

Análisis Comparativo de Algoritmos de Minería de Subgrafos Frecuentes

Santiago Bianco

Grupo de Investigación en Sistemas de Información (UNLa GISI)

Departamento de Desarrollo Productivo y Tecnológico

Universidad Nacional de Lanús

Remedios de Escalada, Buenos Aires, Argentina.

Santiago.bianco.sb@gmail.com.

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 artículo 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.

Palabras clave— Minería de grafos, minería de subgrafos frecuentes, explotación de información, análisis de algoritmos.

I. INTRODUCCIÓN

En este capítulo se proporciona una introducción general al presente documento. Se presenta el marco del artículo (Sección 1.A), se establece la delimitación del problema tratado (Sección 1.B), se detallan los objetivos de la investigación (Sección 1.C) se describen brevemente la solución propuesta (Sección 1.D), se describe la metodología seguida (Sección 1.E) y se da una visión general de la estructura del artículo (Sección 1.F).

A. Marco del Artículo

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 [1; 2; 3].

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 [4]. 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.

B. Delimitación del Problema

El presente artículo 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.

C. Objetivos de la investigación

A continuación se presentan los objetivos de la investigación realizada divididos en Objetivos Generales (Sección I.C.1) y Objetivos Específicos (Sección I.C.2).

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.

2) *Objetivos Específicos*

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

- Investigar y analizar los distintos algoritmos de búsqueda de subgrafos frecuentes desarrollados hasta el momento.
- Desarrollar distintos casos de pruebas y experimentaciones cubriendo un amplio rango de escenarios posibles en los que se puedan ejecutar los algoritmos.
- Desarrollar una capa de software que permita ejecutar distintas implementaciones de algoritmos sobre las mismas bases de datos, permitiendo extraer y comparar resultados.
- 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.

D. *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.

E. *Metodología de Desarrollo*

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

1) *Etapas 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 la investigación.

2) *Etapas 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.

3) *Etapas 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.

4) *Etapas 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 se desprenden las conclusiones del presente artículo.

F. *Visión General del Artículo*

En la sección Introducción, se presenta el marco del presente artículo, se da una delimitación del problema, se plantean los elementos de la solución propuesta, y se da una visión general del proyecto.

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

En la sección Conclusiones, se presentan las aportaciones de esta investigación y se destacan las futuras líneas de investigación que se consideran de interés en base al problema abierto que se presenta en este artículo.

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

II. ESTADO DE LA CUESTIÓN

En esta sección se presentan los conceptos, teorías y algoritmos de interés para el desarrollo del presente artículo. Se comienza con una breve introducción al concepto de grafos (Sección II.A), luego se dan definiciones y se describen otros conceptos de interés (Sección II.B) los cuales son la teoría de grafos (Sección II.B.1), formas de representación para los mismos (Sección II.B.2) y cómo recorrerlos en anchura (Sección II.B.3.a) y profundidad (Sección II.B.3.b). Se continúa con una presentación del concepto de explotación de información (Sección II.C) para luego explicar los procesos de explotación de información en grafos (Sección II.D), la tarea de búsqueda de subgrafos frecuentes (Sección II.D.1) y una descripción de los algoritmos de interés para la investigación: FSG (Sección II.D.1.a), gSpan (Sección II.D.1.b), FFSM (Sección II.D.1.c) y GASTON (Sección II.D.1.d).

A. *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 verse en la figura 1.

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.

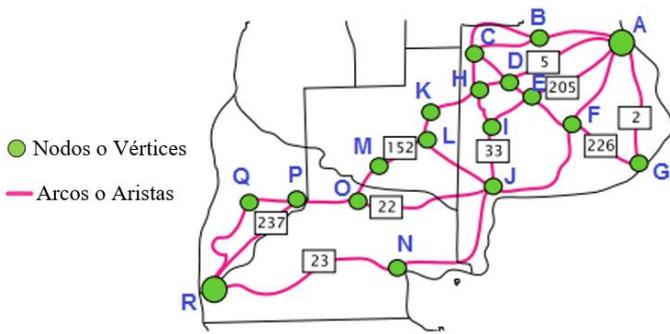


Fig. 1. Ejemplo de un mapa representado mediante el uso de grafos [5].

En este caso, el grafo que representa 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. Estos conceptos y otras variantes de grafos se explican con más detalle en la Sección II.B.1.

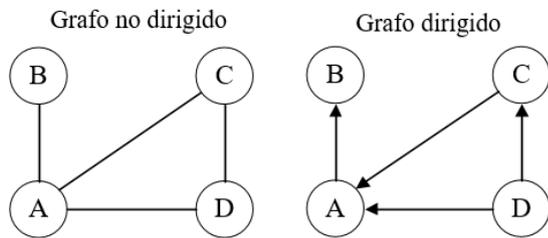


Fig. 2. Ejemplos de un grafo dirigido y un grafo no dirigido.

Prácticamente, cualquier tipo de red puede ser representado mediante grafos, lo que les da un carácter heterogéneo e interdisciplinario. En la figura 3 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.

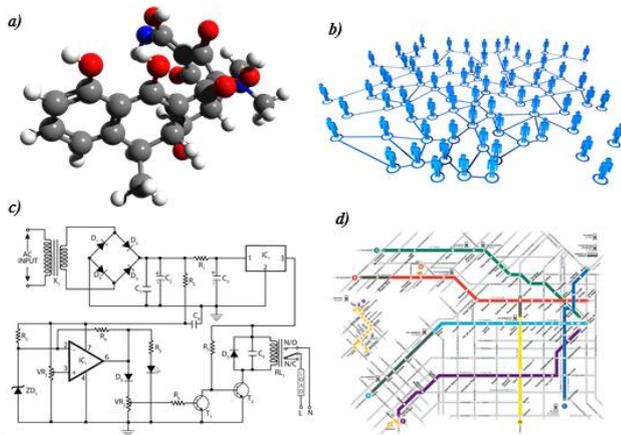


Fig. 3. Ejemplos de elementos que pueden ser modelados con grafos. a) Estructura química. b) Red Social. c) Diagrama Eléctrico. d) Conexión de estaciones de tren subterráneo.

B. Conceptos y Definiciones de Interés

En esta sección se presentan algunos conceptos de interés para esta investigación. Se dan definiciones formales correspondientes a la teoría de grafos (Sección II.B.1), se describen formas de representación (Sección II.B.2) y se

explican las principales maneras de recorrerlos: en anchura (Sección II.B.3.a) y en profundidad (Sección 2.II.3.a).

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, 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 las 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 4 se clarifica esta definición formal con un ejemplo gráfico de 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.

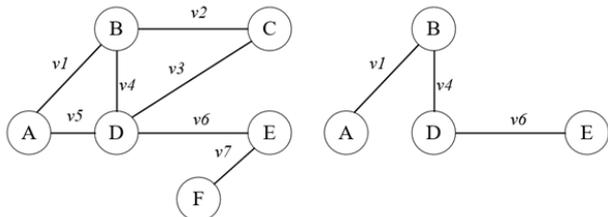


Fig. 4. Ejemplo de un grafo no dirigido y uno de los posibles subgrafos que contiene.

- **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 5 puede observarse un ejemplo de estos conceptos.



Fig. 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 abarcador 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 6 se muestra un árbol en expansión de un grafo.

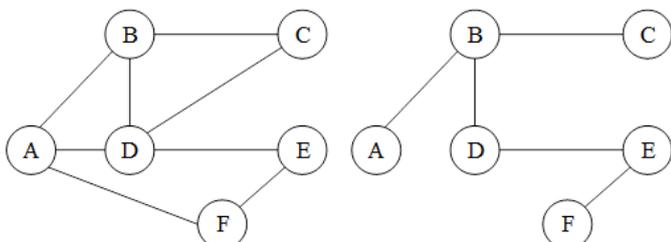


Fig. 6. Ejemplo de un árbol en expansión de un grafo.

- **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 $G_1 \sim G_2$, existe si ambos grafos son topológicamente

idénticos, tal como se ve en la figura 7. En este ejemplo, el isomorfismo se define por $f(A)=1, f(B)=2, f(C)=3, f(D)=4, f(E)=5, g(x_1)=y_1, g(x_2)=y_2, g(x_3)=y_3, g(x_4)=y_4, g(x_5)=y_5$.

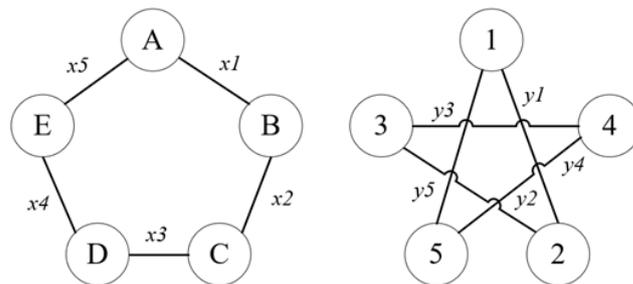


Fig. 7. Ejemplo de dos grafos isomorfos.

- **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) 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, matriz de incidencia, lista de adyacencia y representaciones canónicas.

a) 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 M_{ij} 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 $V = \{a_1, a_2, \dots, a_n\}$, 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 8, lo que no ocurre en el caso de los grafos dirigidos tal como se ve en la figura 9.

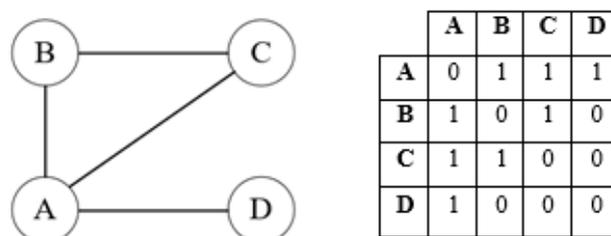


Fig. 8. Grafo no dirigido con su representación como matriz de adyacencia.

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 $|E| \ll |V|^2$, no es conveniente usar este tipo de representación debido a que se desperdicia una gran cantidad de memoria. 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.

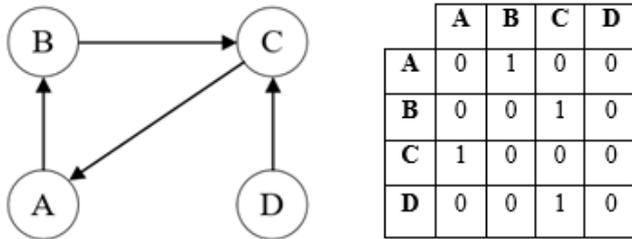


Fig. 9. Grafo dirigido con su representación como matriz de adyacencia.

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.

b) 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 $n \times m$ 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 10 puede observarse un ejemplo de esto. Las filas de la matriz representan a los vértices y las columnas a las aristas.

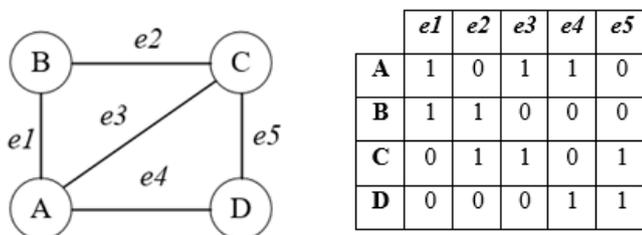


Fig. 10. Grafo no dirigido con su representación como matriz de incidencia.

c) Lista de Adyacencia

La representación con listas de adyacencia de un grafo $G = (V, E)$ cuyo conjunto de vértices es $V = \{a_1, a_2, \dots, a_n\}$, 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 11.

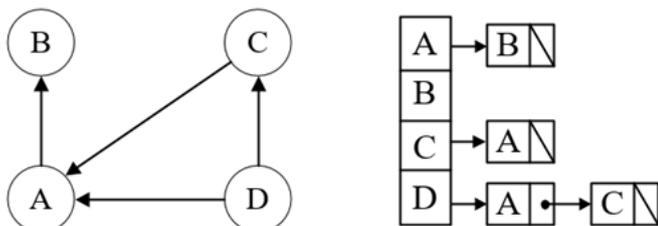


Fig. 11. Grafo dirigido con su representación como lista de adyacencia.

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.

d) 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 II.B.3.b) y se denota como $code(G, T)$. Cada arco del código se representa mediante una tupla con la siguiente forma: $(i, j, li, l(i,j), lj)$, donde i, j son vértices de G , li y lj 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 12. Dependiendo de cómo se haga el recorrido, un grafo puede tener varios códigos DFS, por lo que se usa en DFSM.

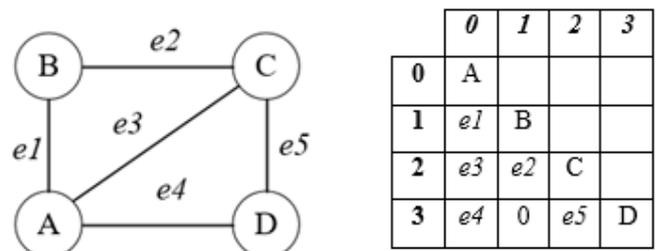


Fig. 12. Grafo con su representación como matriz de adyacencia CAM.

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 $min(G) = min(G')$ [6].

Para la utilización de la forma canónica CAM [7], 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 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: $\{Ae1Be3e2Ce40e5D\}$.

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.

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 II.B.3.a)

y recorrido en profundidad (Sección II.B.3.b). A continuación se introducen las nociones básicas de los mismos.

a) 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 13 se describe el pseudocódigo de un algoritmo que realiza la búsqueda en anchura en un grafo. Posteriormente, en la figura 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: $S=(v_1)$ // S es una lista ordenada y v_1 es la raíz del árbol abarcador
- 2: $V' = \{v_1\}$
- 3: $E' = \emptyset$
- 4: **REPETIR**
- 5: **PARA CADA** $x \in S$
- 6: **PARA CADA** $y \in V-V'$
- 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

Fig. 13. Algoritmo de búsqueda en anchura

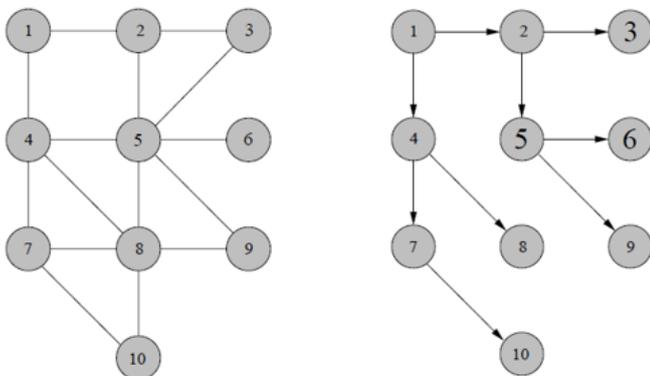


Fig. 14. Ejemplo de un grafo con su árbol abarcador encontrado a partir de la búsqueda en anchura.

b) 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.

En la figura 15 se muestra un árbol abarcador de un grafo generado a partir de la ejecución de DFS. En línea de trazos marca el momento en que no se encontraron caminos y se tuvo que retroceder al nivel anterior.

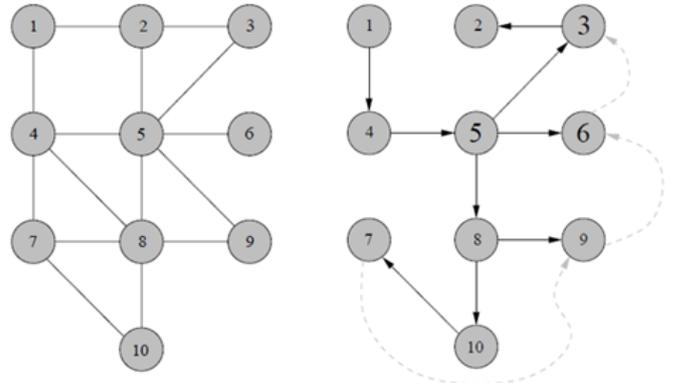


Fig. 15. Ejemplo de un grafo con su árbol abarcador encontrado a partir de la búsqueda en profundidad.

La implementación de este tipo de recorridos puede hacerse de manera recursiva como se observa en la figura 16. Este algoritmo ejemplifica la manera más sencilla de visitar todos los nodos de un grafo en profundidad.

Función: DFS-recursivo

Entrada: Grafo $G=(V, E)$, vértice v inicial

Salida: Todos los nodos de G visitados

- 1: VISITAR(v)
- 2: **PARA CADA** $w / (v, w) \in E$
- 3: DFS(G, w)
- 4: **RETURN**

Fig. 16. Algoritmo de búsqueda en profundidad recursivo

C. 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 [8]. 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 [9], 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 [9] 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.

1) Descubrimiento de Reglas de Comportamiento

El proceso de descubrimiento de reglas de comportamiento [9] 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 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.

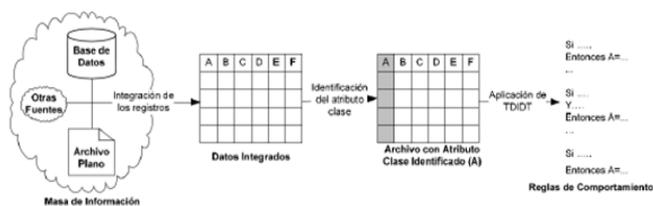


Fig. 17. Proceso de descubrimiento de reglas de comportamiento.

2) Descubrimiento de Grupos

El proceso de descubrimiento de grupos [9] 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 18.

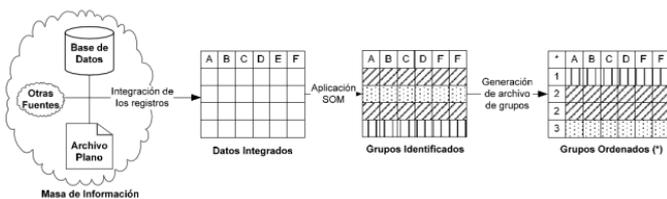


Fig. 18. Proceso de descubrimiento de grupos.

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.

3) Ponderación de Interdependencia de Atributos

El proceso de ponderación de interdependencia de atributos [9] 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 19.

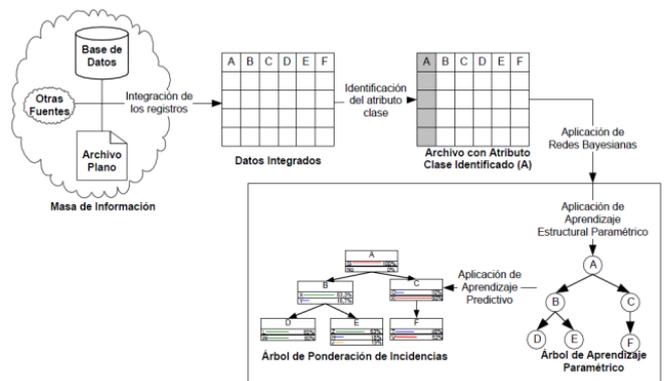


Fig. 19. Proceso de ponderación de interdependencia de atributos.

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 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.

4) Descubrimiento de Reglas de Pertenencia a Grupos

El proceso de descubrimiento de reglas de pertenencia a grupos [9] 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 20.

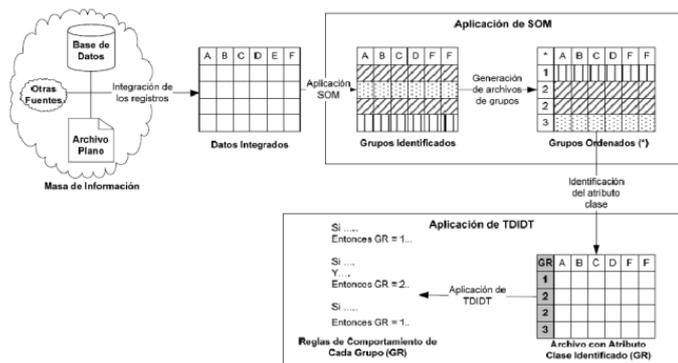


Fig. 20. Proceso de descubrimiento de reglas de pertenencia a grupos.

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 identificado. 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.

5) Ponderación de reglas de Comportamiento o de Pertenencia a Grupos

El proceso de ponderación de reglas de comportamiento o de pertenencia a grupos [9] 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 21.

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 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 figura 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.

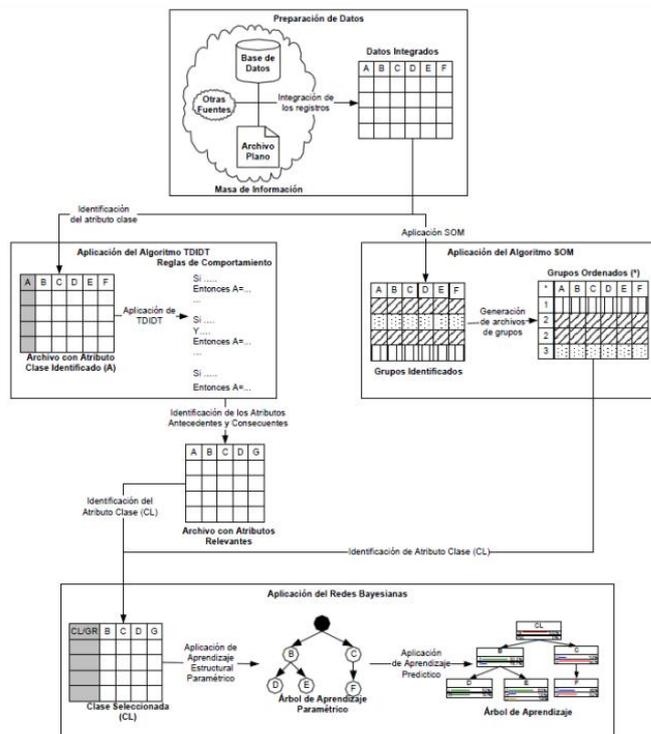


Fig. 21. Proceso de ponderación de reglas de comportamiento o de pertenencia a grupos.

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 se describirá con más detalle en la Sección II.D.

D. Explotación de Información en Grafos

Por lo expuesto en la Sección II.A 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 II.C, 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 [10] 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 [11] es el más conocido.

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

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:

$$\text{Sup}_G = \frac{| \{ G' \in GD \mid G \subseteq G' \} |}{|GD|}$$

Teniendo en cuenta esto, un grafo G es frecuente en una base de datos de grafos si $\text{sup}_G \geq \sigma$, 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 $\text{sup}_G \geq \sigma$.

A continuación se describen las características distintivas de los algoritmos de FSM que se evalúan en el presente artículo, ordenados en orden cronológico en orden ascendente según sus fechas de publicación: FSG (Sección II.D.1.a), gSpan (Sección II.D.1.b), FFSM (Sección II.D.1.c) y GASTON (Sección II.D.1.d).

a) Algoritmo FSG

El algoritmo FSG [12] es el algoritmo más antiguo de los que se evalúan en esta investigación, por lo que introdujo algunas características que luego serían usadas por el resto, que tienen que ver con la forma de representación de los grafos, la generación de subestructuras candidatas 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 implementarlas 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 22 se presenta el pseudocódigo de la ejecución general de la rutina principal del algoritmo FSG. En la tabla 1 se resume la notación utilizada.

TABLA I. NOTACIÓN UTILIZADA PARA LA DESCRIPCIÓN DEL ALGORITMO FSG [12]

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

Algoritmo: FSG

Entrada: D, σ // base de Grafos y minimum 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  $F^{k-1} \neq \emptyset$  HACER
5:    $C^k =$  candidatos-fsg( $F^{k-1}$ )
6:   PARA CADA  $g^k \in C^k$  HACER
7:      $g^k.$ conteo = 0
8:   PARA CADA  $t \in D$  HACER
9:     SI  $g^k$  esta incluido en t ENTONCES
10:        $g^k.$ conteo =  $g^k.$ conteo+1
11:    $F^k = \{ g^k \in C^k \mid g^k.$ conteo  $\geq \sigma \}$ 
12:    $k = k + 1$ 
13: RETURN  $\{ F^1, F^2, \dots, F^{k-2} \}$ 

```

Fig. 22. Rutina principal del algoritmo FSG [12]

b) Algoritmo gSpan

El algoritmo gSpan (graph-based Substructure pattern mining) [6] 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 II.B.2.c).

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 23 se presenta el pseudocódigo correspondiente a la subrutina de búsqueda de patrones del algoritmo gSpan y en la figura 24 la rutina principal. En la tabla 2 se resume la notación utilizada.

TABLA II. NOTACIÓN UTILIZADA PARA LA DESCRIPCIÓN DEL ALGORITMO GSPAN.

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

Subrutina: buscar_subgrafos

Entrada: D, S, s

Salida: Conjunto de subgrafos frecuentes (almacenados en S)

```

1: SI  $s \neq min(s)$  ENTONCES
2:   RETURN
3:  $S = S \cup \{s\}$ 
4: enumerar  $s$  in cada grafo en  $D$  y contar sus hijos
5: PARA CADA hijo de  $c$  de  $s$  HACER
6:   SI  $support(c) \geq minSup$  ENTONCES
7:      $s = c$ 
8:     buscar_subgrafos( $D_s, S, s$ )

```

Fig. 23. Subrutina de búsqueda de subgrafos de gSpan [6].

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

```

Fig. 24. Rutina principal del algoritmo gSpan [6].

c) Algoritmo FFSM

El algoritmo FFSM (Fast Frequent Subgraph Mining) [7] 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 II.B.2.c) 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 25 se muestra el proceso de inicialización para el algoritmo FFSM y en la figura 26 se describe el pseudocódigo del proceso de búsqueda de patrones. 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.

Algoritmo: FFSM

Entrada: conjunto de grafos GD y un support σ

Salida: Conjunto de subgrafos frecuentes (almacenados en S)

- 1: $S = \{\text{códigos CAM de los vértices y aristas frecuentes}\}$
 - 2: $P = \{\text{códigos CAM de las aristas frecuentes}\}$
 - 3: FFSM-explore(P, S)
-

Fig. 25. Proceso de inicialización del algoritmo FFSM [7].

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

Fig. 26. Pseudocódigo del proceso de búsqueda de patrones del algoritmo FFSM [7].

d) Algoritmo GASTON

El algoritmo GASTON (GrAph/Sequence/Tree extractiON) [13] 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 27 se describe el pseudocódigo del proceso de búsqueda de caminos, en la figura 28 la búsqueda de árboles y en la figura 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  $r \in R$  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:        $S = S \cup \{\text{nuevoCiclo}(n)\}$ 
6:     SINO SI  $r$  convierte a  $n$  en un árbol válido ENTONCES
7:        $S = S \cup \{\text{nuevoArbol}(n)\}$ 
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:       $S = S \cup \{\text{nuevoCamino}(n)\}$ 
11:      Gaston_buscar_caminos( $n$ , extender_y_unificar_caminos( $r, R, S$ ))

```

Fig.27. Búsqueda de caminos del algoritmo GASTON [14].

Algoritmo: Gaston_buscar_arboles

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

Salida: Conjunto de estructuras frecuentes en S

```

1: PARA CADA  $r \in R$  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:        $S = S \cup \{\text{nuevoGrafo}(n)\}$ 
6:       Gaston_buscar_grafos( $n, \{r' \in R / r' > r\}, S$ )
7:     SINO SI  $r$  convierte a  $n$  en un árbol válido ENTONCES
8:        $S = S \cup \{\text{nuevoArbol}(n)\}$ 
9:       Gaston_buscar_arboles( $n$ , extender_y_unificar_arboles( $r, R, S$ ))

```

Fig. 28. Búsqueda de árboles del algoritmo GASTON [14].

Algoritmo: Gaston_buscar_grafos

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

Salida: Conjunto de estructuras frecuentes en S

```

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

```

Fig. 29. Búsqueda de grafos del algoritmo GASTON [14].

III. DESCRIPCIÓN DEL PROBLEMA

En este capítulo se presenta el problema de investigación a desarrollar en el presente artículo. Primero se identifica y describe el problema de investigación (Sección III.A), luego se

caracteriza el problema abierto (Sección III.B) y finalmente se desarrolla el sumario de investigación correspondiente (Sección III.C).

A. Identificación del Problema 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 30. Agregando etiquetas a cada elemento se puede añadir información adicional del tipo de comunicación o características particulares de interés de los nodos.

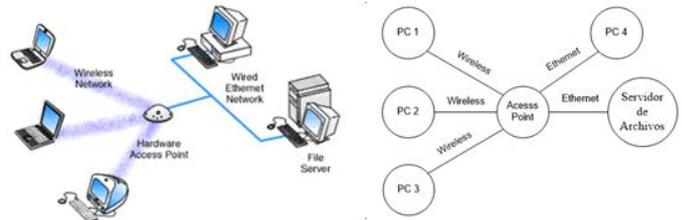


Fig. 30. Ejemplo de red de comunicación con componentes electrónicos y su posible modelización utilizando grafos.

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 [15], búsqueda de isomorfismos [11] o búsqueda de patrones entre varios grafos.

El proceso de interés para esta investigación 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 comprobar la frecuencia de determinado subgrafo. De por sí, solamente este proceso se encuadra dentro de los problemas NP-Complejos [16]. 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 [6; 25; 17]. 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 31. 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.

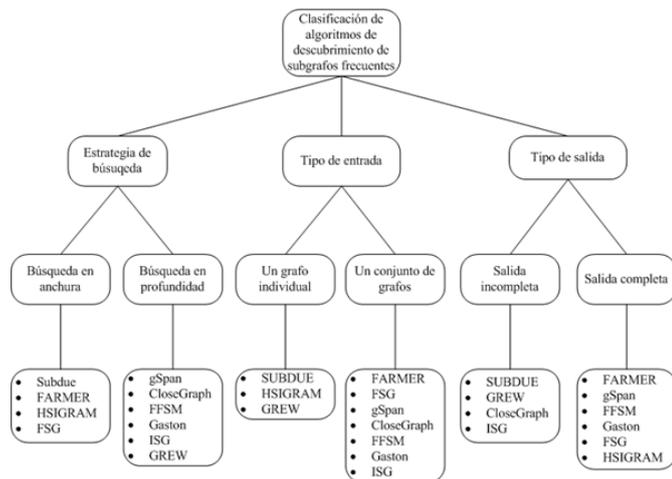


Fig. 31. Clasificación de los algoritmos más relevantes para la búsqueda de patrones en grafos [18].

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.

B. 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 [19].

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 [20].

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 [21]. 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 [18; 22], 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.

C. Resumen de investigación

De lo anteriormente expuesto surgen las siguientes preguntas de investigación:

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?
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?

IV. 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 IV.A). Finalmente se describen detalles de la construcción del banco de pruebas (Sección IV.B), incluyendo el generador de grafos usado (Sección IV.B.1), características de las implementaciones de los algoritmos que se comparan (Sección IV.B.2), como se adaptaron al banco de pruebas (Sección IV.B.3) y por último, un detalle de cómo serán ejecutados los experimentos y cómo se relacionan los distintos elementos del banco de pruebas (Sección IV.B.4) y cómo se evalúan y comparan los resultados (Sección IV.C).

A. 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 IV.A.1). 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 IV.A.2).

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 IV.A.2.a). 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 IV.A.2.a).

- 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 [12; 6]. Sus características pueden verse en la tabla III. Sobre esta base de datos se ejecutan pruebas cien veces y se promedian los resultados.

TABLA III. CARACTERÍSTICAS DEL ARCHIVO A UTILIZAR EN LAS PRUEBAS CON GRAFOS REALES

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

2) Variables

En las siguientes secciones se describen las variables independientes y dependientes que se utilizarán para cada tipo de experimento: con grafos sintéticos (Sección IV.A.2.a) y grafos reales (Sección IV.A.2.b).

a) 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.

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 la tabla IV. 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 la tabla IV. La cantidad de arcos se denota como $|E|$.

TABLA IV. VALORES DE LA VARIABLE CANTIDAD DE ARCOS EN CADA EXPERIMENTO CORRESPONDIENTE A LAS PRUEBAS CON GRAFOS SINTÉTICOS

Incremento Fijo				Incremento Variable			
Exp.	E	Exp.	E	Exp.	E	Exp.	E
1	10	26	60	1	10	26	60
2	15	27	65	2	15	27	90
3	20	28	70	3	20	28	120
4	25	29	75	4	25	29	150
5	30	30	80	5	30	30	180
6	20	31	70	6	20	31	70
7	25	32	75	7	30	32	105
8	30	33	80	8	40	33	140
9	35	34	85	9	50	34	175
10	40	35	90	10	60	35	210
11	30	36	80	11	30	36	80
12	35	37	85	12	45	37	120
13	40	38	90	13	60	38	160
14	45	39	95	14	75	39	200
15	50	40	100	15	90	40	240
16	40	41	90	16	40	41	90
17	45	42	95	17	60	42	135
18	50	43	100	18	80	43	180
19	55	44	105	19	100	44	225
20	60	45	110	20	120	45	270
21	50	46	100	21	50	46	100
22	55	47	105	22	75	47	150
23	60	48	110	23	100	48	200
24	65	49	115	24	125	49	250
25	70	50	120	25	150	50	300

- *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 Mínimo*: 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.

A su vez, 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.

TABLA IV. VALORES DE LA VARIABLE CANTIDAD DE ARCOS EN CADA EXPERIMENTO CORRESPONDIENTE A LAS PRUEBAS CON GRAFOS SINTÉTICOS

Incremento Fijo		Incremento Variable	
Exp.	E	Exp.	E
1	10	26	60
2	15	27	65
3	20	28	70
4	25	29	75
5	30	30	80
6	20	31	70
7	25	32	75
8	30	33	80
9	35	34	85
10	40	35	90
11	30	36	80
12	35	37	85
13	40	38	90
14	45	39	95
15	50	40	100
16	40	41	90
17	45	42	95
18	50	43	100
19	55	44	105
20	60	45	110
21	50	46	100
22	55	47	105
23	60	48	110
24	65	49	115
25	70	50	120

b) *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.

Para estas pruebas se considera únicamente a la siguiente variable independiente:

- *Minimum Support Threshold o Umbral de Frecuencia Mínimo*: 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%.

Además, 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.

B. *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 IV.B.1), las implementaciones de los algoritmos de búsqueda de patrones en grafos que se utilizan (Sección IV.B.2), cómo se adaptaron estas implementaciones al banco de pruebas (Sección IV.B.3) y el proceso de ejecución de los experimentos (Sección IV.B.4).

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 [23]. 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 32.

```

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
Salida: G // Gn,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

```

Fig. 32. Algoritmo generador de grafos aleatorios provisto por la librería Networkx.

Toda la estructura de la librería esta implementada con el paradigma orientado a objetos [24], por lo que la estructura principal de los grafos está implementada con clases

TABLA V. CARACTERÍSTICAS DE LAS IMPLEMENTACIONES DE LOS ALGORITMOS

Algoritmo	Parámetros	Entrada	Salida
FFSM	<i>Support</i> : entero positivo con valores entre 0 y 100. Determina el umbral de frecuencia mínima. <i>Density</i> : densidad mínima del grafo de salida. <i>SizeLimit</i> : la cantidad mínima de nodos de los patrones. <i>SizeupLimit</i> : la cantidad máxima de nodos de los patrones.	Dos archivos: 1. <i>NodeFile</i> : archivo con los nodos de las estructuras. 2. <i>EdgeFile</i> : archivo con los arcos de las estructuras.	Tres archivos: <i>OutNodeFile</i> : nodos de los patrones encontrados. <i>OutEdgeFile</i> : arcos de los patrones encontrados <i>OutFeatureFile</i> : estadísticas de los subgrafos encontrados.
FSG	<i>Support</i> : entero positivo con valores entre 0 y 100. Determina el umbral de frecuencia mínima. <i>Minsize</i> : entero positivo que indica la cantidad mínima de arcos de las estructuras buscadas. <i>Maxsize</i> : entero positivo que indica la cantidad máxima de arcos de las estructuras buscadas. <i>Maximal</i> : Si está habilitado, busca solo subgrafos frecuentes máximos [25].	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	<i>Support</i> : 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	<i>Support</i> : 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.

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 33.

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

```

Fig. 33. Implementación del algoritmo generador de grafos no dirigidos

2) Implementaciones de los Algoritmos de Minería de Grafos

Las implementaciones de los algoritmos no se desarrollan desde cero sino que se utilizan aquellas desarrolladas por los autores de los mismos [12; 6; 7; 13]. 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 V 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.

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 34. 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 símbolo numeral (#) 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.

La implementación del FFSM, en cambio, necesita dos archivos: uno con los nodos y otros con los arcos, como se observa en la figura 35. 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.

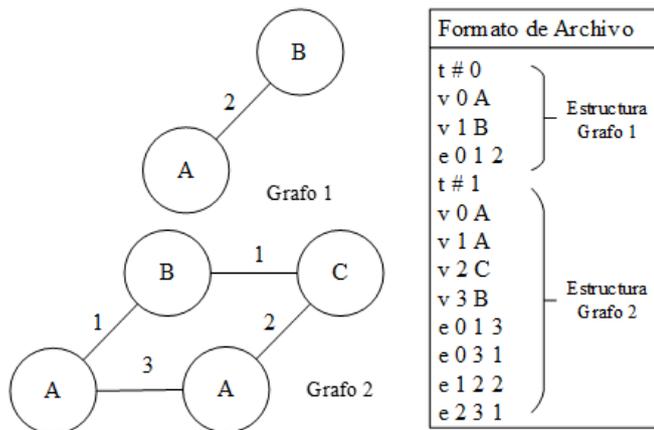


Fig. 34. Formato de los datasets para las implementaciones de FSG, gSpan y GASTON.

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.

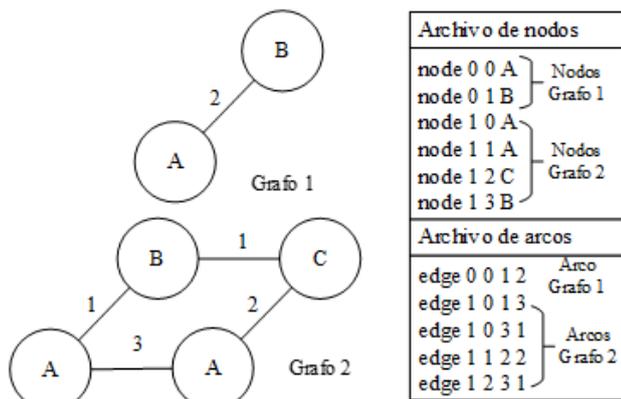


Fig. 35. Formato de los dataset para la implementación del algoritmo FFSM.

El dataset Compounds_422 utilizado para los experimentos con grafos reales contiene el formato de la figura 34. Para utilizar el algoritmo FFSM, se debe modificar su estructura para que concuerde con la descrita en la figura 35.

Para ejecutar los algoritmos en el caso de las pruebas con grafos sintéticos se utiliza la configuración de parámetros presente en la Tabla VI. 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.

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 IV.A.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 VII.

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.

TABLA VI. CONFIGURACIÓN DE LOS PARÁMETROS DE CADA ALGORITMO PARA LOS EXPERIMENTOS CON GRAFOS SINTÉTICOS

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 VII. CONFIGURACIÓN DEL PARÁMETRO SUPPORT DE CADA ALGORITMO PARA LOS EXPERIMENTOS CON GRAFOS REALES

Prueba	Support	
	FFSM	FSG
		gSpan
1	25	0,25
2	30	0,30
3	35	0,35
4	40	0,40
5	45	0,45
6	50	0,50
7	55	0,55
8	60	0,60
9	65	0,65
10	70	0,70
11	75	0,75
12	80	0,80
13	85	0,85
14	90	0,90
15	95	0,95

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 IV.B.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 36. 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.

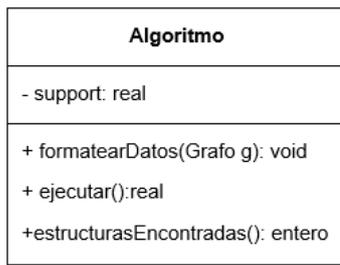


Fig. 36. Ejemplo de clase para adaptar un algoritmo al banco de pruebas.

A grandes rasgos, el procedimiento seguido para poder ejecutar un algoritmo en el banco de pruebas es el que se resume en la figura 37.

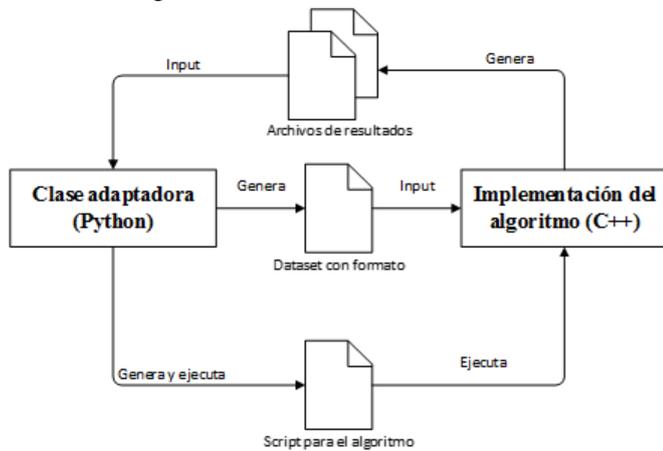


Fig. 37. Ejemplo de ejecución de un algoritmo para una prueba determinada.

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) 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 (IV.B).

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 comprensión, 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 38. 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.

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.

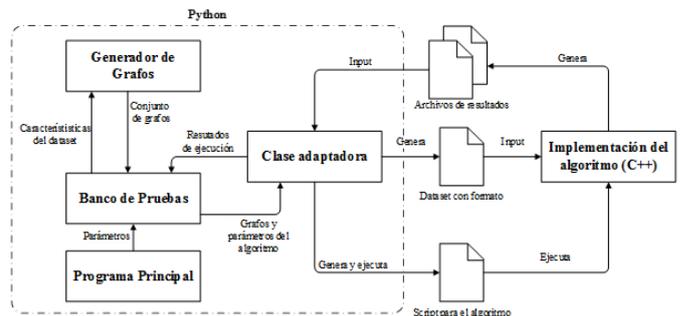


Fig. 38. Ejecución de una prueba para experimentos con grafos sintéticos.

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 para los experimentos con grafos reales es de la figura 39.

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.

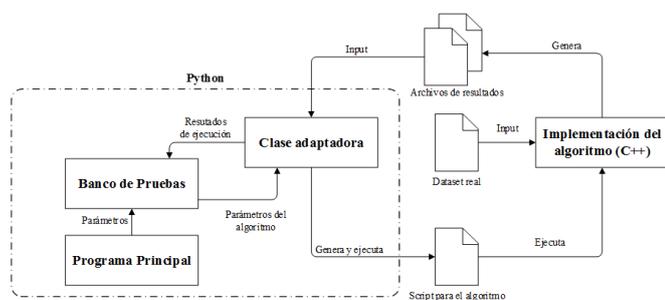


Fig. 39. Ejecución de una prueba para los experimentos con grafos reales.

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

C. 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 cuán-tas 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 VIII puede verse un ejemplo del formato de una tabla de resultados para un determinado tipo de experimento. Se incluye el número de experimento y la cantidad de estructuras encontradas en cada uno por cada algoritmo, junto con el tiempo de ejecución de cada uno expresado en segundos. En base al análisis de estos datos se compara el comportamiento de cada algoritmo en cada escenario diseñado.

V. 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 V.A) y luego se muestran los resultados de las pruebas con grafos sintéticos (sección V.B), los grafos sintéticos con repetición de nodos

(Sección V.C) y finalmente la las pruebas con grafos reales (Sección V.D).

TABLA VIII. FORMATO DE TABLA DE RESULTADOS QUE SE GENERA A PARTIR DE CADA EXPERIMENTO

Exp.	GASTON		gSpan		FFSM		FSG		
	#	Estr.	T (s)	Estr.	T (s)	Estr.	T(s)	Estr.	T(s)
1									
2									
...									
49									
50									

A. 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 V.B y V.C. 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 *minimum support* para los algoritmos siguiendo el criterio que se explica en la sección V.D.

B. 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 VIII, 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 n 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.

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.

TABLA IX. CONFIGURACIÓN DE CADA EXPERIMENTO PARA LAS PRIMERAS PRUEBAS CON GRAFOS SINTÉTICOS SIN REPETICIÓN DE NODOS

Exp.	V	E	Exp.	V	E
1	10	10	26	60	60
2	10	15	27	60	65
3	10	20	28	60	70
4	10	25	29	60	75
5	10	30	30	60	80
6	20	20	31	70	70
7	20	25	32	70	75
8	20	30	33	70	80
9	20	35	34	70	85
10	20	40	35	70	90
11	30	30	36	80	80
12	30	35	37	80	85
13	30	45	38	80	90
14	30	45	39	80	95
15	30	50	40	80	100
16	40	40	41	90	90
17	40	45	42	90	95
18	40	50	43	90	100
19	40	55	44	90	105
20	40	60	45	90	110
21	50	50	46	100	100
22	50	55	47	100	105
23	50	60	48	100	110
24	50	65	49	100	115
25	50	70	50	100	120

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 X. Puede verse que la relación es mucho más constante para tratar de averiguar si es verdaderamente influyente en los resultados.

Los resultados correspondientes a las pruebas con incremento fijo en la cantidad de nodos pueden apreciarse en la tabla XI, mientras que los resultados de las pruebas con incremento variable se resumen en la tabla XII. 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 40 y 41 se grafican los resultados de la tabla X. En la figura 40, se muestra la cantidad de subestructuras encontradas en cada experimento, mientras que en la figura 41 se observa el tiempo de ejecución de cada algoritmo en cada prueba. Contrastando ambas figuras, 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.

TABLA X. CONFIGURACIÓN DE CADA EXPERIMENTO PARA LA SEGUNDA ETAPA PRUEBAS CON GRAFOS SINTÉTICOS SIN REPETICIÓN DE NODOS

Exp.	V	E	Exp.	V	E
1	10	10	26	60	60
2	10	15	27	60	90
3	10	20	28	60	120
4	10	25	29	60	150
5	10	30	30	60	180
6	20	20	31	70	70
7	20	30	32	70	105
8	20	40	33	70	140
9	20	50	34	70	175
10	20	60	35	70	210
11	30	30	36	80	80
12	30	45	37	80	120
13	30	60	38	80	160
14	30	75	39	80	200
15	30	90	40	80	240
16	40	40	41	90	90
17	40	60	42	90	135
18	40	80	43	90	180
19	40	100	44	90	225
20	40	120	45	90	270
21	50	50	46	100	100
22	50	75	47	100	150
23	50	100	48	100	200
24	50	125	49	100	250
25	50	150	50	100	300

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 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.

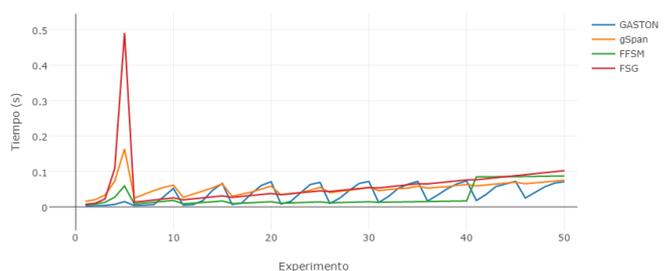


Fig. 40. Tiempo de ejecución en segundos de los algoritmos en cada experimento para las primeras pruebas con grafos sintéticos sin repetición de nodos.

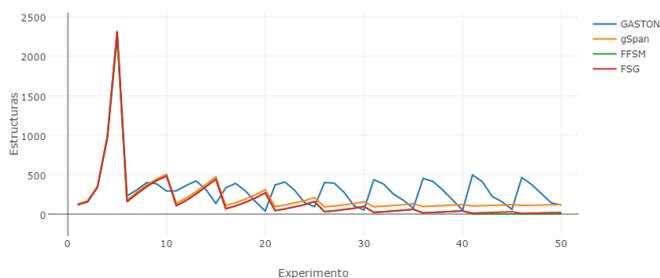


Fig. 41. Cantidad de estructuras encontradas en cada experimento para las primeras pruebas con grafos sintéticos sin repetición de nodos.

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 IX. Estos resultados se ven graficados en las figuras 42 y 43, generadas a partir de la tabla XI.

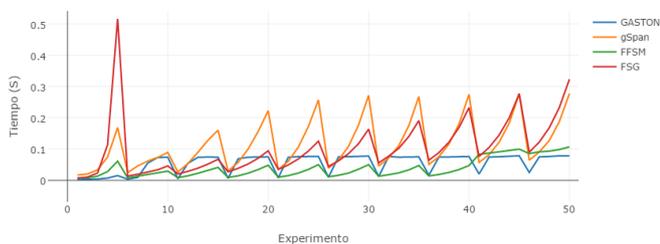


Fig. 42. Tiempo de ejecución en segundos de los algoritmos en cada experimento para las segundas pruebas con grafos sintéticos sin repetición de nodos.

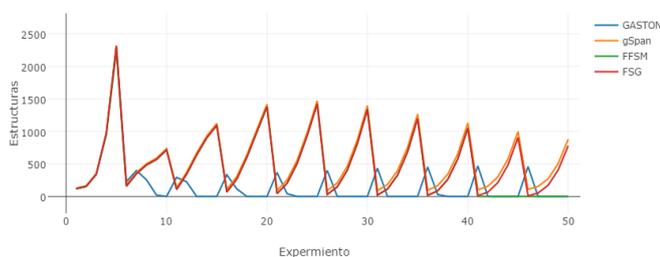


Fig. 43. Cantidad de subestructuras encontradas en cada experimento para las segundas con grafos sintéticos sin repetición de nodos.

Analizando los resultados representados por las figuras 42 y 43, 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.

C. 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, H₂O, 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 44.

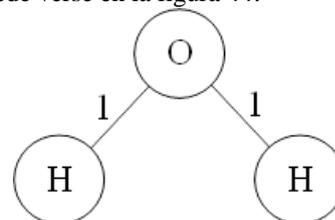


Fig. 44. Ejemplo de construcción de una molécula de agua con estructura de grafo.

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 XIII y XIV. La tabla XIII representa a las pruebas con variación de arcos fija y la XIV 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 IX y X de la sección anterior es una columna llamada nodos únicos. Este valor representa la cantidad máxima de valores distintos de nodos que puede tener un grafo. Por ejemplo, prueba 1 de la tabla XIII 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.

Los resultados correspondientes a las pruebas con un incremento fijo en la cantidad de nodos pueden apreciarse en la tabla XV, mientras que los resultados de las pruebas con incremento variable se resumen en la tabla XVI. 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.

TABLA XI. RESULTADOS DE LOS EXPERIMENTOS PARA LAS PRIMERAS PRUEBAS CON GRAFOS SINTÉTICOS SIN REPETICIÓN DE NODOS

Incremento Fijo								
<i>Exp</i>	<i>GASTON</i>		<i>gSpan</i>		<i>FFSM</i>		<i>FSG</i>	
#	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
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	0.041218	110.92	0.067409	0.0	0.0855402	10.92	0.0930588
48	259.21	0.056597	114.04	0.07041	0.0	0.085972	14.04	0.09587859
49	141.12	0.067091	116.62	0.071919	0.0	0.0864788	16.62	0.09881939
50	114.43	0.070925	119.97	0.0752025	0.0	0.0872213	19.97	0.10213557

TABLA XII. RESULTADOS DE LOS EXPERIMENTOS PARA LA SEGUNDA ETAPA PRUEBAS CON GRAFOS SINTÉTICOS SIN REPETICIÓN DE NODOS.

Exp.	Incremento Variable							
	GASTON		gSpan		FFSM		FSG	
	#	Estructuras	Tiempo (s)	Estructuras	Tiempo (s)	Estructuras	Tiempo (s)	Estructuras
1	118.01	0.002828	128.01	0.01687067	117.99	0.006532754	118.01	0.0076662
2	155.28	0.003127	165.28	0.02016572	154.55	0.00801261	155.28	0.0104978
3	343.53	0.004143	353.53	0.03326593	337.11	0.013275232	343.53	0.0232644
4	971.03	0.007111	981.03	0.07446077	946.54	0.02795877	971.03	0.1137017
5	2312.33	0.015257	2322.33	0.16886492	2234.35	0.0610397	2312.33	0.5161214
6	228.13	0.0035515	176.64	0.024693126	156.64	0.008127362	156.64	0.01370598
7	397.67	0.009304	368.14	0.04586393	348.14	0.01326831	348.14	0.01929100
8	256.67	0.05498	502.88	0.061620399	482.77	0.01828966	482.88	0.02577618
9	20.27	0.073056	589.46	0.07335108	568.81	0.02292137	569.46	0.03376764
10	0.0	0.073893	735.66	0.089208981	712.93	0.02943686	715.66	0.04610991
11	293.49	0.0049399	136.78	0.026590945	106.78	0.008648474	106.78	0.02053321
12	224.43	0.054185	372.46	0.053627724	342.46	0.014469621	342.46	0.02869054
13	0.0	0.0735993	668.53	0.08966207	638.53	0.022522246	638.53	0.03886919
14	0.0	0.074303	929.09	0.1280997	899.05	0.03208657	899.09	0.05187945
15	0.0	0.07388	1122.6	0.1602091	1092.44	0.04127605	1092.6	0.06716967
16	332.62	0.006281	107.54	0.02944285	67.54	0.009130637	67.54	0.02716793
17	115.07	0.0688369	308.33	0.05732474	268.33	0.014514	268.33	0.03823584
18	0.0	0.073873	641.05	0.100125	601.05	0.022906575	601.05	0.05228443
19	0.0	0.0742107	1033.58	0.15621268	993.58	0.034460775	993.58	0.07060092
20	0.0	0.074829	1418.43	0.22276423	1378.41	0.04834433	1378.43	0.094418969
21	368.01	0.007917	94.08	0.034485366	44.08	0.01003717	44.08	0.03492777
22	42.81	0.0737658	251.11	0.0611014	201.11	0.01512068	201.11	0.04916209
23	0.0	0.0755659	554.47	0.1062329	504.47	0.02310528	504.47	0.06825963
24	0.0	0.0764062	981.52	0.17202708	931.52	0.03489588	931.52	0.09365606
25	0.0	0.0764193	1472.17	0.25701951	1422.15	0.050123782	1422.17	0.1253132
26	401.54	0.0096073	90.19	0.03883927	30.19	0.011392	30.19	0.04444644
27	0.0	0.0749072	207.17	0.06431194	147.17	0.016165091	147.17	0.06213757
28	0.0	0.07543205	463.71	0.10872693	403.71	0.02314172	403.71	0.08537832
29	0.0	0.0760219	870.18	0.1769249	810.18	0.0355678	810.18	0.1172985
30	0.0	0.0775315	1397.01	0.27138744	1337.01	0.05007874	1337.01	0.1635236
31	435.72	0.0122244	91.45	0.04571491	21.45	0.0131956	21.45	0.05557765
32	0.0	0.0761942	183.29	0.0711250	113.29	0.01794831	113.29	0.07729588
33	0.0	0.07388	397.07	0.110528	327.07	0.02386432	327.07	0.1022750
34	0.0	0.0748562	755.81	0.174342	685.81	0.03329685	685.81	0.1398108
35	0.0	0.074883	1265.76	0.2677131	1195.76	0.04731438	1195.76	0.19086
36	452.94	0.0163032	96.14	0.05074288	16.14	0.01406423	16.14	0.06418777
37	29.73	0.0743026	166.55	0.07549785	86.55	0.01862834	86.55	0.08858460
38	0.0	0.0745504	341.55	0.1161438	261.55	0.0251444	261.55	0.1221379
39	0.0	0.0752215	656.12	0.180074	576.12	0.03433792	576.12	0.1685983
40	0.0	0.0760226	1129.5	0.27463679	1049.5	0.0472849	1049.5	0.2323901
41	469.28	0.020052	101.88	0.0572447	0.0	0.08322418	11.88	0.07624510
42	0.0	0.074342	156.65	0.08203478	0.0	0.08704325	66.65	0.1050771
43	0.0	0.07519	303.6	0.1218183	0.0	0.09080163	213.6	0.1447607
44	0.0	0.0762297	572.91	0.1825641	0.0	0.09459184	482.91	0.1995426
45	0.0	0.0784369	997.14	0.2758583	0.0	0.09962319	907.14	0.277026
46	459.11	0.02502052	109.07	0.06442543	0.0	0.08540614	9.07	0.08979902
47	0.0	0.0749271	153.19	0.0904417	0.0	0.09116508	53.19	0.1223359
48	0.0	0.07603946	272.8	0.12745806	0.0	0.09346777	172.8	0.1682874
49	0.0	0.0783054	505.78	0.18790324	0.0	0.09829167	405.78	0.2329529
50	0.0	0.0783904	878.85	0.2774244	0.0	0.10694290	778.85	0.3231319

TABLA XIII. CONFIGURACIÓN DE CADA EXPERIMENTO PARA LAS PRIMERAS PRUEBAS CON GRAFOS SINTÉTICOS CON REPETICIÓN DE NODOS

Exp.	V	E	Nodos únicos	Exp.	V	E	Nodos únicos
	10	10	5	26	60	10	30
2	10	15	5	27	60	15	30
3	10	20	5	28	60	20	30
4	10	25	5	29	60	25	30
5	10	30	5	30	60	30	30
6	20	20	10	31	70	20	35
7	20	25	10	32	70	25	35
8	20	30	10	33	70	30	35
9	20	35	10	34	70	35	35
10	20	40	10	35	70	40	35
11	30	30	15	36	80	30	40
12	30	35	15	37	80	35	40
13	30	45	15	38	80	45	40
14	30	45	15	39	80	45	40
15	30	50	15	40	80	50	40
16	40	40	20	41	90	40	45
17	40	45	20	42	90	45	45
18	40	50	20	43	90	50	45
19	40	55	20	44	90	55	45
20	40	60	20	45	90	60	45
21	50	50	25	46	100	50	50
22	50	55	25	47	100	55	50
23	50	60	25	48	100	60	50
24	50	65	25	49	100	65	50
25	50	70	25	50	100	70	50

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

Nuevamente observando ambas tablas XV y XVI 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.

Para facilitar la comprensión de los resultados reflejados en la tabla XV se presentan las figuras 45, 46 y 47. En la figura 45 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 46 y 47.

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 ejecución. El algoritmo FFSG 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.

TABLA XIV. CONFIGURACIÓN DE CADA EXPERIMENTO PARA LAS SEGUNDAS PRUEBAS CON GRAFOS SINTÉTICOS CON REPETICIÓN DE NODOS

Exp.	V	E	Nodos únicos	Exp.	V	E	Nodos únicos
1	10	10	5	26	60	60	30
2	10	15	5	27	60	90	30
3	10	20	5	28	60	120	30
4	10	25	5	29	60	150	30
5	10	30	5	30	60	180	30
6	20	20	10	31	70	70	35
7	20	30	10	32	70	105	35
8	20	40	10	33	70	140	35
9	20	50	10	34	70	175	35
10	20	60	10	35	70	210	35
11	30	30	15	36	80	80	40
12	30	45	15	37	80	120	40
13	30	60	15	38	80	160	40
14	30	75	15	39	80	200	40
15	30	90	15	40	80	240	40
16	40	40	20	41	90	90	45
17	40	60	20	42	90	135	45
18	40	80	20	43	90	180	45
19	40	100	20	44	90	225	45
20	40	120	20	45	90	270	45
21	50	50	25	46	100	100	50
22	50	75	25	47	100	150	50
23	50	100	25	48	100	200	50
24	50	125	25	49	100	250	50
25	50	150	25	50	100	300	50

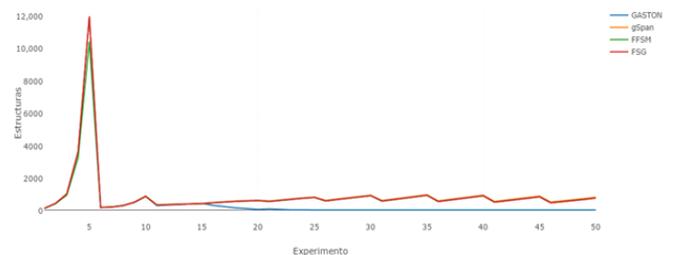


Fig. 45. Cantidad de estructuras encontradas en cada experimento para las primeras pruebas con grafos sintéticos con repetición de nodos.

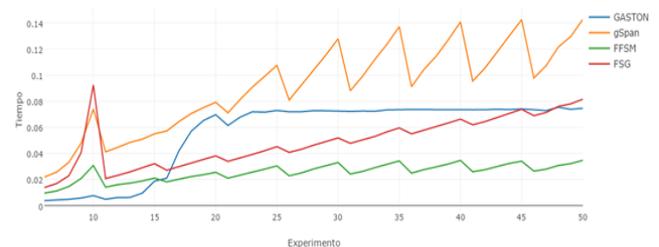


Fig. 46. Tiempo de ejecución de cada algoritmo a partir del experimento seis para las primeras pruebas con grafos sintéticos con repetición de nodos



Fig. 47. Cantidad de estructuras encontradas a partir del experimento seis para las primeras pruebas con grafos sintéticos con repetición de nodos.

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 48, 49 y 50. La figura 48 incluye la cantidad de subestructuras halladas en todos los experimentos. Las figuras 49 y 50 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 poder observar más claramente las diferencias entre los valores.

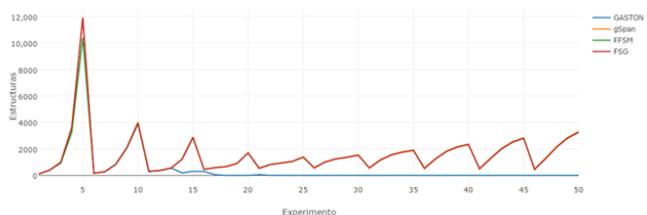


Fig. 48. Subestructuras encontradas en cada experimento para las segundas pruebas con grafos sintéticos con repetición de nodos.

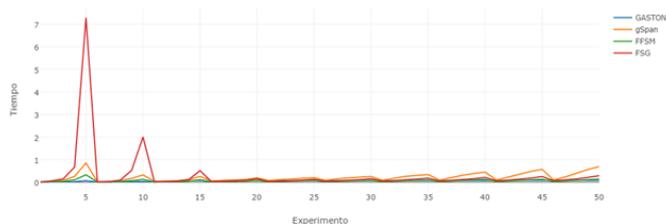


Fig. 49. Tiempo de ejecución de cada algoritmo en cada experimento para las segundas pruebas con grafos sintéticos con repetición de nodos.

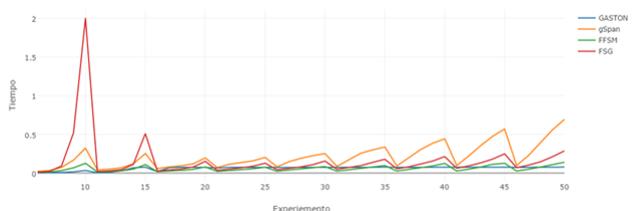


Fig. 50. Tiempo de ejecución de cada algoritmo a partir del experimento seis para las segundas pruebas con grafos sintéticos con repetición de nodos.

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 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).

D. 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 *minimum 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 XVII.

Estas pruebas se ejecutan cien veces cada una y los resultados se promedian. Los números finales pueden verse en la Tabla XVIII. 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 51 y 52 pueden verse gráficamente los resultados de la tabla XVIII, 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.

TABLA XV. 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)						
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
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 XVI. 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)						
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
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 XVIII. RESULTADOS DE LA EJECUCIÓN DE LOS ALGORITMOS PARA CADA EXPERIMENTO CON GRAFOS REALES

Exp.	GASTON		gSpan		FFSM		FSG	
	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 XVII. CONFIGURACIÓN DE LOS ALGORITMOS PARA CADA EXPERIMENTO CON GRAFOS REALES

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

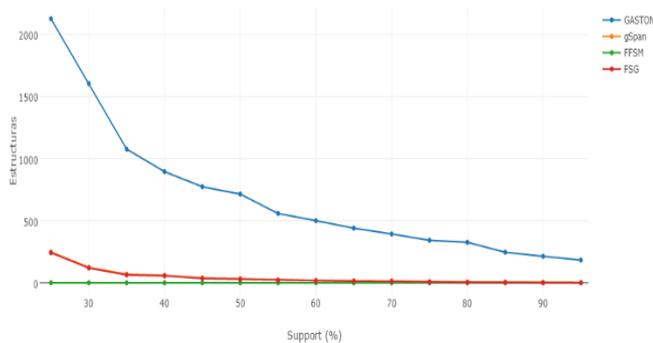


Fig. 51. Subestructuras encontradas por cada algoritmo en base al umbral mínimo para las pruebas con grafos reales

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 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.

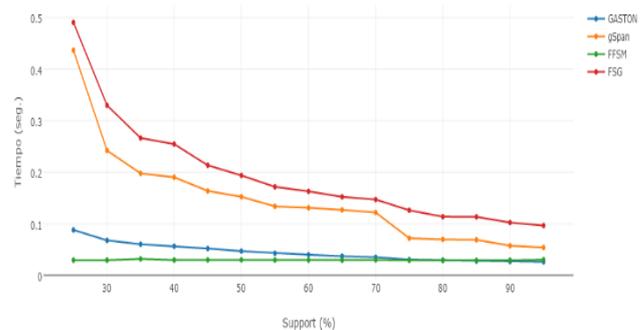


Fig. 52. Tiempo de ejecución por cada algoritmo en base al umbral mínimo para las pruebas con grafos reales

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.

VI. CONCLUSIONES

En esta sección se presentan las conclusiones obtenidas a partir de la investigación desarrollada para este artículo. A continuación se muestra un resumen de los resultados

obtenidos y se hace un análisis comparativo de los mismos (Sección VI.A), seguido de las conclusiones globales (Sección VI.B) y finalmente se describen las futuras líneas de investigación (Sección VI.C).

A. Análisis Comparativo de los Resultados Obtenidos

A lo largo del presente artículo 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 VI.A.1 se muestra el análisis de las pruebas con grafos sintéticos y en la sección VI.A.2 se presenta el análisis de las pruebas con grafos reales.

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 V), los resultados serán divididos en: pruebas con nodos únicos (Sección VI.A.1.a) y pruebas con nodos repetidos (Sección VI.A.1.b). Posteriormente se presentan conclusiones generales de todas las pruebas (Sección VI.A.1.c).

a) 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 53.

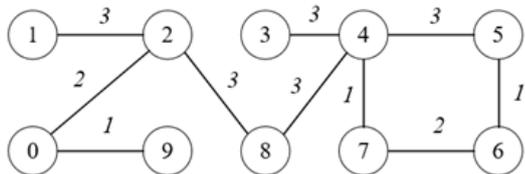


Fig. 53. Ejemplo de grafo para las pruebas con grafos sintéticos sin repetición de nodos.

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 XIX.

TABLA XIX. RESUMEN DE LA CONFIGURACIÓN DE LOS EXPERIMENTOS PARA LAS PRUEBAS CON GRAFOS SINTÉTICOS CON NODOS ÚNICOS

Orden	Exp.	V	E	Orden	Exp.	V	E
1	1	10	10	42	30	60	80
2	2	10	15	43	27	60	90
3	3	10	20	44	28	60	120
4	4	10	25	45	29	60	150
5	5	10	30	46	30	60	180
6	6	20	20	47	31	70	70
7	7	20	25	48	32	70	75
8	8	20	30	49	33	70	80
9	9	20	35	50	34	70	85
10	10	20	40	51	35	70	90
11	9	20	50	52	32	70	105
12	10	20	60	53	33	70	140
13	11	30	30	54	34	70	175
14	12	30	35	55	35	70	210
15	13	30	40	56	36	80	80
16	14	30	45	57	37	80	85
17	15	30	50	58	38	80	90
18	13	30	60	59	39	80	95
19	14	30	75	60	40	80	100
20	15	30	90	61	37	80	120
21	16	40	40	62	38	80	160
22	17	40	45	63	39	80	200
23	18	40	50	64	40	80	240
24	19	40	55	65	41	90	90
25	20	40	60	66	42	90	95
26	18	40	80	67	43	90	100
27	19	40	100	68	44	90	105
28	20	40	120	69	45	90	110
29	21	50	50	70	42	90	135
30	22	50	55	71	43	90	180
31	23	50	60	72	44	90	225
32	24	50	65	73	45	90	270
33	25	50	70	74	46	100	100
34	22	50	75	75	47	100	105
35	23	50	100	76	48	100	110
36	24	50	125	77	49	100	115
37	25	50	150	78	50	100	120
38	26	60	60	79	47	100	150
39	27	60	65	80	48	100	200
40	28	60	70	81	49	100	250
41	29	60	75	82	50	100	300

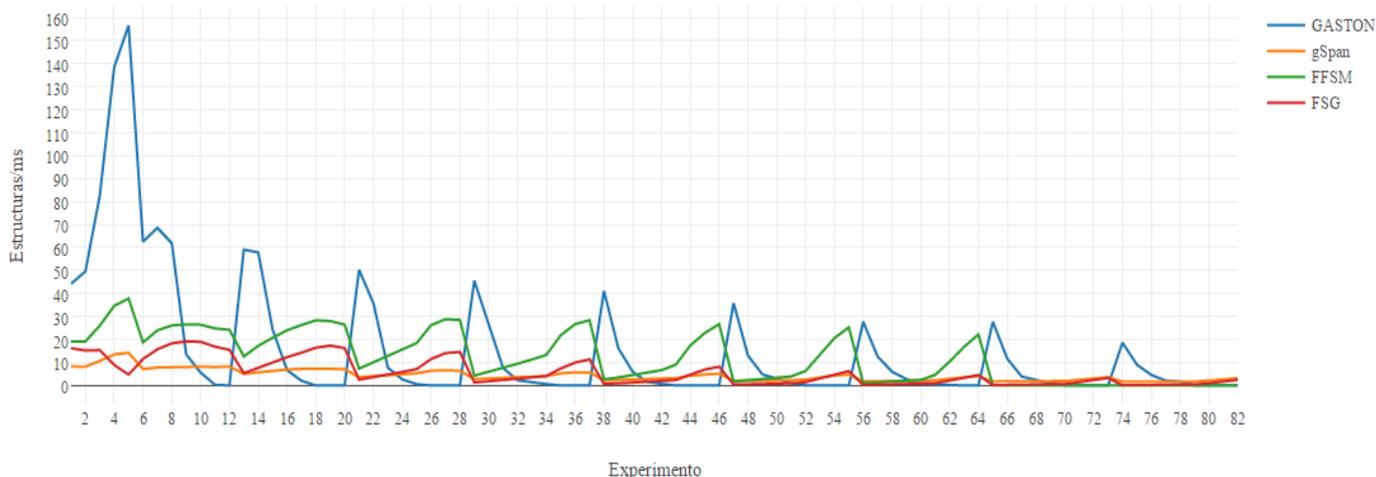


Fig. 54 Gráfico resumen de los resultados de los experimentos con grafos sintéticos con nodos únicos

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.

En la columna experimento se muestra el id original correspondiente a las tablas XIX y X. En la columna orden, se muestra el id que servirá como variable independiente para el gráfico comparativo de la figura 54. 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.

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.

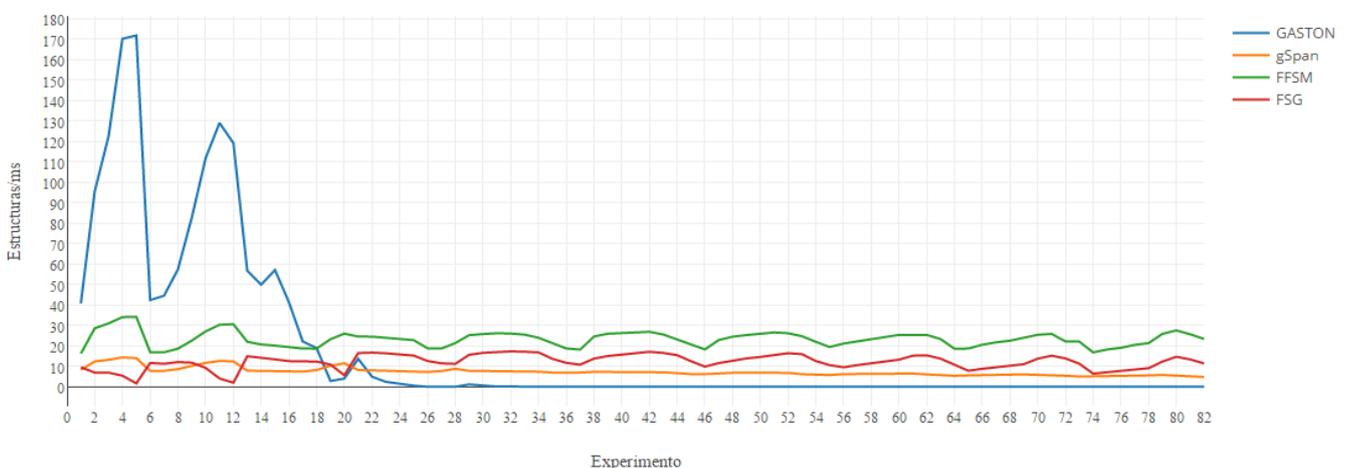


Fig. 56. Gráfico resumen de los resultados de los experimentos con grafos sintéticos con nodos repetidos.

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.

b) 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 55.

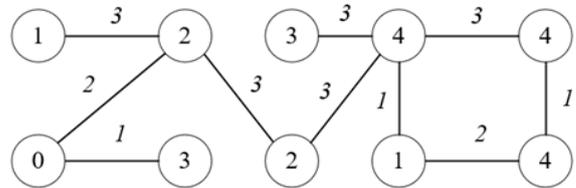


Fig. 55. Ejemplo de grafo para las pruebas con grafos sintéticos con repetición de nodos

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 56.

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 VI.A.1.a. 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.

c) 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.

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 [6; 7]. Las características de esta base se

muestran en la tabla XX. 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 IV.A.2.

TABLA XX. TABLA RESUMEN DE LA BASE DE DATOS USADA PARA LOS EXPERIMENTOS CON GRAFOS REALES

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

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 XIX.

En la figura 57 se presentan los resultados obtenidos, contrastando el minimum support de cada prueba con la cantidad de estructuras encontradas por milisegundo.

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 58. 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.

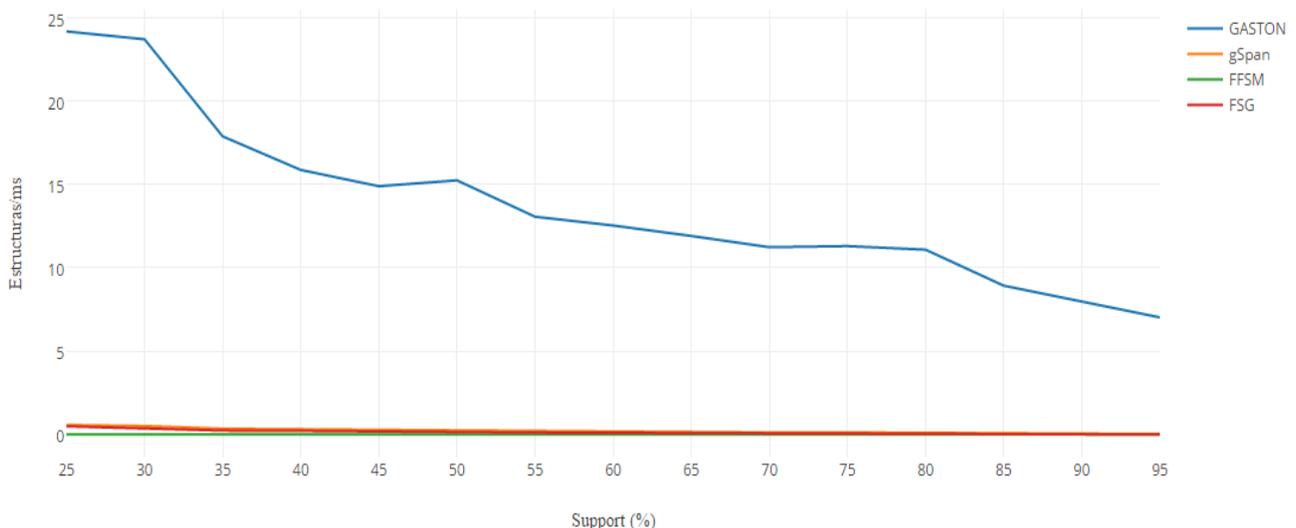


Fig. 57. Resultados para las pruebas con grafos reales

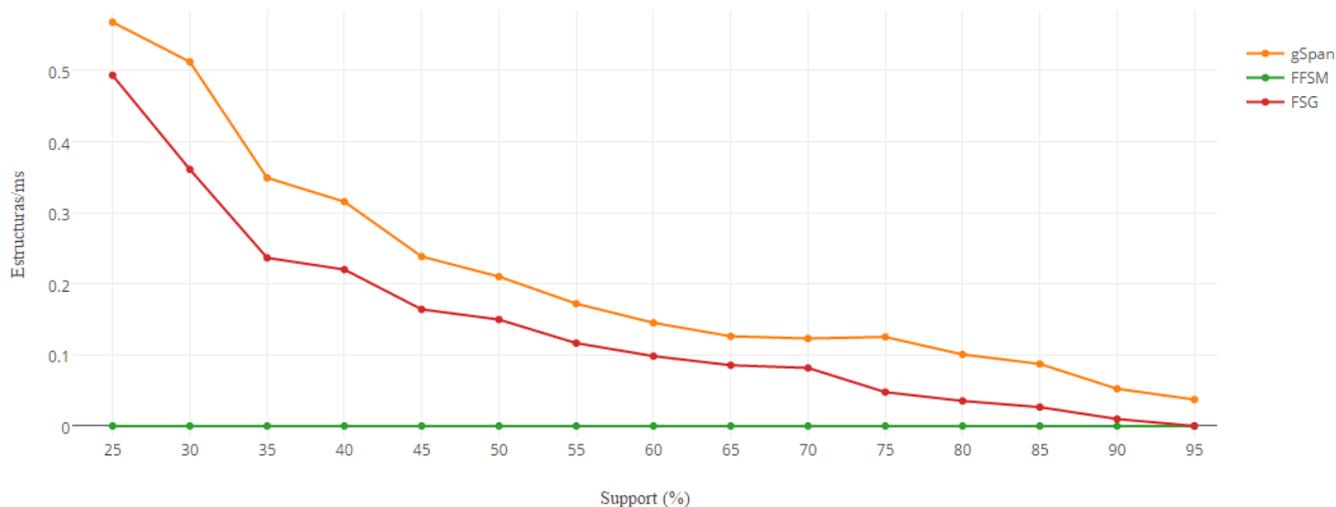


Fig. 58. Detalle de resultados para las pruebas con grafos reales de los algoritmos gSpan, FFSM y FSG

B. 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 alimentan 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.

C. Futuras Líneas de Investigación

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

1. Ampliar los algoritmos a evaluar incluyendo otros como el SPIN o CloseGraph para verificar su comportamiento.
2. 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.
3. 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.
4. Realizar comparaciones similares a las de esta investigación para otros tipos de algoritmos de minería de grafos, como algoritmos de compresión o de clasificación.
5. 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.
6. 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.

REFERENCIAS

- [1] 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.
- [2] Przulj, N., Corneil, D. G., & Jurisica, I. (2006). Efficient estimation of graphlet frequency distributions in protein-protein interaction networks. *Bioinformatics*, 22(8), 974-980.
- [3] 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.
- [4] 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.
- [5] Ramirez, A., Ospina, D., Ocampo, D. 2016. Aplicaciones de los Grafos. <https://sites.google.com/site/aplicaciongrafos/> Pagina Vigente al 15/02/2016.
- [6] 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.
- [7] 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.
- [8] Grossman, R., Kasif, S., Moore, R., Rocke, D., Ullman, J. 1999. *Data Mining Research: Opportunities and Challenges*.
- [9] 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
- [10] Holder, L. B., Cook, D. J., & Djoko, S. (1994, July). Substructure Discovery in the SUBDUE System. In *KDD workshop* (pp. 169-180).
- [11] Ullman, J. R. (1976). An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)*, 23(1), 31-42.
- [12] Kuramochi, M., & Karypis, G. (2001). Frequent subgraph discovery. In *Data Mining, 2001. ICDM 2001. Proceedings IEEE International Conference on* (pp. 313-320). IEEE.
- [13] 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.
- [14] Wörlein, M., Meinel, T., Fischer, I., & Philippsen, M. (2005). A quantitative comparison of the subgraph miners MoFa, gSpan, FFSM, and Gaston (pp. 392-403). Springer Berlin Heidelberg.
- [15] 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.
- [16] Fortin, S. (1996). The graph isomorphism problem. Technical Report 96-20, University of Alberta, Edmonton, Alberta, Canada.
- [17] 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).
- [18] 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.
- [19] Zaiat, J., & Romero-Zaliz, R. *Minería de datos sobre grafos: un enfoque multiobjetivo aplicado a bioremediación*.
- [20] 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.
- [21] Takigawa, I., & Mamitsuka, H. (2013). Graph mining: procedure, application to drug discovery and recent advances. *Drug discovery today*, 18(1), 50-57.
- [22] 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.
- [23] Hagberg, A., Schult, D., Swart, P. 2016. Networkx Framework. [https:// networkx.github.io](https://networkx.github.io) Pagina Vigente al 15/02/2016.
- [24] Rentsch, T. (1982). Object oriented programming. *ACM Sigplan Notices*, 17(9), 51-57.
- [25] 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.



Santiago Bianco. Licenciado en Sistemas por la Universidad Nacional de Lanús (UNLa). Es Investigador Asistente del Laboratorio de Investigación y Desarrollo en Ingeniería de Explotación de Información e Ingeniería de Sistemas Inteligentes del Grupo de Investigación en Sistemas de Información (UNLa GISI) y Docente de la Asignaturas Organización de Computadoras, Arquitectura de Computadoras, Ingeniería de Software II y Sistemas Basados en Conocimiento de la Licenciatura en Sistemas del Departamento de Desarrollo Productivo y Tecnológico de la Universidad Nacional de Lanús.