Implementación de un método para generar coberturas de aristas en grafos simples.

Implementation of a method to generate edge covers on simple graphs.

L. E. Sierra-Alva[1], J. Raymundo Marcial-Romero[1], J. A. Hernández[1], Guiller-mo de Ita[2]
[1]Cerro de Coatepec s/n, Ciudad Universitaria, Facultad de Ingeniería, UAEM, 50100, Toluca, Estado de México, México.
[2]Avenida San Claudio y 14 sur, Ciudad Universitaria, Facultad de Ciencias de la Computación, BUAP, 72592, Puebla, México.
E-mail: darkdraco2008@hotmail.com; jrmarcialr@gmial.com; xoseahernandez@gmail.com ; deita@cs.buap.mx

PALABRAS CLAVE:

Coberturas de Aristas, Teoría de Grafos, Problemas #P.

RESUMEN

En este artículo se presenta un algoritmo para contar las coberturas de aristas de un grafo. El algoritmo, implementado en el lenguaje de programación C++, consiste en dividir el grafo original en subgrafos que cumplan con la propiedad de no tener cíclicos intersectados (ciclos compartidos). Cada subgrafo consti-tuye un nodo de un árbol de subgrafos en donde las hojas del árbol son grafos sin ciclos intersectados. Por lo tanto, el conteo de coberturas se puede realizar sobre las hojas del árbol.

KEYWORDS:

Edge Covers, Graph Theory, #P Problems.

ABSTRACT

In this article we present an algorithm for counting edge covers of graphs. The algorithm, implemented in the C language, divides the input graph into sub-graphs until no intersected cycles are presented (shared cycles). Each sub-graph constitutes a node of a tree where the leaves are su subgraphs without intersected cycles. Hence, the edge covers counting is carry on on the leaves of the tree.

1 INTRODUCCIÓN

La teoría de grafos siempre se ha relacionado con el área de las ciencias exactas y computacionales encasillándose muchas veces como una disciplina sin uso práctico fuera de este tipo de ciencias, y aunque es innegable esta estrecha relación, como en la creación de sistemas multiagentes [1] y las matemáticas [2] puede aplicarse en otras disciplinas.

La teoría de grafos y sus propiedades son útiles incluso en las ciencias y áreas menos pensadas; diversos estudios a lo largo de los años han comprobado su utilidad como herramienta o técnica para explicar fenómenos de áreas como la Biología [3], la Ecología [4,5], la Geografía [6,7], la Economía [8], la Construcción Urbana [9] e incluso en las de campo humanístico como la Psicología [10], la Medicina [11, 12], entre otras.

Existen problemas en teoría de grafos que por su carácter combinatorio pueden ser NP, si son de decisión o #P si son de conteo. Cada uno cuenta con sus propias características específicas, sin embargo históricamente los problemas #P han sido menos investigados que los NP, el conteo de coberturas de aristas forma parte de los problemas #P.

Formalmente, una cobertura de aristas de un grafo G es un conjunto de aristas C donde cada vértice es incidente con al menos una arista en C. Encontrar el total de las coberturas de aristas en un grafo está clasificado como un problema #P en general. Sin embargo, para grafos cuya topología es acíclica el problema se puede resolver en tiempo polinomial.

Trabajos anteriores han intentado darle solución al problema general, normalmente utilizado algoritmos heurísticos [13], que aproximan la solución, sin embargo, no se reportan algoritmos exactos cuya complejidad sea menor a la trivial 2^n.

El algoritmo propuesto en [14], es el primer intento en utilizar un método exacto para solucionar este problema, reduciendo el tiempo de complejidad trivial, sin embargo sigue siendo exponencial. En este artículo, se presenta un análisis e implementación del algoritmo presentado en [14]. La implementación fue desarrollada en el lenguaje de programación C++ para la realización de pruebas y comprobar su funcionamiento, a fin de poder generar una ecuación matemática que permita predecir su comportamiento.

2 Preliminares

Las siguientes convenciones y resultados son de uso común y pueden ser revisados en [15]. Un grafo es un par G= (V, E), donde V es el conjunto de vértices y E es el conjunto de aristas que asocian pares de vértices. El número de vértices y aristas es denotado por v(G) y e(G) respectivamente. El parámetro v(G) es llamado "el orden" mientras que e(G) es el tamaño del grafo.

Cada vértice v del grafo tiene asociado un número d_G (v), llamado el grado de v, y cuenta el número de aristas incidentes a v. Una arista con grado cero es llamada una arista aislada. Puede ser denotada por δ(G) y Δ(G), el mínimo y el máximo grado de un conjunto de vértices de G respectivamente. Es importante enfatizar que el conjunto de vértices V perteneciente al grafo G se denotara como V_G, del mismo modo el conjunto de E, mientras E_v si v pertenece V_G entonces denotamos el conjunto de aristas incidentes a v.

Un grafo simple es un grafo sin ponderación ni dirección, el cual no contiene bucles o múltiples aristas. Para este documento solo se consideraran grafos simples finitos los cuales serán considerados, G es un grafo finito si $|v(G)|< \infty$ y $|e(G)|< \infty$.

Un subgrafo de un grafo G es un grafo G'= (V',E') tal que V´ $\subseteq$ V) V y que E´ $\subseteq$ E) E. Los subgrafos pueden ser calculados removiendo aristas y vértices del grafo original. Si e pertenece E, e puede ser simplemente removida del grafo G, denotando complacientemente un subgrafo como G\e; es obvio que es el grafo (V,E-e). Analógicamente, si v pertenece V, G\v es el grafo (V-v,E') donde E´ $\subseteq$ E) E consiste de las aristas de E excepto aquellos incidentes a v.

Un subgrafo de expansión es un subgrafo que se crea por la eliminación del máximo número de aristas posibles mientras se mantengan todo los vértices conectados. Si S pertenece E es un subconjunto de E, entonces un subgrafo de expansión de G=(V,E) es G^'=G\S, tal que G' no tiene vértices aislados excepto por aquellos que ya son aislados en G.

Una ruta en un grafo es una secuencia lineal de vértices adyacentes, mientras un ciclo en un grafo G es un subgrafo en el cual cada vértice tiene un grado par. Un ciclo es llamado ciclo básico si es conectado y cada uno de sus vértices tiene grado dos.

Una ruta en un grafo y un clico en un grafo que contiene n aristas pueden ser llamados "n-path" y "n-cycle", respectivamente. El conjunto de rutas y ciclos de tamaño variable pueden ser denotados como P y C, respectivamente. Es importante enfatizar que un ciclo o

ruta en un grafo que pertenece al grafo G pueden usar la notación C_G o P_G, como corresponda.

La ruta de grafo que contiene n aristas puede ser precisamente definida como un subgrafo $P=(V_P,E_P)$ donde $V_P=\{v_i|i \text{ pertenece } N_n\}$ y $E_P=\{e_i|e_i=v_i v_{(i+1)}, i \text{ pertenece } N_{(n-1)}\}$, donde $v_i v_{(i+1)}$ representa la arista conectando el vértices v_i al vértice $v_{(i+1)}$; N_n denota el conjunto {1,…., n}. Es importante enfatizar que la ruta P inicia y termina en v,w, respectivamente se puede escribir P(v,w) en vez de P.

Un grafo acíclico es un grafo que no contiene ciclos. Los grafos acícliclos conectados son llamados árboles, y un grafo conectado es un grafo que para cualquier dos pares de vértices que tiene, existe una ruta conectando ambos. El número de componentes conectados de un grafo G es denotado como c(G). No es difícil inferir que en un árbol hay solo una única ruta conectando cada dos pares de vértices. Sea T(v) un árbol T con un vértice raíz v, los vértices en el árbol con un grado menor o igual a uno son llamados hojas.

Un conjunto de coberturas de aristas, o simplemente una e-covert de un grafo G es un subconjunto de aristas en E_G, tal que se encuentra con todos los vértices de G. En otras palabras, un conjunto de coberturas de aristas de G= (V_G,E_G) es una familia ε_G tal que U_(Spertenece ε_G)= E_G que conoce todos los vértices de G. Por lo tanto, los problemas de conteo de conjuntos de coberturas de aristas pueden ser simplemente enunciados como: Teniendo un grafo simple G, sin vértices aislados, entonces el conteo del número total de diferentes conjuntos de coberturas de aristas para el grafo G es simplemente el cálculo de |ε_G |.

3Conteo de Coberturas de Aristas en Grafos con Ciclos

Sea G un grafo conectado, T_G puede denotar un árbol de expansión construido de G. Notar que V(T_G)=V(G). Las aristas en T_G son llamadas aristas del árbol, mientras que las aristas en E(G)-E(T_G) son llamadas aristas de retroceso. Sea e una arista de retroceso dada, la unión de la ruta en T_(G)entre el punto final de e forma un ciclo simple, tal que cada ciclo es llamado como ciclo fundamental (o básico) de G con respecto a T_G. Cada arista de retroceso envuelve una única ruta contenida en un ciclo fundamental.

Considerando C={C_1,…,C_k} el conjunto de ciclos fundamentales encontrados durante una búsqueda en profundidad (DFS) en G. Dada cualquier par de ciclos fundamentales C_i C_j de C, si C_i y C_j comparten aristas son llamadas ciclos intersectados; de otra manera son llamados ciclos independientes.

Sea T_G el grafo construido a partir de una búsqueda en profundidad sobre un grafo G. T_G es conectado y es asumido que tiene ciclos intersectados. El método para contar coberturas de aristas en grafos sin ciclos intersectados es bien conocido de poder resolverse en tiempo polinomial.

Una regla es introducida para reducir el problema del cálculo del número de coberturas de aristas en un grafo que no tenga ciclos intersectados.

Definición 1 (Regla de División): Sea G=(V,E) un grafo que tiene ciclos intersectados. Sea e={u,v} E escogida de la siguiente manera:

e está involucrado en un número máximo de ciclos intersectados en H.

$\delta(u)\geq 3$ o $\delta(v)\geq 3$

La regla de división consiste en generar dos nuevos grafos de H lo cuales son llamados H_1 y H_2 de la siguiente manera:

Sea H_1=(V_1,E_1) donde V_1=V y E_1=E-e.

Sea H_2=(V_2,E_2) donde V_2 es obtenido de la siguiente manera: agregar cada nodo contenido en V excepto u y v. Adicionalmente, por cada arista incidente a {u,v} (escrito como {x,u} o {v,x}) agregar dos nuevos nodos (Mencionamos u_(x_1) y u_(x_2) para {x,u} o v_(x_1) y v_(x_2) para {v,x}). E_2 es obtenido de la siguiente manera: por cada arista {x,u} o {v,x} en E(H) con x≠u y x≠v, agregar dos aristas ({x,u_(x_1) } y {u_(x_1),u_(x_2)}) o ({x,v_(x_1) } y {v_(x_1),v_(x_2) }) respectivamente. Adicionalmente, agrega E(H)\{e U {x,u} U {v,x}}.

Observación 1: Siendo e parte de al menos un par de ciclos intersectados, entonces H_1. es todavía un grafo conectado. |V_1 |=n_1=n,|E_1 |=m_1=m-1. El número de ciclos base en H_1 es nc_1= m_1- n_1=m-n-1=nc-1. Entonces, H_1 contiene al menos un par menos de ciclos intersectados que H.

Observación 2: Sea n_2.= |V_2 | y m_2= |E_2 |. Todos los ciclos de H que contienen a e no son ciclos en H_2, por lo que podemos considerar que m_2≤m-5 desde que δ(v)≥3, δ(v)≥2 y n_2=n-2 por lo tanto, nc_2= m_2-n_2≤m-n-3=nc-3. El número de ciclos básicos H_2 y H son representados por nc_2 y nc respectivamente.

Teorema 1: NE(H_1). es el número de coberturas de aristas donde e no aparece. NE(H_2) es el número de coberturas de aristas donde e aparece.

Comprobación: Desde que las aristas en H_1. con parte de G excepto e, la primera parte del teorema es obviamente cierta. Para la segunda parte tenemos que demostrar que:

|{EC | EC Cε(G),e EC}|=NE(H_2)

En una cobertura de aristas donde e=(u,v) aparece, la arista adyacente ha u y v puede o no puede aparecer desde que u y v ya han sido cubiertas. Así ((δ(u)-1@0)) es el número de coberturas de aristas donde e aparece pero no aparecen aristas adyacentes a e para cubrir a u. ((δ(u)-1@1)) es el número de coberturas de aristas donde e aparece y solo una arista adyacente a e aparece para cubrir a u. ((δ(u)-1@δ(u)-1)) es el número de coberturas de aristas donde e aparece y cada arista adyacente a e aparecen para cubrir a u. La suma de las combinaciones es el número de coberturas de aristas que cubren a u. Hay 2^(δ(u)-1)coberturas de aristas.

Un argumento similar aplica para v. Por otra parte sea l un vecino de u, l ≠v, llegamos a la conclusión que si contamos separadamente la cobertura de aristas a través de los vecinos de e que cubren a u y multiplicamos, obtenemos 2^(δ(u)-1)coberturas de aristas. Estas son dos coberturas de aristas por cada vecino que cubre a u, {e} y {e,l}. Puesto que hay δ(u)-1 vecinos diferentes que u, contando separadamente hay 2^(δ(u)-1)coberturas de aristas que establecen la petición.

Corolario 1: NE(G).=NE(H_1)+ NE(H_2)

Comprobación: Como consecuencia del algoritmo anterior.

El siguiente algoritmo calcula NE(G) construyendo un árbol de enumeración sobre T_G, denotado como A(T_G).Cada nodo de A(T_G) tiene un grafo asociado. El nodo raíz de A(T_G) contiene el grafo de entrada inicial. Los nodos internos de A(T_G) tiene grafos asociados que no corresponden a los grafos básicos mientras los nodos hoja de A(T_G) solo corresponden a grafos básicos.

Por cada nodo interno de A(T_G) con un grafo H asociado, una arista especial e E(H) es escogida en orden de dividir el problema de calcular NE(H). La arista e debería ser parte de un ciclo intersectado entonces la regla de división es aplicado sobre e, contando separadamente el número de coberturas de aristas que contiene a e y aquellas que no.

Algoritmo 1: NE_General(T_G). Procedimiento para descomponer un grafo G que tenga ciclos intersectados.

1: Entrada: H=(V,E): el grafo producido por el algoritmo de búsqueda en profundidad, n= |V(H)|, m= |E(H)| y nc=m-n-1.

2: Salida: NE(G): El número de coberturas de aristas de G.

3: Seleccionar una arista e={u,v} E(H) como en la definición 3. {Notar que la arista e puede ser encontrada en O(nm log m)}

4: Aplicando la regla de división sobre e generando H_1 y H_2.

5: Si H_1 tiene ciclos intersectados entonces

6: NE_General(H_1)

7: Fin si

8: Si H_2 tiene ciclos intersectados entonces

9: NE_General(H_2)

10: Fin si

11: NE(G)= ∑_(G^′ H(A_G)) NE(G^′) donde H(A_G)={ G_h:G_h es el grafo asociado al nodo hoja A(T_G)}.

La exactitud del algoritmo NE_General parte del siguiente teorema.

Teorema 2: El resultado de la recursividad aplicando la regla de división en un grafo G calcula NE(G).

Demostración: Por inducción sobre el numero n de ciclos intersectados en el grafo G.

Ejemplo 1: Sea G=(V,E) el grafo que se muestra en la parte superior de la Figura 1, G tiene ciclos intersectados. Aplicando el procedimiento NE_General(G), obtenemos un árbol enumerativo A(G). La regla de división es aplicada tres veces sobre las aristas resaltadas en negro en los nodos interiores del árbol. Los nodos hojas del árbol son grafos cíclicos no interceptados, entonces su número de coberturas de aristas puede ser calculado con procedimientos en tiempo polinomial. La suma de las coberturas de aristas de cada hoja de los arboles da el número de coberturas para el grafo G. Si el número de hojas de A(G) consiste de grafos cíclicos desconectados no intersectados, el número de coberturas de aristas para la hoja es calculado por el producto de las coberturas de aristas de cada grafo desconectado (ver el hijo derecho de A(G)).

4 Tiempo de Complejidad del Algoritmo

Sea G= (V,E) el grafo de salida del algoritmo N E_ General, m= |E|, n= |V|.Puede ser considerado que el número máximo de ciclos intersectados en G no es mayor que nc=m-n porque a lo más un ciclo no es un ciclo intersectado.

El paso 3 tiene un tiempo de complejidad O(nm log m) . El paso 4 tiene un tiempo de complejidad O(m+n). La aplicación recursiva del algoritmo (paso 6 y 9) genera un árbol enumerativo E_G. Esta reducción genera dos nuevos grafos H_1=(V_1,E_1) y H_2=(V_2,E_2) de G.

El comportamiento en el tiempo del algoritmo reside en los pasos 6 y 9 los cuales corresponden con el número de ciclos intersectados en el grafo asociado con cada nodo de E_G. Si nc denota el máximo número de ciclos intersectados en un grafo asociado con el nodo de E_G, entonces, el tiempo de complejidad del algoritmo puede ser expresado con la recurrencia:

T(nc)=T(nc-1)+T(nc-3).

Y cuya recurrencia termina cuando nc=1.

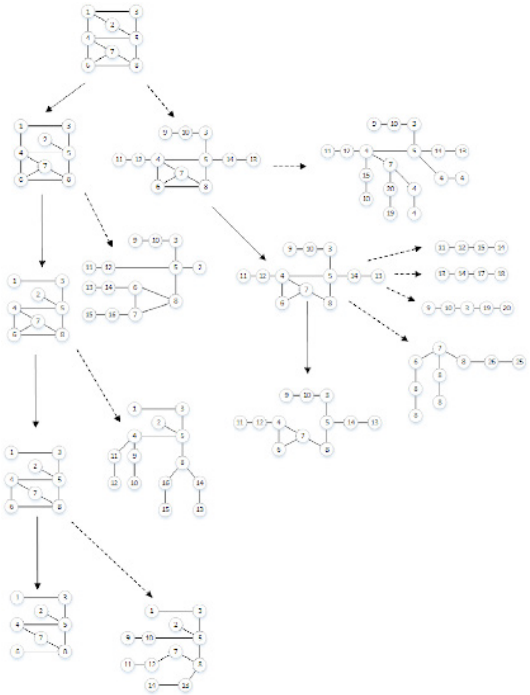

Fig. 1. Ejemplo del algoritmo sobre un grafo para generar las coberturas de aristas.

Esta recurrencia tiene el polinomio característico p(r)= r^3- r^2-1 la cual tiene la máxima raíz real r ≈1.46557. Esto nos lleva a que el límite superior del peor de los casos es O(r^((m-n))*(m+n))≈O(1.465571 ^((m-n))*(m+n)). Creemos que este es el primer límite superior no trivial obtenido por el problema #Edge_Covers.

5 Implementación

En esta sección se describe la implementación realizada del algoritmo utili-zando el lenguaje de programación C con algunas funciones de C++, debido a todas las ventajas que tienen estos lenguajes en cuanto a su rendimiento respecto a otros, lo cual es muy útil en este tipo de algoritmos que necesi-tan mucho tiempo de procesamiento.

5.1 Sistema de Archivos

Para crear el sistema de archivos se utiliza la función getcwd para obtener la ruta desde donde se ejecutó la aplicación, desde esta ruta se crea la carpeta Coberturas y dentro de esta se crean las carpetas IMÁGENES, DOT y Archi-vos usando una llamada al sistema con el comando mkdir, si estas no exis-ten ya.

Dentro de la carpeta Archivos se crea el archivo "Entrada", en este archivo se debe ingresar el grafo al que se le quiere aplicar el algoritmo, mientras que en las carpetas IMÁGENES y DOT se guarda de manera estructurada todos los arboles creados, de modo que a cada grafo le corresponda una carpeta, como se puede observar en la figura 2, en el ejemplo mostrado anteriormente se crean 15 carpetas por cada uno de los grafos creados.

Fig. 2. Estructura de directorios creada para guardar los grafos generados por el algoritmo

5.2Árbol de Expansión

Para almacenar cada grafo se utiliza la clase vector de C++, la cual al tener un tamaño dinámico, brinda la flexibilidad necesaria para poder aplicarse a este algoritmo, cada posición guarda una arista del grafo, se considera que el vértice de mayor grado sea la raíz del grafo ya que esto brinda la posibilidad de romper más ciclos de inicio.

Al momento de aplicar la búsqueda en profundidad, se crea el árbol de ex-pansión, inmediatamente después se obtiene el complemento el cual se almacena en un vector diferente y se manejan tanto el grafo original como su árbol de expansión y su complemento en una sola estructura denomina-da Conjunto, para facilitar el manejo de los tres grafos.

5.3 Split

Como se mencionó anteriormente cada grafo se guarda en un vector de aristas, creando un vector de ciclos que indique en cuantos ciclos participa cada arista, se escoge la que participe más y sobre esa se aplica la regla de división en el grafo original, el grafo resultado de la operación se almacena en un solo vector, sin embargo este se analiza posteriormente para identifi-car si se ha creado un bosque.

Si se creó un bosque, entonces cada árbol es almacenado en un vector dife-rente y estos se agrupan en un vector de vectores para manejar todos los arboles del bosque a la vez. El grafo o los grafos en caso de crearse un árbol se analizan de acuerdo al algoritmo individualmente originando las llamadas recursivas del algoritmo.

5.4 Expansión

Al realizar la operación de expansión al vector se le quita la arista seleccio-nada a través del vector de ciclos simplemente removiendo ese elemento del vector de aristas que representa al grafo.

6 Resultados

En la Tabla 1 se muestran los resultados obtenidos a partir de aplicar el algo-ritmo, se muestra el tiempo de ejecución del algoritmo generando la salida y sin almacenar la salida, para cada grafo se le realizaron 5 pruebas y se muestra el promedio obtenido, así como el número de nodos hojas obteni-do para cada uno de ellos, también se muestra el peso de los archivos crea-dos.

Table 1. Resultados Obtenidos en Grafos Completos Generando la Salida y sin Almacenar la Salida.

Tipo de Grafo	*Tiempo de Ejecución Generando la Salida*	*Tiempo de Ejecución Sin Almacenar la Salida*	*Nodos Hojas*	*Tamaño de los Archivos*
4 x 4 Completo	*189.1938 ms*	*44.113 ms*	*3*	*Total 104 K*
5 x 5 Completo	*189.1938 ms*	*51.11 ms*	*5*	*Total 268 K*
6 x 6 Completo	*2.813 s*	*77.244 ms*	*43*	*Total 1.6 M*
7 x 7 Completo	*10.770 s*	*138.0174 ms*	*173*	*Total 5.7 M*
8 x 8 Completo	*49.341 s*	*449.6138 ms*	*755*	*Total 24 M*
9 x 9 Completo	*3.054 min*	*1.918 s*	*3112*	*Total 95 M*
10 x 10 Completo	*13.974 min*	*9.253 s*	*13002*	*Total 394 M*
11 x 11 Completo	*50.872 min*	*47.656 s*	*55578*	*Total 1.6 G*
12 x 12 Completo	*3.921 horas*	*5.382 min*	*237757*	*Total 6.9 G*
13 x 13 Completo	*/*	*23.16 min*	*1062532*	*/*
14 x 14 Completo	*/*	*1.893 horas*	*4685143*	*/*

La Figura 3.a ilustra cómo incrementa el tiempo de ejecución en relación al número de vértices, sin almacenar la salida, este crecimiento corresponde aproximadamente a la función t=1.64632*exp((-n)/(-0.63224))+0.98744. En la Figura 3.b se ilustra cómo incrementa el tiempo de ejecución, generando la salida del grafo, este crecimiento corresponde aproximadamente a la función t=1.75308*exp((-n)/(-0.65927))+22.96526. Donde t es el tiempo de ejecución y n es el número de vértices.

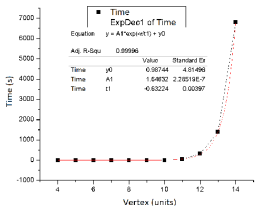

Fig. 3.a Tiempo de Ejecución en Grafos Completos sin Almacenar la Salida.---

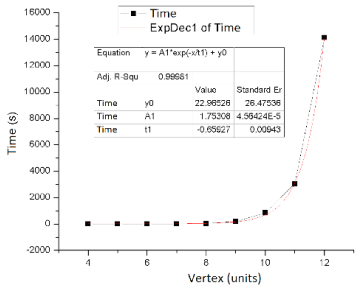

Fig. 3. b Tiempo de Ejecución en Grafos Completos Generando la Salida.

En la Tabla 2 se muestran los resultados obtenidos a partir de aplicar el algo-ritmo, en grafos no completos variando el número de aristas y vértices. Los resultados varían ya que no solo afecta el número de nodos y aristas, sino que también afecta el cómo están distribuidos los ciclos dentro del grafo.

Table 2. Resultados Obtenidos en Grafos No Completos.

No. de Vértices	*No. de Aristas*	*Tiempo de Ejecución*	*Nodos Hojas*
19	*40*	*1.86 min*	*1192*
28	*50*	*51.1 min*	*22357*
45	*55*	*9 s*	*50*
50	*51*	*18 s*	*61*
50	*66*	*1 min*	*308*
50	*71*	*4.1 min*	*920*
50	*77*	*13.35 min*	*4249*
100	*121*	*6.53 min*	*920*
500	*521*	*15.53 min*	*920*

7 Conclusiones

De acuerdo a los resultados obtenidos de las pruebas realizadas a la imple-mentación del algoritmo propuesto en [14], se puede observar que los va-lores de las funciones deducidas son 1.75308 y 1.64632 y se aproximan al valor estimado de 1.465571, además de que permite obtener de manera eficiente grafos simples sin ciclos a los cuales se les puede aplicar un proce-dimiento de tiempo de ejecución polinomial, pare encontrar el total de co-berturas de aristas del grafo.

El funcionamiento del algoritmo en grafos completos se mantiene constan-te, y su tiempo de ejecución aumenta de la manera esperada. El problema se presenta en grafos no completos ya que varía el tiempo dependiendo tanto del número de aristas y vértices, como de la distribución de los ciclos dentro del grafo entre más se centralizan en una determinada sección del grafo, el rendimiento disminuye.

Incluso el mismo número de ciclos intersectados, pero con un número ma-yor de vértices aumenta cada vez más el tiempo de ejecución, por lo que no es eficiente cuando los ciclos están centralizados en una determinada parte del grafo. Un campo de estudio para futuras investigaciones es tratar de encontrar una mejor forma de procesar este tipo de cúmulos de ciclos den-tro del grafo.

REFERENCIAS

1. Jiang, Y., Xia, Z., & Zhong, Y. (2014). Autonomous trust construction in multi-agent systems—a graph theory methodology. Journal of Advances in Engineering Software 36 (2), 59 -66.
2. 2. Núñez, J., Silvero, M., & Villar, M. (2012). Mathematical tools for the future: Graph Theory and graphicable algebras. Journal of Applied Mathematics and Computation 219 (11), 6113 - 6125.
3. 3. González-Díaz, H., Pérez-Montoto, L., & Paniagua, E. (2009). Generalize dlattice graphs for 2D-visualization of biological information. Journal of Theoretical Biology 261 (1), 136 - 147.
4. 4. Thomas, C., Lambrechts, J., Wolanskic, E., & Deleersnijdera, E. (2013). Numerical modelling and graph theory tools to study ecologicalconnectivity in the Great Barrier Reef. Journal of Ecological Modelling 272, 160 -174.
5. 5. Wanga, Y.-c., & Önal, H. (2011). Designing connected nature reserve networks using a graph theory approach. Journal of Acta Ecologica Sinica 31 (5), 235 -240.
6. 6. D. Phillips, J., Schwanghart, W., & Heckmann, T. (2015). Graph theory in the geosciences. Journal of Earth-Science Reviews 143, 147-160.
7. 7. Heckmann, T., Schwanghart, W., & D. Phillips, J. (2014). Graph theory—Recent developments of its application in geomorphology. Journal of Geomorphology.
8. 8. Abdul Rahman, A., Hamdan, M., & Ibrahim, M. (2014). The use of graphs in Malaysian companies' corporate reports: A longitudinal study. Journal of Procedia - Social and Behavioral Sciences 164, 653 - 666.
9. 9. Schroeder Barwaldt, M., de Fátima Franzin, R., & Vanderlei dos Santos, A. (2014). Using the theory of graphs on the implementation of bike lane in small towns. Journal of Procedia - Social and Behavioral Sciences 162, 350 - 358.
10. 10. A. Best, L., D. Smith, L., & Stubbs, D. (2001). Graph use in psychology and other sciences. Journal of Behavioural Processes 54 (1-3), 155-165.
11. 11. Goldenberg, D., & Galván, A. (2015). The use of functional and effective connectivity techniques to understand the developing brain. Journal of Developmental Cognitive Neuroscience 12, 155 -164.
12. 12. J. Phillips, D., McGlaughlin, A., & Soldan, A. (2015). Graph theoretic analysis of structural cconnectivity across the spectrum of Alzheimer3s disease: The importance of graph creation methods. Journal of NeuroImage: Clinical 7, 377 - 390.
13. 13. Lin, C., Liu, J., & Lu, P. (2014). A Simple FPTAS for Counting Edge Covers. Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms, 341-348.
14. 14. Hernández Servín, J., Marcial Romero , J., & De Ita, G. (2014). Low exponential algorithm for counting the number of edge cover on simple graphs. Proceedings of the Ninth Latin American Workshop on Logic/ Languages, Algorithms and New Methods of Reasoning. pp 1--8.
15. 15. Bondy, J., & Murty, U. (2010). Graph Theory. Springer Verlag, Graduate Texts in Mathematics.
16.

Acerca de los autores

Luis Ernesto Sierra-Alva, Egresado de la Facultad de Ingeniería de la Universidad Autónoma del Estado de México en la Carrera de Ingeniería en Computación. Participe como ponente en el congreso COMIA 2015 con el tema "Implementación de un método para generar coberturas de aristas en grafos simples":.

José Raymundo Marcial Romero, Actualmente profesor investigador de tiempo completo en la Facultad de Ingeniería de la Universidad Autónoma del Estado de México. Miembro del Sistema Nacional de Investigadores del Consejo Nacional de Ciencia y Tecnología, Nivel 1. En el año 2000 obtuvo el título de Licenciatura en Ciencias de la Computación y en el año 2007 el grado de Doctor en Ciencias de la Computación por The University of Birmingham, UK, en The School of Compu-ter Science.

Guillermo De Ita, Obtuvo su doctorado en Ingeniería Eléctrica en el CINVESTAV del IPN, México. Trabajó por 10 años como desarrollador y consultor en sistemas de bases de datos y sistemas de información geográfica para diferentes empresas en México. Ha realizado estancias de investiga-ción en la Universidad de Chicago, Texas A&M, INAOEP Puebla, UAEMEX-Toluca y en el insti-tuto INRIA en Lille Francia. Ha asesorado más de 40 trabajos de tesis (ya concluidos) en el área de computación a nivel licenciatura y posgrado y ha publicado más 100 artículos de investigación con arbitraje riguroso. Actualmente es profesor investigador de la Facultad de Ciencias de la Computación, BUAP, Puebla, México y Miembro del SNI Nivel 1.

José Antonio Hernández Servín, Actualmente profesor investigador de tiempo completo en la Facultad de Ingeniería de la Universidad Autónoma del Estado de México. Miembro del Sistema Nacional de Investigadores del Consejo Nacional de Ciencia y Tecnología, Nivel 1. En 1999 obtu-vo el título de Licenciado en Ciencias Fisico-Matématicas y terminó la Maestría en Matemáticas Puras en 2001 en la UMSNH (Univ. Aut. San Nicolás de Hidalgo). Para el año 2005, obtiene el Doctorado en Ciencias (PhD) por The University of Nottingham, UK, en The School of Electrical and Electronic Engineering, en el departamento de Óptica.