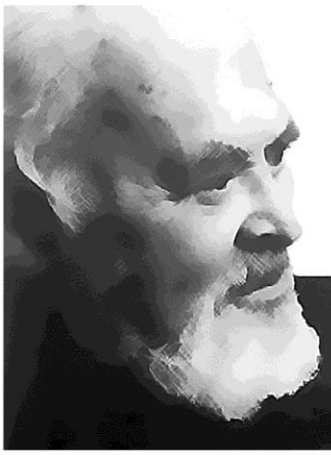
Santiago Bianco

Análisis comparativo de algoritmos de minería de subgrafos frecuentes

Trabajo Final Integrador presentado para la obtención del título de Licenciado en Sistemas del Departamento de Desarrollo Productivo y Tecnológico

Director del Trabajo Final Integrador
Ramón García Martínez y Sebastian Martins



ANALISIS COMPARATIVO DE ALGORITMOS DE MINERÍA DE SUBGRAFOS FRECUENTES

Alumno

APU Santiago BIANCO

Directores

Dr. Ramón GARCIA-MARTINEZ y Lic. Sebastián MARTINS

TRABAJO FINAL PRESENTADO PARA OBTENER EL GRADO
DE
LICENCIADO EN SISTEMAS

DEPARTAMENTO
DE DESARROLLO PRODUCTIVO Y TECNOLOGICO
UNIVERSIDAD NACIONAL DE LANUS

MARZO, 2016

RESUMEN

Gracias a las posibilidades que ofrecen a la hora de representar información y la abstracción conceptual que permiten manejar, los grafos son ampliamente utilizados en investigaciones relacionadas con la informática. A medida que se fueron incrementando las aplicaciones de estas estructuras, la complejidad de los elementos a representar y el volumen de información manejado, aparece la necesidad de utilizar procesos eficientes para extraer información o patrones ocultos en esa gran masa de datos, por lo que se comienza a aplicar la minería de grafos. Dentro las técnicas de minería de grafos se encuentra la búsqueda de subgrafos frecuentes, utilizada para reconocer subestructuras comunes entre un conjunto de grafos. En los últimos años se han llevado a cabo varias investigaciones que resuelven este problema, generando algoritmos diversos aplicando distintos enfoques, entre los cuales se encuentran el FSG, FFSM, gSpan y GASTON. El objetivo de este trabajo es analizar el comportamiento de estos algoritmos a través de distintos experimentos diseñados para identificar si existe un algoritmo superior al resto y, en caso de que no lo haya, poder definir en qué escenarios es más recomendable la elección de cada uno.

ABSTRACT

Given the possibilities offered when representing information and their abstraction level, graphs are widely used in computer science research. As these structures applications, the complexity of the elements to be represented and the volume of information handled increased, the need for efficient processes for extracting information or hidden patterns in that great mass of data appears, giving birth to the concept of graph mining. One of the graph mining techniques applied is the process of frequent subgraph mining (FSM), used to recognize repeated substructures among a set of graphs. In recent years several techniques and approaches have been developed in order to solve this problem, generating different algorithms such as the FSG, FFSM, gSpan and GASTON. The objective of this research paper is to analyse the behaviour of these algorithms through different experiments designed to identify whether there is an algorithm above the rest or, if there is not, define which algorithm is recommended for each scenario.

DEDICATORIA

A mis padres, Patricia y Carlos, por haberme apoyado y acompañado en todo momento.

A mis abuelos, que estuvieron siempre y sin quienes no sería la persona que soy hoy.

Particularmente a mi abuela Nelly (Tata), con quien afortunadamente sigo compartiendo charlas de mate y continúa ayudándome en el día a día.

A mis amigos y familia por su ayuda en este largo camino y a mi novia Virginia, por su paciencia y su amor incondicional.

AGRADECIMIENTOS

A la Universidad Nacional de Lanús por acogerme con generosidad de “alma máter” para que pudiera llevar a cabo mis estudios de Licenciatura en Sistemas.

A mis profesores de carrera quienes me proveyeron de estímulos, respuestas y dudas para permitirme crecer profesional y personalmente.

Al Dr. Ramón García-Martínez por el apoyo y la orientación que recibí durante toda mi carrera y sigo recibiendo, pero principalmente por la paciencia y por haber confiado en mí en todo momento.

Al Lic. Sebastián Martins sin cuya ayuda no hubiera podida terminar este trabajo y quien ha sido un gran profesor, colega y amigo en este camino.

A mis compañeros y amigos que me acompañaron en las cursadas y permitieron que este trayecto sea mucho más placentero.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

NOMENCLATURA

BFS	Breadth First Search
CAM	Canonical Adjacency Matrix
DFS	Depth First Search
DFSM	Minimum DFS Code
FFSM	Algoritmo Fast Frequent Subgraph Mining
FSG	Algortimo Fast SubGraph mining
FSM	Frequent Subgraph Mining
GASTON	Algortimo GrAph/Sequence/Tree extractiON
gSpan	Algoritmo graph-based Subestructure pattern mining
MST	Minimum Spanning Tree
SOM	Self-Organizing Map
TDIDT	Top-Down Induction Decision Tree
TFL	Trabajo Final de Licenciatura

1. INTRODUCCIÓN

En este capítulo se proporciona una introducción general al presente documento. Se presenta el marco del Trabajo Final de Licenciatura (Sección 1.1), se establece la delimitación del problema tratado (Sección 1.2), se detallan los objetivos de la investigación (Sección 1.3) se describen brevemente la solución propuesta (Sección 1.4), se describe la metodología seguida (Sección 1.5) y se da una visión general de la estructura del trabajo (Sección 1.6).

1.1. MARCO DEL TRABAJO FINAL DE LICENCIATURA

Los grafos son estructuras que están compuestas por dos elementos, vértices o nodos, los cuales están relacionados mediante aristas o arcos que pueden ser dirigidos o no. Es un concepto simple pero de gran potencial que puede ser utilizado para modelar diversos elementos de muy variada complejidad y cuyas aplicaciones abarcan distintas disciplinas y áreas del conocimiento [Ferro et al., 2007; Przulj et al., 2006; Ohlrich et al., 1993].

Gracias a las posibilidades que ofrecen a la hora de representar información y la abstracción conceptual que permiten manejar, son ampliamente utilizados en investigaciones relacionadas con la informática. A medida que se fueron incrementando las aplicaciones de estas estructuras, la complejidad de los elementos a representar y el volumen de información manejado, aparece la necesidad de utilizar procesos eficientes para extraer información o patrones ocultos en esa gran masa de datos, por lo que se comienzan a aplicar técnicas de explotación de información.

La explotación de información es la sub-disciplina de los sistemas de información que aporta a la inteligencia de negocio las herramientas para la transformación de información en conocimiento [García-Martínez et al., 2011]. Se define como la búsqueda de patrones interesantes y de regularidades importantes en grandes masas de información.

La explotación de información en grafos es también conocida como minería de grafos y puede constar de varios procesos, dependiendo de la consulta que se haga o de la base de datos con la que se cuente. Las bases de datos pueden ser conjuntos de grafos de pequeño o mediano tamaño o un único grafo conexo de gran tamaño. En base a esto, las consultas más realizadas se consisten en buscar subgrafos similares a una estructura dada o, a partir de una frecuencia de ocurrencia determinada, buscar todos los subgrafos que se repitan con igual o mayor frecuencia entre todos los grafos de la base o dentro de un grafo de gran tamaño.

En los últimos años se han llevado a cabo varias investigaciones que resuelven el problema de la minería de grafos, generando algoritmos diversos para resolver el mismo problema aplicando distintos enfoques.

1.2. DELIMITACIÓN DEL PROBLEMA

El presente trabajo se enfoca en el problema de búsqueda de subgrafos frecuentes en una base de datos compuesta por un conjunto de grafos, considerando tanto la estructura topológica de los mismos como las etiquetas que contienen los arcos y los vértices. Se han desarrollado varios algoritmos para resolver este problema por lo que es de interés para dar soluciones más eficientes a los problemas compararlos para determinar su comportamiento en distintos escenarios de manera que se pueda elegir la mejor opción en base a los datos con los que se cuente. En esta investigación se evaluarán los algoritmos FFSM, FSG, gSpan y GASTON, los cuales buscan estructuras considerando una frecuencia de ocurrencia mínima determinada y como resultado generan una lista con las subestructuras que cumplan con esa condición. Los mismos fueron elegidos ya que cada uno presenta novedosos enfoques y son los más reconocidos en el estado del arte.

1.3. OBJETIVOS DE LA INVESTIGACIÓN

A continuación se presentan los objetivos del trabajo final de licenciatura divididos en Objetivos Generales (Sección 1.3.1) y Objetivos Específicos (Sección 1.3.2).

1.3.1. OBJETIVOS GENERALES

El objetivo global de la presente investigación es analizar el comportamiento de distintos algoritmos de minería de grafos, particularmente de búsqueda de subgrafos frecuentes, en diversos escenarios para determinar la eficiencia de los mismos dependiendo la base de datos y los requerimientos con los que se cuente. De esta manera se busca determinar cuál sería la mejor elección para una situación específica.

Para esto utilizan implementaciones de los algoritmos previamente mencionados y se los somete a varias pruebas cambiando la densidad de la base de datos utilizada, midiendo la cantidad de subestructuras encontradas y el tiempo de ejecución de cada implementación en cada experimento.

1.3.2. OBJETIVOS ESPECÍFICOS

Los objetivos específicos desprendidos de los objetivos generales expuestos anteriormente son los siguientes:

1. Investigar y analizar los distintos algoritmos de búsqueda de subgrafos frecuentes desarrollados hasta el momento.
2. Desarrollar distintos casos de pruebas y experimentaciones cubriendo un amplio rango de escenarios posibles en los que se puedan ejecutar los algoritmos.

3. Desarrollar una capa de software que permita ejecutar distintas implementaciones de algoritmos sobre las mismas bases de datos, permitiendo extraer y comparar resultados.
4. Determinar bajo qué circunstancias es más recomendable la elección de cada algoritmo, o definir si es que existe algún algoritmo superior al resto en cualquier escenario.

1.4. SOLUCIÓN PROPUESTA

La solución propuesta incluye el desarrollo de un conjunto de pruebas que permitan determinar el comportamiento de los distintos algoritmos en variados escenarios, así como también la construcción de un banco de pruebas que posibilite la ejecución de las distintas implementaciones de los algoritmos utilizados para llevar a cabo los experimentos antes mencionados y extraer los resultados.

1.5. METODOLOGÍA DE DESARROLLO

La metodología de desarrollo que sirve de guía para el presente trabajo se planificó de acuerdo con las etapas descritas en las siguientes subsecciones.

1.5.1. ETAPA I

La primera parte consta de una etapa de investigación previa al desarrollo, en la cual se busca información de distintos autores con el objetivo de formar una base sólida acerca de la teoría de grafos, la explotación de información, el proceso de minería de grafos y los distintos algoritmos desarrollados de interés para el trabajo.

1.5.2. ETAPA II

En esta etapa se lleva a cabo la búsqueda de las implementaciones de los algoritmos a utilizar y se desarrolla el software que servirá para generar las bases de datos de prueba, ejecutar los algoritmos y extraer resultados.

1.5.3. ETAPA III

En esta parte de la investigación se diseñan los experimentos en los cuales se ejecutan los algoritmos, teniendo en cuenta características de las bases de datos y resultados esperados

1.5.4. ETAPA IV

En la etapa final se ejecutan los experimentos y posteriormente se analizan y comparan los resultados de las pruebas realizadas mediante el procesamiento de los datos obtenidos por el software de banco de pruebas. De estos análisis de desprenden las conclusiones del presente trabajo.

1.6. VISIÓN GENERAL DEL TFL

En el capítulo Introducción, se presenta el marco del Trabajo Final de Licenciatura, se da una delimitación del problema, se plantean los elementos de la solución propuesta, y se da una visión general del proyecto.

En el capítulo Estado de la Cuestión, se presentan los marcos teóricos correspondientes a la teoría de grafos, explotación de información, minería de grafos y se da una breve explicación del funcionamiento de los algoritmos utilizados para el análisis.

En el capítulo Descripción del problema, se identifica el problema de investigación, en el cual se describen las aplicaciones de los grafos y la necesidad de identificar los algoritmos más eficientes e acuerdo a los datos y los requerimientos que se tengan. Luego se define el problema abierto y se concluye con un sumario de investigación.

En el capítulo Solución, se presentan las características de las pruebas a realizar y se explica cómo es llevado a cabo el desarrollo del banco de pruebas que sirve para ejecutar los algoritmos y extraer los resultados. Se introducen todos los experimentos desarrollados, las bases de datos utilizadas, las variables dependientes e independientes de cada escenario y las implementaciones de los algoritmos utilizados.

En el capítulo Experimentación se muestra el desarrollo de los experimentos así como también los resultados obtenidos y un breve análisis de los mismos.

En el Capítulo Conclusiones, se presentan las aportaciones de este Trabajo Final de Licenciatura y se destacan las futuras líneas de investigación que se consideran de interés en base al problema abierto que se presenta en este trabajo.

En el Capítulo Referencias se listan todas las publicaciones consultadas para el desarrollo de esta investigación.

En el Apéndice A, se presentan los diagramas correspondientes al banco de pruebas desarrollado y en código del mismo.

2. ESTADO DE LA CUESTIÓN

En esta sección se presentan los conceptos, teorías y algoritmos de interés para el desarrollo del presente trabajo. Se comienza con una breve introducción al concepto de grafos (Sección 2.1), luego se dan definiciones y se describen otros conceptos de interés (Sección 2.2) los cuales son la teoría de grafos (Sección 2.2.1), formas de representación para los mismos (Sección 2.2.2) y cómo recorrerlos en anchura (Sección 2.2.3.1) y profundidad (Sección 2.2.3.1). Se continúa con una presentación del concepto de explotación de información (Sección 2.3) para luego explicar los procesos de explotación de información en grafos (Sección 2.4), la tarea de búsqueda de subgrafos frecuentes (Sección 2.4.1) y una descripción de los algoritmos de interés para la investigación: FSG (Sección 2.4.1.1), gSpan (Sección 2.4.1.2), FFSM (Sección 2.4.1.3) y GASTON (Sección 2.4.1.4).

2.1. INTRODUCCIÓN A LOS GRAFOS

Un grafo es un concepto matemático que permite representar situaciones en las que distintos elementos se comunican o se relacionan entre sí de diversas maneras. Puede definirse informalmente como un conjunto de nodos o vértices relacionados por otro conjunto de arcos o aristas. Los nodos se utilizan para representar los objetos que interactúan y los arcos indican el tipo de relación que los une. Por ejemplo, las ciudades en un mapa pueden caracterizarse con vértices y los caminos que conducen a ellas con aristas, como puede se observa en la figura 2.1.

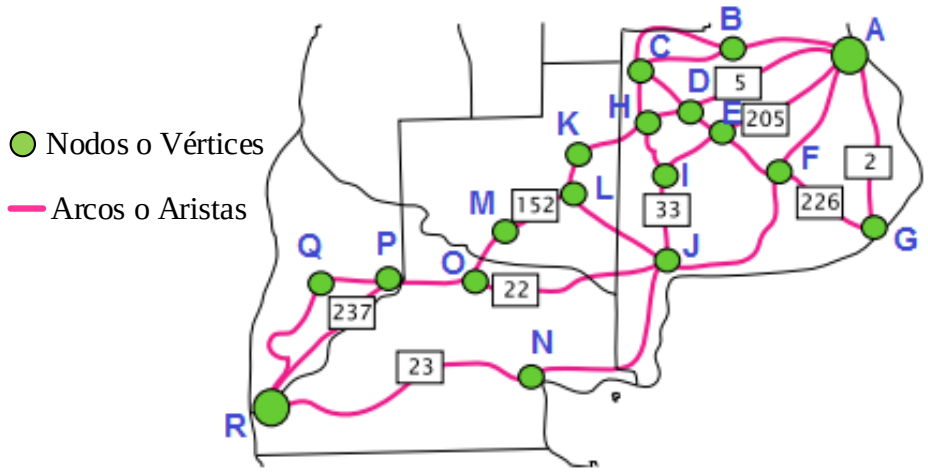


Figura 2.1. *Ejemplo de un mapa representado mediante el uso de grafos [Ramirez et al, 2016].*

Existen distintos tipos de grafos que varían según el tipo de relaciones que puede tener. En el ejemplo de las ciudades, los caminos que las unen pueden ser recorridos en ambas direcciones. En otras palabras, es lo mismo decir que existe un camino que va desde la ciudad A hacia la ciudad B, a decir que existe un camino que va desde la ciudad B hacia la ciudad A. En este caso, el grafo que re-

presenta al mapa es un grafo no dirigido. Si las rutas tuvieran un sentido único, estaríamos hablando de un grafo dirigido. Ejemplos de ambos tipos de grafos pueden verse en la figura 2.2.
Estos conceptos y otras variantes de grafos se explican con más detalle en la Sección 2.2.1.

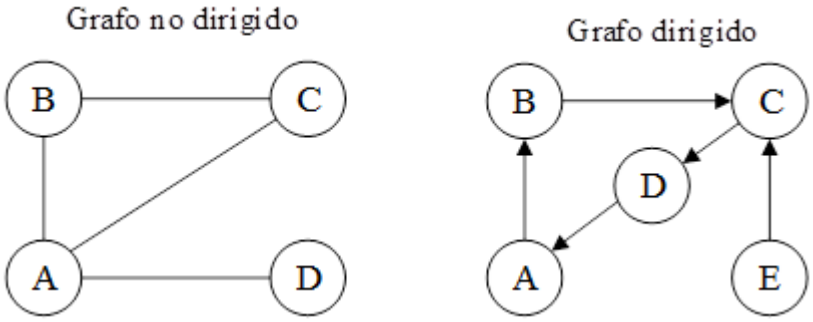


Figura 2.2. *Ejemplos de un grafo dirigido y un grafo no dirigido*

Prácticamente, cualquier tipo de red puede ser representado mediante grafos, lo que les da un carácter heterogéneo e interdisciplinario. En la figuras 2.3.a, 2.3.b, 2.3.c y 2.3.d se muestran algunos tipos de elementos que pueden ser modelados mediante el uso de grafos. Debido a esta versatilidad es que se han llevado al campo de la informática, permitiendo tener una herramienta sumamente útil y de creciente interés en los últimos años, principalmente con la expansión de internet, las redes de comunicación y las redes sociales.

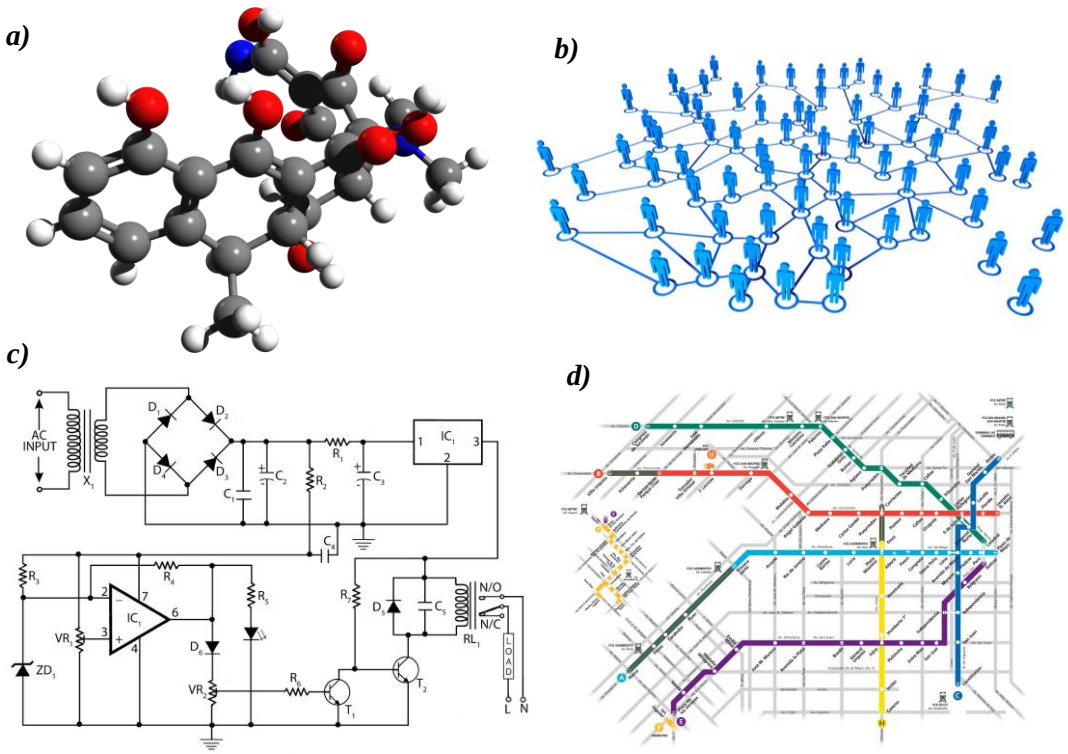


Figura 2.3. *Ejemplos de elementos que pueden ser modelados con grafos. a) Estructura química. b) Red Social. c) Diagrama Eléctrico. d) Conexión de estaciones de tren subterráneo*

2.2. CONCEPTOS Y DEFINICIONES DE INTERÉS

En esta sección se presentan algunos conceptos de interés para este trabajo de investigación. Se dan definiciones formales correspondientes a la teoría de grafos (Sección 2.2.1), se describen formas de representación (Sección 2.2.2) y se explican las principales maneras de recorrerlos: en anchura (Sección 2.2.3.1) y en profundidad (Sección 2.2.3.2).

2.2.1. TEORÍA DE GRAFOS

A continuación se amplían los conceptos anteriormente explicados correspondientes a la teoría de grafos y se agregan definiciones de elementos que serán de utilidad en la investigación.

Grafo (no dirigido): un grafo es un par ordenado $G = (V, E)$, en el cual V es un conjunto de vértices o nodos y E es un conjunto de aristas o arcos. Cada arista $e \in E$ está compuesta por un par no ordenado de vértices u y v tal que $u, v \in V$. Retomando el ejemplo de la Figura 2.2, puede observarse que es la misma arista puede representarse como $e = (A, B)$ y como $e = (B, A)$. La cantidad de vértices de un grafo se denota como $|V|$ y la cantidad de arcos como $|E|$

Grafo dirigido: un grafo dirigido es un par ordenado $G = (V, E)$, en el cual V es un conjunto de vértices o nodos y E es un conjunto de aristas o arcos, en el cual cada arista $e \in E$ está compuesta por un par ordenado de vértices u y v tal que $u, v \in V$. Una arista $e = (A, B)$ representa un arco que se dirige desde el nodo A hacia el nodo B, por lo que difiere de la arista $e = (B, A)$.

Adyacencia: Dado un grafo $G = (V, E)$, dos nodos $u, v \in V$ son adyacentes si existe un arco $e = (u, v)$ con $e \in E$. De manera informal, dos vértices son adyacentes si están unidos por una arista. En un grafo no dirigido, la relación de adyacencia es simétrica, por lo que para una arista (u, v) los vértices u y v son adyacentes entre sí. Esto no ocurre con los grafos dirigidos, ya que para una arista (a, b), el nodo a es adyacente al b, pero el b no es adyacente al a.

Incidencia: Dada una arista (u, v) en un grafo no dirigido, se dice que esta arista incide en los vértices u y v, los cuales son denominados vértices extremos. En un grafo dirigido, se dice que el arco (u, v) incide desde el vértice u, el cual se denomina vértice inicial, hacia el vértice v, llamado vértice final.

Ponderación: Un grafo $G = (V, E)$ es un grafo ponderado si cada arista $e \in E$ tiene asociado un número específico el cual es llamado coste, valor o ponderación. Este tipo de grafos es útil, por

ejemplo, cuando se debe buscar el camino más corto o más rápido para recorrer *n* ciudades conectadas por varias rutas. Las rutas representadas por aristas serán asociadas a un número que puede indicar su longitud o el tiempo que lleva recorrerlas.

En un grafo ponderado se llama peso de un camino a la suma de los pesos de las aristas que lo forman. El camino más corto entre dos vértices dados es el camino con menor peso entre dichos vértices. En contraposición, el camino más largo o mejor conocido como camino crítico es aquel con el peso máximo.

Etiquetado: Generalmente dependiendo de la nomenclatura con la que se trabaje, un grafo etiquetado es sinónimo de un grafo ponderado. Para el desarrollo de la investigación se hará una distinción considerando que un grafo etiquetado, además de tener pesos en las aristas, puede contar cualquier tipo de marca tanto en los nodos como en los arcos, ya sean valores numéricos como nombres.

Subgrafo: Dado un grafo $G = (V, E)$, $G' = (V', E')$ es un subgrafo de G si cumple con la siguientes condiciones:

1. $V' \subseteq V$ y $E' \subseteq E$.
2. Para toda arista $e' \in E'$, si e' incide en v' y w', entonces $v', w' \in V'$

En la Figura 2.4 se clarifica esta definición formal con un ejemplo gráfico se un grafo y un posible subgrafo del mismo. Si bien en el ejemplo se muestra un grafo no dirigido, este concepto se aplica también a los grafos dirigidos.

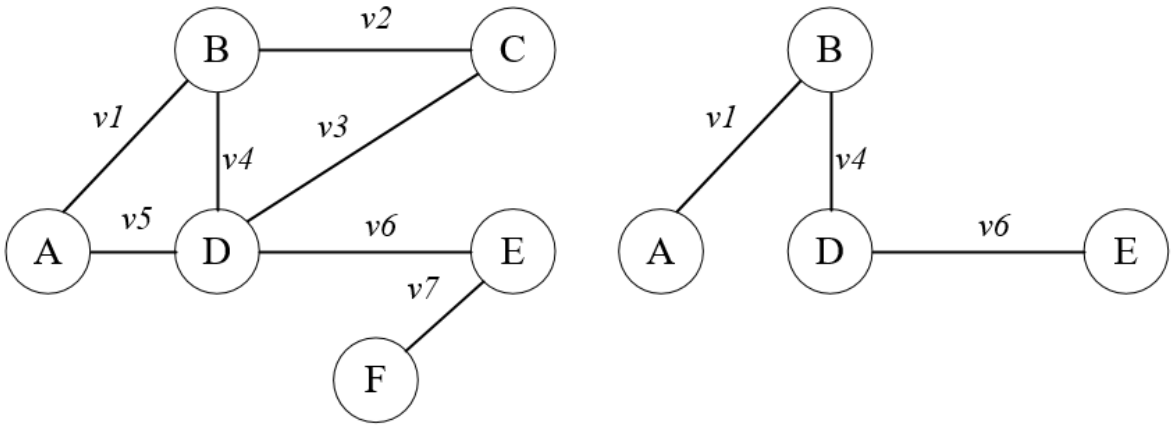


Figura 2.4. *Ejemplo de un grafo no dirigido y uno de los posibles subgrafos que contiene*

Árbol: un árbol es un grafo no dirigido, conexo y acíclico. Esto quiere decir que cualquier par de vértices del grafo está conectado por un único camino simple. Si se saca cualquier arco de un árbol, éste deja de ser conexo. Un grafo no dirigido, acíclico y no conexo es denominado como bosque. En la figura 2.5.a-c puede observarse un ejemplo de estos conceptos.

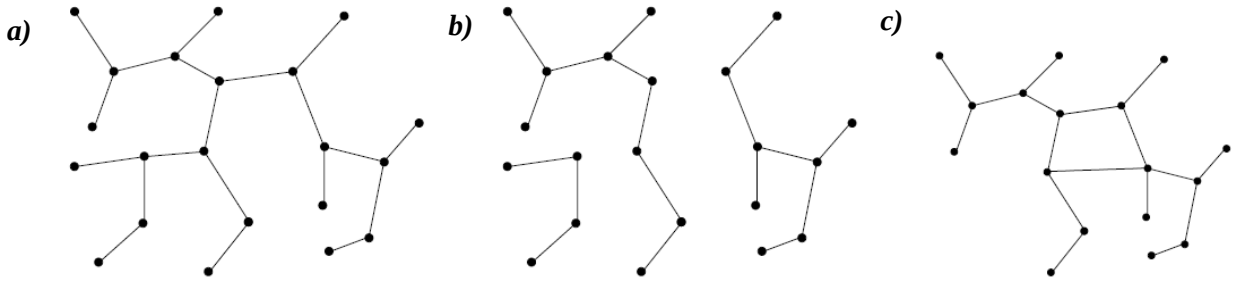


Figura 2.5. *Ejemplo de: a) Un árbol b) Un bosque c) Un grafo que no es ni árbol de bosque*

Árbol en expansión: informalmente un árbol de expansión T de un grafo G es un subgrafo del mismo que contiene todos sus vértices. Este tipo de árboles también es conocido como árbol abarcador, en inglés *Spanning Tree*. El árbol abarcado mínimo (o MST) de un grafo ponderado es aquel que cubre todos los vértices del grafo teniendo el menor peso posible en sus aristas. En la Figura 2.6 se muestra un árbol en expansión de un grafo.

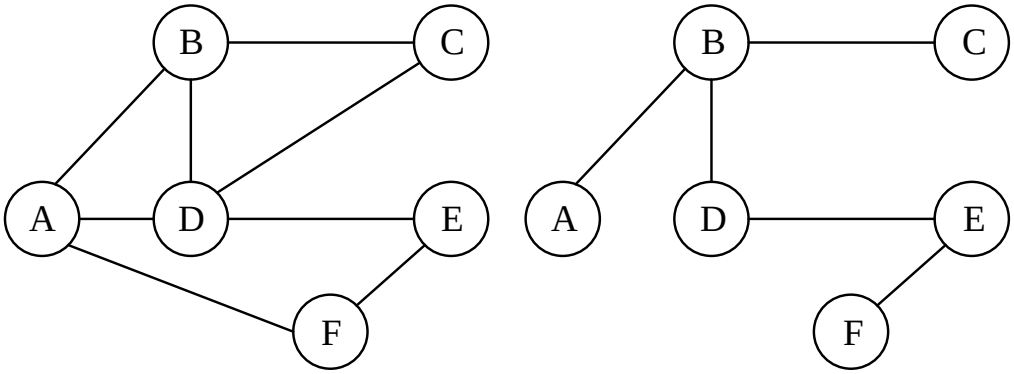


Figura 2.6. *Ejemplo de un árbol en expansión de un grafo*

Isomorfismo: Dos grafos G_1 y G_2 son isomorfos si existe una función *f* uno a uno sobre los vértices de G_1 a los vértices de G_2 y una función g uno a uno sobre las aristas de G_1 sobre las aristas de G_2, de manera que una arista e es incidente en *v* y *w* en G_1 si y sólo si la arista *g(e)* es incidente en *f(v)* y *f(w)* en G_2. Esto quiere decir que la relación de isomorfismo, expresada como *G1~G2*, existe si ambos grafos son topológicamente idénticos, tal como se ve en la Figura 2.7. En este ejemplo, el isomorfismo se define por *f(A)=1, f(B)=2, f(C)=3, f(D)=4, f(E)=5, g(x1)=y1, g(x2)=y2, g(x3)=y3, g(x4)=y4, g(x5)=y5.*

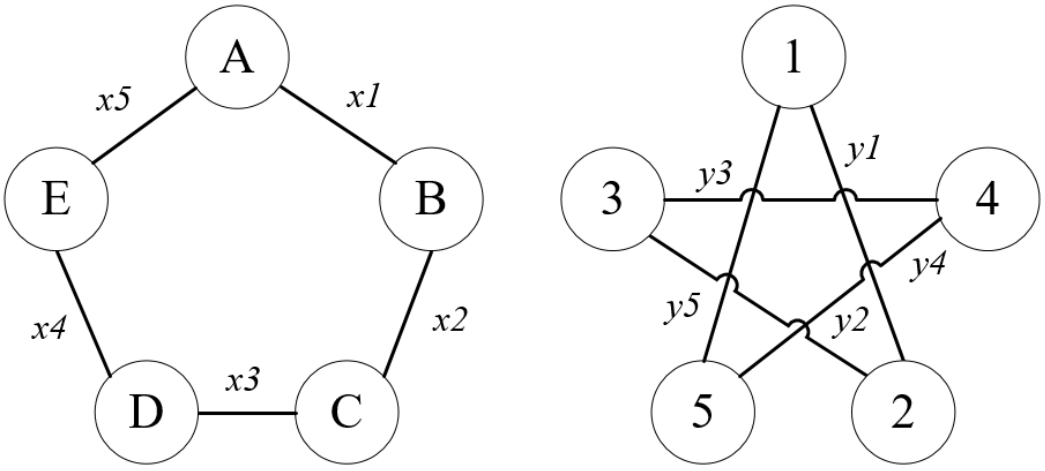


Figura 2.7. *Ejemplo de dos grafos isomorfos.*

Sub-Isomorfismo: La relación de sub-isomorfismo entre un grafo G_1 y un grafo G_2 de menor o igual tamaño existe si el grafo G_2 es isomorfo con un subgrafo de G_1. En otras palabras, el problema de búsqueda de sub-isomorfismos consiste en determinar si G_2 es parte de G_1.

2.2.2. FORMAS DE REPRESENTACIÓN

Para poder implementar el concepto de los grafos en la informática, los mismos deben poder representarse con estructuras de datos que se puedan crear, modificar y consultar. A continuación se explicarán las formas de representación más usuales: matriz de adyacencia (Sección 2.2.2.1), matriz de incidencia (Sección 2.2.2.2) y lista de adyacencia (Sección 2.2.2.3).

2.2.2.1. MATRIZ DE ADYACENCIA

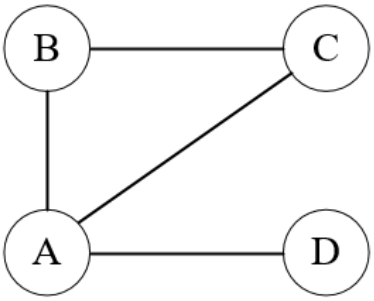
Una matriz de adyacencia es una matriz booleana M con tamaño igual al cuadrado de la cantidad de nodos del grafo $G = (V, E)$ a representar, que contiene un *1* en la posición Mij si se cumple que $(v_i, v_j) \in E$ o un 0 en caso contrario. Suponiendo que $|V| = n$ y el conjunto de vértices es $V = \{a_1, a_2, \ldots, a_n\}$, los elementos de la matriz de adyacencia M quedan definidos por la fórmula:

$$M_{ij} = \begin{cases} 1 \text{ si } (a_i, a_j) \in E \\ 0 \text{ en caso contrario} \end{cases}$$

Si se representa un grafo no dirigido, la matriz siempre será simétrica como se observa en la figura 2.8, lo que no ocurre en el caso de los grafos dirigidos tal como se ve en la figura 2.9.

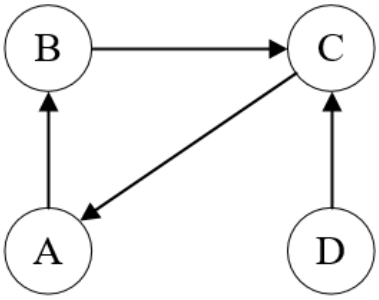
La ventaja de este tipo de representación es que es fácil de implementar y su tamaño depende directamente del número de nodos independientemente de la cantidad de arcos, por lo que es útil para representar grafos densos, en los cuales se cumple que $|E|$ y $|V|^2$ son comparables. En el resto de los

casos en donde generalmente $|E| << |V|^2$, no es conveniente usar este tipo de representación debido a que se desperdicia una gran cantidad de memoria.



	A	**B**	**C**	**D**
A	0	1	1	1
B	1	0	1	0
C	1	1	0	0
D	1	0	0	0

Figura 2.8. *Grafo no dirigido con su representación como matriz de adyacencia.*



	A	**B**	**C**	**D**
A	0	1	0	0
B	0	0	1	0
C	1	0	0	0
D	0	0	1	0

Figura 2.9. *Grafo dirigido con su representación como matriz de adyacencia.*

En cuanto a velocidad de acceso, las matrices de adyacencia ofrecen gran velocidad para hacer consultas acerca de los arcos del grafo, así como también para agregar o quitar aristas. No es así en el caso de que se quiera agregar o quitar vértices ya que se debe redimensionar la matriz, agregando o quitando columnas y filas en cada operación.

Existen variantes de esta representación representar otros tipos de grafos. Por ejemplo, para los grafos ponderados, en lugar de completar las posiciones de las matrices con 1's, se completa con el valor o peso del arco al que hace referencia.

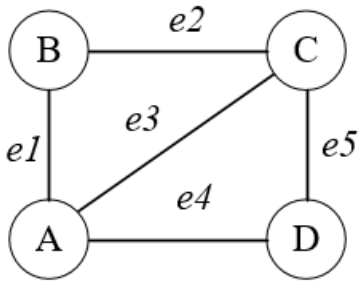
2.2.2.2. MATRIZ DE INCIDENCIA

La matriz de incidencia de un grafo *G = (V, E),* donde $|V| = n$ y $|E| = m$ se define como una matriz booleana M con dimensiones de n x m en la que

$$M_{ij} = \begin{cases} 1 \text{ si la arista } j \text{ incide en el vértice } i \\ 0 \text{ en caso contrario} \end{cases}$$

Obsérvese que usando este tipo de matriz no puede representarse el sentido de las aristas, por lo que es solamente válida para grafos no dirigidos.

En la Figura 2.10 puede observarse un ejemplo de esto. Las filas de la matriz representan a los vértices y las columnas a las aristas.



	e1	***e2***	***e3***	***e4***	***e5***
A	1	0	1	1	0
B	1	1	0	0	0
C	0	1	1	0	1
D	0	0	0	1	1

Figura 2.10. *Grafo dirigido con su representación como matriz de incidencia.*

2.2.2.3. LISTA DE ADYACENCIA

La representación con listas de adyacencia de un grafo $G = (V, E)$ cuyo conjunto de vértices es $V = \{a_1, a_2, ..., a_n\}$, siendo $|V| = n$, se genera mediante un vector de listas de $|V|$ posiciones. En la posición *i-ésima* se almacenará la lista de vértices adyacentes a a_i. Esta estructura queda ejemplificada mediante la Figura 2.11.

A diferencia de lo que ocurre con las matrices de adyacencia, esta representación no tiene los problemas exceso de memoria para los grafos dispersos, lo que lo hace la opción más eficiente en cuanto a almacenamiento para casi todos los casos, exceptuando aquellos en donde $|E|$ y $|V|^2$ son similares. En el caso de la velocidad para realizar operaciones, permite agregar vértices y aristas fácilmente pero no removerlos, ya que para esto último se debe recorrer toda la estructura.

Este tipo de implementación permite representar tanto grafos dirigidos como no dirigidos, así como también agregar etiquetas y cualquier tipo de información correspondiente al grafo en los nodos de las listas.

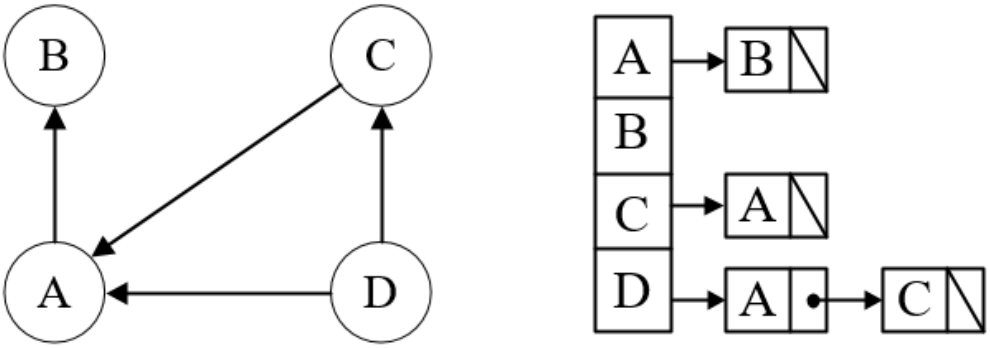


Figura 2.11. *Grafo dirigido con su representación como lista de adyacencia.*

2.2.2.4. REPRESENTACIONES CANÓNICAS

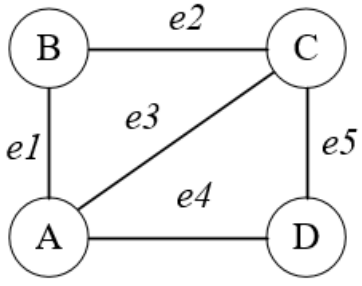
Para facilitar la detección de isomorfismos, se utilizan métodos de representación canónicos, que evitan el problema que pueden llegar a tener las matrices: los grafos pueden ser representados en muchas maneras distintas dependiendo de cómo se enumeren los arcos y los nodos. Para un determinado grafo existe un único código. Si bien existen muchas variantes, los más interesantes para esta investigación son: *Minimum DFS Code (o DFSM)* y *Canonical Adjacency Matrix (o CAM)*.

Un código DFS de un grafo *G = (V, E)* se construye a partir de un árbol abarcador *T* generado utilizando búsqueda en profundidad (Sección 2.2.3.2) y se denota como *code(G, T)*. Cada arco del código se representa mediante una tupla con la siguiente forma: $(i, j, l_i, l_{(i,j)}, l_j)$, donde *i, j* son vértices de G, l_i y l_j son las etiquetas de esos nodos y $l_{(i,j)}$ es la etiqueta del arco que los une. Por ejemplo *(0, 1, A, e1, B)* es el código de la arista e1 del grafo de la Figura 2.12. Dependiendo de cómo se haga el recorrido, un grafo puede tener varios códigos DFS, por lo que se usa en DFSM.

El código DFS mínimo de un grafo *G*, denotado como *min(G)*, es el mínimo código DFS de un grafo en orden lexicográfico. Si se tienen dos grafos *G* y *G'*, estos son isomorfos si y solo si *min(G) = min(G')* [Yan et al., 2002].

Para la utilización de la forma canónica CAM [Huan et al., 2003], primero se debe representar al grafo con una matriz de adyacencia con la siguiente diferencia: en la diagonal principal se pondrán las etiquetas de los nodos y en el resto de las posiciones de la matriz las etiquetas de los arcos correspondientes. En el caso de que no exista una arista, se colocará un 0 en esa posición, tal como se muestra en la Figura 2.12. Para la generación del código, se concatena el triángulo superior o inferior de la matriz generada, incluyendo los valores de las diagonales. En el ejemplo antes mencionado, utilizando el triángulo inferior el código generado sería: {A*e1*B*e3**e2*C*e4*0*e5*D}.

La matriz se puede formar de distintas maneras por lo tanto para una única forma canónica se implementa el código lexicográficamente menor, tal como ocurre con DFSM.



	0	***1***	***2***	***3***
0	A			
1	*e1*	B		
2	*e3*	*e2*	C	
3	*e4*	0	*e5*	D

Figura 2.12. *Grafo con su representación como matriz de adyacencia CAM.*

2.2.3. RECORRIDO DE GRAFOS

Un recorrido de un grafo es un procedimiento que origina una enumeración ordenada de sus vértices y como resultado se genera un árbol abarcador del mismo. Los algoritmos clásicos de la teoría de grafos se basan en dos procedimientos de recorrido denominados recorrido en anchura (Sección 2.2.3.1) y recorrido en profundidad (Sección 2.2.3.2). A continuación se introducen las nociones básicas de los mismos.

2.2.3.1. RECORRIDO EN ANCHURA

La idea principal del recorrido o búsqueda en anchura, más conocido como BFS (*Breadth-First Search*), es la de procesar todos los vértices en un determinado nivel de profundidad antes de moverse al nivel más alto que le sigue. Este tipo de recorridos puede utilizarse para probar si un grafo G con $|V| = n$ es conexo. Si el árbol en expansión T producido tiene n vértices entonces G es conexo. Es útil también para encontrar el MST en un grafo ponderado desde un vértice fijo v a todos los demás vértices.

El proceso consta de varios pasos. Primero se selecciona un orden para los vértices de G. Se elige al primer vértice como raíz y se procede con la búsqueda. Para cada vértice del mismo nivel se buscan aquellos en los cuales inciden. Cuando no encuentra más, pasa al siguiente nivel hasta que no queden vértices por recorrer. Para mayor claridad, en la Figura 2.13 se describe el pseudocódigo de un algoritmo que realiza la búsqueda en anchura en un grafo. Posteriormente, en la Figura 2.14 se muestra un ejemplo de un grafo con el árbol en expansión generado luego de la implementación del algoritmo antes mencionado.

Función: BFS

Entrada: Grafo G con vértices ordenados

Salida: Árbol en expansión T=(V', E')

```
1:  S=(v1) // S es una lista ordenada y v1 es la raíz del árbol abarcador
2:  V' = {v1}
3:  E'= ∅
4:  REPETIR
5:      PARA CADA x ∈ S
6:          PARA CADA y ∈ V-V'
7:              SI (x, y) es una arista
8:                  agregar (x, y) a E'
9:                  agregar y a V'
10:     SI no se agregaron aristas
11:         RETURN T    // con V' y E'
12:     S = hijos de S ordenados siguiendo el orden original
```

Figura 2.13. *Algoritmo de búsqueda en anchura*

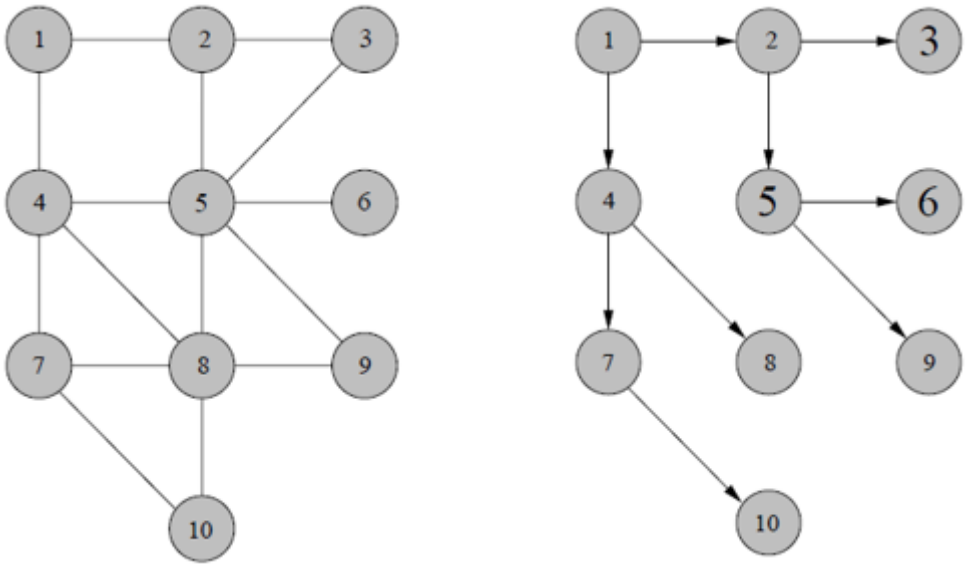


Figura 2.14. *Ejemplo de un grafo con su árbol abarcador encontrado a partir de la búsqueda en anchura.*

2.2.3.2. RECORRIDO EN PROFUNDIDAD

Una alternativa al recorrido BFS es la búsqueda o recorrido en profundidad, también conocido como DFS (*Depth-first search)*. La idea es, partiendo de un nodo raíz, acceder al siguiente nivel de profundidad lo más rápido posible. Cuando no se puede continuar, se retrocede uno o varios niveles de profundidad hasta que se encuentre algún camino sin explorar. En caso de que no exista, el procedimiento termina. El proceso de retornar al nivel anterior es conocido como backtracking. El backtracking es útil para resolver problemas de permutaciones de elementos, para buscar ciclos de Hamilton en grafos y para determinar si dos grafos son isomorfos.

La implementación de este tipo de recorridos puede hacerse de manera recursiva como se observa en la Figura 2.15. Este algoritmo ejemplifica la manera más sencilla de visitar todos los nodos de un grafo en profundidad. En la Figura 2.16 se muestra un árbol abarcador de un grafo generado a partir de la ejecución de DFS. En línea de trazos de marca el momento en que no se encontraron caminos y se tuvo que retroceder al nivel anterior.

Función: DFS-recursivo
Entrada: Grafo G=(V, E), vértice *v* inicial
Salida: Todos los nodos de G visitados
1: VISITAR(*v*) 2: **PARA CADA** *w* / (*v*, *w*) ∈ E 3: DFS(*G*, *w*) 4: **RETURN**

Figura 2.15. *Algoritmo de búsqueda en profundidad recursivo*

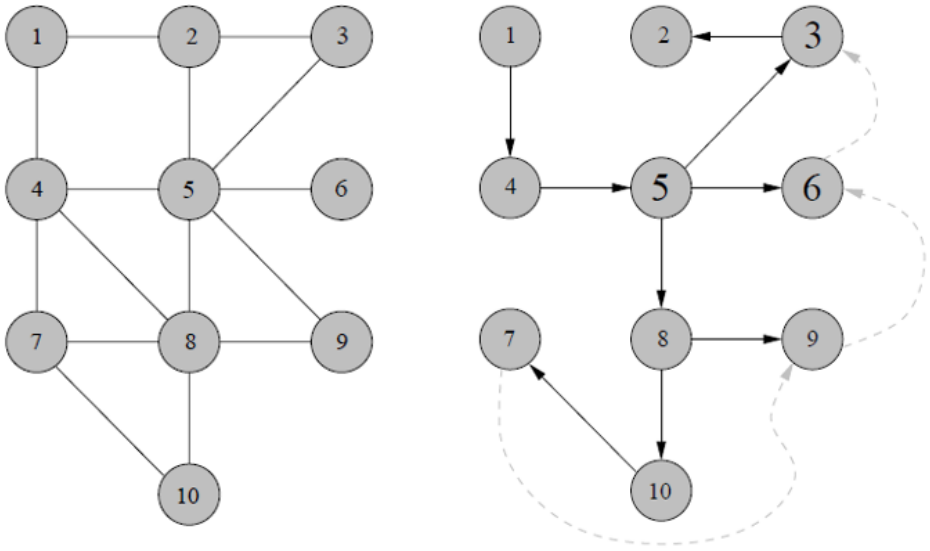


Figura 2.16. *Ejemplo de un grafo con su árbol abarcador encontrado a partir de la búsqueda en profundidad.*

2.3. INTRODUCCIÓN A LA EXPLOTACIÓN DE INFORMACIÓN

La explotación de información, también conocida como minería de datos, se define como la búsqueda de patrones interesantes y de reglas importantes en grandes masas de información [Grossman et al., 1999]. Este proceso se nutre de técnicas matemáticas y estadísticas para extraer información relevante y nuevos conocimientos de repositorios de bases de datos, de manera que se puedan descubrir patrones, tendencias o correlaciones entre los datos que a simple vista no pueden ser reconocidos.

Los procesos de explotación de información se valen de algoritmos de minería de datos y de la aplicación de métodos de sistemas inteligentes para obtener resultados [Britos y García-Martínez, 2009], tales como los árboles de clasificación (también conocidos como Top Down Induction Decision Trees, o TDIDT), algoritmos de clustering, redes neuronales y redes bayesianas. Dependiendo el proceso que se quiera realizar, estos métodos permiten identificar reglas que definan comportamientos (TDIDT), realizar particiones de grandes masas de información (Clustering), identificar factores influyentes en determinado resultado (redes de bayes), entre otras aplicaciones.

A continuación se caracterizan los cinco procesos de explotación de información descritos en [Britos y García-Martínez, 2009] y se identifican los algoritmos de sistemas inteligentes que se pueden aplicar: descubrimiento de reglas de comportamiento, descubrimiento de grupos, descubrimiento de atributos significativos, descubrimiento de reglas de pertenencia a grupos y ponderación de reglas de comportamiento o de pertenencia a grupos.

Descubrimiento de Reglas de Comportamiento [Britos y García-Martínez, 2009]**:** este proceso aplica cuando se requiere identificar cuáles son las condiciones para obtener determinado resultado en el dominio del problema. Son ejemplos de problemas que requieren este proceso: identificación de características del local más visitado por los clientes, identificación de factores que inciden en el alza las ventas de un producto dado, establecimiento de características o rasgos de los clientes con alto grado de fidelidad a la marca, entre otros. Para el descubrimiento de reglas de comportamiento definidos a partir de atributos clases en un dominio de problema que representa la masa de información disponible, se propone la utilización de algoritmos de inducción TDIDT para descubrir las reglas de comportamiento de cada atributo clase. Este proceso y sus subproductos pueden ser visualizados gráficamente en la figura 2.17. Como resultado de la aplicación del algoritmo de inducción TDIDT al atributo clase se obtiene un conjunto de reglas que definen el comportamiento de dicha clase.

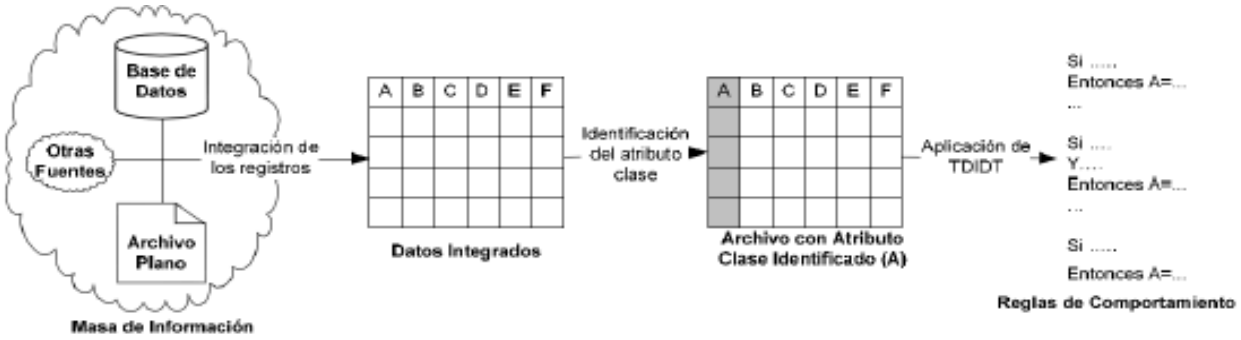


Figura 2.17. *Proceso de descubrimiento de reglas de comportamiento [Britos y García-Martínez, 2009]*

Descubrimiento de Grupos [Britos y García-Martínez, 2009]: aplica cuando se requiere identificar una partición en la masa de información disponible sobre el dominio de problema. Son ejemplos de problemas que requieren este proceso: identificación de segmentos de clientes para bancos y financieras, identificación de tipos de llamadas de clientes para empresas de telecomunicación, identificación de grupos sociales con las mismas características, identificación de grupos de estudiantes con características homogéneas, entre otros. Para el descubrimiento de grupos a partir de masas de información del dominio de problema sobre las que no se dispone ningún criterio de agrupamiento "a priori" se propone la utilización de algoritmos de Clustering, por ejemplo Mapas Auto Organizados de Kohonen o SOM por su sigla en inglés. El uso de esta tecnología busca descubrir si existen grupos que permitan una partición representativa del dominio de problema que la masa de información disponible representa. Este proceso y sus subproductos pueden ser visualizados gráficamente en la Figura 2.18.

En primer lugar se identifican todas las fuentes de información (bases de datos, archivos planos, entre otras), se integran entre sí formando una sola fuente de información a la que se llamará datos integrados. Con base en los datos integrados se aplica SOM. Como resultado de la aplicación de SOM se obtiene una partición del conjunto de registros en distintos grupos a los que se llamará grupos identificados. Para cada grupo identificado se generará el archivo correspondiente.

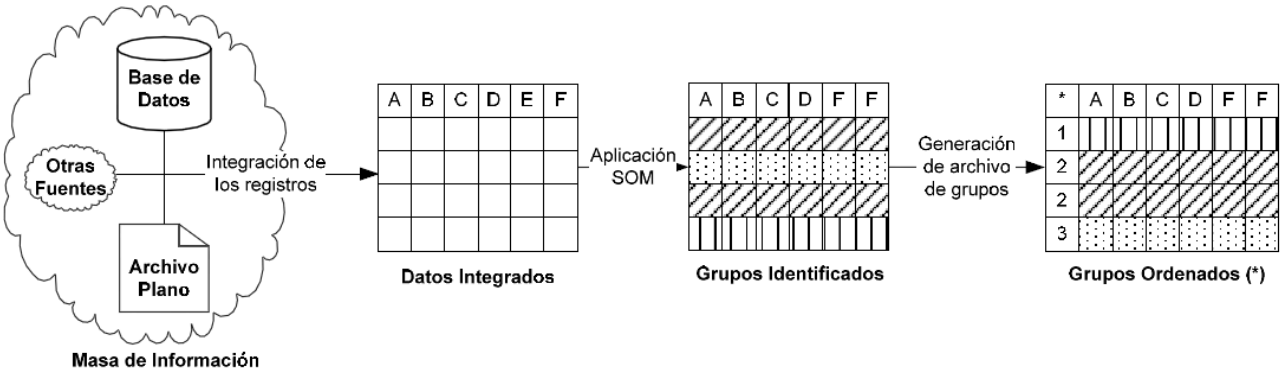


Figura 2.18. *Proceso de descubrimiento de grupos [Britos y García-Martínez, 2009]*

Ponderación de interdependencia de atributos [Britos y García-Martínez, 2009]: aplica cuando se requiere identificar cuáles son los factores con mayor incidencia (o frecuencia de ocurrencia) sobre un determinado resultado del problema. Son ejemplos de problemas que requieren este proceso: factores con incidencia sobre las ventas, rasgos distintivos de clientes con alto grado de fidelidad a la marca, atributos claves que convierten en vendible a un determinado producto, características sobresalientes que tienen los visitantes de un website, entre otros. Para ponderar en qué medida la variación de los valores de un atributo incide sobre la variación del valor de un atributo clase se propone la utilización de Redes Bayesianas. El uso de esta tecnología busca identificar si existe interdependencia en algún grado entre los atributos que modelan el dominio de problema que la masa de información disponible representa. Este proceso y sus subproductos pueden ser visualizados gráficamente en la Figura 2.19.

En primer lugar se identifican todas las fuentes de información (bases de datos, archivos planos, entre otras), se integran entre sí formando una sola fuente de información a la que se llamará datos integrados. Con base en los datos integrados se selecciona el atributo clase (atributo A en la Figura 2.19). Como resultado de la aplicación del aprendizaje estructural de las Redes Bayesianas al archivo con atributo clase identificado se obtiene el árbol de aprendizaje; a este se le aplica el aprendizaje predictivo Redes Bayesianas y se obtiene el árbol de ponderación de interdependencias que tiene como raíz al atributo clase y como nodos hojas a los otros atributos con la frecuencia (incidencia) sobre el atributo clase.

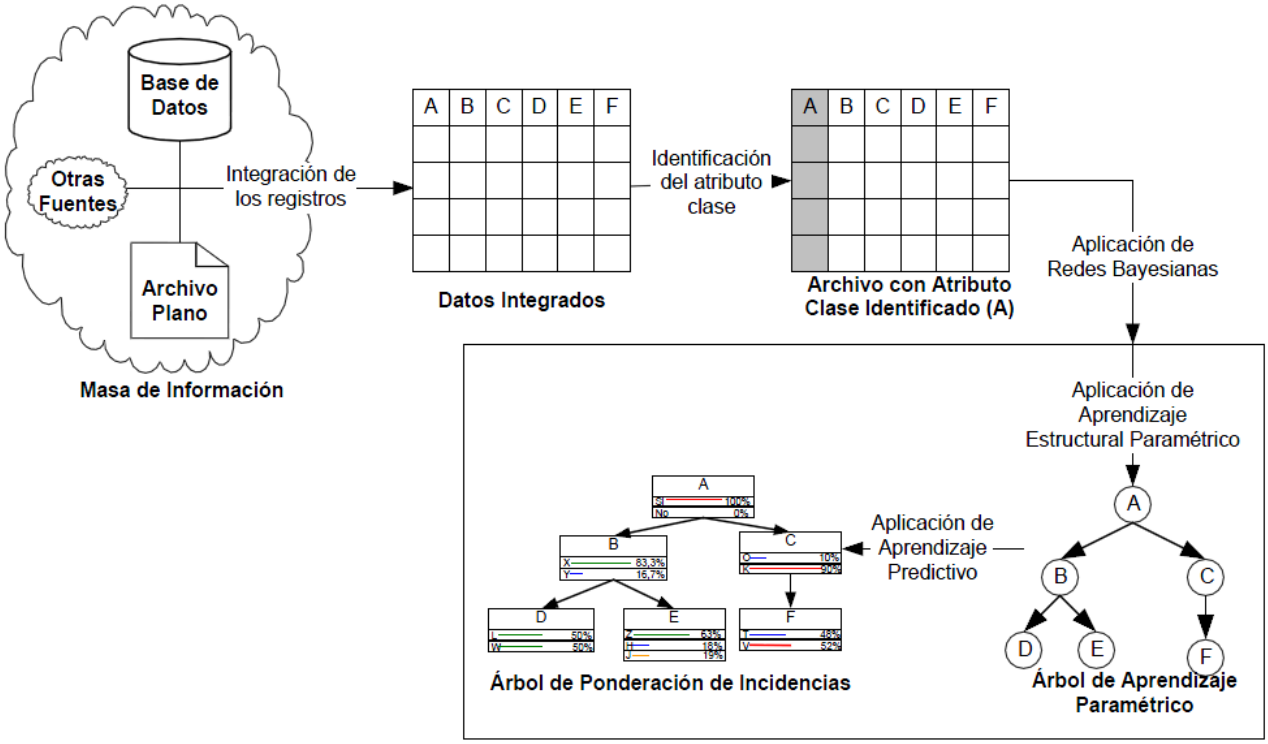


Figura 2.19. *Proceso de ponderación de interdependencia de atributos [Britos y García-Martínez, 2009]*

Descubrimiento de Reglas de Pertenencia a Grupos [Britos y García-Martínez, 2009]: aplica cuando se requiere identificar cuáles son las condiciones de pertenencia a cada una de las clases en una partición desconocida "a priori", pero presente en la masa de información disponible sobre el dominio de problema. Son ejemplos de problemas que requieren este proceso: tipología de perfiles de clientes y caracterización de cada tipología, distribución y estructura de los datos de mi website, segmentación etaria de mis estudiantes y comportamiento de cada segmento, clases de llamadas telefónicas en una región y caracterización de cada clase, entre otros. Para el descubrimiento de reglas de pertenencia a grupos se propone la utilización de SOM para el hallazgo de los mismos y; una vez identificados los grupos, la utilización de algoritmos de inducción (TDIDT) para establecer las reglas de pertenencia a cada uno. Este proceso y sus subproductos pueden ser visualizados gráficamente en la figura 2.20.

En primer lugar se identifican todas las fuentes de información (bases de datos, archivos planos, entre otras), se integran entre sí formando una sola fuente de información a la que se llamará datos integrados. Con base en los datos integrados se aplican mapas auto-organizados (SOM). Como resultado de la aplicación de SOM se obtiene una partición del conjunto de registros en distintos grupos a los que se llama grupos identificados. Se generan los archivos asociados a cada grupo identi-

ficado. A este conjunto de archivos se lo llama grupos ordenados. El atributo "grupo" de cada grupo ordenado se identifica como el atributo clase de dicho grupo, constituyéndose este en un archivo con atributo clase identificado (GR). Se aplica el algoritmo de inducción TDIDT al atributo clase de cada grupo GR y se obtiene un conjunto de reglas que definen el comportamiento de cada grupo.

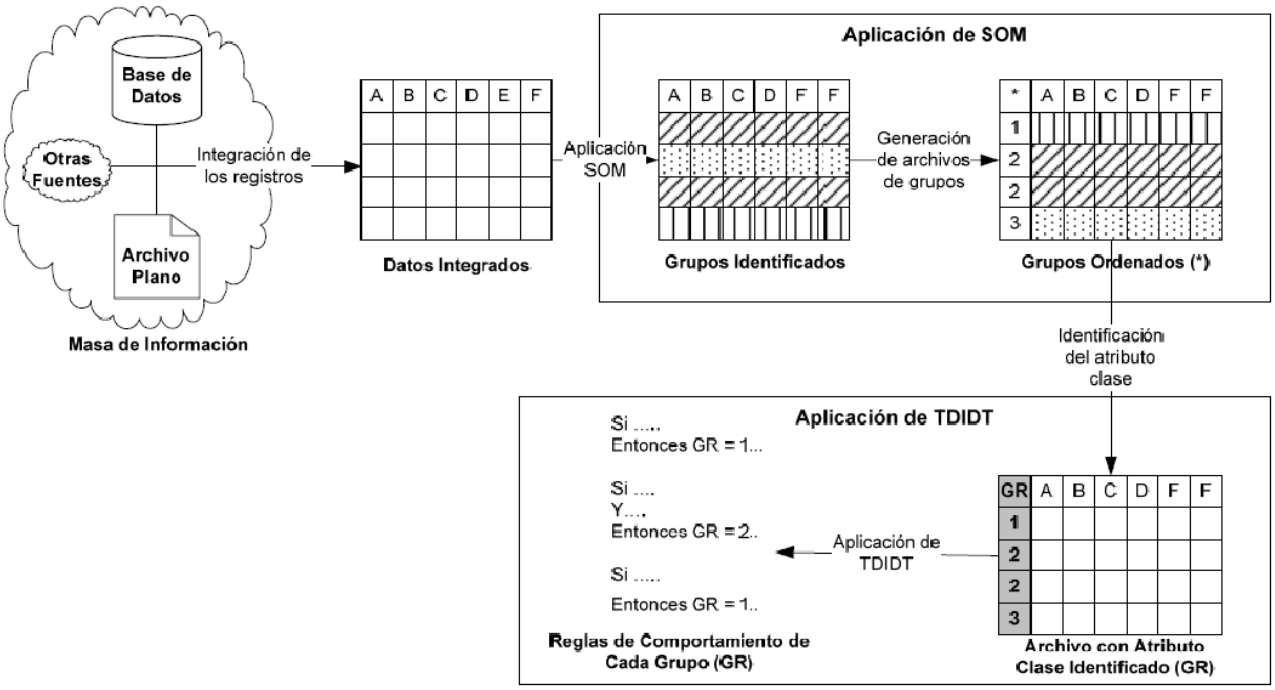


Figura 2.20. *Proceso de descubrimiento de reglas de pertenencia a grupos [Britos y García-Martínez, 2009]*

Ponderación de reglas de Comportamiento o de Pertenencia a Grupos [Britos y García-Martínez, 2009]: aplica cuando se requiere identificar cuáles son las condiciones con mayor incidencia (o frecuencia de ocurrencia) sobre la obtención de un determinado resultado en el dominio del problema, sean estas las que en mayor medida inciden sobre un comportamiento o las que mejor definen la pertenencia a un grupo. Son ejemplos de problemas que requieren este proceso: identificación del factor dominante que incide en el alza las ventas de un producto dado, rasgo con mayor presencia en los clientes con alto grado de fidelidad a la marca, frecuencia de ocurrencia de cada perfil de clientes, identificación del tipo de llamada más frecuente en una región, entre otros. Para la ponderación de reglas de comportamiento o de pertenencia a grupos se propone la utilización de redes bayesianas. Esto puede hacerse a partir de dos procedimientos dependiendo de las características del problema a resolver: cuando no hay clases/grupos identificados; o cuando hay clases/grupos identificados. El procedimiento a aplicar cuando hay clases/grupos identificados consiste en la utilización de algoritmos de inducción TDIDT para descubrir las reglas de comportamiento de cada atributo clase y posteriormente se utiliza redes bayesianas para descubrir cuál de los atributos establecidos

como antecedentes de las reglas tiene mayor incidencia sobre el atributo establecido como consecuente. Este proceso y sus subproductos pueden ser visualizados gráficamente en la Figura 2.21.

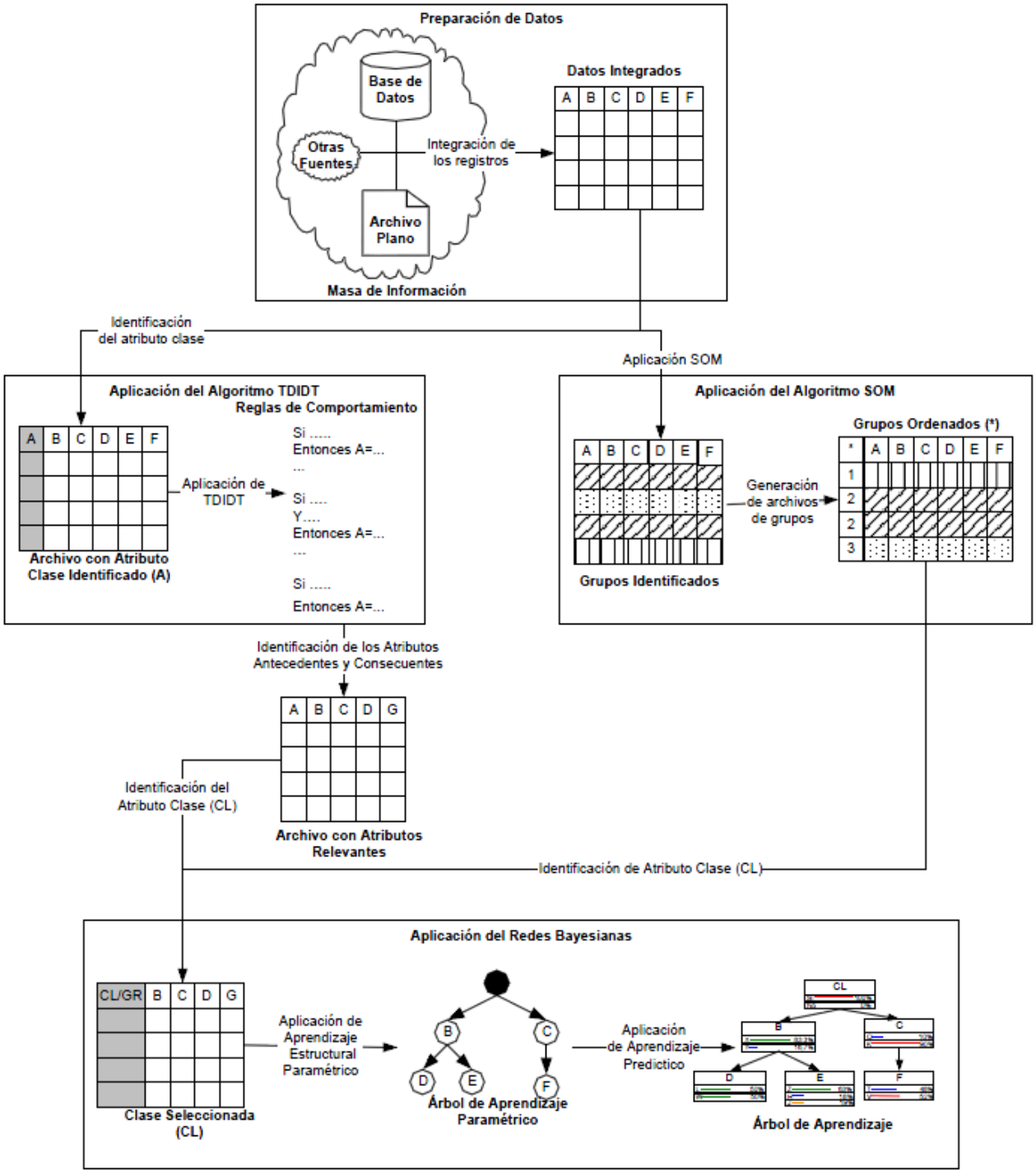


Figura 2.21. *Proceso de ponderación de reglas de comportamiento o de pertenencia a grupos [Britos y García-Martínez, 2009]*

En primer lugar se identifican todas las fuentes de información (bases de datos, archivos planos, entre otras), se integran entre sí formando una sola fuente de información a la que se llamará datos integrados. Con base en los datos integrados se selecciona el atributo clase (atributo A en la figura

2.21). Como resultado de la aplicación del algoritmo de inducción TDIDT al atributo clase se obtiene un conjunto de reglas que definen el comportamiento de dicha clase. Seguidamente, se construye un archivo con los atributos antecedentes y consecuentes identificados por la aplicación del algoritmo TDIDT. Como resultado de la aplicación del aprendizaje estructural de las Redes Bayesianas al archivo con atributo clase obtenido por la utilización del algoritmo TDIDT (en la fFgura 21), se obtiene el árbol de aprendizaje; a este se le aplica aprendizaje predictivo y se obtiene el árbol de ponderación de interdependencias que tiene como raíz al atributo clase (en este caso el atributo consecuente) y como nodos hojas a los atributos antecedentes con la frecuencia (incidencia) sobre el atributo consecuente. El procedimiento a aplicar cuando no hay clases/grupos identificados consiste en identificar todas las fuentes de información (bases de datos, archivos planos, entre otras), se integran entre sí formando una sola fuente de información a la que se llamará datos integrados. Con base en los datos integrados se aplican mapas auto-organizados (SOM). Como resultado de la aplicación de SOM se obtiene una partición del conjunto de registros en distintos grupos a los que se llamará grupos identificados. Para cada grupo identificado se generará el archivo correspondiente. A este conjunto de archivos se lo llama grupos ordenados. El atributo “grupo” de cada grupo ordenado se identifica como el atributo clase de dicho grupo, constituyéndose este en un archivo con atributo clase identificado (GR). Como resultado de la aplicación del aprendizaje estructural se obtiene el árbol de aprendizaje; a este se le aplica el aprendizaje predictivo y se obtiene el árbol de ponderación de interdependencias que tiene como raíz al atributo grupo y como nodos hojas a los otros atributos con la frecuencia (incidencia) sobre el atributo grupo.

Puede observarse que para la ejecución de todos los procesos de explotación de información tradicional, la entrada es una tabla con todos los datos agrupados. Para poder extraer información y patrones interesantes de una base de datos compuesta por grafos, es necesario realizar procedimientos diferentes que permitan analizar la estructura de los mismos. Por esto surge la necesidad de desarrollar otros tipos de algoritmos, distintos a los usados para las bases de datos tradicionales. Esto de describirá con más detalle en la Sección 2.4.

2.4. EXPLOTACIÓN DE INFORMACIÓN EN GRAFOS

Por lo expuesto en la Sección 2.1 se puede ver la cantidad de aplicaciones que tiene la teoría de grafos y su potencial. Se puede observar también, que sería muy útil poder aplicar los conceptos de explotación de información introducidos en la Sección 2.3, de manera que se puedan aprovechar aún más las características de los grafos. Sin embargo, para poder extraer patrones e información oculta de estas estructuras, se aplican distintos enfoques a los previamente mencionados. Estos procesos son comúnmente conocidos como minería de grafos o *Graph Mining* en inglés. Uno de estos

métodos consiste en, dado un gran grafo conexo, buscar la patrones de manera que se pueda comprimir. El algoritmo SUBDUE [Holder et al., 1994] es un buen ejemplo, aunque también puede ser utilizado para otras aplicaciones.

Otro tipo de proceso es la búsqueda de sub-isomorfismos, en el cual se verifica si un grafo es parte de otro grafo de mayor tamaño. El algoritmo de Ullman [Ullman, 1976] es el más conocido.

En las siguientes secciones se explicará el procedimiento de interés para este trabajo junto con los algoritmos escogidos para realizar la comparación: la búsqueda de subgrafos frecuentes o FSM (*Frequent Subgraph Mining).*

2.4.1. BÚSQUEDA DE SUBGRAFOS FRECUENTES

La búsqueda de subgrafos frecuentes se basa en encontrar estructuras recurrentes en un conjunto de grafos. Es decir, buscar subgrafos que se repitan en una base de datos compuesta por grafos. El descubrimiento de estos patrones puede ser el propósito final del proceso o los subgrafos descubiertos pueden ser parte de otro proceso de clasificación.

Para determinar que un subgrafo es frecuente, tiene que superar determinado umbral, denotado como *support.* La definición formal es la siguiente: dado un conjunto de grafos *GD* y un umbral σ (o *threshold* en inglés), de manera que $0 < \sigma \leq 1$, el soporte de un grafo *G,* denotado como sup_G, es igual a la cantidad de grafos en *GD* en los cuales *G* es un sub-isomorfismo. Escrito como fórmula sería:

$$Sup_G = \frac{|\{G' \in GD \mid G \subseteq G'\}|}{|GD|}$$

Teniendo en cuenta esto, un grafo G es frecuente en una base de datos de grafos si $sup_G \geq \sigma$, siendo σ el soporte mínimo o *minimum support*. Por lo tanto, el problema de FSM se resume en, dado un umbral σ y un conjunto de grafos *GD,* encontrar todos los subgrafos frecuentes *G* en *GD* que cumplan con $sup_G \geq \sigma$.

A continuación se describen las características distintivas de los algoritmos de FSM que se evalúan en el presente trabajo de investigación, ordenados en orden cronológico en orden ascendente según sus fechas de publicación: FSG (Sección 2.4.1.1), gSpan (Sección 2.4.1.2), FFSM (Sección 2.4.1.3) y GASTON (Sección 2.4.1.4).

2.4.1.1. ALGORITMO FSG

El algoritmo FSG [Kuramochi et al., 2001] es el algoritmo más antiguo de los que se evalúan en este trabajo, por lo que introdujo algunas características que luego serían usados por el resto, que tienen que ver con la forma de representación de los grafos, la generación de subestructuras candi-

datas y detección de isomorfismos. En primer lugar, utiliza un tipo de representación para grafos dispersos que minimiza costos de procesamiento y almacenamiento. Este tipo de representación es usada para almacenar candidatos intermedios y los subgrafos que se van encontrando. Consiste en transformar las representaciones canónicas de los grafos, inicialmente representadas con matrices de adyacencia, para implementarlos como listas de adyacencia, las cuales disminuyen el uso de memoria y del procesador para estructuras dispersas. En segundo lugar, incrementa el tamaño de los subgrafos a buscar de a una arista por vez, permitiendo que la generación de candidatos sea más eficiente. Finalmente, usa algoritmos simples para implementar las representaciones canónicas de los grafos y las detecciones de isomorfismos que funcionan de manera eficiente tanto para grafos chicos, e incorpora varias optimizaciones para el proceso de generación de candidatos y conteo (determinación de la frecuencia de un subgrafo) que permiten que el algoritmo sea escalable para grandes conjuntos de grafos. En la Figura 2.22 se presenta el pseudocódigo de la ejecución general de la rutina principal del algoritmo FSG. En la Tabla 2.1 se resume la notación utilizada.

Algoritmo: FSG

Entrada: D, σ // base de Grafos y mínimum support

Salida: Conjunto de subgrafos frecuentes

```
1:  F^1 = detección de subgrafos frecuentes en D con 1 arista
2:  F^2 = detección de subgrafos frecuentes en D con 2 aristas
3:  k =3
4:  MIENTRAS F^(k-1) ≠ ∅ HACER
5:      C^k = cantidatos-fsg(F^(k-1))
6:          PARA CADA g^k ∈ C^k HACER
7:              g^k.conteo = 0
8:              PARA CADA t ∈ D HACER
9:                  SI g^k esta incluido en t ENTONCES
10:                     g^k.conteo = g^k.conteo+1
11:     F^k = {g^k ∈ C^k | g^k.conteo ≥ σ }
12:     k=k+1
13: RETURN {F^1, F^2,..., F^(k-2)}
```

Figura 2.22. *Rutina principal del algoritmo FSG [Kuramochi et al.,2001]*

Notación	Descripción
t	*Un grafo (o transacción) de la base de datos D*
g^k	*Subgrafo con k aristas*
C^k	*Conjunto de candidatos con k aristas*
F^k	*Conjunto de subgrafos-k frecuentes*
$cl(g^k)$	*Representación canónica de un grafo-k* g^k

Tabla 2.1. *Notación utilizada para la descripción del algoritmo FSG [Kuramochi et al., 2001].*

2.4.1.2. ALGORITMO GSPAN

El algoritmo gSpan (*graph-based Subestructure pattern mining*) [Yan et al., 2002] busca superar los inconvenientes que tienen los algoritmos que utilizan una estrategia apriorística como el FSG: el costo de la generación de candidatos y la detección de falsos positivos a la hora de la evaluación de isomorfismos. Como el problema de búsqueda de sub-isomorfismos está catalogado como NP-completo, es muy costoso computacionalmente volver a evaluar los resultados.

Lo más destacado de este algoritmo es la implementación del recorrido en profundidad para reducir el espacio de búsqueda y la introducción de dos nuevas técnicas: los códigos DFS y DFSM para generar la representación canónica de los grafos (ver Sección 2.2.2.3).

Otros aspectos destacables incluyen la eliminación de los procesos de generación de candidatos para el descubrimiento de subgrafos, así como también el recorte de falsos positivos. Además, combina los procedimientos de crecimiento y evaluación de subestructuras en uno solo, acelerando el proceso de búsqueda. En la Figura 2.23 se presenta el pseudocódigo correspondiente a la subrutina de búsqueda de patrones del algoritmo gSpan y en la Figura 2.24 la rutina principal. En la Tabla 2.2 se resume la notación utilizada.

Notación	Descripción
D	*Base de datos compuesta por grafos*
S	*Subestructuras encontradas*
s	*Subgrafo*
D_s	*Conjunto de grafos en los cuales s es un subgrafo*
minSup	*σ, minimum support*
min(s)	*Código DFSM de s*

Tabla 2.2. *Notación utilizada para la descripción del algoritmo gSpan.*

Subrutina: buscar_subgrafos

Entrada: *D, S, s*

Salida: Conjunto de subgrafos frecuentes (almacenados en *S*)

```
1: SI s ≠ min(s) ENTONCES
2:     RETURN
3: S = S ∪ {s}
4: enumerar s in cada grafo en D y contar sus hijos
5: PARA CADA hijo de c de s HACER
6:     SI support(c) ≥ minSup ENTONCES
7:         s = c
8:         buscar_subgrafos(Ds, S, s)
```

Figura 2.23. *Subrutina de búsqueda de subgrafos de gSpan [Yan et al., 2002].*

Algoritmo: gSpan	
Entrada: *D*	
Salida: Conjunto de subgrafos frecuentes (almacenados en *S*)	
1:	ordenar las etiquetas en D por su frecuencia
2:	remover los vértices y aristas no frecuentes
3:	volver a etiquetar los vértices y aristas restantes
4:	S^1 = todos los subgrafos frecuentes de D con una arista
5:	ordenar S^1 por su código DFS en orden lexicográfico
6:	$S = S^1$
7:	**PARA CADA** arista $e \in S^1$ **HACER**
8:	s = *e*
9:	D_s = grafos que contengan a *e*
10:	buscar_subgrafos(D, S, s)
11:	D=D - *e*
12:	**SI** |D| < *minSup* **ENTONCES** salir del bucle

Figura 2.24. *Rutina principal del algoritmo gSpan [Yan et al., 2002].*

2.4.1.3. ALGORITMO FFSM

El algoritmo FFSM (*Fast Frequent Subgraph Mining*) [Huan et al., 2003] utiliza el mismo enfoque de búsqueda en profundidad del gSpan, incorporando nuevas técnicas para mejorar su eficiencia.

Entre las mencionadas técnicas está la utilización de una nueva forma canónica CAM (ver Sección 2.2.2.3) y dos operaciones que denominaron FFSM-join y FFSM-extension, que se utilizan para agilizar el proceso de generación de candidatos mediante la manipulación de las matrices de adyacencia. Se introduce un procedimiento para garantizar que todos las subestructuras frecuentes sean enumerada unívocamente y sin ambigüedades (suboptimal CAM tree) y se evita el testeo de sub-isomorfismos, manteniendo una lista de cada subgrafo frecuente. Esta última es quizá la más relevante de todas las características, debido al gran potencial que puede llegar a tener el algoritmo al evitar ese procedimiento tan computacionalmente complejo.

En la figura 2.25 se muestra el proceso de inicialización para el algoritmo FFSM y en la figura 2.26 se describe el pseudocódigo del proceso de búsqueda de patrones.

Algoritmo: FFSM	
Entrada: conjunto de grafos *GD* y un support σ	
Salida: Conjunto de subgrafos frecuentes (almacenados en *S*)	
1:	*S* = {códigos CAM de los vértices y aristas frecuentes}
2:	*P* = {códigos CAM de las aristas frecuentes}
3:	FFSM-explore(*P*, *S*)

Figura 2.25. *Proceso de inicialización del algoritmo FFSM [Huan et al., 2003].*

Subrutina: FFSM-Explore

Entrada: *Suboptimal CAM list P* y un conjunto de códigos CAM *W*

Salida: conjunto de códigos CAM de todos los subgrafos frecuentes buscados hasta el momento, almacenados en *W*

```
1:  PARA CADA X ∈ P HACER
2:      SI X es CAM del grafo que representa ENTONCES
3:          W = W ∪{X}
4:          C = Φ
5:          PARA CADA Y ∈ P HACER
6:              C = C ∪ FFSM-Join(X, Y)
7:          C = C ∪ FFSM-Extension(X)
8:          remover código(s) CAM de C que no son frecuentes o no sean óptimos
9:          FFSM-Explore(C, W)
10: RETURN
```

Figura 2.26. *Pseudocódigo del proceso de búsqueda de patrones del algoritmo FFSM [Huan et al., 2003].*

No se describe el pseudocódigo de los procesos FFSM-Join y FFSM-Extension, los cuales generan un nuevo candidato a partir de dos grafos y extienden en una arista a un candidato respectivamente, utilizando las matrices CAM.

2.4.1.4. ALGORITMO GASTON

El algoritmo GASTON (*GrAph/Sequence/Tree extractiON*) [Nijssen et al., 2004] aprovecha un principio que llamaron 'quickstart principle', que considera siguiente hecho: los grafos, árboles y caminos están incluidos unos en otros, por lo que se puede dividir el proceso en distintos pasos de creciente complejidad, lo que simplifica el procedimiento general. Primero, se buscan los caminos frecuentes, luego los árboles y finalmente los subgrafos frecuentes.

Cada etapa tiene un proceso distinto para representar las estructuras en su forma canónica, debido a las distintas características que presentan dichas estructuras. Sin embargo, debido al *quickstart principle* antes mencionado, los códigos generados en una etapa pueden ser usados para la etapa siguiente, concatenando las nuevas ramificaciones encontradas en el caso de los árboles o los ciclos en el caso de los grafos. De esta manera, se reduce la complejidad del proceso.

Para el proceso de conteo de grafos se utiliza un procedimiento similar al del algoritmo FFSM, con listas que almacenan las subestructuras ya analizadas. Debido a problemas de escalabilidad con este método, también proponen utilizar un proceso alternativo para grafos de gran tamaño, similar al utilizado en el algoritmo FSG. En la Figura 2.27 se describe el pseudocódigo del proceso de búsqueda de caminos, en la Figura 2.28 la búsqueda de árboles y en la figura 2.29 la búsqueda de grafos cíclicos correspondientes al algoritmo GASTON.

Algoritmo: Gaston_buscar_caminos

Entrada: *C, R, S //* camino inicial, refinamientos, estructuras encontradas

Salida: Conjunto de estructuras frecuentes en *S*

```
1:  PARA CADA r ∈ R HACER
2:      n=C extendido por r
3:      SI r genera un ciclo en n ENTONCES
4:          SI n es un nuevo ciclo no encontrado ENTONCES
5:              S = S U {nuevoCiclo(n)}
6:      SINO SI r convierte a n en un árbol válido ENTONCES
7:          S = S U {nuevoArbol(n)}
8:          Gaston_buscar_arboles(n, extender_y_unificar_arboles(r, R), S)
9:      SINO SI r convierte a n en un camino válido ENTONCES
10:         S = S U {nuevoCamino(n)}
11:         Gaston_buscar_caminos(n, extender_y_unificar_caminos(r, R), S)
```

Figura 2.27. *Búsqueda de caminos del algoritmo GASTON [Worlein, 2005]*

Algoritmo: Gaston_buscar_arboles

Entrada: *T, R, S //* árbol inicial, refinamientos, estructuras encontradas

Salida: Conjunto de estructuras frecuentes en *S*

```
1:  PARA CADA r ∈ R HACER
2:      n=T extendido por r
3:      SI r genera un ciclo en n ENTONCES
4:          SI n es un nuevo grafo no encontrado ENTONCES
5:              S = S U {nuevoGrafo(n)}
6:              Gaston_buscar_grafos(n, {r' ∈ R / r' > r}, S)
7:      SINO SI r convierte a n en un árbol válido ENTONCES
8:          S = S U {nuevoArbol(n)}
9:          Gaston_buscar_arboles(n, extender_y_unificar_arboles(r, R), S)
```

Figura 2.28. *Búsqueda de árboles del algoritmo GASTON [Worlein, 2005]*

Algoritmo: Gaston_buscar_grafos

Entrada: *G, R, S //* grafo inicial, refinamientos, estructuras encontradas

Salida: Conjunto de estructuras frecuentes en *S*

```
1:  PARA CADA r ∈ R HACER
2:      n=T extendido por r
3:      SI n es un nuevo grafo no encontrado ENTONCES
4:          S = S U {nuevoGrafo(n)}
5:          Gaston_buscar_grafos(n, {r' ∈ R / r' > r}, S)
```

Figura 2.29. *Búsqueda de grafos del algoritmo GASTON [Worlein, 2005]*

3. DESCRIPCIÓN DEL PROBLEMA

En este capítulo se presenta el problema de investigación a desarrollar en el trabajo final de Licenciatura. Primero se identifica y describe el problema de investigación (Sección 3.1), luego se caracteriza el problema abierto (Sección 3.2) y finalmente se desarrolla el sumario de investigación correspondiente (Sección 3.3).

3.1. IDENTIFICACIÓN DEL PROBLMA DE INVESTIGACIÓN

Los grafos son de gran utilidad cuando se requiere realizar un modelo de una estructura compleja que no pueda ser concebida mediante bases de datos tradicionales. Un ejemplo son las redes de comunicación, en las cuales los nodos representan a los elementos emisores y receptores mientras los arcos unen a aquellos que se comunican, como se ve en la figura 3.1. Agregando etiquetas a cada elemento se puede añadir información adicional del tipo de comunicación o características particulares de interés de los nodos.

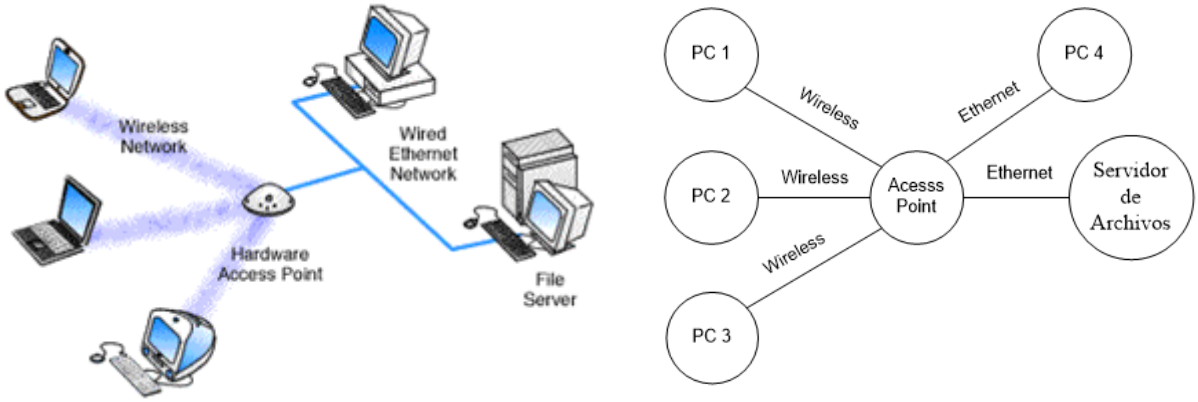


Figura 3.1. *Ejemplo de red de comunicación con componentes electrónicos y su posible modelización utilizando grafos*

Este es un ejemplo simple, pero en una estructura más grande, detallada y compleja podría resultar útil realizar un proceso de explotación de información sobre el modelo de manera que se puedan descubrir patrones o información relevante oculta a simple vista que ayude a mejorar la red, detectar inconvenientes que puedan llegar a ocurrir a futuro o reducir elementos innecesarios.

Dependiendo de la base de datos utilizada y los resultados que se esperen obtener, se pueden realizar procesos como la compresión de grafos [Feder, 1991], búsqueda de isomorfismos [Ullman, 1976] o búsqueda de patrones entre varios grafos.

El proceso de interés para este trabajo es el de encontrar subgrafos frecuentes entre un conjunto de grafos dada la frecuencia de ocurrencia deseada para los mismos. Si bien este proceso tiene un gran potencial, muchas veces se ve limitado por la complejidad computacional que conlleva, teniendo en cuenta que una de las tareas necesarias es realizar una búsqueda de (sub)isomorfismos para com-

probar la frecuencia de determinado subgrafo. De por sí, solamente este proceso se encuadra dentro de los problemas NP-Completos [Fortin, 1996]. Considerando que este solo es un paso que se repite varias veces dentro de todo el proceso, es necesaria la utilización de enfoques que permitan realizar las búsquedas sin el uso desmedido de recursos computacionales.

Dada esta situación, se han desarrollado varios algoritmos de minería de grafos en los últimos años [Yan et al. 2002, Huan et al., 2004, Cook et al., 2000]. Los mismos, varían en la estrategia que utilizan para recorrer los grafos, el tipo de entrada que utilizan y la información de salida que proveen. Un resumen de los más conocidos clasificados según los criterios antes mencionados puede verse en la figura 3.2. En ella puede verse aquellos algoritmos que utilizan búsqueda en profundidad o anchura, aquellos que utilizan un grafo como entrada o un conjunto de grafos y los que como salida brindan todas las subestructuras encontradas o sólo una parte del total, considerada más relevante que el resto. Teniendo tantas posibilidades, es conveniente poder determinar que algoritmo es más apropiado elegir en base al conjunto de datos que se quiera trabajar.

Nótese que para poder realizar una comparación de rendimiento en cuanto a tiempos de ejecución y resultados obtenidos, los algoritmos a comparar deben pertenecer al mismo grupo tanto en tipo de entrada como tipo de salida. En esta investigación se hace hincapié en aquellos que reciben un conjunto de grafos y otorgan como resultado el conjunto entero de subestructuras obtenidas.

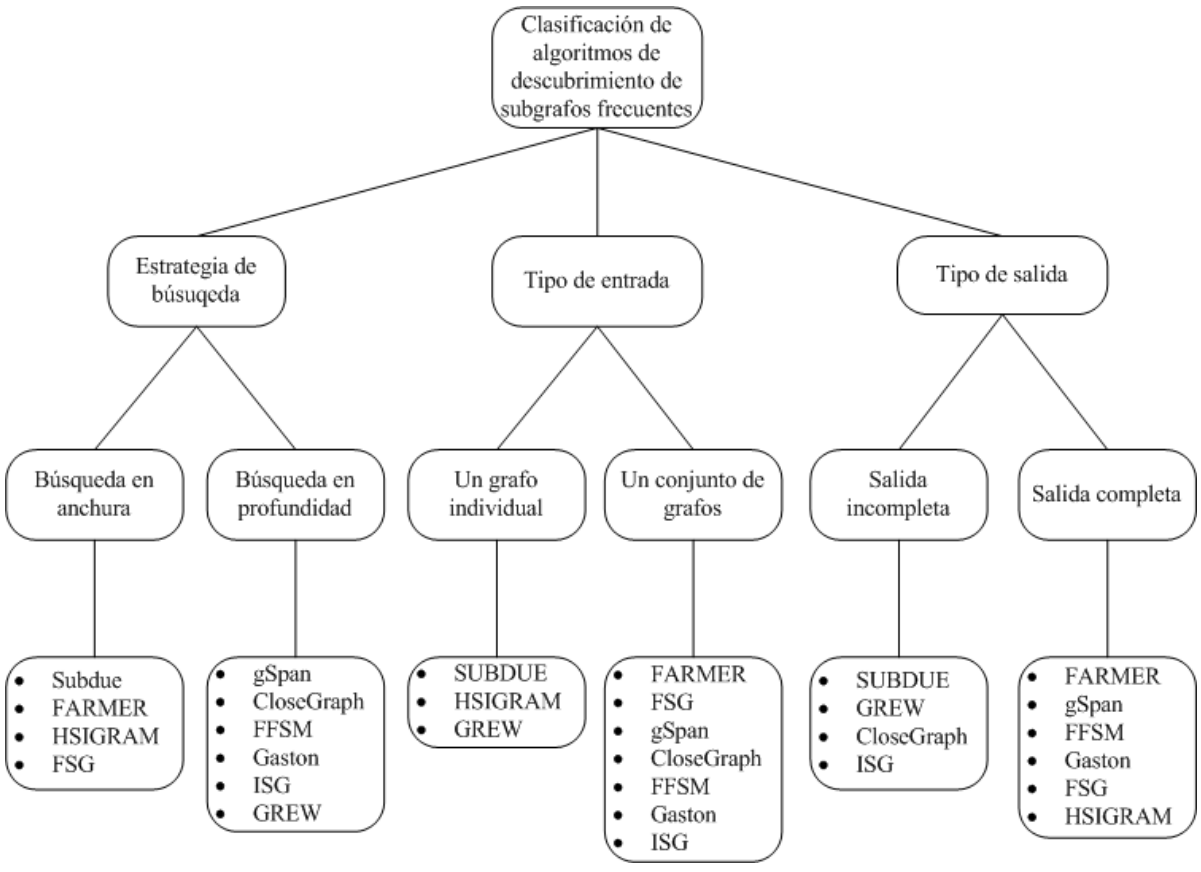


Figura 3.2. *Clasificación de los algoritmos más relevantes para la búsqueda de patrones en grafos [Krishna et al. 2011].*

3.2. PROBLEMA ABIERTO

Con el gran crecimiento de internet y las redes sociales en las últimas décadas el interés por los problemas de búsqueda de patrones en grafos se ha incrementado notoriamente. Teniendo en cuenta la cantidad de datos que se manejan en este tipo de redes, un proceso de explotación de información puede ser extremadamente útil. Sin embargo, el alcance de estos procesos es mucho más amplio y se le han encontrado muchas otras aplicaciones.

Un ejemplo es en el área de la bioremediación, la cual consiste en la explotación de catalizadores biológicos para eliminar agentes contaminantes del ambiente. En este caso, se puede utilizar la minería de grafos para reconocer que componentes de una molécula pueden reaccionar con algún microorganismo para atacar compuestos contaminantes [Zaiat et al., 2007].

Otra aplicación consiste en la de predicción de interacciones entre actores en una red temporal. La idea es buscar subgrafos frecuentes entre un conjunto de grafos que representan interacciones a través del tiempo [Lahiri et al., 2007].

Se ha trabajado también con clasificaciones de compuestos químicos cuya aplicación es empleada, por ejemplo, en la creación de drogas que respondan apropiadamente a una enfermedad con una mínima cantidad de efectos adversos [Takigawa et al. 2013]. En este caso es necesario analizar la estructura molecular de los compuestos intervinientes por lo que la aplicación de procesos de explotación de información en grafos es sumamente útil.

Debido a la cantidad de posibilidades que surgen de este tipo de procesos es que en los últimos años se han desarrollado una gran variedad de algoritmos que resuelven el problema de la búsqueda de subgrafos frecuentes en conjuntos de grafos. El problema con esto surge justamente a partir de la cantidad de algoritmos existentes y la necesidad de escoger los adecuados para el trabajo que se quiera realizar. Es útil conocer cómo se comporta cada algoritmo para elegir al más adecuado.

Si bien se han hecho comparaciones entre algoritmos [Krishna et al. 2011, Rehman et al., 2014], las mismas sólo consideran comparar el tiempo de ejecución de los algoritmos en base a distintas estructuras y definir cuál es el más rápido. Se considera de mayor interés ampliar las investigaciones determinando bajo qué circunstancias es más eficiente uno que otro, no solo en base al tiempo que demore sino también en la cantidad de patrones que pueda encontrar.

3.3. SUMARIO DE INVESTIGACIÓN

De lo anteriormente expuesto surgen las siguientes preguntas de investigación:

> Pregunta 1: ¿Es posible integrar distintas implementaciones de algoritmos para probar su comportamiento en diversos experimentos, permitiendo comparar el desempeño de los mismos en distintos escenarios?

Pregunta 2: De ser posible la experimentación, ¿es posible determinar cuáles son los escenarios más favorables para la utilización de cada algoritmo, de manera que se pueda facilitar la elección de los mismos dependiendo las necesidades que se tengan y los datos disponibles?

4. SOLUCIÓN

De manera que se pueda verificar si existe algún algoritmo más eficiente que otro en determinado escenario, se desarrollan distintos experimentos variando la estructura de los grafos de entrada para después medir los resultados y compararlos. Cada experimento, a su vez, está compuesto por diversas pruebas en las que se comprobará la reacción de los algoritmos a medida que aumenta el tamaño del dataset utilizado.

Para esta tarea se usan implementaciones de los algoritmos GASTON, FFSM, FSG y gSpan desarrolladas por los autores de los mismos y se crea una capa de software que los ejecuta y mide los distintos resultados obtenidos. Esta capa recibe el nombre de banco de pruebas.

A continuación en las siguientes secciones se describen los detalles del diseño experimental. Primero, se describen los procedimientos a realizar y las variables dependientes e independientes que intervienen en cada experimento tanto con grafos sintéticos como con grafos reales (Sección 4.1). Finalmente se describen detalles de la construcción del banco de pruebas (Sección 4.2), incluyendo el generador de grafos usado (Sección 4.2.1), características de las implementaciones de los algoritmos que se comparan (Sección 4.2.2), como se adaptaron al banco de pruebas (Sección 4.2.3) y por último, un detalle de cómo serán ejecutados los experimentos y cómo se relacionan los distintos elementos del banco de pruebas (Sección 4.2.4) y cómo se evalúan y comparan los resultados (Sección 4.3).

4.1. DISEÑO EXPERIMENTAL

El objetivo de la experimentación es la ejecución de los algoritmos GASTON, gSpan, FSG y FFSM en diferentes escenarios, con diferentes conjuntos de datos, para comprobar su comportamiento y determinar en bajo qué circunstancias se comportan más eficientemente. Además de verificar si existe un algoritmo mejor que el resto, se desea descubrir también: ¿cuáles son los factores que influyen en el rendimiento de cada algoritmo? y ¿de qué manera los afecta?

En las siguientes secciones, se provee una descripción más detallada de los experimentos que se llevan a cabo y sus diferentes variantes (Sección 4.1.2). Posteriormente, se establecen las variables dependientes e independientes involucradas en cada una de las pruebas a realizar tanto para los grafos sintéticos como para los grafos reales (Sección 4.1.3).

4.1.1. DESCRIPCIÓN DE LOS TIPOS DE EXPERIMENTOS

Los algoritmos serán probados en diversos escenarios, los cuales utilizan distintas variables y datos de entrada. Estos escenarios o experimentos se clasifican según las siguientes categorías:

- *Experimentos con grafos sintéticos*: se prueban los algoritmos con datos generados aleatoriamente, variando la cantidad de nodos y arcos. Se utilizan conjuntos de cien grafos en todas las pruebas. Las pruebas se repiten cien veces y los resultados finales son un promedio de los resultados parciales. Este tipo de escenario a su vez contiene otros dos, que varían de acuerdo a cómo se incrementa la cantidad de arcos experimento a experimento (ver Sección 4.2.1). Estas pruebas se dividen en:
    - *Experimentos con incremento fijo, y*
    - *Experimentos con incremento variable.*

  Dentro de estos dos últimos hay otras variaciones que consisten en probar los algoritmos en conjuntos de grafos en los que todos sus nodos son diferentes, así como también en estructuras con repeticiones en las etiquetas de los elementos que los componen. Teniendo en cuenta esto, cada experimento a su vez incluye otros dos:
    - *Experimentos sin repetición de nodos,* y
    - *Experimentos con repetición de nodos.*

  En cada uno de ellos se siguen los mismos procedimientos descritos anteriormente, variando la cantidad de nodos únicos en las pruebas con repetición de nodos (ver Sección 4.2.1).
- *Experimentos con grafos reales*: en estas pruebas se ejecutan los algoritmos sobre un dataset real *Compounds_422* distribuido con la implementación de gSpan. El mismo es utilizado por varios autores para testear el rendimiento de sus algoritmos [Kuramochi et al., 2001, Yan et al., 2002]. Sus características pueden verse en la Tabla 4.1. Sobre esta base de datos se ejecutan pruebas cien veces y se promedian los resultados.

Compounds_422: Archivo de prueba con estructuras moleculares	
Cantidad de Grafos	422
Cantidad de etiquetas de arcos distintas	4
Cantidad de etiquetas de nodos distintas	21
Promedio de arcos por grafo	42
Promedio de nodos por grafo	40
Cantidad máxima de arcos por grafo	196
Cantidad máxima de nodos por grafo	189

Tabla 4.1. *Características del archivo a utilizar en las pruebas con grafos reales.*

4.1.2. VARIABLES

En la siguientes secciones se describen las variables independientes y dependientes que se utilizarán para cada tipo de experimento: con grafos sintéticos (sub-sección 4.3.1) y grafos reales (sub-sección 4.3.2).

4.1.2.1. VARIABLES PARA LOS EXPERIMENTOS CON GRAFOS SINTÉTICOS

En este tipo de experimentos se utilizan grafos generados aleatoriamente en base distintos valores relacionados con su densidad. Se busca comparar el comportamiento de los algoritmos en relación con la estructura de los datos de entrada. A continuación se describen las variables independientes y dependientes involucradas.

4.1.2.1.1. VARIABLES INDEPENDIENTES PARA LOS EXPERIMENTOS CON GRAFOS SINTÉTICOS

Se consideran las siguientes variables independientes para este proceso:

- *Cantidad de Nodos:* Indica el total de nodos que tiene cada grafo del conjunto de prueba en cada experimento. Su valor es de 10 en la primera prueba y se incrementa a 100 en la última, con incrementos de a diez por cada cinco pruebas.
- *Cantidad de Arcos:* Indica el total de arcos que tiene cada grafo del conjunto de prueba en cada experimento. Difiere dependiendo del tipo de experimento. Para aquellos con incremento fijo, su valor es de 10 en la primera prueba y se incrementa hasta 120 en la última, con incrementos de a cinco y reajustando su valor por cada cinco pruebas, como se muestra en las Tablas 4.2.a-c. Para aquellos con incremento variable, comienza en 10 y se incrementa hasta 300. Los incrementos se hacen según la fórmula *(Cantidad de Nodos / 2),* como puede verse en las Tablas 4.3.a-c. La cantidad de arcos se denota como $|E|$.
- *Nodos Únicos:* Indica la cantidad máxima de nodos sin repetirse que puede haber en cada grafo. Es utilizada para las pruebas con repeticiones de nodos. Su valor se corresponde con la fórmula *(Cantidad de Nodos / 2),* garantizando que siempre haya nodos con etiquetas repetidas.
- *Cantidad de grafos:* Indica la cantidad de grafos que tendrán los datos generados para las pruebas. Su valor para todos los experimentos de 100.
- *Minimum Support Threshold o Umbral de Frecuencia Mínima:* Indica la cantidad mínima de grafos que deben contener a una subestructura para considerarla de frecuente. En estos experimentos el valor se fija en 5% para todos los algoritmos.

Exp.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$\|E\|$	10	15	20	25	30	20	25	30	35	40	30	35	40	45	50	40	45	50

Tabla 4.2.a. *Valores de la variable Cantidad de Arcos en cada experimento correspondiente a las pruebas con grafos sintéticos con incremento fijo (Parte a).*

Exp.	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
$\|E\|$	55	60	50	55	60	65	70	60	65	70	75	80	70	75	80	85

Tabla 4.2.b. *Valores de la variable Cantidad de Arcos en cada experimento correspondiente a las pruebas con grafos sintéticos con incremento fijo (Parte b).*

Exp.	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
$\|E\|$	90	80	85	90	95	100	90	95	100	105	110	100	105	110	115	120

Tabla 4.2.c. *Valores de la variable Cantidad de Arcos en cada experimento correspondiente a las pruebas con grafos sintéticos con incremento fijo (Parte c).*

Exp.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
$\|E\|$	10	15	20	25	30	20	30	40	50	60	30	45	60	75	90	40	60	50	75

Tabla 4.3.a. *Valores de la variable Cantidad de Arcos en cada experimento correspondiente a las pruebas con grafos sintéticos con incremento variable (Parte a).*

Exp.	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
$\|E\|$	90	40	60	80	100	120	50	75	100	125	150	90	120	150	180	210

Tabla 4.3.b. *Valores de la variable Cantidad de Arcos en cada experimento correspondiente a las pruebas con grafos sintéticos con incremento variable (Parte b).*

Exp.	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
$\|E\|$	80	120	160	200	240	90	135	180	225	270	100	150	200	250	300

Tabla 4.3.c. *Valores de la variable Cantidad de Arcos en cada experimento correspondiente a las pruebas con grafos sintéticos con incremento variable (Parte c).*

4.1.2.1.2. VARIABLES DEPENDIENTES PARA LOS EXPERIMENTOS CON GRAFOS SINTÉTICOS

Se busca comparar las siguientes variables dependientes para poder evaluar el comportamiento de los algoritmos en cada experimento:

- *Cantidad de subestructuras:* Indica la cantidad de subestructuras frecuentes que los algoritmos encuentran en cada experimento. Con esto se busca identificar en qué situaciones un algoritmo puede llegar a ser más útil que el resto.

- *Tiempo de ejecución:* Indica cuánto demora cada algoritmo en conseguir resultados. Esta variable será medida en segundos.

Cotejando ambas variables en conjunto se busca analizar el comportamiento global de cada algoritmo según las características del conjunto de datos en los que se apliquen.

4.1.2.2. VARIABLES PARA LOS EXPERIMENTOS CON GRAFOS REALES

En este tipo de experimentos se utiliza una base de datos única para todos los algoritmos, compuesta por estructuras moleculares. No se varían elementos de la base de datos sino variables correspondientes a la configuración de los algoritmos. A continuación se describen las variables independientes y dependientes involucradas.

4.1.2.2.1. VARIABLES INDEPENDIENTES PARA LOS EXPERIMENTOS CON GRAFOS REALES

En estos experimentos se considera la siguiente variable independiente:

- *Minimum Support Threshold o Umbral de Frecuencia Mínima:* Indica la cantidad mínima de grafos que deben contener a una subestructura para considerarla de frecuente. En estos experimentos el valor se fija en 25% para la primera prueba y aumenta 5% por cada prueba hasta llegar a 95%.

4.1.2.2.2. VARIABLES DEPENDIENTES PARA LOS EXPERIMENTOS CON GRAFOS REALES

Se busca comparar las siguientes variables dependientes para poder evaluar el comportamiento de los algoritmos en cada experimento:

- *Cantidad de subestructuras:* Indica la cantidad de subestructuras frecuentes que los algoritmos encuentran en cada experimento.
- *Tiempo de ejecución:* Indica cuánto demora cada algoritmo en conseguir resultados. Esta variable será medida en segundos.

Cotejando ambas variables en conjunto se busca analizar el comportamiento global de cada algoritmo en un conjunto de datos reales, incrementando en cada paso el umbral mínimo.

4.2. CONSTRUCCIÓN DEL BANCO DE PRUEBAS

Para realizar los experimentos se utilizan las implementaciones de cada algoritmo proporcionadas por los autores de los mismos. Las pruebas, son ejecutadas por una capa de software en el lenguaje

Python, la cual se encarga de generar los datos, formatearlos, aplicar los algoritmos sobre ellos y extraer los resultados.

A continuación se detallará la estructura del banco de pruebas y cómo se llevarán a cabo los experimentos explicando la generación de los grafos de prueba (Sección 4.2.1), las implementaciones de los algoritmos de búsqueda de patrones en grafos que se utilizan (Sección 4.2.2), cómo se adaptaron estas implementaciones al banco de pruebas (Sección 4.2.3) y el proceso de ejecución de los experimentos (Sección 4.2.4).

4.2.1. GENERADOR DE GRAFOS

El algoritmo generador de grafos utilizado en el experimento implementa funciones provistas por la librería Networkx en su versión 1.10 para Python [Hagberg et al., 2016]. La misma facilita la creación y manipulación de cualquier tipo de grafo y contiene distintos generadores. Particularmente se utiliza la función llamada *gnm_random_graph*, descrita con pseudocódigo de manera resumida en la figura 4.1. Toda la estructura de la librería esta implementada con el paradigma orientado a objetos [Rentsch, 1982], por lo que la estructura principal de los grafos está implementada con clases.

Función: Gnm_random_graph

Entrada: n // cantidad de nodos
m // cantidad de arcos
seed // semilla para generar números aleatorios
directed // determina si el grafo va a ser dirigido o no

Salida: G // $G_{n.m}$: objeto grafo con *n* nodos y *m* arcos

```
1:  SI (directed=verdadero) ENTONCES
2:      G=grafoDirigido()
3:  SINO
4:      G=grafoNoDirigido()
5:  DESDE i=0 HASTA n
6:      G.agregarNodo(i)
7:  listaDeNodos=G.nodos()
8:  contadorArcos=0
9:  MIENTRAS (contadorArcos < m)
10:     u = elegirAleatoriamente(listaDeNodos)
11:     v = elegirAleatoriamente(listaDeNodos)
12:     SI  u <> v Y noExisteArco(u,v) ENTONCES
13:         G.agregarArco(u,v)
14:         contadorArcos=contadorArcos+1
15: RETURN G
```

Figura 4.1. *Algoritmo generador de grafos aleatorios provisto por la librería Networkx.*

Puede verse en la figura antes mencionada que se utilizan dos constructores para crear grafos, uno para dirigidos y otros para no dirigidos, así como también varias funciones propias de esas clases. Es válido remarcar que la clase de grafos no dirigidos controla que los arcos entre dos nodos determinados puedan ser creados una sola vez como máximo. Estos aspectos otorgan el nivel de abstracción adecuado para concentrarse en la integración de los algoritmos sin necesidad de desarrollar todas las clases necesarias para la manipulación de este tipo de estructuras. Utilizando como base esta librería, se implementa un algoritmo generador que es el utilizado en todos los experimentos, el cual agrega más flexibilidad permitiendo definir la lista de nodos permitidos, la lista de arcos permitidos y la cantidad máxima de nodos con etiquetas repetidas que puede tener el grafo. Esta modificación puede observarse en la figura 4.2.

Nótese que estos algoritmos se utilizan para generar solo un grafo, por lo que posteriormente se necesita crear una lista para armar los datasets que servirán para la ejecución de las pruebas.

Función: generarGrafoEtiquetado

Entrada: numberNodes // cantidad de nodos
numberEdges // cantidad de arcos
labelListNode // lista de etiquetas para los nodos
labelListEdge // lista de etiquetas para los arcos

Salida: grafo // $G_{n.m}$: objeto grafo con *n* nodos y *m* arcos

```
1:  grafo=gnm_random_graph(numberNodes,numberEdges)
2:  PARA CADA nodo EN grafo
3:      SI (largo(labelListNode)<>numberNodes) ENTONCES
4:          nodo.etiqueta=elegirAleatoriamente(labelListNode)
5:      SINO
6:          nodo.etiqueta=nodo.identificador    // Todos los nodos distintos
7:  PARA CADA arco EN grafo
8:      arco.etiqueta=elegirAleatoriamente(labelListEdge)
9:  RETURN grafo
```

Figura 4.2. *Implementación de algoritmo generador de grafos no dirigidos*

4.2.2. IMPLEMENTACIONES DE LOS ALGORITMOS DE BÚSQUEDA DE PATRONES

Las implementaciones de los algoritmos no se desarrollan desde cero sino que se utilizan aquellas desarrolladas por los autores de los mismos [Kuramochi et al., 2001, Yan et al., 2002, Huan et al., 2003, Nijssen et al., 2004]. Se considera que son los que mejor conocen de los principios y particularidades de cada algoritmo por lo que sus implementaciones deberían ser las más adecuadas para llevar a cabo las pruebas.

Cada una de las implementaciones tiene parámetros de configuración determinados y requieren de un formato particular en el archivo de entrada. En la Tabla 4.4 se resumen los parámetros especiales de cada implementación, el tipo de entrada que recibe y la salida que genera.

Si bien cada algoritmo tiene sus particularidades, hay varios aspectos que tienen en común y son desarrollados en esta sección ya que tienen influencia en el proceso:

1. Cada implementación está desarrollada en lenguaje C++.
2. De las implementaciones del algoritmo GASTON y FFSM se puede obtener el código fuente pero del gSpan y FSG no, sólo el código objeto por lo que no pueden ser modificados.
3. Todas las implementaciones funcionan mediante una interfaz con la terminal del sistema.
4. Los algoritmos se ejecutan sobre archivos formateados con todas las estructuras de los grafos a utilizar. Como salida, se generan archivos con las estructuras encontradas y estadísticas.

Algoritmo	**Parámetros**	**Entrada**	**Salida**
FFSM	*Support:* entero positivo con valores entre 0 y 100. Determina el umbral de frecuencia mínima. *Density:* densidad mínima del grafo de salida. *Sizelimit*: la cantidad mínima de nodos de los patrones. *Sizeuplimit*: la cantidad máxima de nodos de los patrones.	Dos archivos: 1. *NodeFile*: archivo con los nodos de las estructuras. 2. *EdgeFile*: archivo con los arcos de las estructuras.	Tres archivos: *OutNodeFile*: nodos de los patrones encontrados. *OutEdgeFile*: arcos de los patrones encontrados *OutFeatureFile*: estadísitcas de los subgrafos encontrados.
FSG	*Support:* entero positivo con valores entre 0 y 100. Determina el umbral de frecuencia mínima. *Minsize:* entero positivo que indica la cantidad mínima de arcos de las estructuras buscadas. *Maxsize:* entero positivo que indica la cantidad máxima de arcos de las estructuras buscadas. Maximal: Si está habilitado, busca solo subgrafos frecuentes máximos [Huan et al., 2004].	Un archivo con el conjunto de grafos.	Un archivo con extensión .fp con el conjunto de subestructuras encontradas con la frecuencia de ocurrencia de cada una.
gSpan	*Support:* entero positivo con valores entre 0 y 1. Determina el umbral de frecuencia mínima.	Un archivo con el conjunto de grafos.	Un archivo con extensión .fp con el conjunto de subestructuras encontradas con la frecuencia de ocurrencia de cada una.
GASTON	*Support:* entero positivo con valores entre 0 y 100. Determina el umbral de frecuencia mínima.	Un archivo con el conjunto de grafos.	Un archivo con el conjunto de subestructuras encontradas con la frecuencia de ocurrencia de cada una.

Tabla 4.4. *Características de las implementaciones de los algoritmos.*

Los algoritmos FSG, GASTON y gSpan, requieren de un único archivo de entrada que describe la estructura de los grafos con un formato específico, el cual se observa en la Figura 4.3. Los archivos de salida generados que contienen los subgrafos encontrados cuentan con las mismas características. Cada grafo es considerado como una transacción por lo que cada vez que se describa la estructura de alguno, se comenzará escribiendo la letra "t" seguida de un asterisco (#) y un número identificador del grafo. Luego de eso se detallan los nodos, cada uno en una fila distinta, comenzando con la letra v, seguida de un identificador y su etiqueta. Los arcos comienzan con la letra "e", seguido de dos números representando los nodos que une y finalmente un tercer número representando su peso. Como los algoritmos están pensados para nodos no dirigidos, no importa el orden de los identificadores de los nodos que se unen.

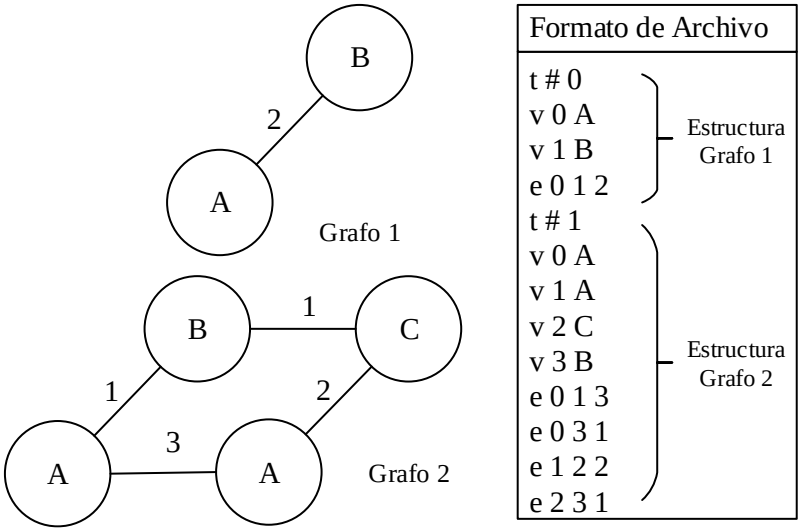


Figura 4.3. *Formato de los datasets para las implementaciones de FSG, gSpan y GASTON*

La implementación del FFSM, en cambio, necesita dos archivos: uno con los nodos y otros con los arcos, como se observa en la Figura 4.4. Los nodos son descritos con la palabra "node", seguido de un identificador del grafo al que corresponden, un identificador del nodo y su valor. Los arcos, se denotan con la palabra "edge", un identificador del grafo al que corresponde, los identificadores de los nodos que se unen y finalmente su peso. De los tres archivos de salida del FFSM, aquellos dos que describen los nodos y los arcos de las subestructuras contienen en mismo formato que los archivos de entrada. El tercero contiene una fila por cada patrón encontrado describiendo información acerca de la frecuencia de ocurrencia de los mismos. Este último no será de interés para los experimentos.

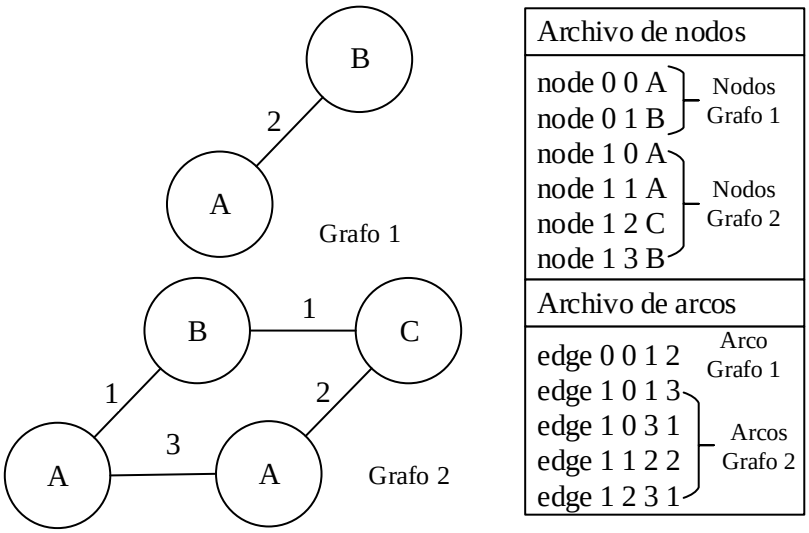


Figura 4.4. *Formato de los dataset para la implementación del algoritmo FFSM*

El dataset Compounds_422 utilizado para los experimentos con grafos reales contiene el formato de la figura 4.3. Para utilizar el algoritmo FFSM, se debe modificar su estructura para que concuerde con la descrita en la figura 4.4.

4.2.2.1. PARÁMETROS UTILIZADOS PARA LA EJECUCIÓN DE LOS ALGORITMOS

Para ejecutar los algoritmos en el caso de las pruebas con grafos sintéticos se utiliza la configuración de parámetros presente en la Tabla 4.5. Esta configuración es fija se utiliza para todos los experimentos sin importar la repetición de nodos o la variación de la densidad de la base de datos.

Algoritmo	**Parámetro**	**Valor**
FFSM	*Support*	5 (equivalente a 5%)
	Density	Default (cualquier densidad)
	Sizelimit	1
	Sizeuplimit	Default (sin límite)
FSG	*Support*	5 (equivalente a 5%)
	Minsize	1
	Maxsize	Default (sin límite)
	Maximal	0 (Deshabilitado)
gSpan	*Support*	0,5 (equivalente al 5%)
GASTON	*Support*	5 (equivalente al 5%)

Tabla 4.5. *Configuración de los parámetros de cada algoritmo para los experimentos con grafos sintéticos.*

En el caso de las pruebas con grafos reales la única variante con respecto a lo expuesto anteriormente es la variación en el valor del parámetro llamado *Support*. Como se explicó en la Sección 4.1.2, la intención de estos experimentos es usar una misma base de datos y verificar el comportamiento de los algoritmos a medida que se incrementa la frecuencia de ocurrencia mínima de las subestructuras encontradas. Por lo tanto, la configuración de los algoritmos se verá afectada a medida que se ejecuten las pruebas, como se muestra en la Tabla 4.6.

	Valor del parámetro Support en cada prueba														
Algoritmo	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
FFSM, FSG, GASTON	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95
gSpan	0,25	0,3	0,35	0,4	0,45	0,5	0,55	0,6	0,65	0,7	0,75	0,8	0,85	0,9	0,95

Tabla 4.6. *Configuración del parámetro Support de cada algoritmo para los experimentos con grafos reales.*

4.2.3. ADAPTACIÓN DE LOS ALGORITMOS AL BANCO DE PRUEBAS

Hasta ahora se muestran las implementaciones de los algoritmos a utilizar, las variables que se miden y el generador de grafos a emplear. El problema que surge es el de incorporar todos los aspectos en un solo sistema integrado. Como se mencionó en la sección anterior, los algoritmos están escritos en C++ y el banco de pruebas en Python. Además, no se cuenta con la posibilidad de acceder al código fuente de dos de los algoritmos, por lo que se descarta la opción de integrarlos directamente por código, algo que Python permite mediante el uso de librerías especiales.

De manera que se pueda resolver este conflicto, se pensó en un proceso alternativo. Para cada algoritmo, se generó una clase que funciona de adaptador entre la implementación y el banco de pruebas, para así poder ejecutar cada una sobre los grafos generados mediante el algoritmo descrito en la sección 4.4.1. Esta clase adaptadora se encargará de formatear los grafos generados en memoria RAM a un archivo compatible con el algoritmo, ejecutar el mismo con las opciones correspondientes y obtener ciertas estadísticas de los archivos resultantes. Como cada uno tiene opciones particulares y no todos poseen el mismo formato de entrada y salida, habrá cuatro adaptadores personalizados para las características particulares de cada implementación.

Un ejemplo de clase adaptadora puede observarse en la figura 4.5. En ella se considera el atributo *support*, común a todos los algoritmos, y las funciones para formatear los grafos, ejecutar el algoritmo y obtener la cantidad de subestructuras encontradas a partir de los archivos de salida que se generen. La función *ejecutar* devuelve el tiempo de ejecución del algoritmo medido en segundos.

A grandes rasgos, el procedimiento seguido para poder ejecutar un algoritmo en el banco de pruebas es el que se resume en la figura 4.6.

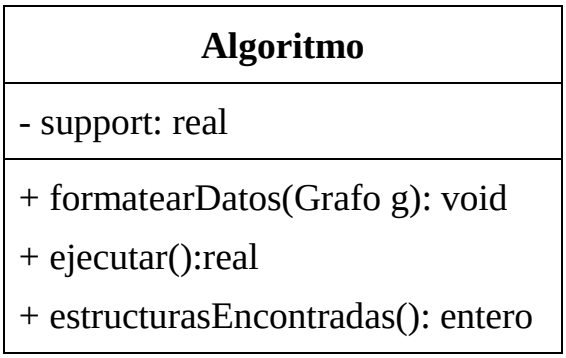


Figura 4.5. *Ejemplo de clase para adaptar un algoritmo al banco de pruebas*

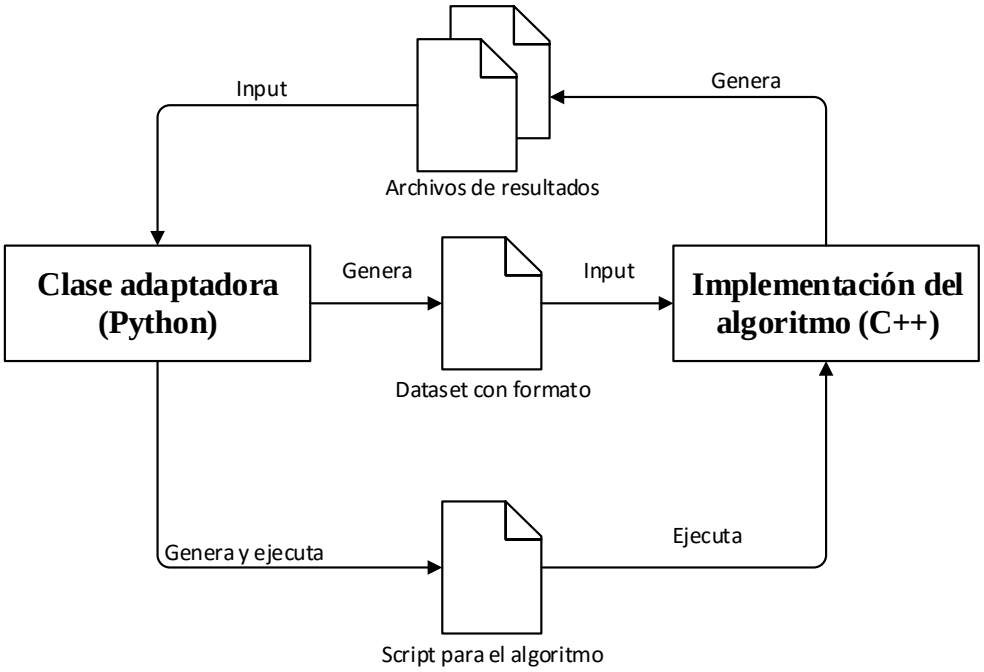


Figura 4.6. *Ejemplo de ejecución de un algoritmo para una prueba determinada*

Una vez que los datos estén formateados, la clase genera un script, el cual será el encargado de llamar al algoritmo con las opciones correspondientes. Mientras se ejecuta, se calcula el tiempo y una vez terminado, puede obtenerse la cantidad de estructuras encontradas a través del archivo resultante que se genere. De esta manera, se evitan los problemas de compatibilidad entre los lenguajes Python y C++, producidos al no tener posibilidad de modificar el código fuente de dos de las implementaciones.

4.2.4. PROCESO DE EJECUCIÓN DE LOS EXPERIMENTOS

Para poder utilizar los adaptadores y llevar a cabo las pruebas, primeramente se requiere configurar los parámetros experimentales siguiendo los pasos descritos a continuación:

1. Inicializar las variables para un tipo de experimento. Recapitulando, los tipos de experimentos pueden clasificarse según sus datasets en:
    a. Grafos sintéticos con incremento fijo sin repetición de nodos.
    b. Grafos sintéticos con incremento fijo con repetición de nodos.
    c. Grafos sintéticos con incremento variable sin repetición de nodos.
    d. Grafos sintéticos con incremento variable con repetición de nodos.
    e. Grafos reales.
2. Inicializar las clases de cada algoritmo con las variables asociadas a los atributos de cada uno.
3. Inicializar el banco de pruebas con las variables definidas en el punto 1.
4. Comenzar los experimentos.

Todos estos pasos se repiten por cada prueba dentro del experimento, modificando las variables definidas en el punto 1, de acuerdo a lo descrito en la sección de variables (4.3).

Una vez establecida la configuración inicial, la clase del banco de pruebas será la encargada de ejecutar cada algoritmo y extraer los resultados, llevando a cabo las siguientes actividades:

1. Generar una lista de grafos con los parámetros iniciales acordes al tipo de experimento que se esté evaluando.
2. Probar cada algoritmo con los grafos generados. Esto incluye:
    a. Transformar el conjunto de grafos en archivos compatibles con las implementaciones.
    b. Ejecutar las implementaciones sobre los archivos generados.
    c. Extraer los datos de los archivos generados y medir el tiempo de ejecución.
3. Volcar los resultados parciales en un archivo para control posterior.
4. Ejecutar los pasos del 1 al 3 cien veces, almacenando los resultados parciales.
5. Realizar un promedio de los resultados obtenidos.
6. Volcar los resultados totales en un archivo final.
7. Repetir los pasos del 1 al 6, modificando los parámetros del paso 1 según el tipo de experimento que se esté realizando.

Para una mejor compresión, se proporciona una descripción gráfica resumida de las actividades que se llevan a cabo para la ejecución de un algoritmo para el caso de los experimentos con grafos sintéticos. La misma puede verse en la figura 4.7. En ella solo se detalla la ejecución completa de una

sola prueba para un algoritmo. Este proceso se repite por cada algoritmo cien veces y luego las variables son reajustadas en el programa principal para la siguiente prueba. Cabe recordar que en un experimento se desarrollan varias pruebas y en cada una de ellas se varía la cantidad de nodos y arcos de los datasets.

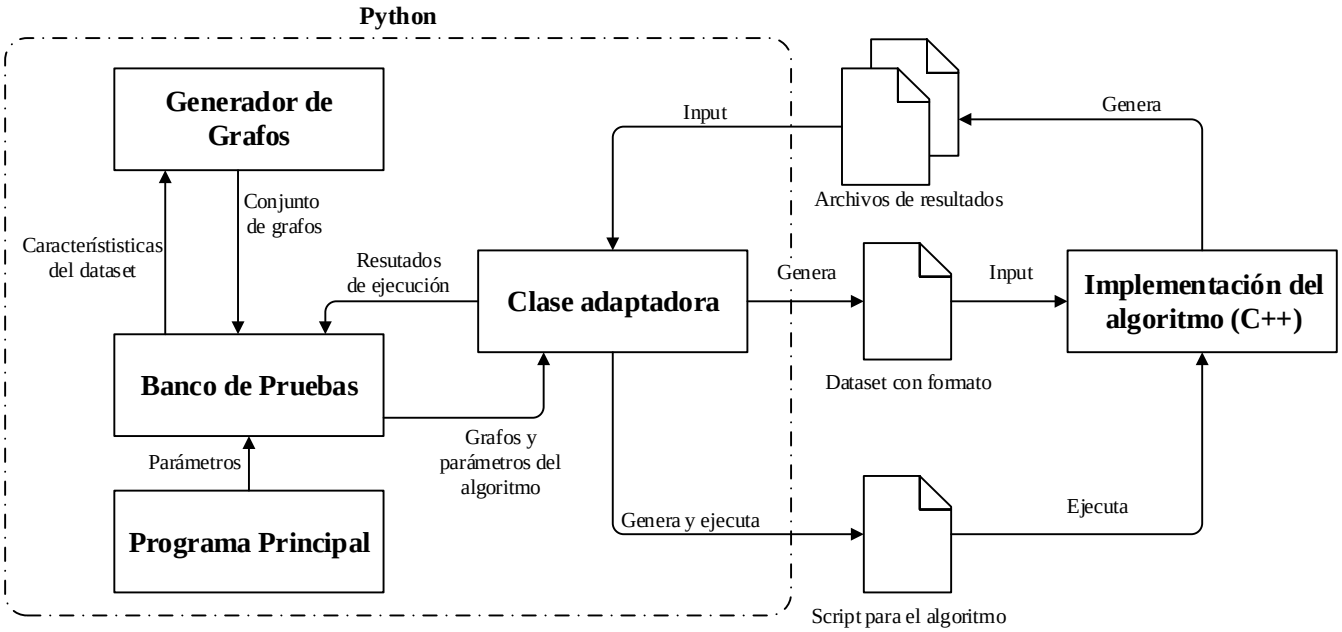


Figura 4.7. *Ejecución de una prueba para experimentos con grafos reales*

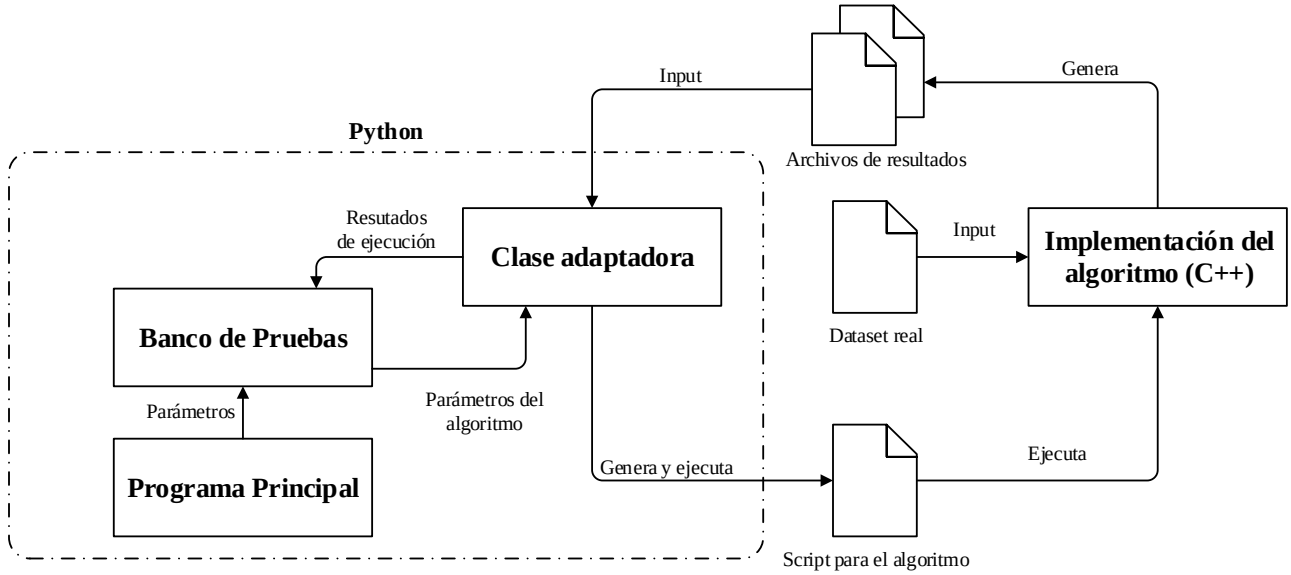


Figura 4.8. *Ejecución de una prueba para experimentos con grafos sintéticos*

En el caso del experimento con grafos reales, el procedimiento es idéntico pero sin la etapa de generación de grafos. Como el dataset ya tiene el formato adecuado para los algoritmos, tampoco es necesario que la clase adaptadora modifique el archivo. Para la implementación de FFSM, como el

formato de entrada es distinto, se creó una copia de la base de datos compatible antes de la ejecución de las pruebas. El proceso resultante es de la figura 4.8.

En ambos casos, el banco de pruebas es el encargado de recopilar las estadísticas de ejecución y elaborar los archivos de resultados finales usados para realizar la comparación del comportamiento de los algoritmos.

Todos los experimentos descritos en la presente sección serán llevados en una misma computadora con las siguientes características:

- *Modelo del procesador:* Intel Core i7 – 4790
- *Frecuencia del procesador:* 3.6 GHz
- *Memoria RAM:* 16 GB
- *Sistema Operativo:* Ubuntu 14.04 64 bits
- *Versión de Python:* 2.7
- *Versión de Networkx:* 1.10

4.3. EVALUACIÓN Y COMPARACIÓN DE RESULTADOS

Posteriormente al proceso de ejecución de las pruebas se continúa con el proceso de evaluación y comparación de resultados. En el mismo se analizan los datos recopilados por el banco de pruebas y se comparan experimento a experimento para determinar si hay algún algoritmo que supere al resto en cualquier escenario. En caso de que no exista, se busca identificar para qué tipo de grafos es más recomendable utilizar cada algoritmo.

Este proceso consta de dos evaluaciones, una considerando la cantidad de estructuras que se obtienen y otra tomando en cuenta el tiempo de ejecución de cada uno. Se considera útil analizar ambos factores por los siguientes motivos:

- Si los algoritmos encuentran un número similar de subgrafos, es interesante analizar cuáles lo hicieron con más velocidad.
- Si se observa a algún algoritmo más veloz que el resto, es necesario también determinar cuántas subestructuras se encontraron. Esto se debe a que si el algoritmo no puede encontrar resultados, algo que no es deseable, su tiempo de resolución es mucho menor ya que se eliminan varios pasos de su ejecución.

Si bien en un proceso de explotación de información completo se pueden llegar a descartar subgrafos encontrados por ser irrelevantes, a lo largo de toda la experimentación se considera que a mayor cantidad de estructuras encontradas mejor.

En la Tabla 4.8 puede verse un ejemplo de cómo se vería una tabla de resultados para un determinado tipo de experimento. En base al análisis de estos datos se compara el comportamiento de cada algoritmo en cada escenario diseñado.

Exp.	GASTON		gSpan		FFSM		FSG	
#	Estructuras	Tiempo (s)	Estructuras	Tiempo (s)	Estructuras	Tiempo (s)	Estructuras	Tiempo (s)
1	118.01	0.0026677	128.01	0.0154917	117.99	0.00618119	118.01	0.00727388
2	156.07	0.0031443	166.07	0.0202955	155.27	0.00810114	156.07	0.01024559
3	341.13	0.0041274	351.13	0.0329506	334.5	0.0128529	341.13	0.02219
4	970.72	0.0070094	980.72	0.072728	944.7	0.027273	970.72	0.10862
5	2311.31	0.0147529	2321.31	0.163199	2233.19	0.0589984	2311.31	0.491165

Tabla 4.8. *Ejemplo de resultados a recopilar en cada experimento*

5. EXPERIMENTACIÓN

En este capítulo se presenta el desarrollo de la experimentación en base a los parámetros establecidos en la sección anterior. Se comienza describiendo de forma general como se llevan a cabo las pruebas (Sección 5.1) y luego se muestran los resultados de las pruebas con grafos sintéticos (Sección 5.2), los grafos sintéticos con repetición de nodos (Sección 5.3) y finalmente la las pruebas con grafos reales (Sección 5.4).

5.1. RESUMEN DEL PROCESO DE EJECUCIÓN DE LAS PRUEBAS

Como se detalló en las secciones anteriores, la metodología para ejecutar las pruebas con grafos sintéticos es la siguiente: primero se generan los grafos aleatoriamente según los criterios correspondientes que se ven en las secciones 5.2 y 5.3. Para cada configuración nodos-arcos, se generan cien grafos aleatorios y se ejecutan los algoritmos sobre ellos, de manera que los algoritmos se analicen sobre el mismo conjunto de datos. Este proceso se repite cien veces y luego se calculan los resultados promedios para cada tipo de prueba.

En el caso de los grafos reales, se utilizan las mismas bases en cien pruebas distintas y luego se calcula el promedio total. Lo que varía en estos casos es el mínimum support para los algoritmos siguiendo el criterio que se explica en la sección 5.4.

5.2. GRAFOS SINTÉTICOS

Para analizar el comportamiento de los algoritmos la primera prueba a realizar es sobre una base de datos compuesta por grafos sintéticos, es decir, grafos generados aleatoriamente que no tienen correspondencia con algún modelo real. Para estos experimentos, los grafos tienen todas las etiquetas de los nodos distintas, es decir, todos los nodos son distintos para los algoritmos. Con este tipo de modelo se intenta simular entornos en los que los elementos involucrados sean únicos para cada proceso, como puede ser una red de comunicación entre empleados de una compañía. En esa red, cada nodo representaría a un empleado distinto.

Esta etapa de pruebas se realiza variando la densidad de cien grafos experimento a experimento, de acuerdo con la Tabla 5.1, y analizando la cantidad de subestructuras encontradas en contraste con el tiempo de ejecución. El umbral mínimo de cada algoritmo permanecerá fijo y con un valor de 5%. Para simular que cada nodo es único, todas las etiquetas de los grafos serán distintas y serán numeradas de acuerdo al id de cada vértice. Por ejemplo, en formato gSpan las primeras líneas del archivo de base de datos quedarían conformadas de la siguiente manera: *t # 0; v 0 0; v 1 1; v 2 2;*

v 3 3; v 4 4; v 5 5, siendo *t # n* el comienzo de la descripción de un grafo *v n m* la descripción de un vértices con *id=n* y *etiqueta=m*.

Con esto se intentará descubrir si existe algún patrón en el comportamiento de los algoritmos que esté directamente relacionado con la densidad del conjunto de datos a analizar y, contrastándolo con las otras pruebas, se buscará determinar qué tan influyente es el hecho de tener o no nodos repetidos en los grafos.

Experimento	Nodos	Arcos
1	10	10
2	10	15
3	10	20
4	10	25
5	10	30
6	20	20
7	20	25
8	20	30
9	20	35
10	20	40
11	30	30
12	30	35
13	30	40
14	30	45
15	30	50
16	40	40
17	40	45
18	40	50
19	40	55
20	40	60
21	50	50
22	50	55
23	50	60
24	50	65
25	50	70

Experimento	Nodos	Arcos
26	60	60
27	60	65
28	60	70
29	60	75
30	60	80
31	70	70
32	70	75
33	70	80
34	70	85
35	70	90
36	80	80
37	80	85
38	80	90
39	80	95
40	80	100
41	90	90
42	90	95
43	90	100
44	90	105
45	90	110
46	100	100
47	100	105
48	100	110
49	100	115
50	100	120

Tabla 5.1. *Configuración de cada experimento para las primeras pruebas con grafos sintéticos sin repetición de nodos.*

En este caso, todos los nodos tienen etiquetas distintas por lo que se los considera componentes diferentes dentro del modelo. Los arcos están etiquetados de manera aleatoria con valores entre uno y tres. Estos números son elegidos considerando los distintos tipos de relaciones que pueden existir entre dos nodos: comunicación dirigida, doblemente dirigida o sin dirección, así como también enlace entre átomos simple, doble o triple.

Como se puede apreciar en la tabla, al llegar a los últimos experimentos, la relación nodos/arcos no es tan notable como en los primeros. Esto se cree que podría tener influencia en la cantidad de estructuras encontradas por los algoritmos, por lo que también se lleva a cabo otro experimento con la configuración mostrada en la Tabla 5.2. Puede verse que la relación es mucho más constante para tratar de averiguar si es verdaderamente influyente en los resultados.

Experimento	Nodos	Arcos
1	10	10
2	10	15
3	10	20
4	10	25
5	10	30
6	20	20
7	20	30
8	20	40
9	20	50
10	20	60
11	30	30
12	30	45
13	30	60
14	30	75
15	30	90
16	40	40
17	40	60
18	40	80
19	40	100
20	40	120
21	50	50
22	50	75
23	50	100
24	50	125
25	50	150

Experimento	Nodos	Arcos
26	60	60
27	60	90
28	60	120
29	60	150
30	60	180
31	70	70
32	70	105
33	70	140
34	70	175
35	70	210
36	80	80
37	80	120
38	80	160
39	80	200
40	80	240
41	90	90
42	90	135
43	90	180
44	90	225
45	90	270
46	100	100
47	100	150
48	100	200
49	100	250
50	100	300

Tabla 5.2. *Configuración de cada experimento para la segunda etapa pruebas con grafos sintéticos sin repetición de nodos.*

Los resultados correspondientes a las pruebas con un incremento fijo en la cantidad de nodos pueden apreciarse en la tabla 5.3, mientras que los resultados de las pruebas con incremento variable se resumen en la tabla 5.4. En las mismas se comparan la cantidad de subestructuras promedio encontradas por cada algoritmo en los experimentos correspondientes así como también el tiempo de ejecución promedio medido en segundos. Para una mejor comprensión en las figuras 5.1(a-b) se grafican los resultados de la tablas 5.3(a-b).

Incremento Fijo								
Exp	GASTON		gSpan		FFSM		FSG	
#	Estructuras	Tiempo (s)	Estructuras	Tiempo (s)	Estructuras	Tiempo (s)	Estructuras	Tiempo (s)
1	118.01	0.0026677	128.01	0.0154917	117.99	0.00618119	118.01	0.00727388
2	156.07	0.0031443	166.07	0.0202955	155.27	0.00810114	156.07	0.01024559
3	341.13	0.0041274	351.13	0.0329506	334.5	0.0128529	341.13	0.02219
4	970.72	0.0070094	980.72	0.072728	944.7	0.027273	970.72	0.10862
5	2311.31	0.0147529	2321.31	0.163199	2233.19	0.0589984	2311.31	0.491165
6	231.31	0.0036959	176.32	0.024729	156.32	0.00835114	156.32	0.0135649
7	306.29	0.004465	275.21	0.035177	255.21	0.0106426	255.21	0.0162178
8	398.25	0.0064374	367.76	0.045916	347.75	0.0133199	347.76	0.0189857
9	383.0	0.028334	444.08	0.0547421	424.08	0.0159533	424.08	0.0221175
10	291.43	0.052093	501.96	0.061315	481.9	0.0182277	481.96	0.025386
11	293.22	0.004962	135.64	0.026762	105.64	0.00837443	105.64	0.0197861
12	362.73	0.0062656	201.8	0.035078	171.8	0.00996654	171.8	0.0222067
13	419.04	0.017254	283.83	0.044571	253.83	0.0122355	253.83	0.0251366
14	309.98	0.04642	372.71	0.053754	342.71	0.0142783	342.71	0.0278304
15	133.7	0.066297	470.67	0.065059	440.67	0.0167706	440.67	0.030926
16	332.59	0.0066106	107.35	0.0299069	67.35	0.00920414	67.35	0.02671636
17	388.31	0.010869	143.35	0.03519	103.35	0.0102648	103.35	0.029174
18	292.77	0.037830	189.64	0.041794	149.64	0.0116693	149.64	0.0317654
19	158.81	0.060544	243.61	0.0500167	203.61	0.0130190	203.61	0.0346818
20	38.6	0.070886	307.64	0.058395	267.63	0.0145079	267.64	0.0375496
21	368.19	0.0080723	93.93	0.0342057	43.93	0.0102349	43.93	0.034407
22	405.71	0.015263	115.24	0.038037	65.24	0.0108867	65.24	0.0367717
23	298.47	0.039458	139.31	0.040402	89.31	0.0112801	89.31	0.0392633
24	143.42	0.062981	169.6	0.046897	119.6	0.0126631	119.6	0.04203099
25	93.4	0.069039	208.34	0.054988	158.34	0.014013	158.34	0.0451004
26	398.85	0.009696	90.73	0.040086	30.73	0.0113612	30.73	0.0434565
27	391.25	0.02413	101.68	0.043268	41.68	0.011996	41.68	0.0458109
28	274.69	0.046392	117.04	0.046546	57.04	0.0127371	57.04	0.0486443
29	105.71	0.066023	133.51	0.05029	73.51	0.013464	73.51	0.05118489
30	50.5	0.071581	155.56	0.05527	95.56	0.0143332	95.56	0.054263
31	435.02	0.012143	92.05	0.045562	22.05	0.0128718	22.05	0.0533188
32	379.56	0.02918	99.23	0.049039	29.23	0.0134045	29.23	0.0560795
33	251.81	0.051992	106.88	0.050536	36.88	0.0140388	36.88	0.058846
34	174.14	0.063815	118.65	0.053038	48.65	0.01466307	48.65	0.0623565
35	74.55	0.072192	130.16	0.058215	60.16	0.0154029	60.16	0.0653039
36	451.81	0.016296	95.29	0.052863	15.29	0.0146406	15.29	0.0647739
37	413.01	0.032745	100.38	0.055061	20.38	0.01509808	20.38	0.0677194
38	304.17	0.050704	106.23	0.056891	26.23	0.0156445	26.23	0.070456
39	177.01	0.064769	113.11	0.06067	33.11	0.0162837	33.11	0.0732472
40	49.05	0.07313	120.18	0.062808	40.18	0.0168435	40.18	0.0759465
41	494.58	0.017836	101.52	0.0592119	0.0	0.0847117	11.52	0.0765025

Tabla 5.3.a *Resultados de los experimentos para las primeras pruebas con grafos sintéticos sin repetición de nodos*

Incremento Fijo								
Exp	GASTON		gSpan		FFSM		FSG	
#	Estructuras	Tiempo (s)	Estructuras	Tiempo (s)	Estructuras	Tiempo (s)	Estructuras	Tiempo (s)
42	409.99	0.035178	105.36	0.0616012	0.0	0.0843574	15.36	0.0793945
43	222.43	0.057303	108.67	0.0642488	0.0	0.0850042	18.67	0.0826094
44	155.22	0.065272	113.09	0.067522	0.0	0.0851599	23.09	0.0853111
45	59.83	0.072065	118.86	0.069542	0.0	0.085254	28.86	0.088176
46	462.27	0.024825	108.81	0.065224	0.0	0.0854097	8.81	0.0903184
47	373.3	0.0412181	110.92	0.067409	0.0	0.0855402	10.92	0.0930588
48	259.21	0.0565975	114.04	0.07041	0.0	0.085972	14.04	0.09587859
49	141.12	0.0670919	116.62	0.071919	0.0	0.0864788	16.62	0.09881939
50	114.43	0.0709258	119.97	0.0752025	0.0	0.0872213	19.97	0.10213557

Tabla 5.3.b (Continuación) *Resultados de los experimentos para las primeras pruebas con grafos sintéticos sin repetición de nodos*

Incremento Variable								
Exp	GASTON		gSpan		FFSM		FSG	
#	Estructuras	Tiempo (s)	Estructuras	Tiempo (s)	Estructuras	Tiempo (s)	Estructuras	Tiempo (s)
1	118.01	0.002828	128.01	0.0168706	117.99	0.0065327	118.01	0.0076662
2	155.28	0.003127	165.28	0.0201657	154.55	0.0080126	155.28	0.0104978
3	343.53	0.004143	353.53	0.0332659	337.11	0.0132752	343.53	0.0232644
4	971.03	0.007111	981.03	0.0744607	946.54	0.0279587	971.03	0.1137017
5	2312.33	0.015257	2322.33	0.1688643	2234.35	0.0610397	2312.33	0.5161214
6	228.13	0.003551	176.64	0.0246931	156.64	0.0081273	156.64	0.0137059
7	397.67	0.009304	368.14	0.045863	348.14	0.0132683	348.14	0.019291
8	256.67	0.054982	502.88	0.0616203	482.77	0.0182896	482.88	0.0257761
9	20.27	0.073056	589.46	0.07335108	568.81	0.0229213	569.46	0.0337676
10	0.0	0.073893	735.66	0.0892089	712.93	0.0294368	715.66	0.0461099
11	293.49	0.004939	136.78	0.0265909	106.78	0.0086484	106.78	0.0205332
12	224.43	0.054185	372.46	0.0536277	342.46	0.0144696	342.46	0.0286905
13	0.0	0.073599	668.53	0.08966207	638.53	0.0225222	638.53	0.0388691
14	0.0	0.074303	929.09	0.1280997	899.05	0.0320865	899.09	0.0518794
15	0.0	0.0738807	1122.6	0.1602091	1092.44	0.041276	1092.6	0.0671696
16	332.62	0.006281	107.54	0.0294428	67.54	0.0091306	67.54	0.0271679
17	115.07	0.068836	308.33	0.0573247	268.33	0.014514	268.33	0.0382358
18	0.0	0.073873	641.05	0.1001251	601.05	0.0229065	601.05	0.0522844
19	0.0	0.0742107	1033.58	0.1562126	993.58	0.0344607	993.58	0.0706009
20	0.0	0.074829	1418.43	0.2227642	1378.41	0.0483443	1378.43	0.0944189
21	368.01	0.007917	94.08	0.0344853	44.08	0.0100371	44.08	0.0349277
22	42.81	0.073765	251.11	0.0611014	201.11	0.0151206	201.11	0.049162
23	0.0	0.075565	554.47	0.1062329	504.47	0.0231052	504.47	0.0682596
24	0.0	0.076406	981.52	0.172027	931.52	0.0348958	931.52	0.093656
25	0.0	0.076419	1472.17	0.2570195	1422.15	0.0501237	1422.17	0.1253132
26	401.54	0.009607	90.19	0.0388392	30.19	0.011392	30.19	0.0444463

Tabla 5.4.a *Resultados de los experimentos para la segunda etapa pruebas con grafos sintéticos sin repetición de nodos.*

Incremento Variable								
Exp	GASTON		gSpan		FFSM		FSG	
#	Estructuras	Tiempo (s)	Estructuras	Tiempo (s)	Estructuras	Tiempo (s)	Estructuras	Tiempo (s)
27	0.0	0.074907	207.17	0.0643119	147.17	0.016165	147.17	0.0621375
28	0.0	0.075432	463.71	0.1087269	403.71	0.023141	403.71	0.0853783
29	0.0	0.076021	870.18	0.1769249	810.18	0.0355678	810.18	0.1172985
30	0.0	0.077531	1397.01	0.2713874	1337.01	0.0500787	1337.01	0.1635236
31	435.72	0.012224	91.45	0.0457149	21.45	0.0131956	21.45	0.0555776
32	0.0	0.076194	183.29	0.071125	113.29	0.0179483	113.29	0.0772958
33	0.0	0.0738802	397.07	0.1105283	327.07	0.0238643	327.07	0.102275
34	0.0	0.074856	755.81	0.1743424	685.81	0.0332968	685.81	0.1398108
35	0.0	0.074883	1265.76	0.2677131	1195.76	0.0473143	1195.76	0.190863
36	452.94	0.016303	96.14	0.0507428	16.14	0.0140642	16.14	0.0641877
37	29.73	0.074302	166.55	0.0754978	86.55	0.0186283	86.55	0.0885846
38	0.0	0.0745504	341.55	0.1161438	261.55	0.0251444	261.55	0.1221379
39	0.0	0.075221	656.12	0.1800742	576.12	0.0343379	576.12	0.1685983
40	0.0	0.076022	1129.5	0.2746367	1049.5	0.0472849	1049.5	0.2323901
41	469.28	0.020052	101.88	0.0572447	0.0	0.0832241	11.88	0.0762451
42	0.0	0.074342	156.65	0.0820347	0.0	0.0870432	66.65	0.1050771
43	0.0	0.075191	303.6	0.1218183	0.0	0.0908016	213.6	0.1447607
44	0.0	0.076229	572.91	0.1825641	0.0	0.0945918	482.91	0.199542
45	0.0	0.078436	997.14	0.2758583	0.0	0.0996231	907.14	0.277026
46	459.11	0.0250205	109.07	0.0644254	0.0	0.0854061	9.07	0.08979902
47	0.0	0.074927	153.19	0.0904417	0.0	0.09116508	53.19	0.1223359
48	0.0	0.076039	272.8	0.1274588	0.0	0.0934677	172.8	0.1682874
49	0.0	0.078305	505.78	0.1879032	0.0	0.0982916	405.78	0.2329529
50	0.0	0.0783904	878.85	0.2774244	0.0	0.1069429	778.85	0.3231319

Tabla 5.4.b (Continuación) *Resultados de los experimentos para la segunda etapa pruebas con grafos sintéticos sin repetición de nodos.*

En la figura 5.1.a, se muestra la cantidad de subestructuras encontradas en cada experimento, mientras que en la figura 5.1.b se observa el tiempo de ejecución de cada algoritmo en cada prueba. Contrastando ambas figuras 5.1.a y 5.1.b, se ve como todos los algoritmos tienen el mismo comportamiento para los primeros experimentos en los que la cantidad de nodos y arcos es relativamente baja. Todos encuentran la misma cantidad de estructuras aunque hay bastante diferencia en los tiempos de ejecución de cada uno. Por ejemplo, en la prueba 5, donde hay treinta arcos y diez nodos, la cantidad de estructuras encontradas difiere por muy poco, mientras que es notable la diferencia en el tiempo de ejecución. Allí puede verse como el algoritmo GASTON supera a todos mientras que el algoritmo FSG es el más lento. Sin embargo, a medida que avanzan las pruebas, el comportamiento difiere. El algoritmo GASTON tiene picos en cuanto estructuras encontradas en las pruebas con una cantidad de nodos y arcos similares. Lo que también puede

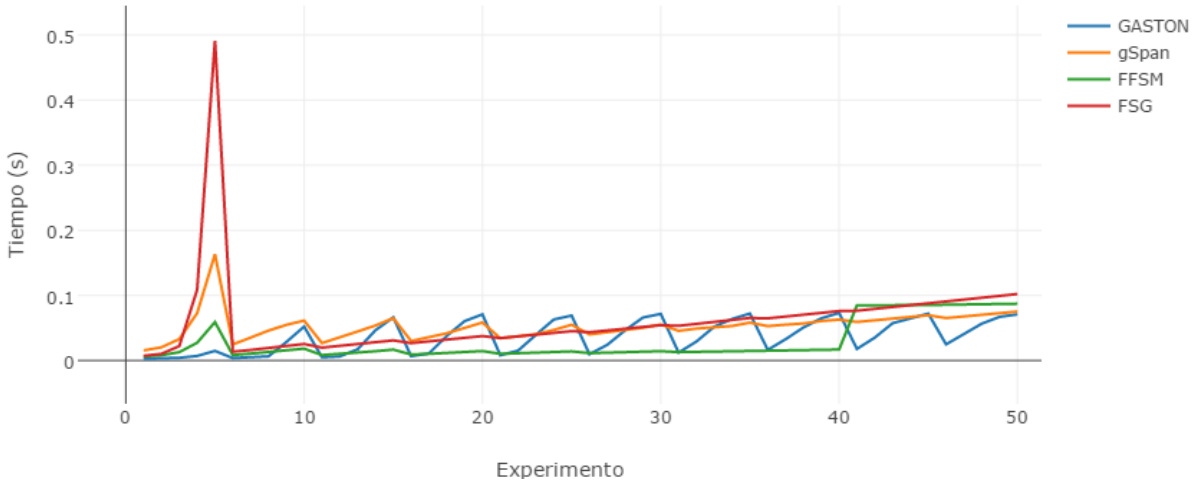


Figura 5.1.a. *Tiempo de ejecución en segundos de los algoritmos en cada experimento para las primeras pruebas con grafos sintéticos sin repetición de nodos*

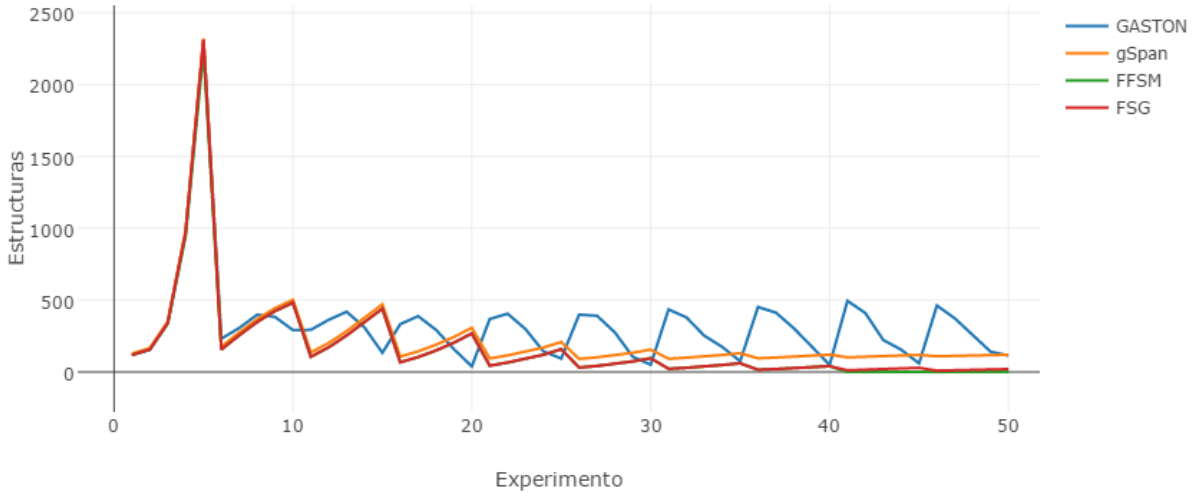


Figura 5.1.b. *Cantidad de estructuras encontradas en cada experimento para las primeras pruebas con grafos sintéticos sin repetición de nodos*

observarse es que este algoritmo es más estable en cuanto a su comportamiento, tanto en tiempo de ejecución como resultados. Los otros algoritmos, en cambio, al llegar a las pruebas más complejas tardan más en ejecutarse y encuentran menos estructuras.

Ahora bien, ¿los algoritmos se comportan igual si aumenta la cantidad de arcos en relación con los nodos? Para responder a esa pregunta se ejecutan las pruebas que se denominan con “incremento variable” debido a que el incremento en el número de arcos que se realiza de prueba en prueba depende de la cantidad de nodos de la prueba anterior, y no es un número fijo, tal como se observa en la tabla 5.2.

En base las figuras 5.2.a y 5.2.b, a primera vista se observa una diferencia en el comportamiento de todos los algoritmos a partir de la prueba cinco. Esto es debido a que hasta ese experimento la cantidad de nodos y arcos es igual a la de la prueba anterior. En este caso puede verse como en algoritmo GASTON es el que tiene más dificultades para encontrar resultados. En varias pruebas, el algoritmo dio como resultado cero estructuras encontradas debido a que nunca llegó a ejecutarse

completamente. En cuanto a los otros tres algoritmos, su comportamiento difiere en el tiempo de ejecución más que en la cantidad de estructuras encontradas. Los algoritmos gSpan y FSG son los que más tiempo tardan. Puede verse que los picos se dan en las pruebas en las que los arcos triplican en número a la cantidad de nodos. La tendencia hasta la prueba 40 indica que el algoritmo FFSM es más rápido y tiene resultados similares al gSpan y FSG. Sin embargo, a partir de esa prueba, comienza a aumentar su tiempo de ejecución y la cantidad de estructuras que encuentra es igual a cero. Esto es debido a que nunca llega a encontrar subestructuras. Por lo tanto, para grafos de gran tamaño con gran conectividad entre sus nodos, los algoritmos más eficientes son el gSpan y FSG, siempre teniendo en cuenta que todos los nodos son distintos en esta prueba.

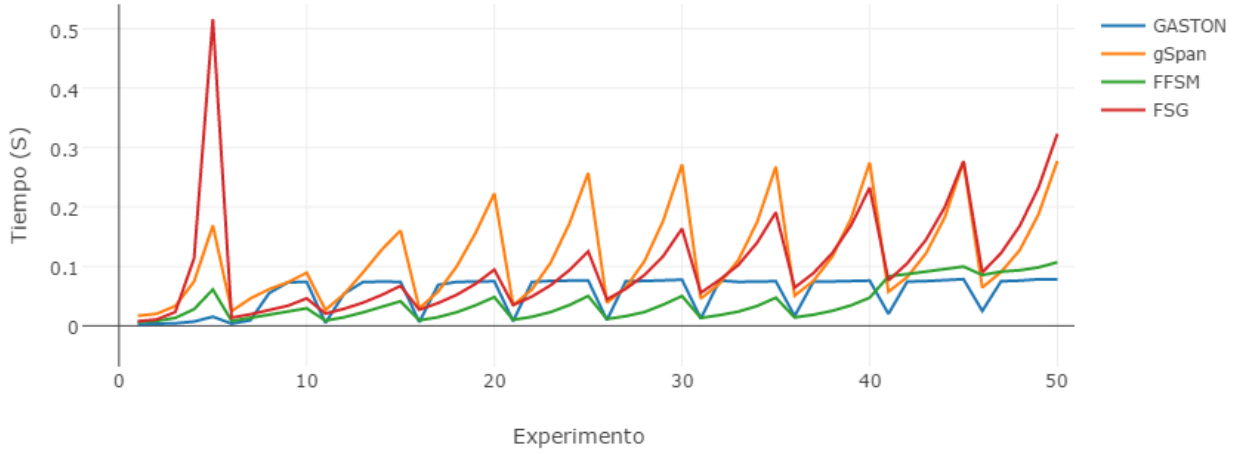


Figura 5.2.a. *Tiempo de ejecución en segundos de los algoritmos en cada experimento para las segundas pruebas con grafos sintéticos sin repetición de nodos*

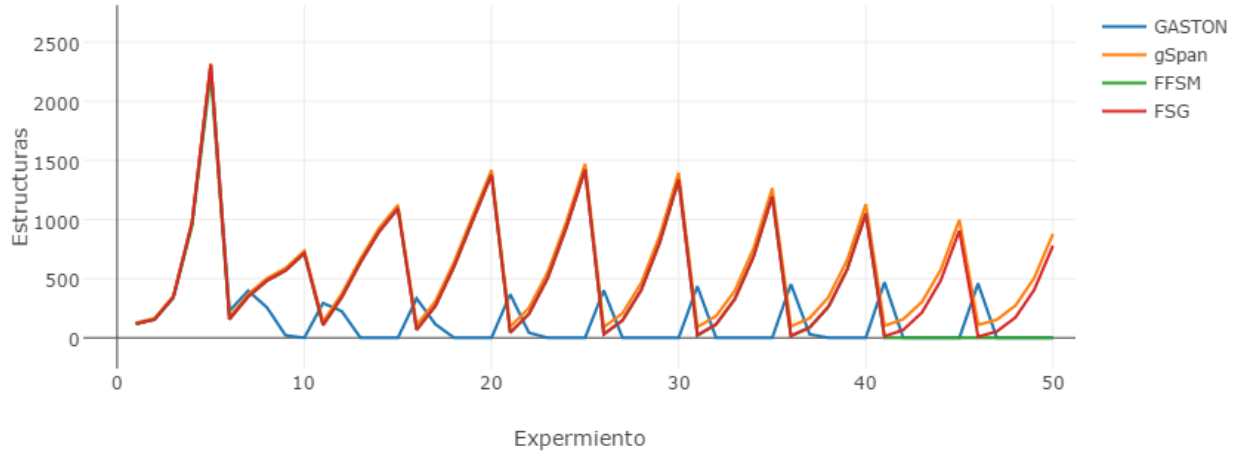


Figura 5.2.b. *Cantidad de subestructuras encontradas en cada experimento para las segundas con grafos sintéticos sin repetición de nodos*

5.3. GRAFOS SINTÉTICOS CON REPETICIÓN DE NODOS

En las pruebas anteriores se llevaron a cabo experimentaciones sobre grafos sintéticos sin repeticiones de nodos. Sin embargo, hay situaciones o modelos en los cuales es frecuente tener elementos con similares características o repetidos dentro de una misma estructura. Un claro ejemplo de esto son las estructuras moleculares. En ellas, varios átomos se relacionan entre sí por medios de enlaces, lo cual puede ser representado mediante nodos con etiquetas iguales para átomos iguales y arcos con diferentes etiquetas para los tipos de enlace. Concretamente, la molécula del agua, H_2O, puede ser representada en un grafo con tres nodos y dos arcos. Dos de ellos nodos tendrán la etiqueta “H” y uno de ellos la etiqueta “O”, como puede verse en la figura 5.3.

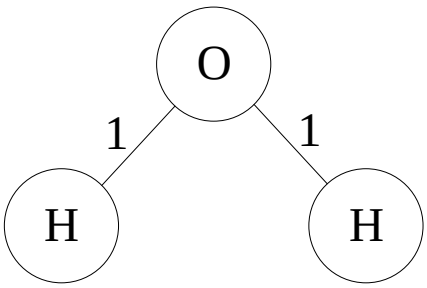


Figura 5.3. *Ejemplo de construcción de una molécula de agua con estructura de grafo.*

Con el objetivo de comprobar el funcionamiento de los algoritmos en este tipo de grafos, se llevaron a cabo experimentos con datos sintéticos en los cuales hay elementos repetidos distribuidos aleatoriamente. Al igual que en las experimentaciones anteriores, se crearon aleatoriamente cien grafos sobre los que se ejecutaron todos los algoritmos para encontrar patrones entre ellos. Este procedimiento se realizó cien veces y se promediaron los resultados para la comparación.

Comprobando que los algoritmos se comportan de manera distinta dependiendo de la densidad de los grafos a utilizar, como se aprecia en la sección anterior, se realizan nuevamente dos tipos de pruebas con distintas configuraciones para verificar que suceda lo mismo utilizando estos tipos de grafos. Las configuraciones utilizadas para las pruebas pueden ser visualizadas en las Tablas 5.5 y 5.6. La tabla 5.5 representa a las pruebas con variación de arcos fija y las 5.6 a las pruebas con incremento de cantidad de arcos variable, dependiendo de la cantidad de nodos que haya. Puede observarse que la única diferencia con las Tablas 5.1 y 5.2 de la sección anterior es una columna llamada nodos únicos. Este valor representa la cantidad máxima de valore distintos de nodos que puede tener un grafo. Por ejemplo, prueba 1 de la tabla 5.5 tiene diez nodos, 10 arcos y 5 nodos únicos. Esto quiere decir que habrá cinco valores posibles que pueden tomar esos diez nodos, por lo que habrá valores repetidos dentro de las estructuras.

Experimento	Nodos	Arcos	Nodos Únicos
1	10	10	5
2	10	15	5
3	10	20	5
4	10	25	5
5	10	30	5
6	20	20	10
7	20	25	10
8	20	30	10
9	20	35	10
10	20	40	10
11	30	30	15
12	30	35	15
13	30	40	15
14	30	45	15
15	30	50	15
16	40	40	20
17	40	45	20
18	40	50	20
19	40	55	20
20	40	60	20
21	50	50	25
22	50	55	25
23	50	60	25
24	50	65	25
25	50	70	25

Experimento	Nodos	Arcos	Nodos Únicos
26	60	60	30
27	60	65	30
28	60	70	30
29	60	75	30
30	60	80	30
31	70	70	35
32	70	75	35
33	70	80	35
34	70	85	35
35	70	90	35
36	80	80	40
37	80	85	40
38	80	90	40
39	80	95	40
40	80	100	40
41	90	90	45
42	90	95	45
43	90	100	45
44	90	105	45
45	90	110	45
46	100	100	50
47	100	105	50
48	100	110	50
49	100	115	50
50	100	120	50

Tabla 5.5. *Configuración de cada experimento para las primeras pruebas con grafos sintéticos con repetición de nodos.*

Los resultados correspondientes a las pruebas con un incremento fijo en la cantidad de nodos pueden apreciarse en la tabla 5.7, mientras que los resultados de las pruebas con incremento variable se resumen en la tabla 5.8. En las mismas se comparan la cantidad de subestructuras promedio encontradas por cada algoritmo en los experimentos correspondientes así como también el tiempo de ejecución promedio medido en segundos. Nuevamente observando ambas tablas 5.7 y 5.8 se observa un cambio en el comportamiento de los algoritmos al ejecutarse el segundo grupo de experimentos, en los cuales la cantidad de arcos en relación a la cantidad de nodos es mayor que en las primeras pruebas, por lo que la densidad del grafo puede tomarse como un factor determinante a la hora de definir la eficiencia de un algoritmo.

De estos resultados también puede concluirse que el hecho de tener o no varios nodos con etiquetas repetidas en influyente tanto en la capacidad de los algoritmos para encontrar subestructuras como en su tiempo de ejecución.

Experimento	Nodos	Arcos	Nodos Únicos
1	10	10	5
2	10	15	5
3	10	20	5
4	10	25	5
5	10	30	5
6	20	20	10
7	20	30	10
8	20	40	10
9	20	50	10
10	20	60	10
11	30	30	15
12	30	45	15
13	30	60	15
14	30	75	15
15	30	90	15
16	40	40	20
17	40	60	20
18	40	80	20
19	40	100	20
20	40	120	20
21	50	50	25
22	50	75	25
23	50	100	25
24	50	125	25
25	50	150	25

Experimento	Nodos	Arcos	Nodos Únicos
26	60	60	30
27	60	90	30
28	60	120	30
29	60	150	30
30	60	180	30
31	70	70	35
32	70	105	35
33	70	140	35
34	70	175	35
35	70	210	35
36	80	80	40
37	80	120	40
38	80	160	40
39	80	200	40
40	80	240	40
41	90	90	45
42	90	135	45
43	90	180	45
44	90	225	45
45	90	270	45
46	100	100	50
47	100	150	50
48	100	200	50
49	100	250	50
50	100	300	50

Tabla 5.6. *Configuración de cada experimento para las segundas pruebas con grafos sintéticos con repetición de nodos.*

Incremento Fijo								
Exp	GASTON		gSpan		FFSM		FSG	
#	Estructuras	Tiempo (s)	Estructuras	Tiempo (s)	Estructuras	Tiempo (s)	Estructuras	Tiempo (s)
1	105.21	0.002584	110.21	0.0132604	101.9	0.0062926	105.21	0.0110314
2	410.35	0.0042985	415.35	0.033652	394.11	0.0137993	410.35	0.0591868
3	998.89	0.0081578	1003.89	0.076189	926.27	0.0299012	998.89	0.14483
4	3607.04	0.02121	3612.04	0.251735	3217.78	0.0945022	3607.04	0.6668216
5	11923.29	0.0694702	11928.43	0.852513	10375.01	0.3038171	11923.43	7.2314162
6	160.53	0.0037901	170.53	0.021864	160.43	0.0094904	160.53	0.0137952
7	189.35	0.004257	199.35	0.0256209	188.46	0.0111954	189.35	0.0168608
8	275.3	0.0047981	285.3	0.033141	271.64	0.0145874	275.3	0.0228346
9	477.51	0.0057785	487.51	0.0477891	467.86	0.0207524	477.51	0.0405711
10	851.09	0.0076154	861.09	0.073923	832.41	0.0307351	851.09	0.0924641
11	273.36	0.004818	324.1	0.041179	309.09	0.0140662	309.1	0.0207287

Tabla 5.7.a *Resultados de los experimentos para las primeras pruebas con grafos sintéticos con repetición de nodos*

Incremento Fijo								
Exp	GASTON		gSpan		FFSM		FSG	
#	Estructuras	Tiempo (s)	Estructuras	Tiempo (s)	Estructuras	Tiempo (s)	Estructuras	Tiempo (s)
12	308.5	0.0061846	346.25	0.0446266	331.2	0.0160252	331.25	0.0231704
13	356.74	0.0062549	360.67	0.0484455	345.55	0.0171834	345.67	0.0258248
14	393.94	0.0095133	378.19	0.0508929	362.81	0.0188261	363.19	0.0289914
15	413.46	0.018696	408.78	0.0549864	392.67	0.0210271	393.78	0.0320991
16	286.23	0.0209885	465.45	0.0571789	445.45	0.0181659	445.45	0.0270719
17	205.64	0.0419965	516.32	0.0644412	496.32	0.0202589	496.32	0.0297483
18	136.42	0.056971	551.55	0.0705012	531.54	0.0222215	531.55	0.0324935
19	93.5	0.0651911	578.24	0.0750739	558.21	0.0236425	558.24	0.0352418
20	38.61	0.0696559	599.77	0.0791986	579.69	0.0254535	579.77	0.0380877
21	71.38	0.0613893	553.85	0.07106	528.85	0.0209767	528.85	0.0338658
22	35.86	0.0677863	628.94	0.0813611	603.94	0.0234416	603.94	0.0364407
23	7.56	0.0719048	695.23	0.0908525	670.23	0.02562	670.23	0.0391896
24	8.03	0.0715823	751.51	0.0990569	726.51	0.0279386	726.51	0.0420316
25	0.0	0.0728913	799.46	0.1074723	774.46	0.0303871	774.46	0.0451856
26	4.67	0.0719043	590.52	0.0809029	560.52	0.0228102	560.52	0.0407569
27	0.0	0.0719412	677.51	0.0924357	647.51	0.0249827	647.51	0.043144
28	0.0	0.0727318	762.64	0.1043449	732.64	0.0280739	732.64	0.0462739
29	0.0	0.0726487	840.12	0.1153829	810.12	0.0303315	810.12	0.0490237
30	0.0	0.0723227	915.32	0.1278489	885.32	0.0329698	885.32	0.0518022
31	5.58	0.0721774	586.39	0.0880134	551.39	0.0241577	551.39	0.0477082
32	0.0	0.0724271	675.11	0.0990888	640.11	0.0261722	640.11	0.0502738
33	0.0	0.072284	768.15	0.1119533	733.15	0.0290275	733.15	0.0529758
34	0.0	0.0733248	858.11	0.1237046	823.11	0.031702	823.11	0.0565245
35	0.0	0.0736592	943.06	0.137176	908.06	0.0342105	908.06	0.0595081
36	0.0	0.0736744	563.76	0.0910974	523.76	0.0248148	523.76	0.054965
37	0.0	0.0736032	650.0	0.1041745	610.0	0.027524	610.0	0.0576311
38	0.0	0.0734686	739.55	0.1143254	699.55	0.0295779	699.55	0.0604374
39	0.0	0.0733649	824.8	0.1270427	784.8	0.0318071	784.8	0.0633014
40	0.0	0.0733196	917.37	0.1407752	877.37	0.0346018	877.37	0.0662494
41	0.0	0.0734437	528.93	0.0954239	483.93	0.0259182	483.93	0.0619265
42	0.0	0.0734752	609.49	0.1053826	564.49	0.0274881	564.49	0.0643492
43	0.0	0.073822	691.59	0.1178699	646.59	0.0297866	646.59	0.0675133
44	0.0	0.0735777	772.19	0.1300329	727.19	0.032313	727.19	0.0709395
45	0.0	0.0740406	859.4	0.1425162	814.4	0.0340041	814.4	0.0738078
46	0.0	0.0734875	492.66	0.097752	442.66	0.0263895	442.66	0.0689091
47	0.0	0.07280303	558.35	0.1071831	508.35	0.0279258	508.35	0.0713558
48	0.0	0.0753546	634.33	0.1215202	584.33	0.0306993	584.33	0.0759814
49	0.0	0.0737942	707.23	0.1294454	657.23	0.0321022	657.23	0.0779554
50	0.0	0.0745483	790.05	0.1427061	740.05	0.0346992	740.05	0.0815442

Tabla 5.7.b *Resultados de los experimentos para las primeras pruebas con grafos sintéticos con repetición de nodos*

Incremento Variable								
Exp	GASTON		gSpan		FFSM		FSG	
#	Estructuras	Tiempo (s)	Estructuras	Tiempo (s)	Estructuras	Tiempo (s)	Estructuras	Tiempo (s)
1	106.62	0.0026233	111.62	0.0130081	102.99	0.0066065	106.62	0.0106761
2	408.01	0.0043295	413.01	0.0347071	391.34	0.0149126	408.01	0.0589455
3	996.56	0.00809	1001.56	0.0768627	924.67	0.0316427	996.56	0.1448134
4	3612.95	0.0213087	3617.96	0.2562375	3231.8	0.1056946	3612.96	0.6694181
5	11900.52	0.0693066	11905.68	0.8501545	10390.0	0.3276350	11900.68	7.2804480
6	160.43	0.0037106	170.43	0.0218384	160.28	0.0095065	160.43	0.0138326
7	275.21	0.0048204	285.21	0.033362	271.86	0.0148959	275.21	0.0228538
8	845.67	0.0077349	855.67	0.0734566	826.12	0.0322127	845.67	0.0916493
9	2100.82	0.0162928	2110.82	0.1673675	2048.13	0.0675929	2100.82	0.5187642
10	3983.62	0.0334508	3993.62	0.3217294	3866.84	0.1264707	3983.62	2.0011749
11	276.56	0.0048405	324.17	0.0402753	309.17	0.0143655	309.17	0.0203531
12	390.5	0.0103344	378.43	0.0506106	363.19	0.0189909	363.43	0.028577
13	564.71	0.0298549	553.59	0.0678877	534.59	0.0286810	538.59	0.0437577
14	190.98	0.0681415	1234.9	0.1200839	1201.74	0.0517711	1219.9	0.1123372
15	306.33	0.0763984	2892.54	0.2522507	2830.96	0.1090459	2877.54	0.5099213
16	301.33	0.0192268	466.74	0.0578561	446.74	0.0186730	446.74	0.0273825
17	52.8	0.0690764	600.09	0.0796088	580.02	0.0262094	580.09	0.0384001
18	0.0	0.0746342	677.48	0.0936988	656.88	0.0349476	657.48	0.0521381
19	0.0	0.0734016	920.48	0.1198327	896.19	0.04801	900.48	0.0787258
20	0.0	0.0730847	1718.55	0.1959726	1681.47	0.0787706	1698.55	0.1516595
21	57.61	0.0648832	555.8	0.071361	530.8	0.0221545	530.8	0.0340788
22	0.0	0.073757	836.03	0.1135122	810.98	0.0338676	811.03	0.0483068
23	0.0	0.0730669	952.77	0.1371097	927.62	0.0434246	927.77	0.0678700
24	0.0	0.0755052	1067.96	0.1581353	1041.83	0.0556465	1042.96	0.0892684
25	0.0	0.074325	1403.15	0.2018926	1372.4	0.0754912	1378.15	0.1278667
26	4.97	0.0726349	589.1	0.0805786	559.09	0.0236885	559.1	0.0416304
27	0.0	0.073628	1039.54	0.1470639	1009.54	0.039536	1009.54	0.061078
28	0.0	0.0737699	1271.1	0.1911096	1241.02	0.0538102	1241.1	0.0803926
29	0.0	0.073971	1392.45	0.2250374	1361.97	0.0661019	1362.45	0.1091242
30	0.0	0.0748016	1556.97	0.2515138	1525.11	0.083571	1526.97	0.1550881
31	0.0	0.0756255	587.2	0.08707	552.2	0.0250038	552.2	0.0479985
32	0.0	0.0752202	1185.89	0.1745819	1150.88	0.0439964	1150.89	0.0702161
33	0.0	0.0741508	1576.85	0.2567089	1541.83	0.0624151	1541.85	0.0967938
34	0.0	0.0746345	1781.84	0.3000367	1746.7	0.0795899	1746.84	0.13906
35	0.0	0.0779486	1921.07	0.3361404	1885.54	0.0971204	1886.07	0.1784911
36	0.0	0.0741671	563.85	0.0918093	523.85	0.0261279	523.85	0.0545607
37	0.0	0.0746845	1272.54	0.1973256	1232.54	0.0487690	1232.54	0.0810597
38	0.0	0.0745061	1843.36	0.3052733	1803.35	0.0711114	1803.36	0.1174609

Tabla 5.8.a *Resultados de los experimentos para la segunda etapa pruebas con grafos sintéticos con repetición de nodos.*

Incremento Variable								
Exp	GASTON		gSpan		FFSM		FSG	
#	Estructuras	Tiempo (s)	Estructuras	Tiempo (s)	Estructuras	Tiempo (s)	Estructuras	Tiempo (s)
39	0.0	0.0773364	2178.24	0.3855851	2138.21	0.0918129	2138.24	0.1557247
40	0.0	0.0767765	2368.82	0.4431846	2328.51	0.1252921	2328.82	0.2130179
41	0.0	0.0743281	529.08	0.0973031	484.08	0.0268911	484.08	0.0624383
42	0.0	0.0754703	1311.7	0.2205812	1266.7	0.049884	1266.7	0.0923277
43	0.0	0.080931	2041.43	0.3543764	1996.42	0.0772829	1996.43	0.1315079
44	0.0	0.0762985	2546.08	0.4749623	2501.07	0.1128506	2501.08	0.1805177
45	0.0	0.0765971	2846.53	0.5692364	2801.43	0.1268665	2801.53	0.2485174
46	0.0	0.0738171	493.87	0.0979481	443.87	0.0266801	443.87	0.0694436
47	0.0	0.0745536	1312.97	0.227998	1262.97	0.0488396	1262.97	0.1024672
48	0.0	0.0758099	2185.6	0.3961061	2135.6	0.0775498	2135.6	0.1456705
49	0.0	0.0765639	2867.88	0.5589798	2817.87	0.1099366	2817.88	0.2110763
50	0.0	0.0771157	3312.26	0.6928587	3262.21	0.1396543	3262.26	0.2864628

Tabla 5.8.b *Resultados de los experimentos para la segunda etapa pruebas con grafos sintéticos con repetición de nodos.*

Para facilitar la compresión de los resultados reflejados en la Tabla 5.7 se presentan las figuras 5.4.a, 5.4.b y 5.4.c. En la figura 5.4.a se observa que el comportamiento de los algoritmos es similar al de las otras pruebas para las primeras cinco pruebas. Por eso se analiza más en detalle lo que ocurre del experimento seis en adelante. Esto puede verse en las figuras 5.4.b y 5.4.c.

De estos resultados puede observarse que el algoritmo GASTON funciona de manera muy eficiente hasta la prueba 25, cuya cantidad de nodos es de cincuenta y arcos setenta, donde deja de encontrar resultados y su tiempo de ejecución incrementa. También, puede verse que los otros algoritmos, el gSpan, FSG y FFSG, se comportan de manera muy similar, obteniendo casi los mismos resultados en cuanto a cantidad de estructuras encontradas. La diferencia se encuentra en el tiempo de

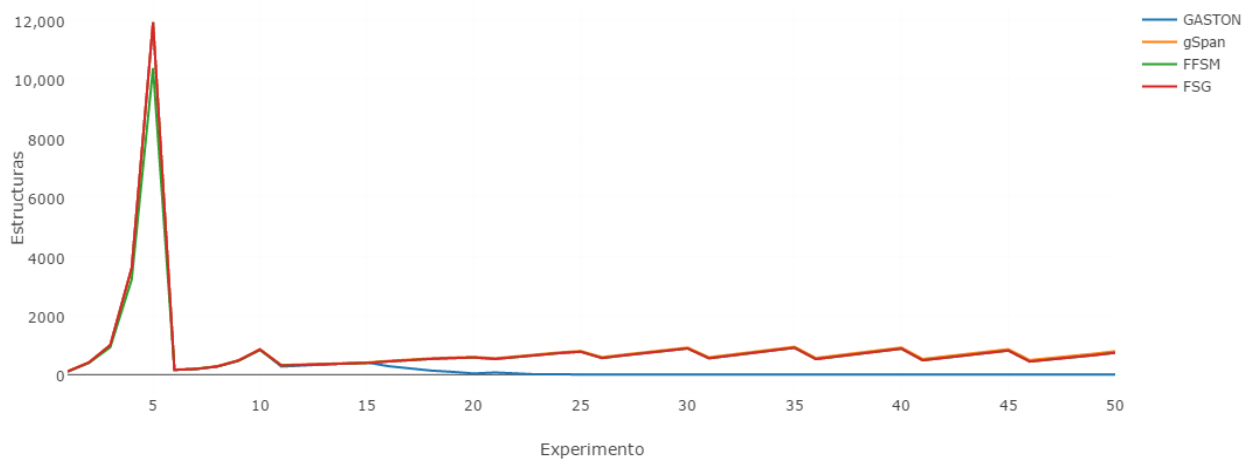


Figura 5.4.a. *Cantidad de estructuras encontradas en cada experimento para las primeras pruebas con grafos sintéticos con repetición de nodos*

ejecución. El algoritmo FFSM es el más rápido y le siguen el FSG y gSpan respectivamente. En estos tres casos, el tiempo de ejecución crece a medida que se aumentan la cantidad de nodos y arcos prueba a prueba. No ocurre lo mismo con la cantidad de estructuras encontradas, ya que desde la prueba 35 a la 50 se observa una disminución progresiva. Los picos de estructuras encontradas coinciden con los picos en los que los algoritmos demoran más, los cuales representan los momentos en que la diferencia entre cantidad de arcos y la cantidad de nodos es mayor.

Ahora resta analizar los resultados correspondientes a los experimentos con incremento variable en la cantidad de arcos. La hipótesis es que, al igual que con los casos de prueba anteriores, habrá un cambio en el comportamiento de los algoritmos a medida que se incremente el tamaño de los grafos. Los resultados correspondientes a este análisis son graficados en las figuras 5.5.a, 5.5.b y 5.5.c. La figura 5.5.a incluye la cantidad de subestructuras halladas en todos los experimentos. Las figuras 5.4.b y 5.4.c muestran el tiempo de ejecución de cada algoritmo. La primera contiene los resultados de todos los experimentos mientras que la última comienza en el experimento seis para

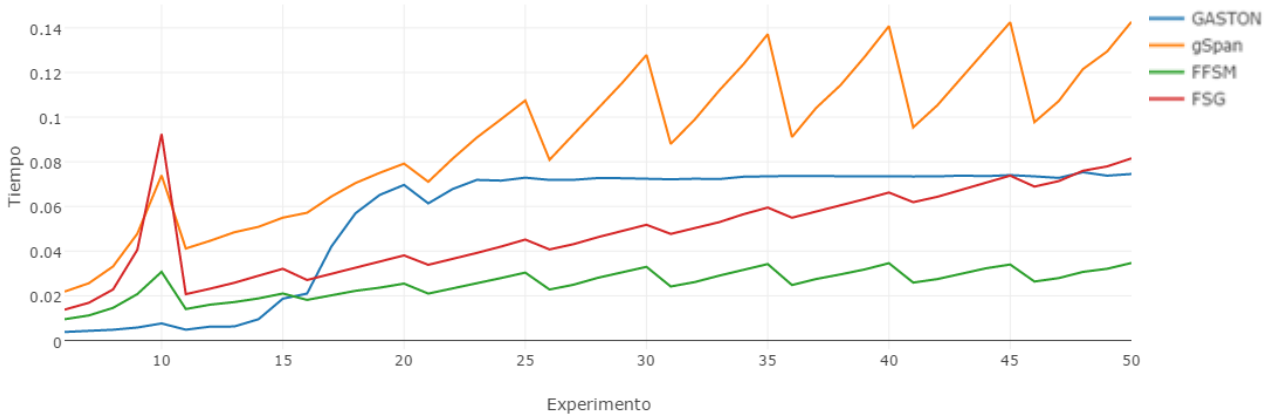


Figura 5.4.b. *Tiempo de ejecución de cada algoritmo a partir del experimento seis para las primeras pruebas con grafos sintéticos con repetición de nodos*

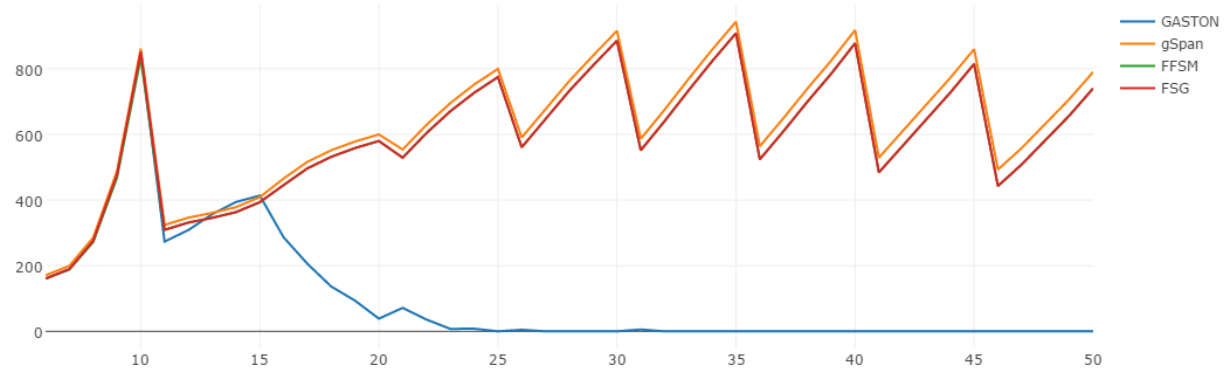


Figura 5.4.c. *Cantidad de estructuras encontradas a partir del experimento seis para las primeras pruebas con grafos sintéticos con repetición de nodos*

poder observar más claramente las diferencias entre los valores.

En cuanto a las estructuras encontradas, el comportamiento es muy similar al de la prueba anterior. Exceptuando al algoritmo GASTON, los tres restantes encuentran una cantidad parecida, la cual incrementa en relación al número de nodos y arcos. Los picos se hayan en las pruebas en las que la cantidad de arcos es tres veces mayor a los nodos.

Al igual que en la prueba anterior, la diferencia principal está en el tiempo de ejecución. El algoritmo GASTON siempre es el más rápido, aunque su problema es que no puede encontrar resultados en las pruebas con grafos grandes. Sus resultados son óptimos en la medida que las estructuras de los grafos a analizar no sean demasiado grandes. El FFSM, en cambio, es un poco más lento que el GASTON, pero siempre encuentra resultados en un número similar al del resto. Le siguen en velocidad el FSG y el gSpan. Llamativamente, el FSG demora mucho más que el resto en las primeras pruebas. Puede verse que el pico de tiempo es de siete segundos, mientras que el resto

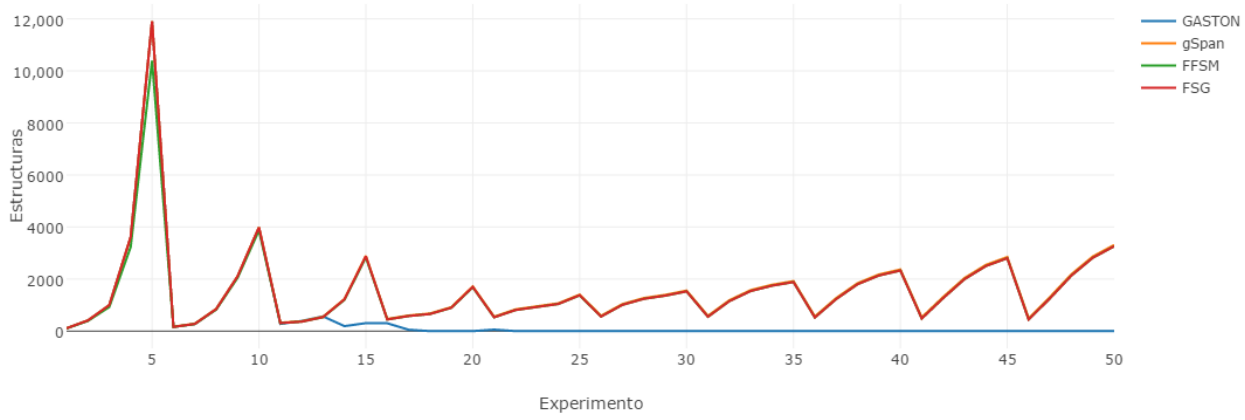


Figura 5.5.a. *Subestructuras encontradas en cada experimento para las segundas pruebas con grafos sintéticos con repetición de nodos*

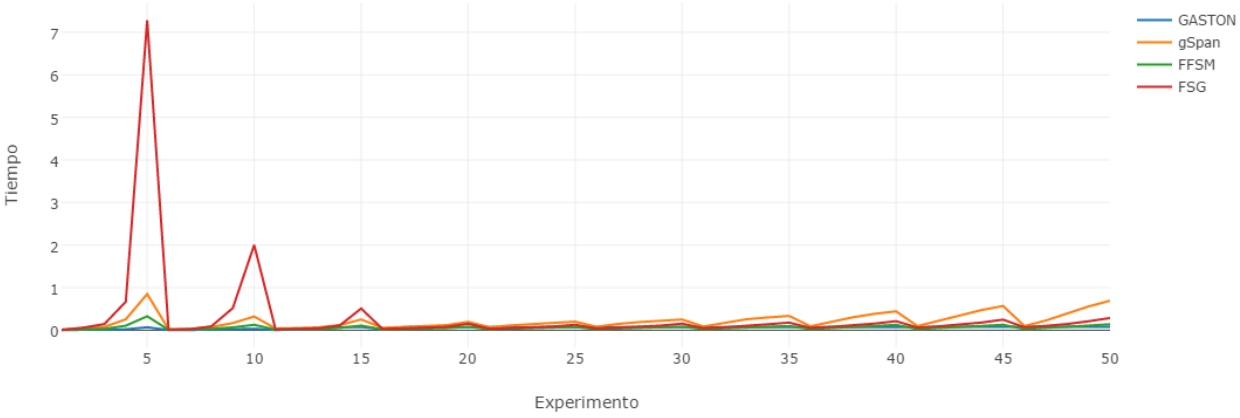


Figura 5.5.b. *Tiempo de ejecución de cada algoritmo en cada experimento para las segundas pruebas con grafos sintéticos con repetición de nodos*

nunca supera a un segundo en ninguna prueba. Sin embargo, a partir del experimento quince, empieza a disminuir su tiempo de ejecución y termina colocándose por debajo del gSpan, solamente superado en velocidad por el FFSM. (El algoritmo GASTON no se tiene en cuenta para esta comparación ya que no encuentra resultados en esa configuración).

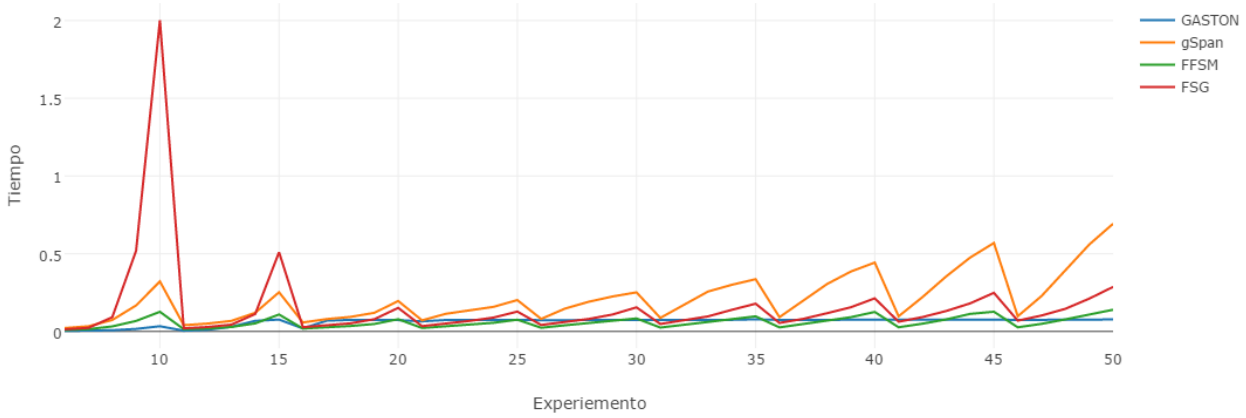


Figura 5.5.c. *Tiempo de ejecución de cada algoritmo a partir del experimento seis para las segundas pruebas con grafos sintéticos con repetición de nodos*

5.4. GRAFOS REALES

En las pruebas anteriores se probaron a los algoritmos en diversos escenarios, intentando cubrir todas las posibles estructuras en las que podrían ser utilizados. Esto se realizó creando grafos aleatoriamente, es decir, grafos que no tienen correspondencia con algún modelo real. Ahora, lo que falta analizar es ver cómo se comportan con un conjunto de datos reales.

Se utiliza el archivo Compounds_422, el cual contiene 422 estructuras de moléculas, o sea, 422 grafos. Este archivo se obtiene de los datos de prueba que se distribuyen junto con la implementación del algoritmo gSpan.

El procedimiento para llevar a cabo estas pruebas difiere con las anteriores debido a que se utiliza solamente un conjunto de datos para todas las pruebas, por lo que el número de arcos, nodos y nodos únicos permanecerá fijo. Lo que se varía de experimento en experimento es el mínimum support, o umbral mínimo. Este número establece la frecuencia de ocurrencia mínima de una subestructura para que sea considerada en los resultados finales. Es decir, si el support es del 5% como en las pruebas anteriores, todos aquellos subgrafos que se hallen por encima de ese número serán considerados subgrafos frecuentes. Ahora, en este caso se comienzan los experimentos ejecutando los algoritmos con un support del 25% y se incrementa de a 5%, como se muestra en la Tabla 5.9.

Experimento	Umbral mínimo (%)
1	25
2	30
3	35
4	40
5	45
6	50
7	55
8	60
9	65
10	70
11	75
12	80
13	85
14	90
15	95

Tabla 5.9. *Configuración de los algoritmos para cada experimento con grafos reales.*

Exp.	GASTON		gSpan		FFSM		FSG	
#	Estructuras	Tiempo (s)	Estructuras	Tiempo (s)	Estructuras	Tiempo (s)	Estructuras	Tiempo (s)
1	2126.0	0.087919	248.0	0.436558	0.0	0.029255	242.0	0.490535
2	1605.0	0.067691	124.0	0.242035	0.0	0.029305	119.0	0.329687
3	1076.0	0.060158	69.0	0.197633	0.0	0.031728	63.0	0.266289
4	895.0	0.056387	60.0	0.190134	0.0	0.029545	56.0	0.254437
5	773.0	0.051898	39.0	0.163599	0.0	0.029712	35.0	0.213378
6	714.0	0.04683	32.0	0.152285	0.0	0.029868	29.0	0.193697
7	558.0	0.042712	23.0	0.13365	0.0	0.029446	20.0	0.171579
8	500.0	0.03988	19.0	0.13092	0.0	0.029379	16.0	0.162837
9	440.0	0.036938	16.0	0.126804	0.0	0.029232	13.0	0.152154
10	393.0	0.034976	15.0	0.121881	0.0	0.029765	12.0	0.146947
11	342.0	0.030263	9.0	0.07186	0.0	0.029112	6.0	0.126082
12	326.0	0.029399	7.0	0.06958	0.0	0.02901	4.0	0.113604
13	246.0	0.02755	6.0	0.068744	0.0	0.028821	3.0	0.11331
14	213.0	0.026795	3.0	0.057347	0.0	0.029134	1.0	0.102313
15	183.0	0.02605	2.0	0.053847	0.0	0.030246	0.0	0.09644

Tabla 5.10. *Resultados de la ejecución de los algoritmos para cada experimento con grafos reales.*

Estas pruebas se ejecutan cien veces cada una y los resultados se promedian. Los números finales pueden verse en la Tabla 5.10. Todos los valores del algoritmo FFSM están en cero porque nunca terminó de ejecutarse el algoritmo. Puede deberse a la cantidad de grafos del conjunto de datos debido a que en las pruebas anteriores no tuvo inconvenientes. Se descarta que sea un problema

relacionado con el umbral mínimo debido a que se probó con valores más chicos pero siempre se obtuvieron los mismos resultados.

En las figuras 5.6.a y 5.6.b pueden verse gráficamente los resultados de la Tabla 5.10, en base a la cantidad de estructuras encontradas y tiempo de ejecución respectivamente. Ocurre algo esperable y es que la cantidad de estructuras encontradas disminuye a medida que se aumenta el umbral mínimo. Lo mismo ocurre con el tiempo de ejecución. Esto sucede debido a que es cada vez más difícil encontrar estructuras que se repitan con tanta frecuencia.

En estas pruebas ocurre algo que podía preverse observando los experimentos con grafos sintéticos: el algoritmo GASTON tiene un desempeño en cuanto a estructuras encontradas y tiempo de ejecución muy superior al resto (sin tener en cuenta al FFSM ya que no encontró resultados). Esto es debido a que la cantidad de nodos y arcos de los grafos del archivo no son muy altas por lo que estaríamos situados entre los experimentos 1 y 10 de las configuraciones anteriores, antes del

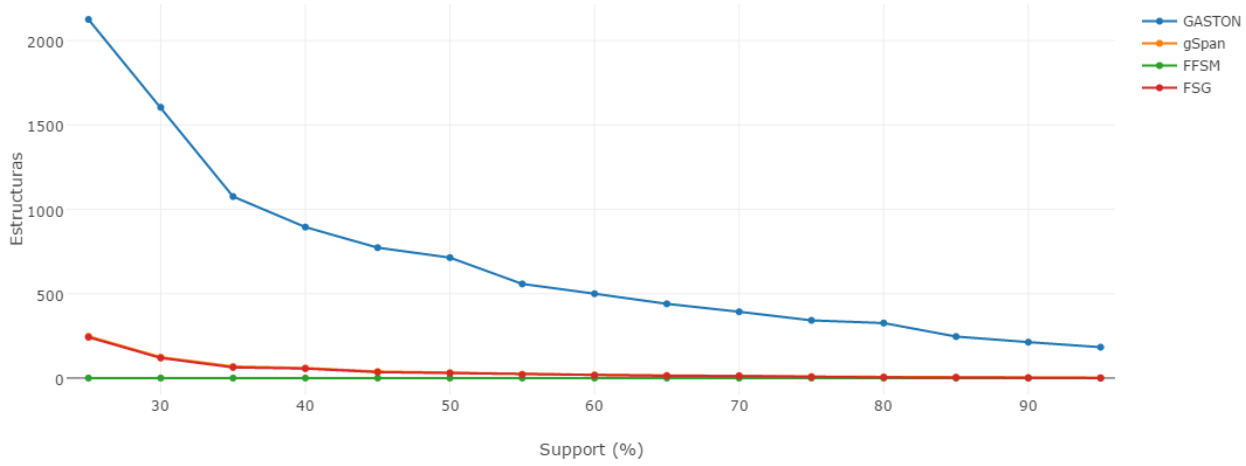


Figura 5.6.a. *Subestructuras encontradas por cada algoritmo en base al umbral mínimo para las pruebas con grafos reales*

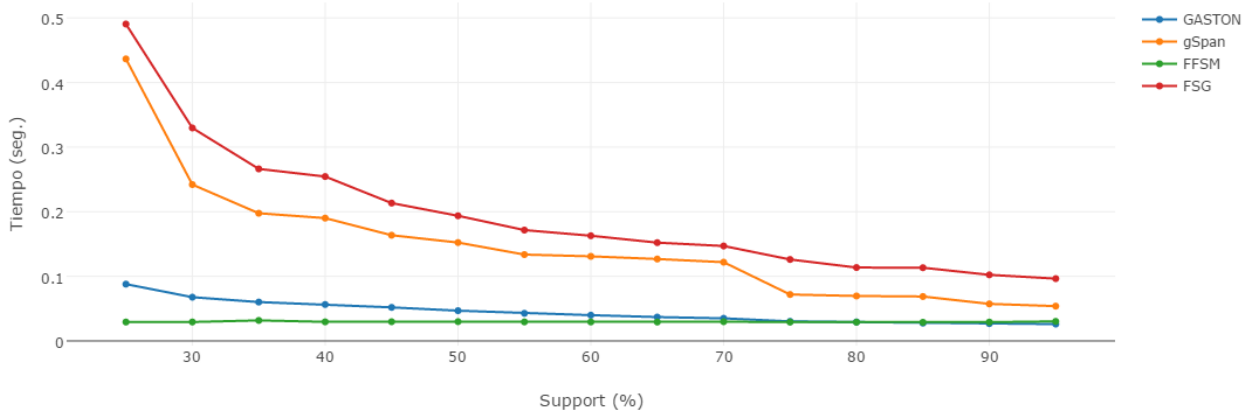


Figura 5.6.b. *Tiempo de ejecución por cada algoritmo en base al umbral mínimo para las pruebas con grafos reales*

desplome de rendimiento del algoritmo. Igualmente, si bien antes se destacaba por el tiempo de ejecución, en esta prueba de destaca mucho más también porque la cantidad de subgrafos frecuentes encontrados es mucho mayor que el que encuentran los otros algoritmos, distinto a lo que ocurrió anteriormente ya que no superaba por tanto a los otros, tal vez por la naturaleza de los grafos utilizados.

Además de la superioridad del algoritmo GASTON, se observa que al igual los algoritmos gSpan y FSG tienen un desempeño similar en cuanto a estructuras encontradas, lo que refuerza los resultados obtenidos anteriormente. También en este caso el algoritmo gSpan supera en velocidad al algoritmo FSG, tal como ocurrió en las primeras configuraciones de los experimentos con grafos sintéticos.

6. CONCLUSIONES

En este capítulo se presentan las conclusiones obtenidas a partir de la investigación desarrollada para este trabajo final de licenciatura. A continuación se muestra un resumen de los resultados obtenidos y se hace un análisis comparativo de los mismos (Sección 6.1), seguido de las conclusiones globales (Sección 6.2) y finalmente se describen las futuras líneas de investigación (Sección 6.3).

6.1. ANÁLISIS COMPARATIVO DE LOS RESULTADOS OBTENIDOS

A lo largo del presente trabajo se llevaron a cabo diferentes experimentos para poner a prueba los algoritmos de minería de grafos escogidos. En las siguientes secciones se realiza un análisis de los resultados obtenidos por cada tipo de prueba y se presentan las conclusiones desprendidas de cada uno: en la sección 6.1.1 se muestra el análisis de las pruebas con grafos sintéticos y en la sección 6.1.2 se presenta el análisis de las pruebas con grafos reales.

6.1.1. ANÁLISIS DE LAS PRUEBAS CON GRAFOS SINTÉTICOS

En las siguientes secciones se presentan las conclusiones correspondientes a todas las pruebas que se realizaron con grafos sintéticos aleatoriamente generados, divididas según el tipo de prueba. En este caso, a diferencia de lo mostrado en la parte de Experimentación (Sección 5), los resultados serán divididos en: pruebas con nodos únicos (Sección 6.1.1.1) y pruebas con nodos repetidos (Sección 6.1.1.2). Posteriormente se presentan conclusiones generales de todas las pruebas (Sección 6.1.1.3).

6.1.1.1. ANÁLISIS DE LAS PRUEBAS CON NODOS ÚNICOS

En estos experimentos se ejecutaron los algoritmos con bases de datos en las cuales todos los vértices de los grafos contenían etiquetas distintas, de manera que cada elemento sea distinto del resto. Un ejemplo de un grafo con diez nodos y diez arcos que podría formar parte de este tipo de pruebas puede observarse en la Figura 6.1.

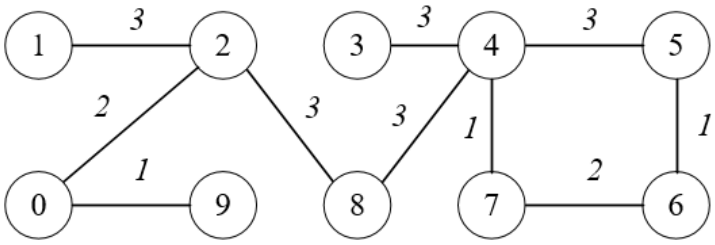


Figura 6.1. *Ejemplo de grafo para las pruebas con grafos sintéticos sin repetición de nodos*

En todos los experimentos con grafos sintéticos las pruebas se realizan con grafos no dirigidos y cuyas aristas pueden tener tres tipos de etiquetas: "*1*", "*2*" o "*3*".

Para el análisis de los resultados obtenidos se tendrán en cuenta los experimentos con las configuraciones presentes en la tabla 6.1. En ella se unificaron los experimentos con incremento fijo y variable para realizar una comparación global, generando un total de 82 distintos escenarios al combinar los dos tipos de prueba y eliminando aquellas en las que se repiten las variables |V| y |E|. En estos casos se realizó un promedio de los resultados de ambas pruebas.

Las celdas coloreadas en celeste indican las configuraciones tomadas de las pruebas con incremento variable. El resto corresponde a las pruebas con incremento fijo.

Orden	Exp.	|V|	|E|
1	1	10	10
2	2	10	15
3	3	10	20
4	4	10	25
5	5	10	30
6	6	20	20
7	7	20	25
8	8	20	30
9	9	20	35
10	10	20	40
11	9	20	50
12	10	20	60
13	11	30	30
14	12	30	35
15	13	30	40
16	14	30	45
17	15	30	50
18	13	30	60
19	14	30	75
20	15	30	90
21	16	40	40
22	17	40	45
23	18	40	50
24	19	40	55
25	20	40	60
26	18	40	80
27	19	40	100
28	20	40	120

Orden	Exp.	|V|	|E|
29	21	50	50
30	22	50	55
31	23	50	60
32	24	50	65
33	25	50	70
34	22	50	75
35	23	50	100
36	24	50	125
37	25	50	150
38	26	60	60
39	27	60	65
40	28	60	70
41	29	60	75
42	30	60	80
43	27	60	90
44	28	60	120
45	29	60	150
46	30	60	180
47	31	70	70
48	32	70	75
49	33	70	80
50	34	70	85
51	35	70	90
52	32	70	105
53	33	70	140
54	34	70	175
55	35	70	210

Orden	Exp.	|V|	|E|
56	36	80	80
57	37	80	85
58	38	80	90
59	39	80	95
60	40	80	100
61	37	80	120
62	38	80	160
63	39	80	200
64	40	80	240
65	41	90	90
66	42	90	95
67	43	90	100
68	44	90	105
69	45	90	110
70	42	90	135
71	43	90	180
72	44	90	225
73	45	90	270
74	46	100	100
75	47	100	105
76	48	100	110
77	49	100	115
78	50	100	120
79	47	100	150
80	48	100	200
81	49	100	250
82	50	100	300

Tabla 6.1. *Tabla resumen de la configuración de los experimentos para las pruebas con grafos sintéticos con nodos únicos*

En la columna experimento se muestra el id original correspondiente a las Tablas 5.1 y 5.2. En la columna orden, se muestra el id que servirá como variable independiente para el gráfico comparativo de la figura 6.2. En el eje de abscisas del gráfico mencionado, se presentan los resultados expresados en estructuras/milisegundos. De esta manera, puede observarse el desempeño general de cada algoritmo a medida que se incrementa el tamaño de las bases de datos, y como son afectados por la variación en la densidad de los grafos que las componen.

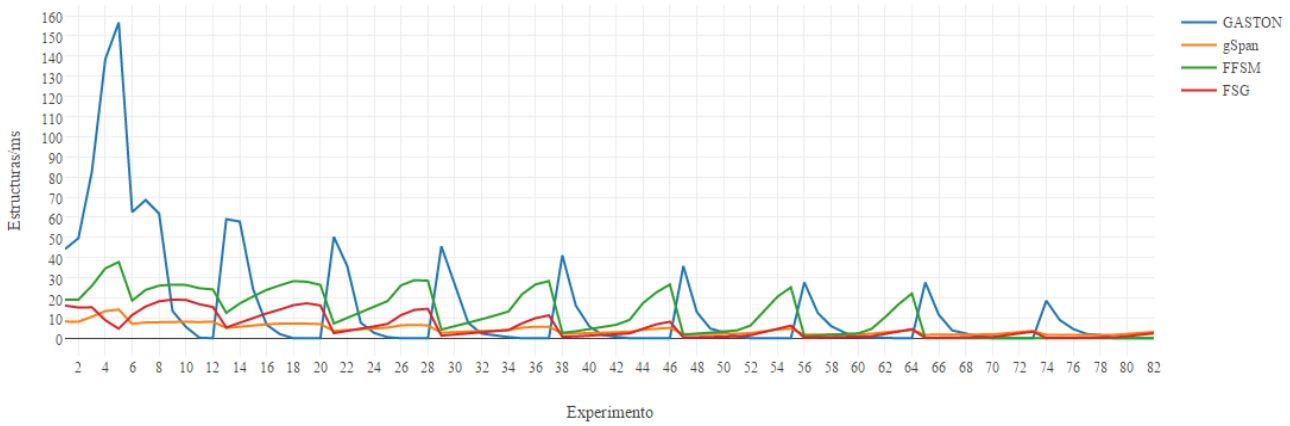


Figura 6.2. *Gráfico resumen de los resultados de los experimentos con grafos sintéticos con nodos únicos*

Analizando el gráfico puede notarse que el algoritmo GASTON es superior al resto para los primeros experimentos, en los que las bases son relativamente chicas. Sin embargo, su rendimiento decae abruptamente en las pruebas 11 y 12, en las cuales se incrementa la cantidad de aristas. En estos escenarios, la implementación directamente no es capaz de encontrar ninguna subestructura y cesa su ejecución, mientras que el resto de los algoritmos sí lo hace. Este comportamiento es recurrente cada vez que se aumenta la densidad de los grafos (pruebas 18, 25, 35, 42, 52, 60, 70 y 78). En el resto de las pruebas, cuando es capaz de encontrar estructuras, sus resultados son mejores que los que producen los otros algoritmos.

En el caso del FFSM, su comportamiento es más constante que el GASTON y genera mejores resultados que el gSpan y el FSG hasta la prueba 65. En ese momento deja de encontrar resultados de la misma manera que ocurre con el GASTON. De todas maneras, lo que parece influir más en el comportamiento del FFSM es el tamaño de la base de datos y no la densidad, ya que en la prueba 74, en la que las bases tienen 100 vértices y 100 aristas, el algoritmo GASTON encuentra resultados y el FFSM no.

Los otros dos algoritmos, el gSpan y el FSG son más lentos pero son capaces de encontrar resultados en todos los escenarios, incluso en las últimas pruebas. También se puede observar que sus comportamientos son más lineales que los del resto.

Comparando al gSpan y FSG, éste último obtiene mejores resultados en gran parte de las pruebas, pero esta tendencia se revierte en los experimentos finales, a partir del número 48. Desde allí, el gSpan es ligeramente superior, y luego de la prueba 78 el gSpan supera no sólo al FSG sino a los demás algoritmos también.

6.1.1.2. ANÁLISIS DE LAS PRUEBAS CON NODOS REPETIDOS

En estos experimentos se ejecutaron los algoritmos con bases de datos en las cuales los vértices de los grafos contenían etiquetas repetidas, de manera que se puedan modelar situaciones en las cuales existan dos o varios objetos iguales dentro de la red. En estas pruebas, se utiliza una variable llamada nodos únicos que es igual a $|V|/2$ (ver Sección 4.2), la cual determina la cantidad máxima de vértices con distintas etiquetas que puede tener un grafo. Por ejemplo, si la variable nodos únicos tiene un valor de 5, quiere decir que los nodos tendrán etiquetas desde el 0 al 4. Con esto se garantiza que haya elementos repetidos. Un ejemplo de un grafo con diez nodos, diez arcos y cinco nodos únicos que podría formar parte de este tipo de pruebas puede observarse en la Figura 6.3

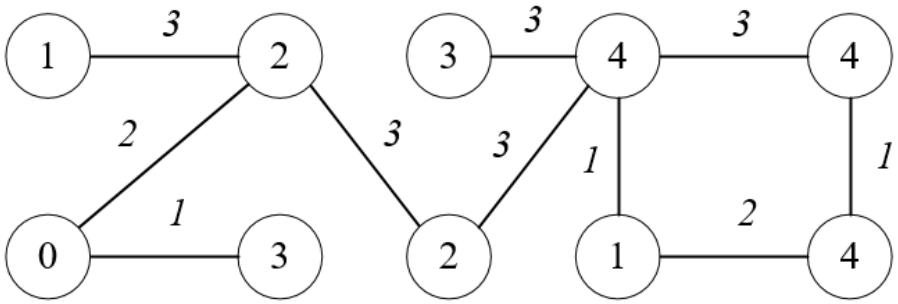


Figura 6.3. *Ejemplo de grafo para las pruebas con grafos sintéticos con repetición de nodos*

En este caso para el análisis de los algoritmos también se combinarán los resultados obtenidos de las pruebas con incremento fijo e incremento variable. Al igual que en los experimentos anteriores, la configuración de las bases de datos es igual a la de la tabla 6.1, siempre recordando que la cantidad máxima de etiquetas distintas va a ser igual a la mitad de la cantidad de vértices ($|V|/2$). El gráfico resumen de los resultados obtenidos se puede observar en la figura 6.4.

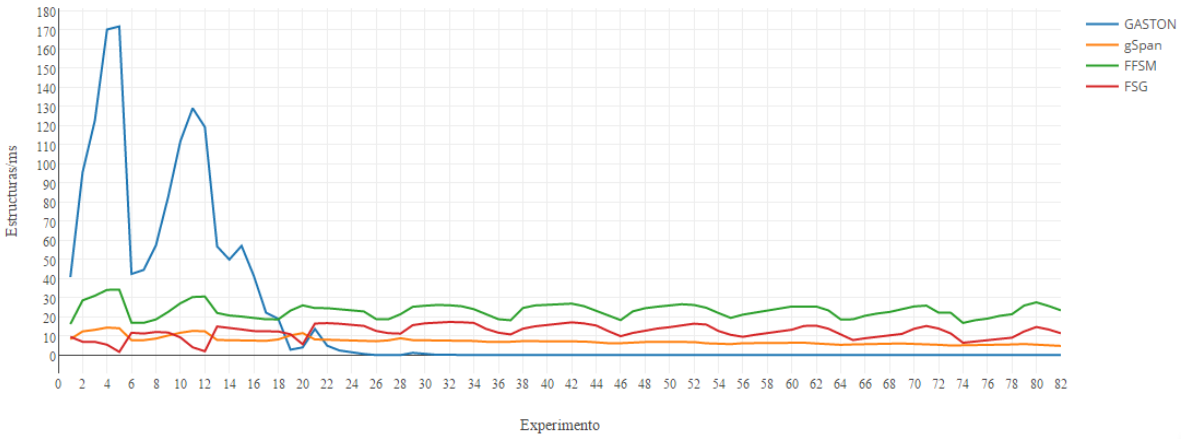


Figura 6.4. *Gráfico resumen de los resultados de los experimentos con grafos sintéticos con nodos repetidos*

Analizando el gráfico generado a partir de los resultados, a simple vista se puede observar un cambio en el comportamiento de los algoritmos con respecto a los presentados en la sección 6.1.1.1. El GASTON sigue siendo muy superior para las primeras pruebas pero hasta la prueba 17, luego su rendimiento decae y a partir de la prueba 24 no es capaz de encontrar subestructuras, exceptuando la prueba 29. Se siguen observando dificultades a medida que aumenta la densidad de la base de datos, aunque ahora este problema se ve más acentuado.

El resto de los algoritmos se comporta de manera más constante que en las pruebas anteriores y todos son capaces de encontrar resultados hasta el final, siendo el FFSM el que encuentra más resultados en menor tiempo de comienzo a fin.

En las primeras pruebas el gSpan y el FSG arrojan resultados cambiantes hasta la prueba 20, en la cual ambos se estabilizan y el FSG consigue tener un mejor rendimiento.

6.1.1.3. CONCLUSIONES GENERALES DE LAS PRUEBAS CON GRAFOS SINTÉTICOS

De lo expuesto en las secciones anteriores se pueden derivar las siguientes conclusiones con respecto a los experimentos realizados con grafos sintéticos generados aleatoriamente:

1. La conformación de la base altera el rendimiento de los algoritmos. No es lo mismo tener estructuras con elementos únicos, como una red social, a tener estructuras con elementos repetidos, como puede ser una molécula. El tamaño de los grafos y la densidad de los mismos también afecta al comportamiento de todos los algoritmos.
2. El algoritmo GASTON es más eficiente que el resto para una base de datos compuesta por grafos chicos, de hasta aproximadamente 30 vértices y 60 aristas, pero se ve fuertemente afectado al aumentarse la densidad de la base y cuando se tienen muchos nodos repetidos.
3. El algoritmo FFSM es el segundo mejor algoritmo en las primeras pruebas, y el más eficiente luego de que disminuye el rendimiento del GASTON. Su rendimiento es mejor para las pruebas con grafos repetidos, ya que siempre es capaz de encontrar resultados. En los otros experimentos hay varias pruebas en la que no puede (56 y de la 66 en adelante).
4. El comportamiento de los algoritmos gSpan y FSG son muy similares en las pruebas con nodos únicos, siendo el gSpan superior hacia las pruebas finales. En el caso de los experimentos con vértices repetidos, el FSG es mejor a partir de la prueba 20 hasta el final.

6.1.2. ANÁLISIS DE LAS PRUEBAS CON GRAFOS REALES

Luego de los experimentos con grafos sintéticos, se llevó a cabo una serie de pruebas con una base de datos con estructuras moleculares, utilizada por varios autores para probar sus algoritmos [Yan et al., 2002, Huan et al., 2003]. Las características de esta base se muestran en la Tabla 6.2. En

la misma pueden verse los promedios de la cantidad de vértices y aristas, así como también la cantidad de etiquetas. Más detalles de estas pruebas se describen en la Sección 4.1.2.

Base de datos para pruebas con grafos reales	
Cantidad de Grafos	422
Cantidad de etiquetas de arcos distintas	4
Cantidad de etiquetas de nodos distintas	21
Promedio de arcos por grafo	42
Promedio de nodos por grafo	40
Cantidad máxima de arcos por grafo	196
Cantidad máxima de nodos por grafo	189

Tabla 6.2. *Tabla resumen de la base de datos usada para los experimentos con grafos reales*

Mirando la tabla puede notarse que la estructura de los grafos es muy variada, teniendo en cuenta que el grafo con mayor cantidad de vértices posee 189 y el promedio es de 40. Considerando sólo los promedios y haciendo una analogía con las pruebas con grafos sintéticos, esta base se ubicaría dentro de los experimentos con nodos repetidos, aproximadamente entre las pruebas 13 y 21 según la tabla 6.1.

En la figura 6.5 se presentan los resultados obtenidos, contrastando el *minimum support* de cada prueba con la cantidad de estructuras encontradas por milisegundo.

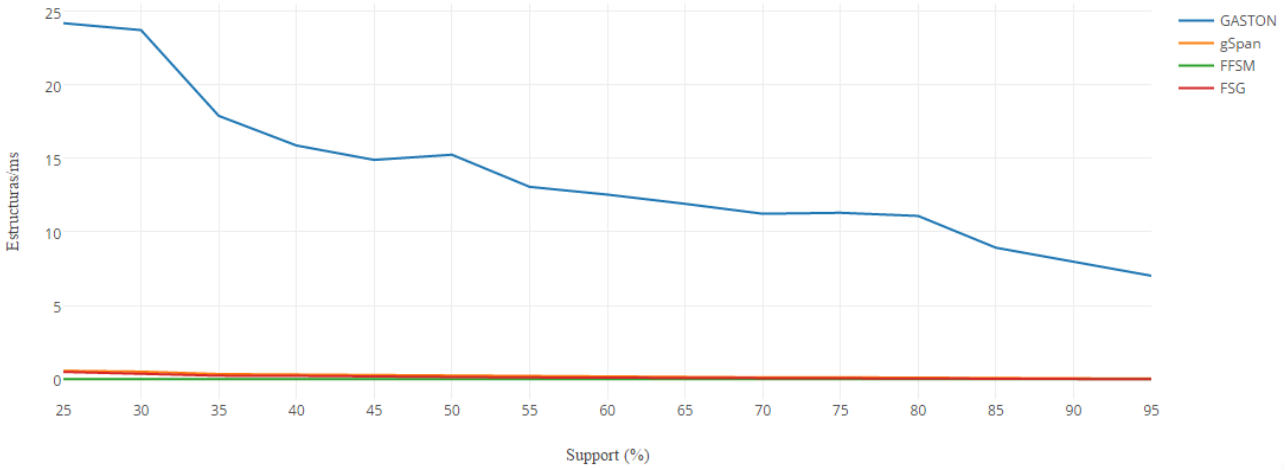


Figura 6.5. *Resultados para las pruebas con grafos reales*

A simple vista puede verse como el algoritmo GASTON supera ampliamente al resto de los algoritmos en las pruebas. Esto se debe a que las estructuras que conforman la base de datos no son tan densas y tal como ocurrió en las pruebas con grafos sintéticos, es estos escenarios el GASTON es el que se comporta de manera más eficiente.

Para analizar lo ocurrido con el resto de los algoritmos con más detalle se presenta la figura 6.6. En la misma se observa que el algoritmo gSpan es capaz de encontrar resultados en todas las pruebas y más eficientemente que el FSG. Este último deja de encontrar resultados cuando el support es de 95%.

El algoritmo FFSM, a pesar de haber conseguido buenos resultados en las pruebas con grafos sintéticos, no pudo encontrar resultados con esta base de datos. Tal vez se deba al tamaño de la base en relación al support utilizado ya que en las pruebas con grafos sintéticos, el umbral mínimo era del 5% y las bases estaban compuestas por 100 grafos, en lugar de los 400 que se utilizaron ahora.

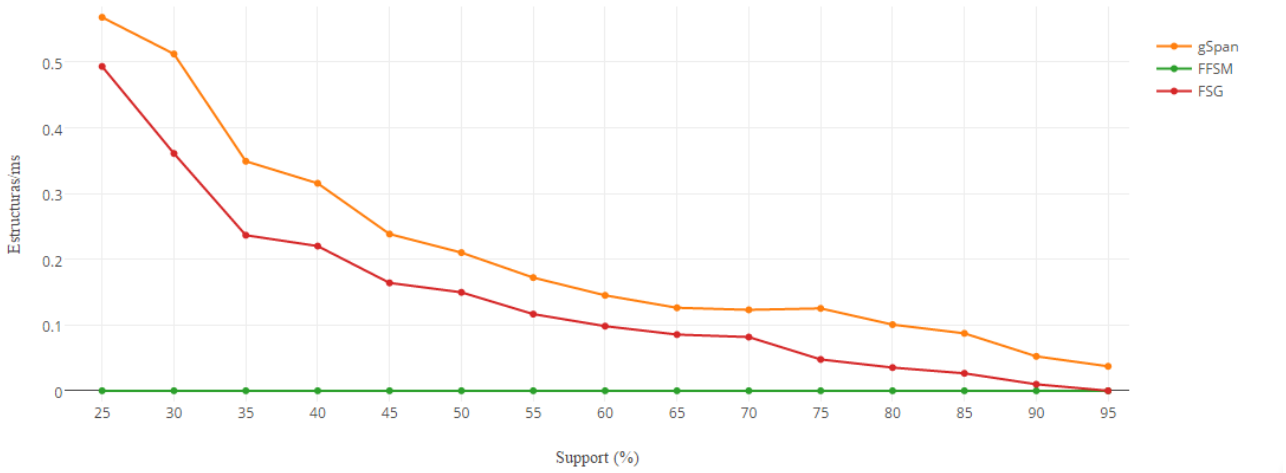


Figura 6.6. *Detalle de resultados para las pruebas con grafos reales de los algoritmos gSpan, FFSM y FSG*

6.2. CONCLUSIONES FINALES

En la presente investigación se llevaron a cabo experimentos sobre cuatro algoritmos de minería de grafos: GASTON, gSpan, FFSM y FSG. Mediante la construcción de un banco de pruebas se identifica la posibilidad de comparar su comportamiento en distintos escenarios, respondiendo a la primera pregunta planteada en el sumario de investigación de la sección 3.3. De esta manera se pudieron extraer conclusiones correspondientes a la segunda pregunta planteada: *¿es posible determinar cuáles son los escenarios más favorables para la utilización de cada algoritmo, de manera que se pueda facilitar la elección de los mismos dependiendo las necesidades que se tengan y los datos disponibles?*

Esto fue respondido mediante las pruebas con grafos sintéticos cuyas conclusiones fueron verificadas con una prueba con una base de datos real, compuesta por estructuras moleculares. Se determinó que las características de los grafos que alimenten a los algoritmos afectan sus comportamientos, se pudo comprobar que no hay un algoritmo superior al resto en todos los escenarios y además se identificó en que situación responde mejor cada uno.

El algoritmo GASTON es el más apropiado cuando las bases de datos no son muy densas y no poseen mucha complejidad. Si las estructuras son complejas pero con un tamaño moderado, el algoritmo FFSM es la mejor opción, sobre todo si los grafos cuentan con etiquetas repetidas. En el caso de grandes estructuras, las mejores opciones son el FSG y el gSpan, siendo este último el único

que pudo encontrar subestructuras en todos los experimentos, por lo que se lo considera el más estable.

De las pruebas con grafos reales se pudo verificar también el hecho de que los algoritmos siempre encontraron la misma cantidad de subestructuras en cada iteración de las pruebas, por lo que no es necesario ejecutar los algoritmos reiteradas veces para tener resultados confiables una vez que se hayan definido los parámetros a utilizar.

6.3. FUTURAS LÍNEAS DE INVESTIGACIÓN

La minería de grafos es un campo de constante crecimiento en la cual podrían desarrollarse las siguientes líneas de investigación:

I. Ampliar los algoritmos a evaluar incluyendo otros como el SPIN o CloseGraph para verificar su comportamiento.
II. Ampliar las pruebas con grafos reales de manera similar a lo realizado con grafos sintéticos: usar estructuras más simples y más complejas a la base de datos utilizada en esta investigación y comparar los resultados.
III. Desarrollar otras pruebas que involucren conjuntos de datos distintos en cuanto al tamaño o la conformación de las bases de datos. Podrían utilizarse grafos dirigidos o en el caso de que se usen bases de datos reales, podrían utilizarse otro tipo de estructuras que no sean moléculas.
IV. Realizar comparaciones similares a las de esta investigación para otros tipos de algoritmos de minería de grafos, como algoritmos de compresión o de clasificación.
V. Desarrollar un algoritmo propio de FSM, de manera que se mejoren los rendimientos de los algoritmos ya implementados. En este caso también tendrían que desarrollarse pruebas similares a las que se llevaron a cabo en esta investigación, para comprobar si el algoritmo nuevo efectivamente supera al resto en todos los escenarios posibles.
VI. Mejorar el banco de pruebas agregando una interfaz gráfica y la posibilidad de integrar nuevos algoritmos, nuevos escenarios y bases de datos sin la necesidad de manipular el código fuente.

7. REFERENCIAS

Britos, P. y García Martínez, R. 2009. Propuesta de Procesos de Explotación de Información. Proceedings XV Congreso Argentino de Ciencias de la Computación Workshop de Base de Datos y Minería de Datos. Págs. 1041-1050. ISBN 978-897-24068-4-1

Feder, T., & Motwani, R. (1991, January). Clique partitions, graph compression and speeding-up algorithms. In Proceedings of the twenty-third annual ACM symposium on Theory of computing (pp. 123-133). ACM.

Ferro, A., Giugno, R., Pigola, G., Pulvirenti, A., Skripin, D., Bader, G. D., & Shasha, D. (2007). NetMatch: a Cytoscape plugin for searching biological networks. Bioinformatics, 23(7), 910-912.

Fortin, S. (1996). The graph isomorphism problem. Technical Report 96-20, University of Alberta, Edomonton, Alberta, Canada.

García-Martínez, R., Britos, P., Pesado, P., Bertone, R., Pollo-Cattaneo, F., Rodríguez, D., Pytel, P., Vanrell. J. 2011. Towards an Information Mining Engineering. En Software Engineering, Methods, Modeling and Teaching. Sello Editorial Universidad de Medellín. ISBN 978-958-8692-32-6. Páginas 83-99.

Gonzalez, J., Jonyer, I., Holder, L. B., & Cook, D. J. (2000, July). Efficient mining of graph-based data. In Proceedings of the AAAI Workshop on Learning Statistical Models from Relational Data (pp. 21-28).

Grossman, R., Kasif, S., Moore, R., Rocke, D., Ullman, J. 1999. Data Mining Research: Opportunities and Challenges.

Hagberg, A., Schult, D., Swart, P. 2016. Networkx Framework. https:// networkx.github.io Pagina Vigente al 15/02/2016.

Holder, L. B., Cook, D. J., & Djoko, S. (1994, July). Substucture Discovery in the SUBDUE System. In KDD workshop (pp. 169-180).

Huan, J., Wang, W., & Prins, J. (2003, November). Efficient mining of frequent subgraphs in the presence of isomorphism. In Data Mining, 2003. ICDM 2003. Third IEEE International Conference on (pp. 549-552). IEEE.

Huan, J., Wang, W., Prins, J., & Yang, J. (2004, August). Spin: mining maximal frequent subgraphs from graph databases. In Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 581-586). ACM.

Krishna, V., Suri, N. N. R. R., & Athithan, G. (2011). A comparative survey of algorithms for frequent subgraph discovery. Current Science (Bangalore), 100(2), 190-198.

Kuramochi, M., & Karypis, G. (2001). Frequent subgraph discovery. In Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on (pp. 313-320). IEEE.

Lahiri, M., & Berger-Wolf, T. Y. (2007, March). Structure prediction in temporal networks using frequent subgraphs. In Computational Intelligence and Data Mining, 2007. CIDM 2007. IEEE Symposium on (pp. 35-42). IEEE.

Nijssen, S., & Kok, J. N. (2004, August). A quickstart in frequent structure mining can make a difference. In Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 647-652). ACM.

Ohlrich, M., Ebeling, C., Ginting, E., & Sather, L. (1993, July). SubGemini: identifying subcircuits using a fast subgraph isomorphism algorithm. In Proceedings of the 30th international Design Automation Conference (pp. 31-37). ACM.

Przulj, N., Corneil, D. G., & Jurisica, I. (2006). Efficient estimation of graphlet frequency distributions in protein-protein interaction networks. Bioinformatics, 22(8), 974-980.

Ramirez, A., Ospina. D., Ocampo, D. 2016. Aplicaciones de los Grafos. https://sites.google.com/site/aplicaciongrafos/ Pagina Vigente al 15/02/2016.

Rehman, S. U., Khan, A. U., & Fong, S. (2012, August). Graph mining: A survey of graph mining techniques. In Digital Information Management (ICDIM), 2012 Seventh International Conference on (pp. 88-92). IEEE.

Rentsch, T. (1982). Object oriented programming. ACM Sigplan Notices, 17(9), 51-57.

Takigawa, I., & Mamitsuka, H. (2013). Graph mining: procedure, application to drug discovery and recent advances. Drug discovery today, 18(1), 50-57.

Ullman, J. R. (1976). An algorithm for subgraph isomorphism. Journal of the ACM (JACM), 23(1), 31-42.

Wörlein, M., Meinl, T., Fischer, I., & Philippsen, M. (2005). A quantitative comparison of the subgraph miners MoFa, gSpan, FFSM, and Gaston (pp. 392-403). Springer Berlin Heidelberg.

Yan, X., & Han, J. (2002). gspan: Graph-based substructure pattern mining. In Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on (pp. 721-724). IEEE.

Zaiat, J., & Romero-Zaliz, R. Mineria de datos sobre grafos: un enfoque multiobjetivo aplicado a bioremediación.

A. DESARROLLO DEL BANCO DE PRUEBAS

En este anexo se describen los aspectos correspondientes al diseño y codificación del banco de pruebas desarrollado para llevar a cabo los experimentos con los algoritmos. Primero se presentan los requerimientos de la herramienta (Sección A.1), a continuación se muestran los diagramas de caso de uso derivados de los requerimientos (Sección A.2), el diagrama de clases general (Sección A.3) y finalmente el código implementado en el lenguaje de programación Python (Sección A.4).

A.1. REQUERIMIENTOS

Para llevar a cabo satisfactoriamente todos los experimentos propuestos en la Sección 4, es necesario desarrollar una herramienta que cuente con las siguientes características:

- Integración: debe poder permitir integrar las distintas implementaciones de los algoritmos en un ambiente de pruebas homogéneo, de manera que se puedan comparar sus comportamientos en distintos escenarios.

- Generación de bases de datos: para las pruebas con grafos sintéticos, debe posibilitar la generación de conjuntos de grafos con características definidas por el experimentador que luego servirán como entrada para todos los algoritmos, permitiendo variar la cantidad de vértices y aristas también así como el etiquetado de los grafos. A su vez, la herramienta debe convertir cada base de datos al formato correspondiente para cada algoritmo, como se establece en la Sección 4.2.2.

- Variación de los parámetros de los algoritmos: en las pruebas con grafos reales será necesario modificar el *minimum support* de cada implementación por lo que se debe poder manipular los parámetros de las implementaciones.

- Ejecución de los experimentos: Para cada experimento, se debe poder ejecutar cada algoritmo sobre la misma base de datos varias veces y extraer los resultados obtenidos en cada ejecución, para luego realizar un promedio.

- Almacenamiento de resultados: Para cada tipo experimento y para cada conjunto de variables, la herramienta debe almacenar el promedio de tiempo de ejecución y subestructuras encontradas por cada algoritmo de manera que se pueda realizar un análisis posterior, Estos datos se presentarán con un formato similar al de la Tabla A.1.

- Interfaz: para esta investigación no será necesario el desarrollo de una interfaz gráfica.

Formato de resultados para grafos sintéticos				
Vértices	*Aristas*	*Support*	*Estructuras*	*Tiempo*
10	15	5	127.1	0.012
…	…	…	…	…
Formato de resultados para grafos sintéticos				
Support	*Estructuras*	*Tiempo*		
25	257	0.022		
…	…	…		

Tabla A.1. *Formato de archivo de salida para cada algoritmo en cada experimento.*

A.2. CASOS DE USO

De lo expuesto en la sección anterior se pueden derivar los casos de uso representados en la Figura A.1, en la cual se describen las actividades que debe permitir realizar la herramienta con intervención del experimentador.

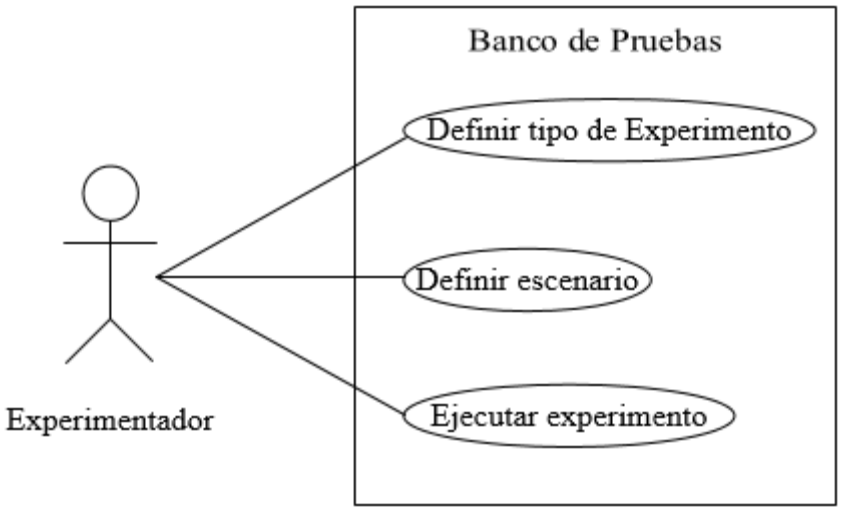


Figura A.1. *Diagrama de casos de uso general del banco de pruebas.*

Definir Escenario: consta en determinar las variables iniciales de los experimentos, ya sea la cantidad de nodos y arcos de la base de datos para los experimentos con grafos sintéticos, el *minimum support* para los experimentos con grafos reales, la cantidad de pruebas o la cantidad de repeticiones de cada prueba.

Definir tipo de Experimento: el experimentador debe poder definir el tipo de prueba que se llevará a cabo. Estos tipos están definidos en la Sección 4.1.1.

Ejecutar Experimento: este caso de uso consta de realizar la ejecución completa de un experimento. Internamente, el software realiza la ejecución de cada prueba una cierta cantidad de veces, la modificación de las variables correspondientes entre prueba y prueba, el almacenamiento parcial de los resultados, el promedio de cada prueba y la extracción de los archivos correspondientes a los resultados de la experimentación.

Además de los casos de uso mencionados, la herramienta deberá realizar otras funciones sin intervención del experimentador. Estos casos de uso pueden observarse en la Figura A.2, en la cual se muestra el diagrama completo del software desarrollado.

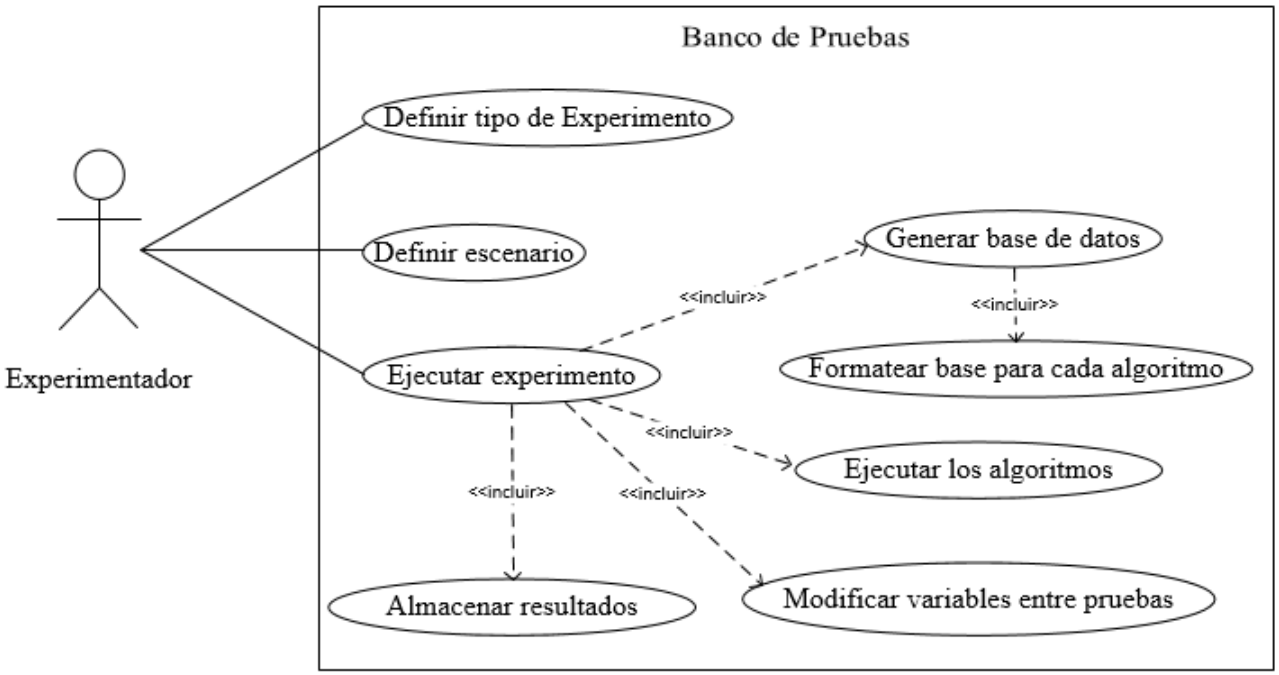


Figura A.2. *Diagrama de casos de uso completo del banco de pruebas.*

A.3. DIAGRAMA DE CLASES

En la presente sección se presenta un diagrama de clases de la herramienta desarrollada, incluyendo las clases adaptadoras para cada algoritmo (ver Sección 4.2.3), la clase encargada de generar los grafos pertenecientes a los experimentos con grafos sintéticos y la clase principal del banco de pruebas. Dicho diagrama se muestra en la Figura A.3.

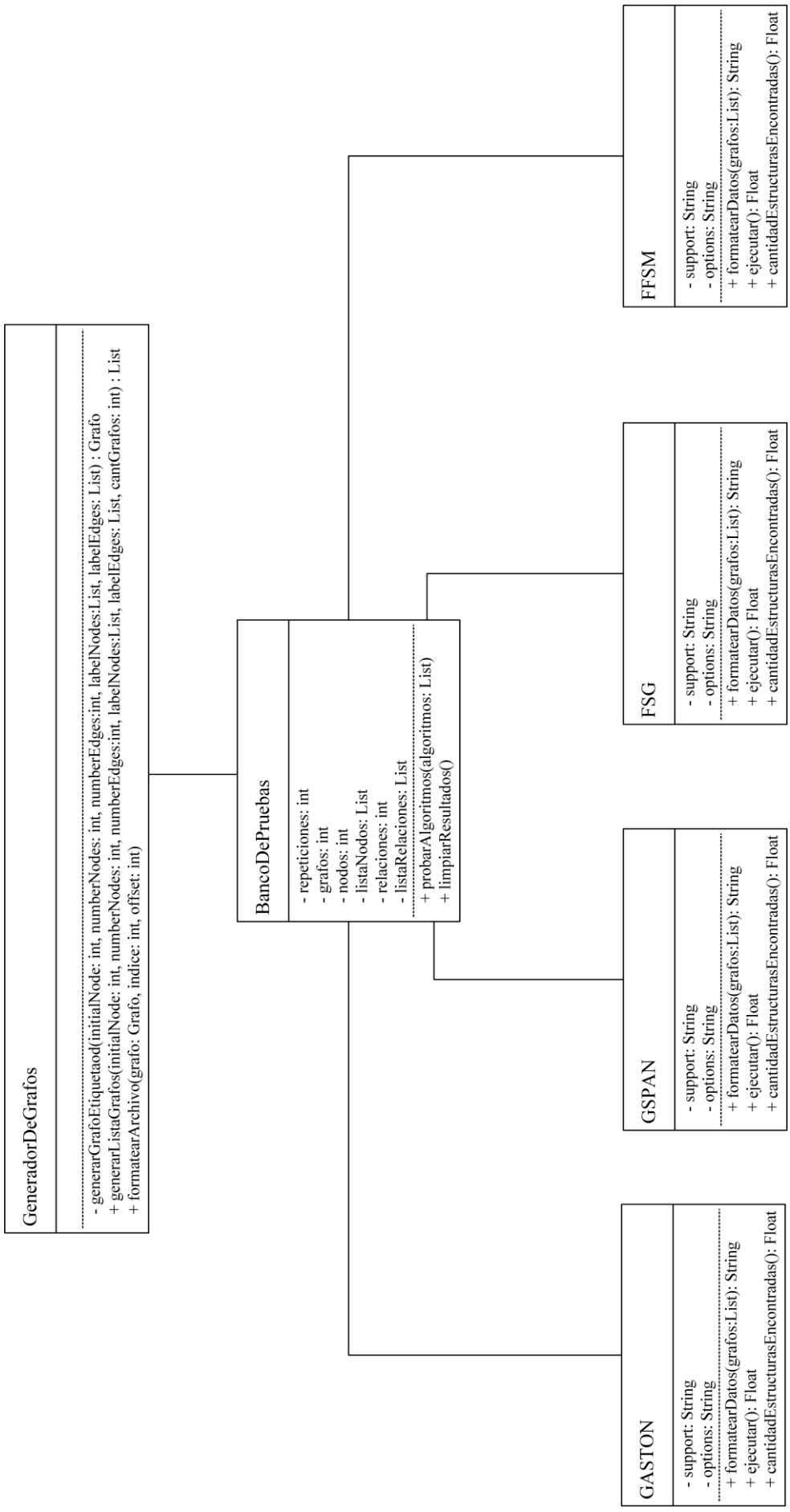


Figura A.3. *Diagrama de clases del banco de pruebas.*

Generador de Grafos: Esta clase no posee atributos y contiene dos métodos públicos: generar lista de grafos y formatear archivos. El primero es el encargado de generar la base de datos a utilizar en las pruebas con grafos sintéticos en memoria RAM, indicando la cantidad de grafos, cantidad de arcos, cantidad de nodos y las etiquetas correspondientes para las aristas y los vértices.
El segundo, se encarga de formatear los grafos generados en archivos con el formato que necesite cada algoritmo.

Banco de Pruebas: Esta es la clase encargada de ejecutar todas las pruebas y recopilar los resultados en los experimentos con grafos sintéticos. Mediante el método probar algoritmos se llevan a cabo todas las pruebas de un determinado experimento, se recopilan los resultados y se generan los archivos con los promedios. El método limpiar resultados se utiliza antes de cada nuevo tipo de experimento para generar nuevos archivos de resultados.
Esta clase posee seis atributos:

- *Repeticiones*: determina cuantas veces se repetirá una prueba para calcular el promedio.
- *Grafos:* indica la cantidad de grafos del experimento.
- *Nodos:* indica la cantidad de vértices de los grafos del experimento.
- *Relaciones:* indica la cantidad de aristas de los grafos del experimento.
- *Lista de nodos:* indica las etiquetas permitidas para los vértices de los grafos del experimento.
- *Lista de relaciones:* indica las etiquetas permitidas para las aristas de los grafos del experimento.

FFSM, FSG, GASTON, GSPAN: Cada una de estas clases se encarga de ejecutar un algoritmo y extraer los resultados de la ejecución para que luego la clase de Banco de Pruebas realice los promedios. El atributo s*upport* se corresponde con el minimum support, el cual será utilizado principalmente para los experimentos con grafos reales. El atributo *options* se utiliza en caso de que se quiera inicializar algún atributo especial de cada implementación. Es un String opcional, que contiene todas las opciones que se quieran agregar con su valor. Si se encuentra vacía, se utiliza la configuración por defecto.
El método ejecutar se encarga de realizar una ejecución del algoritmo sobre la base de datos y retornar el tiempo que tarda en segundos.
El método cantidad de estructuras encontradas analiza el archivo de salida del algoritmo y retorna la cantidad de subestructuras que haya encontrado en esa ejecución.

A.4. CODIFICACIÓN

En la presente sección se presenta el código implementado en Python de las clases descritas en la sección anterior. Además se incluyen los procesos de inicialización para las pruebas con grafos sintéticos y los experimentos con grafos reales.

A.4.1. GENERADOR DE GRAFOS

```
1.  import networkx as nx
2.  import random
3.
4.  def generarGrafoEtiquetado(initialNode,numberNodes,numberEdges,labelListNode,labelListEdge
    ):
5.      grafo=nx.gnm_random_graph(numberNodes, numberEdges)
6.      for n in grafo:
7.          if(len(labelListNode)!=numberNodes):
8.              grafo.node[n]['label']=random.choice(labelListNode)
9.          else: grafo.node[n]['label']=str(n)
10.     for n in grafo.edges(data=True):
11.         grafo[n[0]][n[1]]['label']=random.choice(labelListEdge)
12.     return grafo
13.
14. def generarListaGrafos(initialNode,numberNodes,numberEdges,labelListNode,labelListEdge,can
    t):
15.     grafos=list()
16.     for n in range(cant):
17.         gra-
    fos.append(generarGrafoEtiquetado(initialNode,numberNodes,numberEdges,labelListNode,labelL
    istEdge))
18.     return grafos
19.
20. def formatearArchivo(grafo,indice=None,offset=0):
21.     if indice==None:
22.         nodos=""
23.         arcos=""
24.         for nodo in grafo:
25.             id=str(nodo+offset)
26.             label= grafo.node[nodo]['label']
27.             nodos=nodos+"v "+id+" "+label+"\n"
28.         for n in grafo.edges(data=True):
29.             id1=str(n[0]+offset)
30.             id2=str(n[1]+offset)
31.             label=str(n[2]['label'])
32.             arcos=arcos+"e "+id1+" "+id2+" "+label+"\n"
33.         return [nodos,arcos,nx.number_of_nodes(grafo)]
34.     else:
35.         nodos=""
36.         arcos=""
37.         indice=str(indice)
38.         for nodo in grafo:
39.             id=str(nodo)
40.             label= grafo.node[nodo]['label']
41.             nodos=nodos+"node "+" "+indice+" "+id+" "+label+"\n"
42.         for n in grafo.edges(data=True):
43.             id1=str(n[0])
44.             id2=str(n[1])
45.             label=str(n[2]['label'])
46.             arcos=arcos+"edge "+" "+indice+" "+id1+" "+id2+" "+label+"\n"
47.         return [nodos,arcos]
```

A.4.2. BANCO DE PRUEBAS

```
1.  import GraphGenerator as grgen
2.  import os
3.  from glob import glob
4.  import csv
5.
6.  class BancoPruebas():
7.      repeticiones=0
8.      grafos=0
9.      nodos=0
10.     listaNodos=None
11.     relaciones=0
12.     listaRelaciones=None
13.
14.     def __init__(self,cantRepeticiones,cantGrafos,cantNodos,cantTiposNodos,cantRelaciones,
    cantTiposRelaciones):
15.         self.repeticiones=cantRepeticiones
16.         self.grafos=cantGrafos
17.         self.nodos=cantNodos
18.         self.relaciones=cantRelaciones
19.         if self.listaNodos is None:
20.             self.listaNodos = []
21.         if self.listaRelaciones is None:
22.             self.listaRelaciones=[]
23.         for i in range(cantTiposNodos):
24.             self.listaNodos.append(str(i))
25.         for i in range(cantTiposRelaciones):
26.             self.listaRelaciones.append(str(i+1))
27.
28.     def probarAlgoritmos(self,algoritmos):
29.         f=open('resultados','a')
30.         #f.write("\n"+self.strGrafos()+" "+self.strNodos()+" "+self.strRelaciones()+" "+se
    lf.strLabelNodos()+" "+self.strLabelRelaciones())
31.         f.write("\n"+self.strNodos()+" "+self.strRelaciones())
32.         f.flush()
33.         struct=[]
34.         ciclos=[]
35.         t=[]
36.         files=[]
37.         errores=[]
38.         for algoritmo in algoritmos:
39.             struct.append(0)
40.             ciclos.append(0)
41.             t.append(0.0)
42.             errores.append(0)
43.             faux=open(algoritmo.nombre()+".csv",'a')
44.             files.append(faux)
45.         for i in range(self.repeticiones):
46.             nAlg=0
47.             gra-
    fos=grgen.generarListaGrafos(0, self.nodos, self.relaciones, self.listaNodos, self.listaRe
    laciones, self.grafos)
48.             for algoritmo in algoritmos:
49.                 algoritmo.formatearDatos(grafos)
50.                 tiempo=algoritmo.ejecutar()
51.                 if algoritmo.cantidadEstructurasEncontradas()==0: errores[nAlg]=errores[nA
    lg]+1
52.                 struct[nAlg]=struct[nAlg]+algoritmo.cantidadEstructurasEncontradas()
53.                 ciclos[nAlg]=ciclos[nAlg]+algoritmo.ciclos()
54.                 t[nAlg]=t[nAlg]+tiempo
55.                 nAlg=nAlg+1;
56.         nAlg=0
57.         for algoritmo in algoritmos:
```

```
58.             f.write("\n"+algoritmo.nombre()+" (support="+algoritmo.getSupport()+')\nEstruc
    turas:'+str(self.promedio(struct[nAlg]))+'\nTiempo:'+str(self.promedio(t[nAlg]))+"\n")
59.             fi-
    les[nAlg].write(str(self.nodos)+";"+str(self.relaciones)+";"+algoritmo.getSupport()+";"+st
    r(self.promedio(struct[nAlg]))+";"+str(self.promedio(t[nAlg]))+";"+str(errores[nAlg])+'\n'
    )
60.             nAlg=nAlg+1
61.             f.flush()
62.         for i in range(len(algoritmos)):
63.             files[i].close()
64.         f.close()
65.
66.     def limpiarResultados(self):
67.         for f in glob('*.csv'):
68.             os.unlink (f)
69.         if os.path.isfile('resultados'): os.remove('resultados')
70.
71.     def promedio(self,valor):
72.         return valor/self.repeticiones
73.
74.     def strGrafos(self):
75.         return "CantGrafos: "+str(self.grafos)
76.
77.     def strRelaciones(self):
78.         return "CantRelaciones: "+str(self.relaciones)
79.
80.     def strNodos(self):
81.         return "CantNodos: "+str(self.nodos)
82.
83.     def strLabelNodos(self):
84.         nodos=""
85.         for nodo in self.listaNodos:
86.             nodos=nodos+nodo+"-"
87.         return "Nodos: "+nodos
88.
89.     def strLabelRelaciones(self):
90.         relaciones=""
91.         for relacion in self.listaRelaciones:
92.             relaciones=relaciones+relacion+"-"
93.         return "Relaciones: "+relaciones
```

A.4.3. CLASE ADAPTADORA DEL ALGORTIMO FFSM

```
1.  import GraphGenerator as grgen
2.  import os
3.  import time
4.
5.  class  FFSM():
6.      support = ""
7.      option=""
8.
9.      def __init__(self,support,options=""):
10.         self.support=str(support)
11.         if options!="":
12.             self.option="-"+options+" "
13.
14.     def formatearDatos(self,grafos):
15.         fn = open('FFSM/nodos.in', 'w')
16.         fa = open('FFSM/arcos.in','w')
17.         indice=0
18.         for grafo in grafos:
19.             datos=grgen.formatearArchivo(grafo,indice)
20.             fn.write(datos[0])
21.             fa.write(datos[1])
22.             indice=indice+1
```

```
        fn.close()
        fa.close()
        return datos

    def ejecutar(self):
        if os.path.isfile('FFSM/estadisticas'): os.remove('FFSM/estadisticas')
        ffsmPath=os.path.abspath("FFSM")
        coman-
    do="\n"+"cd "+ffsmPath+"\n"+"./FFSM nodos.in arcos.in nodos.out arcos.out feature.out "+se
    lf.support+" "+self.density+" "+self.sizelimit+" "+self.sizeuplimit+"\n"
        f=open("command.sh","w")
        f.write(comando)
        print comando
        f.close()
        os.system("chmod +x command.sh")
        start=time.time()
        os.system("./command.sh")
        end=time.time()
        os.remove("command.sh")
        return end-start

    def cantidadEstructurasEncontradas(self):
        f=open('FFSM/feature.out','r')
        patrones=0
        for line in f:
            patrones=patrones+1
        f.close()
        return float(patrones)

    def getNombre(self):
        return "FFSM"

    def getSupport(self):
        return self.support
```

A.4.4. CLASE ADAPTADORA DEL ALGORTIMO GSPAN

```
import GraphGenerator as grgen
import os
import time

class GSpan():

    support=""
    option=""

    def __init__(self,support,options=""):
        self.support=str(support)
        if options!="":
            self.option="-"+options+" "

    def formatearDatos(self,grafos):
        f = open('gSpan/input', 'w')
        indice=0
        for grafo in grafos:
            datos=grgen.formatearArchivo(grafo)
            f.write("t # "+str(indice)+"\n"+datos[0]+datos[1])
            indice=indice+1
        f.close()

    def ejecutar(self):
        if os.path.isfile('gSpan/estadisticas'): os.remove('gSpan/estadisticas')
        gastonPath=os.path.abspath("gSpan")
        #disk=gastonPath[0]+gastonPath[1]
```

```
28.         comando="\n"+"cd "+gastonPath+"\n"+"./gSpan -f input -s "+self.support+" -o -
    i "+self.option+"\n"
29.         f=open("command.sh","w")
30.         f.write(comando)
31.         print comando
32.         f.close()
33.         os.system("chmod +x command.sh")
34.         start=time.time()
35.         os.system("./command.sh")
36.         end=time.time()
37.         os.remove("command.sh")
38.         return end-start
39.
40.     def tiempoDeEjecucion(self):
41.         return float(0)
42.
43.     def cantidadEstructurasEncontradas(self):
44.         f=open("gSpan/input.fp","r")
45.         estructuras=0
46.         for line in f:
47.             if line[0]=='t':
48.                 estructuras=estructuras+1
49.         return float(estructuras)
50.
51.     def nombre(self):
52.         return "gSpan"
53.
54.     def getSupport(self):
55.         return self.support
```

A.4.5. CLASE ADAPTADORA DEL ALGORTIMO GASTON

```
1.  import GraphGenerator as grgen
2.  import os
3.  import time
4.
5.  class Gaston():
6.
7.      support=""
8.      option=""
9.
10.     def __init__(self,support,options=""):
11.         self.support=str(support)
12.         if options!="":
13.             self.option="-"+options+" "
14.
15.     def formatearDatos(self,grafos):
16.         f = open('GASTON/input', 'w')
17.         indice=0
18.         for grafo in grafos:
19.             datos=grgen.formatearArchivo(grafo)
20.             f.write("t # "+str(indice)+"\n"+datos[0]+datos[1])
21.             indice=indice+1
22.         f.close()
23.
24.     def ejecutar(self):
25.         if os.path.isfile('GASTON/estadisticas'): os.remove('GASTON/estadisticas')
26.         gastonPath=os.path.abspath("GASTON")
27.         #disk=gastonPath[0]+gastonPath[1]
28.         coman-
    do="\n"+"cd "+gastonPath+"\n"+"./GASTON "+self.support+" "+self.option+"input output \n"
29.         f=open("command.sh","w")
30.         f.write(comando)
31.         f.close()
```

```
32.         os.system("chmod +x command.sh")
33.         start=time.time()
34.         os.system("./command.sh")
35.         end=time.time()
36.         os.remove("command.sh")
37.         return end-start
38.
39.      def cantidadEstructurasEncontradas(self):
40.         f=open("GASTON/output","r")
41.         estructuras=0
42.         for line in f:
43.             if line[0]=='#':
44.                 estructuras=estructuras+1
45.         return float(estructuras)
46.
47.     def nombre(self):
48.         return "GASTON"
49.
50.     def getSupport(self):
51.         return self.support
```

A.4.6. CLASE ADAPTADORA DEL ALGORTIMO FSG

```
1.  import GraphGenerator as grgen
2.  import os
3.  import time
4.
5.  class FSG():
6.
7.      support=""
8.      option=""
9.
10.     def __init__(self,support,options=""):
11.         self.support=str(support)
12.         if options!="":
13.             self.option="-"+options+" "
14.
15.     def formatearDatos(self,grafos):
16.         f = open('FSG/input', 'w')
17.         indice=0
18.         for grafo in grafos:
19.             datos=grgen.formatearArchivo(grafo)
20.             f.write("t # "+str(indice)+"\n"+datos[0]+datos[1])
21.             indice=indice+1
22.         f.close()
23.
24.     def ejecutar(self):
25.         if os.path.isfile('FSG/estadisticas'): os.remove('FSG/estadisticas')
26.         gastonPath=os.path.abspath("FSG")
27.         #disk=gastonPath[0]+gastonPath[1]
28.         comando="\n"+"cd "+gastonPath+"\n"+"./FSG -
    s "+self.support+" input "+self.option+"\n"
29.         f=open("command.sh","w")
30.         f.write(comando)
31.         f.close()
32.         os.system("chmod +x command.sh")
33.         start=time.time()
34.         os.system("./command.sh")
35.         end=time.time()
36.         os.remove("command.sh")
37.         return end-start
38.
39.     def cantidadEstructurasEncontradas(self):
40.         f=open("FSG/input.fp","r")
```

```
        estructuras=0
        for line in f:
            if line[0]=='t':
                estructuras=estructuras+1
        return float(estructuras)

    def nombre(self):
        return "FSG"

    def getSupport(self):
        return self.support
```

A.4.7. INICIALIZACIÓN PARA EXPERIMENTOS CON GRAFOS SINTÉTICOS

```
from GASTON import *
from ffsm import *
from FSG import *
from gSpan import *
import BancoPruebas as bp
import os
import time

if __name__ == '__main__':
    #####PARAMETROS INICIALES
    #FIJOS PARA TODAS LAS PRUEBAS
    repeticiones=100
    cantGrafos=100
    cantNodos=10
    cantRelaciones=10
    relacionesUnicas=3  #Distintos tipos de relaciones
    nodosUnicos=cantNodos
    loopsNodos=10
    loopsRelaciones=5
    #DEFINEN AL TIPO DE EXPERIMENTO
    incFijo=True
    repeticionNodos=True

    comienza=True
    for i in range(0,loopsNodos):
        if repeticionNodos: nodosUnicos=cantNodos//2
        print(nodosUnicos)
        for n in range(0,loopsRelaciones):
            prue-
ba=bp.BancoPruebas(repeticiones,cantGrafos,cantNodos,nodosUnicos,cantRelaciones,relaciones
Unicas)
            if comienza:
                prueba.limpiarResultados()
                comienza=False
            prueba.probarAlgoritmos([Gaston(5),FSG(5),GSpan(0.05),FFSM(5)])
            if(incFijo):
                cantRelaciones+=5
            else:
                cantRelaciones+=cantNodos//2
        cantNodos=cantNodos+10
        cantRelaciones=cantNodos
```

A.4.8. INICIALIZACIÓN PARA EXPERIMENTOS CON GRAFOS REALES

```
1.  from GASTON import *
2.  from ffsm import *
3.  from FSG import *
4.  from gSpan import *
5.  import BancoPruebas as bp
6.
7.  def probarAlgoritmos(algoritmos,repeticiones):
8.      f=open('resultados','a')
9.      f.flush()
10.     struct=[]
11.     ciclos=[]
12.     t=[]
13.     files=[]
14.     for algoritmo in algoritmos:
15.         struct.append(0)
16.         ciclos.append(0)
17.         t.append(0.0)
18.         faux=open(algoritmo.nombre()+".csv",'a')
19.         files.append(faux)
20.     for i in range(repeticiones):
21.         nAlg=0
22.         for algoritmo in algoritmos:
23.             tiempo=algoritmo.ejecutar()
24.             struct[nAlg]=struct[nAlg]+algoritmo.cantidadEstructurasEncontradas()
25.             t[nAlg]=t[nAlg]+tiempo
26.             nAlg=nAlg+1;
27.     nAlg=0
28.     for algoritmo in algoritmos:
29.         f.write("\n"+algoritmo.nombre()+" (support="+algoritmo.getSupport()+')\nEstructura
    s:'+str(struct[nAlg]/repeticiones)+'\nTiempo:'+str(t[nAlg]/repeticiones)+"\n")
30.         fi-
    les[nAlg].write(algoritmo.getSupport()+";"+str(struct[nAlg]/repeticiones)+";"+str(t[nAlg]/
    repeticiones)+'\n')
31.         nAlg=nAlg+1
32.         f.flush()
33.     for i in range(len(algoritmos)):
34.         files[i].close()
35.     f.close()
36.
37.
38. for support in range(25,100,5):
39.     probarAlgorit-
    mos([Gaston(support,'p'),FSG(support),GSpan(support*0.01),FFSM(support)],100)
```