representan datos discretos[1] o continuos[2]. Los algoritmos que funcionan bajo la metodología *con generación de candidatos* comparten características similares con el algoritmo Apriori para minar conjuntos frecuentes [1], fusionando subgrafos para generar nuevos posibles candidatos. Éstos algoritmos suelen generar muchos candidatos. Por otra parte, los basados en *crecimiento de patrones* (PG, por sus siglas en inglés) o *sin generación de candidatos*, suelen hacer primero una Búsqueda Primero en Profundidad (DFS, por sus siglas en inglés) para encontrar candidatos de subgrafos frecuentes, la generación de candidatos en este paradigma se basa en operaciones de extensión. A partir de estas estrategias generales, se han propuesto por la comunidad científica diferentes algoritos en el área de MDBG, como son: AGM [17], FSG [20] [19], PATH-BASED [24], gSpan [25] y MoFa [3]; y otros que se enfocan a encontrar Patrones Frecuentes y Subgrafos Isomorfos como SI-COBRA [22]. Hay que tener en cuenta que estas dos metodologías no son las únicas que existen; más adelante se mencionarán otras técnicas que han mostrado muy buenos resultados.

En el presente trabajo se expone una propuesta para la identificación de Subgafos Conexos Frecuentes visto desde el problema de Graph Matching, en el cual se intenta encontrar en un grafo, una estructura igual o lo más parecida posible a otro grafo dado, es decir detección de subisomorfismos, tanto exactos (Exact Graph Matching) e inexactos (Inexact Graph Matching).

En la propuesta, se suponen un grafo (patrón o muestra) que se pretende identificar en un conjunto de grafos (conjunto de grafos donde se busca) si es que está presente, donde a partir de los mismos se generan representaciones canónicas, las cuales representan las estructuras principales de los grafos, que son usadas como referencia en el proceso de búsqueda de subgrafos conexos frecuentes. Al grafo a buscar se le conoce como *grafo de muestra*[3] y a los grafos donde se busca se les denomina *grafos de búsqueda*[4].

[1] Al decir discreto/a nos referimos a datos que pertenecen a conjuntos en los cuales, los elementos pueden verse como individuales y entre un par de elementos del conjunto, no existe un tercero (como es el caso de los números reales).

[2] Son datos que describen de forma cuantitativa a un objeto o relación. Los podemos ver por ejemplo, como atributos tipo: altura, velocidad, volumen, etc.

[3] Un grafo de muestra es un grafo específico que pretendemos sea encontrado dentro de otros en su totalidad o de forma parcial

[4] Los grafos de búsqueda son los grafos que representan los datos en los que queremos encontrar subestructuras interesantes, valiéndonos de otros grafos llamados grafos de muestra, que nos dan una idea de qué buscar.

1.1. Objetivos

1.1.1. Objetivo General

Diseñar un algoritmo e implementar un sistema para la minería de datos con soporte para grafos etiquetados, que sea capaz de hallar los subgrafos conexos frecuentes en colecciones de grafos para la generación de conocimiento. El algoritmo se basará en la generación de listas de códigos canónicas para representar un grafo de interés que se intentará encontrar en otros grafos; para las cuales se propondrán las reglas y formas de recorrido para su creación. Este algoritmo servirá para extraer conocimiento de dominios estructurados.

1.1.2. Objetivos Particulares

1. Proponer los criterios y reglas de recorrido en grafos para la mejor generación posible de las listas canónicas de códigos que representarán al grafo que se desea encontrar.

2. Diseñar un algoritmo para la identificación de subgrafos conexos frecuentes que trabaje sobre las listas canónicas de códigos formados con anterioridad, basado en la estrategia de expansión sin generación de candidatos preferentemente.

3. Implementación del método para generación de formas canónicas y el algoritmo de identificación de SCF.

4. Realizar una serie de experimentos empíricos con grafos artificiales, con el objetivo de crear un cuadro del rendimiento del algoritmo.

1.2. Estado del arte

A lo largo del trabajo se tocarán principalmente dos puntos, por un lado están las técnicas que existen para la representación canónica de los grafos y por el otro, se encuentran algoritmos y sistemas encargados del descubrimiento de subgrafos conexos frecuentes.

1.2.1. Formas canónicas para Grafos

El propósito de generar una forma o código canónico, es el reconocimiento de un grafo de forma única. Para crear estas formas es necesario valerse de ordenamientos en los vértices y arístas, por orden lexicográfico, grado de adyacencia, incidencia, etc. Ejemplos de formas canónicas son:

CAM (Canonical Adjacency Matrix). La estructura de un grafo etiquetado puede ser expuesta como una matriz de adyacencia. A partir de esta representación es posible definir una secuencia única para representar dicho grafo.

Algunos autores han definido la CAM de un grafo G como la matriz de adyacencia que da lugar al código lexicográficamente mayor (o menor) entre todos los posibles códigos de las matrices de adyacencia asociadas con G [18] [13] [14]. Dada la metodología de creación de las secuencias, si dos códigos que representan a un mismo grafo, son iguales, esto quiere decir que se construyeron en base a la misma matriz de adyacencia.

DFS (Depth First Search). La codificación DFS se basa en el diseño de las formas canónicas en árboles T (T, por su sigla en inglés), a través de recorridos en profundidad en los grafos, por ejemplo $G =< V, E, L, l >$; el grafo G puede tener varios árboles DFS porque casi siempre existe más de un recorrido en profundidad.

Un código DFS es una secuencia de aristas construida a partir de un árbol DFS, ordenando lexicográficamente las aristas.

Este tipo de forma canónica fue introducido por Yan y Han [25].

BFS (Breadth First Search). La codificación BFS, también esta basada en recorridos a través del grafo, con la diferencia de que la forma de recorrido es en amplitud. Esta forma canónica se introdujo formalmente por Borgelt [2].

Al igual que en DFS, pueden existir varios T árboles BFS del mismo grafo ya que puede haber más de un recorrido de este tipo sobre el mismo grafo. El árbol T define un orden para los vértices de G, mismo que se dá por el momento en que fueron visitados durante el recorrido.

El código BFS que se construye con el árbol resultante del recorrido en BFS, toma como primer elemento la etiqueta del nodo raíz en el árbol T, y sigue tomando los elementos durante el recorrido en amplitud que dá lugar a T.

1.2.2. Algoritmos para Minería de Grafos

Ya que se tiene una idea de como formar códigos canónicos de grafos, pasaremos a conocer algoritmos para minería de grafos que han sido base para la creación de otros, que son populares e inovadores en la detección de SCF. Estos algoritmos se dividirán en: *Con Generación de Candidatos* y *Sin Generación de Candidatos*, también mencionaremos otros principios sobre los que se han trabajado en el tema.

- Con Generación de Candidatos - [9]

Desde que las bases de datos de transacción contienen un gran número de distintos elementos individuales, sus combinaciones llegan a formar grandes números de conjuntos de elementos, y se ha vuelto un reto el desarrollo de métodos que puedan escalarse para la minería de conjuntos de elementos frecuentes en una base de datos de gran tamaño. Agrawal y Srikant (1994) observaron una propiedad interesante llamada Apriori.

Un conjunto de n elementos es frecuente si y solo si todos sus subconjuntos son frecuentes. Esto es, los conjuntos de elementos frecuentes pueden ser extraídos por una primera exploración de la base de datos para encontrar elementos frecuentes de tamaño 1, inmediatamente se utilizan estos para generar candidatos de conjuntos de elementos frecuentes de tamaño 2. El procedimiento se repite hasta que no haya más conjuntos frecuentes que se puedan generar para algún n.

Ejemplos de algoritmos basados en la generación de candidatos bajo la propiedad Apriori son:

AGM (Apriori based Graph Mining) [17] Es conocido como el primer algoritmo de minería de subgrafos frecuentes, y se diseño para encontrar subgrafos frecuentes sin tener en cuenta la conexidad.

Este algoritmo utiliza un método de *generación de candidatos basada en vértices* que aumenta el tamaño de las subestructuras por un vértice en cada iteración. Dos grafos frecuentes de tamaño k se pueden fusionar solo si los dos grafos tienen el mismo subgrafo de tamaño *(k-1)*. En este punto el tamaño del grafo hace referencia al número de vértices en el grafo. Los nuevos candidatos formados incluyen el subgrafo común de tamaño *(k-1)* y los dos vértices adicionales de los dos patrones de tamaño k. Debido a que no se puede saber con certeza si hay una arista conectando los dos vértices adicionales, en realidad AGM puede generar dos candidatos y esta característica en especial es la que provoca el alto número de candidatos que hay que procesar. Se tiene que verificar cada subgrafo procesado para saber si es soportado por la CAM (conocer si existen o no un número mínimo de repeticiones para poder decir que es un subgrafo frecuente).

De la misma manera que AGM, las adaptaciones que se han hecho de este algoritmo para la minería de SCF (AcGM [18], Topology [13], Generalized AcGM [16]) miden el tamaño de los grafos contando el número de vértices. Los grafos candidatos son representados mediante su CAM, que es usada para la generación de candidatos [4].

FSG (Frecuent SubGraph discovery) [19] [20] Dada la necesidad de resolver algunos problemas que se presentan en el algoritmo AGM, como la generación de una gran cantidad de candidatos y el inconveniente de no tomar en cuenta la conexidad en los subgrafos; es que se desarrolla FSG.

Primero se enumeran todos los subgrafos de una arista que cumplan con ser soportadas (que tengan un mínimo de repeticiones para que se consideren como frecuentes). El algoritmo adopta una estrategia de *generación de candidatos basada en aristas*, que incrementa el tamaño de la subestructura en una arista por cada iteración.

FSG representa los grafos como etiquetas canónicas, por lo que si dos grafos tienen la misma etiqueta, se dirá son isomorfos. Y si dos grafos varían en solo una arista, se pueden fusionar dando lugar a un nuevo grafo más grande.

Debido a la reducción de candidatos creados, este procedimiento acelera las operaciones de encontrar estructuras interesantes.

PATH-BASED [24] Es otra variante del algoritmo de generación de candidatos, la técnica es usar caminos disjuntos. Su forma de identificar el tamaño de los grafos se da por la cantidad de caminos disjuntos que lo forman. Un grafo cadidato de tamaño *k+1* se crea tomando dos grafos de tamaño k que se diferencien por un sólo camino disjunto.

- Sin Generación de Candidatos - [9]

En muchas ocasiones, los algoritmos tipo Apriori usados en los métodos de generación de candidatos, reduce de manera significativa el tamaño de conjuntos de candidatos, sin embargo, puede sufrir de dos costos no triviales: el primero es "generar un gran número de conjuntos candidatos"; el segundo, escanear la base de datos repetidamente y revisar los candidatos por patrones correspondientes (isomorfismos). En [10], se presenta un método llamado FP-growth, que mina por completo el conjunto de elementos frecuentes sin hacer una generación de candidatos previa, procedimiento que da lugar a la *localización de SCF con una técnica de crecimiento de patrones sin generación de candidatos.*

FP-growth trabaja de la manera *divide y conquista.* La primera vez que se escanea la base de datos, se obtiene una lista de elementos frecuentes, en la cual, los elementos están ordenados por frecuencia de manera descendente. Con la lista, podemos ver la base de datos de una manera comprimida, como un árbol de patrones frecuentes (FP-tree), que tiene la información asociada al conjunto de elementos de la base.

El procedimiento para encontrar los SCF comienza minando el FP-tree, empezando por cada patrón de longitud 1, construyendo su *patrón condicional base* (sub base de datos que consiste del conjunto de caminos de prefijos en el FP-tree que co-ocurren con el patrón de sufijos), y entonces tenemos que construir su FP-tree para poder realizar el minado de forma recursiva en tal árbol. El crecimiento de patrones se archiva por la concatenación del patrón de sufijos, junto con la frecuencia de patrones generados del FP-tree condicional.

Estudios realizados demuestran que el método reduce el tiempo de búsqueda sustancialmente. Ejemplos de algoritmos que utilizan el método de crecimiento de patrones se dan a continuación:

gSpan (graph-based Substructure pattern mining) [25] Es el primer algoritmo de SCF basado en crecimiento de patrones, y también el que introduce las formas canónicas mediante codificaciones DFS para representar los grafos.

La forma de búsqueda en gSpan está definida por un árbol DFS, con códigos DFS en sus nodos. Ya que el árbol es generado mediante una búsqueda en profundidad, cada código DFS mínimo es hijo de algún otro código DFS mínimo, entonces al recorrer el espacio de búsqueda para una arista frecuente, se logra una búsqueda completa en el árbol. Ya que los códigos DFS se generan tomando en cuenta el orden lexicográfico, se reduce en gran medida la posibilidad de generar duplicados de códigos mínimos, debido a que se verifica que aún no exista el patrón candidato.

gRed (graph Candidate Reduction Miner) [4] Basado en la forma de trabajo de gSpan, utiliza su idea básica, pero gRed hace notar su más grande aportación, haciendo uso de propiedades de los códigos DFS que no habían sido explotadas en otros trabajos. gRed reduce el número de candidatos y también el tiempo que se dedica a la búsqueda de duplicados, realizando búsquedas binarias que permiten separar extensiones que producen códigos no mínimos de las que sí (los códigos no mínimos no son aptos para ser candidatos).

MoFa (mining Molecular Fragments) [3] Es un algoritmo para minar conjuntos frecuentes, creado en principio para el análisis de fragmentos de estructuras moleculares y ofrece gran cantidad de funcionalidades en el trabajo con este tipo de estructuras y datos moleculares, por otro lado, en estudios se muestra que demora en sus tiempos de respuesta en comparación con otros algoritmos.

Este algoritmo, utiliza como método para representar sus candidatos la codificación BFS.

FFSM (Fast Frecuent Subgraph Mining) [14] El método combina estrategias de los dos paradigmas para minar SCF (Apriori y crecimiento de patrones); para la representación de los grafos aplica CAM como codificación.

Punto importante del algoritmo es que trunca en gran medida las extensiones de los grafos, por lo que el resultado no garantiza tener todos los SCF, y es en este punto donde entra la idea de *fusión* de las formas Apriori, mezclando candidatos pequeños para generar candidatos de mayor tamaño.

Gaston (Graph/sequence/tree extraction) [21] Por muchos autores, considerado el mejor algoritmo para minar SCF y junto con otros es de los tomados como base para el desarrollo de demás algoritmos.

Antes de buscar subgrafos frecuentes, busca subestructuras simples, como árboles; el procedimiento comienza buscando subcaminos, a partir de los cuales construye subárboles y por último subgrafos. Si bien es cierto que esta etapa toma la mayor parte del tiempo de ejecución, es bien justificada, ya que la construcción de las estructuras de correspondencia, evita las costosas pruebas de subisomorfismo y aceleran el resto del procedimiento.

SPIN (Spanning tree based maximal graph mining) [15] Es un algoritmo de minería de datos, que a diferencia de la mayoría de algoritmos, mina solamente los subgrafos frecuentes *maximales*. Esta idea de solo minar los subgrafos frecuentes maximales reduce en el mejor caso de manera exponencial el tamaño del conjunto de salida de resultados. SPIN soporta trabajo con bases de datos bastante grandes debido a que no toma en cuenta subgrafos frecuentes de un tamaño menor al establecido por default, esto con ayuda de un algoritmo de compresión.

- Inspirados en otros principios -

SUBDUE (Inspirado en MDL) [11] [12] [23] Es un sistema de descubrimiento de conocimiento basado en grafos que encuentra estructuras, patrones de relación en entidades representadas por datos y relaciones.

SUBDUE utiliza el principio de *longitud mínima de descripción* (MDL, por sus siglas en inglés) para comprimir una subestructura y trabajar con menos nodos.

Puede trabajar de manera supervisada y no supervisada, así como clusterizar y aprender de gramáticas libres de contexto.

Algoritmos inspirados en MSD [6] MSD (*Markovian Spectrally Distinguishable graphs*), fue inspirado en un principio del algoritmo de PageRank de Google.

La idea de introducir MSD en el problema de Graph Matching es otorgar firmas que obedezcan la topología de cada nodo en el grafo, mismas que son usadas para construir mapeo entre dos grafos. Un *camino aleatorio* (RW, por sus siglas en inglés) en cierto grafo, es descrito por un modelo probabilístico que permite calcular la probabilidad de cada vértice en un tiempo determinado. De cierta manera se supone que *el caminante aleatorio* se puede mover de un nodo a otro a través de una arista o saltar hasta el nodo en cuestión, dependiendo de la probabilidad de distribución y topología de los nodos.

Se busca que el *caminante aleatorio* vaya pisando los nodos con los que es más probable formar isomorfismos.

1.3. Alcances y Limitaciones

Terminado el trabajo, se pretende tener como resultado un nuevo algoritmo conformado por un método para la generación de códigos canónicos y uno para la localización de patrones en grafos; siendo implementado en la forma de un sistema para minería de datos basado en grafos etiquetados, que arroje como respuesta a los patrones de SCF más grandes posibles encontrados.

Por otro lado, algoritmo y sistema estarán acotados a la búsqueda de subisomorfismos bajo el problema de Graph Matching, pero no se contempla hacer Error-Correcting Graph Matching (Corrección de Error en Emparejamiento de Grafos) y tampoco se medirá la distancia entre un grafo y otro.

Capítulo 2

Notación y Definiciones

Para poder comprender de una buena manera lo que se expone y propone en este trabajo de tesis, a continuación se explican los conceptos necesarios y que se estarán utilizando de ahora en adelante. Se definirán entre otras cosas, qué es un grafo, características y propiedades, formas de representación, equivalencias, etc.

Comencemos con el concepto de grafo, que es la base para este trabajo. Un grafo *G=(V,E)* consiste en un conjunto finito V de elementos llamados vértices y un conjunto finito E de elementos llamados aristas, en el cual a una arista le corresponden un par de vértices [5]. Para nuestro caso, es esencial aumentar la definición, ya que trabajamos con grafos que están etiquetados [1] de manera discreta. Tomaremos la siguiente definición de grafo:

Definición 1 *-Grafo-* Un grafo G es una 6-tupla *G=(V, E, L_V, L_E, α, β)*, donde:

- $V = \{v_i | i = 1, ..., n\}$, *es un conjunto finito de vértices, donde* $V \neq \emptyset$ *y* n *representa al número de vértices en el grafo;*
- $E \subseteq V \times V$, donde los elementos del conjunto se denotan como $e = \{v_i, v_j\}, v_i, v_j \in V$;
- L_V *es el conjunto finito de etiquetas de los vértices;*
- L_E *es el conjunto finito de etiquetas de las aristas;*
- α*:* $V \rightarrow L_V$ *es la función de asignación de etiquetas sobre los vértices;*
- β*:* $E \rightarrow L_E$ *es la función de asignación de etiquetas sobre las aristas.*

[1] Una etiqueta es una cadena asociada a un vértice o arista en el grafo.

En esta notación, a cada vértice se le asigna arbitrariamente un índice numérico x con la finalidad de poder identificar de manera única a cada vértice en G. En este trabajo, se asume que para cualquier v_x, $x \in [1, n]$. Denotaremos por $deg(v_x)$ al grado de adyacencia del vértice v_x, mientras que con $frec(L_V(v_x))$ a la frecuencia de aparición de la etiqueta de v_x en G.

La definición describe el tipo específico de grafos no dirigidos. Un grafo dirigido, se describe como $G^{\rightarrow} = (V, E, L_V, L_E, \alpha, \beta)$, un grafo donde las aristas se representan por (v_i, v_j), tal que v_i y v_j son un par ordenado de vértices. Tenemos entonces que $(v_i, v_j) \neq (v_j, v_i)$. En la Figura 2.1a) se muestra un grafo etiquetado no dirigido $G = (V, E, L_V, L_E, \alpha, \beta)$, donde:

- $V = \{1, 2, 3, 4\}$;
- $E = \{\{1, 3\}, \{2, 4\}, \{3, 4\}, \{4, 1\}\}$;
- $L_V = \{A, R, U, Z\}$;
- $L_E = \{a, b, c\}$;
- $\alpha(1) = A, \alpha(2) = U, \alpha(3) = R, \alpha(4) = Z$;
- $\beta(\{1, 3\}) = b, \beta(\{1, 4\}) = a, \beta(\{2, 4\}) = a, \beta(\{3, 4\}) = c$.

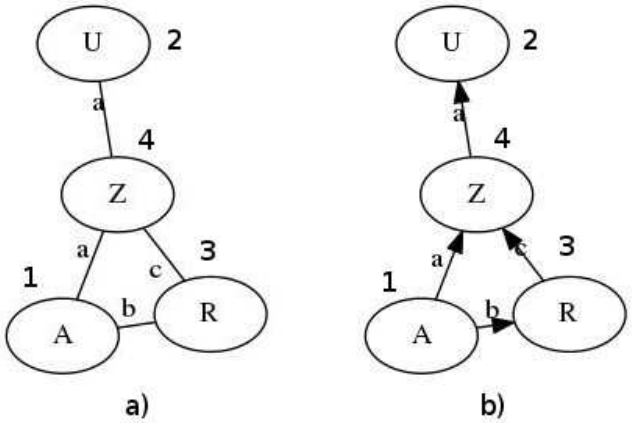


Figura 2.1: Observamos dos grafos distintos, a)un grafo no dirigido y b)grafo dirigido.

Por otro lado, en la Figura 2.1b) tenemos un grafo dirigido, el cual parece ser igual que 2.1a), pero ya que las aristas indican de que vértice salen y a cual llegan, se convierten en grafos distintos. Podemos describir a 2.1b) haciendo notar que las aristas son pares ordenanos de vértices y se definirá como:

- $V = \{1, 2, 3, 4\}$;
- $E = \{(1, 3), (1, 4), (3, 4), (4, 2)\}$;
- $L_V = \{A, R, U, Z\}$;
- $L_E = \{a, b, c\}$;
- $\alpha(1) = A, \alpha(2) = U, \alpha(3) = R, \alpha(4) = Z$;
- $\beta((1, 3)) = b, \beta((1, 4)) = a, \beta((3, 4)) = c, \beta((4, 2)) = a$.

Un grafo no etiquetado se puede definir como $G = (V, E, L_V, L_E, \alpha, \beta)$, en el cual $|L_V| = |L_E| = 1$.

Dos vértices v_i, $v_j \in G$ son **adyacentes** o vecinos, denotado por $v_i \sim v_j$ si $\exists$ $e \in E$, donde $e = \{v_i, v_j\}$ y e es una arista incidente en v_i y v_j. Un ejemplo de vértices adyacentes son v_2 y v_4 en Figura 2.1a). El **grado** de un vértice v, se denota por $deg(v_x)$ donde x es el número de vértice y el grado es el número de vértices adyacentes al vértice v_x; por ejemplo, en la misma Figura 2.1a) tenemos que el vértice v_4 tiene un grado igual a 3, denotado por $deg(v_4) = 3$, ya que v_1, v_2 y v_3 son adyacentes a él. El concepto de grado cambia cuando lo introducimos a grafos dirigidos, en estos casos tenemos **grado de entrada**, que es el número de aristas entrantes al vértice, y **grado de salida**, que es igual al número de aristas que salen del vértice; en Figura 2.1b) tenemos un grado de entrada igual a $deg\_in(v_4) = 2$ y un grado de salida igual a $deg\_out(v_4) = 1$.

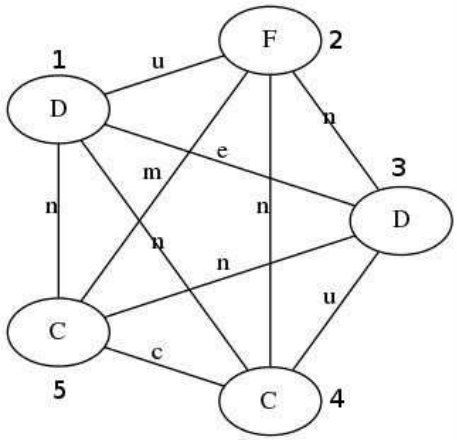


Figura 2.2: Grafo completo de 5 vértices.

Definición 2 -*Grafo Completo*- Un grafo G es completo si $\forall v_i, v_j \in V$, $v_i \neq v_j : v_i \sim v_j$.

Frecuentemente, un grafo completo se donata por K_n, donde n es el número de vértices en el grafo. Si $G = (V, E, L_V, L_E, \alpha, \beta)$ es completo, entonces $\forall v \in V : deg(v) = n - 1$, donde $|V| = n$. En la Figura 2.2 tenemos un ejemplo de grafo completo.

Definición 3 -*Grafo Conectado*- Se dice que un grafo G está conectado si existe una secuencia de aristas $\{v_i, v_j\}$, $\{v_j, v_k\}$, ..., $\{v_y, v_z\}$ para cada par de vértices v_i y v_z en G, con $v_i \neq v_z$.

Dicho de otra forma, un grafo es conectado si existe un camino de aristas entre dos vértices cualesquiera. En Figura 2.3 vemos G un grafo conectado y G' uno no conectado, en G' no podemos ir de v_2 a v_4 por ejemplo.

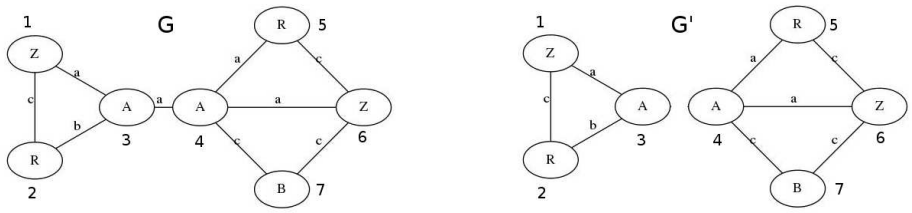


Figura 2.3: Grafos G conectado y G' no conectado.

Definición 4 -*Subgrafo*- Sea G un grafo, donde $G=(V, E, L_V, L_E, \alpha, \beta)$. Un subgrafo G^S de G, se denota por $G^S \subseteq G$ donde, $G^S=(V^S, E^S, L_V^S, L_E^S, \alpha^S, \beta^S)$ es un grafo tal que $V^S \subseteq V$, $E^S \subseteq E$, $L_V^S \subseteq L_V$, $L_E^S \subseteq L_E$, $\alpha^S \subseteq \alpha$ y $\beta^S \subseteq \beta$. En la Figura 2.4b) se muestra un ejemplo de subgrafo de la Figura 2.4a).

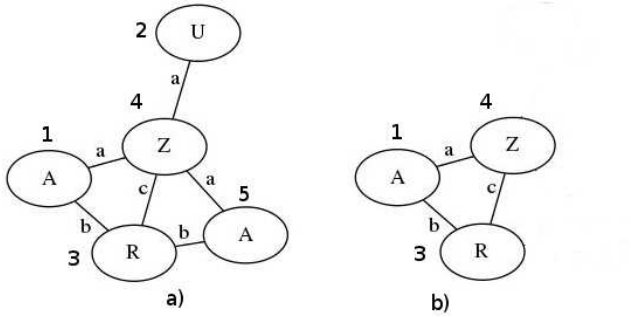


Figura 2.4: En b) tenemos un subgrafo de a).

Definición 5 -*Subgrafo Inducido*- Sea G^S un subgrafo de G. G^S es un subgrafo inducido de G si $\forall v_i, v_j \in V^S : \{v_i, v_j\} \in E \leftrightarrow \{v_i, v_j\} \in E^S$.

En otras palabras, si v_i y $v_j \in V^S$, entonces toda arista en E formada por v_i y v_j debe estar también en E^S. El grafo b) que se muestra en la Figura 2.4, es un ejemplo de grafo inducido del grafo a).

Definición 6 -*Cliqué*- Un cliqué en un grafo G es un subconjunto $V^S \subseteq V$, tal que el grafo inducido de G en V^S, es un grafo completo. Un grafo puede tener más de un cliqué.

En la Figura 2.4 encontramos dos cliqués en el grafo a). El primero está formado por el subconjunto de vértices $V^S = \{v_1, v_3, v_4\}$ que inducen un grafo completo tomando las aristas $E^S = \{\{v_1, v_3\}, \{v_1, v_4\}, \{v_3, v_4\}\}$. El segundo lo tenemos en el subconjunto de vértices $V^S = \{v_3, v_4, v_5\}$ e inducido por el subconjunto de aristas $E^S = \{\{v_3, v_4\}, \{v_3, v_5\}, \{v_4, v_5\}\}$.

Definición 7 -*Isomorfismo*- Dados dos grafos $G' = (V', E', L'_V, L'_E, \alpha', \beta')$ y $G = (V, E, L_V, L_E, \alpha, \beta)$, G' es isomorfo a G, denotado por $G' \cong G$, si existen biyecciones $f : V' \to V$ y $g : E' \to E$, donde:

- $\forall v' \in V', \alpha'(v') = \alpha(f(v'))$;
- $\forall \{v'_i, v'_j\} \in E', \beta'(\{v'_i, v'_j\}) = \beta(g(\{v'_i, v'_j\}))$.

Dicho de otro modo, G' y G son isomorfos si y solo si: *1)*si los vértices v'_x y v'_y en G' tienen una relación de correspondencia con los vértices v_i y v_j en G respectivamente, entonces una arista en G' conformada como $\{v'_x, v'_y\}$ debe corresponder también a una arista $\{v_i, v_j\}$ en G y viceversa; *2)*las etiquetas de ambos grafos deben ser exactamente las mismas.

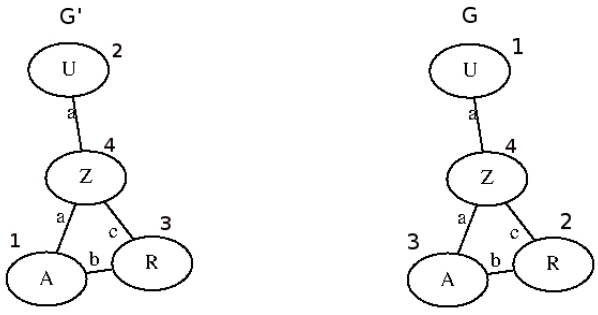


Figura 2.5: Dos grafos isomorfos, ambos tienen el mismo etiquetado y topología.

Hay que notar que si $|L'_V| = |L'_E| = 1$, entonces tenemos una definición más básica de isomorfismo en la que existe una correspondencia de uno a uno entre los conjuntos de vértices y también entre los conjuntos de aristas, tal que las aristas correspondientes de G' y G son incidentes a sus vértices correspondientes en G' y G.

Tenemos un ejemplo de grafos isomorfos en la Figura 2.5, donde:

- $f(v'_1) = v_3, f(v'_2) = v_1, f(v'_3) = v_2, f(v'_4) = v_4$;
- $g(\{v'_1, v'_3\}) = \{v_3, v_2\}$, $g(\{v'_1, v'_4\}) = \{v_3, v_4\}$, $g(\{v'_2, v'_4\}) = \{v_1, v_4\}$, $g(\{v'_3, v'_4\}) = \{v_2, v_4\}$;
- $\alpha'(v'_1) = \alpha(v_3)$, $\alpha'(v'_2) = \alpha(v_1)$, $\alpha'(v'_3) = \alpha(v_2)$, $\alpha'(v'_4) = \alpha(v_4)$;
- $\beta'(\{v'_1, v'_3\}) = \beta(\{v_3, v_2\})$, $\beta'(\{v'_1, v'_4\}) = \beta(\{v_3, v_4\})$, $\beta'(\{v'_2, v'_4\}) = \beta(\{v_1, v_4\})$, $\beta'(\{v'_3, v'_4\}) = \beta(\{v_2, v_4\})$.

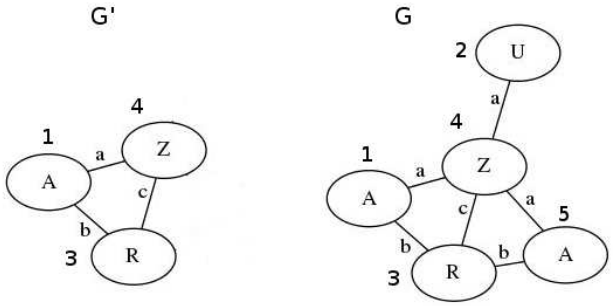


Figura 2.6: En G podemos ver dos subgrafos que son isomorfos a G' ya que se puede establecer una relación de correspondencia entre sus conjuntos de vértices y arcos.

Definición 8 ***-Subgrafo Isomorfo-*** Sean G' y G dos grafos. G' es un subgrafo isomorfo de G si existe $G^S \subseteq G$ tal que $G' \cong G^S$.

En Figura 2.6 podemos definir G^S un subgrafo de G, de la siguiente manera, $G^S \subseteq G : G^S = \{V^S, E^S, L_V^S, L_E^S, \alpha^S, \beta^S\}$, donde:

- $V^S = \{v_3, v_4, v_5\}$;
- $E^S = \{\{v_3, v_4\}, \{v_4, v_5\}, \{v_5, v_3\}\}$;
- $L_V^S = \{A, R, Z\}$;
- $L_E^S = \{a, b, c\}$;

- $\alpha^S(v_3) = R$, $\alpha^S(v_4) = Z$, $\alpha^S(v_5) = A$;
- $\beta^S(\{v_3, v_4\}) = c$, $\beta^S(\{v_4, v_5\}) = a$, $\beta^S(\{v_5, v_3\}) = b$.

Tenemos entonces $G^S \subseteq G$ y $G^S \cong G'$, con lo que G' es un subgrafo isomorfo de G. Se puede ver también que hay un segundo subgrafo en G formado por el subconjunto de vértices $V_S = \{v_1, v_3, v_4\}$ y las correspondientes aristas entre ellos, teniendo así otro subgrafo al que G' es isomorfo.

Al problema de encontrar si existe algún $G^S \subseteq G$ tal que $G^S \cong G'$, se conoce como el *Problema de Subgrafos Isomorfos* (SI-problem, por su notación en inglés).

Definición 9 -*Homomorfismo*- Dados dos grafos $G' = (V', E', L'_V, L'_E, \alpha', \beta')$ y $G = (V, E, L_V, L_E, \alpha, \beta)$, G' es homomorfo a G si y sólo si existe una función $f : V' \to V$ sobreyectiva tal que para $v'_i, v'_j \in V'$ y $v_i, v_j \in V$ se cumplen:

- $\{v'_i, v'_j\} \in E' \to \{f(v'_i), f(v'_j)\} \in E$;
- $\{v_i, v_j\} \in E \to \exists v'_k, v'_l \in V' : \{v'_k, v'_l\} \in E' \to f(v'_k) = v_i, f(v'_l) = v_j$.

Definición 10 -*Emparejamiento Exacto de Grafos (Exact Graph Matching)*- Dados dos grafos G_M (grafo de muestra) y G_B (grafo de búsqueda) que tienen $|V_M| = |V_B|$, se dice que existe un mapeo o *emparejamiento exacto* de los grafos, si y sólo si existe una función f donde: $f : V_B \to V_M : \{v_i, v_j\} \in E_B \leftrightarrow \{f(v_i), f(v_j)\} \in E_M$. Es decir, hay un emparejamiento exacto entre G_M y G_B si y sólo si G_M y G_B son isomorfos. Existe el emparejamiento exacto para grafos y subgrafos (grafos isomorfos y subgrafos isomorfos).

Podemos observar cuatro grafos en la Figura 2.7, de los cuales G_1 y G_2 tienen un emparejamiento exacto, al igual que G_3 y G_4. Es claro que los pares de grafos que se pueden emparejar, tienen un etiquetado idéntico; lo que probablemente no sea tan evidente es que describan la misma forma. Hay que tener cuidado cuando de emparejamiento de grafos se trata, ya que para un isomorfismo la forma aparente que tomen los grafos no interesa, siempre y cuando se respete la topología de los grafos en cuestión.

Definición 11 -*Emparejamiento Inexacto de Grafos (Inexact Graph Matching)*- Dados dos grafos G_M y G_B, con $|V_M| < |V_B|$ si existe una función f que: $f : V_B \to V_M | \{v_i, v_j\} \in E_B \leftrightarrow \{f(v_i), f(v_j)\} \in E_M$, entonces tenemos un *emparejamiento inexacto* entre G_M y G_B; esto es, la búsqueda de un grafo pequeño dentro de uno más grande. De forma análoga, si existe una función f para un par de grafos G y G', con $V' \subseteq V$ y $E' \subseteq E$, donde: $f : V' \to V | \{v_i, v_j\} \in E' \leftrightarrow \{f(v_i), f(v_j)\} \in E$, entonces ocurre un *emparejamiento de subgrafos* o *isomorfismo de subgrafos* entre G y G'.

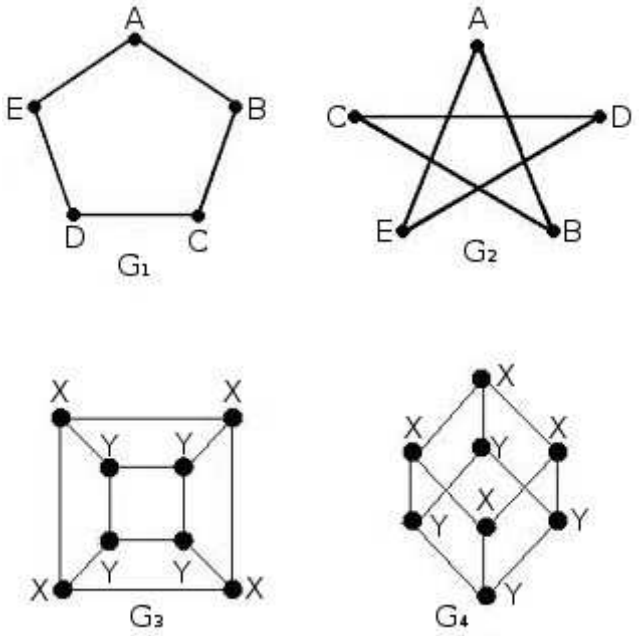


Figura 2.7: G_1 y G_3 son isomorfos a G_2 y G_4 respectivamente.

Si nos fijamos nuevamente en la Figura 2.6, vamos encontrar que podemos identificar emparejamientos inexactos entre G' y G, teniendo claramente dos subgrafos en G que tienen una correspondencia con G'.

Definición 12 -*Anillo*- Un grafo G es un anillo si existe una secuencia de vértices distintos $< v_1, v_2, \ldots, v_n >$, $|V| = |E| = n$, donde $\forall v_i, v_{i+1} : v_i \sim v_{i+1}, v_n \sim v_1$.

Definición 13 -*Camino, Recorrido, Circuito, Ciclo*- Para hacer sencillas las definiciones y comprensión de éstos términos se presenta el Cuadro2.1.

Vértice(s) repetido(s)	Arista(s) repetida(s)	Abierto	Cerrado	Nombre
Sí	Sí	Sí		Camino
Sí	Sí		Sí	Camino (cerrado)
Sí	No	Sí		Recorrido
Sí	No		Sí	Circuito
No	No	Sí		Camino simple
No	No		Sí	Ciclo

Cuadro 2.1: Tomado de [7].

Definición 14 -*Árbol*- Un árbol T es un grafo conectado, no dirigido y sin ciclos.

La Figura 2.8 muestra diversos árboles, todos son conectados y sin ciclos.

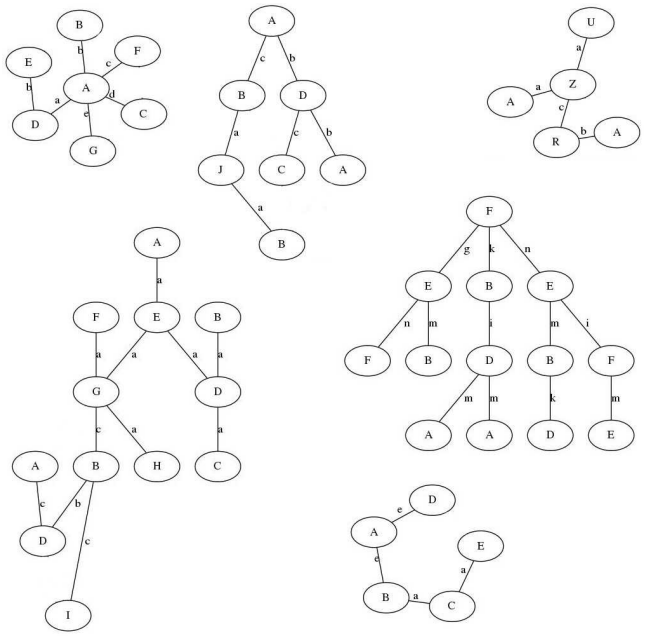


Figura 2.8: Ejemplos de vistas de árboles.

Definición 15 -*Árbol de Expansión*- Un árbol de expansión de un grafo G, es un subgrafo $G^S \subseteq G$, tal que $V^S = V$ y G^S es un árbol.

Los vértices y aristas marcados con líneas sólidas en la Figura 2.9 representan un T árbol de expansión del grafo en cuestión. En éste árbol T se encuentran incluidos todos los vértices del grafo. Hay que notar también que no solo existe un árbol de expansión para cada grafo.

Definición 16 -*Forma Canónica (FC)*- Es una representación que *identifica de forma única* a un grafo. Una forma canónica puede estar dada como matriz (CAM), cadena de códigos, como un simple árbol o una lista de códigos. No existe una sola forma canónica para un grafo; estas dependen del tipo de recorrido que se emplea, las reglas y el orden establecido para la exploración.

Es necesario mencionar que si dos grafos G' y G son isomorfos, su FC será exactamente igual si y solo si son obtenidas a través del mismo algoritmo

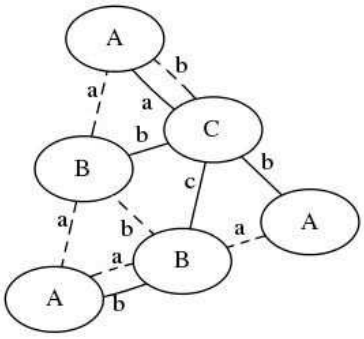


Figura 2.9: Las aristas sólidas describen un T árbol de expansión. Las aristas punteadas son parte del grafo, pero no del árbol. T incluye a todos los vértices.

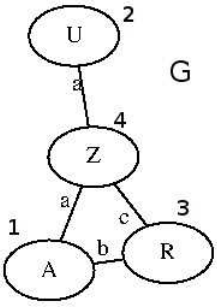
generador de la FC. Es decir, que se emplee el mismo tipo de recorrido, con las mismas reglas y el mismo orden para la exploración de los grafos.

Definición 17 *-Lista Canónica de Códigos (LCC)-* Una lista canónica de códigos para un grafo $G = \{V, E, L_V, L_E, \alpha, \beta\}$, es una lista de la forma $LCC = \{(v_i, v_j, l_{(v_i,e,v_j)}) : \text{para todo } v_i, v_j \in V,\ e \in E,\ l_{(v_i,e,v_j)} \in L_{VEV}\}$. L_{VEV} es una lista que se deriva de la combinación de etiquetas que tienen los vértices y las aristas. Dado un grafo G, una lista L_{VEV} se define por: $L_{VEV} = \{l_{(v_i,e,v_j)} : \alpha(v_i), \alpha(v_j) \in L_V, \beta(e) \in L_E, i = 1, ..., |V|, j = 1, ..., |V|\}$, donde $|L_{VEV}| = |E|$. Los elementos de LCC son ordenados de acuerdo a criterios predefinidos (criterios de ordenación), de tal forma que utilizando una técnica de exploración (profundidad, anchura, u otra variante) se establece el orden en el cual aparece cada entrada del código.

Definición 18 *-Lista Canónica Derivada (LCD)-* Definimos a las LCD's como subconjuntos de alguna lista LCC, denotadas como, $LCD \subseteq LCC$.

Una LCD representa un subgrafo de una LCC y es obtenida a partir de una LCC existente. Las LCD también son LCC y se generan de la misma manera. Al crear una LCD se elimina el código raíz de la LCC que se derivará y no se vuelve a tomar en cuenta, se selecciona el segundo código de la lista y a partir de éste se prosigue con el algoritmo generador de LCC. Así logramos la forma canónica de un subgrafo.

En la Figura 2.10 se observan la LCC y primera LCD de G, es claro que LCD es la forma canónica de un subgrafo de G.



LCC
(1, 3, AbR)
(1, 4, AaZ)
(2, 4, UaZ)
(3, 4, RcZ)

LCD
(1, 4, AaZ)
(2, 4, UaZ)
(3, 4, RcZ)

Figura 2.10: Grafo G con sus representaciones canónicas LCC y LCD.

Capítulo 3

Propuesta

La idea para el sistema de minería de datos que identifica subgrafos isomorfos está compuesta de dos módulos fundamentales:

1. **Creación de la forma canónica.** Módulo en que se procesa un grafo y genera una FC en forma de lista de códigos, es decir, una LCC. La misma técnica es usada para crear las LCD's que se ocuparán en el proceso de identificación de subgrafos isomorfos;

2. **Algoritmo para la detección de isomorfismos.** Procedimiento que apoyado de las LCC's, busca subgrafos que emparejen correctamente con algún subgrafo que se desea localizar, procurando encontrar en el grafo de búsqueda, los subgrafos más frecuentes y grandes que son isomorfos al grafo de muestra o uno de sus subgrafos.

A continuación se explican los principales módulos y después como trabajan en conjunto dentro del sistema.

3.1. Creación de Formas Canónicas

Consideremos que como entrada se tienen dos grafos conexos no dirigidos G_M y G_B, con etiquetas discretas tanto en vértices como en arcos. La tarea a resolver es determinar si $G_M \cong G_B$, o detectar todos los subgrafos isomorfos, es decir, encontrar $\forall G_S$ subgrafo conexo tal que: $G_S \subseteq G_M$ y $G_S \subseteq G_B$.

Para resolver la tarea, se propone generar una LCC que represente a G_M, de tal forma que la detección de los subgrafos comunes se lleve acabo mediante los códigos LCC. Con esto, se pretende reducir la múltiple representación que puede tener un grafo, permitiendo acelerar el proceso de detección de isomorfismos.

Para generar el LCC del grafo G_M, el primer paso consiste en generar una lista con la combinacin de las etiquetas vértice - arco - vértice que se

presentan en el grafo G_M (L_{VEV}). Con la finalidad de establecer un orden más estricto al momento de definir los códigos de LCC, L_{VEV} es ordenada lexicográficamente. Esto es, para dos códigos $l_{(v_i,e,v_j)}$, $l_{(v_m,e',v_n)} \in L_{VEV}$, se define que $l_{(v_i,e,v_j)} \leq l_{(v_m,e',v_n)}$ si se cumple alguna de las siguientes condiciones:

1. $\alpha(v_i) < \alpha(v_m)$ (el código ASCII de $\alpha(v_i)$ es menor que el código ASCII de $\alpha(v_m)$), o;
2. $\alpha(v_i) = \alpha(v_m)$ y $\beta(e) < \beta(e')$ (el código ASCII de $\alpha(v_i)$ es igual que el código ASCII de $\alpha(v_m)$ y además el código ASCII de $\beta(e)$ es menor que el código ASCII de $\beta(e')$), o;
3. $\alpha(v_i) = \alpha(v_m)$, $\beta(e) = \beta(e')$ y $\alpha(v_j) \leq \alpha(v_n)$ (el código ASCII de $\alpha(v_i)$ es igual que el código ASCII de $\alpha(v_m)$, el código ASCII de $\beta(e)$ es igual que el código ASCII de $\beta(e')$ y además el código ASCII de $\alpha(v_j)$ es menor o igual que el código ASCII de $\alpha(v_n)$).

A partir de L_{VEV}, se generan las entradas del código LCC. Para su construcción, se usa un algoritmo de exploración Primero el Mejor (BFS, por sus siglas en inglés). El criterio para determinar el orden entre los códigos de LCC se basa en el grado de adyacencia, frecuencia de aparición lexicográfica de etiquetas l_{vev}, orden lexicográfico de las etiquetas l_{vev}, así como el índice numérico asignado a los vértices del grafo, **reglas DFLE**. Esto es, dados dos códigos $(v_i, v_j, l_{(v_i,e,v_j)})$, $(v_m, v_n, l_{(v_m,e',v_n)}) \in LCC$, $(v_i, v_j, l_{(v_i,e,v_j)}) \leq (v_m, v_n, l_{(v_m,e',v_n)})$ si se cumple uno de los siguientes criterios:

1. $deg(v_i) + deg(v_j) < deg(v_m) + deg(v_n)$ (grado de adyacencia ascendente);
2. $deg(v_i) + deg(v_j) = deg(v_m) + deg(v_n)$ y $frec(l_{(v_i,e,v_j)}) > frec(l_{(v_m,e',v_n)})$ (frecuencia de aparición descendente de las etiquetas de vértice - arco - vértice);
3. $deg(v_i) + deg(v_j) = deg(v_m) + deg(v_n)$, $frec(l_{(v_i,e,v_j)}) = frec(l_{(v_m,e',v_n)})$ y $l_{(v_i,e,v_j)} < l_{(v_m,e',v_n)}$ (orden lexicográfico);
4. $deg(v_i) + deg(v_j) = deg(v_m) + deg(v_n)$, $frec(l_{(v_i,e,v_j)}) = frec(l_{(v_m,e',v_n)})$, $l_{(v_i,e,v_j)} = l_{(v_m,e',v_n)}$, y $\min\{i,j\} \leq \min\{m,n\}$(enumeración de los vértices).

Con las reglas se nota que la intención es buscar las estructuras de interés partiendo de los códigos que tengan menor adyacencia, pero que a su vez sean los más frecuentes. Si tenemos un código con poca adyacencia, se reducen las posibilidades de asociarlo con otros arcos en el proceso de mapeo. Si se eligieran primero los arcos con mayor grado de adyacencia, entonces el número de posibles patrones que se pueden asociar será mayor en principio, pero si las elecciones no coinciden, se estarían creando patrones sin futuro por la probabilidad de asociación con el resto de los códigos en la secuencia. En caso de que dos códigos salgan empatados con los dos primeros criterios, entonces se introducen criterios

de desempate: el primero y muy común es el orden lexicográfico de los códigos; el segundo y que desempatará cualquier par de códigos, es la enumeración de los vértices (pero que es variable según la asignación inicial, por lo que se deja hasta el final).

Algoritmo 1 CalculaFormaCanonica

Entrada: Listas V, E, L_{VEV} de un grafo G.
Salida: formaCanonica, Cola de códigos $l_{(v_i,v_j,l_{(v_i,e,v_j)})}$, ordenada de acuerdo a las reglas DFLE.

```
1: lcc ← GenerarLCC(V, E, L_VEV) {Genera lista de códigos l_(v_i,v_j,l_(v_i,e,v_j))}
2: Ordenar lcc respecto a reglas DFLE
3: Insertar lcc[1] en Sucesores {Sucesores es cola de prioridad bajo DFLE}
4: Marcar lcc[1] como LISTO
5: mientras Sucesores ≠ ∅ hacer
6:    codlcc ← sacar dato de Sucesores
7:    Marcar codlcc como VISITADO
8:    Insertar codlcc en formaCanonica
9:    Insertar Adyacentes(codlcc) en Sucesores
10: fin mientras
11: devolver formaCanonica
```

Con el Algoritmo1 se genera un código LCC ordenado bajo las reglas DFLE, que incluye todos los vértices y arcos del grafo G_M.

En el algoritmo se hace uso de un método llamado *GenerarLCC*, dicho método toma las listas V, E y L_{VEV} y devuelve una lista de códigos $l_{(v_i,v_j,l_{(v_i,e,v_j)})}$ que describen a un arco por la enumeración de sus vértices y su etiqueta l_{vev}. Posteriormente la lista que se crea con *GenerarLCC* se guarda en *lcc* y es ordenada de acuerdo a las reglas DFLE. La creación de la lista de códigos $l_{(v_i,v_j,l_{(v_i,e,v_j)})}$ es un procedimiento que puede realizarse sin ser incluido en el algoritmo y solamente pasar la lista resultado como un parámetro en lugar de las tres listas V, E y L_{VEV}.

Luego de tener ordenada la lista de códigos, se toma a su primer elemento y se inserta en *Sucesores*, acto seguido es marcado con un estado de *LISTO*, que indica que está preparado para ser usado.

Una vez que se tiene un elemento en *Sucesores*, quiere decir que se ha elegido una raíz para la FC y se comienza con el proceso de exploración usando la técnica BFS[1]. Para formar la lista con códigos $l_{(v_i,v_j,l_{(v_i,e,v_j)})}$ que representará a la FC, se prosigue sacando el primer dato de *Sucesores*, se cambia su estado a

[1]Hay que recordar que en éste trabajo, al decir BFS se hace referencia a una Búsqueda Primero el Mejor y no a una Búsqueda Primero en Amplitud

VISITADO e inserta a la FC, *formaCanonica* en el caso de este algoritmo. A continuación se procede a insertar en *Sucesores* a los adyacentes de la arista cuyo código fue recién agregado a la FC. Acto seguido se repiten las operaciones de sacar el primer dato de *Sucesores*, cambiar el estado del código e insertar nuevos adyacentes a *Sucesores*. Hay que tener muy en cuenta que jamás se insertan códigos repetidos o que ya han estado en *Sucesores*. También el hecho de que *Sucesores* es una cola de prioridad, por lo que aún insertando nuevos elementos, dicha estructura siempre se encontrará ordenada de acuerdo a las reglas DFLE.

El algoritmo *CalculaFormaCanonica* devolverá como resultado una FC del grafo G_M. La FC representará al grafo G_M de manera única y también a otros grafos si es que son isomorfos a G_M. En caso de que un grafo G_B no sea isomorfo a G_M, pero sí uno de sus subgrafos, es decir, que exista un grafo $G_B^S \subseteq G_B$: $G_B^S \cong G_M$, entonces la FC de G_M no será la misma que la de G_B aunque parte de su estructura sea isomorfa.

La estructura *formaCanonica* es una LCC (Lista Canónica de Códigos). Pero en el proceso de detección de isomorfismos también se necesitan las denominadas en éste trabajo LCD's (Lista Canónica Derivada). Para crear una LCD se emplea el mismo algoritmo *CalculaFormaCanonica*, variando la manera de obtener la raíz. Para esto se elimina la raíz de la actual FC y a continuación se toma el ahora primer código, que será insertado en *Sucesores* y se cambiará su estado por *LISTO*. Una vez que se tiene raíz el procedimiento continúa como en Algoritmo1, a partir de la línea 5. Una LCD es una LCC, por lo que al hacer el procedimiento se obtiene una FC más pequeña, en la que notamos que no se incluye el código que fue raíz anteriormente.

El uso del método *CalculaFormaCanonica* se realizará tanto como lo requiera *EmparejarGrafos*, método que busca isomorfismos y estudiaremos a continuación.

3.2. Algoritmo para la detección de Isomorfismos

El proceso de identificación de isomorfismos tiene la tarea de detectar si dos determinados grafos son isomorfos y en caso de que no sea así, buscar subgrafos dentro del grafo de búsqueda G_B, que sean isomorfos al grafo de muestra G_M o a uno de sus subgrafos.

En la tarea de detección de isomorfismos se hace uso de diversas rutinas esenciales para una búsqueda e identificación claras e indiscutibles. Las rutinas presentadas a continuación son la base del algoritmo de detección y las que lo hacen ver, de alguna manera, simple.

3.2.1. Calcular Formas Canónicas

Ya se ha explicado la manera en que se calculan las formas canónicas y qué representan. Durante la identificación de isomorfismos se calcula una FC y las FC's derivadas que se requieran. La primera FC es una LCC y las siguientes FC's, es decir, las FC's derivadas que se calculen, serán LCD's. La primera FC es una representación de todo el grafo G_M y las FC's derivadas son representaciones de subgrafos de G_M.

3.2.2. Identificación de Expansiones

La *Identificación de Expansiones* es un proceso en el cual se revisan los patrones que se tienen hasta el momento. La intensión es la de encontrar ninguno, uno o más de un patrón que comparta aristas emparejadas con algún o algunos de los códigos de la FC con la que se está trabajando en el momento. El objetivo es el de marcar los patrones que tienen aristas en común para que en el futuro otros métodos los identifiquen rápidamente y procedan a buscar expansiones para ellos.

Para que un patrón pueda ser marcado para su expansión, no basta con que tenga aristas en común con la actual FC. Las aristas encontradas como iguales tanto en el patrón como en la FC deben tener cierta correspondencia, es decir, si en el patrón se encuentra una sola arista que coincide con otra en la FC, entonces se puede marcar el patrón para una *posible* expansión en futuros procesos. En realidad es suficiente una sola coincidencia para poder ser marcado.

¿Qué pasa si se encuentran más de un par de aristas que son iguales entre el patrón y la FC? La respuesta: *EL PATRÓN PUEDE SER MARCADO*; pero se tendrá que tener cuidado cuando se pretenda buscar una expansión, ya que las aristas que parecen ser iguales, tendrán que compartir su topología, o sea, las relaciones que existan de un lado, el patrón por ejemplo, tienen que coincidir del otro lado, la FC, al igual que las posibles expansiones, ésto se verá más adelante.

3.2.3. Descubrimiento de Raíces

Para poder localizar isomorfismos entre grafos o subgrafos se requiere tener una estructura que comparar, al igual que para poder decidir si un patrón (un subgrafo) se podría expandir. El *Descubrimiento de Raíces* es un proceso que busca y entrega las primeras estructuras o subgrafos que son isomorfos. Éstas primeras estructuras se denominan raíces y son los primeros y más pequeños subgrafos isomorfos que se descubren, tales subgrafos son de tamaño *1 arista*.

La manera de producir éstas raíces es la siguiente:

1. Calcular FC (hay que recordar que la FC se calcula sobre el grafo G_M);
2. Se toma el primer código de la FC, digamos FC_1 (su raíz);

3. Se buscan coincidencias de FC_1 en las aristas que conforman el grafo G_B, cada coincidencia es una raíz;

4. Cada raíz encontrada formará un nuevo patrón, dicho patrón estará listo para expansión.

La búsqueda de raíces se realiza tantas veces como lo requiera el algoritmo principal de emparejamiento de grafos.

3.2.4. Expansión de Patrones

Para hacer un descubrimiento de isomorfismos de calidad es necesario poder reportar subgrafos $G^S = (V^S, E^S, L_V^S, L_E^S, \alpha, \beta)$ en los cuales $|V^S| \geq 1$; tamaño que tienen los subgrafos devueltos por el proceso de *Descubrimiento de Raíces*. Para cumplir dicha necesidad se debe buscar una ampliación de los subgrafos iniciales.

Una vez que se han realizado ciertos procesos como *Calcular la FC* del grafo G_M, *Identificar Expansiones* de patrones con opción de crecer y *Descubrir Raíces* que dan lugar a nuevos patrones que se pueden expandir, es entonces que se puede realizar el proceso de *Expansión de Patrones*, uno por uno todos los patrones marcados como aptos para expansión.

La expansión se lleva a cabo considerando el orden de aparición de los códigos L_{VEV} de la lista que contiene la FC, de tal manera que un nuevo elemento $(v_u, v_v, l_{(v_u,e,v_v)})$ se apendiza a la lista de códigos del patrón en cuestión si $l_{(v_u,e,v_v)} \in L_{VEV}$ de G_M es idéntico al código L_{VEV} de la i-ésima posición de la FC.

Cada que un código es tomado para ser candidato a formar parte de la expansión de cierto patrón, hay que tener en cuenta que cumpla con las mismas adyacencias que el código de la FC con el que se le está asociando, Figura 3.1 [2]. De la misma manera, si se llegasen a encontrar dos o más códigos en la lista de L_{VEV} del grafo de búsqueda G_B, que son idénticos al L_{VEV} de la FC al que pueden ser asociados y también cumplen con las adyacencias necesarias, se tendrá que proceder de la siguiente manera para realizar los emparejamientos correspondientes:

1. Crear $n - 1$ (n = número de códigos concordantes) clones del patrón en expansión;

2. El primer código $(v_u, v_v, l_{(v_u,e,v_v)})$ que puede ser emparejado es añadido a la lista de códigos del patrón original;

[2] En la Figura 3.1.c), las aristas representadas por líneas continuas conforman al subgrafo de b) que es isomorfo a a). En c), la arista 2, 3 vista por una línea punteada, no puede sustituir a la arista 2, 1 ya que no cumple con las adyacencias requeridas 1, 3, 3, 4; de ser así, se tendrían dos isomorfismos, uno formado con la arista 2, 5 y el otro con la arista 2, 3.

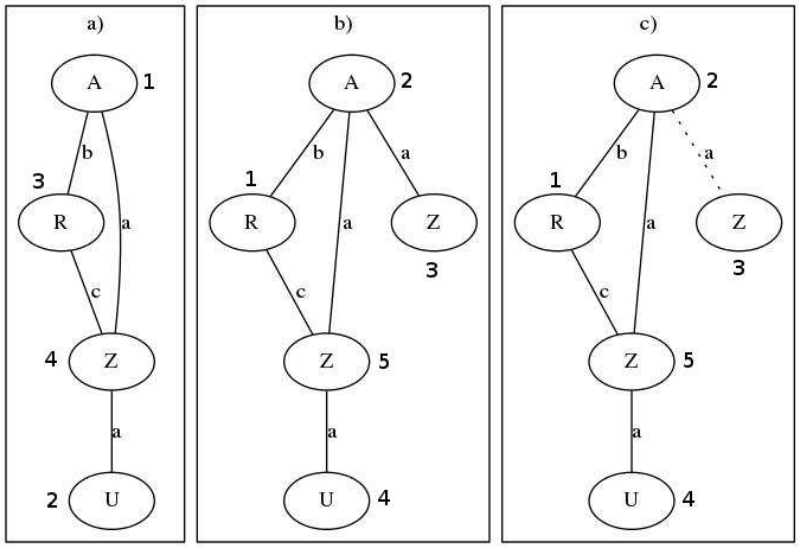


Figura 3.1: En c) se muestra el subgrafo de b) que es isomorfo a a). El arco punteado de c) no puede formar parte del patrón ya que no cumple con las adyacencias necesarias.

3. Los códigos concordantes restantes se insertan uno por cada clon creado del patrón en cuestión.

Cada uno de los clones que se haya creado, es marcado como candidato a ser expandido. Todos los patrones y clones son expandidos uno a uno.

3.2.5. Algoritmo

Ahora que se conocen los principales módulos podemos explicar el funcionamieno del algoritmo de detección de isomorfismos.

El primer paso en éste proceso es la creación de una FC primaria, de la cual nos apoyamos para hacer el primer ciclo de descubrimientos. Siempre que se genera una FC, ya se la primera o una FC *derivada*, el resultado es un *vector* ordenado bajo ciertos criterios, las reglas DFLE en este caso específico.

Calculada la FC (*formaCanonica*, para el algoritmo presentado) se entra en un ciclo cuyo propósito es hacer crecer el tamaño de los patrones. Las formas en que se puede salir del bucle son tres:

- La estructura que contiene la FC (*formaCanonica*) se encuentra vacía;
- El grafo G_B no tiene arcos, es decir, es un grafo tamaño $|E| = 0$;

- Las dos condiciones anteriores.

Algoritmo 2 EmparejarGrafos

Entrada: G_M grafo de muestra y G_B grafo de búsqueda.
Salida: Patrones, Lista de listas de códigos $l_{(v_i,v_j,l_{(v_i,e,v_j)})}$.

```
1: formaCanonica ← CalculaFormaCanonica(V, E, L_VEV) {Los parámetros que recibe son estructuras del grafo G_M}
2: mientras G_B y formaCanonica ≠ ∅ hacer
3:    MarcarExpansiones(Patrones)
4:    Insertar ObtenerRaices(formaCanonica[1]) a Patrones
5:    ExpandirPatrones(Patrones)
6:    EliminaMatcheados(G_B)
7:    formaCanonica ← CalculaCanonicaDerivada(formaCanonica)
8: fin mientras
9: devolver Patrones
```

Una vez que se entra al ciclo, lo primero es marcar dentro de los patrones, aquellos que podrían ser expandidos. Aquí tenemos dos posibilidades dependiendo del estado en el que se encuentre la ejecución del algoritmo:

1. Si es la primera iteración que se realiza la tarea de marcado termina, ya que no existen estructuras que puedan ser marcadas y se continúa;

2. En otro caso, se procede a buscar candidatos para la expansión como se explicó en la sección 3.2.2.

Proseguimos a buscar todas las posibles nuevas raíces que puedan dar lugar a otros patrones, para esto se toma el primer elemento de la FC, siendo éste el código raíz que buscaremos emparejar con algunos otros existentes dentro del grafo G_B. El procedimiento de selección de nuevas raíces ya ha sido explicado. Ya que se tienen las nuevas raíces, éstas son insertadas cada una como una lista independiente a la estructura que contiene todos los patrones encontrados (en Algoritmo 2 la estructura se llama *Patrones*). Una vez insertadas las raíces como nuevos patrones, se marcan para ser expandidos junto con los que fueron seleccionados un paso atrás.

Hechos los procedimientos de *marcado y obtención de nuevas raíces*, el algoritmo continua con la *expansión* de patrones marcados, el procedimiento de expansión es llevado a cabo un patrón a la vez y está detallado en la sección 3.2.4. de éste trabajo. Hay que tener presente cuáles de los arcos del grafo G_B han sido emparejados. Si un arco fue emparejado en un patrón, es agregado a una lista; si es emparejado a dos o más, solo aparecerá una vez en dicha lista. El control de arcos del grafo G_B que son emparejados con alguno de los códigos de la FC se debe a que el siguiente paso en el algoritmo es *eliminar* de G_B

todos los arcos que fueron emparejados durante la iteración. La eliminación de arcos dentro del grafo de búsqueda G_B, reduce futuras comparaciones y agiliza el proceso.

Eliminados los arcos que han sido emparejados, se ha concluido con el crecimiento de los patrones dada cierta raíz. Ahora para poder descubrir nuevas raíces y volver a expandir, es necesario recalcular la FC a usar. Para obtener una nueva FC se toma como base a la actual FC y se lleva a cabo un proceso que genera una FC'(Forma Canónica Derivada, para más referencias dirigirse a la Definición 18 de Capítulo 2, sección 3.1.). La FC'se obtiene a través del mismo algoritmo con el que se genera la FC, pero su raíz es tomada como el segundo código de la FC actual. El primer código de la FC actual es desechado.

El algoritmo repetirá todos los pasos del ciclo hasta que se cumpla alguna de las condiciones antes mencionadas. Una vez alcanzado el fin del ciclo, solo queda devolver la lista de patrones que representan los *subgrafos isomorfos* al grafo G_M o alguno de sus subgrafos.

3.3. Aviso

Mencionemos y hagamos reflexión sobre ciertas cosas. Debido que el algoritmo elimina aristas del grafo G_B, es posible que en futuras iteraciones, las aristas que pudieron ser una raíz ya existan, habiendo la posibilidad de que el subgrafo que se pudo haber formado, no se encuentre. Conforme a las reglas por las que se rige el algoritmo, tomando en cuenta el método que genera la FC, tipo de recorrido sobre los grafos, las podas y el hecho de que al generar las FC derivadas se elimina un arco por cada derivación, también se pueden llegar a perder subgrafos isomorfos.

Dichas las cuestiones indican que el algoritmo propuesto no es completo y en algunos casos no reportará todos los subisomorfismos existentes, sin embargo, reportará la mayoría en un tiempo de ejecución aceptable.

Capítulo 4

Resultados

Para examinar el funcionamiento del sistema que implementa al algoritmo de descubrimiento de isomorfismos, se realizaron pruebas con diferentes grafos de control. Los experimentos propuestos se llevaron a cabo con grafos de topologías tipo árbol, anillo, estrella, malla, etc. Entre los grafos de pruebas hay simples, con ciclos, completos y con características diversas.

4.1. Parámetros de entrada del sistema

El sistema toma inicialmente dos grafos G_M y G_B, Grafo de Muestra y Grafo de Búsqueda respectivamente. Tales grafos deben estar descritos en el formato solicitado por el sistema SUBDUE [23], con una linea extra al inicio de la descripción, indicando el nombre del grafo. Si es necesario hacer anotaciones o comentarios, éstos se realizarán de manera tradicional, anteponiendo un signo % (signo de porcentaje) en la linea del comentario.

```
1 %grafo1
2 v 1 A
3 v 2 U
4 v 3 R
5 v 4 Z
6 u 1 3 b
7 u 1 4 a
8 u 2 4 a
9 u 3 4 c
```

Formato de SUBDUE

```
1 Grafo1
2 %grafo1
3 v 1 A
4 v 2 U
5 v 3 R
6 v 4 Z
7 u 1 3 b
8 u 1 4 a
9 u 2 4 a
10 u 3 4 c
```

Nuestro formato

Figura 4.1: Formato de entrada de los grafos.

Normalmente el formato de SUBDUE hace diferencia cuando al agregar aristas se trata de aristas dirigidas o no dirigidas y encontramos los identificadores **e** (arista, por su sigla en inglés), que es dirigida a menos que se especifique lo contrario; **d** (dirigida, por su sigla en inglés), siempre es dirigida; **u** (no dirigida, por su sigla en inglés), siempre es no dirigida. En el caso de nuestro sistema se trabaja con grafos no dirigidos, por lo que las aristas se toman como no dirigidas y no importa que identificador se use.

En la Figura 4.1 vemos los formatos de ambos sistemas, SUBDUE y el nuestro. Es notorio el parecido de los formatos, en realidad solo varía en una linea, la primera, indicando el nombre del grafo. Sin embargo, se vé que ambos formatos incluyen el nombre del grafo como comentario. Nuestro sistema puede aceptar el primer formato de forma correcta, ya que leerá la primera linea como nombre del grafo, aquí no interesa si es comentario o no.

Para poder ejecutar una prueba en el sistema se necesitan los siguientes parámetros:

- Dirección del grafo de muestra;
- Dirección del grafo de búsqueda;
- Dirección del archivo de salida;
- Tamaño mínimo de los patrones a reportar (parámetro opcional).

El tamaño del patrón está dado por el número de aristas de su subgrafo, es decir, $|E|$ la cardinalidad del conjunto de aristas.

4.2. Sintaxis de salida del sistema

El sistema devuelve un resultado describiendo los patrones encontrados de la siguiente manera:

1. Dos líneas indicando los grafos de muestra y búsqueda del respectivo experimento;
2. Una tercera línea con el total de patrones de tamaño mayor o igual al n indicado en los parámetros del sistema (si no se indico $n \geq$ a 0);
3. Separados por una línea en blanco, los patrones descritos como a continuación:
   - En la primera línea el número de patrón, tamaño y la iteración en que fue creado;
   - k líneas describiendo los vértices involucrados;
   - m líneas describiendo las aristas involucradas.

4. Una linea mostrando el tiempo de CPU en milisegundos;
5. Línea final con el número total de iteraciones realizadas.

En las siguientes secciones se mostrarán ejemplos de los grafos utilizados para las pruebas y cómo resultaron los experimentos, así como un caso de estudio y una tabla en la que se pueden visualizar los tiempos de ejecución y cantidades de patrones reportados.

4.3. Grafos con topología Árbol

En la tarea de detección de isomorfismos, los grafos con topología de árbol son de los más sencillos de identificar debido a la ausencia de ciclos, ya sea si se buscan en un grafo tipo árbol o en cualquier otro. En los experimentos se usaron

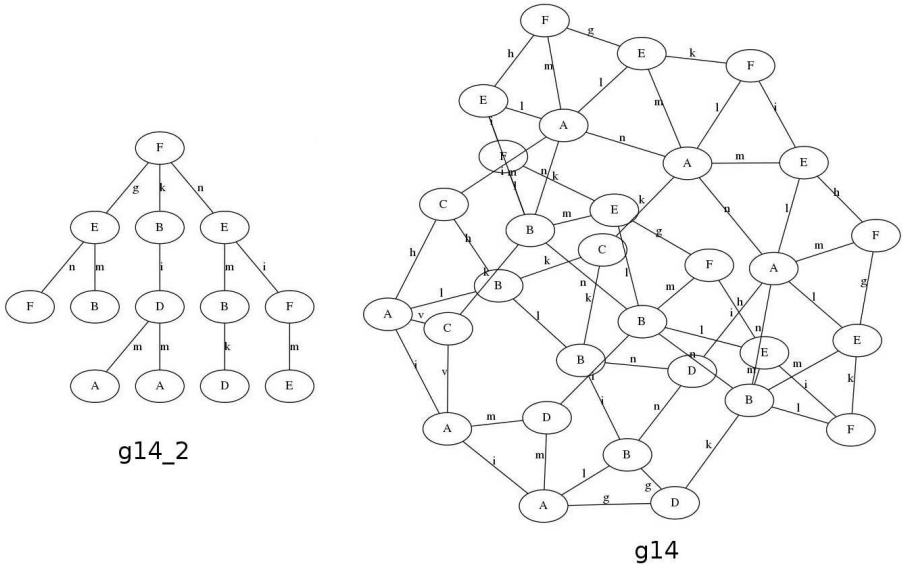


Figura 4.2: Ejemplo de árbol y grafo usados en uno de los experimentos.

pocos grafos con una topología puramente de árbol, en su mayoría los grafos con topología árbol también incluyen alguna como malla o con un subgrafo con ciclos. Debido a que la mayoría de los grafos que se tienen son híbridos, las pruebas en donde hay árboles son por lo general contra un grafo que no es árbol. Una ejecución que busca un árbol en otro grafo es la realizada entre los grafos que tienen los nombres *g14_2* y *g14*, que podemos ver en la Figura 4.2.

El resultado de la búsqueda de *g14_2* en *g14* nos dice que se encontraron 5 patrones, es decir, 5 subgrafos de *g14_2* que tienen isomorfismos en *g14*. De

éstos patrones, 1 de ellos tiene tamaño de 3 aristas, 2 tienen tamaño de 2 aristas y los 2 restantes tamaño 1 arista.

4.4. Grafos con topología Malla

En grafos con ésta topología siempre hay un camino de algún vértice a a un vértice b. Los grafos de este tipo con los que se trabajaron para los experimentos varían en cardinalidad de vértices y aristas, es el caso de los grafos presentados en la Figura 4.3.

Para el experimento que involucra a los grafos de la Figura 4.3, *gGrid* se utilizó como el grafo G_M a buscar y *g16* como el grafo G_B de búsqueda. El resultado de buscar subgrafos isomorfos de G_M en G_B fue de 60 subgrafos, todos con cardinalidades $|V| = 4$ y $|E| = 3$. Hay que tener presente que siempre se trata de encontrar los patrones más grandes, por lo que en éste caso, no se reportaron subgrafos con cardinalidad $|E| = 2$ o $|E| = 1$ que evidentemente existen, pero que son subgrafos de otros con mayor tamaño.

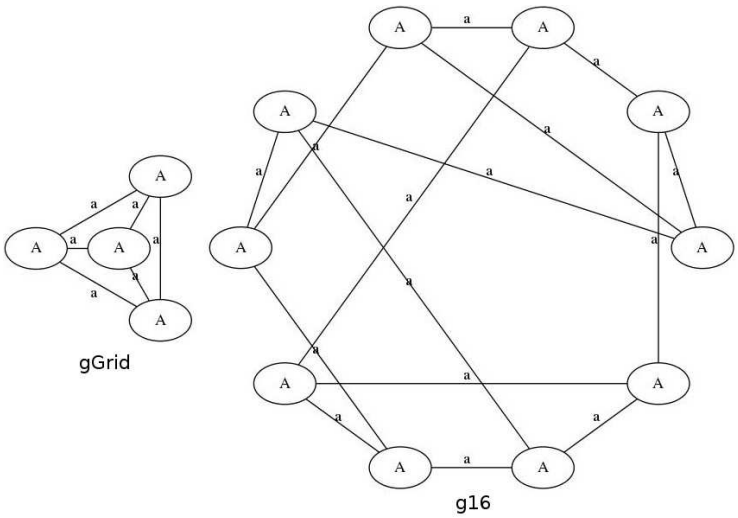


Figura 4.3: Dos grafos tipo malla. gGrid también es un grafo completo.

4.5. Grafos con topología híbrida

Para poder lograr buenos experimentos es necesario tener un conjunto de grafos que si bien no tienen porque ser enormes, sí tienen que tener ciertas características, cumplir con propiedades sobre las que se conoce que pueden haber complicaciones al hacer descubrimiento. Por esta razón es que muchos

de los grafos con los que se trabajaron en los experimentos tienen topología híbrida y combinan propiedades entre proporción de vértices y arcos, así como características sencillas que introducen complicaciones.

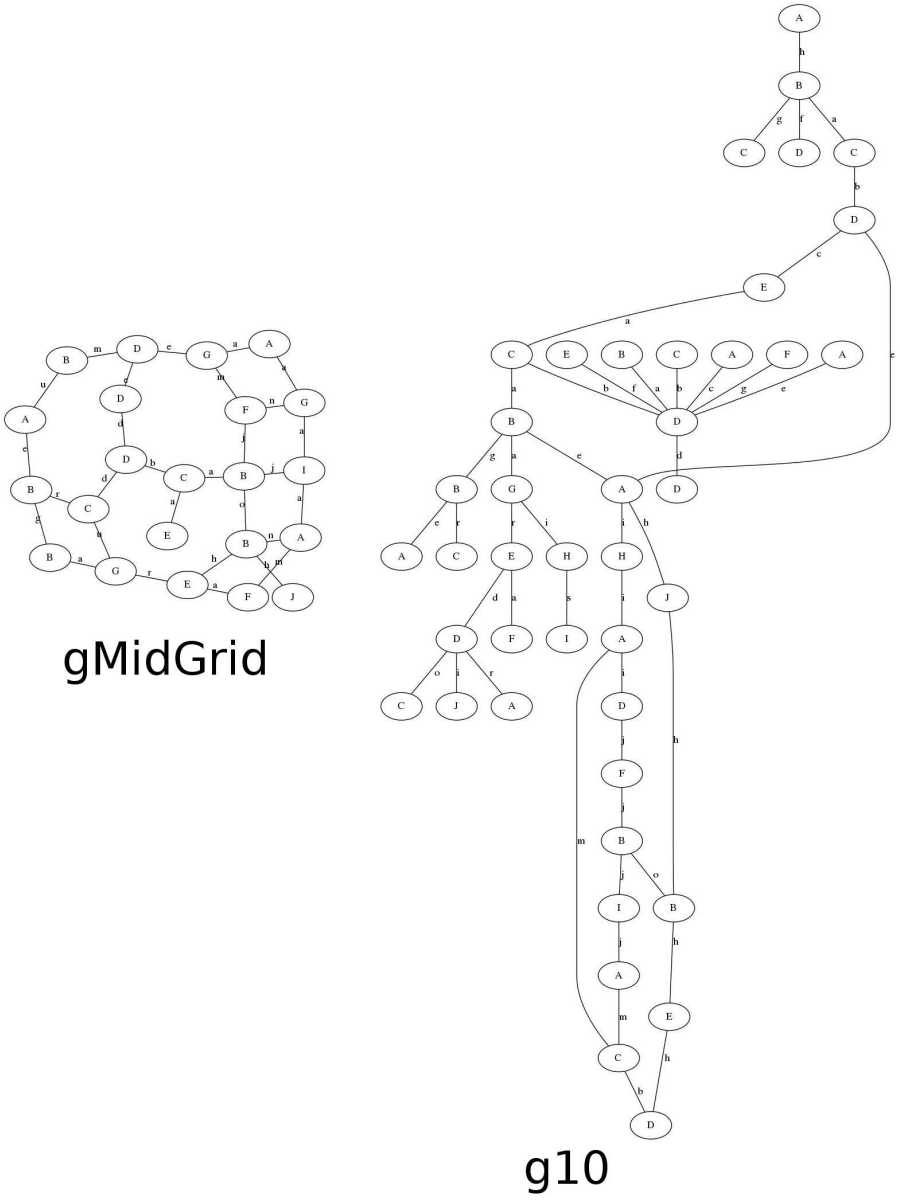


Figura 4.4: Grafo gMidGrid tipo malla y g10 tipo híbrido.

Veámos como ejemplo la Figura 4.4, tomemos al grafo *gMidGrid* como G_M y al grafo *g10* como G_B. Nuestro grafo G_M tiene una topología de tipo red o malla, es fácil ver que tiene ciclos y vértices conectados al grafo por solo un arco. El grafo G_B, es un grafo con topologías mezcladas, podemos distinguir subgrafos tipo árbol, estrella y malla; también observamos ciclos anidados. A simple vista es notable la diferencia de estructuras y la forma de etiquetado, sin embargo, después de realizar una prueba utilizando estos grafos, se encontraron: 7 patrones; 1 de tamaño 1, 3 de tamaño 2, 1 de tamaño 4, 1 de tamaño 5 y 1 de tamaño 6, es decir, de cardinalidades $|E| = 1$, $|E| = 3$, $|E| = 1$, $|E| = 1$ y $|E| = 1$, respectivamente.

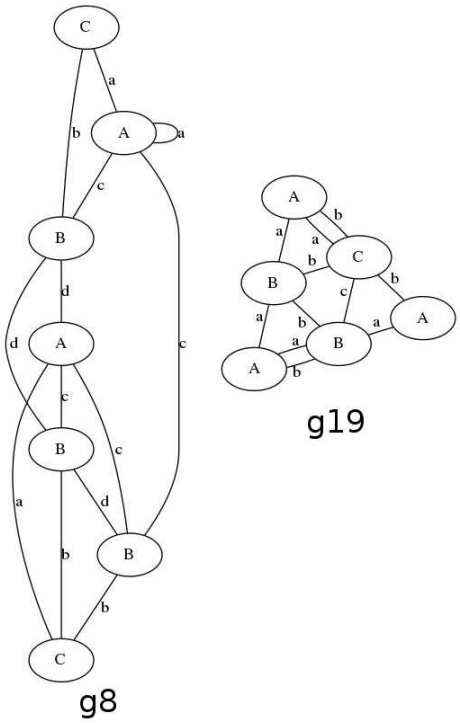


Figura 4.5: Los grafos presentados cuentan con diversas propiedades como bucles y cliqués.

Otro ejemplo en donde vemos topologías mezcladas son los grafos *g8* y *g19* de la Figura 4.5, de nuevo diremos que *g8* será G_M y *g19* tendrá el papel de G_B. G_M tiene una mezcla de las topologías estrella y malla, a diferencia del ejemplo anterior, ahora podemos observar que G_M también cuenta con un bucle y varios cliqués. G_B es tipo malla y también tiene varios subgrafos que cumplen con ser cliqués, además de que G_B es multigrafo, al igual que algunos de sus subgrafos. Al experimentar con estos grafos los resultados obtenidos fueron de: 1 patrón de tamaño 2.

4.6. Grafos con Ciclos

Grafos que contienen ciclos presentan dificultades al hacer búsqueda de isomorfismos, ya que al tomar un camino que aparentemente expandirá el grafo, podríamos estar dando fin a la expansión; o quedar con el patrón del tamaño de un ciclo siendo que tal vez existen ciclos anidados que corresponden al grafo que queremos encontrar.

Tenemos en la Figura 4.6 dos grafos, cada uno con el mismo número de ciclos y a pesar de que se vean diferentes su topología es la misma, aunque no todas sus etiquetas. La respuesta del sistema de tratar de encontrar isomorfismos de *g3* en *g3_2* es de: 1 patrón encontrado de tamaño 7, esto es un subgrafo de *g3_2* que incluye todas las aristas coincidentes entre los 2 grafos, y tal subgrafo tiene $|E| = 7$.

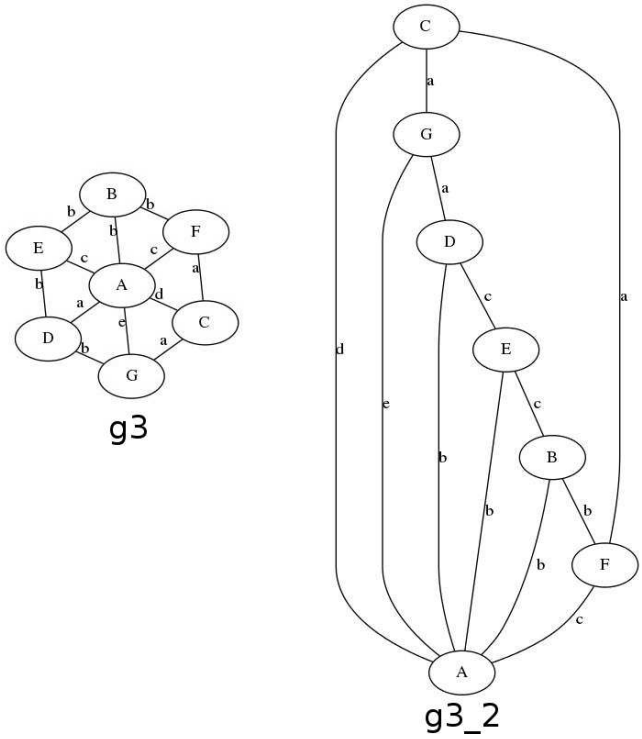


Figura 4.6: Dos grafos con la misma estructura topológica.

Si nos fijamos en la Figura 4.3 podemos notar que ambos grafos tienen bastantes ciclos y en el caso de *g16* son muchos los ciclos anidados. Del experimento con los grafos de esta figura ya se han dicho los resultados.

4.7. Grafos Completos

Al hacer experimentos sobre grafos completos, incrementa en gran medida la cantidad de comparaciones para verificar las adyacencias y correctas correspondencias entre grafos, por lo que estamos ante un reto en cuanto a tiempo de solución. Como se ha comentado, en estas pruebas no se buscó que los grafos fueran muy grandes, sino más bien con las características correctas para probar lo mejor posible el sistema.

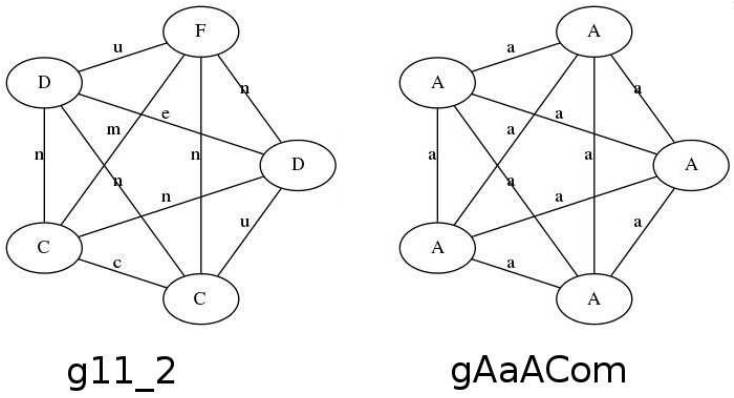


Figura 4.7: Grafos completos con igual estructura topológica.

Para el caso de los grafos en la Figura 4.7, con *g11_2* como G_M y *gAaACom* como G_B, el sistema reportó 0 patrones encontrados. Ésto es correcto ya que a pesar de tener una estructura idéntica, ninguna de sus etiquetas coincide.

Cuando se explicó el algoritmo de descubrimiento de isomorfismos, se dijo que se producirían tantas FC *derivadas* como se requirieran; en este experimento se generan todas las FC posibles a partir del grafo G_M que es *g11_2*. Aunque se generaron todas las FC a partir del algoritmo establecido en las condiciones especificadas, no se obtienen patrones por la principal razón de que ninguna raíz logra tener una correspondencia en G_B.

4.8. Grafos con etiquetas iguales y distintas

En algunas de las pruebas que se realizaron, fueron utilizados grafos con todas las etiquetas iguales, de esa manera dependiendo de la estructura de cada grafo, se podrían encontrar subgrafos isomorfos del tamaño del grafo de muestra G_M o subgrafos pequeños de no más de unos cuantos pares de aristas.

Ya hemos visto un ejemplo en el que el grafo G_M tiene etiquetas distintas entre sus arcos y el grafo G_B todas las etiquetas iguales, pero no coinciden con

alguna de G_M. Es el caso del experimento de los grafos en la Figura 4.7, en donde el resutado es de 0 patrones.

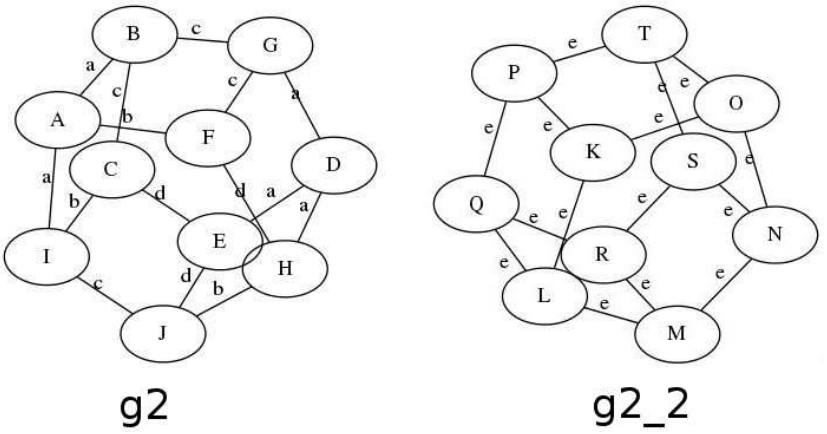


Figura 4.8: Grafos con topología igual y etiquetado distinto.

Ahora, experimentando con *g2* y *g2_2* de la Figura 4.8, siendo *g2* el grafo G_M y *g2_2* el grafo G_B, $\nexists l_{(v_i,e,v_j)}, l_{(v_u,e,v_v)} : l_{(v_i,e,v_j)} \in L_{VEV}$ de G_M y $l_{(v_u,e,v_v)} \in L_{VEV}$ de G_B, con $l_{(v_i,e,v_j)} \cong l_{(v_u,e,v_v)}$. Dicho de otro modo, observamos que todas las etiquetas de sus vértices son diferentes, por lo que una correspondencia de aristas tomando en cuenta vértice-arco-vértice, es imposible. Ésto sin importar que todas las etiquetas de las aristas de G_B sean iguales. El resultado que tenemos de buscar isomorfismos es de 0 patrones encontrados.

Ahora tenemos otro caso en Figura 4.9 donde *g17* tiene cardinalidad $|L_V| = |L_E| = 1$, sin embargo hay que notar que la etiqueta de vértices es distinta a la etiqueta de aristas[1] y *g5* repite etiquetas tanto en vértices, aristas y L_{VEV}. De nuevo, renombremos, siendo *g17* y *g5* los grafos G_M y G_B respectivamente. Podría no ser claro, pero G_M y G_B tienen una estructura topológica idéntica, por otro lado su etiquetado es todo lo contrario. Al hacer la prueba con tales grafos tenemos 0 patrones encontrados que es lo esperado.

[1]A los grafos con cardinalidades de 1 en etiquetas de vértice y aristas con las etiquetas idénticas, es decir, todas A o todas a para vértices y aristas, por ejemplo, se les llama *grafos no etiquetados*, dado que serían igualmente identificables si no las tuvieran.

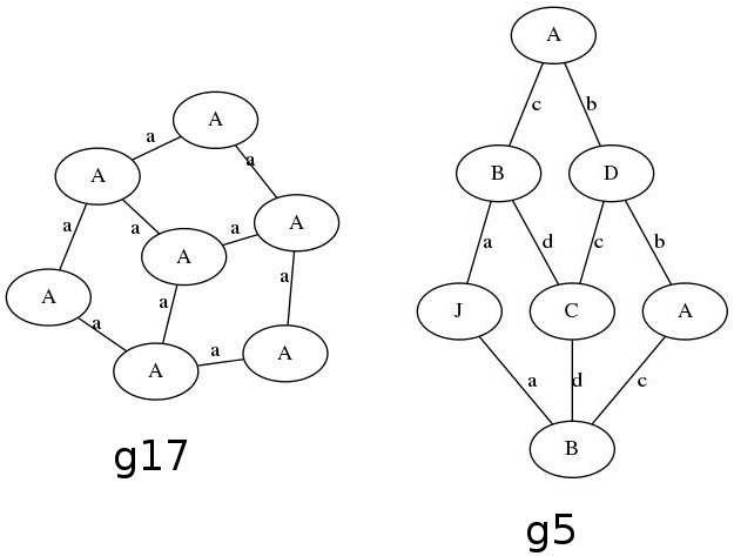


Figura 4.9: g17 es un grafo con $|L_V| = |L_E| = 1$. g5 es un grafo en el que se repiten etiquetas de vértices, aristas y también L_{VEV}.

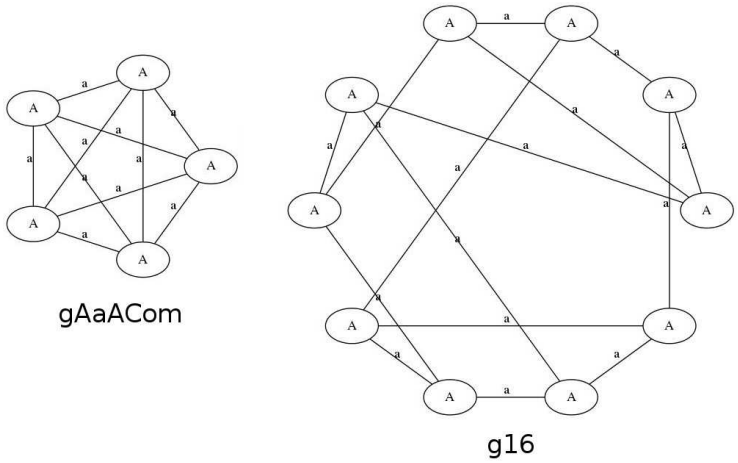


Figura 4.10: Dos grafos etiquetados igual en vértices y aristas.

Un ejemplo más, pero en el caso seguido ambos grafos tienen la etiqueta "*A*"en todos sus vértices y "*a*"como etiqueta de aristas. Estamos hablando de *gAaACom* que fungirá como G_M y *g16* que tomará el lugar de G_B, grafos que podemos apreciar en la Figura 4.10. A simple vista la topologia parece ser distinta, pero es probable que haya subgrafos que sean isomorfos. Hecho el experimento se obtuvo lo siguiente: 40 patrones de tamaño 4 y 80 patrones de

tamaño 5, teniendo un total de 120 patrones correspondientes a 120 subgrafos de G_B que son isomorfos a alguno de G_M.

4.9. Caso de análisis

Cuando se propone un nuevo paradigma, diseño, estrategía, algoritmo, etc., hay la posibilidad de que dadas algunas ventajas que se ofrecen, también se sufran ciertas pérdidas. Tal es el caso de nuestro algoritmo, como de muchos otros. A continuación estudiaremos paso a paso uno de los experimentos elaborados para la prueba del sistema.

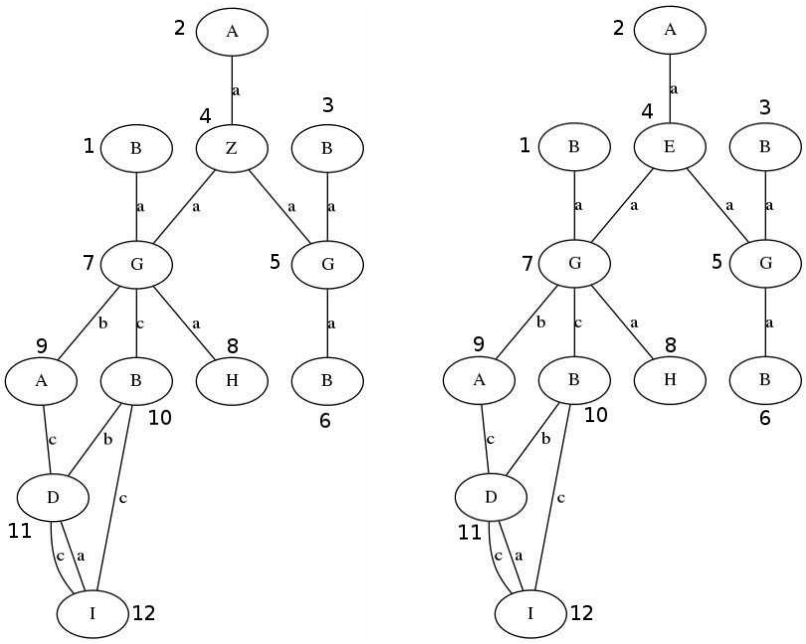


Figura 4.11: A la izquierda grafo *g50*, a la derecha grafo *g52*.

Se proponen 2 grafos llamados *g50* y *g52*, ambos con idéntica topologia y enumeración de vértices; incluso la forma en que están etiquetados sería igual de no ser por en el vértice número 4, en *g50* tiene la etiqueta "*Z*"y en *g52* está etiquetado como "*E*". Dichos grafos los podemos apreciar en la Figura 4.11.

Esta ocasión tomaremos a *g50* como nuestro grafo de muestra G_M y *g52* tomará el lugar del grafo de búsqueda G_B. Al observar ambos grafos rápidamente podemos identificar dos subgrafos en *g52* que son isomorfos a los subgrafos conformados por los mismos nodos y arcos en *g50*, en la Figura 4.12 se muestran delimitados ambos subgrafos.

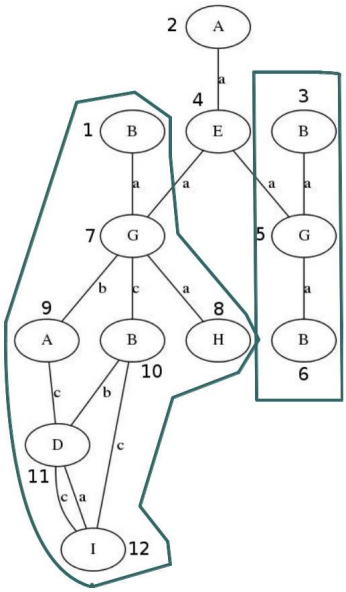


Figura 4.12: Encerrados, se muestran los subgrafos más grandes de *g52* que tienen emparejamiento en *g50*.

Comencemos con la búsqueda de patrones para poder comparar la salida del algoritmo ante estos grafos contra lo que distinguimos a simple vista.

El primer paso en la identificación con respecto al Algoritmo 2 es calcular la FC de G_M que en nuestro caso es el grafo *g50*. Presentamos el orden de los códigos dentro de la FC en el Cuadro4.1: En el paso 2 entramos a un ciclo en el que las formas de salir es que la FC de *g50* se quede sin elementos o que se hayan eliminado todas las aristas de *g52*.

Llegando al paso 3 hay que marcar los patrones que son candidatos a expandirse, pero recién empezamos el procedimiento y aún no tenemos algo que verificar. Proseguimos al paso 4 en el que hay que obtener las raíces e insertarlas al conjunto de patrones.

Tenemos que la raíz de la FC es la arista formada por los vértices 3 y 5 con etiqueta $l_{v_i,e,v_j} = BaG$, del grafo *g50*. Hay entonces que buscar aristas con ésta etiqueta. Encontramos en *g52* 3 aristas con la etiqueta buscada y creamos un patrón por cada una de las aristas que representan la raíz de cada nuevo patrón.

El estado del conjunto de patrones es: *Patrón 1* con raíz *BaG* del vértice 3 al vértice 5; *Patrón 2* con raíz *BaG* del vértice 6 al vértice 5; y *Patrón 3* con raíz *BaG* del vértice 1 al vértice 7. No hay que confundir a qué aristas

Vértice i	Vértice j	l_{v_i,e,v_j}
3	5	BaG
6	5	BaG
5	4	GaZ
2	4	AaZ
7	4	GaZ
1	7	BaG
7	8	GaH
9	7	AbG
9	11	AcD
10	11	BbD
10	12	BcI
11	12	DaI
11	12	DcI
10	7	BcG

Cuadro 4.1: Primera FC de *g50*.

hacemos referencia, éstas pertenecen a *g52*, al igual que los patrones que están en formación, no confundirse con los arcos del grafo de muestra.

Para el paso 5 vamos a expandir los patrones que se hayan marcado y los creados recientemente. En esta ocasión solo expandiremos los patrones de recién creación, no se marcaron otros porque no existen otros. Al expandir, solo se pueden agregar a los patrones, aristas que correspondan en adyacencia a las aristas en *g50* que son adyacentes a BaG conformado por los vértices 3 y 5. En la Figura 4.13 tenemos el grafo muestra *g50*, con color café se vé la arista raíz y con color naranja están marcadas las aristas adyacentes inmediatas.

Comenzamos con el *Patrón 1*, recordemos que la raíz de este patrón va del vértice 3 al 5. Adyacentes a nuestra raíz están las aristas GaE y BaG, de los vértices 5 a 4 y del 6 a 5 respectivamente. De acuerdo a las opciones que tenemos y las adyacencias que se ven en la Figura 4.13, la única arista que podemos agregar al patrón es BaG de 6 a 5, emparejado con BaG de 6 a 5 en *g50*. Después de haber insertado la arista que se pudo emparejar, ya no es posible seguir con la expansión, para esto es necesario tener una arista GaZ adyacente a nuestra raíz compartiendo el vértice con la etiqueta G; tal arista no existe en *g52*, por lo tanto no podemos expandir. Si bien es cierto que se podrían emparejar otras aristas, el patrón sería un grafo no conexo y lo que buscamos es grafos conexos.

Pasamos a expandir el *Patrón 2* con raíz que va del vértice 6 al vértice 5. La raíz tiene como aristas adyacentes a BaG de 3 a 5 y a GaE de 5 a 4. De nuevo, tomando en cuenta las aristas que tenemos como adyacentes y a las que

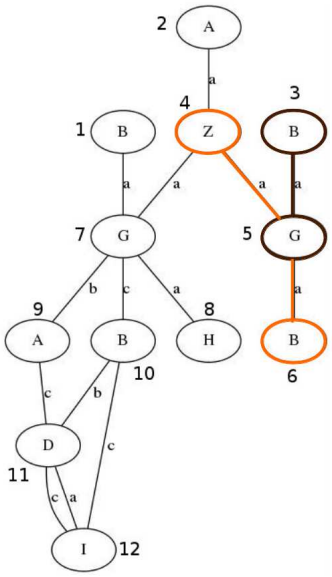


Figura 4.13: En color naranja se muestran las aristas inmediatas adyacentes a la raíz.

podemos emparejarlas, solo es viable la correspondencia entre la arista *BaG* de 3 a 5 de *g52* con la arista *BaG* de 6 a 5 en *g50*. La expansión en el patrón llega a su fin por la misma razón que el *Patrón 1* no continuó con la suya.

Contamos con un último patrón, el *Patrón 3* con raíz *BaG* que va del vértice 1 al vértice 7 del grafo *g52*. Para el patrón en cuestión contamos con 4 aristas adyacentes: *EaG* de 4 a 7, *GaH* de 7 a 8, *BcG* de 10 a 7 y *AbG* de 9 a 7. Sería fácil emparejar, o por lo menos parecería correcto hacerlo con las aristas que están etiquetadas y tienen la misma enumeración en el grafo *g50*, pero para eso nuestra raíz tendría que haber sido emparejada en primera instancia con la arista *BaG* de 1 a 7 en *g50*, esto no fue así.

Recordemos que cuando buscábamos aristas que tuvieran etiquetas iguales a las de la raíz de la FC, la raíz que seleccionamos fue emparejada con la arista *BaG* que va del vértice 3 al vértice 5 que se encuentran en *g50* (es importante tener en cuenta que grafo es nuestra base y que arista es nuestro punto de partida). Por lo tanto, son aristas como las que vemos en la Figura 4.13 con las que podemos hacer un emparejamiento. Ya que ninguna de las adyacencias con la que contamos nos es útil, una correspondencia es imposible y no se inserta arista alguna al patrón. Al no poder ser conexa una expansión posible, el *Patrón 3* quedará como un subgrafo de tamaño 1 arista.

Hecha la expansión uno por uno de los patrones, es hora de eliminar las aristas de *g52* que pudieron ser emparejadas en algún patrón, en la Figura 4.14 mostramos en líneas punteadas los arcos que desaparecen.

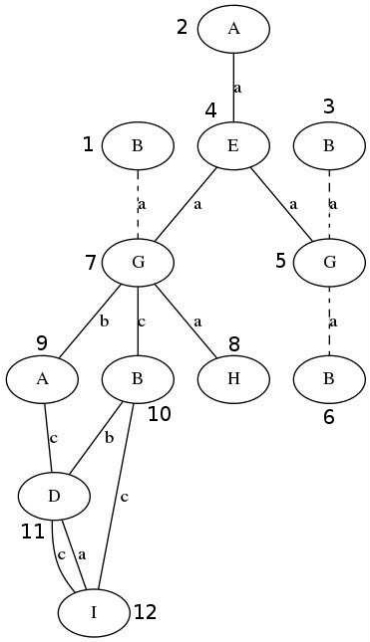


Figura 4.14: Grafo *g52*. Las líneas punteadas representan las aristas eliminadas.

Para éstas alturas está por terminar la primera iteración del algoritmo de descubrimiento de isomorfismos. Para finalizar y poder pasar a la siguiente vuelta del ciclo, solo queda recalcular la FC. Para esto se calcula la *Primera Derivada* de la *Lista Canónica de Códigos* que forma la FC. En el Cuadro4.2 tenemos cómo queda la FC después de ser recalculada.

De acuerdo a la nueva FC, volvemos a tener una raíz con la etiqueta BaG, esta vez corresponde a la arista que va de los vértices 6 a 5 del grafo *g50*. En la Figura 4.14 se puede apreciar que el grafo *g52* ya no cuenta con aristas etiquetadas de la forma BaG, entonces no se encontrarán aristas con las cuales poder hacer un emparejamiento y entonces expandir los patrones resultantes. Es por ésto que saltaremos los pasos hasta llegar de nuevo al cálculo de la próxima FC.

Durante las siguientes iteraciones, las nuevas FC tienen como raíces códigos de aristas con etiquetas que no existen en *g52*, razón por la cual no se pueden crear y tampoco expandir nuevos patrones. En las situaciones de los Cuadros4.3, 4.4, 4.5, las raíces incluyen en sus aristas un vértice con etiqueta Z, en *g52* no

existen vértices con tal etiqueta por lo que no hay emparejamiento. En el caso de la FC descrita en el Cuadro4.6, la raíz vuelve a tener una etiqueta BaG y sabemos que ya no hay aristas etiquetadas así.

Vértice i	Vértice j	l_{v_i,e,v_j}
6	5	BaG
5	4	GaZ
2	4	AaZ
7	4	GaZ
1	7	BaG
7	8	GaH
9	7	AbG
9	11	AcD
10	11	BbD
10	12	BcI
11	12	DaI
11	12	DcI
10	7	BcG

Cuadro 4.2: Primera derivación de la FC de *g50*.

Vértice i	Vértice j	l_{v_i,e,v_j}
5	4	GaZ
2	4	AaZ
7	4	GaZ
1	7	BaG
7	8	GaH
9	7	AbG
9	11	AcD
10	11	BbD
10	12	BcI
11	12	DaI
11	12	DcI
10	7	BcG

Cuadro 4.3: Segunda derivación de la FC de *g50*.

Vértice i	Vértice j	l_{v_i,e,v_j}
2	4	AaZ
7	4	GaZ
1	7	BaG
7	8	GaH
9	7	AbG
9	11	AcD
10	11	BbD
10	12	BcI
11	12	DaI
11	12	DcI
10	7	BcG

Cuadro 4.4: Tercera derivación de la FC de *g50*.

Vértice i	Vértice j	l_{v_i,e,v_j}
7	4	GaZ
1	7	BaG
7	8	GaH
9	7	AbG
9	11	AcD
10	11	BbD
10	12	BcI
11	12	DaI
11	12	DcI
10	7	BcG

Cuadro 4.5: Cuarta derivación de la FC de *g50*.

Vértice i	Vértice j	l_{v_i,e,v_j}
1	7	BaG
7	8	GaH
9	7	AbG
9	11	AcD
10	11	BbD
10	12	BcI
11	12	DaI
11	12	DcI
10	7	BcG

Cuadro 4.6: Quinta derivación de la FC de *g50*.

Es al final del sexto ciclo que se crea una FC que tiene una raíz que puede ser emparejada en *g52*. En el Cuadro4.7, el primer código que vemos y tomamos como raíz, va del vértice 7 al vértice 8 y le pertenece una etiqueta GaH.

Vértice i	Vértice j	l_{v_i,e,v_j}
7	8	GaH
9	7	AbG
9	11	AcD
10	11	BbD
10	12	BcI
11	12	DaI
11	12	DcI
10	7	BcG

Cuadro 4.7: Sexta derivación de la FC de *g50*.

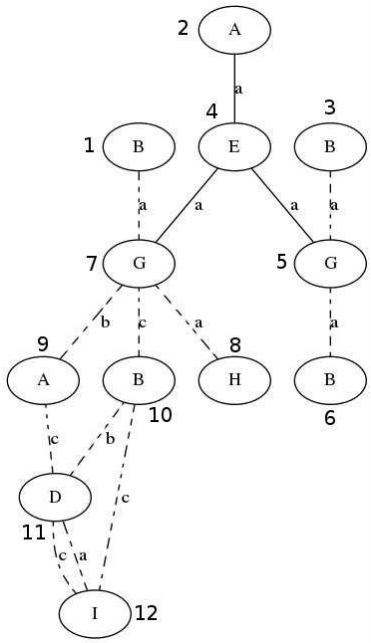


Figura 4.15: Grafo *g52* después de eliminar las aristas terminada la séptima iteración. Las líneas punteadas representan las aristas eliminadas.

Sigamos los pasos del algoritmo. Lo primero es marcar los patrones que tienen posibles expansiones pero, ninguno de los patrones contiene un código que pueda ser correspondido con alguno de los códigos de la FC. Continuamos y se encuentra una arista con las características para ser raíz y la tomamos, queda

entonces la raíz de la FC *GaH* de 7 a 8, emparejada con la arista *GaH* de 7 a 8 del grafo *g52*.

Con la nueva raíz se crea el *Patrón 4*. Se prosigue con la expansión. Siguiendo el orden de la FC hay que emparejar los códigos *AbG* de 9 a 7, *AcD* de 9 a 11, *BbD* de 10 a 11, *BcI* de 10 a 12, *DaI* de 11 a 12, *DcI* de 11 a 12 y *BcG* de 10 a 7 y se hace con las aristas de *g52* *AbG* de 9 a 7, *AcD* de 9 a 11, *BbD* de 10 a 11, *BcI* de 10 a 12, *DaI* de 11 a 12, *DcI* de 11 a 12 y *BcG* de 10 a 7 respectivamente.

Efectuada la expansión, eliminamos los arcos en *g52* que fueron emparejados, quedando el grafo como en la Figura 4.15.

Durante las últimas iteraciones se continúa derivando la *Lista Canónica de Códigos* (estructura en la que está representada la FC), pero ya no hay aristas en *g52* que puedan ser ocupadas como raíces.

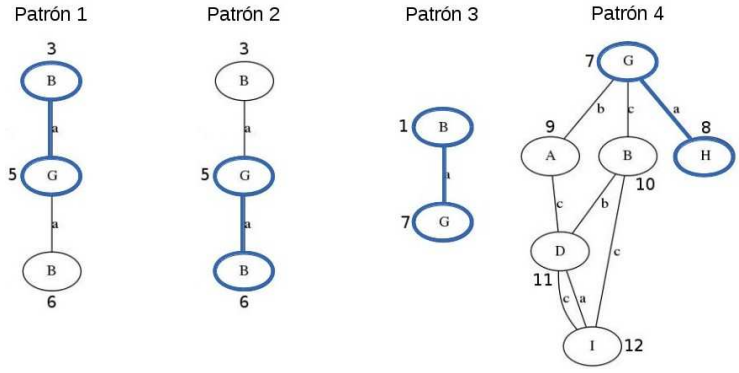


Figura 4.16: Patrones resultantes de *g52* encontrados dadas las FC de *g50*.

Al salir del ciclo tenemos como resultado 4 patrones, mostrados en Figura 4.16, en los cuales está resaltada la arista raíz. Por último en la Figura 4.17 apreciamos los subgrafos de *g52* que se localizaron como isomorfos a *g50*.

Después de haber seguido el experimiento con los grafos *g50* y *g52*, se obtuvo un resultado distinto al esperado. Como vemos en Figura 4.17 los subgrafos que dieron lugar a los patrones no son exactamente iguales a los que se proponían en Figura 4.12. Ésto no quiere decir que el algoritmo esté mal o que el sistema falle, simplemente, que conforme a las reglas por las que se rige el algoritmo, tomando en cuenta el método que genera la FC, tipo de recorrido sobre los grafos, las podas, etc., en algunas ocasiones la metodología propuesta no encontrará los patrones más grandes posibles, ya que trabaja bajo la premisa de que "*lo primero que tiene en el momento es lo mejor.*"

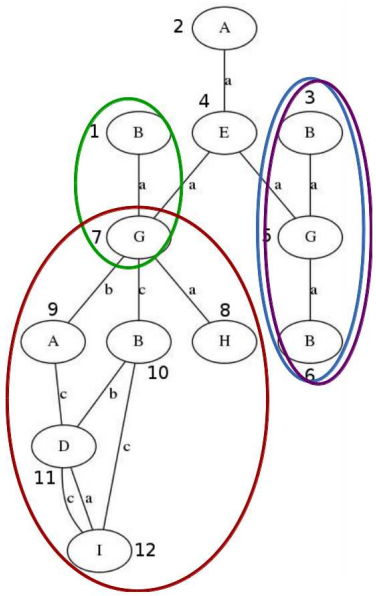


Figura 4.17: Subgrafos de *g52* que dieron lugar a los patrones que reporta el sistema.

Aún con casos como el que recién analizamos, el sistema que implementa el algoritmo propuesto tiene un buen desempeño en cuanto a cantidad de subgrafos isomorfos encontrados y el tiempo de ejecución. En la siguiente sección se presenta una tabla con los resultados de los experimentos.

4.10. Tabla de resultados

Durante las pruebas al sistema se ocuparon distintos tipos de grafos, con tamaños variables, etiquetados igual, de formas similares y tambiíen totalmente distintas. En algún caso la prueba se elaboró utilizando el mismo grafo como G_M y G_B. Otras ocasiones los grafos G_B fueron grafos inducidos del grafo utilizado como G_M, algunas otras, una vez generados los grafos inducidos, se cambiaron los papeles, siendo éstos los que se usaron como G_M y los originales como G_B.

Son diversas las maneras en que se combinaron los tipos de grafos para efectuar los experimentos y los resultados son variados, a continuación se presenta la tabla con las pruebas hechas, describiendo cada una con los grafos, cantidad de patrones encontrados y tiempo de ejecución de la respectiva prueba.

	Grafo de Muestra			Grafo de Búsqueda				T. ejecución
Exp.	V	E	Grafo	V	E	Grafo	Patrones	Milisegundos
1	4	4	g1	5	5	g1_3	2	0
2	4	4	g1	7	8	g1_4	2	0
3	10	15	g2	10	15	g2_2	0	0
4	7	12	g3	7	12	g3_2	1	2
5	7	9	g5	7	9	g5_2	3	1
6	7	9	g7	3	3	g7_2	1	0
7	3	3	g7_2	7	9	g7	1	0
8	5	6	g7_3	7	9	g7	1	0
9	5	7	g8_2	7	13	g8	4	3
10	5	7	g8_3	7	13	g8	5	2
11	5	5	g10_2	42	45	g10	4	3
12	15	30	g12	15	30	g12	3	12
13	10	20	g13_2	20	40	g13	5	36
14	4	6	gGrid	7	9	g17	36	12
15	13	12	g14_2	30	60	g14	5	12
16	5	10	gAaACom	10	15	g16	120	155
17	4	6	gGrid	10	15	g16	60	38
18	10	15	g16	10	15	g18	76	216
19	7	9	g17	7	9	g5	0	0
20	7	9	g17	10	15	g16	340	451
21	7	13	g8	6	11	g19	1	2
22	22	29	gMidGrid	42	45	g10	7	53
23	5	10	g11_2	5	10	gAaACom	0	0
24	12	14	g50	12	14	g52	4	4

Cuadro 4.8: Experimentos con diversos grafos.

El sistema fue desarrollado con las librerías estándar del lenguaje de programación C++ 11 y las pruebas se realizaron sobre una sistema operativo Fedora 19 y con las siguientes características en hardware:

- Memoria RAM de 2 GB;
- Procesador Pentium(R) Dual-Core CPU T4200 corriendo a 2.00GHz;
- Caché de procesador de 1 MB.

De los experimentos podemos decir que se obtuvieron resultados satisfactorios, producto de las propuestas de algoritmos de *Detección de Isomorfismos* en conjunto con el de *Creación de Formas Canónicas basadas en Listas Canónicas de Codigos*.

El sistema trabaja en buen tiempo en relación a la cantidad y calidad de subgrafos encontrados, siendo una buena opción de herramienta para la detección de patrones, análisis y toma de decisiones.

4.11. Comparación con otros sistemas

Siempre que se hace una propuesta para resolver un problema sobre el cual existe investigación previa, es bueno hacer una comparación con los trabajos que atacan el mismo problema.

Debido a que la mayoría de sistemas dedicados a la solución del problema de Emparejamiento de grafos son difíciles de conseguir por completo, es el sistema SUBDUE, aquel con el que se podría hacer una comparativa.

Desafortunadamente, la forma interna de trabajo de SUBDUE y nuestro sistema no permite una comparación apropiada, quedando en desventaja uno u otro sistema dependiendo del método usado por SUBDUE para los experimentos. Expliquemos.

SUBDUE podría competir con nuestro sistema principalmente con dos de sus herramientas:

- **sgiso**, herramienta que solo arroja un resultado cuando dados dos grafos, éstos son isomorfos;
- **subdue** con la opción *-ps*, haciendo el trabajo de Subisomorfismo en Grafos.

El problema con la primera opción es que solo daría el resultado de si los grafos de entrada son isomorfos, estando en desventaja contra nuestro sistema que reportará todos los subisomorfismos que encuentre, incluidos los grafos enteros si es que son isomorfos.

La cuestión con la segunda opción es que se pueden tener resultados muy variables dependiendo de los parámetros que se ocupen, siendo injusta la comparación para uno u otro sistema dependiendo de estos parámetros. Si pedimos que SUBDUE tenga o no en cuenta traslapes, reportará más o menos patrones encontrados, lo justo en éste caso es reportar todo. De igual manera, usar el parámetro de tamaño máximo para los subgrafos y todas la iteraciones necesarias para el mejor resultado posible. Nos topamos con un problema ya que al hacer ésto, SUBDUE reportará en algunos casos menos de los subgrafos que realmente encuentra, debido a que usa el principio de MDL, dejando algunos patrones dentro de otros.

El punto es que no hay una manera acertada de equilibrar los experimentos y poder hacer una comparación adecuada en cuanto a cantidad y calidad de los resultados, por lo tanto tampoco del tiempo; aunque corriendo las mismas entradas en ambos casos, SUBDUE con parámetros ajustados para dar sus mejores resultados y nuestro sistema, tenemos mejores tiempos en la mayoría de ellos.

Como se describe en párrafos anteriores, lo que en la sección actual se menciona es solo un aviso, ya que al no poder hacer una oportuna comparación no podemos decir si los resultados de un sistema son mejores que los del otro.

Capítulo 5

Conclusiones y trabajo a futuro

En el presente trabajo de tesis se ha expuesto un algoritmo para el descubrimiento de isomorfismos entre grafos, basado en lista canónicas de códigos; al igual que el algoritmo del que se apoya para la construcción de dichas listas canónicas de códigos.

Se introdujo el concepto de *Lista Canónica Derivada*. Tales listas representan FC de subgrafos que se encuentran en una FC que describe a un grafo original.

Los experimentos se ejecutaron en grafos de diversas topologías, tamaños y con etiquetas iguales y distintas. Los experimentos que se llevaron a cabo arrojaron resultados satisfactorios. El sistema que implementa el algoritmo encontró grafos y subgrafos isomorfos cuando los había, reportando completamente el grafo si fue isomorfo. En los casos en que los grafos fueron completamente distintos la búsqueda arrojó resultado de cero patrones encontrados.

En general podemos concluir que el algoritmo, de acuerdo a las reglas de orden establecidas para creación de las FC y recorrido de grafos, tiene un gran porcentaje de patrones reportados, dando un buen resultado. En su mayoría son los subgrafos más pequeños los que forman parte del porcentaje que no se reporta, es decir, los grafos maximales sí son reportados casi en su totalidad.

Se continuará la investigación para mejorar el algoritmo y minimizar las restricciones de los grafos de entrada. En posteriores trabajos también se incluirá al algoritmo un módulo de discretización y otro de eliminación de patrones repetidos por si llegasen a existir.

Una buena mejora como trabajo a futuro es poder realizar una medición de la distancia de similitud entre los grafos e incluso hacer *error correcting* (corrección de errores), para llevar el sistema algunos pasos adelante.

Otro modelo de cómputo sobre el que se puede trabajar en futuras investigaciones e implementaciones es el cómputo paralelo, ya que la tarea de detección de isomorfismos es altamente paralelizable.

Bibliografía

[1] Agrawal, R., and Srikant, R. Fast algorithms for mining association rules. In *Proceedings of the 1994 International Conference on Very Large Data Bases (VLDB'94)* (Santiago, Chile, 1994), pp. 487 – 499.

[2] Borgelt, C. Canonical forms for frequent graph mining. In *In Proceedings of the 30th Annual Conference of the Gesellschaft fr Klassifikation e.V.* (Freie Universitt, Berlin, 2006), p. 8 10.

[3] Borgelt, C., and Berthold, M. Mining molecular fragments: Finding relevant substructures of molecules. In *In Proceedings of the 2002 International Conference on Data Mining (ICDM02)* (Maebashi, Japan, 2002), p. 211 218.

[4] Gago, A., C. J., and Medina, J. Minera de subgrafos conexos frecuentes en colecciones de grafos etiquetados. Tech. rep., Coordinacin de Ciencias Computacionales, Instituto Nacional de Astrofsica, ptica y Electrnica, 2009.

[5] Giudici, R., B. A. *Introduccin a la Teora de Grafos*. EQUINOCCIO, 1997.

[6] Gori, M., M. M. S. L. Exact and approximate graph matching using random walks. In *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, vol. VOL. 27. July 2005, p. 1100 1111.

[7] Grimaldi, R. P. *Matemáticas Discreta y Combinatoria, una introducción con aplicaciones*, 3a edición ed. PEARSON, Prentice Hall, 1998.

[8] Haag, S. *Management Information Systems for the Information Age*. No. 28. McGraw-Hill Irwin, 2004.

[9] Han, J., C. H. X. D., and Yan, X. Frequent pattern mining: Current status and future directions. *Data Mining Knowledge Discovery (DMKD07), 10th Aniversary Issue Vol. 15*, Number 1 (2007), 55–86.

[10] Han, J., P. J. Y. Y. Mining frequent patterns without candidate generation. In *In Proceedings of the 2000 ACM SIGMOD International Conference on Management of data (SIGMOD00)* (Dallas, TX, 2000), p. 1 12.

[11] Holder, L. B., C. D. J., and Djoko, S. Substructure discovery in the subdue system. Tech. rep., Department of Computer Science and Engineering, University of Texas, Arlington, 1994.

[12] Holder, L. B., C. D. J., and Djoko, S. Knowledge discovery from structural data. *Journal of Intelligent Information Systems* (1995), 229 248.

[13] Hong, M., Z. H. W. W., and Shi, B. An efficient algorithm of frequent connected subgraph extraction. In *In Proceedings of the 7th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD03)* (2003), p. 40 51.

[14] Huan, J., W. W., and Prins, J. Efficient mining of frequent subgraph in the presence of isomorphism. In *In Proceedings of the 2003 International Conference on Data Mining (ICDM03)* (Melbourne, FL., 2003), p. 549 552.

[15] Huan, J., W. W. P. J., and Yang, J. Spin: Mining maximal frequent subgraphs from graph databases. In *In Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, 2004).

[16] Inokuchi, A. Mining generalized substructures from a set of labeled graphs. In *In proceedings of the 4th IEEE International Conference on Data Mining (ICDM04).* (Los Alamitos, CA., 2004), p. 415 418.

[17] Inokuchi, A., W. T., and Motoda, H. An apriori-based algorithm for mining frecuent substructures from graph data. In *Proceedings of the 2000 European Symposium on the Principle of Data Mining and Knowledge Discovery (PKDD00)* (Lyon, France, 2000), pp. 13 – 23.

[18] Inokuchi, A., W. T. N. K., and Motoda, H. A fast algorithm for mining frequent connected subgraphs. Tech. rep., IBM Research, Tokio Research Laboratory, 2002.

[19] Kuramochi, M., and Karypis, G. Frequent subgraph discovery. In *In Proceedings of the 2001 International Conference on Data Mining (ICDM01)* (San Jose, CA., 2001), pp. 313 – 320.

[20] Kuramochi, M., and Karypis, G. An efficient algoritmh for discovering frequent subgraphs. Tech. rep., Departament of Computing Science, University of Minnesota, 2002.

[21] Nijssen, S., and Kok, J. A quickstart in frequent structure mining can make a difference. In *In Proceedings of the 2004 ACM SIGKDD International Conference on Knowledge Discovery in Databases (KDD04)* (Seattle, WA, 2004), p. 647 652.

[22] Olmos, I., G. J., and Osorio, M. Subgraph isomorphism detection using a code based representation. In *In Proceedings of the 18th International FLAIRS Conference* (2005), p. 474 479.

[23] SUBDUE, G. B. K. D. `https://ailab.wsu.edu/subdue/`.

[24] Vanetik, N., G. E., and Shimony, S. E. Computing frequent graph patterns from semistructured data. In *In Proceedings of the 2002 International Conference on Data Mining (ICDM02)* (Maebashi, Japan, 2002), p. 435 458.

[25] Yan, X., H. J. gspan: Graph-based substructure pattern mining. In *In Proceedings of the 2002 International Conference on Data Mining (ICDM02)* (Maebashi, Japan, 2002), p. 721 724.