Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Tesis de Licenciatura

Nueva heurística para el Problema de Recubrimiento de Grafos por Ciclos Acotados

Ernesto Adrián Álvarez Capandeguy
María Fernanda Managó

Directora: Dra. Irene Loiseau

1 Resumen

El despliegue de enlaces de alta capacidad en redes de telecomunicaciones trajo como consecuencia la necesidad de que grandes redes deban operar sin ninguna falla. Debido al uso de esta clase de enlaces muy pocos transportan la información de gran cantidad de canales, que tradicionalmente dependían de muchos enlaces de menor capacidad. La posibilidad de que un número pequeño de fallas pudiera causar la caída de una red hizo que la creación de enlaces tolerantes a fallos se volviera un tema de mucha importancia. Una técnica para proveer este tipo de robustez a la red es formarla usando Self Healing Rings (SHR).

En este trabajo proponemos una heurística para resolver el Bounded Cycle Cover Problem (BCCP), el cual consiste en encontrar un cubrimiento de un grafo 2-conexo con ciclos de costo mínimo y una cantidad acotada de ejes, que puede aplicarse a la creación de SHR tolerantes a fallos. La solución que proponemos es una variante de GRASP, cuya principal diferencia está en el mecanismo de selección de los ciclos candidatos. Mientras que GRASP lista los vecinos más ventajosos según la función greedy, y luego elige uno al azar, nuestra heurística selecciona vecinos al azar y se queda con el más ventajoso de ellos.

Nuestro algoritmo, comparado con otras heurísticas que aproximan soluciones para el mismo problema consigue soluciones de mejor calidad, en algunos casos a costa de un mayor tiempo de cómputo.

Abstract

An important consequence of the development of modern communications is a need for highly reliable networks that can operate for extended periods of time without any faults. Due to the deployment of high speed links most traffic that would have traditionally been sent through multiple low capacity links is now being sent on a few high capacity fiber optic channels, creating few single points of failure. Due to the possibility of massive disruption due to the failure of one of those links, research into fault tolerant networks and failover techniques became a priority. One of these techniques is to provide resilence to a network is using Self Healing Rings (SHR).

In this document we propose a heuristic approach to solve the Bounded Cycle Cover Problem (BCCP), finding a minimum cost cycle cover for a 2-connected graph, with a limited number of hops per cycle, which can be used in the construction of fault tolerant networks composed of SHRs. Our proposed solution is a variant of GRASP, the main difference being the cycle selection mechanism: while GRASP obtains a list of possible neighbors based on a greedy funtions and chooses one at random, our heuristic selects a few neighbours at random and selects the best one according to the greedy function.

Our algorithm, when compared with other heuristics used for obtaining solutions for the same problem, yields better quality solutions, at a higher computational cost.

Índice

2 Introducción

El despliegue de enlaces de alta capacidad en redes de telecomunicaciones trajo como consecuencia la necesidad de que grandes redes deban operar sin ninguna falla. Debido al uso de enlaces de alta capacidad muy pocos de estos transportan la información de gran cantidad de canales, que tradicionalmente dependían de muchos enlaces de menor capacidad. La posibilidad de que un número pequeño de fallas pudiera causar la caída de una red, especialmente considerando las exigencias de calidad a la que están sujetas las compañias de telefonía, hizo que desde finales de los años 80 aumentara la preocupación por asegurar que los enlaces de fibra óptica que conectan los concentradores de telecomunicaciones sean tolerantes a fallos, convirtiéndose en un área de alta prioridad. [14]

Una posible solución al problema de fallas es la utilización de mecanismos de hardware encargados de derivar el tráfico por un enlace alternativo, en caso de detectarse una falla en el enlace principal. Self-healing rings (SHR) es una técnica que provee tolerancia a fallos a través de estos mecanismos.

La idea detrás de un SHR consiste en conectar las estaciones que se desea enlazar en forma de anillo bidireccional, lo que provee dos caminos posibles entre estaciones, en combinación con hardware y señalización capaz de detectar la caída de un enlace y derivar el tráfico en el sentido contrario al punto de fallo. Es importante aclarar que aunque conceptualmente se trate de un anillo bidireccional, está formado por dos anillos unidireccionales. Teniendo en cuenta esto, de aquí en adelante hablaremos de anillos bidireccionales, no preocupándonos más de este detalle de implementación. Esta técnica permite que las estaciones continúen comunicadas aún cuando hubiera un fallo que sacara de operación a una de ellas, o un corte en las fibras que unen dos estaciones.

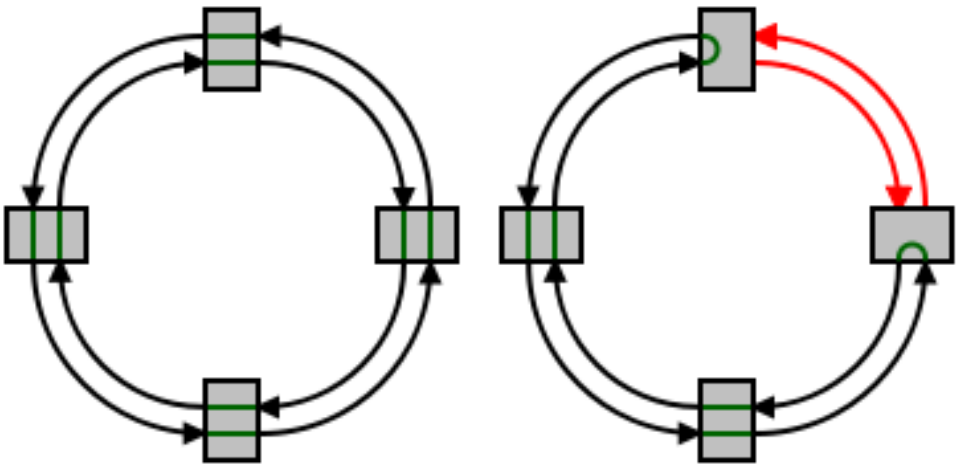


Figura 1: Un self healing ring y su forma de operar. En el anillo de la izquierda, el SHR está operando normalmente, mientras que en el SHR de la derecha un problema en la red causa que el tráfico sea desviado por un camino alternativo.

Ante una falla, el desvío de tráfico se realiza en los nodos próximos al punto de corte, ya que estos son los que están en condiciones de detectarlo directamente, y en

consecuencia pueden reaccionar inmediatamente sin la intervención de los otros nodos del anillo. Esto es importante, ya que es deseable minimizar el tiempo en el cual el anillo no está operando correctamente, así como la pérdida de datos. Si en lugar de ser los nodos próximos a la falla los que tomen la medida correctiva, el nodo de origen fuera quien despachara el tráfico evitando la falla, sería necesario un mecanismo más complejo de coordinación, y la pérdida de datos sería mayor, puesto que dependería de que cada estación de origen altere su comportamiento, en lugar de contener el problema en el punto de falla.

Aunque la idea de SHR ya había sido aplicada en redes de corto alcance y baja velocidad, mejoras en la tecnología hicieron posible la creación de SHR como medio para agregar tolerancia a fallos en enlaces de mediano alcance y de alta velocidad. Esto permitió la creación de redes entre concentradores de telecomunicaciones con alta tolerancia a fallos y bajo tiempo de reacción, aunque las distancias que estas redes pueden cubrir son acotadas.

Recientemente, el uso de redes de computadoras para aplicaciones de tiempo real y tareas críticas se incrementó notablemente. Esto impulsó un aumento en la exigencia de los niveles de calidad de servicio, ya no siendo suficiente la idea de “best effort”, que venía siendo el paradigma más usado, en ciertos casos requiriendo niveles de servicio similares a los de las redes de telecomunicaciones clásicas.

Estos cambios llevaron a la necesidad de mejorar el ruteo y el tiempo de respuesta de las redes de computadoras. A pesar de que las técnicas convencionales de ruteo son adecuadas, su tiempo de reacción ante cambios no siempre lo es, pudiendo tardar hasta varios minutos en registrar una modificación de topología, que podría haber sido provocada por la caída de un enlace y que causaría una pérdida de comunicación durante ese tiempo.

El uso de SHR puede reducir el tiempo de reacción de la red ante una caída a meros milisegundos y permitir que la reparación ocurra localmente y de forma automática, lo que evita la intervención de un algoritmo de ruteo y la pérdida de tiempo que esto implica, por lo que esta técnica tambíen se aplica a redes de computadoras. Un ejemplo de este tipo de redes es FDDI [12], la cual usa un anillo doble para proveer redundancia [10]. A pesar de esta tolerancia a fallos, FDDI es considerado obsoleto desde la introducción de Fast Ethernet y Gigabit Ethernet debido a razones de costo y performance.

Para poder usar SHR en una red, puede ser necesario realizar actualizaciones sobre sus enlaces, lo que conlleva costos que pueden incluir el tendido de fibra óptica y la compra de equipo adicional. Para minimizar estos costos se puede utilizar el Problema de Recubrimiento de Grafos por Ciclos Acotados, también conocido como Bounded Cycle Cover Problem (BCCP). BCCP consiste en encontrar un cubrimiento de costo mínimo, de un grafo 2-conexo con ciclos cuya cantidad de ejes está acotada.

Es importante notar que BCCP es un problema NP-Hard [13], por lo que puede ser demasiado costoso obtener una solución exacta del problema. Para obtener una solución aproximada en un tiempo razonable se pueden emplear heurísticas. En este documento describiremos una heurística similar a GRASP llamada HBCCP.

En la sección 3 presentaremos una descripción del problema y detallaremos cómo se relaciona con BCCP.

En la sección 4 mostraremos un resumen de otras técnicas utilizadas para la resolución de este problema.

En la sección 5 hablaremos de la heurística HBCCP y analizaremos cómo la aplicación de la misma sirve para la resolución del problema. Adicionalmente en esta sección mostraremos cómo esta heurística se relaciona con GRASP.

En la sección 6 mostraremos los resultados obtenidos, junto con un análisis de los mismos, explicando los distintos casos resueltos, y las fortalezas y falencias de la heurística.

En la sección 7 presentaremos las conclusiones de nuestro trabajo, con recomendaciones para trabajos futuros en la sección 8.

3 Descripción del problema

3.1 Motivación

Dada una red con más de un camino entre cualquiera de sus nodos (una topología de red 2-conexa), es posible convertirla en un sistema de SHRs duplicando algunos de sus enlaces. Si bien esta mejora se puede realizar de forma desorganizada, es importante tener en cuenta que el tendido de una red es una operación costosa, por lo que resulta deseable minimizar el costo incurrido por agregar los enlaces necesarios para actualizar la red de forma tal que puedan usarse con esta técnica de hardware.

Este problema se representa directamente con BCCP, que consiste en cubrir un grafo con circuitos tales que ninguno de ellos tenga más que una cantidad preestablecida de ejes, minimizando el costo total del cubrimiento. En BCCP, el grafo representa a la red original, mientras que el agregado de los enlaces redundantes se lo puede representar con el cubrimiento de este grafo usando ciclos. La longitud máxima de los anillos se corresponde directamente con la cantidad máxima de ejes que pueden tener los ciclos del cubrimiento.

3.2 El Bounded Cycle Cover Problem (BCCP)

Sea G un grafo con n nodos y m ejes, es decir, G = (V, E) y n = |V|, m = |E|. Sea c_e el costo asignado a un eje. Definimos un camino simple $P_{i,j}$ como una secuencia de nodos y ejes adyacentes en el cual ningún nodo o eje aparece más de una vez, y que comienza en el nodo i y termina en el nodo j. Un ciclo es un camino $P_{i,i}$ de longitud mayor a 0. El costo c_P de un camino o ciclo P está definido como la suma de los costos de los ejes que lo componen, es decir, $c_P = \sum_{e \in P} c_e$. Se define el largo de un camino o ciclo, L_P, como la cantidad de ejes que que lo componen, es decir $L_P = \sum_{e \in P} 1$.

Un ciclo R cubre a un eje e si este pertenece a la secuencia de ejes de R. Se define cycle cover o cubrimiento con ciclos de un grafo G a un conjunto de ciclos que cubren a todos los ejes de G, no necesariamente una vez por eje. El costo del cubrimiento de un grafo se define como la suma de los costos de los ciclos que lo componen.

Un grafo es 2-conexo si entre cada par de nodos existen dos caminos disjuntos en nodos.

Un grafo es mixto si tiene algunos ejes orientados y otros no orientados.

Definimos como k-ciclo a un ciclo formado por a lo sumo k ejes.

El BCCP se define de la siguiente manera:

Dado un grafo 2-conexo G=(V,E), un costo c_e asociado a cada eje e ∈ E y un entero k, se quiere encontrar un cubrimiento de G con k-ciclos, tal que su costo sea mínimo.

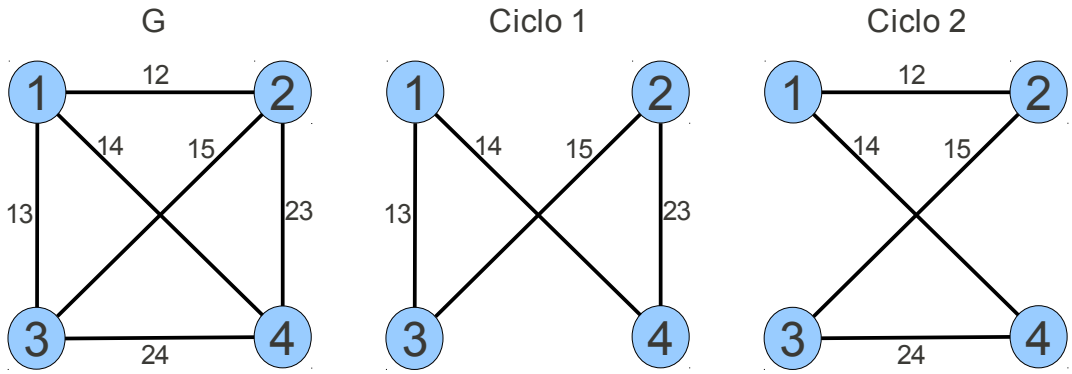


Figura 2: Ejemplo de BCCP con un largo máximo de ciclo de 4 ejes.

BCCP es NP-Hard debido a que existe un problema NP-Completo, llamado MCCP [13], que es una instancia del mismo [8].

Un problema que resulta útil para evaluar la calidad de un cubrimiento es el problema del Cartero Chino (CPP). El mismo consiste en encontrar el ciclo de costo mínimo en un grafo, que pasa por todos los ejes al menos una vez. Este problema tiene solución polinomial cuando se aplica a grafos orientados o no orientados, y NP-HARD para grafos mixtos [9].

CPP en apariencia no es muy similar a BCCP, pero es posible descomponer un recorrido de CPP en una serie de ciclos, algunos de ellos de longitud 2 (ciclos degenerados), que forman un cubrimiento del grafo de entrada. Las diferencias principales entre este cubrimiento y uno de BCCP es que el cubrimiento derivado de CPP admite ciclos de longitud mayor a K y admite estos ciclos degenerados, cosas que no están permitidas en un cubrimiento de BCCP.

El problema del Cartero Chino provee una cota inferior para BCCP[8], y dado que es un problema polinomial cuando se aplica a grafos no dirigidos, este valor se lo puede emplear como base para determinar la calidad de las soluciones aproximadas de BCCP.

Dada la complejidad de BCCP en este documento presentaremos una heurística para tratar de obtener una solución aproximada pero aceptable al problema.

4 Estado del Arte

A continuación mencionaremos trabajos de otros autores sobre BCCP.

Olinick y Hochbaum[8] propusieron un algoritmo para el BCCP, reformulando el problema como un cubrimiento de conjuntos (SCP), con dos formas de obtener el conjunto de ciclos candidatos a usar: uno basado en bases de ciclos, y otro modificando el algoritmo del cartero chino.

El primer algoritmo es una modificación del algoritmo de Horton [5] para obtener una base de ciclos, de costo mínimo, agregando ciclos adicionales para cubrir los ejes que la base no hubiera cubierto.

El segundo algoritmo se basa en una solución del problema del cartero chino, modificándola para eliminar ciclos cuya cantidad de ejes excedan el valor máximo permitido por el problema, y eliminando los ciclos degenerados (es decir los que parten del nodo i, pasan por un nodo vecino j y vuelven al nodo i sin pasar por otros nodos), reemplazándolos por ciclos adecuados al problema de BCCP.

Los ciclos obtenidos con los algoritmos anteriores son incluídos como columnas en un problema de programación lineal entera que provee entonces una solución aproximada.

Recientemente Guillermo Picardi propuso aplicar Tabú Search para la aproximación de BCCP. El algoritmo que presenta en su trabajo consta de dos etapas. En la primera genera un conjunto de ciclos de tamaño menor o igual al máximo preestablecido. Para la generación de los mismos usó dos métodos: en uno de ellos implementó una variante del algoritmo presentado en el trabajo de Liu H. y Wang J. [6], mientras que en el otro utilizó la técnica de Cycle Basis usada por Olinick y Hochbaum [8].

En la segunda fase aplica un algoritmo de Tabú Search con el objeto de dicidir qué ciclos seleccionar, de manera que estos cubran todos los ejes de la red original, minimizando la cantidad y/o los costos de los ciclos involucrados.

5 Heurística

En esta sección presentaremos una heurística con algunos componentes golosos y otros aleatorios para la obtención de soluciones aproximadas para BCCP. La misma se basa fuertemente en GRASP [3]. Llamaremos a esta heurística HBCCP.

HBCCP se basa en la obtención de una serie de cubrimientos válidos, es decir, que cumplan con las restricciones impuestas por BCCP, usando un algoritmo goloso con componentes aleatorios. A estas posibles soluciones se las intenta mejorar usando una búsqueda local. Luego, de este conjunto de cubrimientos se selecciona al de costo mínimo como solución. El pseudocódigo de este proceso se puede ver en la siguiente figura:

```
Funcion HBCCP
Entrada:
    Grafo 2-conexo G = ( V,E )
    Entero k
Salida:
    Conjunto de k-ciclos en G
Variables:
    Conjunto de k-ciclos Sf, Sg y Sl


1.    Sf ← ∅
2.    Mientras no lleguemos al número límite de iteraciones sin
      encontrar una mejor solución
3.          Sg ← GreedyRandom(G,k)
4.          Sl ← LocalSearch(Sg,G,k)
5.          Si CostoSolucion(Sl) < CostoSolucion(Sf)
6.                Sf ← Sl
7.          Fin Si
8.    Fin Mientras
9.    Return Sf
```

La función CostoSolucion() calcula la suma de los costos de los ciclos que componen una solución. En caso de que la misma sea vacía, la función devuelve infinito.

El método GreedyRandom consiste en buscar varios ciclos al azar, que son los candidatos a ser agregados a la solución. Cada uno de estos candidatos son evaluados por una función, que llamaremos Greedy, la cual le asigna un valor en base a la cantidad de ejes que cubre y a la suma de los costos de sus ejes, agregando a la solución el ciclo que sea más ventajoso, de acuerdo con esta función.

El algoritmo GreedyRandom se define de la siguiente forma:

```
Función: GreedyRandom
Entrada:
    Grafo 2-conexo G = (V,E)
    Entero k
Salida:
    Conjunto k-ciclos de G
Variables:
    Conjunto de k-ciclos S
    K-Ciclo Cf, Ci
Constantes:
    Entero T (cantidad de ciclos a generar)

1.    S ← ∅
2.    Mientras G no esté completamente cubierto por S
3.          Cf ← ∅
4.          Para i desde 1 hasta T
5.                Ci ← GenerarCicloAlAzar(G,k)
6.                Si Greedy(Ci,G) < Greedy(Cf,G)
7.                      Cf ← Ci
8.                Fin Si
9.          Fin Para
10.         S ← S ∪ { Cf }
11.   Fin Mientras
12.   Return S
```

La función Greedy se define de la siguiente manera:

Sea C un ciclo y G un grafo.

$$\text{Greedy(C,G)} = \begin{cases} \dfrac{Costo(C)}{CantidadEjesQueCubre(C,G)} & \text{si CantidadEjesQueCubre(C,G)} > 0 \\ \infty & \text{si CantidadEjesQueCubre(C,G)} = 0 \end{cases}$$

La función GenerarCicloAlAzar(G,k) construye un ciclo partiendo de un nodo al azar y moviendose aleatoriamente, saltando de vecino en vecino hasta que se forme un ciclo,

no necesariamente cerrándolo en el nodo inicial. Dado que el ciclo no puede tener una longitud mayor a k, la función toma en cuenta solamente los últimos k nodos visitados para la formación del mismo, requiriendo que este cierre ocurra contra uno de los k nodos mencionados. Para evitar que la función cicle indefinidamente, en caso de que no sea posible construir un ciclo porque las condiciones del problema hagan que no exista una solución, la ejecución de esta función se detiene una vez recorrida una cantidad determinada de ejes, dependiente del tamaño del grafo a recorrer.

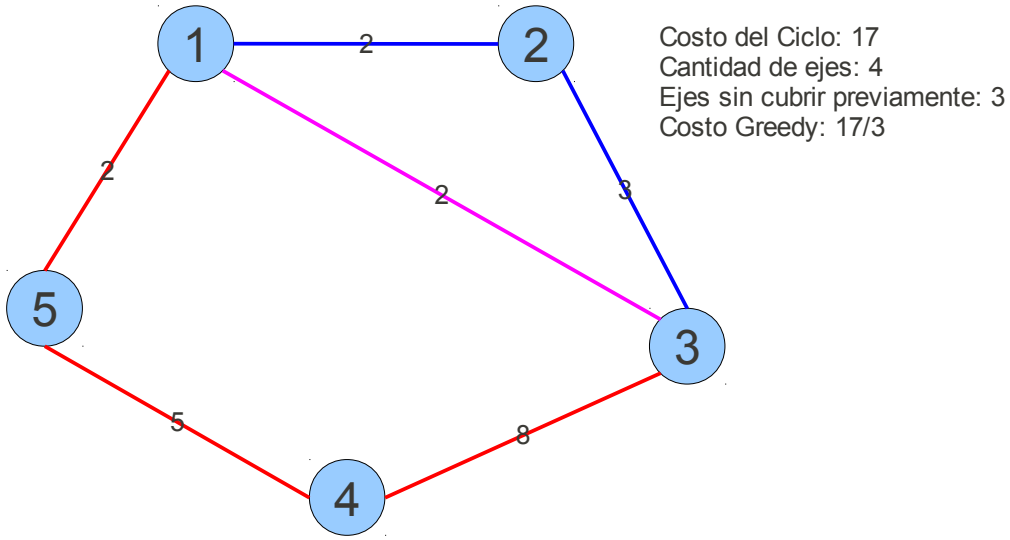


Figura 3: En este ejemplo se calcula el costo de terminar de cubrir un grafo con el ciclo 1-3-4-5. En rojo, los ejes pertenecientes exclusivamente a 1-3-4-5, en azul los ejes pertenecientes exclusivamente a 1-2-3, previamente agregado a la solución, en violeta los ejes comunes. El costo depende del costo total del ciclo y de la cantidad de ejes nuevos cubiertos.

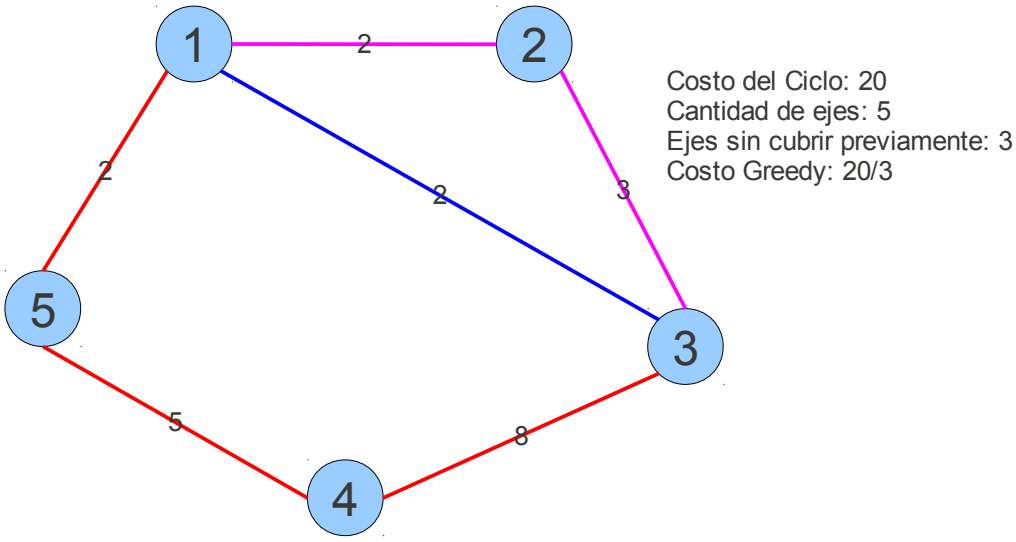


Figura 4: Una segunda posibilidad para el cubrimiento, usando el ciclo 1-2-3-4-5. A pesar de que el ciclo tiene una longitud de 5, solamente tres de estos ejes no están previamente cubiertos, por lo que el costo total se sigue dividiendo por 3. Debido a esto, el costo greedy de este candidato es mayor que el de la Figura 3. Si el ciclo 1-2-3 no formara parte de la solución al momento de incorporar a 1-2-3-4-5, el costo sería más bajo que el de la Figura 3.

Función: GenerarCicloAlAzar

Entrada:

Grafo 2-conexo G = (V,E)

Entero k

Salida:

K-Ciclo

Variables:

Cola de nodos C (de longitud máxima k, en caso de exceder, descarta los primeros elementos ingresados)

Nodo A, E, P, U

K-ciclo RV

Entero i

Constantes:

Entero N (cantidad de ejes a recorrer)

```
1.    A ← Nodo al azar de G
2.    Encolar A en C
3.    E ← Nodo al azar de G, vecino de A
4.    Encolar E en C
5.    Para i desde 2 a N
```

```
6.          P ← Un nodo vecino de E en G distinto de A, al azar
7.          Si hay un elemento U igual a P en la cola C
8.                Descolar y descartar los elementos de C
                  anteriores a U
9.                RV ← Un ciclo formado por {U,C[2],C[3],..,P}
10.               Return RV
11.         Else
12.               Encolar P en C
13.               A ← E
14.               E ← P
15.         Fin Si
16.   Fin Para
17.   Error (no se pudo encontrar un ciclo)
```

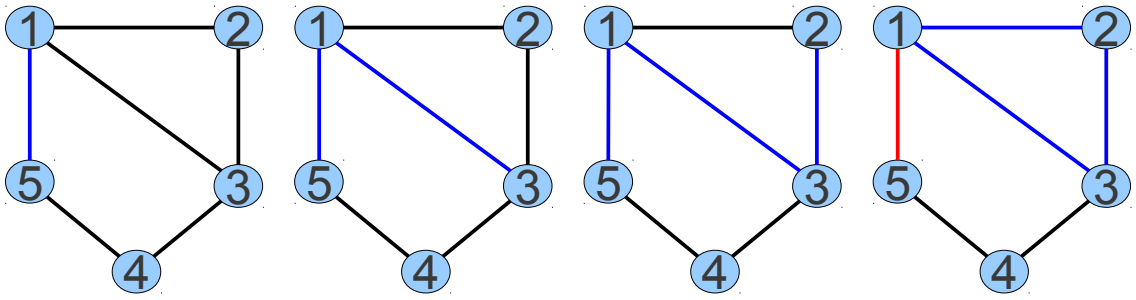


Figura 5: Una búsqueda de un ciclo candidato. En la primera imagen, se parte de un nodo al azar a un vecino. Las segunda y tercera imagenes muestran un recorrido parcial, y la cuarta muestra al ciclo formado. Notar que el mismo se cerró en un nodo distinto al origen, y que el primer eje encontrado (en rojo) no forma parte del ciclo.

Una vez obtenida una posible solución, intentamos mejorarla realizando un proceso de búsqueda local. El objetivo de este proceso es reducir el costo total de la solución. Para ello, sustituímos cada eje de un ciclo por un camino con mayor cantidad de ejes, pero de menor costo, siempre que el eje sea cubierto por más de un ciclo. Esta sustitución se realiza por medio del siguiente algoritmo:

```
Función: LocalSearch
Entrada:
    Grafo 2-conexo G = (V,E)
    Entero k
    Conjunto de k-ciclos Sg (formando un cubrimiento de ejes de G)
Salida:
    Conjunto k-ciclos de G (formando un cubrimiento de ejes de G)
```

```
Variables:
    Eje e
    Conjunto de tuplas de <k-ciclo, k-ciclo, entero> AT
    Camino P
    k-Ciclo C
    Entero H

1.    Para todos los ejes e de G, ordenados de mayor a menor costo
2.          Si e está cubierto por más de un ciclo de Sg
3.          AT ← ∅
4.                Para todo ciclo i ∈ Sg que cubre a e
5.                      P ← CaminoAlternativo(G,i,e,k)
6.                      C ← Sustituir(i,e,P)
7.                      H ← Costo(i) - Costo(C)
8.                      Agregar <i,C,H> a AT
9.                Fin Para
10.               Para todo elemento de AT, excepto el de menor H
11.                     Sg ← Sg \ i
12.                     Sg ← Sg ∪ { C }
13.               Fin para
14.         Fin Si
15.   Fin Para
16.   Return Sg
```

Las funciones Sustituir y Costo se definen de la siguiente forma:

- Sustituir(i,e,P) reemplaza en i a e por P.
- Costo(i) es la suma de los costos de los ejes que componen i.

Para asegurar que se mantenga el cubrimiento de todos los ejes, solo se pueden reemplazar aquellos que estén cubiertos por más de un ciclo, evitando que sea sustituído en al menos uno de ellos. Este es el motivo del condicional de la línea 2 de la función LocalSearch.

La decisión de cuál ciclo se debe dejar cubriendo al eje se basa en la ganancia que podría obtenerse si se realizara dicha sustitución. La ganancia, representada por la variable H, es la diferencia entre el costo del eje y el costo del camino alternativo. Ante la posibilidad de sustituir un eje en uno de dos ciclos posibles, siempre es más conveniente realizar la sustitución en el ciclo que daría mayor ganancia, puesto que en ese caso el costo total de la solución resultante será menor que si se sustituyera en el ciclo con menor ganancia. Esto explica porqué se elige al ciclo de menor ganancia como

el ciclo a dejar, ya que tomar esta decisión minimiza el costo resultante en este paso.

El algoritmo de la función CaminoAlternativo es el siguiente:

```
Función: CaminoAlternativo
Entrada:
    Grafo 2-conexo G = (Vg,Eg)
    k-ciclo i = (Vi,Ei)
    Eje e ∈ E / e=(n1,n2)
    Entero k
Salida:
    Camino en G
Variables:
    Entero k2
    Grafo Z = (Vz,Ez)
    Conjunto de nodos Vt
1.    Vt ← Vi \ {n1,n2}
2.    Ez ← Eg \ {todos los ejes adyacentes a los nodos de Vt}
3.    Vz ← Vg
4.    k2 ← k - CantidadEjes(i)
5.    Return DistanceVector(Z,k2,n1,n2)
```

Durante una sustitución, no todos los ejes del grafo pueden usarse para formar el camino alternativo. Este camino solamente puede estar formado por ejes que no sean adyacentes a nodos que pertenezcan al ciclo, siendo las únicas dos excepciones los nodos que forman el eje que se pretende sustituir, el cual será devuelto en caso de no encontrar otro camino mejor. La razón por la que no se pueden usar los ejes mencionados es que el resultado final de la sustitución debe ser un ciclo, y si se repitiera un eje, esta condición no se preservaría. En el caso particular de nuestra heurística tampoco permitimos el uso de ejes adyacentes a nodos pertenecientes al ciclo, puesto que se dejaría de obtener un ciclo, a menos que se lo dividiera en dos en el nodo común.

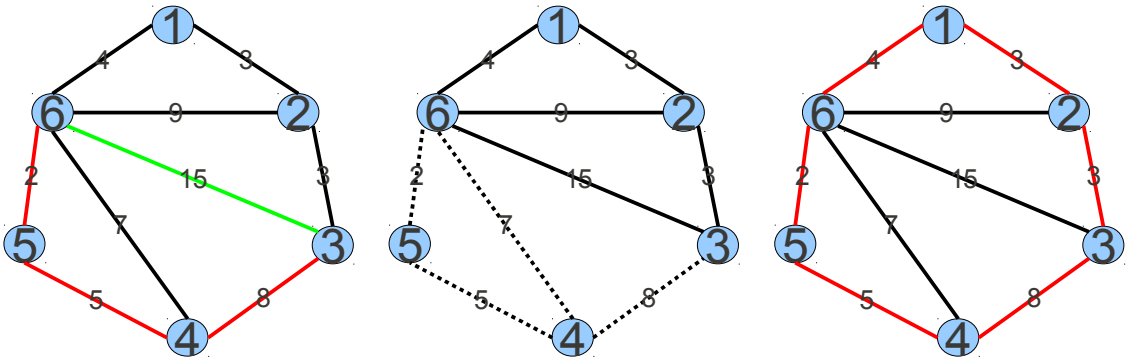


Figura 6: El grafo de la izquierda muestra el ciclo que se pretende mejorar, con el eje a sustituir en verde.El grafo central muestra con líneas continuas los ejes que pueden ser parte del camino alternativo, mientras que los ejes punteados no pueden ser elegidos. En el grafo de la derecha se muestra en rojo el nuevo ciclo optimizado.

Por último, la búsqueda local comienza desde los ejes de mayor a menor costo para limitar la cantidad de pasos a realizar. Cuando un eje es sustituído por un camino de costo menor pero con mayor cantidad de saltos, los ejes que componen este camino deben, por definición, tener un costo menor. Al realizar la sustitución partiendo desde el eje más costoso hasta el eje de menor costo, podemos garantizar que en una iteración todos los posibles descensos serán tomados, sin necesidad de realizar una segunda pasada. Esta propiedad nos permite asegurar que la búsqueda local termina después de calcular a lo sumo $c \times m$ caminos alternativos, en donde c es la cantidad de ciclos de la solución obtenida de la fase greedy y m es la cantidad total de ejes del grafo.

Para la obtención de los caminos alternativos es necesario contar con un algoritmo de cálculo de camino mínimo que adicionalmente permita calcular resultados que respeten las condiciones impuestas por el problema. Aunque la mayoría de las condiciones están aseguradas por las manipulaciones que le realizamos al grafo de entrada, el algoritmo de camino mínimo debe poder limitar la cantidad de saltos del camino alternativo para respetar la condición de la longitud máxima de los ciclos que forman el cubrimiento. Esta cota no solo depende de los parámetros del problema sino también de la longitud del ciclo que está siendo evaluado.

Una variante del algoritmo de camino mínimo de Bellman-Ford, usado en el cálculo de ruteo en redes, posee las características necesarias. Esta variante, utilizada en los protocolos conocidos como «Distance Vector», consiste en el algoritmo de Bellman-Ford, pero usando como posibles nodos intermedios a los vecinos directos del nodo de origen. La descripción de esta variante puede encotrarse en [4].

```
Función: DistanceVector
Entrada:
    Grafo G = (V,E)
    Entero k
    Nodo O ∈ V
    Nodo D ∈ V
```

```
   Salida:
       Camino en G
   Constantes:
       Entero N = Cantidad de nodos de G
   Variables:
       Matriz de nodos de [N x N] SS
       Matriz de enteros de [N x N] C, PC
       Nodo n1,n2
       Camino R
1.     Para todo nodo n1 de G
2.           Para todo nodo n2 de G
3.                 Si n2 es vecino de n1
4.                       C[n1][n2] ← CostoEje(n1,n2)
5.                       SS[n1][n2] ← n2
6.                 Else
7.                       C[n1][n2] ← ∞
8.                       SS[n1][n2] ← null
9.                 Fin Si
10.          Fin Para
11.    Fin Para
12.    PC ← C
13.    Para i desde 2 a k
14.          Para todo nodo n1 de G
15.                Para todo nodo n2 de G / n2 ≠ n1
16.                      Para todo nodo n3 de G / n3 ≠ n2 ∧ n3 ≠ n1
                         ∧ n1 es vecino de n3
17.                            Si C[n1][n2] > C[n3][n2] +
                               CostoEje(n1,n3)
18.                                  PC[n1][n2] ← C[n3][n2] +
                                     CostoEje(n1,n3)
19.                                  SS[n1][n2] ← n3
20.                            Fin Si
21.                      Fin Para
22.                Fin Para
23.          Fin Para
24.          C ← PC
25.    Fin Para
```

```
26.   R ← ∅
27.   NA ← n1
28.   Mientra NA ≠ n2
29.         Agregar NA a R
30.         NA ← SS[NA][n2]
31.   Fin Mientras
32.   Return R
```

Debido a que Distance Vector evalúa los nuevos caminos fijándose en los vecinos inmediatos de cada nodo, la longitud de un posible camino está acotada por la cantidad de iteraciones que transcurrieron, es decir que en la i-ésima iteración el camino tendrá a lo sumo i cantidad de saltos. Esta propiedad nos permite limitar el largo del camino, limitando la cantidad de iteraciones que se le permite correr al algoritmo. Gracias a esto, y conociendo el largo máximo permitido para un ciclo y la longitud del ciclo que está siendo mejorado, es posible obtener caminos alternativos que cumplan con las condiciones del problema.

Nuestra heurística, al igual que GRASP, se basa en obtener una gran cantidad de soluciones, todas con pequeñas diferencias en la forma de generarlas, y quedándonos con la mejor de todas ellas.

La principal diferencia con GRASP está en el mecanismo de selección de los ciclos candidatos. Mientras que GRASP lista los vecinos más ventajosos según la función greedy, y luego elige uno al azar, nuestra heurística selecciona vecinos al azar y se queda con el más ventajoso de ellos. La idea detrás de este mecanismo de selección es que una búsqueda al azar no cubre toda la vecindad, por lo que se introducirán variaciones en el mecanismo de formación de la solución, que lo diferenciarán de una solución completamente greedy.

Una limitación importante que nos impidió hacer GRASP directamente es la imposibilidad de enumerar nuestra vecindad, es decir, todos los ciclos del grafo, ya que la cantidad de estos es exponencial respecto a los ejes y nodos, por lo que el cálculo de esta vecindad no sería realizable en un tiempo razonable.

6 Resultados e implementación

6.1 Introducción

En esta sección presentaremos los resultados de ejecutar nuestro programa sobre diversos grafos y analizaremos su comportamiento.

Con el fin de realizar una comparación entre nuestros resultados y los de [8], utilizamos la serie empleada en el trabajo de dichos autores. La misma fue generada con la rutina plane_miles de Stanford GraphBase, que crea grafos planares y euclideanos seleccionando ciudades al azar de la guía Mc Nally & Company's Standard Highway Mileage. Los grafos de esta serie tienen 10, 15, 20, 25, 30 y 35 nodos, en donde el costo de los ejes representa la distancia en millas entre dichas ciudades.

En primera instancia, fue necesario ajustar los parámetros del algoritmo para que los resultados fueran aceptables. Para ello realizamos algunas pruebas sobre un subconjunto de la serie, quedándonos con cuatro variantes con distintas relaciones entre calidad de la solución y tiempo de ejecución invertido. Los detalles de estas pruebas se describen en la subsección 6.2.

Una vez determinadas las variantes las aplicamos a la serie completa, analizando la calidad de la solución y tiempo de CPU necesario para llegar a la misma. Estas pruebas están detalladas en la subsección 6.3.

Posteriormente aplicamos las mismas pruebas a una serie de grafos ralos generados de la misma forma que los anteriormente mencionados.

Para determinar la calidad de las soluciones, utilizamos una métrica que nos permite comparar objetivamente si una solución es mejor que otra. Como fue visto en la sección 3, el problema del cartero chino para un grafo G es cota inferior de la solución óptima de BCCP. Basándonos en esta propiedad medimos la calidad de nuestras soluciones en base al error relativo respecto a esta cota. El error relativo se define de la siguiente forma:

$$E_r = \frac{BCCP(G,k) - CPP(G)}{CPP(G)}$$

Esta misma métrica la usamos en todas las ocasiones en las que fue necesario evaluar la calidad de una solución obtenida en las secciones siguientes de este documento.

Los valores CPP para cada grafo los calculamos utilizando una implementación en Visual C++ 6.0, cuyo autor es Dominic Battré [1].

En la subsección 6.4 presentamos una comparación de los resultados obtenidos usando HBBCP con los presentados en [8].

Por último, en la subsección 6.5 presentamos algunas modificaciones al algoritmo para analizar posibles mejoras a realizarle.

Las pruebas fueron realizadas usando una implementación en C++ de nuestra heurística, la ejecución de las mismas fue llevada a cabo en una DELL Studio 15 con un procesador Intel i3-330M con un reloj de 2,13 Ghz y 4 GB de memoria RAM.

6.2 Cálculo de parámetros

Antes de poder realizar las pruebas de performance, fue necesario ajustar los parámetros del algoritmo. Para ello realizamos algunas pruebas sobre los grafos 10.2, 15.4, 20.2 y 25.3 de la serie de los densos. El objetivo de las mismas fue determinar los mejores valores para los siguientes parámetros:

- El número de iteraciones usadas en HBCCP
- La cantidad de ciclos de la vecindad, es decir el entero T en GreedyRandom
- La cantidad de ejes a recorrer al crear un ciclo, o sea el entero N en GenerarCicloAlAzar

Para determinar la cantidad óptima de iteraciones a usar, realizamos dos tipos de prueba.

La primera prueba consistió en ejecutar varias veces el programa variando la cantidad de iteraciones realizadas de 50 hasta 2000, incrementando de a 50 iteraciones y conservando el costo relativo de cada solución obtenida. Este proceso fue realizado 10 veces para cada uno de los grafos con el objetivo de reducir las variaciones inducidas azarosamente, debido a que nuestro algoritmo no es determinístico. El promedio de estas 10 corridas puede verse en el siguiente gráfico:

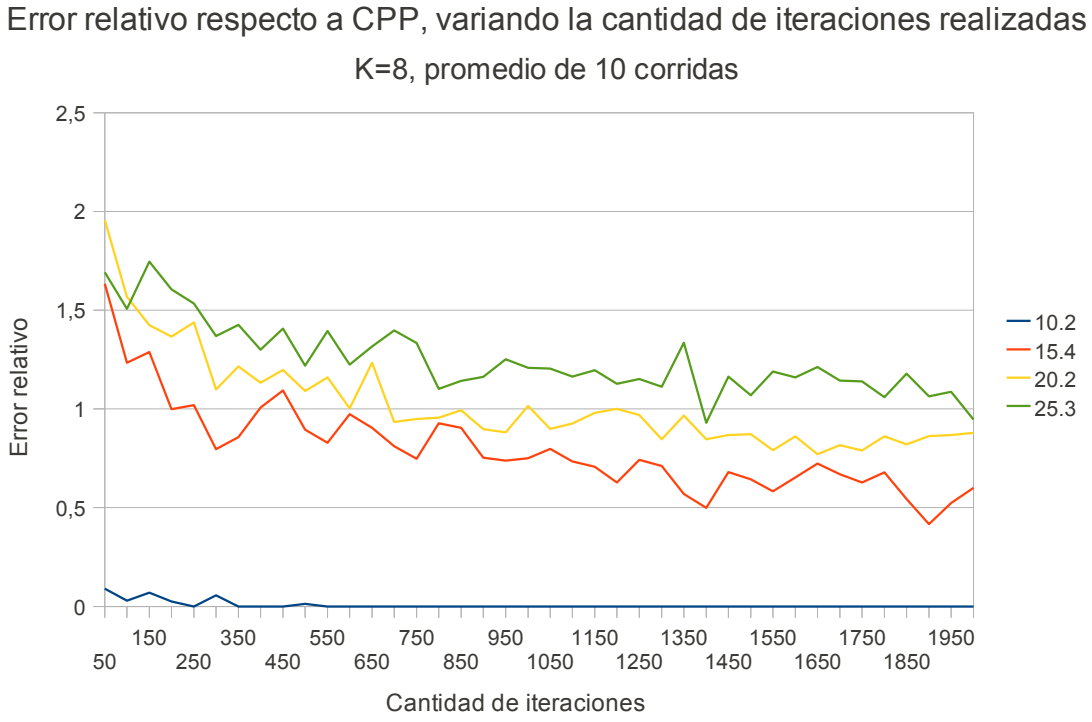


Figura 7: Error relativo dependiendo de la cantidad de iteraciones realizadas. Barrido lineal desde 50 a 2000.

En la figura 7 se puede ver una reducción del error a medida que se incrementa el número de iteraciones, más marcado al principio y que a partir de aproximadamente las 300 iteraciones se vuelve mucho más lento.

Para poder apreciar mejor este descenso, realizamos un barrido en el que fuimos incrementando la cantidad de iteraciones usando una escala logarítmica, empezando en 10 iteraciones y llegando hasta 10000. Esto nos permitió mostrar un rango de valores más grande, mostrando con detalle los cambios ocurridos para valores bajos del parámetro.

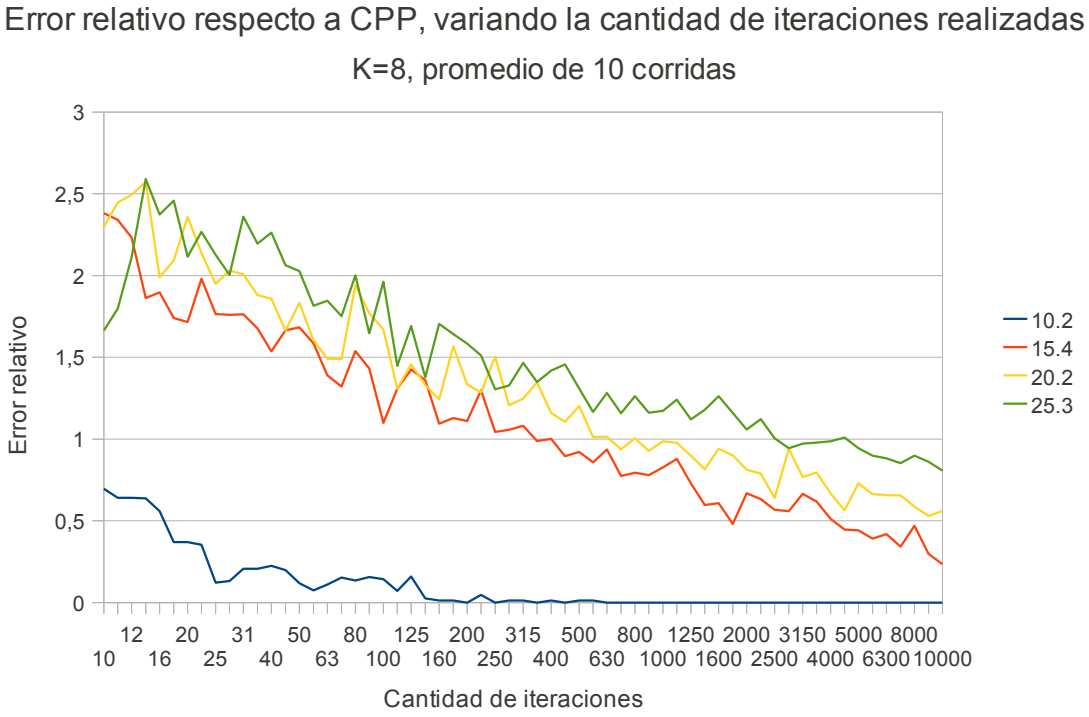


Figura 8: Error relativo dependiendo de la cantidad de iteraciones realizadas. Barrido en escala logarítmica de 10 a 10000.

En la figura 8 se ve más claramente que a medida que se incrementa el número de iteraciones realizadas, el error relativo decrece. La tasa de descenso se reduce a medida que se incrementa el número de iteraciones. En el caso de 10.2, se encuentra rápidamente la solución óptima, por lo que el error desciende rápidamente hasta llegar a cero después de aproximadamente 150 iteraciones, no desviándose significativamente pasado este punto.

En la segunda prueba ejecutamos la fase greedy del programa, tomando nota de las soluciones halladas y el número de iteraciones que fue necesario correr hasta hallarla.

Para esta prueba se realizaron 10 corridas sobre cada uno de los cuatro grafos de prueba para valores de K=6, 7 y 8, realizando en total 120 ejecuciones de nuestro algoritmo. El numero máximo de iteraciones lo fijamos en 10000, lo que consideramos un valor sufientememente alto para permitirnos ver claramente cuántas iteraciones son realmente necesarias. Cada solución fue marcada en un scatter plot, indicando el costo relativo de la misma respecto a la cantidad de iteraciones.

A continuación mostramos dos ejemplos de scatter plot obtenido para los grafos 10.2 y 20.2 con K=8. La serie completa se encuentra en la sección 9, Apéndice: Resultados .

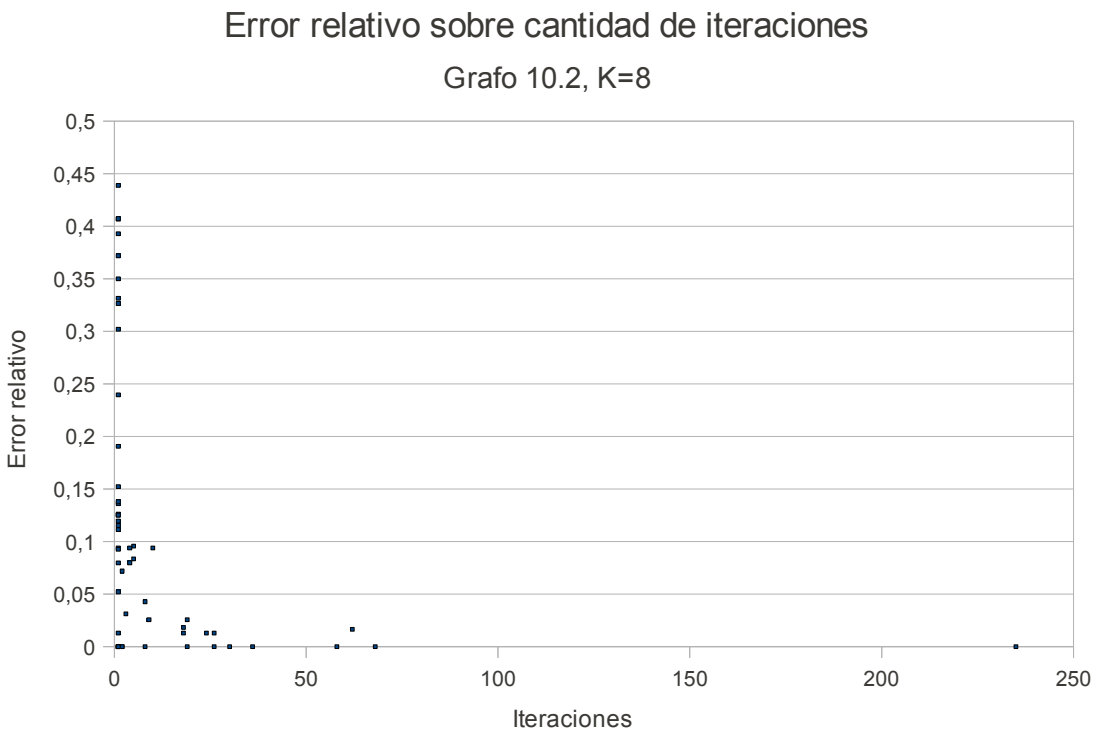


Figura 9: Scatter plot de soluciones obtenidas y las iteraciones necesarias para obtenerlas para el grafo 10.2.

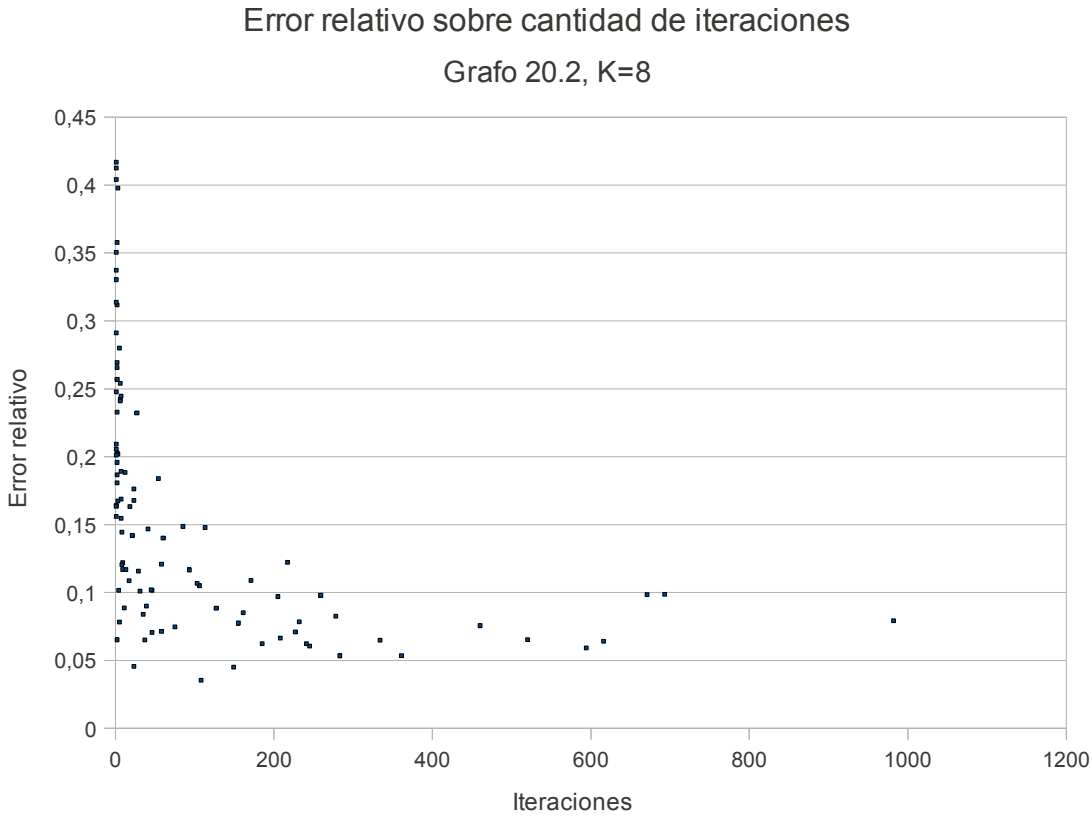


Figura 10: Scatter plot de soluciones obtenidas y las iteraciones necesarias para obtenerlas para el grafo 20.2.

Los resultados obtenidos mostraron fuertes diferencias entre los distintos grafos debido a que las diferentes características de cada uno hacen que varíe la dificultad del problema. Para el grafo 10.2 la mayoría de las soluciones fueron encontradas sin necesidad de exceder las 50 iteraciones. Para los otros dos grafos la mayoría de las soluciones fueron halladas antes de las 400 iteraciones, en ningún caso requiriéndose más de 1000 iteraciones.

Los dos tipos de pruebas realizados sobre el número de iteraciones indican que a medida que el número de iteraciones se incrementa, el error relativo tiende a reducirse, pero que la mayoría de las soluciones se encuentran antes de alcanzar las 50 para uno de los casos y aproximadamente 300 para los otros.

Estos resultados nos indican que para obtener buenos resultados, se deberían realizar al menos unas 300 iteraciones, pero que sería innecesario exceder las 1000.

El segundo parámetro evaluado fue la cantidad de ciclos a generar para utilizar en la fase greedy. Este parámetro define cuántos ciclos deben generarse y probarse cuando se desea agregar un nuevo ciclo a una posible solución.

Para determinar el número de ciclos a usar, variamos la cantidad de ciclos a generar desde 4 hasta 248, incrementando de a 4 ciclos y realizando 10 corridas para cada grafo. Adicionalmente, la cantidad de iteraciones fue reducida a 25 para evitar que el gran número de iteraciones enmascare el cambio en la cantidad de ciclos generados.

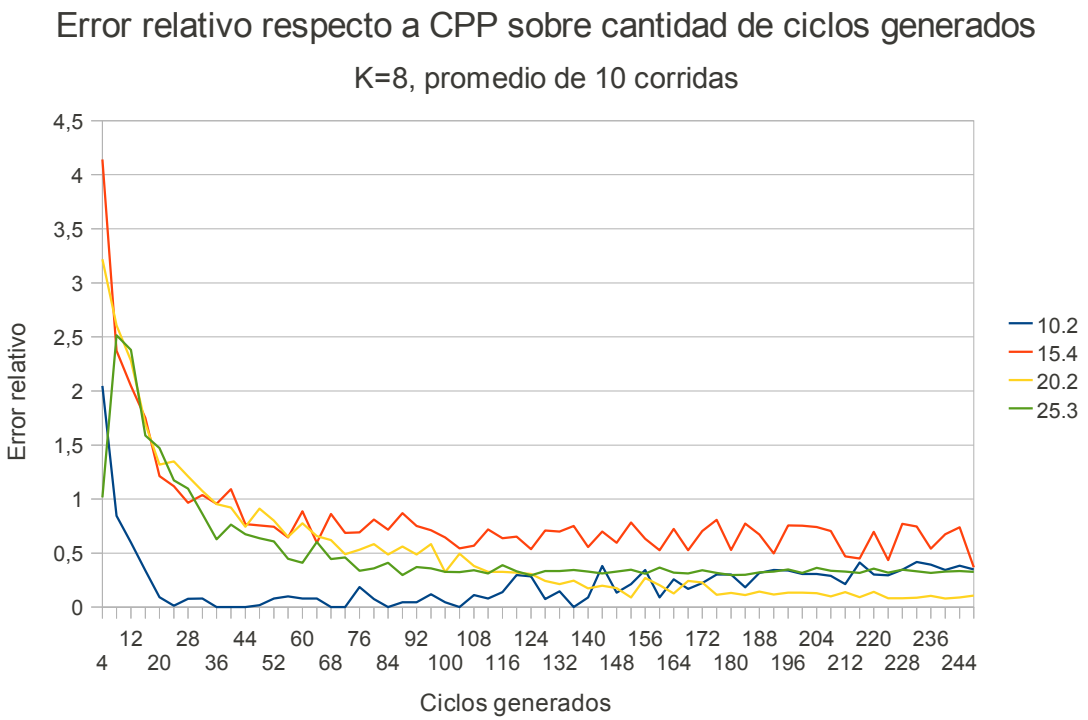


Figura 11: Error relativo de las soluciones, a medida que se varía el número de ciclos a generar para competir cada vez que se desea agregar un nuevo ciclo a la solución. Barrido lineal de 4 a 248.

En la figura 11 se puede apreciar que con un número reducido de ciclos las soluciones tienden a ser malas. A medida que se aumenta la cantidad de ciclos a generar el error tiende a reducirse hasta estabilizarse pasado cierto punto dependiente del grafo de entrada. En nuestros ejemplos, este valor varía de 20 para 10.2 hasta aproximadamente 140 para 20.2.

Para poder apreciar las variaciones del error en un rango más grande de cantidad de iteraciones y poder determinar mejor el valor más conveniente de este parámetro, variamos el mismo desde 4 hasta 248.

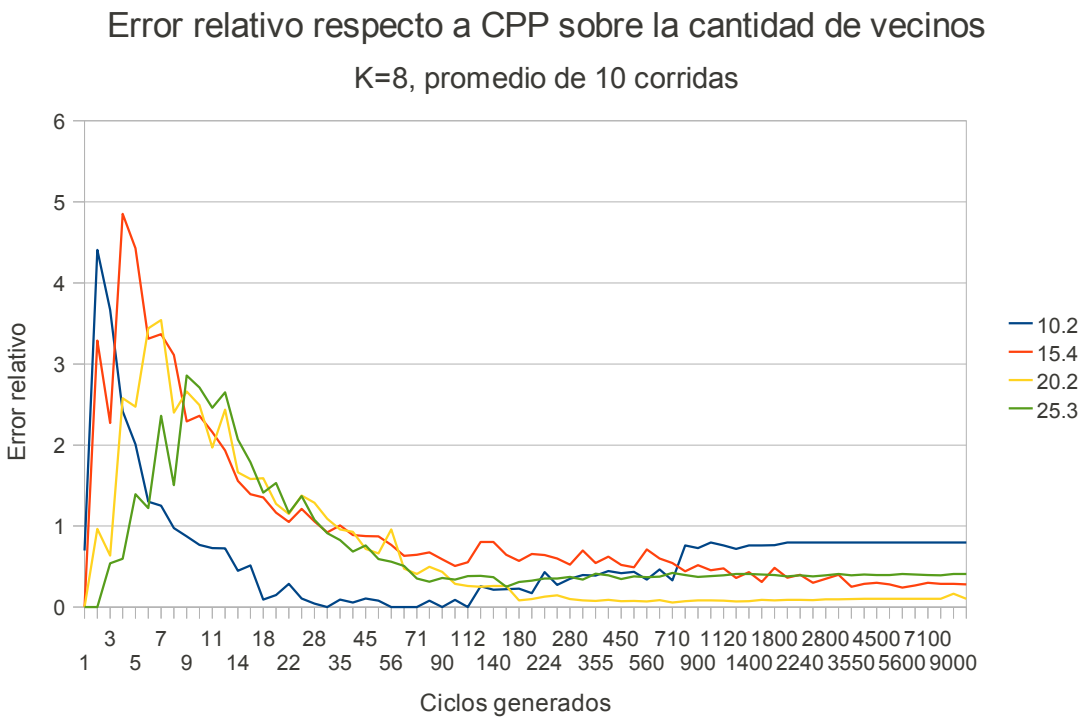


Figura 12: Error relativo de las soluciones, a medida que se varía el número de ciclos a generar para competir cada vez que se desea agregar un nuevo ciclo a la solución. Barrido con escala logaritmica de 1 a 10000.

Analizando la figura 12 se ven resultados similares a la figura 11, pero también se aprecia un aumento del error para la mayoría de los grafos pasado cierto número de ciclos generados.

El decrecimiento inicial del error se debe a que cuando el número de ciclos generados es demasiado pequeño, la elección de un nuevo candidato tiende a ser al azar, con un escaso componente greedy. Al elegir ciclos al azar, sin considerar la calidad de los mismos, la solución final tiende a ser de baja calidad.

A medida que la cantidad de ciclos generados se incrementa, las soluciones tienden a mejorar, puesto que se establece un balance entre la tendencia a elegir una solución greedy y el componente azaroso que permite evaluar soluciones que a corto plazo no parecen convenientes, pero que resultan en costos menores en el resultado final.

Por último, una vez excedido cierto valor, las soluciones tienden a empeorar una vez más, ya que al incrementar excesivamente los ciclos generados aumenta la posibilidad de que aparezcan ciclos aparentemente muy buenos de acuerdo con la función greedy, lo que causa que el algoritmo se vuelve excesivamente goloso y pierda la capacidad de explorar en busca de soluciones alternativas, impidiendole detectar otras de mejor calidad que las alcanzables por un algoritmo goloso tradicional.

Basándonos en las figuras 11 y 12, consideramos conveniente generar entre 20 y 80 ciclos para comparar usando la función greedy. Este valor no debería ser demasiado alto no solo debido a que las soluciones tienden a empeorar si este parámetro es excesivamente grande, sino que también obtener una gran cantidad de ciclos en esta

etapa es costoso e incrementa significativamente el tiempo de ejecución.

Para evaluar el valor más conveniente de la cantidad de ejes a probar al generar un ciclo, realizamos pruebas similares a las usadas para determinar los valores de los otros parámetros, variándolo desde 5 ejes por ciclo a 200, incrementando de a 5. Al igual que para el cálculo de los otros, para cada combinación de grafo y valor del parámetro se realizaron 10 corridas, que fueron promediadas en el gráfico.

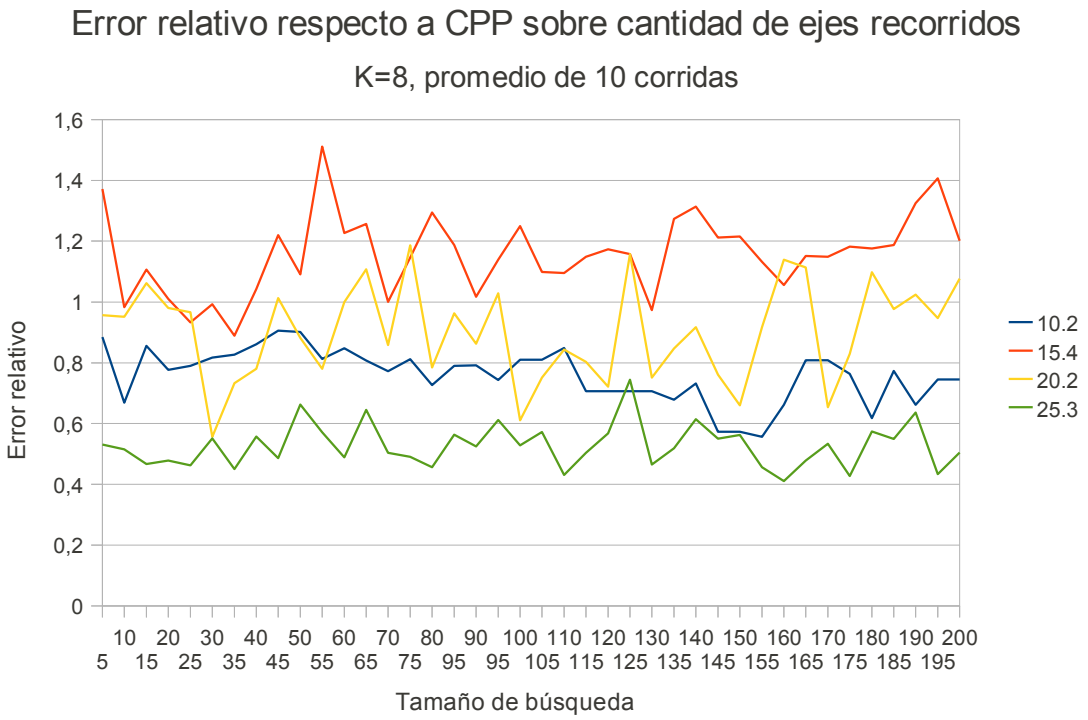


Figura 13: Error relativo respecto a la cantidad de ejes recorridos durante la generación de un ciclo.

Como se ve en la figura 13, este parámetro no guarda relación con la calidad de la solución obtenida. Esto se debe a que una vez que se recorren suficientes ejes, se comienzan a encontrar ciclos con facilidad, y rara vez se terminan recorriendo la cantidad de ejes máxima.

Este parámetro no puede ser excesivamente corto, siendo K el tamaño mínimo. Debido al poco impacto que tiene en la calidad de la solución, decidimos dejarlo en la cantidad de nodos del grafo de entrada, para no limitar excesivamente la generación de ciclos, pero no volvimos a considerar este parámetro en nuestras pruebas.

Una vez obtenida una aproximación de los parámetros a usar, decidimos hacer un ajuste fino de los mismos. Para realizar este ajuste, calculamos una serie de soluciones variando los dos parámetros que determinamos que afectaban la calidad de las soluciones obtenidas, tomando el error promedio y el tiempo de CPU utilizado.

Costos (% sobre CPP)	Iteraciones									
Vecinos	100	200	300	400	500	600	700	800	900	1000
10	11,629312	10,136963	10,033488	9,011957	7,959467	7,468082	7,866565	7,498558	6,808078	6,824914
15	7,237533	5,857979	6,585349	5,389996	5,346164	5,012775	4,505015	4,532362	4,284026	4,894403
20	5,494189	4,762946	4,608918	4,06811	3,754107	2,852346	2,86883	2,617097	2,752197	2,707233
25	4,618436	3,890994	3,128382	2,864094	2,846615	2,412566	1,970503	1,8272708	2,0842852	1,850171
30	3,784003	2,4410404	2,618384	2,118963	1,829981	1,7995672	1,5386142	1,6042292	1,2996887	1,4032961
35	3,435783	2,445818	1,650422	1,4034369	1,4062891	1,2681607	1,2617709	1,3511156	1,0560603	0,9309488
40	2,77244	2,048677	1,647437	1,199613	1,4745224	0,8563809	1,1084767	1,0412126	0,8236331	0,9187965
45	2,332736	1,5240571	1,2424397	1,324039	1,0552381	1,0355024	0,9110821	0,6904561	0,8573808	0,7614084
50	2,06497	1,5938462	1,1704922	1,2320075	0,962887	0,87945162	0,7363417	0,7402184	0,6854122	0,7770246

Figura 14: Error relativo promedio dependiendo de la cantidad de iteraciones realizadas y de los ciclos candidatos para formar la solución. El promedio se realizó sobre los grafos 10.2, 15.4, 20.2 y 25.3.

Para evitar variaciones causadas por el componente azaroso de la primera parte de la heurística consideramos el promedio de los resultados obtenidos de correrla 10 veces para cada combinación, tanto para calidad como para tiempo.

Tiempos (segundos de CPU)	Iteraciones									
Vecinos	100	200	300	400	500	600	700	800	900	1000
10	2,55375	5,201	6,75275	9,83075	11,4155	15,50475	18,90875	17,58025	20,80875	23,706
15	3,65125	7,5115	10,57775	14,97875	19,18375	22,39925	21,91825	28,5355	34,3645	37,963
20	4,28825	8,59675	14,128	18,0935	25,89125	26,8865	34,475	36,01975	41,4595	45,032
25	5,71725	10,80725	15,141	19,019	24,48125	29,2505	34,2285	38,68625	44,28175	47,15025
30	5,3755	12,6065	15,10375	23,2185	30,6465	32,58675	37,047	40,2535	48,387	58,4215
35	6,20325	12,24325	14,9205	24,207	31,74575	36,4315	41,6325	48,55225	56,3045	56,4255
40	5,933	11,75325	18,58075	25,7365	30,821	33,63825	44,79975	52,739	52,04475	58,75625
45	6,82375	13,35775	17,51875	25,06125	31,058	39,24	39,60675	45,8145	49,9645	59,6375
50	5,757	14,8775	19,10575	26,42025	32,087	46,537	42,81	48,92325	60,60525	63,74375

Figura 15: Tiempo de CPU promedio necesario para calcular las soluciones, dependiendo de la cantidad de iteraciones realizadas y de los ciclos candidatos para formar la solución. El promedio se realizó sobre los grafos 10.2, 15.4, 20.2 y 25.3.

Una vez obtenidas las tablas de calidad y tiempos, figuras 14 y 15 respectivamente, usamos los datos para crear una tercera tabla conteniendo la relación entre ellas. Esta tabla, que se muestra en la figura 16, indica cuál es la combinación de parámetros que obtiene la mejor solución usando el menor tiempo posible.

Relacion performance/tiempo (1/costo * 1/tiempo)	Iteraciones									
Vecinos	100	200	300	400	500	600	700	800	900	1000
10	0,03367190	0,01896729	0,01475936	0,01128741	0,01100579	0,00863627	0,00672283	0,00758573	0,00705878	0,00618080
15	0,03784146	0,02272613	0,01435582	0,01238614	0,00975044	0,00890612	0,01012740	0,00773197	0,00679263	0,00538195
20	0,04244400	0,02442250	0,01535749	0,01358578	0,01028822	0,01303958	0,01011093	0,01060815	0,00876388	0,00820263
25	0,03787197	0,02378068	0,02111182	0,01835799	0,01434953	0,01417058	0,01482638	0,01414622	0,01083473	0,01146315
30	0,04916201	0,03249604	0,02528610	0,02032556	0,01783087	0,01705261	0,01754354	0,01548567	0,01590128	0,01219770
35	0,04691967	0,03339482	0,04060894	0,02943514	0,02239958	0,02164455	0,01903650	0,01524397	0,01681776	0,01903701
40	0,06079439	0,04153063	0,03266841	0,03238988	0,02200401	0,03471360	0,02013714	0,01821079	0,02332863	0,01852365
45	0,06282193	0,04912080	0,04594323	0,03013676	0,03051238	0,02461047	0,02771235	0,03161266	0,02334343	0,02202231
50	0,08411821	0,04217194	0,04471646	0,03072202	0,03236649	0,02443372	0,03172308	0,02761371	0,02407343	0,02018960

Figura 16: Relación entre calidad de la solución y tiempo necesario para obtenerla.

Basandonos en estos resultados, decidimos usar cuatro variantes de los parámetros. La primera variante, 500 iteraciones y 50 ciclos generados como vecinos, fue usada como configuración por defecto al ser una combinación de parámetros que brinda soluciones de una calidad aceptable de acuerdo con la información obtenida en esta subsección y manteniendo un tiempo de cómputo razonablemente bajo. Una segunda variante,

usando 100 iteraciones y 15 vecinos fue elegida como una combinación que sacrifica calidad de solución para minimizar el tiempo de cálculo. La tercera variante, 100 iteraciones y 50 ciclos generados se corresponde con la configuración más eficiente en relación tiempo versus calidad de acuerdo con la figura 16. Por último elegimos una combinación que maximiza la calidad de la solución a costa del tiempo de cómputo, con 1000 iteraciones y 50 vecinos.

6.3 Resultados obtenidos

Una vez hecho el ajuste de parámetros, realizamos corridas con el fin de obtener datos para comparar precisión y tiempo de ejecución entre las distintas variantes.

Para estos cálculos usamos las familias de grafos planares y euclideanos de [8] tanto la de los densos como la de los ralos. Estas familias de grafos se dividen en grupos de 10, 15, 20, 25, 30 y 35 nodos. Cada grupo cuenta con 5 grafos distintos, salvo en la serie de 10 de la familia de ralos, en donde hay solamente 4. Adicionalmente no consideramos el grafo 20.2 de la misma familia debido a que tiene un eje con costo 0, lo que no permite que sea procesado por nuestro programa. Nuestra heurística fue probada con valores de K = 4, 5, 6, 7 y 8, por lo que realizamos un total de 150 corridas para la familia de los densos, y 140 para los grafos ralos, para cada variante de nuestro algoritmo.

En la tabla 17 reportamos el error relativo promedio para las cuatro variantes del algoritmo determinadas en la subsección 6.2, aplicadas sobre la familia de grafos densos.

Grafos		Porcentaje de desvío respecto a CPP			
K	N	Variante 1	Variante 2	Variante 3	Variante 4
4	10	1,44	1,07	1,39	1,39
4	15	2,01	5,03	1,93	1,18
4	20	4,51	8,46	7,05	3,97
4	25	4,59	9,40	5,00	4,86
4	30	7,22	12,34	8,77	6,09
4	35	4,28	14,70	5,30	3,76
5	10	1,39	1,63	1,39	1,44
5	15	0,57	5,81	0,80	0,48
5	20	4,28	10,73	4,78	3,07
5	25	4,33	12,53	6,16	4,43
5	30	6,22	17,20	7,49	5,64
5	35	4,46	15,63	5,09	4,13
6	10	1,44	0,79	1,44	1,39
6	15	0,68	4,25	1,45	0,48
6	20	3,77	11,87	5,04	3,02
6	25	4,97	14,35	5,52	4,48
6	30	7,00	19,68	6,93	5,87
6	35	4,78	17,48	5,71	4,41
7	10	1,39	2,07	1,44	1,39
7	15	0,48	6,76	1,29	0,34
7	20	3,48	8,79	5,44	3,12
7	25	4,95	11,53	4,94	4,54
7	30	6,11	18,56	7,47	6,33
7	35	4,72	16,12	6,17	3,90
8	10	1,39	1,16	1,44	1,39
8	15	0,48	6,14	0,63	0,42
8	20	4,28	11,38	4,87	3,11
8	25	4,63	12,52	6,89	3,78
8	30	6,19	17,01	8,01	5,87
8	35	4,95	19,55	6,98	3,75
Promedio		**3,70**	**10,48**	**4,56**	**3,27**

Figura 17: Error relativo con respecto a CPP de las cuatro variantes de nuestra heurística.

La tabla presentada arriba describe la calidad de las soluciones obtenidas usando como métrica el porcentaje promedio del error relativo respecto a CPP. En cada fila de la tabla evaluamos un grupo de grafos de un tamaño en particular, realizando una corrida con el K indicado. Para cada grafo en el grupo obtuvimos el error relativo y promediamos estos errores. La tabla a su vez se divide en columnas, cada una de ellas indicando la performance de cada una de las variantes descriptas al final de la subsección 6.2. Cada celda de la columna contiene los errores relativos descriptos en el párrafo anterior obtenidos al aplicar la variante indicada en la primera fila. En la última fila promediamos los errores relativos de todas las combinaciones de grafos y tamaños de K. Esta fila puede usarse para evaluar el comportamiento general de cada variante bajo una gran diversidad de situaciones.

De las cuatro variantes usadas, tres de ellas tienen una performance similar. La segunda

variante obtuvo soluciones de una calidad notoriamente peor, con un error relativo promedio superior al doble del de los resultados de las otras variantes. Esta es la variante usada para minimizar el tiempo de ejecución, en perjuicio de la calidad de las soluciones, por lo que era de esperar que el error fuera mayor que en las otras.

A continuación presentamos una tabla detallando el tiempo utilizado por cada variante del algoritmo durante el cómputo de las soluciones.

Grafos		Tiempo de CPU en segundos											
		Variante 1			Variante 2			Variante 3			Variante 4		
K	N	Greedy	LS	Total	Greedy	LS	Total	Greedy	LS	Total	Greedy	LS	Total
4	10	1,772	0,246	2,02	0,204	0,08	0,28	0,412	0,048	0,46	3,588	0,554	4,14
4	15	5,946	0,956	6,90	0,628	0,196	0,82	1,546	0,242	1,79	12,2	2,054	14,25
4	20	13,652	3,016	16,67	1,134	0,312	1,45	4	0,868	4,87	35,52	7,872	43,39
4	25	26,47	6,346	32,82	2,116	0,396	2,51	5,848	1,484	7,33	50,266	12,134	62,40
4	30	45,182	11,04	56,22	2,774	0,352	3,13	10,44	2,352	12,79	75,506	18,166	93,67
4	35	69,35	14,656	84,01	3,854	0,372	4,23	12,684	2,794	15,48	115,804	25,272	141,08
5	10	1,81	0,466	2,28	0,168	0,142	0,31	0,402	0,11	0,51	3,572	0,934	4,51
5	15	5,946	2,002	7,95	0,552	0,492	1,04	1,38	0,434	1,81	11,042	3,652	14,69
5	20	14,538	7,264	21,80	1,464	1,04	2,50	3,348	1,642	4,99	28,93	14,544	43,47
5	25	24,338	14,32	38,66	1,76	1,146	2,91	4,954	2,916	7,87	39,614	23,85	63,46
5	30	34,458	21,78	56,24	3,148	1,626	4,77	7,84	4,874	12,71	75,124	46,28	121,40
5	35	86,896	51,462	138,36	4,43	1,324	5,75	13,296	7,716	21,01	107,698	63,742	171,44
6	10	1,696	0,63	2,33	0,254	0,36	0,61	0,402	0,118	0,52	3,382	1,194	4,58
6	15	6,97	3,706	10,68	0,562	0,668	1,23	1,302	0,688	1,99	11,024	5,574	16,60
6	20	13,752	11,392	25,14	0,93	1,386	2,32	4,098	3,426	7,52	40,264	33,944	74,21
6	25	32,318	33,356	65,67	1,404	1,896	3,30	6,514	6,732	13,25	52,174	54,736	106,91
6	30	42,12	46,67	88,79	2,746	2,738	5,48	10,346	11,588	21,93	81,45	89,848	171,30
6	35	53,396	58,246	111,64	5,004	3,05	8,05	12,294	13,708	26,00	135,716	147,524	283,24
7	10	2,114	1,01	3,12	0,174	0,296	0,47	0,418	0,244	0,66	3,548	1,62	5,17
7	15	7,162	5,102	12,26	0,474	0,876	1,35	1,594	1,108	2,70	11,812	8,446	20,26
7	20	14,244	16,998	31,24	1,78	4,314	6,09	2,88	3,436	6,32	38,63	45,894	84,52
7	25	25,958	39,256	65,21	1,688	3,688	5,38	7,71	11,866	19,58	67,794	100,944	168,74
7	30	38,202	63,392	101,59	2,366	4,254	6,62	9,31	15,2	24,51	73,612	121,818	195,43
7	35	52,19	87,542	139,73	3,826	4,11	7,94	13,806	23,954	37,76	137,598	229,834	367,43
8	10	1,752	0,954	2,71	0,138	0,344	0,48	0,352	0,26	0,61	3,904	2,296	6,20
8	15	6,652	5,924	12,58	0,596	1,674	2,27	1,372	1,266	2,64	12,18	11,474	23,65
8	20	13,174	20,406	33,58	1,158	3,788	4,95	2,566	4,022	6,59	26,754	41,276	68,03
8	25	30,712	61,408	92,12	1,872	5,694	7,57	6,474	13,09	19,56	48,556	95,188	143,74
8	30	44,87	99	143,87	3,176	6,91	10,09	10,742	23,864	34,61	81,972	182,612	264,58
8	35	74,038	172,286	246,32	3,402	6,182	9,58	11,838	27,27	39,11	109,458	258,402	367,86
Promedio		**26,39**	**28,69**	**55,08**	**1,79**	**1,99**	**3,78**	**5,67**	**6,24**	**11,92**	**49,96**	**55,06**	**105,01**

Figura 18: Tiempo de CPU de las cuatro variantes de nuestra heurística.

Esta tabla muestra el tiempo de cómputo empleado en el cálculo de los resultados presentados en la tabla 17. La organización de esta tabla es similar a la de la anterior aunque en lugar de una celda por cada combinación se presentan tres: una indicando el tiempo empleado en la fase greedy, otra con el tiempo usado en la búsqueda local, y una tercera con la suma de ambos.

Se puede observar en la tabla 18 que dos de las variantes, la variante 2 y la 3, no utilizaron gran cantidad de tiempo de CPU, mientras que las variantes 1 y 4

consumieron una cantidad significativamente mayor, siendo el tiempo empleado por la variante 4 el doble del de la 1.

Las diferencias de tiempo y performance vistas en ambas tablas pueden ser explicadas si se tiene en cuenta las razones que llevaron a la creación de las distintas variantes. La primera variante, como fue visto en la sección 6.2, fue creada específicamente con la calidad de los resultados en mente: los valores de todos los parámetros se corresponden con puntos en donde la solución mejora considerablemente. En contraposición, la variante 3 fue elegida en base a la relación tiempo/preformance, aún sabiendo que la calidad de las soluciones que se esperaba que brindara sería menor.

En la categoría de las variantes pensadas para performance, la 4 emplea casi el doble de tiempo que la variante 1. Los resultados son similares, siendo el resultado de la variante 4 apenas 10% menor al de la variante 1. Considerando que ambos errores son bajos, aproximadamente 3,5% sobre el valor de CPP, esta mejora no es significativa.

Es importante tener en cuenta que estas comparaciones son respecto al error relativo sobre CPP, que es solo una cota inferior de la solución exacta de BCCP, cuyo costo probablemente sea mayor.

Respecto a las variantes 2 y 3, pensadas para obtener resultados rápidamente, a pesar de que la variante 3 consume tres veces más tiempo que la variante 2, los resultados obtenidos por la 3 tienen la mitad de error promedio sobre CPP. Consideramos que la misma es significativamente mejor que la variante 2, puesto que la calidad de las soluciones obtenidas está cerca de los valores devueltos por otras variantes del algoritmo que consumen mucho más tiempo de CPU, mientras que el incremento requerido no es excesivamente grande en términos absolutos, debido a que ambas variantes consumen poco tiempo de procesador para obtener sus respuestas (3,78 segundos para la variante 2 y 11,92 para la variante 3).

Comparando entre los dos grupos, podemos ver que la variante 1 necesita unas 5 veces más tiempo que la variante 3, logrando una mejora moderada de alrededor del 2% de error relativo.

Una vez analizados los grafos densos, repetimos las pruebas sobre la familia de grafos ralos. A diferencia de los densos, no todas los grafos pudieron ser resueltos para todos los valores de K usados anteriormente. En la tabla 19 mostramos para cuales encontramos soluciones de BCCP. Usando esta familia de grafos, cinco de las instancias no tuvieron solución para K=7, otras cinco no tuvieron para K=6, seis adicionales no pudieron ser resueltas para K=5 y solamente 10.2 devolvió una solución válida para K=4.

K	4	5	6	7	8
10,2	Si	Si	Si	Si	Si
10,3	No	Si	Si	Si	Si
10,4	No	Si	Si	Si	Si
10,5	No	No	Si	Si	Si
15,1	No	Si	Si	Si	Si
15,2	No	Si	Si	Si	Si
15,3	No	No	Si	Si	Si
15,4	No	No	No	Si	Si
15,5	No	No	No	No	Si
20,1	No	Si	Si	Si	Si
20,3	No	No	Si	Si	Si
20,4	No	No	No	Si	Si
20,5	No	No	No	No	Si
25,1	No	Si	Si	Si	Si
25,2	No	Si	Si	Si	Si
25,3	No	No	Si	Si	Si
25,4	No	No	No	Si	Si
25,5	No	No	No	No	Si
30,1	No	Si	Si	Si	Si
30,2	No	Si	Si	Si	Si
30,3	No	No	Si	Si	Si
30,4	No	No	No	Si	Si
30,5	No	No	No	No	Si
35,1	No	Si	Si	Si	Si
35,2	No	Si	Si	Si	Si
35,3	No	No	Si	Si	Si
35,4	No	No	No	Si	Si
35,5	No	No	No	No	Si

Figura 19: Grafos ralos resolubles.

En la tabla 20 mostramos el resultado de repetir las pruebas de la tabla 17, usando los grafos de la familia de los ralos. En los casos para los que no pudimos encontrar una solución, no se consideró el grafo, realizando el promedio con los grafos restantes.

Grafos		Porcentaje de desvío respecto a CPP			
K	N	Variante 1	Variante 2	Variante 3	Variante 4
4	10	4,66045	4,66045	4,66045	4,66045
4	15				
4	20				
4	25				
4	30				
4	35				
5	10	6,27779	6,27779	10,092	6,27779
5	15	8,72021	8,72021	8,72021	8,72021
5	20	5,20995	5,20995	9,87558	6,92068
5	25	6,19906	9,66285	6,46982	5,47704
5	30	8,41783	14,8563	11,4551	10,5972
5	35	8,42253	11,7074	9,57723	8,09412
6	10	1,18138	0,645321	1,18138	0,645321
6	15	8,68636	5,42982	8,88225	4,93109
6	20	5,54148	8,20021	6,39685	4,69655
6	25	6,38095	13,1562	9,40442	6,40833
6	30	6,06434	14,7115	9,61971	6,7469
6	35	6,28557	11,607	6,43963	6,28665
7	10	0,645321	3,43944	0,645321	0,645321
7	15	6,72498	6,92143	8,67684	8,48095
7	20	6,13141	4,22732	6,35346	4,44225
7	25	6,38435	10,7371	6,95458	5,76487
7	30	6,42265	15,5305	7,54676	5,34963
7	35	5,45211	11,0363	5,28261	5,54788
8	10	0,645321	0,645321	0,645321	0,645321
8	15	4,92548	4,56647	4,92548	4,85405
8	20	6,61983	9,77972	9,77202	6,88468
8	25	5,42186	10,9349	5,96161	4,34307
8	30	5,86338	14,975	9,62306	6,02776
8	35	5,77651	13,7452	6,82984	6,08432
Promedio		**5,72**	**8,86**	**7,04**	**5,58**

Figura 20: Error relativo con respecto a CPP de las cuatro variantes de nuestra heurística usando la familia de grafos ralos.

Grafos		Tiempo de CPU en segundos											
		Variante 1			Variante 2			Variante 3			Variante 4		
K	N	Greedy	LS	Total	Greedy	LS	Total	Greedy	LS	Total	Greedy	LS	Total
4	10	1,23	0,13	1,36	0,08	0,02	0,10	0,27	0,02	0,29	2,46	0,31	2,77
4	15												
4	20												
4	25												
4	30												
4	35												
5	10	1,21333	0,26333	1,48	0,08	0,05	0,13	0,22667	0,05667	0,28	2,52	0,51333	3,03
5	15	3,135	0,685	3,82	0,3	0,085	0,39	0,63	0,13	0,76	5,55	1,03	6,58
5	20	11,86	4,72	16,58	0,41	0,36	0,77	1,41	0,55	1,96	20,87	10,04	30,91
5	25	14,525	4,625	19,15	0,87	0,22	1,09	4,99	1,58	6,57	33,375	11,11	44,49
5	30	35,155	10,81	45,97	2,41	0,365	2,78	5,915	1,415	7,33	74,52	18,135	92,66
5	35	28,665	11,865	40,53	2,695	1,05	3,75	8,75	3,8	12,55	76,5	33,18	109,68
6	10	0,98	0,2875	1,27	0,0875	0,065	0,15	0,22	0,0475	0,27	1,9425	0,5975	2,54
6	15	2,55	1,10667	3,66	0,26667	0,23	0,50	0,74667	0,31333	1,06	5,9	2,15333	8,05
6	20	10,37	6,94	17,31	0,685	0,695	1,38	1,94	1,145	3,09	20,65	13,385	34,04
6	25	14,8367	8,49333	23,33	0,91	0,72333	1,63	5,56	3,66667	9,23	25,97	15,0867	41,06
6	30	31,8067	19,0433	50,85	1,52667	0,67	2,20	7,11333	4,57667	11,69	68,7967	47,56	116,36
6	35	41,4233	37,6267	79,05	2,05333	1,57667	3,63	10,1633	9,39333	19,56	81,2867	80,3267	161,61
7	10	0,9425	0,3775	1,32	0,07	0,125	0,20	0,2175	0,0675	0,29	1,855	0,7325	2,59
7	15	3,02	1,4425	4,46	0,2825	0,36	0,64	0,5825	0,3775	0,96	5,6925	2,5975	8,29
7	20	8,44	8,64667	17,09	0,58333	1,16333	1,75	2,10667	2,18667	4,29	14,5767	14,89	29,47
7	25	15,0625	18,395	33,46	1,4	1,895	3,30	4,1975	4,0675	8,27	29,19	33,2525	62,44
7	30	30,055	34,795	64,85	1,9325	2,3075	4,24	7,4225	8,425	15,85	53,655	65,345	119,00
7	35	47,49	71,65	119,14	2,355	3,6625	6,02	12,625	18,815	31,44	100,615	145,825	246,44
8	10	0,975	0,4575	1,43	0,0725	0,1425	0,22	0,225	0,075	0,30	1,88	0,9775	2,86
8	15	2,626	2,078	4,70	0,216	0,546	0,76	0,586	0,454	1,04	5,454	4,316	9,77
8	20	9,7375	12,7975	22,54	0,7675	2,305	3,07	2,205	2,77	4,98	16,175	19,2	35,38
8	25	15,19	23,178	38,37	0,92	2	2,92	2,816	3,886	6,70	44,076	65,994	110,07
8	30	27,742	49,792	77,53	1,998	4,47	6,47	7,144	13,44	20,58	73,252	129,04	202,29
8	35	53,242	102,266	155,51	3,084	7,074	10,16	8,272	16,55	24,82	86,172	171,16	257,33
Promedio		**16,49**	**17,30**	**33,79**	**1,04**	**1,29**	**2,33**	**3,85**	**3,91**	**7,77**	**34,12**	**35,47**	**69,59**

Figura 21: Tiempo de CPU de las cuatro variantes de nuestra heurística usando la familia de grafos ralos.

El promedio de error para las cuatro variantes fue más alto que en el caso de los grafos densos. A diferencia de los resultados densos, las cuatro variantes tuvieron una performance similar, habiendo una pequeña diferencia entre la calidad de las soluciones de las variantes 1 y 4 en comparación con las variantes 2 y 3.

Como se puede ver en la tabla 21, la relación de los tiempos entre las distintas variantes se mantiene similar a los mostrados en la tabla 18, correspondiente a familia de los grafos densos. Para la familia de grafos ralos, nuestra heurística consumió menos tiempo de CPU que para la de los grafos densos, necesitando en promedio la mitad de tiempo para resolver las instancias.

Para verificar si los parámetros elegidos para las cuatro variantes, que fueron elegidos en base a una muestra de la familia de los grafos densos, siguen siendo adecuados para la familia de los grafos ralos, realizamos las mismas pruebas que las realizadas en la sección 6.2, figuras 14, 15 y 16, usando una muestra de la familia de los ralos.

Costos (% sobre CPP)	Iteraciones									
Vecinos	100	200	300	400	500	600	700	800	900	1000
10	10,771101	9,636032	9,649337	7,863396	7,437464	7,733952	6,850086	6,597727	7,186995	6,615243
15	8,53017	7,623766	6,238732	5,739937	5,753256	5,265198	5,48399	4,98155	5,015631	4,978996
20	6,917745	5,393137	5,268412	4,771979	4,634316	4,395183	4,370477	4,25499	4,190186	4,059565
25	6,037972	4,705052	4,264507	4,281473	4,138374	4,064061	3,951386	4,033888	3,70412	3,825153
30	5,339915	4,652124	3,914096	4,06946	3,705815	3,694584	3,604198	3,529657	3,434566	3,324775
35	4,583278	4,125682	3,915365	3,732103	3,657323	3,516275	3,429051	3,404698	3,092824	3,174094
40	4,214012	3,89868	3,779381	3,654539	3,467288	3,305129	3,199903	3,103679	3,037773	2,999168
45	4,490538	3,685162	3,61132	3,542722	3,463771	3,136877	3,012553	3,098961	2,86169	2,791671
50	4,187683	3,694054	3,54371	3,450668	3,147929	3,145861	3,040255	2,970879	2,853556	2,999409

Figura 22: Error relativo promedio dependiendo de la cantidad de iteraciones realizadas y de los ciclos candidatos para formar la solución. Realizado sobre 10.4, 15.4, 20.3, 25.2, 30.3 y 35.1 de la serie de grafos ralos.

Comparando la figura 22 con la 14 podemos ver que con los grafos ralos hay menos dispersión en los errores, aunque los promedios son superiores. En general la distribución de la calidad de las soluciones es similar para ambas familias.

Tiempos (segundos de CPU)	Iteraciones									
Vecinos	100	200	300	400	500	600	700	800	900	1000
10	2,437833	5,583671	8,347997	9,506667	15,15167	17,01799	18,91634	19,81018	23,477	27,06134
15	4,414167	8,696663	12,57766	17,69716	21,688	26,80416	29,66933	32,39367	37,13581	40,46316
20	6,135339	11,19867	18,40516	21,95217	33,60882	36,59519	38,54366	50,49166	55,42684	63,75749
25	7,532502	13,88516	20,8	25,57267	37,44566	39,1045	42,31934	51,5355	57,80349	70,02635
30	7,293834	17,06084	24,11499	33,5535	38,64184	45,21499	54,82117	61,26018	65,98866	77,13285
35	8,486162	16,4085	26,294	37,3185	41,15267	48,81267	65,31933	59,10401	75,23503	84,00263
40	9,114163	19,12984	25,06799	35,78483	44,38682	59,57033	64,57449	73,074	76,19334	85,49587
45	10,227673	17,72167	30,92984	38,97516	41,34551	54,66834	70,565	74,98633	77,26367	91,83372
50	11,673336	20,04267	31,88484	40,07967	49,62249	57,08183	65,22033	84,7727	86,82382	91,92129

Figura 23: Tiempo de CPU promedio necesario para calcular las soluciones, dependiendo de la cantidad de iteraciones realizadas y de los ciclos candidatos para formar la solución. El promedio se realizó sobre los grafos 10.4, 15.4, 20.3, 25.2, 30.3 y 35.1 de la serie de grafos ralos.

Los tiempos presentados en la figura 23 son ligeramente superiores a los de la figura 15, aunque se debe tomar en cuenta que la muestra de los grafos ralos contiene algunos grafos de mayor tamaño (mayor cantidad de nodos) que la muestra de los densos. De todas formas, la distribución de los tiempos es similar, aunque no sufre una penalidad tan grande al incrementar el número de iteraciones como en el caso de los grafos densos.

Relacion performance/tiempo (1/costo * 1/tiempo)	Iteraciones									
Vecinos	100	200	300	400	500	600	700	800	900	1000
10	0,03808342	0,01858583	0,01241424	0,01337709	0,00887390	0,00759784	0,00771733	0,00765098	0,00592666	0,00558605
15	0,02655789	0,01508266	0,01274394	0,00984440	0,00801432	0,00708571	0,00614604	0,00619691	0,00536885	0,00496362
20	0,02356117	0,01655740	0,01031290	0,00954606	0,00642039	0,00621726	0,00593633	0,00465459	0,00430573	0,00386357
25	0,02198719	0,01530681	0,01127374	0,00913336	0,00645311	0,00629235	0,00598014	0,00481027	0,00467047	0,00373327
30	0,02567496	0,01259935	0,01059452	0,00732361	0,00698326	0,00598621	0,00506108	0,00462476	0,00441224	0,00389941
35	0,02571062	0,01477187	0,00971340	0,00717996	0,00664414	0,00582619	0,00446462	0,00496941	0,00429759	0,00375048
40	0,02603679	0,01340822	0,01055504	0,00764660	0,00649765	0,00507904	0,00483952	0,00440921	0,00432044	0,00389991
45	0,02177333	0,01531224	0,00895275	0,00724228	0,00698269	0,00583132	0,00470409	0,00430330	0,00452274	0,00390062
50	0,02045649	0,01350645	0,00885029	0,00723057	0,00640172	0,00556881	0,00504321	0,00397063	0,00403622	0,00362701

Figura 24: Relación entre calidad de la solución y tiempo necesario para obtenerla para los grafos ralos.

La figura 24 muestra que la relación entre performance y tiempo no es exactamente igual. El punto óptimo en la familia de los ralos ocurre con menos ciclos generados como vecinos. Esto se debe a que los tiempos aumentan mucho más que con los grafos densos cuando se incremetan los vecinos, como se puede ver en la figura 23.

Basándonos en los datos presentados, concluímos que los parámetros elegidos para las variantes son aceptables también para la familia de grafos ralos.

6.4 Comparación de los resultados

En esta sección comparamos los resultados obtenidos con los presentados en el trabajo de D. Hochbaum y E. Olinick [8] con el fin de determinar la calidad de nuestra heurística en relación a otros algoritmos de resolución de BCCP.
En su trabajo, Olinick y Hochbaum usaron un proceso de dos fases. En la primera usaron distintas heurísticas para generar un conjunto de ciclos candidatos que contenga un cubrimiento de K-ciclos del grafo, no necesariamente óptimo. Una vez obtenido este conjunto, emplearon dos técnicas de programación lineal entera para resolver el problema de cubrimiento de conjuntos, obteniendo una solución exacta en un caso (basada en el procedimiento de branch and bound) y aproximada en el otro.

Para la fase de generación de ciclos usaron las siguientes heurísticas:

1. Greedy Postman Heuristic: Partiendo de un cubrimiento obtenido de una resolución de CPP, se generan los K-ciclos necesarios para conseguir un cubrimiento acotado por ciclos, reemplazando los degenerados y los que superan el tamaño límite por otros K-ciclos válidos.
2. Cycle Basis (CB): Construye un conjunto limitado de K-ciclos, para luego aplicar un algoritmo goloso que construye el cubrimiento.
3. Augmented Cycle Basis (AB): Aplica Cycle Basis, y en caso de no encontrar un cubrimiento, agrega los K-ciclos necesarios para completarlo.

Debido a que para nuestras corridas usamos los mismos grafos que en el trabajo de Olinick y Hochbaum [8], y que la forma de medir los resultados también fue realizada de la misma manera, podemos comparar directamente con los datos obtenidos en la subsección 6.3.

A continuación presentamos una comparación entre los resultados obtenidos por las cuatro variantes de HBCCP con los resultados de aplicar la heurística aproximada de programación lineal a las distintas formas de generar los ciclos candidatos empleadas por Olinick y Hochbaum.

Grafos		Porcentaje de desvío respecto a CPP						
		HBCCP				Programación lineal		
K	N	Variante 1	Variante 2	Variante 3	Variante 4	Greedy Heuristic	Cycle Basis (CB)	Augmented Basis (AB)
6	10	1,44	0,79	1,44	1,39	1,82	4,57	13,23
6	15	0,68	4,25	1,45	0,48	6,28	6,63	9,72
6	20	3,77	11,87	5,04	3,02	8,52	22,47	16,82
6	25	4,97	14,35	5,52	4,48	11,77	16,12	11,55
6	30	7,00	19,68	6,93	5,87	5,94	32,58	29,28
6	35	4,78	17,48	5,71	4,41	10,29	42,44	29,02
7	10	1,39	2,07	1,44	1,39	1,82	4,57	13,23
7	15	0,48	6,76	1,29	0,34	4,75	6,63	9,72
7	20	3,48	8,79	5,44	3,12	8,43	22,47	16,82
7	25	4,95	11,53	4,94	4,54	8,95	16,12	11,55
7	30	6,11	18,56	7,47	6,33	5,94	32,58	29,28
7	35	4,72	16,12	6,17	3,90	10,28	42,44	29,02
8	10	1,39	1,16	1,44	1,39	1,82	4,57	13,23
8	15	0,48	6,14	0,63	0,42	3,56	6,63	9,72
8	20	4,28	11,38	4,87	3,11	7,18	22,47	16,82
8	25	4,63	12,52	6,89	3,78	6,76	16,12	11,55
8	30	6,19	17,01	8,01	5,87	4,95	32,58	29,28
8	35	4,95	19,55	6,98	3,75	7,67	42,44	29,02
Promedio		**3,65**	**11,11**	**4,54**	**3,20**	**6,49**	**20,80**	**18,27**

Figura 25: Comparación entre los resultados de las variantes de HBCCP y los obtenidos por Olinick y Hochbaum usando programación lineal. Errores expresados como porcentaje sobre CPP.

En esta comparación se puede ver que HBCCP obtiene en la mayoría de los casos mejores resultados que los obtenidos en [8]. La única excepción ocurrió en la segunda variante de HBCCP al compararla con Greedy Heuristic, esta variante fue pensada para minimizar el tiempo de CPU empleado a costa de la calidad de las soluciones.

Adicionalmente comparamos los resultados de las cuatro variantes de HBCCP con los resultados obtenidos en [8] al aplicar la opción de programación lineal basada en branch and bound como segunda fase.

En esta variante Olinick y Hochbaum propusieron dos formas adicionales para generar los ciclos candidatos.

4. Greedy + CB: Une el conjuntos de ciclos obtenido usando la heurística Greedy Postman con el de los ciclos generados con Cycle Basis.
5. Greedy + AB: Similar a la opción anterior, pero uniendo los ciclos de Greedy Postman con los de Augmented Basis.

Grafos		Porcentaje de desvío respecto a CPP								
		HBCCP				Branch and bound				
K	N	Variante 1	Variante 2	Variante 3	Variante 4	Greedy Heuristic	Cycle Basis (CB)	Augmented Basis (AB)	Greedy + CB	Greedy + AB
6	10	1,44	0,79	1,44	1,39	1,82	4,57	4,44	0,56	0,56
6	15	0,68	4,25	1,45	0,48	6,28	5,18	5,02	2,85	2,71
6	20	3,77	11,87	5,04	3,02	8,52	6,35	5,58	2,67	2,2
6	25	4,97	14,35	5,52	4,48	11,77	3,34	2,99	2,59	2,59
6	30	7,00	19,68	6,93	5,87	5,94	4,82	4,6	2,98	2,71
6	35	4,78	17,48	5,71	4,41	10,29	3,69	3,41	2,49	2,38
7	10	1,39	2,07	1,44	1,39	1,82	4,57	4,44	0,56	0,56
7	15	0,48	6,76	1,29	0,34	4,75	5,18	5,02	2,26	2,12
7	20	3,48	8,79	5,44	3,12	8,43	6,35	5,58	2,58	2,11
7	25	4,95	11,53	4,94	4,54	8,95	3,34	2,99	2,53	2,53
7	30	6,11	18,56	7,47	6,33	5,94	4,82	4,6	2,98	2,71
7	35	4,72	16,12	6,17	3,90	10,28	3,69	3,41	2,49	2,38
8	10	1,39	1,16	1,44	1,39	1,82	4,57	4,44	0,56	0,56
8	15	0,48	6,14	0,63	0,42	3,56	5,18	5,02	1,07	1,07
8	20	4,28	11,38	4,87	3,11	7,18	6,35	5,58	1,9	1,67
8	25	4,63	12,52	6,89	3,78	6,76	3,34	2,99	2,42	2,37
8	30	6,19	17,01	8,01	5,87	4,95	4,82	4,6	2,63	2,37
8	35	4,95	19,55	6,98	3,75	7,67	3,69	3,41	2,14	2,03
Promedio		**3,65**	**11,11**	**4,54**	**3,20**	**6,49**	**4,66**	**4,34**	**2,13**	**1,98**

Figura 26: Comparación entre los resultados de las variantes de HBCCP y los obtenidos por Olinick y Hochbaum usando branch and bound. Errores expresados como porcentaje sobre CPP.

Como se puede observar en la tabla 26, los resultados obtenidos por HBCCP son de una calidad similar a los obtenidos usando las dos variantes de Cycle Basis, con resultados claramente mejores usado las variantes 1 y 4 de HBCCP. Sin embargo, las versiones que combinan la heurística greedy con CB y AB son superiores a las cuatro variantes de HBCCP.

A continuación, en la figura 27 mostraremos una comparación de tiempos entre las variantes de HBCCP y las pruebas realizadas en [8]. Es importante tener presente que esta comparación es meramente informativa, ya que las pruebas fueron realizadas en computadoras con características muy distintas. Los ensayos del trabajo de Olinick y Hochbaum fueron realizados en una computadora Sun Microsystems SPARCstation 20 modelo 16, con un procesaror SuperSPARC de 60Mhz y 32 MB de memoria RAM.

Grafos		Tiempo de CPU en segundos						
		HBCCP				Programacion lineal		
K	N	Variante 1	Variante 2	Variante 3	Variante 4	Greedy Heuristic	Cycle Basis (CB)	Augmented Basis (AB)
6	10	2,326	0,614	0,520	4,576	0,044	0,122	0,136
6	15	10,676	1,230	1,990	16,598	0,058	0,852	0,890
6	20	25,144	2,316	7,524	74,208	0,094	3,444	3,476
6	25	65,674	3,300	13,246	106,910	0,136	7,640	7,732
6	30	88,790	5,484	21,934	171,298	0,234	16,460	16,634
6	35	111,642	8,054	26,002	283,240	0,330	37,304	37,710
7	10	3,124	0,470	0,662	5,168	0,044	0,122	0,136
7	15	12,264	1,350	2,702	20,258	0,058	0,852	0,890
7	20	31,242	6,094	6,316	84,524	0,094	3,444	3,476
7	25	65,214	5,376	19,576	168,738	0,136	7,640	7,732
7	30	101,594	6,620	24,510	195,430	0,234	16,460	16,634
7	35	139,732	7,936	37,760	367,432	0,330	37,304	37,710
8	10	2,706	0,482	0,612	6,200	0,150	0,122	0,136
8	15	12,576	2,270	2,638	23,654	0,058	0,852	0,890
8	20	33,580	4,946	6,588	68,030	0,094	3,444	3,476
8	25	92,120	7,566	19,564	143,744	0,136	7,640	7,732
8	30	143,870	10,086	34,606	264,584	0,234	16,460	16,634
8	35	246,324	9,584	39,108	367,860	0,330	37,304	37,710
Promedio		**55,084**	**3,783**	**11,916**	**105,012**	**0,155**	**10,970**	**11,096**

Figura 27: Comparación de tiempo de ejecución de HBCCP con las variantes de programación lineal de Olinick y Hochbaum. Los tiempos exactos para K=6 y 7 no fueron presentados en [8], pero se aclara que son similares a los tiempos con K=8.

Comparando los tiempos de ejecución es claro que HBCCP es más costosa en comparación con las tres heurísticas con las que se compara. De las cuatro variantes, sólo las variantes 2 y 3 emplean una cantidad similar de segundos de CPU. Esta diferencia es aún más marcada si consideramos que el CPU empleado para realizar los cálculos de HBCCP es mucho más poderoso que el empleado por Olinick y Hochbaum en su trabajo.

Grafos		Tiempo de CPU en segundos								
		HBCCP				Branch and Bound				
K	N	Variante 1	Variante 2	Variante 3	Variante 4	Greedy Heuristic	Cycle Basis (CB)	Augmented Basis (AB)	Greedy + CB	Greedy + AB
6	10	2,326	0,614	0,520	4,576	0,008	0,008	0,044	0,018	0,844
6	15	10,676	1,230	1,990	16,598	0,000	0,004	0,066	0,016	0,092
6	20	25,144	2,316	7,524	74,208	0,004	0,016	0,060	0,026	0,186
6	25	65,674	3,300	13,246	106,910	0,012	0,018	0,374	0,028	1,210
6	30	88,790	5,484	21,934	171,298	0,040	0,028	0,570	0,058	5,546
6	35	111,642	8,054	26,002	283,240	0,060	0,032	1,846	0,258	104,984
7	10	3,124	0,470	0,662	5,168	0,006	0,006	0,040	0,012	0,046
7	15	12,264	1,350	2,702	20,258	0,004	0,008	0,054	0,020	0,074
7	20	31,242	6,094	6,316	84,524	0,002	0,010	0,070	0,034	0,198
7	25	65,214	5,376	19,576	168,738	0,004	0,014	0,404	0,058	1,018
7	30	101,594	6,620	24,510	195,430	0,002	0,028	0,596	0,056	5,452
7	35	139,732	7,936	37,760	367,432	0,008	0,032	1,838	0,614	62,634
8	10	2,706	0,482	0,612	6,200	0,006	0,006	0,036	0,016	0,046
8	15	12,576	2,270	2,638	23,654	0,004	0,010	0,056	0,022	0,046
8	20	33,580	4,946	6,588	68,030	0,008	0,022	0,064	0,024	0,112
8	25	92,120	7,566	19,564	143,744	0,004	0,016	0,398	0,070	0,948
8	30	143,870	10,086	34,606	264,584	0,012	0,022	0,608	0,050	7,018
8	35	246,324	9,584	39,108	367,860	0,004	0,036	1,880	0,480	64,850
Promedio		**55,084**	**3,783**	**11,916**	**105,012**	**0,010**	**0,018**	**0,500**	**0,103**	**14,184**

Figura 28: Comparación de tiempo de ejecución de HBCCP con las variantes de branch and bound de Olinick y Hochbaum.

En la figura 28 se puede apreciar una diferencia aún mayor entre los tiempos empleados por HBCCP y las variantes que usan branch and bound. Sólo las variantes 2 y 3 emplean un tiempo menor que Greedy + AB.

En la tabla 29 presentada a continuación realizamos la misma comparación que en la tabla 25 pero aplicada a la familia de los grafos ralos.

Grafos		Porcentaje de desvío respecto a CPP						
		HBCCP				Programación lineal		
K	N	Variante 1	Variante 2	Variante 3	Variante 4	Greedy Heuristic	Cycle Basis (CB)	Augmented Basis (AB)
6	10	1,18	0,65	1,18	0,65	10,78	16,48	7,34
6	15	8,69	5,43	8,88	4,93	8,99	13,04	6,02
6	20	5,54	8,20	6,40	4,70	10,82	15,00	4,01
6	25	6,38	13,16	9,40	6,41	11,37	12,59	5,89
6	30	6,06	14,71	9,62	6,75	11,34	17,60	6,90
6	35	6,29	11,61	6,44	6,29	9,29	32,28	5,51
7	10	0,65	3,44	0,65	0,65	10,78	16,48	7,34
7	15	6,72	6,92	8,68	8,48	12,18	23,12	8,24
7	20	6,13	4,23	6,35	4,44	5,85	15,25	2,37
7	25	6,38	10,74	6,95	5,76	9,68	14,83	5,78
7	30	6,42	15,53	7,55	5,35	8,51	18,16	6,04
7	35	5,45	11,04	5,28	5,55	9,83	31,35	5,27
8	10	0,65	0,65	0,65	0,65	7,53	16,48	4,09
8	15	4,93	4,57	4,93	4,85	9,60	23,11	7,39
8	20	6,62	9,78	9,77	6,88	7,98	19,07	4,68
8	25	5,42	10,93	5,96	4,34	11,82	16,33	4,63
8	30	5,86	14,98	9,62	6,03	7,50	20,08	5,73
8	35	5,78	13,75	6,83	6,08	8,50	31,00	5,29
Promedio		**5,29**	**8,90**	**6,40**	**4,93**	**9,58**	**19,57**	**5,70**

Figura 29: Comparación entre los resultados de las variantes de HBCCP y los obtenidos por Olinick y Hochbaum para los experimentos con grafos ralos. Errores expresados como porcentaje sobre CPP.

Con esta familia de grafos los resultados obtenidos por HBCCP fueron mejores que los consiguidos en [8] en la mayoría de los casos. Sólo la variante de Augmented Basis obtuvo mejores resultados en múltiples casos.

Grafos		Tiempo de CPU en segundos						
		HBCCP				Programación lineal		
K	N	Variante 1	Variante 2	Variante 3	Variante 4	Greedy Heuristic	Cycle Basis (CB)	Augmented Basis (AB)
6	10	1,27	0,15	0,27	2,54	0,010	0,070	0,054
6	15	3,66	0,50	1,06	8,05	0,014	0,088	0,136
6	20	17,31	1,38	3,09	34,04	0,036	0,248	0,431
6	25	23,33	1,63	9,23	41,06	0,050	0,494	1,286
6	30	50,85	2,20	11,69	116,36	0,070	0,638	1,424
6	35	79,05	3,63	19,56	161,61	0,114	1,124	135,553
7	10	1,32	0,20	0,29	2,59	0,014	0,072	0,054
7	15	4,46	0,64	0,96	8,29	0,016	0,144	0,217
7	20	17,09	1,75	4,29	29,47	0,040	0,440	0,699
7	25	33,46	3,30	8,27	62,44	0,078	0,902	1,311
7	30	64,85	4,24	15,85	119,00	0,106	1,778	2,696
7	35	119,14	6,02	31,44	246,44	0,144	2,538	569,580
8	10	1,43	0,22	0,30	2,86	0,014	0,062	0,048
8	15	4,70	0,76	1,04	9,77	0,024	0,156	0,210
8	20	22,54	3,07	4,98	35,38	0,042	0,506	0,694
8	25	38,37	2,92	6,70	110,07	0,086	1,096	1,354
8	30	77,53	6,47	20,58	202,29	0,128	2,332	4,626
8	35	155,51	10,16	24,82	257,33	0,176	3,424	164,962
Promedio		**39,77**	**2,73**	**9,13**	**80,53**	**0,065**	**0,895**	**49,185**

Figura 30: Comparación de tiempo de ejecución de HBCCP con las variantes de grafos ralos de Olinick y Hochbaum.

Al igual que con la familia de grafos densos, lo tiempos utilizados por HBCCP son mayores que los necesario para correr las heurísticas usadas por Olinick y Hochbaum. Cabe notar que la variante de Augmented Basis necesitó una gran cantidad de tiempo para llegar a un resultado con los grafos de 35 nodos, comparado con el resto.

6.5 Modificaciones experimentales del algoritmo

En esta subsección describimos una serie de modificaciones que le realizamos al algoritmo para evaluar posibles formas de mejorarlo, tanto respecto a la calidad de las soluciones como al tiempo de cómputo necesario para obtenerlas.

6.5.1 Precómputo de ciclos

Esta modificación consiste en calcular inicialmente un conjunto de ciclos y reutilizarlos en las distintas iteraciones. Esto fue logrado sustituyendo las rutinas de generación de ciclos mencionadas en la sección 5 para retornar un ciclo al azar de este conjunto en lugar de calcular uno nuevo en cada invocación.

El objetivo de esta modificación fue evitar el descarte y el recómputo de los ciclos en cada iteración y de esa forma reducir el costo computacional. Esta mejora fue pensada estrictamente con esta reducción en mente, aunque fue necesario verificar que las soluciones obtenidas sigan siendo de una calidad similar a las que se pueden conseguir con el algoritmo original.

Las pruebas fueron realizadas pregenerando 500 ciclos antes de iniciar las iteraciones del ciclo principal de HBCCP. Una vez generado este conjunto, cualquier invocación a

la función encargada de obtener un ciclo lo toma del conjunto aleatoriamente en lugar de generar uno nuevo.

En las figuras 31 y 32 mostramos los resultados de esta variante aplicados a la famila de los grafos densos.

Grafos		Porcentaje de desvío respecto a CPP			
K	N	Variante 1	Variante 2	Variante 3	Variante 4
4	10	1,43745	2,12964	1,43745	1,43745
4	15	1,1826	4,49074	2,03843	0,827754
4	20	5,50461	10,6412	7,55082	3,9102
4	25	5,18984	13,9087	7,25617	5,16888
4	30	6,87815	17,1417	8,93805	6,49698
4	35	4,93278	16,9638	6,51673	4,25657
5	10	1,43745	1,02068	1,43745	1,43745
5	15	0,840742	4,44355	0,502608	0,475937
5	20	4,4497	10,6815	5,17928	3,83387
5	25	5,03331	13,4845	6,96967	4,11202
5	30	7,81391	18,7646	7,24685	6,80058
5	35	4,34741	21,1625	5,76566	4,07626
6	10	1,02068	0,739093	1,43745	1,39271
6	15	0,502608	5,51228	1,60558	0,467343
6	20	4,32789	11,4507	6,10011	3,77518
6	25	5,96112	12,3317	6,88502	4,41342
6	30	6,40093	20,1602	7,6806	7,13522
6	35	4,90223	16,9243	6,18505	5,51207
7	10	1,11112	2,29653	1,43745	1,43745
7	15	0,768305	5,60834	3,31087	0,363259
7	20	3,35178	9,85142	6,13729	3,51743
7	25	3,90665	14,0262	6,31691	4,62403
7	30	7,378	20,2296	7,42729	5,23718
7	35	5,72877	18,6581	8,40812	4,5371
8	10	1,39271	1,11015	1,37002	1,02068
8	15	0,485615	7,46136	1,85931	0,502608
8	20	3,89735	11,2108	5,37599	4,84261
8	25	4,19228	14,2452	7,58614	4,74509
8	30	7,25101	18,6755	9,82726	6,90284
8	35	5,9781	17,8886	5,29241	4,97661
Promedio		**3,92**	**11,44**	**5,17**	**3,61**

Figura 31: Resultados obtenidos usando la variante de precómputo, aplicada a la familia de los grafos densos. Los valores dados están en porcentaje sobre CPP.

Grafos		Tiempo de CPU en segundos											
		Variante 1			Variante 2			Variante 3			Variante 4		
K	N	Greedy	LS	Total	Greedy	LS	Total	Greedy	LS	Total	Greedy	LS	Total
4	10	0,96	0,29	1,25	0,088	0,044	0,13	0,238	0,052	0,29	1,866	0,576	2,44
4	15	3,958	1,2	5,16	0,324	0,184	0,51	0,876	0,248	1,12	7,374	2,062	9,44
4	20	8,426	2,76	11,19	0,7	0,292	0,99	2,098	0,768	2,87	21,722	6,874	28,60
4	25	19,46	6,502	25,96	1,464	0,494	1,96	4,088	1,316	5,40	37,686	13,804	51,49
4	30	25,996	8,394	34,39	2,144	0,432	2,58	7,346	2,152	9,50	78,158	25,52	103,68
4	35	44,904	12,802	57,71	3,552	0,316	3,87	10,116	2,892	13,01	123,018	30,882	153,90
5	10	0,882	0,416	1,30	0,094	0,132	0,23	0,22	0,104	0,32	1,8	0,794	2,59
5	15	3,404	2,05	5,45	0,246	0,362	0,61	0,806	0,492	1,30	6,322	3,554	9,88
5	20	10,402	7,784	18,19	0,76	0,808	1,57	2,002	1,522	3,52	18,022	14,508	32,53
5	25	21,032	17,728	38,76	1,562	0,982	2,54	5,764	5,23	10,99	44,352	37,092	81,44
5	30	33,586	28,432	62,02	2,134	1,622	3,76	6,268	5,54	11,81	62,424	49,514	111,94
5	35	45,592	32,38	77,97	4,036	1,45	5,49	7,708	6,028	13,74	117,334	76,618	193,95
6	10	0,872	0,672	1,54	0,094	0,22	0,31	0,222	0,162	0,38	1,906	1,272	3,18
6	15	3,214	2,916	6,13	0,25	0,69	0,94	0,718	0,644	1,36	6,174	5,53	11,70
6	20	12,566	16,052	28,62	0,714	1,192	1,91	1,756	2,364	4,12	15,828	21,09	36,92
6	25	17,258	24,728	41,99	1,308	1,852	3,16	3,81	5,308	9,12	44,7	67,934	112,63
6	30	43,686	60,71	104,40	1,85	2,908	4,76	7,34	10,448	17,79	83,428	124,166	207,59
6	35	65,034	79,08	144,11	3,646	3,362	7,01	9,608	11,934	21,54	78,258	101,854	180,11
7	10	1,13	1,086	2,22	0,084	0,176	0,26	0,17	0,202	0,37	1,892	1,524	3,42
7	15	3,776	4,478	8,25	0,244	0,944	1,19	0,556	0,758	1,31	5,642	6,928	12,57
7	20	12,818	23,196	36,01	0,712	2,458	3,17	2,826	5,252	8,08	23,634	42,374	66,01
7	25	22,192	47,778	69,97	1,53	3,57	5,10	4,118	8,778	12,90	30,88	67,14	98,02
7	30	46,454	91,94	138,39	1,97	3,222	5,19	7,008	15,27	22,28	70,13	148,522	218,65
7	35	43,346	89,806	133,15	3,032	2,992	6,02	8,9	18,858	27,76	98,926	228,478	327,40
8	10	0,872	1,03	1,90	0,07	0,27	0,34	0,224	0,182	0,41	1,716	1,952	3,67
8	15	3,242	4,936	8,18	0,374	1,66	2,03	0,658	0,96	1,62	7,538	11,9	19,44
8	20	10,576	25,654	36,23	0,62	2,994	3,61	1,924	4,736	6,66	29,81	71,74	101,55
8	25	16,774	47,016	63,79	1,292	4,36	5,65	5,2	14,422	19,62	46,224	124,756	170,98
8	30	27,942	82,942	110,88	1,656	5,562	7,22	6,814	20,354	27,17	49,326	149,688	199,01
8	35	41,384	121,002	162,39	3,916	7,928	11,84	11,134	30,928	42,06	94,74	281,132	375,87
Promedio		**19,72**	**28,19**	**47,92**	**1,35**	**1,78**	**3,13**	**4,02**	**5,93**	**9,95**	**40,36**	**57,33**	**97,69**

Figura 32: Tiempo de CPU, en segundos, de las cuatro variantes de nuestra heurística, aplicado a la familia de grafos densos.

Como podemos ver al comparar la figura 31 con la figura 17, la calidad de las soluciones son similares, con pequeñas mejoras en los errores relativos obtenidos con la variante original. Respecto a los tiempos, mirando las figuras 32 y 18 podemos ver que las corridas consumieron prácticamente lo mismo, terminando unos pocos segundos antes que la versión original.

En las figuras 33 y 34 mostramos los resultados de esta variante aplicados a la famila de los grafos ralos.

Grafos		Porcentaje de desvío respecto a CPP			
K	N	Variante 1	Variante 2	Variante 3	Variante 4
4	10	4,66045	4,66045	4,66045	4,66045
4	15				
4	20				
4	25				
4	30				
4	35				
5	10	10,092	6,27779	6,80304	10,092
5	15	8,72021	8,52432	8,72021	8,72021
5	20	6,92068	5,20995	5,20995	9,56454
5	25	5,30532	10,9552	11,0819	6,69545
5	30	6,73712	18,09	11,4223	10,4982
5	35	8,58908	11,9423	10,448	8,52364
6	10	1,18138	0,578744	1,18138	1,11481
6	15	9,17608	9,49746	8,88225	8,88225
6	20	5,71626	11,5476	7,80697	3,54576
6	25	7,03899	17,2322	7,76669	6,36741
6	30	6,61472	16,0342	8,90355	6,06444
6	35	6,30436	14,2143	6,97301	4,5754
7	10	0,645321	1,13852	0,645321	0,645321
7	15	8,67684	8,91642	8,67684	8,84824
7	20	6,13569	7,45347	5,35541	4,34232
7	25	8,61048	9,89412	9,91597	7,41405
7	30	8,43148	15,5956	6,97956	5,87401
7	35	6,5158	14,6582	8,62783	5,70303
8	10	0,645321	0,578744	0,645321	0,645321
8	15	5,10178	7,1673	5,10178	6,44779
8	20	8,42781	10,8202	11,0423	4,15025
8	25	7,34347	13,2515	7,69308	5,08765
8	30	6,68381	15,1555	10,3983	6,38755
8	35	6,78084	15,3304	8,55766	6,58045
Promedio		**6,44**	**10,19**	**7,34**	**6,06**

Figura 33: Resultados obtenidos usando la variante de precómputo, aplicada a la familia de los grafos ralos. Los valores dados están en porcentaje sobre CPP.

Grafos		Tiempo de CPU en segundos											
		Variante 1			Variante 2			Variante 3			Variante 4		
K	N	Greedy	LS	Total	Greedy	LS	Total	Greedy	LS	Total	Greedy	LS	Total
4	10	0,63	0,16	0,79	0,04	0,03	0,07	0,16	0,03	0,19	1,25	0,34	1,59
4	15												
4	20												
4	25												
4	30												
4	35												
5	10	0,80333	0,31333	1,12	0,06667	0,04333	0,11	0,12	0,03667	0,16	1,46	0,52	1,98
5	15	1,57	0,705	2,28	0,155	0,11	0,27	0,43	0,115	0,55	3,24	1,175	4,42
5	20	4,41	2,71	7,12	0,26	0,32	0,58	1,57	0,85	2,42	13,55	8,66	22,21
5	25	10,17	4,26	14,43	0,82	0,27	1,09	2,37	1,075	3,45	23,525	7,12	30,65
5	30	26,97	6,015	32,99	1,96	0,39	2,35	5,125	1,335	6,46	33,23	15	48,23
5	35	35,665	20,51	56,18	2,255	0,52	2,78	6,605	2,42	9,03	66,82	45,965	112,79
6	10	0,515	0,265	0,78	0,03	0,085	0,12	0,1275	0,065	0,19	1,2675	0,6375	1,91
6	15	2,02667	1,33667	3,36	0,13333	0,28667	0,42	0,37667	0,23333	0,61	5,11667	3,4	8,52
6	20	7,31	4,765	12,08	0,6	0,755	1,36	1,365	1,26	2,63	9,935	9,735	19,67
6	25	11,6133	9,93667	21,55	1,07	0,66333	1,73	1,7	2,1	3,80	29,78	32,79	62,57
6	30	27,34	28,185	55,53	1,60333	1,46333	3,07	5,16	4,69667	9,86	61,525	70,835	132,36
6	35	29,7	35,74	65,44	1,73667	0,93	2,67	5,45667	5,91333	11,37	55,9933	72,64	128,63
7	10	0,4925	0,3775	0,87	0,045	0,115	0,16	0,0975	0,0825	0,18	0,945	0,7625	1,71
7	15	1,7475	1,68	3,43	0,11	0,37	0,48	0,31	0,3	0,61	3,1425	2,7475	5,89
7	20	4,60333	6,89333	11,50	0,47	1,49333	1,96	1,16	1,93667	3,10	8,98667	14,1067	23,09
7	25	14,0475	21,9925	36,04	1,0075	1,935	2,94	2,55	3,6475	6,20	25,1967	49,6667	74,86
7	30	18,125	29,9825	48,11	1,17	1,6125	2,78	4,52333	5,79667	10,32	38,7225	63,8025	102,53
7	35	32,875	46,4875	79,36	2,2175	3,39	5,61	4,7875	8,175	12,96	71,7375	138,365	210,10
8	10	0,4875	0,4875	0,98	0,0625	0,1825	0,25	0,09	0,11	0,20	1,05	0,82	1,87
8	15	1,66	2,298	3,96	0,164	0,644	0,81	0,392	0,5	0,89	2,964	3,912	6,88
8	20	4,465	8,5575	13,02	0,5075	2,5075	3,02	0,9375	1,955	2,89	12,2175	24,4675	36,69
8	25	10,89	21,838	32,73	0,69	2,074	2,76	3,01	5,9225	8,93	22,984	46,886	69,87
8	30	14,85	29,836	44,69	1,328	3,048	4,38	4,068	10,046	14,11	36,702	82,222	118,92
8	35	36,548	105,94	142,49	2,55	6,362	8,91	6,618	17,658	24,28	55,75	149,36	205,11
Promedio		**11,98**	**15,65**	**27,63**	**0,84**	**1,18**	**2,03**	**2,36**	**3,05**	**5,41**	**23,48**	**33,84**	**57,32**

Figura 34: Tiempo de CPU, en segundos, de las cuatro variantes de nuestra heurística, aplicado a la familia de grafos ralos.

En el caso de la familia de los grafos ralos, comparando las figuras 33 y 34 con las 20 y 21 respectivamente, podemos apreciar que la variante de precómputo obtuvo una diferencia mayor que con los grafos densos. La diferencia de tiempos fue también mayor, por ejemplo terminando más de 12 segundos antes en promedio para la variante 4 (aproximadamente 20% más rápido).

6.5.2 Generación ponderada de ciclos

En esta variante modificamos la forma de generar los ciclos. En lugar de recorrer un grafo al azar sin tener en cuenta el cubrimiento actual, tomamos en cuenta la cantidad de ciclos que cubren al mismo en el momento de la generación, haciendo más probable la elección de ejes sin cubrir.

Durante el recorrido, en el momento de seleccionar el nodo vecino para continuar

armando el ciclo se le asignan “unidades” a cada eje de acuerdo con la cantidad de veces que el mismo fue cubierto. Para esto usamos la siguiente fórmula:

$$Ui = ceil\left(\frac{N}{2^{Ci}}\right)$$

Donde i es un eje adyacente, N es la cantidad de nodos del grafo, Ci es la cantidad de veces que el eje i está cubierto por un ciclo y Ui es la cantidad de unidades a asignar al eje i.

Cuando un eje no está cubierto, esta fórmula asigna N unidades a ese eje. Por cada vez que es cubierto, este valor se divide en dos. Si un eje está cubierto más de log2(N) veces, se le asigna una unidad.

Una vez calculadas las unidades que le corresponde a cada eje adyacente, la probabilidad de elección de cada uno de ellos es:

$$Pi = \frac{Ui}{U}$$

Donde Ui es la cantidad de unidades asignadas al eje i, U es la cantidad total de unidades asignadas a todos los ejes vecinos y Pi es la probabilidad de elegir al eje i.

En las figuras 35 y 36 mostramos los resultados de esta variante aplicados a la famila de los grafos densos.

Grafos		Porcentaje de desvío respecto a CPP			
K	N	Variante 1	Variante 2	Variante 3	Variante 4
4	10	1,43745	0,96324	1,43745	1,39271
4	15	1,26292	1,97645	2,21309	0,679961
4	20	4,60822	6,03315	5,31345	3,37172
4	25	4,65793	7,12964	5,52689	3,44221
4	30	5,85745	9,17903	7,20939	5,88333
4	35	2,59459	6,6614	4,45549	2,63014
5	10	1,43745	1,06542	1,7319	1,39271
5	15	0,475937	0,940306	0,880045	0,475937
5	20	3,09397	5,4826	4,0161	2,31145
5	25	3,09948	5,37847	4,48474	2,94058
5	30	4,49415	7,96523	4,972	4,04343
5	35	2,1783	6,65591	3,51163	1,74583
6	10	1,43745	0,783831	1,77664	1,39271
6	15	0,475937	0,698499	0,475937	0,291342
6	20	1,93349	4,75311	3,14521	1,95859
6	25	3,35616	4,9171	3,31907	2,58486
6	30	4,26144	8,56833	5,08976	3,50689
6	35	2,45828	6,35121	3,04743	1,70577
7	10	1,02068	1,06964	1,43745	1,39271
7	15	0,475937	1,67846	0,475937	0,35097
7	20	1,86821	4,18195	3,71253	1,84741
7	25	2,84438	5,40795	3,68307	2,48197
7	30	4,48553	9,01792	5,29333	4,11191
7	35	1,95703	6,99842	3,38477	1,59519
8	10	1,39271	0,506709	1,43745	1,39271
8	15	0,475937	1,70779	0,475937	0,139638
8	20	2,25164	4,80022	3,01413	1,30692
8	25	2,78753	7,15997	4,02115	2,35732
8	30	4,58897	8,19052	4,86308	3,67679
8	35	1,95943	7,48611	3,74217	1,27245
Promedio		**2,51**	**4,79**	**3,27**	**2,12**

Figura 35: Resultados obtenidos usando cálculo ponderado de ciclos , aplicada a la familia de los grafos densos. Los valores dados están en porcentaje sobre CPP.

Grafos		Tiempo de CPU en segundos											
		Variante 1			Variante 2			Variante 3			Variante 4		
K	N	Greedy	LS	Total	Greedy	LS	Total	Greedy	LS	Total	Greedy	LS	Total
4	10	2,918	0,214	3,13	0,214	0,05	0,26	0,63	0,062	0,69	5,776	0,536	6,31
4	15	9,136	0,93	10,07	0,772	0,184	0,96	2,484	0,27	2,75	22,268	2,286	24,55
4	20	26,664	4,276	30,94	1,948	0,616	2,56	5,1	0,802	5,90	47,962	7,866	55,83
4	25	53,698	9,662	63,36	2,728	0,9	3,63	9,658	1,738	11,40	96,814	17,4	114,21
4	30	56,948	11,546	68,49	7,976	2,336	10,31	12,564	2,622	15,19	119,324	24,2	143,52
4	35	114,388	23,43	137,82	6,724	1,704	8,43	21,024	4,408	25,43	164,698	33,734	198,43
5	10	2,848	0,406	3,25	0,206	0,094	0,30	0,642	0,088	0,73	5,574	0,866	6,44
5	15	7,17	1,476	8,65	0,83	0,554	1,38	1,792	0,368	2,16	15,814	3,106	18,92
5	20	22,532	7,98	30,51	1,762	1,408	3,17	5,226	1,806	7,03	51,222	17,842	69,06
5	25	40,922	17,052	57,97	3,594	2,918	6,51	8,774	3,684	12,46	82,492	34,06	116,55
5	30	55,24	26,956	82,20	3,192	2,628	5,82	8,802	4,186	12,99	118,406	57,494	175,90
5	35	86,278	44,042	130,32	5,72	4,352	10,07	21,2	10,77	31,97	179,232	93,954	273,19
6	10	2,63	0,584	3,21	0,198	0,13	0,33	0,58	0,142	0,72	5,182	1,132	6,31
6	15	7,404	2,356	9,76	0,794	0,766	1,56	1,926	0,588	2,51	13,672	4,006	17,68
6	20	22,736	12,798	35,53	1,296	1,858	3,15	5,27	2,952	8,22	38,94	21,842	60,78
6	25	35,334	24,894	60,23	2,832	4,274	7,11	8,852	6,134	14,99	80,398	55,002	135,40
6	30	57,84	50,07	107,91	3,696	5,996	9,69	12,23	10,278	22,51	138,224	119,072	257,30
6	35	107,66	98,808	206,47	6,652	8,842	15,49	24,47	22,574	47,04	193,892	180,052	373,94
7	10	3,158	0,916	4,07	0,27	0,23	0,50	0,618	0,17	0,79	5,732	1,682	7,41
7	15	7,348	3,018	10,37	0,652	1,028	1,68	1,85	0,698	2,55	15,482	6,568	22,05
7	20	19,968	15,166	35,13	1,69	3,568	5,26	4,544	3,56	8,10	51,49	40,228	91,72
7	25	41,156	41,148	82,30	1,938	5,018	6,96	8,084	8,242	16,33	73,128	74,928	148,06
7	30	61,824	77,248	139,07	3,76	9,628	13,39	11,352	13,648	25,00	106,972	131,776	238,75
7	35	89,598	120,772	210,37	5,556	14,462	20,02	22,186	29,668	51,85	154,472	215,972	370,44
8	10	2,624	0,92	3,54	0,23	0,28	0,51	0,584	0,208	0,79	6,28	2,224	8,50
8	15	8,088	4,374	12,46	0,778	1,66	2,44	1,694	0,838	2,53	15,466	8,098	23,56
8	20	23,638	23,824	47,46	1,42	4,11	5,53	3,862	3,904	7,77	61,532	61,848	123,38
8	25	49,468	63,804	113,27	2,308	7,596	9,90	9,672	12,87	22,54	79,308	107,426	186,73
8	30	67,276	109	176,28	3,392	12,2	15,59	12,502	20,226	32,73	113,936	189,862	303,80
8	35	102,378	186,776	289,15	5,156	16,9	22,06	19,08	34,476	53,56	161,056	289,6	450,66
Promedio		**39,63**	**32,81**	**72,44**	**2,61**	**3,88**	**6,49**	**8,24**	**6,73**	**14,97**	**74,16**	**60,16**	**134,31**

Figura 36: Tiempo de CPU, en segundos, de las cuatro variantes de nuestra heurística usando generación ponderada de ciclos, aplicado a la familia de grafos densos.

Comparando las figuras 35 con 17 podemos observar que las soluciones son mucho mejores para todas las variantes, obteniendo una diferencia significativa en la variante 2. Aunque los tiempos de computos, como puede verse al mirar las figuras 36 y 18, son mayores en todos los casos.

Grafos		Porcentaje de desvío respecto a CPP			
K	N	Variante 1	Variante 2	Variante 3	Variante 4
4	10	4,66045	4,66045	4,66045	4,66045
4	15				
4	20				
4	25				
4	30				
4	35				
5	10	10,092	6,27779	10,092	10,092
5	15	8,72021	8,72021	8,72021	8,72021
5	20	10,1866	5,20995	10,1866	6,92068
5	25	5,52217	6,51495	6,69545	5,21507
5	30	10,102	12,2806	10,3994	8,97901
5	35	8,52364	10,4486	8,97763	8,58908
6	10	1,18138	0,578744	1,18138	1,18138
6	15	8,88225	4,74422	9,17608	8,88225
6	20	5,54148	5,34307	6,83095	4,63601
6	25	5,85705	6,6872	5,99091	5,21728
6	30	5,82161	8,82401	7,75124	5,68975
6	35	5,7159	9,14929	6,17836	4,9442
7	10	0,645321	0,578744	0,645321	0,645321
7	15	8,57841	7,01798	8,67684	8,67684
7	20	3,96691	5,89813	5,5494	4,25919
7	25	5,0898	6,65863	6,40111	5,57001
7	30	4,40076	9,63961	5,56054	4,70133
7	35	4,36725	7,84943	6,02616	3,65641
8	10	0,645321	0,578744	0,645321	0,645321
8	15	3,78887	2,94533	4,92548	4,33058
8	20	8,58278	6,63698	8,80899	6,53676
8	25	4,6096	6,01002	5,83512	4,1749
8	30	5,47932	8,12673	5,92979	5,14634
8	35	3,81382	8,78121	5,18227	3,53846
Promedio		**5,79**	**6,41**	**6,44**	**5,42**

Figura 37: Resultados obtenidos usando cálculo ponderado de ciclos , aplicada a la familia de los grafos ralos. Los valores dados están en porcentaje sobre CPP.

Grafos		Tiempo de CPU en segundos											
		Variante 1			Variante 2			Variante 3			Variante 4		
K	N	Greedy	LS	Total	Greedy	LS	Total	Greedy	LS	Total	Greedy	LS	Total
4	10	2,36	0,19	2,55	0,15	0,03	0,18	0,4	0,06	0,46	4,34	0,3	4,64
4	15												
4	20												
4	25												
4	30												
4	35												
5	10	1,81	0,23667	2,05	0,12333	0,06667	0,19	0,45	0,04667	0,50	3,2	0,38333	3,58
5	15	4,69	0,655	5,35	0,41	0,09	0,50	1,21	0,205	1,42	8,165	1,24	9,41
5	20	14	4,95	18,95	1,19	0,94	2,13	2,59	0,91	3,50	58,39	20,26	78,65
5	25	20,465	6,86	27,33	1,65	0,8	2,45	4,02	1,345	5,37	49,405	16,16	65,57
5	30	64,32	15,885	80,21	3,11	1,595	4,71	6,815	1,675	8,49	106,885	28,495	135,38
5	35	49,345	20,61	69,96	4,645	2,88	7,53	15,02	6,635	21,66	152,125	73,185	225,31
6	10	1,5575	0,2575	1,82	0,1525	0,08	0,23	0,3175	0,0525	0,37	2,865	0,425	3,29
6	15	4,67	1,05667	5,73	0,64333	0,54	1,18	1,09	0,26	1,35	7,86333	2,09	9,95
6	20	19,94	9,005	28,95	0,77	0,935	1,71	4,33	2,005	6,34	33,5	14,81	48,31
6	25	30,52	17,25	47,77	3,52	4,02333	7,54	7,32667	4,08	11,41	44,1367	24,2567	68,39
6	30	34,6333	22,62	57,25	2,55667	2,51333	5,07	12,8067	8,59333	21,40	52,63	34,19	86,82
6	35	70,73	63,4067	134,14	4,76333	6,7	11,46	13,74	11,44	25,18	144,287	118,877	263,16
7	10	1,5375	0,34	1,88	0,14	0,1025	0,24	0,3075	0,075	0,38	2,73	0,6075	3,34
7	15	3,9275	1,515	5,44	0,3925	0,4375	0,83	0,8	0,345	1,15	8,1725	2,8	10,97
7	20	15,6133	10,6033	26,22	1,21	2,55667	3,77	3,18667	2,17333	5,36	23,8433	16,3767	40,22
7	25	43,075	35,445	78,52	1,515	3,1025	4,62	5,1025	4,575	9,68	46,8425	42,9975	89,84
7	30	38,1225	42,5425	80,67	3,845	7,6725	11,52	7,87	8,8875	16,76	84,765	93,4675	178,23
7	35	66,795	90,6925	157,49	5,33	13,5725	18,90	11,4875	16,05	27,54	128,62	171,74	300,36
8	10	1,475	0,46	1,94	0,1325	0,1275	0,26	0,3075	0,115	0,42	2,7325	0,8675	3,60
8	15	3,896	2,422	6,32	0,382	0,71	1,09	0,804	0,58	1,38	6,886	3,874	10,76
8	20	16,3325	14,51	30,84	0,7775	1,935	2,71	2,9825	2,4725	5,46	20,6575	16,675	37,33
8	25	22,95	22,984	45,93	2,16	5,396	7,56	5,046	5,128	10,17	43,806	44,136	87,94
8	30	33,268	49,982	83,25	2,856	9,376	12,23	8,176	12,278	20,45	78,162	120,934	199,10
8	35	70,306	131,808	202,11	4,272	16,298	20,57	15,014	26,772	41,79	115,404	212,85	328,25
Promedio		**25,45**	**22,65**	**48,11**	**1,87**	**3,30**	**5,17**	**5,25**	**4,67**	**9,92**	**49,22**	**42,48**	**91,70**

Figura 38: Tiempo de CPU, en segundos, de las cuatro variantes de nuestra heurística usando generación ponderada de ciclos, aplicado a la familia de grafos ralos.

En el caso de los grafos ralos, como puede verse al comparar las figuras 37 y 38 con 20 y 21 respectivamente, no encontramos una ganancia significativa en la calidad de las soluciones, aunque hay pequeñas mejoras en todos los casos. Respecto al tiempo de cómputo, la versión del programa ponderada fue mucho más costosa.

6.5.3 Verificación del tamaño del ciclo a optimizar

Esta variante consiste en verificar que la cantidad de ejes del ciclo no sea igual a K antes de intentar mejorarlo en la fase de búsqueda local, ya que en ese caso sería imposible reemplazar ninguno de sus ejes por un camino de menor costo si el reemplazo se realiza eje por eje. Esto permite evitar intentar operaciones de cómputo de camino mínimo cuando se puede garantizar que no se van a obtener mejoras.

Para esta versión del programa no presentamos el error relativo sobre CPP, puesto que

los resultados son iguales a los de la versión original. A continuación, en las tablas 39 y 40, presentamos los tiempos de CPU empleados por esta versión para las dos familias de grafos.

Grafos		Tiempo de CPU en segundos											
		Variante 1			Variante 2			Variante 3			Variante 4		
K	N	Greedy	LS	Total	Greedy	LS	Total	Greedy	LS	Total	Greedy	LS	Total
4	10	1,83	0,218	2,05	0,168	0,064	0,23	0,398	0,054	0,45	4,326	0,536	4,86
4	15	5,696	0,954	6,65	0,6	0,216	0,82	1,584	0,278	1,86	12,17	1,964	14,13
4	20	18,002	3,868	21,87	1,05	0,258	1,31	3,262	0,652	3,91	43,504	9,31	52,81
4	25	25,546	5,894	31,44	1,92	0,394	2,31	6,008	1,35	7,36	49,54	11,46	61,00
4	30	46,092	10,812	56,90	2,472	0,38	2,85	11,458	2,656	14,11	89,264	20,842	110,11
4	35	50,736	10,71	61,45	3,516	0,396	3,91	16,214	3,56	19,77	153,888	32,424	186,31
5	10	2,074	0,536	2,61	0,2	0,116	0,32	0,398	0,114	0,51	4,134	0,974	5,11
5	15	7,016	2,344	9,36	0,692	0,542	1,23	1,378	0,456	1,83	12,216	3,994	16,21
5	20	18,492	8,958	27,45	0,99	0,708	1,70	2,564	1,242	3,81	33,122	16,674	49,80
5	25	25,018	14,402	39,42	1,68	1,034	2,71	5,904	3,306	9,21	63,382	36,348	99,73
5	30	37,374	23,29	60,66	2,67	1,32	3,99	7,156	4,45	11,61	95,282	58,52	153,80
5	35	67,426	39,302	106,73	3,936	1,04	4,98	15,31	9,156	24,47	105,726	62,332	168,06
6	10	2,062	0,78	2,84	0,216	0,244	0,46	0,374	0,14	0,51	3,552	1,11	4,66
6	15	5,982	3,144	9,13	0,47	0,702	1,17	1,522	0,802	2,32	14,014	7,268	21,28
6	20	12,536	10,344	22,88	1,13	1,468	2,60	4,418	3,624	8,04	32,79	26,578	59,37
6	25	32,072	32,42	64,49	1,872	2,184	4,06	5,116	5,21	10,33	54,334	56,574	110,91
6	30	50,218	55,018	105,24	2,444	2,54	4,98	9,192	10,062	19,25	105,256	117,582	222,84
6	35	48,102	53,958	102,06	4,65	3,432	8,08	14,576	16,8	31,38	143,226	157,63	300,86
7	10	1,778	0,75	2,53	0,176	0,278	0,45	0,426	0,222	0,65	3,522	1,512	5,03
7	15	5,598	3,972	9,57	0,38	0,83	1,21	1,506	1,04	2,55	10,858	7,652	18,51
7	20	18,904	22,144	41,05	1,246	2,78	4,03	4,046	4,784	8,83	27,592	32,262	59,85
7	25	29,51	43,948	73,46	1,828	3,892	5,72	5,71	8,736	14,45	46,47	69,318	115,79
7	30	35,842	58,056	93,90	2,784	5,12	7,90	10,312	17,186	27,50	137,574	223,172	360,75
7	35	64,874	113,58	178,45	4,87	5,308	10,18	11,17	19,39	30,56	122,744	210,906	333,65
8	10	1,926	1,082	3,01	0,184	0,324	0,51	0,4	0,258	0,66	4,134	2,516	6,65
8	15	7,884	7,326	15,21	0,434	1,282	1,72	1,39	1,274	2,66	11,636	10,866	22,50
8	20	18,032	27,676	45,71	1,062	3,354	4,42	3,316	5	8,32	32,652	47,606	80,26
8	25	28,118	56,318	84,44	1,978	6,404	8,38	6,886	14,012	20,90	59,964	120,116	180,08
8	30	47,422	102,98	150,40	2,96	7,306	10,27	9,898	22,036	31,93	86,242	187,644	273,89
8	35	69,198	163,126	232,32	4,032	8,446	12,48	12,382	28,562	40,94	169,036	399,25	568,29
Promedio		**26,18**	**29,26**	**55,44**	**1,75**	**2,08**	**3,83**	**5,81**	**6,21**	**12,02**	**57,74**	**64,50**	**122,24**

Figura 39: Tiempo de CPU, en segundos, de las cuatro variantes de nuestra heurística verficando el largo de los ciclos en la búsqueda local, aplicado a la familia de grafos densos.

Grafos		Tiempo de CPU en segundos											
		Variante 1			Variante 2			Variante 3			Variante 4		
K	N	Greedy	LS	Total	Greedy	LS	Total	Greedy	LS	Total	Greedy	LS	Total
4	10	1,24	0,15	1,39	0,09	0,02	0,11	0,28	0,06	0,34	2,41	0,35	2,76
4	15												
4	20												
4	25												
4	30												
4	35												
5	10	1,11667	0,21	1,33	0,13	0,06667	0,20	0,28667	0,04	0,33	2,19	0,44	2,63
5	15	2,785	0,615	3,40	0,245	0,085	0,33	0,655	0,12	0,78	5,53	1,165	6,70
5	20	14,05	6,25	20,30	0,46	0,26	0,72	1,3	0,56	1,86	19,67	8,56	28,23
5	25	11,555	3,77	15,33	1,465	0,295	1,76	2,915	0,8	3,72	32,26	10,315	42,58
5	30	30,34	7,91	38,25	3,79	0,645	4,44	9,22	1,9	11,12	92,41	20,305	112,72
5	35	33,965	13,63	47,60	2,65	0,76	3,41	13,21	5,51	18,72	88,19	37,64	125,83
6	10	0,985	0,27	1,26	0,065	0,0875	0,15	0,2575	0,0575	0,32	2,3575	0,595	2,95
6	15	2,99333	1,14333	4,14	0,34	0,33	0,67	0,64	0,18667	0,83	6,98	2,84	9,82
6	20	11,025	6,49	17,52	0,895	0,99	1,89	1,95	1,335	3,29	25,74	14,365	40,11
6	25	20,8133	13,0733	33,89	1,60333	1,11	2,71	3,18333	2,18333	5,37	44,1967	22,65	66,85
6	30	28,4767	17,3933	45,87	2,00667	0,97667	2,98	6,95	4,08	11,03	68,16	47,96	116,12
6	35	42,8767	41,79	84,67	1,80667	1,41667	3,22	8,7	7,73667	16,44	122,777	112,387	235,16
7	10	0,9875	0,33	1,32	0,0725	0,125	0,20	0,205	0,08	0,29	1,9	0,705	2,61
7	15	2,49	1,2875	3,78	0,3	0,425	0,73	0,6375	0,2825	0,92	4,9825	2,4325	7,42
7	20	6,50333	6,38	12,88	0,69	1,41333	2,10	1,76667	1,68	3,45	13,9333	14,1467	28,08
7	25	16,3925	16,51	32,90	1,32	2,4275	3,75	4,9125	5,6575	10,57	30,41	36,73	67,14
7	30	34,9725	42,425	77,40	1,8675	2,7225	4,59	6,4775	8,57	15,05	51,04	62,79	113,83
7	35	50,66	78,8875	129,55	2,7175	3,6975	6,42	7,75	10,5275	18,28	89,145	137,75	226,90
8	10	0,9825	0,475	1,46	0,1275	0,155	0,28	0,21	0,1025	0,31	1,9075	0,9075	2,82
8	15	2,49	1,818	4,31	0,242	0,636	0,88	0,638	0,498	1,14	5,718	4,606	10,32
8	20	8,4175	9,6475	18,07	0,53	1,3425	1,87	1,4475	1,7075	3,16	13,83	16,155	29,99
8	25	20,574	32,908	53,48	1,182	2,478	3,66	3,366	4,568	7,93	37,42	52,362	89,78
8	30	30,162	54,496	84,66	1,848	4,144	5,99	6,432	10,656	17,09	52,172	94,52	146,69
8	35	37,242	75,204	112,45	2,314	5,1	7,41	6,798	13,342	20,14	73,044	152,584	225,63
Promedio		**16,56**	**17,32**	**33,89**	**1,15**	**1,27**	**2,42**	**3,61**	**3,29**	**6,90**	**35,53**	**34,21**	**69,75**

Figura 40: Tiempo de CPU, en segundos, de las cuatro variantes de nuestra heurística verficando el largo de los ciclos en la búsqueda local, aplicado a la familia de grafos ralos.

Al comparar con las tablas 18 y 21 podemos apreciar que los tiempos de CPU en ambos casos se mantienen similares, no obteniendo una ganancia significativa con este intento de optimización. Esto se debe a que, como se puede ver en la tabla 41, en la mayoría de los casos el cálculo de camino mínimo se realiza de todas formas, mientras que es necesario hacer la verificación de la condición para todos los posibles ciclos.

	hits	misses	%hits	hits	misses	%hits	hits	misses	%hits	hits	misses	%hits	hits	misses	%hits
10,1	2325	4955	31,94	315	6474	4,64	112	7109	1,55	54	6846	0,78	6	7749	0,08
10,2	1786	5880	23,30	477	7070	6,32	590	5957	9,01	343	6143	5,29	193	6307	2,97
10,3	456	7073	6,06	295	7511	3,78	156	7647	2,00	48	7705	0,62	6	8176	0,07
10,4	816	10791	7,03	546	11243	4,63	312	12872	2,37	65	10781	0,60	17	12131	0,14
10,5	366	5703	6,03	706	5322	11,71	366	5680	6,05	155	5996	2,52	17	6229	0,27
15,1	2435	16442	12,90	790	7786	9,21	308	8611	3,45	247	9832	2,45	139	8938	1,53
15,2	2253	4763	32,11	1313	7190	15,44	1075	8019	11,82	335	9266	3,49	163	8957	1,79
15,3	3398	12834	20,93	1464	14223	9,33	755	17136	4,22	676	29399	2,25	307	19409	1,56
15,4	2177	14112	13,36	1458	15723	8,49	1191	20558	5,48	506	16063	3,05	722	34187	2,07
15,5	3395	9397	26,54	1395	12253	10,22	737	16221	4,35	511	15205	3,25	305	13992	2,13
20,1	4955	25256	16,40	1460	19181	7,07	701	18723	3,61	414	20804	1,95	642	48794	1,30
20,2	8056	19552	29,18	4148	33677	10,97	1868	30009	5,86	790	19451	3,90	1055	40496	2,54
20,3	4820	22108	17,90	1804	15207	10,60	1484	30176	4,69	1044	41159	2,47	489	30293	1,59
20,4	4001	22736	14,96	2004	21404	8,56	2129	38336	5,26	2476	71390	3,35	388	19540	1,95
20,5	8581	23423	26,81	2091	19335	9,76	1339	23377	5,42	942	25640	3,54	1272	41444	2,98
25,1	12149	37828	24,31	4857	42668	10,22	3287	59165	5,26	1137	34412	3,20	596	29677	1,97
25,2	10282	41443	19,88	4998	32706	13,26	2804	54262	4,91	1163	46109	2,46	376	25660	1,44
25,3	3779	16426	18,70	1628	18513	8,08	798	18926	4,05	1317	60569	2,13	505	34042	1,46
25,4	8260	22383	26,96	2932	25449	10,33	1836	38958	4,50	1231	48253	2,49	730	42667	1,68
25,5	5713	18274	23,82	2971	26121	10,21	1290	29591	4,18	1296	48589	2,60	634	40685	1,53
30,1	9645	21653	30,82	4662	30610	13,22	1975	26005	7,06	3312	73873	4,29	1198	42354	2,75
30,2	6420	21318	23,15	4051	36161	10,07	2261	47850	4,51	1086	39357	2,69	1136	74104	1,51
30,3	8914	33813	20,86	2877	25650	10,09	2081	42897	4,63	914	36768	2,43	355	31702	1,11
30,4	11373	37459	23,29	2086	19610	9,61	1540	37036	3,99	1206	52105	2,26	891	50194	1,74
30,5	5993	23364	20,41	4260	45635	8,54	1630	37143	4,20	1532	55831	2,67	619	33869	1,79
35,1	15217	39504	27,81	2583	22116	10,46	1904	31828	5,64	1836	52385	3,39	666	30871	2,11
35,2	11326	31410	26,50	5961	29405	16,86	4902	42878	10,26	2403	53414	4,31	2855	41350	6,46
35,3	12490	44560	21,89	4338	39701	9,85	1493	26313	5,37	853	26885	3,08	483	25349	1,87
35,4	9590	30362	24,00	3123	34372	8,33	925	18954	4,65	1357	54249	2,44	912	49860	1,80
35,5	5631	16331	25,64	3935	26033	13,13	1787	29225	5,76	2349	76568	2,98	2239	98761	2,22

Figura 41: Cantidad de veces que fue evitado el cálculo de un camino mínimo para mejorar una solución por tener el ciclo el tamaño máximo (hits) contra cantidad de ejecuciones del algoritmo de camino mínimo (misses). Para cada par de valores se presenta el porcentaje de hits sobre el total de verificaciones, es decir el total de veces que se hubiera buscado un camino alternativo en la versión original. Estos cálculos fueron ejecutados sobre la variante 1 de HBCCP.

7 Conclusiones

En este documento presentamos una heurística para la obtención de soluciones aproximadas al Bounded Cycle Cover Problem. Nuestra heurística es similar a GRASP, generando varias soluciones de forma golosa con un componente aleatorio y conservando la mejor solución obtenida.

Una ventaja importante es la posibilidad de variar los parámetros del algoritmo dependiendo del uso que se le quiere dar: si se aumenta el número de iteraciones o la cantidad de ciclos, se pueden obtener mejores soluciones, a un costo de cómputo mayor.

Estudiando estos parámetros llegamos a la conclusión de que aumentar la cantidad de iteraciones nunca empeora las soluciones obtenidas, sin embargo a partir de determinado punto la mejora no es significativa en relación con el aumento del tiempo de cómputo. Por otro lado, incrementar el número de ciclos candidatos excesivamente puede empeorar la calidad de las soluciones, ya que el algoritmo puede volverse demasiado goloso, por lo tanto es importante lograr un balance adecuado con este parámetro para evaluar posibles ciclos sin caer en soluciones extremádamente golosas.

En base a pruebas realizadas, definimos cuatro variantes que pueden utilizarse para distintas situaciones y que obtienen soluciones de una calidad razonable para el tiempo de cómputo invertido.

Para evaluar la performance de la heurística fue importante poder contar con los grafos utilizados en [8] y las soluciones obtenidas, ya que nos permitió establecer un marco de referencia con el cual comparar la calidad de nuestras soluciones.

Por otra parte, el poder contar con la implementación de un programa para resolver el problema del Cartero Chino [1], que constituye una cota inferior de nuestro problema, nos permitió calcular el error reativo entre nuestras soluciones y CPP. Dado que los resultados en [8] se presentaron como error relativo con respecto a CPP, realizar este cálculo fue imprescindible para la comparación de las soluciones.

Comparado con las heurísticas presentadas en [8] que aproximan soluciones para el mismo problema nuestro algoritmo generalmente consigue soluciones de mejor calidad, a costa de un mayor tiempo de cómputo. En comparación con los resultados presentados en [11], HBCCP es considerablemente más rápida, pero las soluciones en muchos de los casos son de menor calidad.

Analizando los resultados notamos que el tiempo necesario para obtener las soluciones crece significativamente a medida que crece K, aparentemente porque la búsqueda local pasa mucho más tiempo calculando ciclos alternativos para intentar mejorar la solución.

Los resultados de los experimentos realizados en la subsección 6.5 con la intención de explorar posibles mejoras a HBCCP nos llevan a pensar que tanto la calidad de las soluciones como el tiempo de CPU podrían mejorarse realizando algunos ajustes. En la modificación en la que realizamos el precómputo de los ciclos los tiempos y calidades tal vez podrían mejorarse ajustando la cantidad de ciclos pregenerados, ya que en los experimentos realizados no hicimos una calibración precisa de este parámetro. Para reducir el tiempo de cómputo sería conveniente realizar una modificación en las estructuras de datos usadas en el programa para tomar ventaja de las nuevas variantes.

8 Trabajos futuros

A continuación presentaremos posibles variantes basadas en nuestro trabajo que podrían aplicarse para el tratamiento de BCCP.

Una posible modificación para mejorar HBCCP consiste en generar un conjunto de ciclos a usar como candidatos en una etapa inicial, separada del resto de los cálculos. Esta modificación, tal como la probamos en la subsección 6.5.1, permite ahorrar una cantidad moderada de tiempo de CPU, debido a que no se generan los mismos ciclos candidatos una y otra vez. Como trabajo futuro sería conveniente determinar la mejor cantidad de ciclos a precalcular, de forma tal que las soluciones tengan una calidad razonable sin malgastar tiempo de cómputo.

El hecho de precalcular los ciclos también daría la posibilidad de seleccionar un porcentaje de los mismos en lugar de estar obligados a elegir una cantidad fija de ciclos a evaluar, simplificando el proceso de calibración de parámetros presentada en la subsección 6.2.

Otra posibilidad es utilizar el conjunto de ciclos precalculados mencionado anteriormente como vecindad y aplicar GRASP [3]. Al conocer el tamaño de la misma en cada iteración de la fase greedy la aplicación de GRASP es trivial.

Adicionalmente, para pregenerar los ciclos se podría utilizar alguno de los métodos utilizados en [8], como Greedy Postman o Cycle Basis. No es posible usar estos métodos con la versión actual de HBCCP porque son adecuados para generar cubrimientos de todos los ejes de un grafo, mientras que la generación en HBCCP se realiza eje por eje.

Una mejora importante para la fase de búsqueda local de HBCCP consiste en la paralelización del cómputo de los caminos alternativos. Debido a que el algoritmo de Distance Vector es, en realidad, un algoritmo de ruteo distribuido usado en redes de computadoras, se lo puede paralelizar muy fácilmente emulando el comportamiento de un protocolo de ruteo que lo implementa, como por ejemplo RIP. El mismo se encuentra especificado en [4]. Esta especificación define exactamente cómo debe actuar cada nodo y los mensajes que deben ser pasados entre ellos, necesitando solamente un mecanismo de sincronizacion, que es lo único que no está incluído en el algoritmo de ruteo por no ser necesario en ese caso.

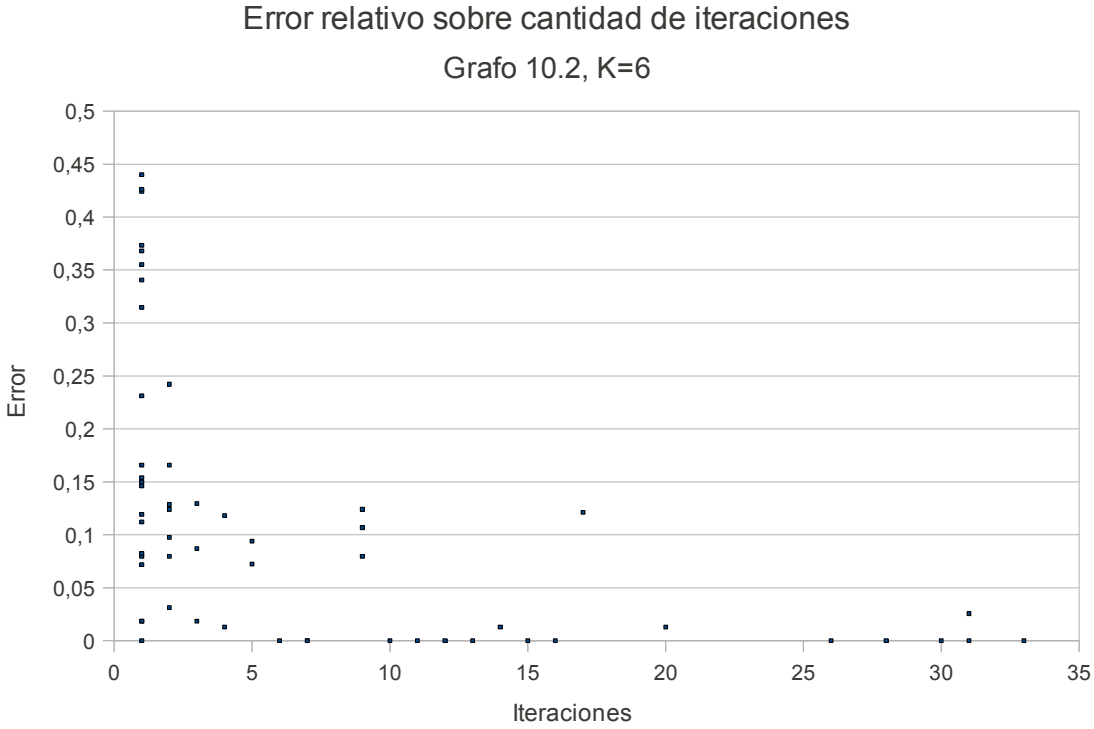
Otra posible mejora para la búsqueda local consiste en, durante el calculo de un camino alternativo, permitir que este tenga nodos en comun con el ciclo al que pertenece el eje a reemplazar, siempre que no haya coincidencia de ejes. El ciclo luego debe ser dividido en los nodos en los que el camino alternativo y el ciclo original se crucen.

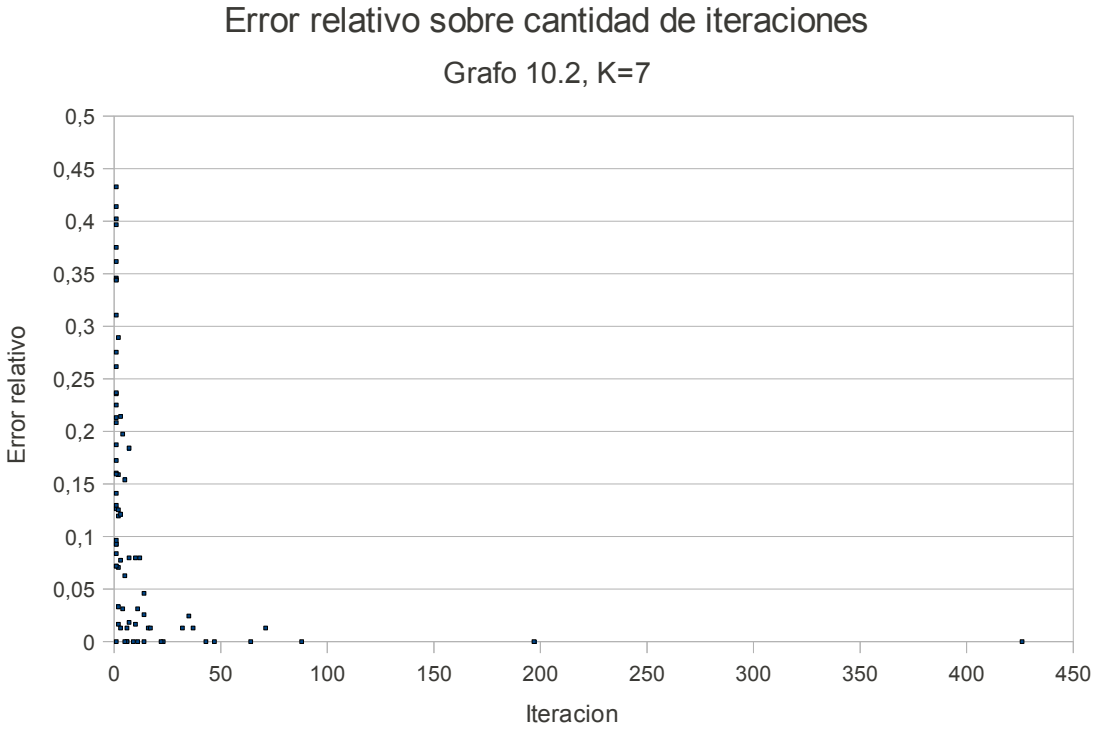
Como trabajos futuros también podrían implementarse las mejoras presentadas en la subsección 6.5 usando estructuras de datos optimizadas para dichas variantes y eventualmente combinarlas.

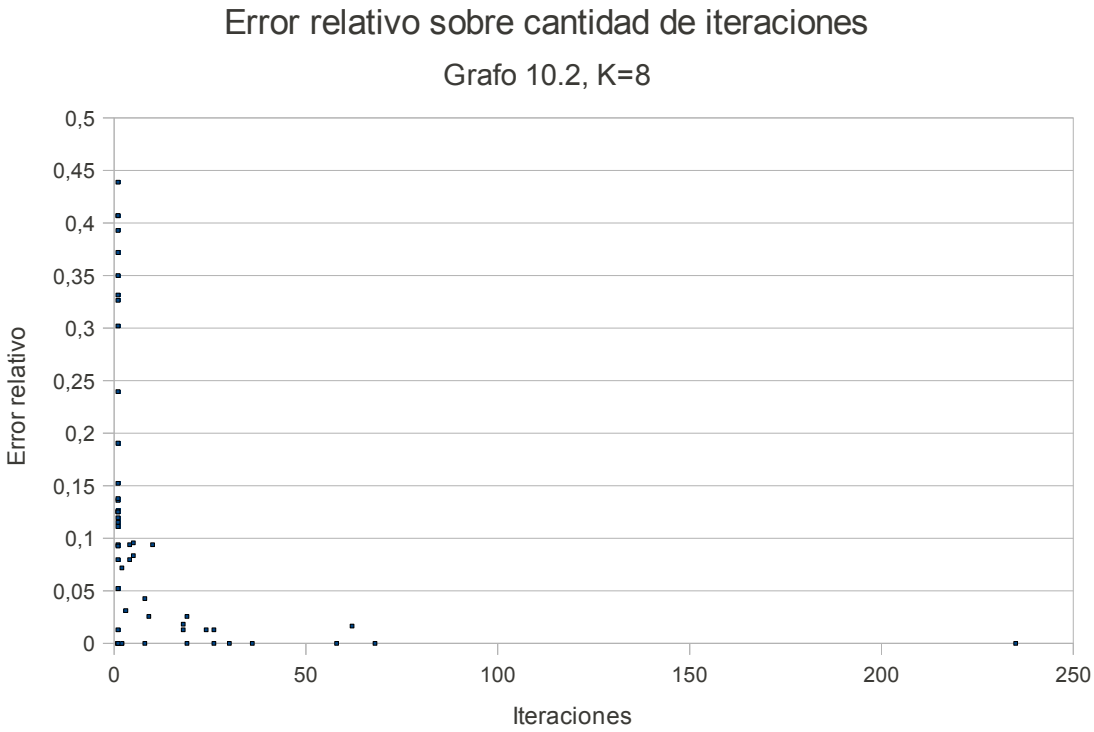
Una mejora en la búsqueda local podría consistir en sustituir Distance Vector, que es una variante de Bellman-Ford, por el algoritmo de Dijkstra para camino mínimo. Para ello, se necesitaría implementar un mecanismo para restringir los posibles ejes a elegir de forma tal que el largo de los caminos obtenidos cumplan con la cota impuesta por BCCP.

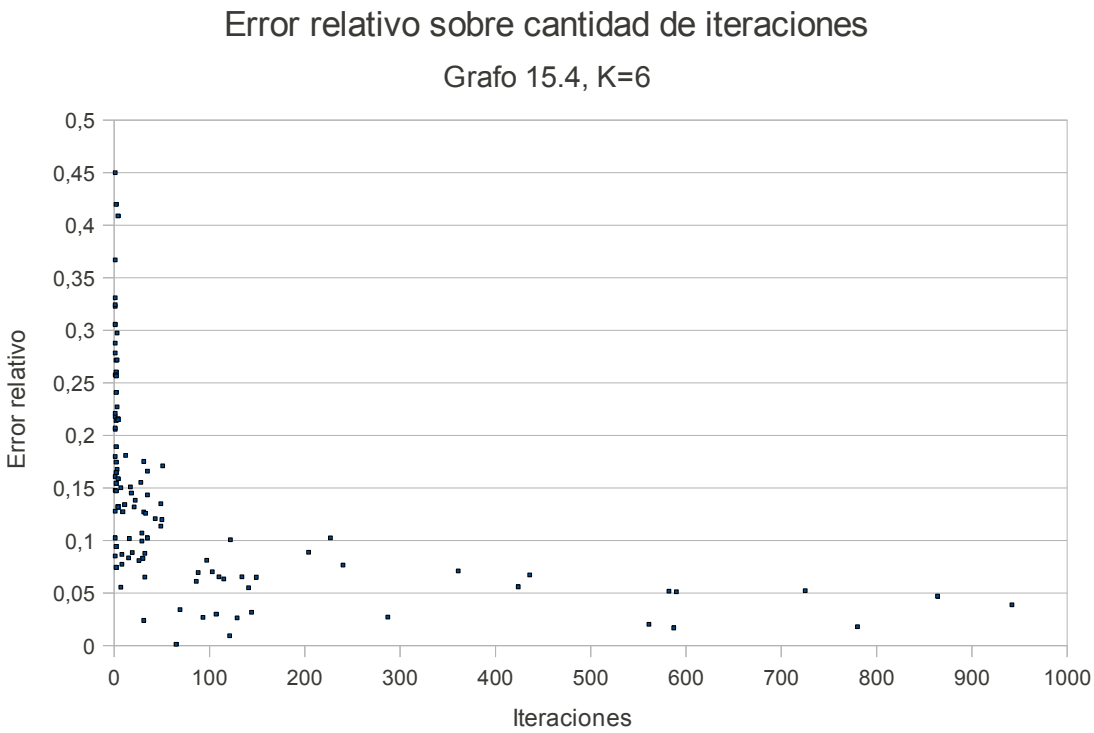
9 Apéndice: Resultados

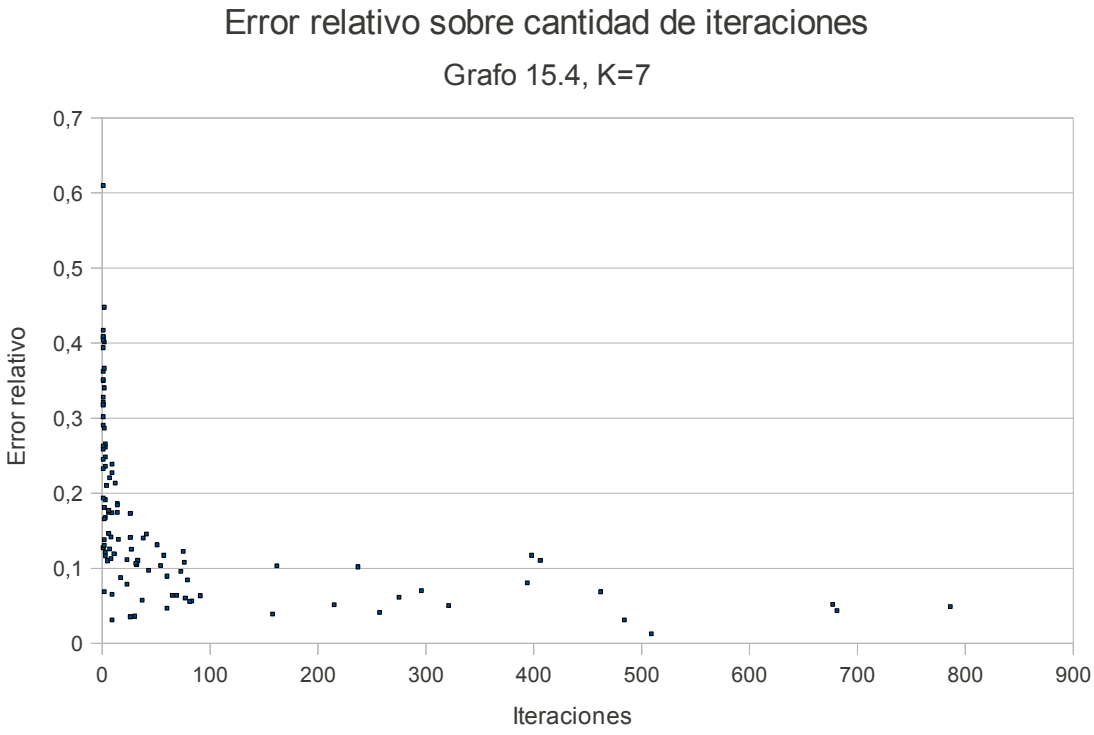
En esta sección presentamos la serie completa de resultados de las que se obtuvieron las figuras 9 y 10, presentadas en la subsección 6.2.

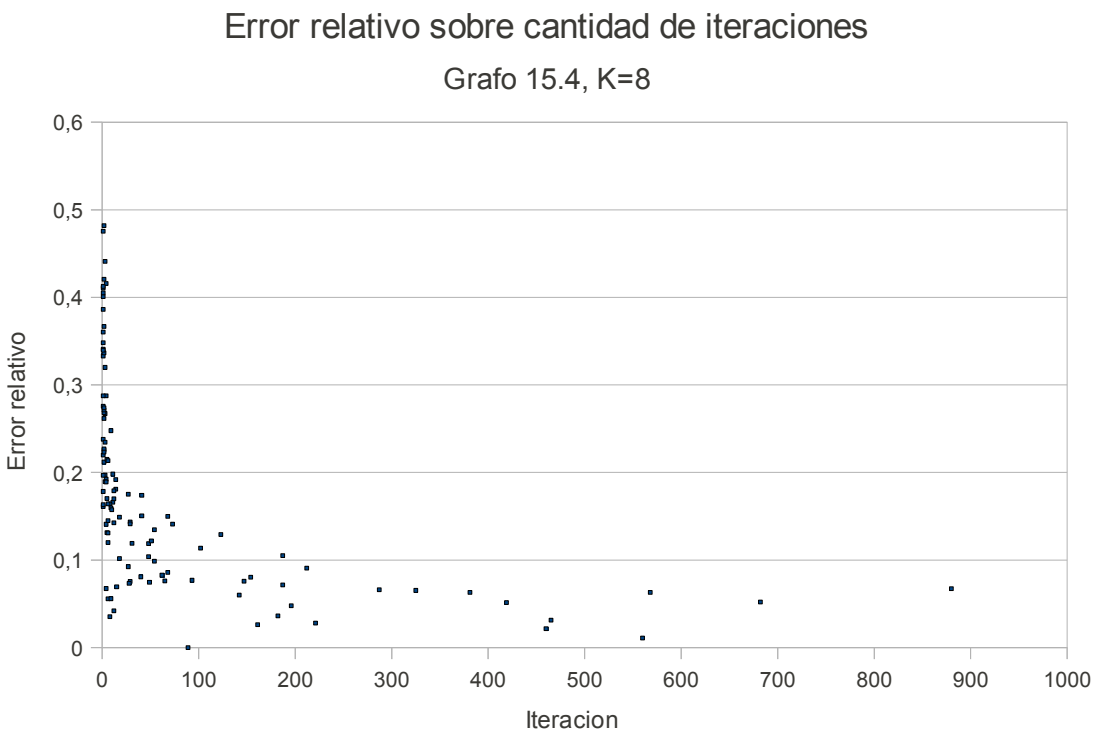


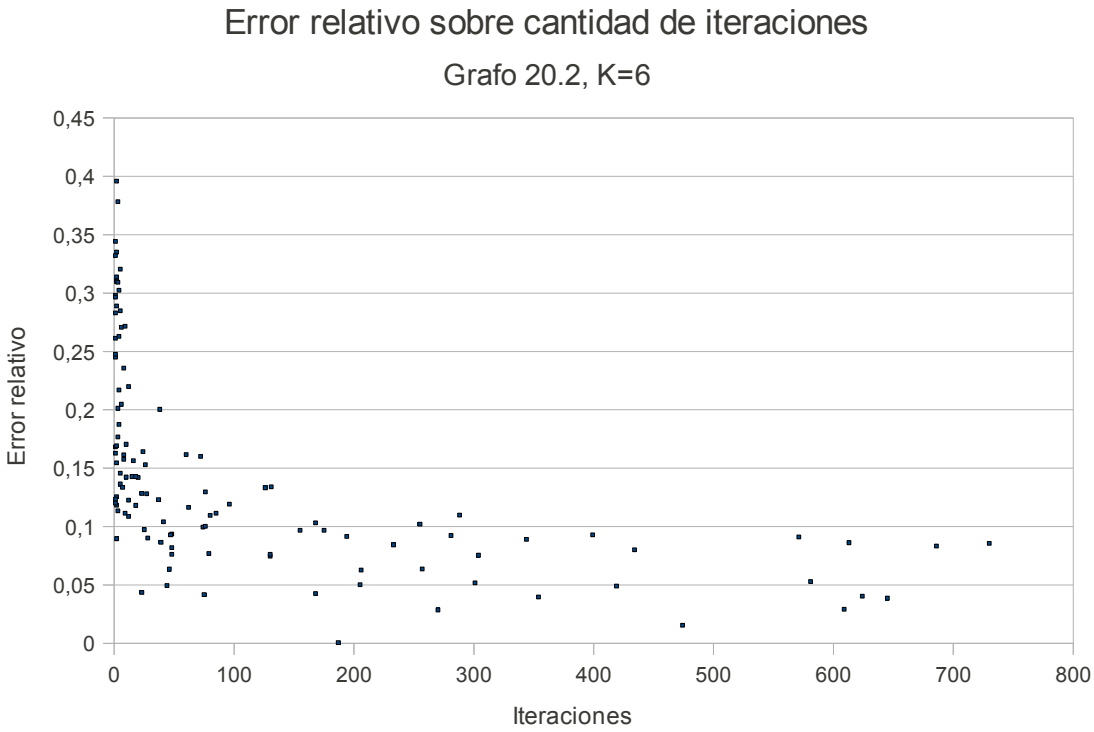


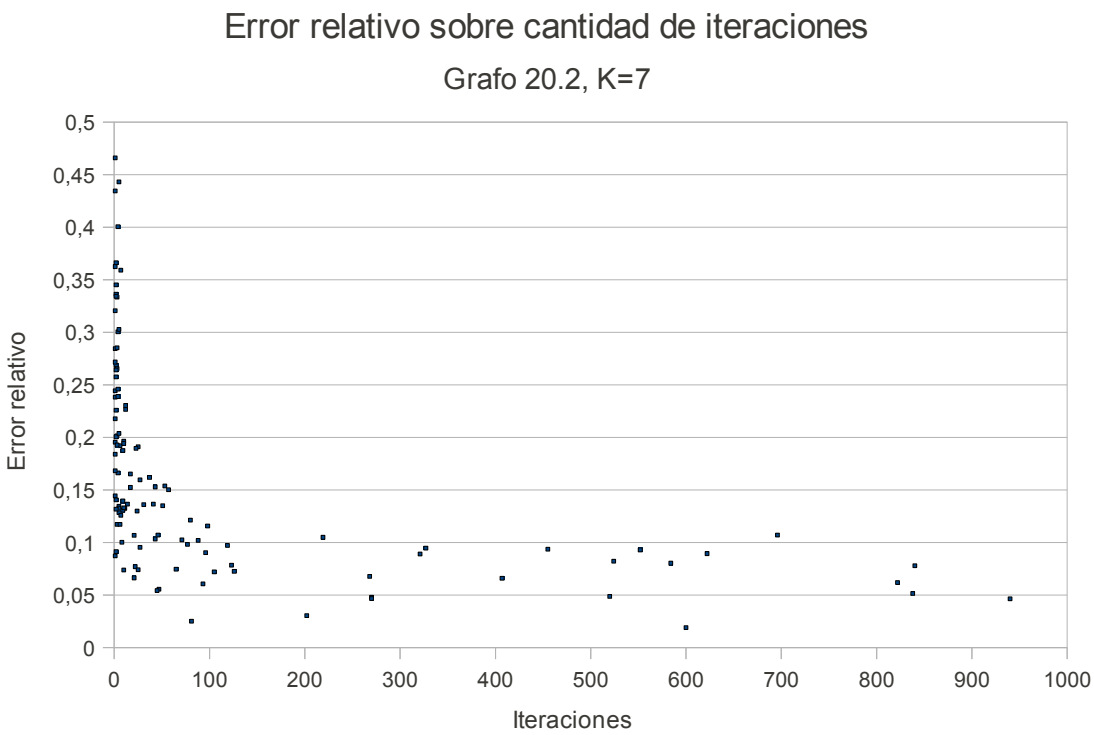

Error relativo sobre cantidad de iteraciones
Grafo 10.2, K=8
Error relativo
0,5
0,45
0,4
0,35
0,3
0,25
0,2
0,15
0,1
0,05
0
0
50
100
150
200
250
Iteraciones

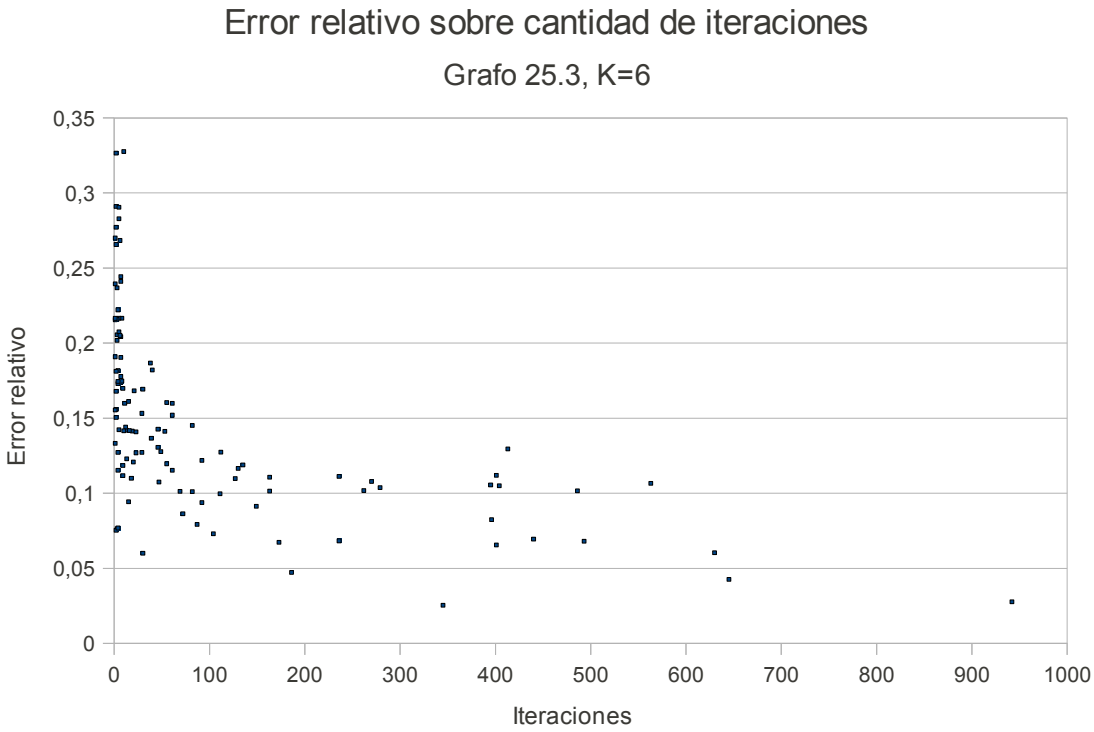

Error relativo sobre cantidad de iteraciones
Grafo 15.4, K=6
Error relativo
0,5
0,45
0,4
0,35
0,3
0,25
0,2
0,15
0,1
0,05
0
0
100
200
300
400
500
600
700
800
900
1000
Iteraciones

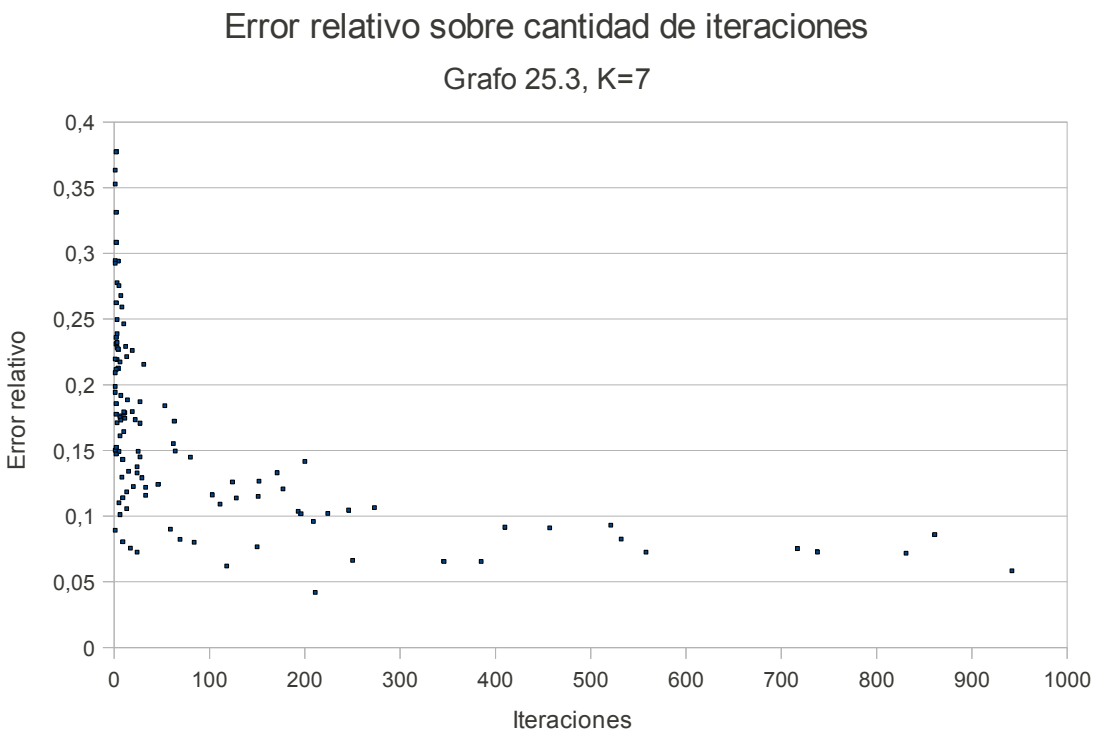

Error relativo sobre cantidad de iteraciones
Grafo 15.4, K=7
Error relativo
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
0
100
200
300
400
500
600
700
800
900
Iteraciones

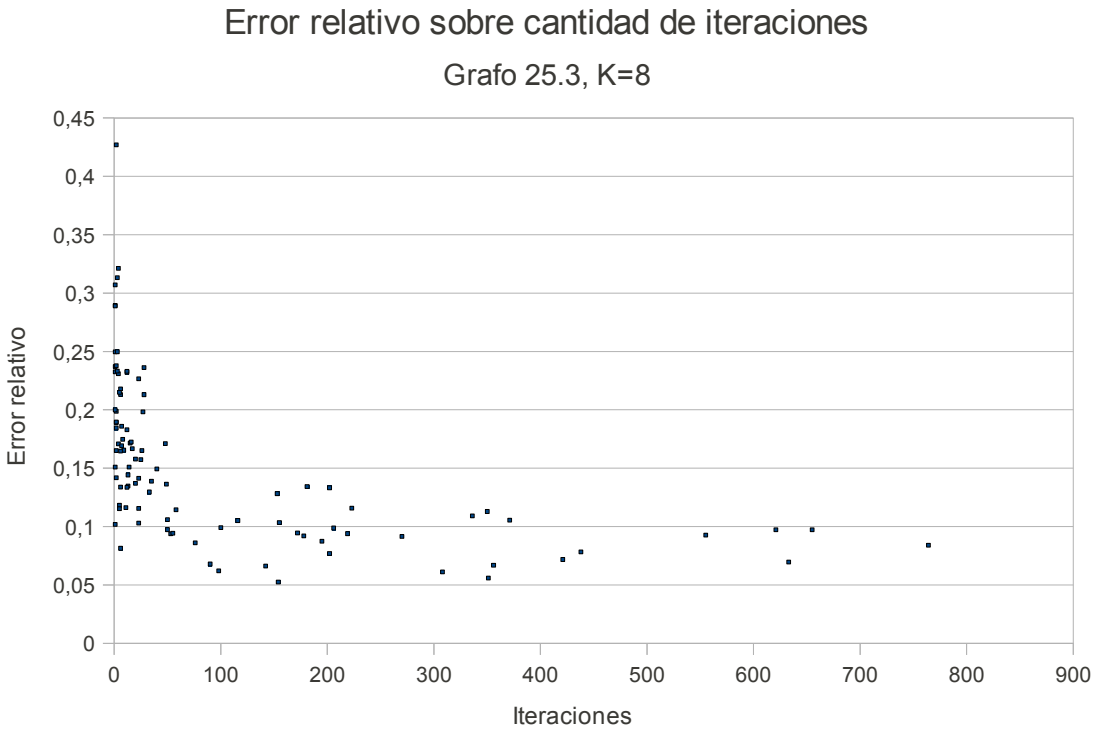

Error relativo sobre cantidad de iteraciones
Grafo 15.4, K=8
Error relativo
0,6
0,5
0,4
0,3
0,2
0,1
0
0
100
200
300
400
500
600
700
800
900
1000
Iteracion


Error relativo sobre cantidad de iteraciones
Grafo 20.2, K=6
Error relativo
0,45
0,4
0,35
0,3
0,25
0,2
0,15
0,1
0,05
0
0
100
200
300
400
500
600
700
800
Iteraciones


Error relativo sobre cantidad de iteraciones
Grafo 20.2, K=7
Error relativo
0,5
0,45
0,4
0,35
0,3
0,25
0,2
0,15
0,1
0,05
0
0
100
200
300
400
500
600
700
800
900
1000
Iteraciones


Error relativo sobre cantidad de iteraciones
Grafo 25.3, K=6
Error relativo
0,35
0,3
0,25
0,2
0,15
0,1
0,05
0
0
100
200
300
400
500
600
700
800
900
1000
Iteraciones


Error relativo sobre cantidad de iteraciones
Grafo 25.3, K=7
Error relativo
0,4
0,35
0,3
0,25
0,2
0,15
0,1
0,05
0
0
100
200
300
400
500
600
700
800
900
1000
Iteraciones


Error relativo sobre cantidad de iteraciones
Grafo 25.3, K=8
Error relativo
0,45
0,4
0,35
0,3
0,25
0,2
0,15
0,1
0,05
0
0
100
200
300
400
500
600
700
800
900
Iteraciones

10 Bibliografía

[1] D. Battré. Chinese Postman Problem, Technical University of Berlin. http://penguin.ewu.edu/cscd320/Topic/Graph/AllPaths/ChinesePostman/engl_start.htm

[2] J. Edmonds, "The Chinese postman problem", Operations Research 13 Suppl. 1 (1965) 373.

[3] T. Feo y M. Resende. Greedy Randomized Adaptive Search Procedures. Journal of Global Optimization, 6, 109–134, 1995.

[4] C. Hedrick. Routing Information Protocol. RFC-1058, Junio 1988.

[5] J. Horton. A polynomial-time algorithm to find the shortest cycle basis of a graph. SIAM Journal on Computing, 16(2):358-366, 1987.

[6] H. Liu y J. Wang. A new way to enumerate cycles in graph. Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services, 2006.

[7] K. Mei-Ko, "Graphic programming using odd or even points", Chinese Mathematics 1 (1962) 273-277.

[8] E. Olinick y D. Hochbaum. The Bounded Cycle-Cover Problem. INFORMS Journal on computing, Vol. 13, No 2, 2001.

[9] C. Papadimitriou. On the complexity of edge traversing. Journal of the ACM, 23(3):544–554, 1976.

[10] L. Peterson y B. Davie. Computer Networks A System Approach (Second Edition) pp 132-133, 2000.

[11] G. Picardi. Un algoritmo Tabú Search para el cubrimiento de Redes de Comunicación con ciclos acotados. Tesis de Licenciatura, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, 2009.

[12] A. Tanenbaum. Computer Networks (Third Edition), pp 319-321, 1996.

[13] C. Thomassen. On the complexity of finding a minimum cycle cover of a graph. SIAM journal of computing, 26:675–6777, 1997.

[14] T. Wu, D. Kolar y R. Cardwell. Survivable Network Architectures for Broad-Band Fiber Optic Networks: Model and Performance Comparison. Journal of Lightwave Technology, Vol. 6, No 11, Noviembre 1988.