



**I
N
A
O
E**

Búsqueda de patrones interesantes en un solo grafo utilizando correspondencia inexacta

Marisol Flores Garrido, J. Ariel Carrasco-Ochoa,
José Fco. Martínez-Trinidad

Reporte Técnico No. CCC-12-001
23 de octubre de 2012

© Coordinación de Ciencias Computacionales
INAOE

Luis Enrique Erro 1
Sta. Ma. Tonantzintla,
72840, Puebla, México.



Búsqueda de patrones interesantes en un solo grafo utilizando correspondencia inexacta

Marisol Flores Garrido J. Ariel Carrasco-Ochoa José Fco. Martínez-Trinidad

Coordinación de Ciencias Computacionales
Instituto Nacional de Astrofísica, Óptica y Electrónica
Luis Enrique Erro # 1, Santa María Tonantzintla, Puebla, 72840, México
E-mail: {mflores, ariel, fmartine}@inaoep.mx

Abstract

La búsqueda de patrones frecuentes en grafos es un problema de interés en diversas áreas de investigación. La mayoría de los algoritmos reportados en la literatura se enfoca en el caso en el que los patrones se buscan en una colección de grafos y se usa el isomorfismo de grafos para determinar las ocurrencias de cada patrón. En esta propuesta de investigación se aborda el problema de buscar patrones en un solo grafo, utilizando correspondencia inexacta al momento de determinar las ocurrencias de un patrón. Adicionalmente, se propone no dar como resultado el conjunto de todos los patrones frecuentes, sino un subconjunto de patrones que sean interesantes, con el propósito de reducir la redundancia y facilitar el análisis del conjunto de patrones de salida. Permitir correspondencia inexacta, entre un patrón y sus ocurrencias, hace posible encontrar patrones que pasarían desapercibidos con otros algoritmos, pero plantea dificultades no triviales, entre ellas, definir una función para establecer, de manera eficiente, la similitud entre grafos y una estrategia para recorrer el espacio de búsqueda. Además de describir el problema propuesto, su contexto y los lineamientos de la investigación que se plantea, en esta propuesta se introduce, como parte de los resultados preliminares de la investigación, una función para comparar grafos usando correspondencia inexacta y un algoritmo, AGraP, capaz de identificar patrones que pueden tener diferencias estructurales, tanto en vértices como en aristas, respecto a sus ocurrencias. A través de los resultados experimentales se muestran los logros y las limitaciones del algoritmo propuesto hasta el momento.

Palabras Clave: *Patrones frecuentes en un solo grafo, correspondencia inexacta, patrones interesantes en grafos*

1. Introducción

El problema que se propone explorar en esta investigación doctoral es la búsqueda de patrones frecuentes e interesantes en un solo grafo usando correspondencia inexacta. La importancia de la búsqueda de patrones en grafos no resulta sorprendente dada la utilidad que las representaciones basadas en grafos han demostrado en distintos problemas aplicados, al punto que, en los últimos años, la minería de datos basada en grafos se ha convertido en uno de los temas más estudiados en el campo de la Minería de Datos [AW10, CH07, TSK06]. En particular, los patrones frecuentes en grafos han demostrado ser útiles para caracterizar conjuntos [BB02], discriminar entre diferentes grupos [HWB⁺04], clasificar [DKWK05], agrupar y construir índices [YYH04], etc.

Los algoritmos que abordan el problema de identificar patrones frecuentes en grafos difieren entre sí por la estrategia de búsqueda que emplean, la forma en que generan candidatos a patrones, la naturaleza de los grafos que examinan y el conjunto de patrones que encuentran [KRSA11]. En esta investigación, el problema específico de nuestro interés queda definido por tres características importantes del escenario en el que se desea trabajar: la búsqueda de patrones se realiza en un solo grafo, se emplea correspondencia inexacta al momento de comparar grafos y no se desea obtener el conjunto de todos los patrones frecuentes, sino un subconjunto de patrones interesantes. Cada uno de estos aspectos representa un eje de investigación en el trabajo que proponemos e introduce dificultades importantes que deben tomarse en cuenta para desarrollar una solución al problema.

El primer supuesto del problema que nos interesa es que la búsqueda de patrones se realiza en un solo grafo. En este escenario, una de las principales dificultades radica en la forma de definir el soporte de cada grafo candidato a patrón. En una colección de grafos el soporte de un candidato se define como la cantidad de grafos que lo contienen; en el caso de un solo grafo, debido al posible traslape de ocurrencias, definir el soporte de un candidato como el número de sus ocurrencias en el grafo causa la pérdida de la propiedad de antimonotonidad (la propiedad establece que si un grafo es frecuente, todos sus subgrafos son frecuentes) y, en consecuencia, los enfoques tradicionales para podar el espacio de búsqueda no pueden ser aplicados. Con el objetivo de evitar la pérdida de la antimonotonidad, se han propuesto diversas formas de calcular el soporte de un patrón en un solo grafo [KK01, FB07, BN08, CYZH07], pero quizá por ésta y otras dificultades que presenta, la minería de patrones en un solo grafo ha sido relativamente poco estudiada; hasta ahora, la búsqueda de patrones frecuentes en grafos se ha centrado principalmente en el caso de una colección de grafos y para esto se han desarrollado algoritmos muy eficientes, como GraphSig [RS09], Gaston [NK04], gSpan [YH02] o gRed [GAMPCOMT08], entre otros. Sin embargo, a pesar de que ha sido poco estudiado, encontrar patrones frecuentes en un solo grafo es un problema que merece atención por presentarse en distintas áreas de investigación, por ejemplo el análisis de redes sociales [Moo01, AAG11] o de comunicación [Les09], y por la generalidad que en cierto sentido encierra: los algoritmos para encontrar patrones en un sólo grafo pueden ser adaptados para el caso en el que los patrones se buscan en una colección de grafos, mientras que la situación inversa no es posible [KK05]. Para usar un algoritmo de búsqueda de patrones diseñado para una colección de grafos al caso en el que se tiene un solo grafo, es necesario dividir, de alguna manera, al grafo original para entonces aplicar el algoritmo; desafortunadamente, al dividir al grafo se corre el riesgo de “partir” patrones importantes y, en todo caso, decidir la mejor manera de dividir un grafo no es una tarea trivial.

El segundo aspecto que se explorará en esta investigación es la búsqueda de patrones en grafos utilizando correspondencia inexacta, es decir, al identificar las ocurrencias que un candidato a patrón tiene en el grafo original, se desea permitir diferencias estructurales, en vértices y aristas, entre el grafo candidato y los subgrafos que se consideren una aparición del mismo. La gran mayoría de los algoritmos que identifican

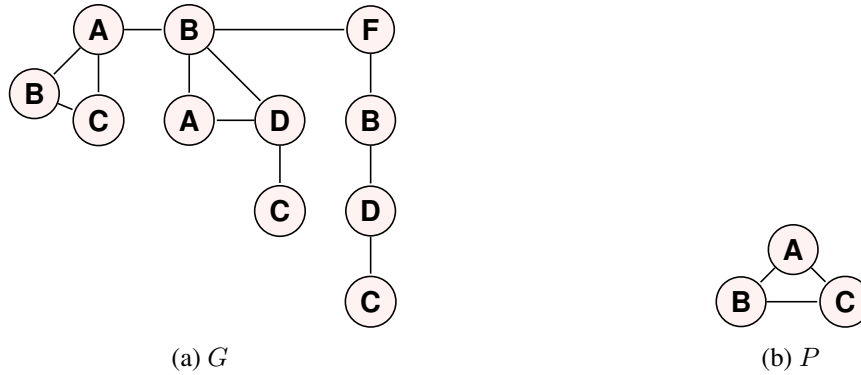


Figura 1: Usando un umbral de frecuencia $\sigma = 2$, el patrón P no puede ser indentificado en G usando isomorfismo de grafos, pero sí cuando se usa correspondencia inexacta.

patrones en grafos no permiten tales diferencias y, de alguna manera, requieren dar solución al problema de isomorfismo de grafos para establecer si existe correspondencia entre el patrón que se está examinando y algún subgrafo dado. Permitir diferencias tiene sentido cuando los datos con los que se trabaja están sujetos a, por ejemplo, ruido o errores de muestreo; en tales casos (muy frecuentes en problemas reales) es fácil intuir que una correcta identificación de patrones requiere la tolerancia a pequeños cambios en la información contenida en vértices y aristas. Pero, más allá de un contexto en el que los datos puedan estar contaminados por errores, utilizar correspondencia inexacta introduce cierta flexibilidad en los patrones que se analizan y permite encontrar algunos que, usando correspondencia exacta, podrían pasar desapercibidos. La Fig. 1 muestra un ejemplo de un patrón que, con un umbral de frecuencia $\sigma = 2$, es identificable utilizando correspondencia inexacta, pero que no puede ser encontrado cuando se usa isomorfismo de grafos.

Existen en la literatura algoritmos de búsqueda de patrones en grafos que permiten diferencias entre un patrón y sus ocurrencias. Los algoritmos gApprox [CYZH07] y APGM [JZH11] se centran en casos en donde las etiquetas en los nodos pueden ser diferentes y permiten algunas diferencias estructurales en aristas, mientras que GraMi [SK11] busca patrones usando lo que los autores denominan *una generalización del isomorfismo de grafos*, en donde la noción de una arista entre dos nodos se reemplaza por la de un camino entre los mismos. Sin embargo, hasta donde sabemos, no existe un algoritmo que permita diferencias de carácter estructural tanto en vértices como en aristas; es en este tipo de diferencias que queremos centrar nuestra atención, de manera que dos grafos (un patrón y alguna de sus ocurrencias, por ejemplo) puedan ser considerados similares aún si tienen diferente tamaño u orden. Además, nos interesa que las diferencias permitidas entre grafos considerados similares sean, de alguna manera, proporcionales al tamaño de los grafos que se comparan, de modo que diferencias pequeñas tengan más peso en grafos pequeños que en grafos grandes, aunque esto conlleve a la pérdida de la antimonotonía.

Sin embargo, aunque utilizar correspondencia inexacta permite identificar patrones que pueden ser importantes y que, de otra manera, pasarían desapercibidos, también introduce retos importantes al proceso de minería, por ejemplo, ¿cómo recorrer el espacio de búsqueda si no es posible poderlo en la forma habitual? ¿cómo manejar la elevada cantidad de ocurrencias que cada patrón puede tener? ¿cómo decidir si dos grafos son suficientemente parecidos para que uno sea considerado ocurrencia del otro? ¿cómo evitar que la cantidad de patrones que se obtienen al final sea tan grande que analizarlos sea poco viable? Todas éstas son dificultades con las que debe lidiarse para obtener una solución adecuada al problema que se plantea.

El tercer aspecto importante de nuestra investigación retoma la última de las preguntas planteadas en

el párrafo anterior: dada la gran cantidad de patrones encontrados al final del proceso de minería, ¿cómo puede decidirse cuáles son de interés? Algoritmos como Gaston [NK04] y gSpan [YH02], entre otros, permiten encontrar el conjunto completo de patrones frecuentes en una colección de grafos, pero ¿qué tan útil resulta encontrar *todos* los patrones frecuentes? La cantidad de patrones que pueden generarse en un problema de tamaño moderado puede llegar a ser prohibitivamente grande, constituyendo un cuello de botella computacional y, en el mejor de los casos (si se logra completar el proceso de minería), dando lugar a un conjunto de patrones tan grande que no resulte verdaderamente informativo o útil para el análisis de los datos de entrada. Para ilustrar esto consideremos el ejemplo referido por Hasan *et al.* [AHCS⁺07], en el que se utilizó un algoritmo de búsqueda en profundidad [AHCS⁺05] para analizar tres grafos, cada uno con 2154 nodos y un promedio de 81607 aristas; los autores reportan que al abortar el proceso de minería después de un día, se habían encontrado más de 8 millones de subgrafos frecuentes, el más grande con tan sólo 22 aristas. Algoritmos como SPIN [HWPY04], MARGIN [TVK10] y CloseGraph [YH03] tratan de mitigar este problema concentrándose en la búsqueda de patrones maximales o cerrados, pero aún en este caso la cantidad de patrones encontrados puede ser tan grande que no sea posible para un experto analizar el resultado. Por esta razón, la atención en la minería de patrones en grafos se ha desplazado, en los últimos años, de encontrar el conjunto completo de patrones frecuentes, a encontrar un subconjunto significativo, es decir, una menor cantidad de grafos que, de alguna manera, retenga la información del conjunto completo de patrones [AW10, Cap. 12]. Siguiendo con esta tendencia, en esta investigación queremos obtener un conjunto de grafos frecuentes que pueda considerarse representativo (interesante) del grafo analizado y que, por contener un menor número de patrones, permita un análisis más valioso de los datos estudiados.

La combinación de los tres aspectos descritos anteriormente define el problema de investigación que se propone explorar en esta investigación doctoral. La principal contribución que se espera al final de la investigación es un algoritmo capaz de identificar patrones frecuentes e interesantes en un solo grafo utilizando correspondencia inexacta, que permita diferencias estructurales de vértices y aristas. En el camino para obtener este algoritmo se esperan contribuciones adicionales, como una medida de similitud entre grafos que pueda ser evaluada eficientemente y una definición de patrón interesante, posiblemente en el entorno de una aplicación particular. Al respecto de dicha aplicación, cabe mencionar que nos interesa utilizar el algoritmo propuesto en el contexto del análisis de grafos dinámicos. Los grafos dinámicos son, *grosso modo*, grafos que varían con el tiempo, es decir, algunos vértices y/o aristas pueden aparecer o desaparecer a lo largo del tiempo; sobre este tipo de grafos y sobre la forma en que nuestro algoritmo podría utilizarse para su análisis se hablará brevemente en el siguiente capítulo.

Hasta el momento, siguiendo la metodología que se describe más adelante, nuestro esfuerzo se ha concentrado en definir un primer algoritmo de búsqueda de patrones frecuentes en un solo grafo utilizando correspondencia inexacta. Como parte de los resultados preliminares de esta investigación, se han propuesto una función de similitud, sensible al tamaño de los grafos que se comparan, basada en una variante de la distancia de edición y una estrategia para identificar patrones que pueden tener diferencias estructurales, tanto en vértices como en aristas, con sus ocurrencias. La medida de similitud y la estrategia propuestas, se han utilizado para construir el algoritmo AGraP, que, llevando a cabo una búsqueda en profundidad, es capaz de identificar patrones que no es posible encontrar con otros algoritmos del estado del arte.

El resto de esta propuesta está organizado como sigue: en el capítulo 2 se describe el contexto del problema, mencionando los principales trabajos relacionados; en el capítulo 3, se establecen los objetivos de la investigación y se presenta la metodología y el cronograma de trabajo propuestos para llevarla a cabo. Posteriormente, en el capítulo 4, se describen los resultados preliminares obtenidos hasta el momento, lo que incluye una descripción del algoritmo AGraP, y se muestran resultados experimentales que respaldan la eficacia del algoritmo propuesto. Finalmente, en la sección 5 se presentan nuestras conclusiones.

2. Antecedentes

Como se mencionó anteriormente, el problema central que se plantea en esta investigación doctoral es la búsqueda de patrones frecuentes e interesantes en un solo grafo utilizando correspondencia inexacta.

En este capítulo se presentan las ideas necesarias para entender el contexto del problema y se señalan los principales trabajos relacionados con la investigación que proponemos llevar a cabo. En las primeras tres secciones se describen, respectivamente, la tres características que definen la búsqueda de patrones frecuentes en la que se centra nuestro interés: en un sólo grafo, utilizando correspondencia inexacta y con interés en obtener patrones interesantes. En la cuarta sección se mencionan los principales enfoques que existen en la literatura para establecer la similitud entre grafos. Finalmente, en la quinta sección se describe la búsqueda de patrones en grafos dinámicos, que representa un contexto en el que se planea experimentar durante el desarrollo de nuestro trabajo.

2.1. Búsqueda de patrones en un solo grafo

Los distintos algoritmos que existen para encontrar patrones frecuentes en grafos pueden clasificarse de acuerdo a diferentes criterios, por ejemplo, la estrategia de búsqueda (en amplitud o en profundidad), la completitud del conjunto de patrones encontrados (todos o parte de ellos), la estrategia de generación de candidatos (extender o combinar patrones encontrados) y el tipo de datos que se están analizando (si es una colección de grafos o un solo grafo). Esta última división define claramente dos escenarios distintos del problema, con dificultades diferentes.

El problema de buscar patrones frecuentes consiste en encontrar, en los datos de entrada D , aquellos subgrafos cuyo soporte sea mayor o igual que un umbral de mínimo soporte establecido por el usuario. Cuando lo que se tiene es una colección de grafos, $D = \{G_1, G_2, \dots, G_n\}$, el soporte de un subgrafo g está dado por $s(g) = |D_g|/|D|$, donde D_g representa al conjunto de grafos de soporte de g definido como $D_g = \{G_i | g \subset G_i, G_i \in D\}$, con $g \subset G_i$ indicando que g es un subgrafo de G_i . Entre los algoritmos más conocidos para encontrar patrones en una colección de grafos se encuentran SUBDUE [Hol88], gSpan [YH02], Gaston [NK04], CloseGraph [YH03], GraphSig [RS09], gRed [GAMPCOMT08] y MoFa [BB02]. Una descripción de estos algoritmos, sus diferencias, ventajas y desventajas puede encontrarse en [CYH10] o [KRSA11].

El escenario de buscar patrones en un solo grafo, por otro lado, ha sido mucho menos explorado. A pesar de esto, es un problema que aparece con mucha frecuencia en aplicaciones [Moo01, AAG11, Les09, LGR⁺12], y se sabe que los algoritmos encontrados para este caso se pueden adaptar con relativa facilidad al problema de una colección de grafos; en ese caso la idea principal es resolver, para cada grafo en la colección, el problema de identificar si un patrón dado tiene al menos una ocurrencia en ese (solo) grafo. Los algoritmos para una colección de grafos, por otro lado, no pueden ser directamente aplicados al problema de buscar patrones en un solo grafo [KK05].

La principal diferencia cuando se buscan patrones en un solo grafo, respecto a buscar patrones en una colección de grafos, radica en la forma de contar la frecuencia de un patrón dado, pues en el escenario de un solo grafo las ocurrencias de un patrón pueden traslaparse (tener uno o más vértices en común) y, en ese caso, es necesario establecer cuántas veces debe considerarse que ocurre un patrón; un conteo inadecuado puede dificultar el recorrido del espacio de búsqueda. La falta de una estrategia para lidiar con ocurrencias que se traslapan es una de las razones por las que los algoritmos diseñados para una colección de grafos no funcionan en el caso de un solo grafo.

Idealmente, la función establecida como soporte de un patrón en un solo grafo debe satisfacer la pro-

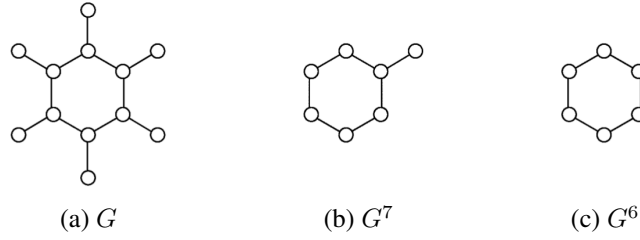


Figura 2: En esta figura pueden observarse patrones que no satisfacen la propiedad de antimonotonicidad, si se cuentan ocurrencias traslapadas. El grafo G^6 es subgrafo de G^7 , pero tiene una menor frecuencia en G .

propiedad de *antimonotonicidad*: un subgrafo de tamaño k es frecuente solamente si todos sus subgrafos son frecuentes. Esta propiedad es importante porque permite la poda del espacio de búsqueda del problema. Los algoritmos que buscan subgrafos frecuentes empiezan considerando subgrafos de tamaño 1, es decir, vértices, y hacen crecer gradualmente los subgrafos frecuentes encontrados. Para cada subgrafo se calcula el soporte que tiene y si éste es menor al umbral establecido por el usuario, el subgrafo es eliminado; sólo los subgrafos frecuentes se conservan para generar subgrafos de tamaño mayor. De esta manera se poda el espacio de búsqueda, dado que al eliminar subgrafos infrecuentes no es posible que un supergrafo tenga soporte mayor y, por lo tanto, ningún supergrafo será frecuente.

Establecer la medida de soporte de un patrón dentro de un solo grafo no es una tarea trivial. Si al buscar patrones se ignoran los posibles traslapes y simplemente se cuentan todas las ocurrencias de un patrón (subgrafo), se pierde la propiedad de antimonotonicidad. Esto se ilustra en la Figura 2. Las estructuras G^7 y G^6 son ambas subgrafos de G ; sin embargo, aunque G^6 ocurre sólo una vez en G y está contenido en G^7 , G^7 tiene 6 ocurrencias en G .

En 2005 Kuramochi y Karypis [KK05] proponen los algoritmos Hsigram y Vsigram para encontrar patrones frecuentes en un solo grafo. Los algoritmos propuestos difieren sólo en la estrategia que se sigue para recorrer el espacio de búsqueda; uno utiliza búsqueda en amplitud y el otro en profundidad. Ambos algoritmos utilizan un etiquetado de vértices que permite ordenarlos y, de este modo, evitar repetir comparaciones; de este modo se gana eficiencia. Para establecer la frecuencia de un patrón, los algoritmos construyen un grafo de traslapes en ocurrencias del patrón, en el que cada vértice representa una ocurrencia y se conectan los vértices mediante una arista si las ocurrencias representadas se traslapan; la Figura 3 muestra un ejemplo. Los autores definen el soporte de un patrón como el máximo conjunto independiente del grafo de traslape de ocurrencias asociado al patrón. Recordemos que, en Teoría de Grafos, el máximo conjunto independiente de un grafo es el conjunto más grande de vértices del grafo que cumple con la propiedad de que ningún vértice del conjunto es adyacente a otro; encontrar este conjunto es un problema NP-difícil.

Más adelante, los mismos autores proponen un nuevo algoritmo, GREW [KK04], que representa una especialización de Vsigram para el caso particular de un solo grafo, no dirigido, para el cual existe garantía de que, entre otras propiedades, la cantidad de ocurrencias sin traslape para cada patrón es al menos tan grande como el umbral de frecuencia establecido por el usuario. En este nuevo algoritmo se sacrifica la completitud del conjunto de patrones frecuentes obtenidos, pero se reporta un incremento considerable en cuanto a la eficiencia.

En 2007 Fiedler y Borgelt [FB07] sugieren una definición de soporte que permite ciertos traslapes, siempre y cuando no sean *dañinos*, es decir, siempre que no afecten la propiedad de antimonotonicidad. Posteriormente, en 2008, Bringmann y Nijssen [BN08] sugieren una medida de soporte que no requiere la

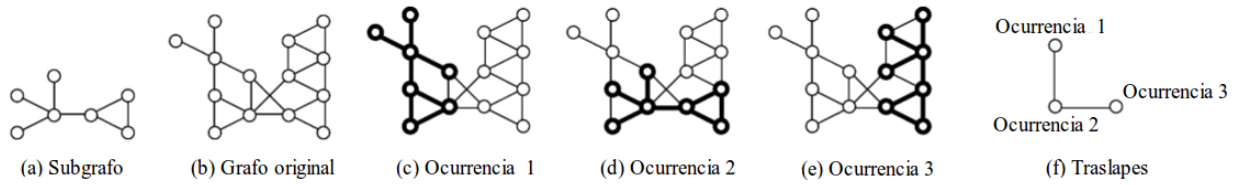


Figura 3: Ejemplo de un grafo de traslapes de ocurrencias. Este tipo de grafo se usa en el trabajo de Kuramochi y Karypis [KK05] para definir la frecuencia de un patrón dado.

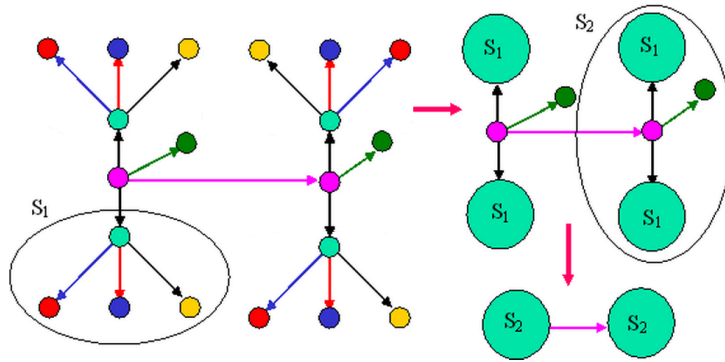


Figura 4: Ejemplo de la idea que sigue el algoritmo SUBDUE: se trata de indentificar la subestructura que mejor comprime al grafo original, se reemplaza dicha estructura por un solo vértice y se aplica recursivamente el procedimiento al grafo resultante.

construcción del grafo de traslapes ni la identificación del máximo conjunto independiente y que, por lo tanto, resulta menos costosa computacionalmente; dado un grafo G y un grafo patrón $P = (V, E)$ se define su soporte basado en la mínima imagen en G como

$$\sigma(P, G) = \min_{v \in V_P} |\{\varphi(v) : \varphi \text{ es el mapeo entre } P \text{ y una de sus ocurrencias en } G\}|.$$

SUBDUE [HCD94] también permite encontrar patrones en un solo grafo, aunque su enfoque es distinto. SUBDUE no busca propiamente los subgrafos frecuentes, sino aquellos que más comprimen al grafo original, siguiendo el principio de *Minimum Description Length*. Usando este principio, con un algoritmo *greedy* y *beam search*, SUBDUE identifica las estructuras que permiten la mejor compresión del grafo original y las reemplaza por vértices para, posteriormente, repetir el proceso; la Figura 4 muestra un ejemplo del procedimiento. SUBDUE no garantiza encontrar todas las estructuras frecuentes, ni, propiamente, se concentra en encontrar éstas.

Todos los algoritmos mencionados hasta ahora identifican las ocurrencias de un patrón candidato resolviendo el problema de isomorfismo de grafos (o subgrafos), que consiste en determinar si un grafo es isomorfo a otro, es decir, dados los grafos $G_1 = (V_1, E_1)$ y $G_2 = (V_2, E_2)$, se busca determinar si existe una función biyectiva $f : V_1 \mapsto V_2$ tal que los vértices u y v en G_1 , son adyacentes si y sólo si $f(u)$ y $f(v)$ son adyacentes en G_2 . Este problema es claramente NP, pero no se ha establecido si se trata de un problema NP-completo, aunque se cree que lo es [LP09]. Por lo tanto, resolverlo es impráctico, pues todos los algoritmos conocidos son exponenciales respecto al tamaño de los grafos que se estén considerando. Como consecuencia, se tienen al menos dos opciones: usar un algoritmo aproximado que quizá produzca

soluciones no óptimas, o aplicar el algoritmo sólo a un subconjunto de los datos. Los algoritmos mencionados en esta sección emplean la segunda opción y cuentan con estrategias de preprocesamiento de datos, entre otras, para resolver el problema de isomorfismo de grafos tan pocas veces como sea posible.

El algoritmo que proponemos desarrollar en esta investigación doctoral difiere de los algoritmos mencionados anteriormente en que no usará isomorfismo de grafos para contar las ocurrencias de un patrón. Las ventajas y desventajas de esta característica se discuten brevemente en la siguiente sección, pero, en definitiva, puede decirse que utilizar correspondencia inexacta requiere de una estrategia diferente a la de los algoritmos descritos. El único trabajo en la literatura que realiza búsqueda de patrones en un solo grafo utilizando correspondencia inexacta, y que por lo tanto es el más parecido a lo que proponemos, es gApprox [CYZH07]; este algoritmo se describirá en la siguiente sección.

2.2. Búsqueda de patrones con correspondencia inexacta

El problema de correspondencia entre grafos se refiere a, dados dos grafos, determinar si son suficientemente parecidos para que uno sea considerado una ocurrencia del otro. En la correspondencia exacta, para que dos grafos se consideren equivalentes se requiere que exista una correspondencia total entre sus vértices y aristas (y etiquetas, en el caso de grafos etiquetados), en otras palabras, que exista un isomorfismo entre ellos; por otro lado, cuando se permite que existan algunas diferencias entre los grafos, se tiene una correspondencia inexacta. Puesto que muchos grafos se construyen a partir de datos de la vida real y están sujetos a errores, la correspondencia inexacta entre grafos, a través de medidas de similitud y algoritmos que las implementan, ha sido aplicada en diferentes problemas, como, por ejemplo, el procesamiento de imágenes [HCR99, CBBLn05] y la minería de textos [AC05], entre otros.

El uso de correspondencia inexacta en la búsqueda de patrones en grafos ha sido poco explorado; entre los trabajos reportados en la literatura podemos citar al algoritmo Monkey [ZYC07], que permite detectar árboles frecuentes aproximados en datos estructurados; el algoritmo MUSE [ZLGZ09], que busca patrones frecuentes aproximados en datos con incertidumbre; y el algoritmo SUBDUE [Hol88, HCD94], que ofrece la posibilidad de introducir un umbral de similitud y una *funcion de distorsión* que asigne un costo específico a cada operación de edición entre grafos (cambio de etiqueta, inserción o eliminación de un vértice o una arista), de manera que dos grafos sean similares si el costo total de transformar un grafo en el otro está por debajo del umbral. Aunque en estos tres algoritmos hay búsqueda de patrones aproximados, ninguno de ellos se centra en el problema de nuestro interés; de hecho, sólo encontramos en la literatura dos algoritmos cuyo objetivo principal es encontrar patrones frecuentes en grafos usando correspondencia inexacta: gApprox [CYZH07] (en un solo grafo) y APM [JZH11] (en una colección de grafos).

El algoritmo gApprox fue propuesto en 2006 por Chen et al. como una herramienta para encontrar patrones aproximados en grafos. Este algoritmo tiene como motivación capturar patrones interesantes que pasarían desapercibidos con correspondencia exacta y hacerlo en un solo grafo, lidiando con el problema de establecer el soporte de un patrón. Para alcanzar el primer objetivo, capturar patrones aproximados, los autores proponen permitir que dos grafos puedan ser considerados similares a pesar de diferencias en el etiquetado y en la cantidad de aristas. En el caso de las diferencias entre etiquetas, se asume que el usuario provee de una lista inicial que indica, para cada etiqueta, cuáles etiquetas pueden reemplazarla. En gApprox se debe tener un mapeo uno a uno entre los vértices de dos grafos similares, así que no se permiten diferencias a este respecto. La similitud entre dos grafos se mide en términos del total de etiquetas que difieren (en todos los casos las diferencias de etiquetas deben ser reemplazos admisibles, de acuerdo con la lista provista por el usuario) y la diferencia que existe entre las aristas de los grafos que se están comparando. Con esto, la medida de similitud que se obtiene es una variante de la distancia de edición [GXTL10], pues cuenta

las diferencias (de aristas y etiquetas) entre las dos estructuras consideradas. Dado un umbral de error, se considera que dos grafos se corresponden si su similitud (la cantidad de diferencias entre ellos) es menor que el umbral.

gApprox explora el espacio de búsqueda siguiendo un enfoque de crecimiento de patrones. Para el cálculo del soporte sigue la estrategia de no contar ocurrencias con traslape, pero dado que esto es muy costoso (como los autores argumentan), el soporte de cada patrón se establece mediante el cálculo de una cota superior para el máximo número de ocurrencias disjuntas que existen en el conjunto de ocurrencias identificadas. Dicha cota superior se calcula de una manera relativamente eficiente, especialmente si se compara con la costosa operación de identificar el máximo conjunto independiente de un grafo de traslapes, como fue propuesto por Kuramochi y Karypis [KK05]; sin embargo, representa una medida que, aunque conserva la propiedad de antimonotonidad (y, en consecuencia, permite la poda del espacio de búsqueda), podría causar que se descarten patrones de interés. gApprox tiene el mérito de ser el primer algoritmo que aborda el problema de búsqueda de patrones aproximados en un solo grafo.

En 2011, Jia, Zhang y Huan proponen APGM [JZH11], un método para realizar minería de grafos con datos afectados por ruido en aplicaciones del mundo real. Los autores se centran en el problema particular de estructuras en proteínas, señalando que los datos en este problema pueden tener ruido y distorsiones provenientes de cambios en los aminoácidos de las proteínas o mediciones experimentales imperfectas, entre otras fuentes, de manera que emplear correspondencia exacta implica imponer una restricción poco realista que puede pasar por alto patrones importantes. Para evitar esto, proponen usar una medida de similitud entre grafos basada en una *matriz de compatibilidad*, M , cuadrada, real, cuyos índices corresponden a las etiquetas de vértices presentes en la base de datos que se está analizando y cada entrada $M_{ij} \in [0, 1]$ indica la probabilidad de que la etiqueta i sea confundida por la etiqueta j . Dada una matriz de compatibilidad y un umbral de similitud τ , se define un grafo $G = (V, E)$ como aproximadamente isomorfo a un subgrafo de $G' = (V', E')$ si existe una función inyectiva $f : V \mapsto V'$ de manera que el producto, S_f , de la similitud (normalizada) entre cada pareja de vértices (v, v') sea mayor que τ ; en cada caso la similitud entre dos vértices se normaliza dividiéndola entre la similitud que se espera entre la etiqueta del vértice v y ella misma (las entradas en la diagonal de la matriz de compatibilidad). Posteriormente, se define la similitud S , entre dos grafos, como el máximo valor que puede tomar S_f , considerando todas las posibles funciones f entre los grafos que se están comparando. Los autores señalan la posibilidad de tener una matriz de compatibilidad para las aristas e incluir una etiqueta especial de “arista nula”, para, de esta forma, permitir diferencias estructurales entre los grafos; sin embargo, no se profundiza en esta idea.

APGM está diseñado para trabajar en una colección de grafos D , el soporte de un patrón dado, P , es definido como:

$$\text{sup}_P = \sum_{G' \in D_P} S(P, G')/|D|,$$

donde D_P representa el subconjunto de grafos en D que contienen un subgrafo aproximadamente isomorfo a P , dado el umbral de similitud τ . Un patrón P es frecuente si su soporte es mayor o igual que un umbral de frecuencia σ dado por el usuario. Cabe destacar que, por tratarse de una colección de grafos y no un solo grafo, los autores no tienen que lidiar con el problema del traslape de ocurrencias.

Finalmente, debe destacarse que, en APGM, las podas al espacio de búsqueda dependen por completo de que: (1) la matriz de compatibilidad M sea *estable*, es decir, que cada elemento de la diagonal principal de la matriz sea estrictamente mayor que el resto de las entradas de su fila; (2) que la medida de similitud entre grafos establecida cumpla la propiedad de antimonotonidad.

La principal diferencia entre el trabajo de estos autores y el que proponemos, es que nosotros queremos realizar la búsqueda en un solo grafo y las diferencias estructurales que queremos considerar incluyen el

caso en el que dos grafos tienen diferente cantidad de vértices, lo cual requiere de una estrategia de búsqueda diferente.

Respecto al algoritmo gApprox podemos decir que, de entre todos los trabajos reportados hasta el momento en la literatura, es el que guarda mayor similitud con el trabajo que proponemos. La diferencia fundamental con nuestro trabajo radica en que no queremos limitar la similitud entre grafos al caso en el que ambos tienen la misma cantidad de vértices; esto introduce más flexibilidad pero también complica la búsqueda y manejo de ocurrencias de un patrón. Otro punto en el que se difiere de gApprox es la medida de soporte que utiliza, ya que pensamos que la cota superior empleada por gApprox es demasiado amplia y esto origina que se desechen patrones que podrían ser interesantes; para resolver esto se requiere de una medida de soporte que pueda ser calculada eficientemente. Finalmente, cabe mencionar que nos interesan funciones de similitud que posiblemente no cumplen con la propiedad de antimonotonidad y esto introduce otra fuente de complejidad al problema de buscar patrones frecuentes en un solo grafo.

2.3. Búsqueda de patrones representativos

La complejidad inherente a los datos representados mediante grafos causa la explosión combinatoria del problema de búsqueda de patrones en grafos; como consecuencia, el interés actual en los algoritmos de búsqueda de patrones en grafos no es tanto obtener el conjunto completo de grafos frecuentes, sino centrar la atención en los grafos que, además de frecuentes, son significativos; esto puede llevar a reducir el costo computacional del análisis de los patrones obtenidos e incrementar la aplicabilidad de los mismos.

Una primera idea para obtener patrones frecuentes y significativos, consiste en encontrar el conjunto de patrones frecuentes, P , y, posteriormente, analizar P para seleccionar cuáles son los patrones interesantes. Aunque lo *interesante* de un patrón depende en gran medida del contexto en el que se está trabajando y del interés del usuario, los trabajos reportados en la literatura coinciden en que, mientras al buscar patrones frecuentes se toma en cuenta el soporte individual de cada patrón, al buscar un subconjunto de patrones interesantes es necesario tomar en cuenta la información que los patrones, como grupo, proporcionan. Se pueden mencionar al menos tres enfoques presentes en la literatura para, dado P , seleccionar un subconjunto de patrones interesantes: (1) elegir un subconjunto que permita la reconstrucción del conjunto completo [CG02, BL06], (2) agrupar patrones en P (*clustering*) y luego seleccionar un representante de cada grupo, obteniendo así un resumen de la información en el conjunto completo [PDZH02], y (3) seleccionar un subconjunto de patrones que sean informativos. En esta última categoría existe diversidad al definir qué es un patrón informativo, pero, en general, se busca minimizar la redundancia en la información que brindan los patrones del subconjunto seleccionado; un ejemplo en esta categoría es el trabajo de Bringmann y Zimmermann [BZ09], que propusieron evaluar cada patrón con base en la partición que induce en la base de datos analizada. Una desventaja evidente en todos estos trabajos consiste en que primero debe encontrarse el conjunto P de todos los patrones frecuentes, lo que puede ser muy costoso y, al final, una buena parte de estos patrones terminan por ser desechados. Debe señalarse que los algoritmos mencionados están pensados para encontrar patrones en general, no específicamente en grafos.

En el caso de grafos, encontrar el conjunto completo de patrones frecuentes y después seleccionar unos cuantos, puede ser tan costoso que resulte poco útil. Quizá por eso ORIGAMI [AHCS⁺07], el único trabajo (hasta donde sabemos) que aborda el tema de seleccionar patrones informativos en grafos, sigue un enfoque aleatorio. En 2007, Hasan *et al.* propusieron el algoritmo ORIGAMI para, dada una colección de grafos D , encontrar un conjunto α -ortogonal, β -representativo de grafos frecuentes maximales, \mathcal{M} , en D . Intuitivamente, dos patrones son α -ortogonales si su similitud está por debajo de un umbral α y un patrón es β -representativo de otro si la similitud entre ambos es al menos β . La ortogonalidad entre patrones de

un conjunto \mathcal{R} permite disminuir la redundancia en la información que proveen, mientras que la representatividad asegura que cualquier patrón que no esté en el conjunto tiene un representante, muy similar, en \mathcal{R} . De esta forma, un conjunto α -ortogonal β -representativo se vuelve una especie de resumen del conjunto completo de subgrafos frecuentes.

ORIGAMI, en una primera etapa genera, mediante caminatas aleatorias, una muestra $\widehat{\mathcal{M}}$ de subgrafos frecuentes maximales en D . La atención se centra en subgrafos frecuentes maximales porque hay un menor número de subgrafos frecuentes maximales que de subgrafos frecuentes; sin embargo, puesto que puede ser muy costoso encontrar el conjunto completo de grafos maximales, se utiliza un enfoque aleatorio. En una segunda etapa, se construye un conjunto α -ortogonal que sea β -representativo de $\widehat{\mathcal{M}}$.

Lo más interesante de ORIGAMI es la idea de un conjunto que minimiza la redundancia al mismo tiempo que retiene información sobre el conjunto de grafos que representa.

2.4. Medidas de similitud entre grafos

Durante las últimas décadas se ha realizado mucho trabajo en el estudio de medidas de distancia, o similitud, entre grafos, en diferentes dominios de aplicación. De acuerdo con Xiao *et al.* [XDW⁺08] estas medidas pueden agruparse en tres categorías:

- *Medidas de distancia basadas en costo, o distancias de edición* [GXTL10]. La similitud entre grafos queda definida como el mínimo costo requerido para transformar un grafo en otro; cada operación para transformar un grafo (inserción, eliminación o sustitución de vértices o aristas) tiene asignado un costo, dado por el usuario, que posiblemente incorpora información específica del contexto del problema. Aunque, dados dos grafos G_1 y G_2 , pueden existir distintas secuencias de operaciones que transformen G_1 en G_2 , la distancia de edición se define como el mínimo costo asociado a la transformación, y éste es único; encontrarlo, sin embargo, no es una tarea sencilla y el método que se sigue para calcularlo depende del tipo de grafos que se esté considerando. Establecer la similitud entre los vértices y aristas correspondientes es un problema que no está resuelto; en el caso de grafos etiquetados, los distintos métodos para calcular la distancia de edición utilizan las etiquetas de vértices y/o aristas [NB04, NB05, NB06]. En el caso de grafos sin etiquetas, generalmente los grafos se transforman en cadenas de texto (*strings*), utilizando información sobre los vértices, aristas y su conectividad, y la distancia de edición entre dos grafos se calcula a partir de la distancia de edición existente entre las cadenas de texto correspondientes [RKH03, RKH04, YH06].
- *Medidas de distancia basadas en estructura* [BS98, HP04]. El grado de similitud entre dos grafos se mide por una o varias subestructuras que tengan en común. Un ejemplo de este tipo de medidas es la métrica basada en el máximo subgrafo común [BS98]; si g es un subgrafo común entre G_1 y G_2 , se dice que es máximo si no existe ningún otro subgrafo común a G_1 y G_2 que tenga más vértices que g . Si $msc(G_1, G_2)$ denota al máximo subgrafo común entre G_1 y G_2 , Bunke y Shearer definen la distancia entre dos grafos como

$$d(G_1, G_2) = \frac{|V_{msc(G_1, G_2)}|}{\max(|V_{G_1}|, |V_{G_2}|)}$$

y demuestran que esta medida satisface las propiedades de una métrica. Encontrar el máximo subgrafo común entre dos grafos puede ser muy costoso computacionalmente, pero se ha demostrado [Bun97] que el cálculo del máximo subgrafo común entre dos grafos es equivalente al cálculo de la distancia de edición entre los grafos, utilizando cierta función de costo; esto permite que algoritmos para el

cálculo de la distancia de edición puedan adaptarse para el cálculo de la distancia basada en el máximo subgrafo común.

- *Medidas de distancia basadas en características* [DES07]. Cada grafo se asocia a un vector de características; la similitud entre grafos queda definida por la distancia entre sus respectivos vectores asociados. La construcción de los vectores depende de las características estructurales de interés para el problema particular que se esté tratando.

Un enfoque distinto a los anteriores para calcular la similitud entre grafos es el uso de kernels para grafos. Una función kernel es una función simétrica que mapea parejas de patrones a números reales [NB07]. En principio, un kernel K corresponde al producto interno en algún *feature space* que, en general, es diferente del espacio de representación de las instancias. La idea de construir kernels para grafos fue propuesta por Kondor y Lafferty [KL02] y extendida por Smola y Kondor [SK03] y por Gärtner [Gar02]. Desde entonces se han propuesto numerosas funciones que permiten la comparación de grafos en problemas específicos; según señalan Bunke y Riesen en [BR11], estas funciones pueden agruparse en tres familias principales: kernels de difusión [KL02], kernels basados en caminatas aleatorias [GFW03, VBKS08] y kernels de convolución [RG03, SVP⁺09]. Desafortunadamente, los kernels pueden ser muy costosos de evaluar, lo que los hace una opción poco conveniente cuando se tienen que realizar numerosas comparaciones entre grafos. Una descripción detallada de los kernels para grafos, sus propiedades, características y usos, se puede encontrar en [Gär08] o [NB07].

2.5. Patrones en grafos dinámicos

Como se mencionó anteriormente, un área de investigación en la que el algoritmo que deseamos desarrollar puede ser utilizado es la concerniente a los grafos dinámicos. Este tipo de grafos ha atraído la atención de la comunidad científica en la última década; según señalan Desikan *et al.* [DS04] “los cambios en los datos en una dimensión temporal revelan un nuevo tipo de información” e incorporar una dimensión temporal a problemas típicamente modelados a través de grafos resulta de interés en áreas como el análisis de la Web y otras redes de información [FMNW03, KNRT05], el estudio de la trayectoria de transmisión de enfermedades en simulaciones epidemiológicas [KM96, EGK⁺04], el estudio de la propagación de influencia e información en redes sociales [KKT03, TWH05] y el estudio de grafos de correlaciones de precios de mercado [KKLK02], entre otras.

Un grafo dinámico modela cambios que ocurren a través del tiempo en un grafo dado G ; conceptualmente se puede representar como una sucesión de grafos G_1, G_2, \dots, G_T de manera que $G_t = (V_t, E_t)$, con $1 \leq t \leq T$, corresponde a una captura (*snapshot*) del grafo G en el tiempo t . Uno de los principales retos al intentar determinar patrones en un grafo dinámico consiste en incorporar la dimensión temporal al análisis que se está realizando; una técnica comúnmente utilizada por los métodos de minería de patrones en grafos dinámicos consiste en identificar los patrones en cada representación del grafo en un tiempo específico determinado, *i.e.*, identificar los patrones en cada grafo G_t , y, posteriormente, determinar de alguna manera el cambio existente entre los patrones que pertenecen a dos grafos consecutivos para, así, describir la evolución ocurrida [YHC08, IW08, Rob09]. Desafortunadamente, establecer la correspondencia entre los patrones de G_t y G_{t+1} no es una tarea trivial y, además, usando esta estrategia es difícil identificar patrones temporales que no suceden en tiempos consecutivos.

Una alternativa presentada por Borgwardt *et al.* [BKW06] en 2006 consiste en representar toda la información del grafo dinámico en *un sólo grafo*. El trabajo supone que los vértices permanecen constantes a lo

largo de toda la evolución del grafo y representa el cambio en las relaciones entre vértices, *i.e.*, el comportamiento de las aristas, mediante etiquetas de 0s y 1s que señalan la presencia (o ausencia) de una arista en un tiempo dado. Esta forma de representación elimina la necesidad de establecer la relación entre los patrones de grafos consecutivos, pero deja abierto el problema de identificar patrones en un solo grafo con etiquetas y elementos que pueden variar. Consideramos que, por las características de este tipo de grafos, aplicar nuestro algoritmo sería de gran utilidad, por lo tanto, los grafos dinámicos proveen un contexto interesante en el cual experimentar durante el desarrollo de nuestra investigación.

3. Propuesta

3.1. Preguntas de investigación

En este trabajo se pretende contestar las siguientes preguntas de investigación:

- ¿Qué función de similitud puede emplearse para comparar subgrafos de manera que se permitan diferencias estructurales en vértices y aristas en grafos considerados similares?
- ¿De qué manera puede calcularse el soporte de un patrón en un sólo grafo tomando en cuenta las características del grafo y el uso de correspondencia inexacta?
- ¿De qué manera puede recorrerse el espacio de búsqueda para identificar patrones en un grafo utilizando correspondencia inexacta?
- ¿Cómo puede obtenerse un conjunto interesante de grafos frecuentes, de manera que se tenga un conjunto representativo y, a la vez, se disminuya la redundancia en los patrones reportados?

3.2. Objetivo general

El objetivo general de esta investigación doctoral es el siguiente:

Desarrollar un algoritmo de búsqueda de patrones frecuentes e interesantes en un solo grafo, usando correspondencia inexacta, permitiendo variaciones estructurales en vértices y aristas.

3.3. Objetivos específicos

1. Definir una medida de similitud entre grafos que admita diferencias estructurales tanto en vértices como en aristas y tome en cuenta el tamaño de los grafos que se comparan.
2. Proponer una estrategia que, usando la medida de similitud definida anteriormente, permita la búsqueda de patrones con correspondencia inexacta.
3. Definir una medida de soporte para determinar la frecuencia de un patrón dado dentro de un grafo, considerando correspondencia inexacta entre grafos.
4. Integrar e implementar un algoritmo para encontrar subgrafos frecuentes en un solo grafo, utilizando las estrategias y medidas establecidas anteriormente.
5. Diseñar e implementar una estrategia para obtener grafos que, además de frecuentes, resulten interesantes.

6. Evaluar la calidad de los patrones encontrados en el contexto de una aplicación y tareas de minería de datos como clasificación supervisada o extracción de reglas.

3.4. Contribuciones esperadas

Las principales contribuciones esperadas al término de esta investigación doctoral son las siguientes:

- Una medida de similitud entre grafos que admita diferencias estructurales tanto en vértices como en aristas y que pueda ser evaluada eficientemente.
- Una estrategia que permita recorrer el espacio de búsqueda tomando en cuenta el incremento de ocurrencias para cada patrón que puede originar el uso de correspondencia inexacta y la posible pérdida de la antimonotonidad.
- Una medida de soporte de patrones en un solo grafo.
- Una estrategia para obtener un subconjunto de grafos frecuentes interesantes.
- Un algoritmo que integre las contribuciones anteriores y permita la búsqueda de patrones frecuentes e interesantes en un solo grafo usando correspondencia inexacta.

3.5. Metodología

La metodología para poder alcanzar los objetivos propuestos en este trabajo es la siguiente:

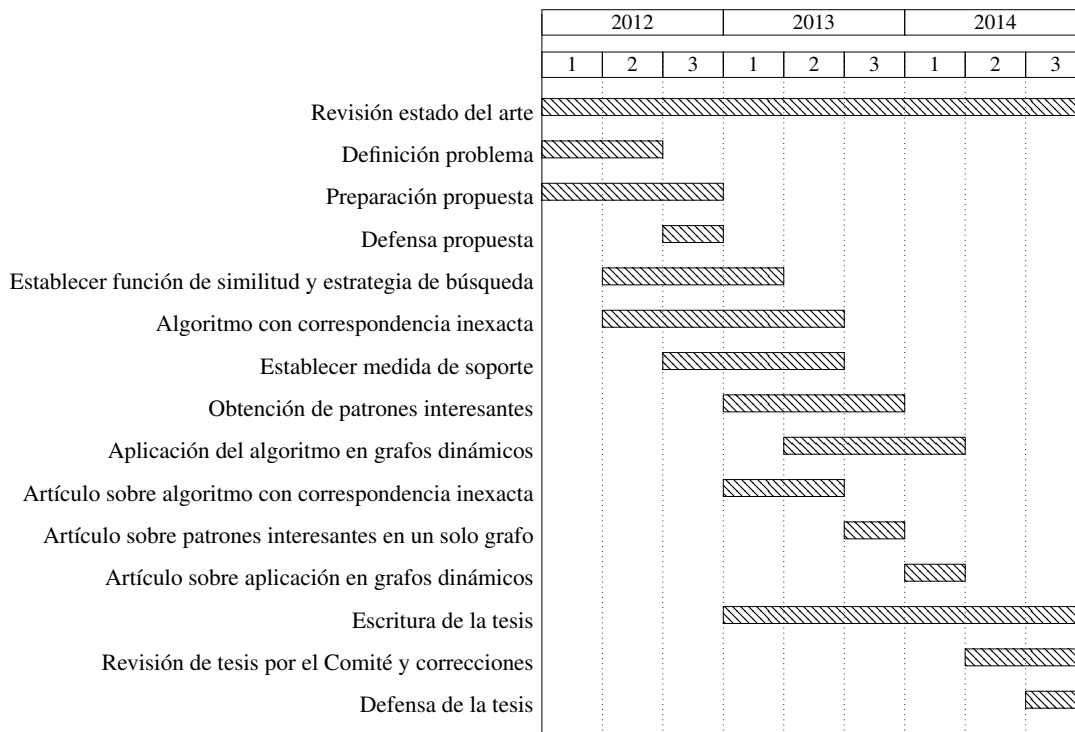
1. Definir una medida de similitud entre grafos que admita diferencias estructurales tanto en vértices como en aristas y tome en cuenta el tamaño de los grafos que se comparan.
 - a) Evaluar las medidas existentes en la literatura y decidir cuáles de ellas tienen las características que buscamos: permitir comparar dos subgrafos de manera eficiente, ser suficientemente flexible para considerar similares dos grafos que guardan diferencias estructurales entre vértices y aristas, y permitir diferencias entre grafos de manera proporcional al tamaño de los grafos que se están comparando.
 - b) Proponer una medida de similitud, tal vez ponderando alguna medida encontrada en el paso anterior con el tamaño de los grafos. Si ninguna medida reportada en la literatura permite lo que buscamos, se propondrá una nueva.
2. Proponer una estrategia que, usando la medida de similitud definida anteriormente, permita la búsqueda de patrones con correspondencia inexacta.
 - a) Establecer la forma en que puede recorrerse el espacio de búsqueda tomando en cuenta el aumento de ocurrencias, de cada patrón, que utilizar correspondencia inexacta trae como consecuencia.
 - b) Determinar cómo pueden manejarse las ocurrencias de un patrón de manera que, siguiendo un enfoque de crecimiento de patrones, se pueda identificar las ocurrencias de un nuevo patrón, considerando que pueden existir diferencias estructurales tanto en vértices como en aristas.

- c) Desarrollar, con la estrategia propuesta, un algoritmo de búsqueda de patrones con correspondencia inexacta, para así poder evaluar la correcta identificación, en grafos construidos para este propósito, de patrones que sólo pueden ser hallados mediante el uso de correspondencia inexacta. En este algoritmo “intermedio” se planea usar una estrategia de crecimiento de patrones en profundidad y alguna de las medidas de soporte (o cotas para ellas) propuestas en la literatura [KK05, FB07, CYZH07, BN08].
3. Definir una medida de soporte para determinar la frecuencia de un patrón dentro de un grafo, considerando correspondencia inexacta entre grafos.
 - a) Analizar las medidas de soporte existentes para decidir si alguna es adecuada para las condiciones que se proponen y si, además, puede calcularse eficientemente. Si las medidas reportadas en la literatura no satisfacen estos requisitos, se deberá definir una nueva de la siguiente manera:
 - 1) Analizar inicialmente si debería evitarse cualquier tipo de traslape al momento de contar las ocurrencias de un patrón.
 - 2) Establecer una medida de soporte compatible con la respuesta anterior y con el uso de correspondencia inexacta.
 - b) Analizar la posibilidad de, con el fin de incrementar la eficiencia en su cálculo, utilizar una aproximación a la medida propuesta.
 4. Integrar e implementar un algoritmo para encontrar subgrafos frecuentes en un solo grafo, que utilice las estrategias y medidas establecidas anteriormente.
 - a) Integrar la estrategia de identificación de patrones con correspondencia inexacta propuesta en el paso 1 y la medida de soporte definida en el paso 2, para desarrollar un algoritmo de búsqueda de patrones con diferencias estructurales, tanto en vértices como en aristas, en un solo grafo.
 - b) Analizar si es posible mejorar la eficiencia del algoritmo combinando propiedades de las medidas establecidas para calcular la similitud entre grafos y el soporte de un patrón, como se hace en el algoritmo APGM [JZH11] para estimar el soporte de un grafo.
 - c) Comparar la cantidad y calidad de patrones encontrados mediante el algoritmo desarrollado, contra los que se obtienen utilizando gApprox [CYZH07], que, hasta el momento, es el único algoritmo reportado en la literatura que realiza búsqueda de patrones frecuentes aproximados en un solo grafo. La calidad de los patrones se medirá en una tarea de minería de datos, posiblemente clasificación supervisada.
 5. Diseñar e implementar una estrategia para obtener grafos que, además de frecuentes, resulten interesantes.
 - a) Una posibilidad es iniciar situando el problema en el contexto de una aplicación particular. En este caso sería necesario analizar el contexto de la aplicación seleccionada para determinar qué información es de mayor interés. Las aplicaciones que se han considerado para esto son grafos dinámicos [DS04, YHC08, IW08, Rob09], redes de sistemas sociales y organizacionales [Car09, PC11], redes con signo [KLB09, LHK10, KSL⁺10] y redes de comunicación [KS05, EPL09].
 - b) Tomando en cuenta el análisis anterior y algunas propiedades descritas en la literatura para reconocer patrones “interesantes” [PJZ05, Rob09], adaptar la medida de similitud definida en el paso 1, para comparar grafos midiendo la redundancia en la información que contienen.

- c) Diseñar una estrategia que permita obtener un subconjunto de patrones frecuentes, de manera que la redundancia entre los patrones elegidos sea mínima y, al mismo tiempo, representen la información importante del conjunto de patrones frecuentes. Esta estrategia podría integrarse al algoritmo del punto anterior con el objetivo de producir directamente sólo los patrones interesantes, o bien, llevarse a cabo en un paso de post-procesamiento, *i.e.*, calculando todos los patrones y, posteriormente, seleccionando sólo los interesantes.
6. Evaluar la calidad de los patrones encontrados en el contexto de una aplicación y tareas de minería de datos como clasificación supervisada, extracción de reglas o agrupamiento.
- a) Determinar las bases de datos que se usarán para la tarea. En el caso de los grafos dinámicos, algunas bases utilizadas comúnmente en la literatura son la base de datos de emails ENRON [KY04] y la de bibliografía en Ciencias Computacionales DBLP [oT, Ley02].
 - b) Construir grafos a partir de la aplicación y base de datos seleccionadas. Si esta aplicación es la de grafos dinámicos, los grafos contendrán información temporal siguiendo la forma de representación de grafos dinámicos propuesta por Borgwardt *et al.* [BKW06], en la cual la información se condensa en un solo grafo etiquetado; de esta manera será posible el uso de nuestro algoritmo sin necesidad de modificaciones significativas.
 - c) Evaluar la calidad de los patrones obtenidos, comparando su utilidad para discriminar entre clases de grafos, contra la utilidad de patrones frecuentes aproximados generados por gApprox [CYZH07]. También se considera la posibilidad de comparar contra algoritmos diseñados específicamente para encontrar patrones en el contexto de la aplicación que se haya seleccionado; por ejemplo, en el caso de grafos dinámicos podría usarse el algoritmo *Dynamic GREW* [BKW06], que encuentra patrones en grafos dinámicos usando correspondencia exacta (isomorfismo).

3.6. Plan de trabajo

El cronograma de actividades propuesto se muestra a continuación. Las columnas corresponden a periodos de cuatro meses.



4. Resultados preliminares

Como se señaló en el capítulo anterior, el primer objetivo específico de esta investigación consiste en proponer una medida de similitud entre grafos y una estrategia que, usando la medida definida, permita la búsqueda de patrones con correspondencia inexacta, admitiendo diferencias estructurales tanto en vértices como en aristas. En este capítulo se describe el trabajo que se ha llevado a cabo hasta el momento para alcanzar el objetivo establecido.

En la primera sección se presenta una medida de comparación que permite establecer la similitud entre grafos usando una variante de la distancia de edición y tomando en cuenta el tamaño de los grafos que se están comparando. Posteriormente, en la segunda sección, se describen las estrategias propuestas para identificar ocurrencias de un patrón que difieren estructuralmente del mismo. En la tercera sección se introduce un algoritmo para búsqueda de patrones frecuentes en un solo grafo con correspondencia inexacta, que usa la medida de similitud propuesta en la sección 4.1, e implementa la estrategia descrita en la sección 4.2. Finalmente, se muestran algunos experimentos llevados a cabo, con resultados que indican la efectividad del algoritmo propuesto.

4.1. Medida de similitud entre grafos

Una de las diferencias más importantes entre el algoritmo que se busca desarrollar en esta investigación y los algoritmos del estado del arte, es que nuestro objetivo es encontrar ocurrencias de un patrón que pueden diferir, no sólo en las etiquetas de sus vértices o aristas, sino en la estructura misma, es decir, que pueden tener más (o menos) aristas o vértices que el patrón con el que se están comparando. En este escenario de correspondencia inexacta, se requiere de una función de comparación entre grafos para determinar si cada

posible ocurrencia de un patrón es suficientemente parecida al patrón (con una disimilitud por debajo de un umbral Δ , dado por el usuario) para ser considerada como una ocurrencia del mismo.

La función de comparación que utilizamos está basada en la distancia de edición existente entre los grafos comparados. Dados dos grafos $g_1 = (V_1, E_1, L_1)$ y $g_2 = (V_2, E_2, L_2)$ y una relación binaria biunívoca $m \subseteq V_1 \times V_2$, inicialmente definimos una medida de comparación entre g_1 y g_2 al estilo de la usada por gApprox [CYZH07] como

$$f_1(g_1, g_2) = \kappa_1 v_{edit} + \kappa_2 e_{edit} \quad (1)$$

donde κ_1 y κ_2 son los costos de edición asociados a vértices y aristas, respectivamente, y satisfacen la relación $\kappa_1 + \kappa_2 = 1$, mientras que las expresiones v_{edit} y e_{edit} se definen como

$$v_{edit} = \sum_{v \in V_1 \setminus R_{V_1}} d_v(v, m(v)) + |R_{V_1}| + |R_{V_2}| \quad (2)$$

con

$$\begin{aligned} R_{V_1} &= \{v_1 \in V_1 \mid \nexists v_2 \in V_2 \text{ tal que } m(v_1) = v_2\} \\ R_{V_2} &= \{v_2 \in V_2 \mid \nexists v_1 \in V_1 \text{ tal que } m(v_1) = v_2\} \\ d_v(v_1, v_2) &= \text{costo de sustituir } L_1(v_1) \text{ por } L_2(v_2) \end{aligned}$$

y

$$e_{edit} = \sum_{(u,v) \in E_1 \setminus R_{E_1}} d_e((u, v), (m(u), m(v))) + |R_{E_1}| + |R_{E_2}|$$

con

$$\begin{aligned} R_{E_1} &= \{(u, v) \in E_1 \mid \nexists u', v' \in V_2 \text{ tales que } m(u) = u', m(v) = v' \\ &\quad \text{y } (u', v') \in E_2\} \\ R_{E_2} &= \{(u', v') \in E_2 \mid \nexists u, v \in V_1 \text{ tales que } m(u) = u', m(v) = v' \\ &\quad \text{y } (u, v) \in E_1\} \\ d_e((u, v), (u', v')) &= \text{costo de sustituir } L_1((u, v)) \text{ por } L_2((u', v')) \end{aligned}$$

Es importante notar que, mientras gApprox requiere que m sea una función biyectiva, en nuestro caso se trata simplemente de una relación binaria biunívoca, así que pueden existir tanto vértices en V_1 que no estén relacionados con algún vértice en V_2 , como vértices en V_2 que no sean la imagen de algún vértice en V_1 ; ésta es la razón por la que se incluye en v_{edit} las expresiones $|R_{V_1}|$ y $|R_{V_2}|$, que representan el costo de estas diferencias. Esto permite considerar diferencias estructurales en los vértices. Del mismo modo, en e_{edit} se incluye $|R_{E_1}|$ y $|R_{E_2}|$ que representan, respectivamente, el costo de las aristas de E_1 que no están mapeadas en E_2 y el de las aristas de E_2 que no son imagen de ninguna arista en E_1 .

Respecto a la función f_1 , hay dos aspectos que vale la pena destacar. El primero es que contar las diferencias de vértices y aristas por separado permite, en caso de que la aplicación así lo requiera, ponderar de manera diferente las diferencias entre vértices y las diferencias entre aristas. Hemos definido, además, que $\kappa_1 + \kappa_2 = 1$, de manera que f_1 puede verse como una combinación lineal convexa de v_{edit} y e_{edit} .

Desafortunadamente, aunque la función f_1 permite comparar dos grafos y evaluar diferencias estructurales tanto en vértices como en aristas, la similitud que establece entre dos grafos no depende del tamaño de

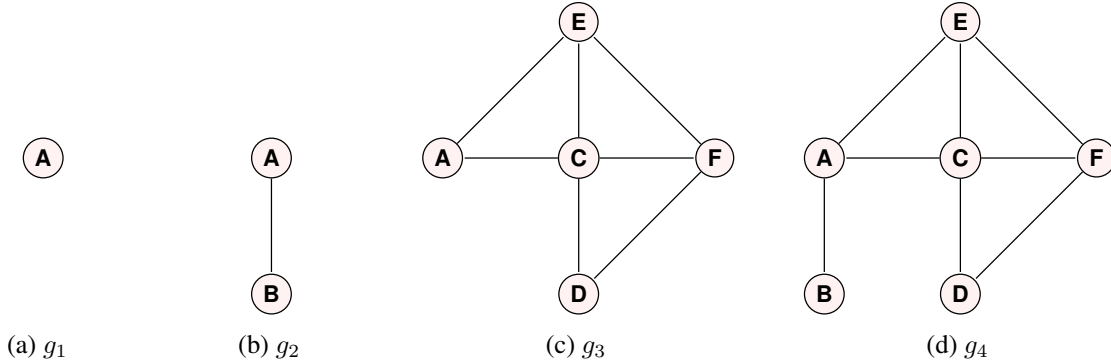


Figura 5: Aunque entre g_1 y g_2 existe la misma distancia de edición que entre g_3 y g_4 (en ambos casos la diferencia es sólo un vértice y una arista), la diferencia debería ser más importante al comparar g_1 y g_2 , por el tamaño de los grafos. Esta observación motiva a buscar una medida de comparación que sea relativa al tamaño de los grafos comparados.

los mismos y, por lo tanto, una diferencia de un vértice se valorará igual si los grafos comparados tienen 2 vértices que si tienen 5000; esto puede ser inadecuado en aplicaciones donde pequeñas diferencias en grafos grandes sean menos importantes que diferencias similares en grafos pequeños. Por ejemplo, en la Fig. 5 tanto los grafos g_1 y g_2 como los grafos g_3 y g_4 difieren en un solo vértice y, por lo tanto, $f_1(g_1, g_2) = f_1(g_3, g_4)$, sin embargo, g_3 y g_4 pueden considerarse más parecidos que g_1 y g_2 , pues concuerdan en 5 vértices, mientras que g_1 y g_2 concuerdan sólo en 1.

Para resolver este problema, siguiendo las ideas de [BS98] dividiremos las diferencias obtenidas entre el tamaño de los grafos comparados. Así, para comparar al patrón en turno, $P = (V_P, E_P)$, con una de sus posibles ocurrencias, $g = (V_g, E_g)$, y establecer si esta última es suficientemente similar a P para ser tomada en cuenta a la hora de calcular el soporte del patrón, utilizamos la medida dada por

$$f_2(P, g) = \frac{\kappa_1 v_{edit}}{|V_P| + |V_g|} + \frac{\kappa_2 e_{edit}}{|E_P| + |E_g|}. \quad (3)$$

Si se sigue una estrategia de crecimiento de patrones, inicialmente P consta de un solo vértice y se puede identificar sus ocurrencias (vértices con la misma etiqueta o una equivalente) comparando la etiqueta de P contra las etiquetas del resto de los vértices en el grafo; la medida f_2 se utilizará para patrones en donde $|V_P| \geq 2$ y $|E_P| \geq 1$, por lo tanto, los denominadores en la expresión anterior nunca serán cero.

Cada sumando en la expresión (3) representa la distancia de edición existente entre P y g , considerando sólo vértices o sólo aristas, normalizada por la cantidad total de vértices o aristas, existentes en ambos grafos; esto permite que el impacto de un valor de v_{edit} o e_{edit} dado, dependa del tamaño y orden del patrón y la ocurrencia que se están considerando. A diferencia de una distancia de edición, en donde sólo se toma en cuenta la cantidad de transformaciones necesarias para llegar de un grafo a otro, queremos que el tamaño del patrón sea relevante para determinar cuántas diferencias se permite entre un grafo y otro: una diferencia de un vértice entre P y g , por ejemplo, debería ser más importante si P tiene 2 vértices que si tiene 5000. De esta manera, para los grafos que se muestran en la Fig. 5, tenemos, suponiendo que cada operación de edición tiene un costo de $\frac{1}{2}$, i.e., $\kappa_1 = \kappa_2 = \frac{1}{2}$, que

$$f_1(g_1, g_2) = f_1(g_3, g_4) = 1,$$

pero

$$f_2(g_1, g_2) = 0.66\bar{6} \quad \text{y} \quad f_2(g_3, g_4) = .07\bar{8}.$$

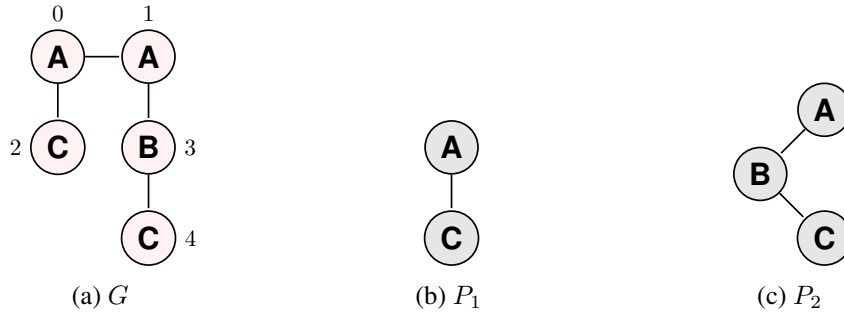


Figura 6: Se desea encontrar patrones con diferencias estructurales en vértices, de manera que P_1 y P_2 sean frecuentes en G para un umbral de frecuencia igual a 2.

Una ventaja que podemos señalar de f_2 respecto a f_1 es que, mientras esta última no está acotada, f_2 toma valores entre 0 y 1, por lo que podría ser más fácil determinar un umbral para f_2 , como un porcentaje de parecido entre los grafos que se están comparando, que para f_1 , en donde debe especificarse la máxima diferencia permitida sin saber el tamaño de los patrones que se están buscando.

Desafortunadamente, al utilizar f_2 se pierde la propiedad de antimonicidad, y esto debe tomarse en cuenta al recorrer el espacio de búsqueda.

Cabe señalar que estas medidas de similitud, f_1 y f_2 , han sido propuestas de forma preliminar y probablemente se modificarán a lo largo de la investigación, pero, de momento, nos permiten concentrar la atención en la forma en que se identificarán ocurrencias de un patrón utilizando correspondencia inexacta y satisfacen los requerimientos que se tienen, *i.e.*, permitir diferencias estructurales de vértices y aristas, así como, en el caso de f_2 , ser sensible al tamaño de los grafos que se comparan.

Finalmente, se debe también mencionar que f_1 y f_2 pueden calcularse durante el proceso de búsqueda de patrones siguiendo una estrategia de crecimiento de patrones, sin aumentar la complejidad del algoritmo.

4.2. Identificación de patrones con diferencias estructurales

El principal objetivo que se estableció fue el de encontrar patrones con correspondencia inexacta que pudieran tener diferente cantidad de vértices que sus ocurrencias. Considérese, por ejemplo, el grafo G que se muestra en la Fig. 6a. Utilizando correspondencia exacta y un umbral de frecuencia $\sigma = 2$, sólo sería posible identificar como patrones frecuentes a los vértices A y C por separado. Si, en cambio, se permiten diferencias en la estructura de los grafos, sería posible identificar, por ejemplo, al patrón P_1 (Fig. 6b) con los subgrafos inducidos por los conjuntos de vértices en G $\{0, 2\}$ y $\{1, 3, 4\}$ como sus ocurrencias y al patrón P_2 (Fig. 6c) con las mismas ocurrencias. A continuación se describe la forma en que identificamos ocurrencias de patrones con diferencias estructurales, utilizando la función f_2 para comparar subgrafos.

4.2.1. Ocurrencias en donde sobran vértices

En este caso quisiéramos que al buscar el patrón P_1 que se muestra en la Fig. 6b en el grafo G (Fig. 6a), los vértices $\{1, 3, 4\}$ fueran identificados como una ocurrencia del patrón, mapeando al vértice 0 con el 1 y al 2 con el 4, aunque el vértice 3 “sobra”. Esto se puede lograr si al formar un nuevo patrón $P' = P \cup \{v\}$ y buscar expandir una ocurrencia g , del patrón P a un vértice u que se corresponda con v (*i.e.*, que tenga la misma etiqueta o una etiqueta que, dependiendo del problema, pueda sustituir a la etiqueta de v), se reemplaza el requerimiento de que exista una arista entre g y el vértice u por el requerimiento de que exista

un camino entre g y el vértice u , como se hace en GraMi [JZH11] pero considerando correspondencia inexacta (algo que GraMi no hace). En el ejemplo que se menciona, aunque no existe una arista entre los vértices 1 y 4 del grafo G , existe un camino y, por lo tanto, puede considerarse una ocurrencia del patrón P_1 .

Una pregunta que surge al seguir este planteamiento es de qué longitud puede ser el camino permitido entre dos vértices. Puesto que la medida de similitud que estamos utilizando (f_2) depende del tamaño de los grafos considerados, la longitud del camino permitido también debería depender del tamaño del patrón que se esté considerando. Por ello, al momento de expandir cada ocurrencia de P , estimamos la longitud máxima ℓ que puede tener un camino para evitar sobrepasar el umbral de similitud Δ proporcionado por el usuario. Para estimar esta longitud consideremos un patrón P y una ocurrencia g con una distancia de edición entre ellos representada por v_{edit} y e_{edit} . Si el patrón P se extiende para formar $P' = P \cup \{v\}$ y la ocurrencia g se extiende a un vértice (supongamos un vértice que coincide de manera exacta con v) conectado a g por un camino de longitud ℓ (con $\ell - 1$ vértices intermedios), formando la ocurrencia g' , se tendría que

$$f_2(P', g') = \frac{\kappa_1 v_{edit} + \kappa_1(\ell - 1)}{(|V_P| + 1) + (|V_g| + \ell)} + \frac{\kappa_2 e_{edit} + \kappa_2 \ell + \kappa_2 e_{new}}{(|E_P| + e_{new}) + (|E_g| + \ell)},$$

donde κ_1 y κ_2 representan los costos de inserción de vértices y aristas, respectivamente, y e_{new} es la cantidad de aristas que conectan al patrón P con el nuevo vértice v . Al considerar un camino de longitud ℓ , el costo de edición relativo a los vértices se modifica porque, para que la ocurrencia g' coincida con el patrón P' , deben eliminarse $\ell - 1$ vértices (numerador del primer sumando), mientras que en el caso de las aristas se debe añadir el costo de eliminar las ℓ aristas que constituyen el camino y el de agregar e_{new} aristas para conectar el nuevo vértice con g (numerador del segundo sumando).

Para que g' sea considerada una ocurrencia de P' se requiere que $f_2(P', g') \leq \Delta$; a partir de esta desigualdad es posible estimar el tamaño máximo ℓ de un camino para que la similitud entre el patrón y la ocurrencia esté por debajo del umbral establecido. Resolver directamente la desigualdad lleva a la resolución de una ecuación cuadrática en ℓ con coeficientes que involucran muchos términos. Por ello, puede resultar más sencillo estimar la longitud máxima de ℓ usando un esquema iterativo de bisección, sobre el intervalo $[0, |E_G|]$. Encontrando el máximo valor ℓ que permite que $f_2(P', g')$ esté por debajo del umbral de similitud Δ , puede implementarse la idea descrita al inicio de esta sección y encontrar ocurrencias de un patrón dado en las que existen vértices “de más”.

4.2.2. Ocurrencias en donde faltan vértices

Dado el patrón P_2 que se muestra en la Fig. 6c, quisiéramos que los vértices $\{0, 2\}$ representaran una de sus ocurrencias en G , a pesar de que entre los vértices 0 y 2 “falta” un vértice con etiqueta B . Para conseguir esto, al momento de construir el conjunto de ocurrencias de un patrón $P' = P \cup \{v\}$, resultado de extender el patrón P con el vértice v , a partir del conjunto de ocurrencias P , conservamos aquellas ocurrencias que no pudieron crecer y mantenemos registro de que hay un vértice faltante en esa ocurrencia, además de actualizar apropiadamente la distancia de edición entre la ocurrencia que no pudo crecer y el patrón P' , esto último si el cálculo de las similitudes está integrado al proceso de búsqueda de patrones.

Como ya se explicó, la motivación para mantener estas ocurrencias que no pueden crecer en un cierto punto del algoritmo es que quizá, más adelante, tengan coincidencias con un supergrafo de P' , de tal manera que, a pesar de los vértices faltantes, la similitud, de acuerdo a f_2 , entre las ocurrencias extendidas y el nuevo patrón sea suficiente para que pueda decirse que ambos grafos se corresponden (de manera inexacta).

También cabe señalar que, aunque se conservan las ocurrencias de P que no pudieron expandirse, al calcular el soporte de P' sólo se toman en cuenta aquellas ocurrencias con disimilitud menor al umbral Δ establecido por el usuario.

4.2.3. Diferencias estructurales en aristas

Para permitir diferencias estructurales en aristas, dado el patrón $P' = P \cup \{v\}$, al momento de expandir una ocurrencia, g de P buscamos un vértice que se corresponda con v y que esté conectado a g , no importa que las aristas que lo conectan con g no correspondan a las que conectan al vértice v con P . Por supuesto, es importante contar las diferencias existentes e incluir esa información en la distancia de edición que se registra entre la ocurrencia y el patrón considerado (si este cálculo está integrado a la búsqueda).

4.3. Algoritmo propuesto

El algoritmo que se propone como resultado preliminar para buscar patrones frecuentes en un sólo grafo con correspondencia inexacta que permite diferencias estructurales en vértices y aristas, llamado AGraP (*Approximate Graph Patterns*), difiere de otros algoritmos en el estado del arte en la forma de identificar y contar ocurrencias de un patrón dado, para lo cual sigue la estrategia propuesta en la sección 4.2, usando la función de comparación de grafos f_2 , introducida en la sección 4.1. En AGraP, al igual que en otros algoritmos como gApprox [CYZH07], la exploración del espacio de búsqueda se lleva a cabo mediante una búsqueda en profundidad, se utiliza un enfoque de crecimiento de patrones para generar candidatos a patrón y el soporte de un patrón se basa en el máximo número de ocurrencias disjuntas (no traslapadas) del patrón.

El Algoritmo 1, que utiliza las funciones Explorar, Recorrer, Expandir y ExpandirOcurrencias, constituye el pseudocódigo de AGraP. El algoritmo inicialmente selecciona un nodo v en G y emplea la función Explorar para encontrar todos los grafos conectados en G que incluyen a v . Posteriormente el vértice se marca como “explorado” y no se toma en cuenta en la generación de patrones posteriores. La función Explorar, junto con las funciones Recorrer y Expandir, genera patrones que se pueden construir a partir del vértice v . Primero, la función Recorrer identifica los vértices conectados a v (o, en llamadas posteriores, a un patrón dado P) que no han sido explorados y crea el conjunto V_{expand} . La función Expandir hace crecer al patrón P (inicialmente $P = \{v\}$) a cada vértice en V_{expand} y explora, mediante llamadas recursivas a Recorrer, quien a su vez llama a Expandir, los nuevos patrones.

Las diferencias entre etiquetas de los vértices son permitidas mediante el manejo de un diccionario D (línea 4 en la función Explorar y líneas 5 y 15 en la función ExpandirOcurrencias), con las etiquetas de los vértices como llaves, que indica qué sustituciones son permitidas.

En la función ExpandirOcurrencias se identifican las ocurrencias de un patrón dado, y es ahí donde se permiten las diferencias estructurales entre grafos. Como se mencionó antes, nuestro algoritmo sigue un enfoque de crecimiento de patrones, en el cual expandimos un patrón dado P a un nuevo patrón $P' = P \cup \{v\}$ agregando un vértice v . Cada vez que el patrón P crece, las ocurrencias de P se analizan para intentar expandirlas, añadiendo un nuevo vértice u que se corresponda con v (en caso de que exista). De esta manera, el conjunto M'_P de ocurrencias de P' se construye a partir del conjunto M_P de ocurrencias de P .

En ExpandirOcurrencias, es decir, al momento de construir el conjunto M'_P se implementan las ideas descritas en las secciones 4.2.1 (líneas 11-17), 4.2.2 (líneas 3-10) y 4.2.3 (al momento de calcular la distancia de edición), conservando en el conjunto M'_P las ocurrencias que no pudieron crecer, así como las ocurrencias en donde el vértice añadido se conectó al subgrafo por un camino (en lugar de una arista).

Al momento de expandir cada ocurrencia en M_P , mantenemos un registro de la distancia de edición que existe entre la ocurrencia expandida y el patrón P' . Las diferencias entre los grafos comparados se cuentan

Algoritmo 1: Algoritmo AGraP

Entrada: G : grafo que se está analizando,
 σ : umbral de frecuencia,
 Δ : umbral de similitud,
 D : diccionario de equivalencia entre etiquetas (*opcional*)

Salida: P : conjunto de patrones frecuentes en G .

for $v \in G$ **do**

└ explorado(v) \leftarrow Falso;

$P \leftarrow \emptyset$;

for $v \in G$ **do**

└ explorado(v) \leftarrow Verdadero;
└ $P_v \leftarrow$ Explorar(G, σ, Δ, D, v);
└ $P \leftarrow P \cup P_v$;

Función Explorar(G, σ, Δ, D, v)

Entrada: G : grafo que se está analizando,
 σ : umbral de frecuencia,
 Δ : umbral de similitud,
 D : diccionario de equivalencia entre etiquetas (*opcional*),
 v : vértice en V_G a partir del cual se generarán patrones

Salida: P_v : conjunto de patrones con inicio en el vértice v

1 for $u \in G$ **do**

2 └ marcado(u) \leftarrow Falso;

3 $P_v \leftarrow \emptyset$;

4 $M_v \leftarrow$ Lista de vértices con la misma etiqueta que v o con una etiqueta equivalente de acuerdo al diccionario D ;

5 $C_v \leftarrow$ Lista de costos de edición entre cada vértice en M_v y el vértice-patrón v ;

6 if $|M_v| \geq \sigma$ **then**

7 └ $P_v \leftarrow P_v \cup \{v\}$

8 $P_{Recorrer} \leftarrow$ Recorrer($G, \{v\}, M_v, C_v, \sigma, \Delta, D$);

9 $P_v \leftarrow P_v \cup P_{Recorrer}$

Función Recorrer($G, P, M_P, C_P, \sigma, \Delta, D$)

Entrada: G : grafo que se está analizando,
 P : candidato a patrón,
 M_P : lista de ocurrencias de P ,
 C_P : lista con costos de edición entre P y sus ocurrencias,
 σ : umbral de frecuencia,
 Δ : umbral de similitud,
 D : diccionario de equivalencia entre etiquetas (*opcional*)

Salida: P_{Exp} : conjunto de patrones que se pueden obtener haciendo crecer P .

```
1  $V_{expand} \leftarrow$  Vértices conectados a  $P$  que no han sido explorados ni marcados;  
2 for  $u \in V_{expand}$  do  
3    $\text{marcado}(u) \leftarrow$  Verdadero;  
4  $P_{Exp} \leftarrow$  Expandir( $G, P, M_P, C_P, V_{expand}, \sigma, \Delta, D$ );  
5 for  $u \in V_{expand}$  do  
6    $\text{marcado}(u) \leftarrow$  Falso;
```

Función Expandir($G, P, M_P, C_P, V_{expand}, \sigma, \Delta, D$)

Entrada: G : grafo que se está analizando,
 P : candidato a patrón,
 M_P : lista de ocurrencias de P ,
 C_P : lista con costos de edición entre P y sus ocurrencias,
 V_{expand} : lista de vértices sin explorar conectados a P ,
 σ : umbral de frecuencia,
 Δ : umbral de similitud,
 D : diccionario de equivalencia entre etiquetas (*opcional*)

Salida: P_P : conjunto de patrones obtenidos expandiendo P .

```
1  $P_P, P_H, P_V \leftarrow \emptyset$ ;  
2 for  $v_{exp} \in V_{expand}$  do  
3    $P' \leftarrow P \cup \{v_{exp}\}$ ;  
4    $M'_P, C'_P \leftarrow$  ExpandirOcurrencias( $G, P, M_P, C_P, v_{exp}, \Delta, D$ );  
5    $sup_{P'} \leftarrow$  soporte del patrón  $P'$ ;  
6   if  $sup_{P'} \geq \sigma$  then  
7      $P_P \leftarrow P_P \cup P'$   
8   if  $|M'_P| > \sigma$  then  
9      $V'_{expand} \leftarrow V_{expand} \setminus \{v_{exp}\}$ ;  
10     $P_H \leftarrow$  Expandir( $G, P', M'_P, C'_P, V'_{expand}, \sigma, \Delta, D$ );  
11     $P_V \leftarrow$  Recorrer( $G, P', M'_P, C'_P, \sigma, \Delta, D$ );  
12  $P_P \leftarrow P_P \cup P_H \cup P_V$ ;
```

Función ExpandirOcurrencias($G, P, M_P, C_P, newVertex, \Delta, D$)

Entrada: G : grafo que se está analizando,

P : candidato a patrón,

M_P : lista de ocurrencias de P ,

C_P : lista con costos de edición entre P y sus ocurrencias,

$newVertex$: vértice conectado a P Δ : umbral de similitud,

D : diccionario de equivalencia entre etiquetas (*opcional*)

Salida: M'_P : lista de ocurrencias de $P' = P \cup \{newVertex\}$,

C'_P : lista con el costo de edición asociado a cada ocurrencia en M'_P (con respecto al nuevo patrón P')

1 $M'_P, C'_P \leftarrow \emptyset$;

2 **for** *ocurrencia* O **en** M_P **do**

 /* Analizando vértices correspondientes con $newVertex$ y
 conectados, por una o varias aristas, a la ocurrencia O */

3 $v_Vecinos \leftarrow$ Lista de vértices en $V_G \setminus O$ conectados a O ;

4 **for** $vertexV \in v_Vecinos$ **do**

5 **if** $label(vertexV) = label(newVertex)$ **OR** $label(vertexV) \in D[newVertex]$ **then**

6 Agregar $O \cup \{vertexV\}$ a la lista M'_P ;

7 Calcular el costo de edición entre la nueva ocurrencia y P' , y agregarlo a C'_P ;

8 **else**

9 Agregar O a la lista M'_P y marcar la ausencia de un nuevo vértice con el símbolo '-';

10 Calcular el costo de edición entre O y P' y agregarlo a C'_P ;

 /* Analizando vértices conectados a O por caminos */

11 $\ell \leftarrow$ Estimación de la máxima longitud que puede tener un camino entre la ocurrencia O y un
 nuevo vértice, sin que se sobrepase el umbral de disimilitud Δ ;

12 **for** $i = 2 : \ell$ **do**

13 $vertCamino \leftarrow$ Vértices en $V_G \setminus (O \cup v_Vecinos)$ conectados a O por un camino de
 longitud i ;

14 **for** $vertexC \in vertCamino$ **do**

15 **if** $label(vertexC) = label(newVertex)$ **OR** $label(vertexC) \in D[newVertex]$ **then**

16 Agregar $O \cup \{vertexC\}$ a la lista M'_P ;

17 Calcular el costo de edición entre la nueva ocurrencia y P' , y agregarlo a C'_P ;

por separado para los vértices y las aristas, de manera que, después de ser expandida, cada ocurrencia tiene asociados dos números, v_{edit} y e_{edit} , que cuantifican sus diferencias, en vértices y aristas (respectivamente), respecto al patrón P' . En este proceso simplemente se cuentan los pasos de edición, sin tomar en cuenta el umbral de disimilitud Δ .

Finalmente, para establecer el soporte de cada patrón se selecciona, de la lista de ocurrencias M'_P y utilizando los costos de edición asociados a cada una de ellas, al conjunto de ocurrencias cuya disimilitud con el patrón P' está por debajo del umbral Δ . Con esta sublista de ocurrencias se calcula el soporte del patrón usando la definición dada por Bringmann y Nijssen [BN08], que se basa en la mínima imagen de un grafo y se describió en la sección 2.1.

4.4. Experimentos

Se realizaron dos tipos de experimentos. En el primer experimento se utilizaron grafos pequeños contruidos con el propósito de examinar el tipo de patrones encontrados y mostrar los alcances y limitaciones que en este momento tiene el algoritmo propuesto. En el segundo experimento se utilizaron grafos un poco más grandes para observar cómo varía la cantidad de patrones obtenidos por nuestro algoritmo para diferentes valores del umbral de frecuencia. En ambos experimentos, se compararon los patrones obtenidos con nuestro algoritmo con los patrones encontrados por gApprox [CYZH07], que es el algoritmo más cercano a nuestro trabajo y el único que hemos encontrado para buscar patrones frecuentes en un solo grafo con correspondencia inexacta.

Los experimentos se llevaron a cabo en una computadora con procesador Intel Core i7 (3.4 Ghz) con 16GB en RAM, corriendo bajo el sistema operativo GNU Linux con distribución Ubuntu 11.10 de 64 bits. La implementación de nuestro algoritmo y de gApprox se realizó en Python, utilizando la biblioteca de funciones NetworkX.

4.4.1. Experimento 1

En este experimento utilizamos grafos pequeños para comparar, uno por uno, los patrones frecuentes obtenidos con AGraP contra los obtenidos con gApprox. Los grafos usados en este experimento se muestran en la Fig. 7.

El grafo G_1 (Fig. 7a) se usó en la sección 4.2 para ejemplificar el tipo de patrones de nuestro interés. En la Tabla 1 pueden observarse los patrones frecuentes encontrados con AGraP y con gApprox usando $\sigma = 2$ y $\Delta = 4$, sin tener equivalencia entre etiquetas. La primera columna muestra los patrones encontrados; en la segunda, se enlistan las ocurrencias dentro de G_1 de cada patrón, junto con los costos de edición, para vértices y aristas, respecto al patrón. En la lista de los vértices que componen cada ocurrencia se señala a cada vértice faltante con el símbolo '- ', mientras que un signo negativo precediendo al identificador de un vértice indica que éste “sobra”. Para observar el tipo de ocurrencias que AGraP encuentra, en la segunda columna se muestran todas las ocurrencias que se identificaron de un patrón, sin tomar en cuenta si su similitud con el patrón está por debajo del umbral Δ (al calcular el soporte sólo se toman en cuenta las ocurrencias que son suficientemente similares al patrón). Finalmente, en la tercer y cuarta columnas se muestra el soporte de cada patrón en G_1 ; las letras N.E. (No Encontrado) se usan para mostrar los casos en que gApprox no encontró un determinado patrón. Es importante mencionar que, para los grafos G_1 y G_2 , utilizamos las mismas funciones que gApprox para establecer la similitud entre grafos (la suma del costo de edición en vértices y aristas, expresión (4.1)) y el soporte de cada patrón (de ahí que el valor de Δ no esté entre 0 y 1).

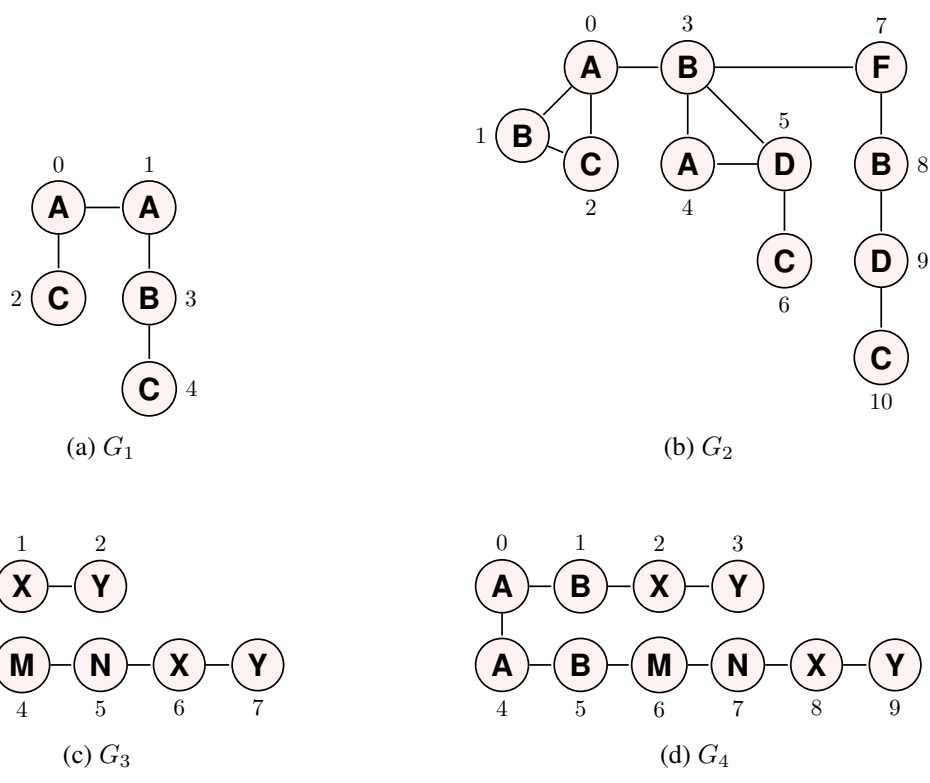


Figura 7: Grafos que se analizaron para comparar los patrones encontrados por *gApprox* [CYZH07] contra los patrones encontrados por el algoritmo propuesto.

Cuadro 1: Patrones frecuentes encontrados al analizar el grafo G_1 que se muestra en la Fig. 7, usando $\sigma = 2$ y $\Delta = 4$. En la segunda columna, el símbolo '-' indica la ausencia de un vértice, mientras que un signo negativo precede a un vértice que "sobra". La expresión N.E. en la tercera columna indica que el algoritmo correspondiente no encontró el patrón que se muestra en la primera columna de esa fila.

Patrón	Ocurrencias encontradas $\{Vértices\} / (v_{edit}, e_{edit})$	Soporte	
		gApprox	AGraP
(A)	$\{0\} / (0, 0)$ $\{1\} / (0, 0)$	2	2
(C)	$\{2\} / (0, 0)$ $\{4\} / (0, 0)$	2	2
	$\{0, 2\} / (0, 0)$		
(A) (C)	<i>Sólo AGraP:</i> $\{0, -\} / (1, 1)$ $\{1, -0, 2\} / (1, 3)$ $\{1, -3, 4\} / (1, 3)$ $\{1, -\} / (1, 1)$	1/N.E.	2
(A) (B)	<i>Sólo AGraP:</i> $\{1, 3\} / (0, 0)$ $\{1, -\} / (1, 1)$ $\{0, -1, 3\} / (1, 3)$ $\{0, -\} / (1, 1)$	1/N.E.	2
(A) (B) (C)	<i>Sólo AGraP:</i> $\{1, 3, 4\} / (0, 0)$ $\{1, 3, -0, 2\} / (1, 3)$ $\{1, 3, -\} / (1, 1)$ $\{1, -, -\} / (2, 2)$ $\{0, -1, 3, 2\} / (1, 5)$ $\{0, -, 2\} / (1, 3)$	1/N.E.	2

Como puede observarse en la Tabla 1, al analizar G_1 con los parámetros especificados, gApprox encontró dos patrones y AGraP encontró cinco: los dos que encontró gApprox y tres patrones adicionales. En este primer ejemplo se muestra que AGraP es capaz de encontrar los mismos patrones que gApprox, más algunos otros que pasan desapercibidos cuando no se permiten diferencias estructurales en los vértices.

Al analizar el grafo G_2 (Fig. 7) con $\sigma = 2$, $\Delta = 4$ y aceptando la etiqueta ‘F’ como reemplazo de la etiqueta ‘A’, se obtuvieron 23 patrones con gApprox y 63 con AGraP (los 23 que encontró gApprox y 40 adicionales). En la Tabla 2 se muestran dos de los patrones frecuentes que se encuentran con AGraP pero no con gApprox. En el primer caso se trata de un patrón que pasa desapercibido al utilizar gApprox, ya que, sin permitir diferencias estructurales en los vértices, aparece sólo una vez en G_2 . En el segundo caso se trata de un patrón que gApprox tampoco encuentra porque no puede detectar suficientes ocurrencias del mismo para que su soporte esté por encima del umbral de frecuencia. Con este ejemplo podemos ver que permitir diferencias estructurales en los vértices, durante la búsqueda de patrones frecuentes, hace posible encontrar patrones que de otra manera pasarían desapercibidos.

Es muy importante subrayar que, en todos los resultados que se analizaron en este grupo de experimentos, pudo constatar que AGraP encuentra todos los patrones frecuentes que encuentra gApprox, más otros patrones adicionales.

Examinar los patrones frecuentes encontrados con AGraP al analizar los grafos G_3 y G_4 (Fig. 7), usando la función de similitud f_2 propuesta en la sección 4.1, permite observar una de las limitaciones de nuestro algoritmo.

Al buscar patrones frecuentes en G_3 usando $\sigma = 2$, $\Delta = 0.5$ y ninguna equivalencia entre etiquetas, el algoritmo no devuelve el patrón P_2 , que se muestra en la Tabla 3, aunque esperaríamos que $\{0, 1, 2\}$ y $\{3, -4, -5, 6, 7\}$ se consideraran ocurrencias del patrón (la segunda con vértices, 4 y 5, que sobran) y que, por lo tanto, el patrón fuera frecuente.

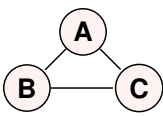
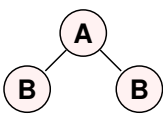
¿Por qué AGraP no identifica al patrón P_2 como frecuente? Con la medida de similitud usada (f_2), debido a la cota ℓ que se impone al número de vértices que pueden sobrar (ver sección 4.2.1) el patrón P_1 no puede tener ocurrencias en dónde sobren vértices y, por lo tanto, los vértices $\{3, 4, 5, 6\}$ no forman una ocurrencia del patrón. Al extender el patrón P_1 al patrón P_2 (ver Tabla 3), la similitud entre P_2 y el subgrafo formado por $\{3, 4, 5, 6, 7\}$ se encuentra por debajo de Δ , pero, puesto que el conjunto de ocurrencias de P_2 se forma extendiendo las ocurrencias de P_1 , no es posible encontrar esta ocurrencia de P_2 y el patrón no se identifica como frecuente. Es importante notar que sucedería lo mismo aun cuando los vértices 2 y 7 de G_3 representaran subgrafos idénticos de cualquier tamaño.

En contraste, el patrón P_3 (Tabla 3) sí puede identificarse en G_4 (el valor de ℓ al formar el patrón P_3 es 2, y, por lo tanto, los vértices $\{4, 5, 6, 7, 8\}$ sí representan una ocurrencia del patrón) y, en consecuencia, también es posible encontrar al patrón P_4 , que se forma extendiendo P_3 .

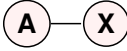
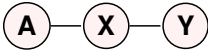

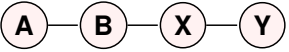
Con esto se muestra que hay patrones que, a pesar de satisfacer los valores de σ y Δ especificados, podrían no ser encontrados con nuestro algoritmo, debido a la cota que se impone al número de vértices que pueden sobrar y que, por el momento, se usa para reducir el costo de la búsqueda.

En resumen, con base en los resultados obtenidos en este primer experimento, puede afirmarse que AGraP encuentra patrones que tienen diferencias estructurales, tanto en vértices como en aristas, con sus ocurrencias, aunque es posible que no encuentre *todos* los patrones que satisfacen los umbrales σ y Δ proporcionados por el usuario. No obstante, se muestra (aunque falta demostrarlo teóricamente, lo cual se hará más adelante) que AGraP es capaz de encontrar todos los patrones que gApprox encuentra, y, adicionalmente, encontrar patrones que no se detectan usando gApprox.

Cuadro 2: Algunos patrones frecuentes encontrados al analizar el grafo G_2 que se muestra en la Fig. 7, usando $\sigma = 2$, $\Delta = 4$ y especificando que la etiqueta 'F' puede reemplazar a la etiqueta 'A'. En la segunda columna, el símbolo '-' indica la ausencia de un vértice, mientras que un signo negativo precede a un vértice que "sobra". La expresión N.E. en la tercera columna indica que el algoritmo correspondiente no encontró el patrón que se muestra en la primera columna de esa fila.

Patrón	Ocurrencias encontradas $\{Vértices\} / (v_{edit}, e_{edit})$	Soporte	
		gApprox	AGraP
	$\{0, 1, 2\} / (0, 0)$		
	$\{0, 3, 2\} / (0, 1)$		
	<i>Sólo AGraP:</i>		
	$\{0, 1, -\} / (1, 1)$		
	$\{0, 3, -1, 6\} / (1, 4)$		
	$\{4, 3, -1, 2\} / (1, 4)$		
	$\{4, 3, -1, 6\} / (1, 4)$	1/N. E.	3
	$\{4, 3, -\} / (1, 1)$		
	$\{7, 8, -1, 10\} / (2, 4)$		
	$\{7, 8, -\} / (2, 1)$		
	$\{7, 3, -1, 2\} / (2, 4)$		
	$\{7, 3, -1, 6\} / (2, 4)$		
	$\{7, 3, -\} / (2, 1)$		
		$\{0, 1, 3\} / (0, 0)$	
$\{7, 3, 8\} / (1, 0)$			
$\{7, 8, 3\} / (1, 0)$			
<i>Sólo AGraP:</i>			
$\{0, 1, -\} / (1, 1)$			
$\{0, 3, 1\} / (0, 0)$		1/N.E.	2
$\{0, 3, -1, 8\} / (1, 3)$			
$\{4, 3, -1, 1\} / (1, 3)$			
$\{4, 3, -1, 8\} / (1, 3)$			
$\{4, 3, -\} / (1, 1)$			
$\{7, 3, -1, 1\} / (2, 3)$			

Cuadro 3: Algunos patrones frecuentes en los grafos G_3 y G_4 que se muestran en la Fig. 7. AGraP se usó con $\sigma = 2$, $\Delta = 0.5$ y la función de correspondencia inexacta f_2 . En la segunda columna, el símbolo '-' indica la ausencia de un vértice, mientras que un signo negativo precede a un vértice que "sobra". La expresión N.E. en la tercera columna indica que el algoritmo correspondiente no encontró el patrón que se muestra en la primera columna de esa fila.

	Patrón	Ocurrencias encontradas $\{Vértices\} / (v_{edit}, e_{edit})$	Soporte AGraP	
En G_3 :	P_1 	$\{0, 1\} / (0, 0)$	1/N.E.	
		$\{0, -\} / (1, 1)$		
		$\{3, -\} / (1, 1)$		
	P_2 	$\{0, 1, -\} / (1, 1)$	1/N.E.	
		$\{0, -, -\} / (2, 2)$		
		$\{3, -, -\} / (2, 2)$		
En G_4 :	P_3 	$\{0, 1, 2\} / (0, 0)$	2	
		$\{0, 1, -\} / (1, 1)$		
		$\{0, -, -\} / (2, 2)$		
		$\{4, 5, -0, -1, 2\} / (2, 4)$		
		$\{4, 5, -6, -7, 8\} / (2, 4)$		
			$\{4, 5, -\} / (1, 1)$	
	P_4 	$\{0, 1, 2, 3\} / (0, 0)$	2	
		$\{0, 1, 2, -\} / (1, 1)$		
		$\{0, 1, -, -2, 3\} / (2, 4)$		
		$\{0, 1, -, -\} / (2, 2)$		
$\{0, -, -, -\} / (3, 3)$				
$\{4, 5, -0, -1, 2, 3\} / (2, 4)$				
		$\{4, 5, -6, -7, 8, 9\} / (2, 4)$		
		$\{4, 5, -, -\} / (2, 2)$		

Cuadro 4: Características de los grafos usados en el experimento 2. La cuarta columna señala la cantidad de etiquetas en vértices. La quinta columna indica los parámetros que se usaron en el generador de grafos de Kuramochi y Karypis [KK01] para obtener el grafo correspondiente.

Grafo	Vértices	Aristas	Etiquetas	Parámetros en generador					
G_{100}	100	346	20	-D1	-E1	-i87	-L1730	-T340	-V20
G_{150}	150	716	30	-D1	-E1	-i188	-L3750	-T750	-V30
G_{175}	175	1303	35	-D1	-E1	-i335	-L6700	-T1340	-V35
G_{200}	200	1455	40	-D1	-E1	-i375	-L7500	-T1500	-V40

Cuadro 5: Cantidad de patrones encontrados con AGrAP y gApprox en el experimento 2, usando $\Delta = 4$ y distintos valores de σ .

σ	G_{100}		G_{150}		G_{175}		G_{200}	
	AGraP	gApprox	AGraP	gApprox	AGraP	gApprox	AGraP	gApprox
3	9,329	1,835	8,275	893	121,758	15,205	121,869	10,358
4	1,650	300	1,190	208	16,228	1,783	11,354	949
5	425	129	325	123	2,700	314	1,523	182
6	152	66	113	87	499	114	328	116
7	81	54	45	45	137	67	83	68

4.4.2. Experimento 2

En el segundo experimento se utilizaron grafos más grandes y se analizó la cantidad de patrones encontrados por AGrAP para diferentes valores del umbral de frecuencia σ . En todos los casos se comparó la cantidad de patrones encontrados contra la cantidad de patrones que encuentra gApprox; para esto se usaron en AGrAP las funciones de similitud y de soporte que gApprox utiliza.

Para este experimento se generaron datos usando el generador de grafos desarrollado por Kuramochi *et al.* [KK01], también empleado por Chen *et al.* en los experimentos que reportan sobre gApprox en [CYZH07]. La Tabla 4 muestra las características de los grafos usados. En la última columna se muestra el valor de los parámetros involucrados en la generación de los grafos; de acuerdo con la documentación del generador de grafos, dichos parámetros son: cantidad de transacciones (D), cantidad de etiquetas en aristas (E), tamaño promedio de un grafo frecuente (i), cantidad de grafos frecuentes (L), tamaño promedio de cada grafo (T) y cantidad de etiquetas en vértices (V).

La Tabla 5 muestra la cantidad de patrones encontrados en este experimento para cada grafo y valor de σ ; la Fig. 8 muestra la proporción de patrones encontrados por AGrAP respecto a los patrones encontrados por gApprox. La proporción es en todos los casos mayor o igual a 1, lo cual era de esperarse dado que AGrAP encuentra los patrones que gApprox encuentra, más algunos otros adicionales. Puede observarse que, en general, un umbral más pequeño se traduce en un incremento de la proporción de patrones adicionales que AGrAP encuentra, llegando a encontrar (para el grafo G_{200} con $\sigma = 4$) hasta doce veces la cantidad de los patrones que gApprox encuentra. Esto nos lleva a concluir que AGrAP, al permitir diferencias estructurales tanto en vértices como en aristas, identifica una cantidad considerable de patrones que de otra forma pasarían desapercibidos; de esta manera el algoritmo propuesto cumple el propósito con el que fue diseñado.

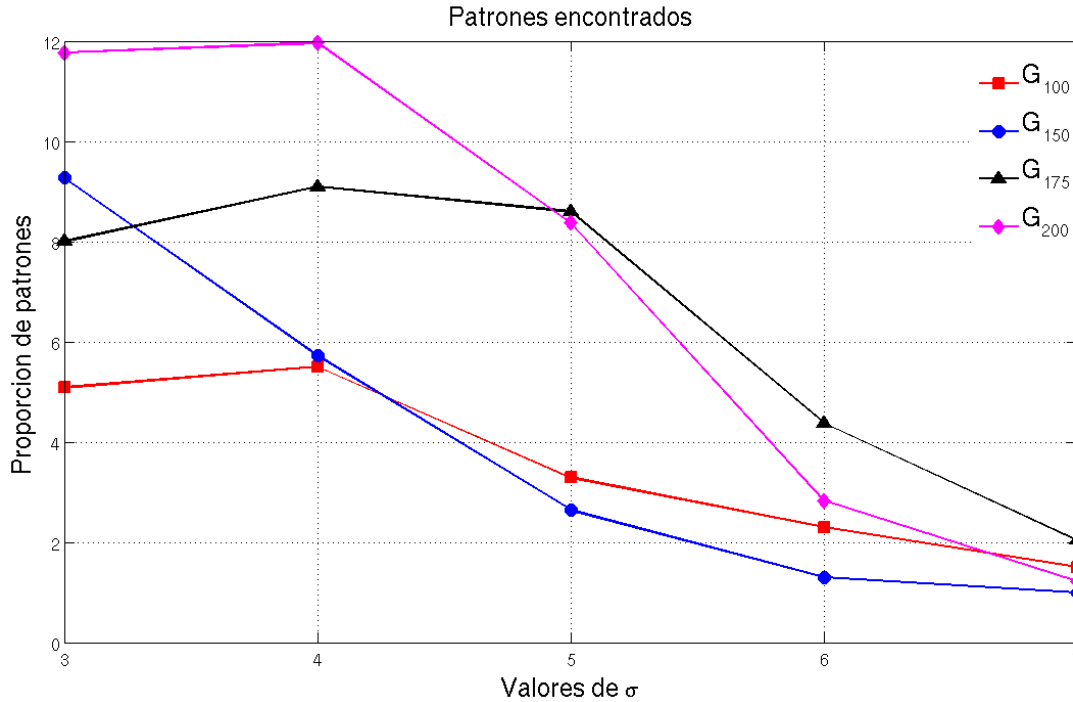


Figura 8: *Proporci3n de patrones encontrados por AGraP, relativa a la cantidad de patrones encontrados por gApprox, usando $\Delta = 4$ y diferentes valores de σ .*

La Fig. 9 muestra el tiempo de ejecuci3n que requiri3 cada algoritmo en este experimento. Cabe mencionar que, puesto que en este experimento se utiliza la medida de similitud f_1 , que es antimonot3nica, fue posible podar el espacio de b3squeda y tener un desempe1o aceptable del algoritmo propuesto. Sin embargo, tomando en cuenta que queremos usar la medida de similitud f_2 , consideramos que el siguiente paso importante en el desarrollo del algoritmo es encontrar una forma de lidiar con la falta de antimonotonicidad de f_2 , de manera que el tiempo de ejecuci3n requerido no sea un obst3culo para analizar grafos del tama1o considerado en este experimento.

5. Conclusiones

La b3squeda de patrones en un grafo usando correspondencia inexacta, de manera que los patrones puedan tener diferencias estructurales en v3rtices y aristas, respecto a los subgrafos que son considerados sus ocurrencias, es un problema que plantea dificultades no triviales. Se debe lidiar con aspectos que van desde la definici3n de una funci3n de similitud hasta la p3rdida de la propiedad de antimonotonicidad, que sirve para podar el espacio de b3squeda; adem3s, al tratarse de patrones en un solo grafo es necesario establecer una forma de evaluar el soporte de cada patr3n que tome en cuenta los posibles traslapes entre ocurrencias.

En esta propuesta de investigaci3n se plantea el desarrollo de un algoritmo para la b3squeda de patrones frecuentes interesantes en un solo grafo utilizando correspondencia inexacta. No s3lo se desea encontrar patrones frecuentes, sino obtener un subconjunto de los mismos que sea interesante y que, por ser m3s peque1o que el conjunto de grafos frecuentes, resulte m3s f3cil de analizar y m3s 3til en ciertas aplicaciones. En particular, se plantea la posibilidad de usar el algoritmo propuesto en el contexto de grafos din3micos.

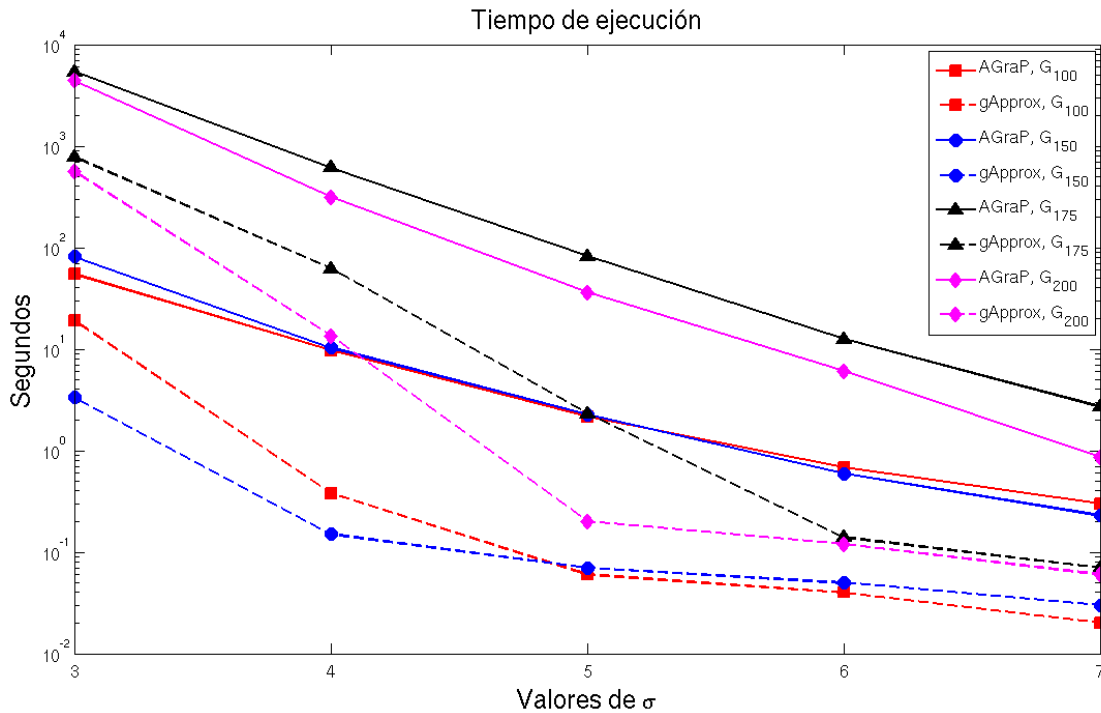


Figura 9: Tiempo requerido por AGraP y gApprox para encontrar los patrones frecuentes en los grafos del experimento 2, usando $\Delta = 4$ y diferentes valores de σ . El tiempo se muestra en segundos y en escala logarítmica.

Como parte de los resultados preliminares se propuso un algoritmo que permite encontrar patrones frecuentes en un solo grafo, que permite diferencias estructurales en vértices y aristas e identifica patrones que pasan desapercibidos para otros algoritmos del estado del arte. Para definir este algoritmo se propuso una función de comparación entre grafos que está basada en costos de edición y que toma en cuenta el tamaño de los grafos que se se están comparando. También se propusieron estrategias para identificar ocurrencias con diferencias estructurales en vértices, de manera que éstos puedan “faltar” o “sobrar”, respecto al patrón que se considera.

Aunque el desempeño del algoritmo propuesto es aceptable cuando se usa una función de correspondencia inexacta antimotónica (como f_1), para funciones que no sean antimotónicas (como f_2) la poda usada no puede ser aplicada, lo cual afecta negativamente su desempeño a medida que se incrementa el tamaño de los grafos analizados. Este problema y propuestas para su solución se explorarán más adelante, en el transcurso de la investigación. El siguiente paso en nuestra investigación consistirá en definir una medida de soporte para, tomando en cuenta el uso de correspondencia inexacta, determinar la frecuencia de un patrón dentro de un grafo.

Con base en los resultados reportados en la literatura y el análisis realizado en este trabajo, podemos decir que el problema planteado es importante. Además, a pesar de que nuestro algoritmo se encuentra en construcción, los resultados preliminares muestran que la solución al problema propuesto es alcanzable, en el tiempo planteado, siguiendo la metodología descrita.

Referencias

- [AAG11] R. Alguliev, R. Aliguliyev, and F. Ganjaliyev. Extracting a Heterogeneous Social Network of Academic Researchers on the Web Based on Information Retrieved from Multiple Sources. *American Journal of Operations Research*, 1(2):33–38, 2011.
- [AC05] M. Aery and S. Chakravarthy. eMailSift: Email Classification Based on Structure and Content. In *Proceedings of the Fifth IEEE International Conference on Data Mining, ICDM '05*, pages 18–25. IEEE Computer Society, 2005.
- [AHCS+05] M. Al-Hasan, V. Chaoji, S. Salem, N. Parimi, and M. J. Zaki. DMTL: a generic data mining template library. In *In Workshop on Library-Centric Software Design (w/ OOPSLA)*, 2005.
- [AHCS+07] M. Al-Hasan, V. Chaoji, S. Salem, J. Besson, and M. J. Zaki. Origami: Mining representative orthogonal graph patterns. In *ICDM*, pages 153–162. IEEE Computer Society, 2007.
- [AW10] C. C. Aggarwal and H. Wang. *Managing and Mining Graph Data*. Springer, 2010.
- [BB02] C. Borgelt and M.R. Berthold. Mining molecular fragments: Finding relevant substructures of molecules. In *Proceedings. 2002 IEEE International Conference on Data Mining*, pages 51–58, 2002.
- [BKW06] Karsten M. Borgwardt, Hans-Peter Kriegel, and Peter Wackersreuther. Pattern mining in frequent dynamic subgraphs. In *ICDM*, pages 818–822. IEEE Computer Society, 2006.
- [BL06] F. Bonchi and C. Lucchese. On condensed representations of constrained frequent patterns. *Knowl. Inf. Syst.*, 9(2):180–201, 2006.
- [BN08] B. Bringmann and S. Nijssen. What is frequent in a single graph? In T. Washio, E. Suzuki, K. Ting, and A. Inokuchi, editors, *Advances in Knowledge Discovery and Data Mining*, volume 5012 of *Lecture Notes in Computer Science*, pages 858–863. Springer Berlin / Heidelberg, 2008.
- [BR11] Horst Bunke and Kaspar Riesen. Recent advances in graph-based pattern recognition with applications in document analysis. *Pattern Recogn.*, 44:1057–1067, May 2011.
- [BS98] H. Bunke and K. Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recogn. Lett.*, 19(3-4):255–259, 1998.
- [Bun97] H. Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recogn. Lett.*, 18(9):689–694, 1997.
- [BZ09] B. Bringmann and A. Zimmermann. One in a million: picking the right patterns. *Knowl. Inf. Syst.*, 18(1):61–81, 2009.
- [Car09] K. Carley. Computational modeling for reasoning about the social behavior of humans. *Computational & Mathematical Organization Theory*, 15:47–59, 2009.

- [CBBLn05] R. M. Cesar, E. Bengoetxea, I. Bloch, and P. Larrañaga. Inexact graph matching for model-based recognition: Evaluation and comparison of optimization algorithms. *Pattern Recogn.*, 38(11):2099–2113, 2005.
- [CG02] T. Calders and B. Goethals. Mining all non-derivable frequent itemsets. In *Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery*, PKDD '02, pages 74–85. Springer-Verlag, 2002.
- [CH07] D.J. Cook and L.B. Holder. *Mining graph data*. Wiley-Interscience, 2007.
- [CYH10] H. Cheng, X. Yan, and J. Han. Mining graph patterns. In C. Aggarwal and H. Wang, editors, *Managing and Mining Graph Data*, volume 40 of *Advances in Database Systems*, pages 365–392. Springer, 2010.
- [CYZH07] C. Chen, X. Yan, F. Zhu, and J. Han. gapprox: Mining frequent approximate patterns from a massive network. In *ICDM*, pages 445–450. IEEE Computer Society, 2007.
- [DES07] M. Dehmer and F. Emmert-Streib. Comparing large graphs efficiently by margins of feature vectors. *Applied Mathematics and Computation*, 188(2):1699–1710, 2007.
- [DKWK05] M. Deshpande, M. Kuramochi, N. Wale, and G. Karypis. Frequent substructure-based approaches for classifying chemical compounds. *IEEE Trans. Knowl. Data Eng.*, 17(8):1036–1050, 2005.
- [DS04] P. Desikan and J. Srivastava. Mining temporally evolving graphs. In *Proceedings of the Sixth WEBKDD Workshop in conjunction with the 10th ACM SIGKDD conference*, pages 13–22. ACM Press, 2004.
- [EGK⁺04] S. Eubank, H. Guclu, VS. Kumar, M.V. Marathe, A. Srinivasan, Z. Toroczkai, and N. Wang. Modelling disease outbreaks in realistic urban social networks. *Nature*, 429:180–184, 2004.
- [EPL09] N. Eagle, A. Pentland, and D. Lazer. Inferring social network structure using mobile phone data,. In *Proceedings of the National Academy of Sciences (PNAS)*, volume 106, pages 15274–15278, 2009.
- [FB07] M. Fiedler and C. Borgelt. Support computation for mining frequent subgraphs in a single graph. In *5th Int. Workshop on Mining and Learning with Graphs*, pages 25–30, 2007.
- [FMNW03] D. Fetterly, M. Manasse, M. Najork, and J. Wiener. A large-scale study of the evolution of web pages. In *Proceedings of the 12th international conference on World Wide Web*, WWW '03, pages 669–678. ACM, 2003.
- [GAMPCOMT08] A. Gago-Alonso, J. Medina-Pagola, J. Carrasco-Ochoa, and J. Martínez-Trinidad. Mining frequent connected subgraphs reducing the number of candidates. In Walter Daelemans, Bart Goethals, and Katharina Morik, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 5211 of *Lecture Notes in Computer Science*, pages 365–376. Springer Berlin / Heidelberg, 2008.

- [Gar02] T. Gartner. Exponential an geometric kernels for graphs. In *NIPS*02 workshop on unreal data*, volume Principles of modeling nonvectorial data, 2002.
- [Gär08] T. Gärtner. *Kernels For Structured Data*. Series in machine perception and artificial intelligence. World Scientific, 2008.
- [GFW03] T. Gartner, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Conference on Learning Theory*, pages 129–143, 2003.
- [GXTL10] X. Gao, B. Xiao, D. Tao, and X. Li. A survey of graph edit distance. *Pattern Anal. Appl.*, 13(1):113–129, 2010.
- [HCD94] L. Holder, D. Cook, and S. Djoko. Substructure discovery in the subdue system. In *In Proc. of the AAAI Workshop on Knowledge Discovery in Databases*, pages 169–180, 1994.
- [HCR99] B. Huet, A. Cross, and Hancock. E. R. Shape retrieval by inexact graph matching. In *In: Proc. IEEE Int. Conf. on Multimedia Computing Systems. Volume 2., IEEE Computer*, pages 40–44. Society Press, 1999.
- [Hol88] L. B. Holder. Substructure discovery in subdue. Technical Report UILU-ENG-88-2220, Department of Computer Science, University of Illinois, Urbana, 1988.
- [HP04] D. Hidovic and M. Pelillo. Metrics for attributed graphs based on the maximal similarity common subgraph. *IJPRAI*, pages 299–313, 2004.
- [HWB⁺04] J. Huan, W. Wang, D. Bandyopadhyay, J. Snoeyink, J. Prins, and A. Tropsha. Mining spatial motifs from protein structure graphs. In *In Proc. of the 8th Annual Int. Conf. on Research in Computational Molecular Biology (RECOMB'04)*, pages 308–315, 2004.
- [HWPY04] J. Huan, W. Wang, J. Prins, and J. Yang. Spin: mining maximal frequent subgraphs from graph databases. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '04*, pages 581–586. ACM, 2004.
- [IW08] A. Inokuchi and T. Washio. A fast method to mine frequent subsequences from graph sequence data. In *ICDM*, pages 303–312. IEEE Computer Society, 2008.
- [JZH11] Y. Jia, J. Zhang, and J. Huan. An efficient graph-mining method for complicated and noisy data with real-world applications. *Knowl. Inf. Syst.*, 28(2):423–447, 2011.
- [KK01] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *Proceedings of the Int. Conf. Data Mining (ICDM'01)*, page 313–320, 2001.
- [KK04] M. Kuramochi and G. Karypis. Grew - a scalable frequent subgraph discovery algorithm. In *Proceedings of the Fourth IEEE International Conference on Data Mining*, pages 439 – 442, 2004.
- [KK05] M Kuramochi and G. Karypis. Finding frequent patterns in a large sparse graph^{*}. *Data Min. Knowl. Discov.*, 11(3):243–271, 2005.

- [KKLK02] H.J. Kim, I.M. Kim, Y. Lee, and B. Kahng. Scale-free network in stock markets. *J. Korean Phys. Soc.*, 40:1105–1108, 2002.
- [KKT03] D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pages 137–146. ACM, 2003.
- [KL02] R. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete input spaces. *International Conference on Machine Learning (ICML)*, 2002.
- [KLB09] J. Kunegis, A. Lommatzsch, and C. Bauckhage. The slashdot zoo: mining a social network with negative edges. In *World Wide Web Conference Series*, pages 741–750, 2009.
- [KM96] M. Kretzschmar and M. Morris. Measures of concurrency in networks and the spread of infectious disease. *Math Biosci.*, 133(2):165–195, 1996.
- [KNRT05] R. Kumar, J. Novak, P. Raghavan, and A. Tomkins. On the bursty evolution of blogspace. *World Wide Web*, 8(2):159–178, 2005.
- [KRSA11] V. Krishna, N.N.R. Ranga Suri, and G. Athithan. A comparative survey of algorithms for frequent subgraph discovery. *Curren Science*, 100(2):190–198, 2011.
- [KS05] P. S. Keila and D. B. Skillicorn. Structure in the enron email dataset. *Comput. Math. Organ. Theory*, 11(3):183–199, 2005.
- [KSL⁺10] J. Kunegis, S. Schmidt, A. Lommatzsch, J. Lerner, E. W. De Luca, and S. Albayrak. Spectral analysis of signed graphs for clustering, prediction and visualization. In *SDM*. SIAM, 2010.
- [KY04] B. Klimt and Y. Yang. Introducing the enron corpus. In *CEAS*, 2004.
- [Les09] J. Leskovec. Stanford Large Network Dataset Collection. <http://snap.stanford.edu/data/>, 2009.
- [Ley02] M. Ley. The DBLP computer science bibliography: Evolution, research issues, perspectives. In A. H. F. Laender and A. L. Oliveira, editors, *SPIRE*, volume 2476 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2002.
- [LGR⁺12] D.P. Jr. Lima, H.C. Giacomini, Takemoto R.M., Agostinho A.A., and Bini L.M. Patterns of interactions of a large fish-parasite network in a tropical floodplain. *Anim Ecol.*, 81(4):905–913, 2012.
- [LHK10] J. Leskovec, D. P. Huttenlocher, and J. M. Kleinberg. Signed networks in social media. In *Proceedings of the 28th International Conference on Human Factors in Computing Systems*, pages 1361–1370. ACM, 2010.
- [LP09] José Luis López-Presa. *Efficient Algorithms for Graph Isomorphism Testing*. PhD thesis, Universidad Rey Juan Carlos, Madrid, España, 2009.

- [Moo01] J. Moody. Peer influence groups: identifying dense clusters in large networks. *SOCIAL NETWORKS*, 23:261–283, 2001.
- [NB04] M. Neuhaus and H. Bunke. A probabilistic approach to learning costs for graph edit distance. In *ICPR (3)*, pages 389–393, 2004.
- [NB05] M. Neuhaus and H. Bunke. Self-organizing maps for learning the edit costs in graph matching. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 35(3):503–514, 2005.
- [NB06] M. Neuhaus and H. Bunke. A convolution edit kernel for error-tolerant graph matching. In *ICPR (4)*, pages 220–223. IEEE Computer Society, 2006.
- [NB07] M. Neuhaus and H. Bunke. *Bridging the Gap Between Graph Edit Distance and Kernel Machines*. World Scientific, 2007.
- [NK04] S. Nijssen and J. N. Kok. 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*, KDD '04, pages 647–652. ACM, 2004.
- [oT] University of Trier. The dblp computer science bibliography. Sitio web: <http://www.informatik.uni-trier.de/~ley/db/index.html>. Última consulta realizada el 21 de Junio de 2012.
- [PC11] J. Pfeffer and K. M. Carley. Modeling and calibrating real world interpersonal networks. In *Proceedings of the 2011 IEEE Network Science Workshop*, NSW '11, pages 9–16. IEEE Computer Society, 2011.
- [PDZH02] J. Pei, G. Dong, W. Zou, and J. Han. On computing condensed frequent pattern bases. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, ICDM '02, pages 378–. IEEE Computer Society, 2002.
- [PJZ05] J. Pei, D. Jiang, and A. Zhang. On mining cross-graph quasi-cliques. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, KDD '05, pages 228–238. ACM, 2005.
- [RG03] J. Ramon and T. Gartner. Expressivity versus efficiency of graph kernels. Technical report, First International Workshop on Mining Graphs, Trees and Sequences, 2003.
- [RKH03] A. Robles-Kelly and E. R. Hancock. Graph matching using spectral seriation and string edit distance. In *Proceedings of the 4th IAPR international conference on Graph based representations in pattern recognition*, GbRPR'03, pages 154–165. Springer-Verlag, 2003.
- [RKH04] A. Robles-Kelly and E. R. Hancock. String edit distance, random walks and graph matching. *Int. Journal of Pattern Recognition and Artificial Intelligence*, 18(3):315–327, 2004.
- [Rob09] Céline Robardet. Constraint-based pattern mining in dynamic graphs. In *ICDM*, pages 950–955. IEEE Computer Society, 2009.

- [RS09] S. Ranu and A.K. Singh. Graphsig: A scalable approach to mining significant subgraphs in large graph databases. In *IEEE 25th International Conference on Data Engineering*, pages 844 – 855, 2009.
- [SK03] A. J. Smola and I. R. Kondor. Kernels and regularization on graphs. In B. Schölkopf and M. K. Warmuth, editors, *Proc. Annual Conf. Computational Learning Theory*, pages 144–158, 2003.
- [SK11] M. E. Saeedy and P. Kalnis. GraMi: generalized frequent pattern mining in a single large graph. Technical report, Division of Mathematical and Computer Sciences and Engineering, King Abdullah University of Science and Technology, 2011.
- [SVP⁺09] N. Shervashidze, S. V. N. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt. Efficient graphlet kernels for large graph comparison. In *Proceedings of International Conference on Artificial Intelligence and Statistics*, 2009.
- [TSK06] P.N. Tan, M. Steinbach, and V. Kumar. *Introduction to data mining*. Pearson International Edition. Pearson Addison Wesley, 2006.
- [TVK10] L. T. Thomas, S. R. Valluri, and K. Karlapalem. Margin: Maximal frequent subgraph mining. *ACM Trans. Knowl. Discov. Data*, 4(3):10:1–10:42, 2010.
- [TWH05] J. Tyler, D. Wilkinson, and B. Huberman. E-mail as spectroscopy: Automated discovery of community structure within organizations. *The Information Society*, 21(2):143–153, 2005.
- [VBKS08] S. V. N. Vishwanathan, Karsten Borgwardt, Imre Kondor, and Nicol Schraudolph. Graph kernels. *Journal of Machine Learning Research*, 9:1–41, 2008.
- [XDW⁺08] Y. Xiao, H. Dong, W. Wu, M. Xiong, W. Wang, and B. Shi. Structure-based graph distance measures of high degree of precision. *Pattern Recognition*, 41(12):3547–3561, 2008.
- [YH02] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining, ICDM '02*, 2002.
- [YH03] X. Yan and J. Han. Closegraph: mining closed frequent graph patterns. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '03*, pages 286–295. ACM, 2003.
- [YH06] H. Yu and E. R. Hancock. String kernels for matching seriated graphs. *Pattern Recognition, International Conference on*, 4:224–228, 2006.
- [YHC08] C. H. You, L. B. Holder, and D. J. Cook. Graph-based data mining in dynamic networks: Empirical comparison of compression-based and frequency-based subgraph mining. In *ICDM Workshops*, pages 929–938. IEEE Computer Society, 2008.
- [YYH04] X. Yan, P. S. Yu, and J. Han. Graph indexing: A frequent structure-based approach. In *SIGMOD Conference*, pages 335–346. ACM, 2004.

- [ZLGZ09] Z. Zou, J. Li, H. Gao, and S. Zhang. Frequent subgraph pattern mining on uncertain graph data. In *Proceedings of the 18th ACM conference on Information and knowledge management, CIKM '09*, pages 583–592. ACM, 2009.
- [ZYC07] S. Zhang, J. Yang, and V. Cheedella. Monkey: Approximate graph mining based on spanning trees. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 1247–1249, april 2007.