Tesis de Licenciatura en Ciencias de la computación

“Un algoritmo Tabu Search para el problema del cubrimiento de redes por ciclos acotados”

Guillermo Picardi - L.U. 113/98

Directora: Dra. Irene Loiseau

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

Octubre 2009

Resumen

En sus diversas fases de diseño, construcción y mantenimiento, las redes de comunicación traen aparejados grandes desafíos tecnológicos, logísticos, económicos y algorítmicos, entre otros. Debido a la demanda a la que están sometidas permanentemente; al grado de dependencia creciente en su uso, manifiesta en las miles de actividades e intercambios que estas posibilitan; y a la continuidad del servicio a la que están sometidas, se requiere diseñar y/o configurar redes seguras, que además minimicen sus costos operativos. Entendiendo por una red segura, una que ante eventuales fallas en algunos de sus enlaces, tenga la capacidad de seguir trasmitiendo a través de sus enlaces restantes.

Nuestro trabajo tiene por objeto proponer un algoritmo para el problema del Cubrimiento de Redes por Ciclos Acotados o Bounded Cycle Cover Problem (BCCP). Esto es, dada una red física, se requiere construir subredes de manera tal que cubran toda la red original, minimizando los costos involucrados de esta. Cada subred tiene forma de anillo o ciclo y, asimismo, cada uno de estos ciclos no deberá superar un cierto tamaño preestablecido. Estos ciclos modelan el problema de los *Self healing ring (SHR)*. Los mismos son bidireccionales y muy utilizados en comunicaciones de fibra óptica.

La estructura de un *SHR*, hace que si se cae un enlace del mismo, se pueda seguir transmitiendo información en los enlaces restantes. Entonces, si se garantiza la comunicación dentro de cada ciclo, se garantiza la comunicación en toda la red.

Dada una red, nuestra solución propone, en una primera etapa, generar ciclos de tamaño menor o igual al máximo preestablecido. Cada uno de ellos tiene un costo asociado. Ulteriormente, en una segunda etapa, aplicamos un algoritmo de Tabú Search con el objeto de decidir qué ciclos seleccionaríamos, de manera tal que estos cubran todos los ejes de la red original, minimizando la cantidad y/o los costos de los ciclos involucrados. Los resultados obtenidos fueron muy buenos comparados con los disponibles en la literatura.

Abstract

In the process of its design, construction and maintenance, communication networks lead to important technological, logistical, economical and algorithmical challenges among others.

Due to the demand they are permanently put under, to the growing dependency on its use – noticeable in the thousands of activities and interchanges that they make possible– and to the continuity of service that they are required to provide, it is needed to design and configure safe networks that, additionally, have minimum operative costs. It is understood by safe network, such a network that, in the presence of a failure of one of its links, is able to keep transmitting through its others links.

The goal of this work is to propose an algorithm to face the Bounded Cycle Cover problem. That is, for a given a physical network, it is required to build subnets such that they cover the whole original network, minimizing the involved costs. Every subnet has a ring or cycle structure; neither of them will exceed some predetermined size. Those cycles model the Self Healing Ring (SHR) problem. They are bidirectional, and heavily used in optical fiber communications. The structure of an SHR allow, in the presence a failure on one of the network links, to keep transmitting through the remaining active links. As a consequence, if the communication inside each cycle is guaranteed, communication throughout the entire network is ensured.

Given a computer network, our solution proposes, in a first phase, the generation of cycles with a length lower than or equal to a preset maximum value. Each of them has an associated cost.

Later, in a second phase, a Tabu Search algorithm is applied to decide which cycles can be selected, so that they cover the whole original network, minimizing the amount of them and their cost.

The achieved results were very good when compared to the ones available in the literature.

1 Introducción

Al igual que un fluido vital, en nuestra sociedad la información digital está en todas partes: circula en las redes, se exhibe en las pantallas, se oye en los teléfonos celulares, etc. Nuevas formas de comunicación, como ser e-mails, chat, telefonía celular, mensajes de texto, telefonía IP, video llamadas, videoconferencias, nos permiten contactar a otras personas en tiempo real, en cualquier lugar del mundo. Todos los artefactos materiales que en otra época estaban asociados a nuestras prácticas de acceso a la información, tales como ser libros, diarios, música, documentales, películas, álbumes de fotos, etc. ceden terreno ante las herramientas electrónicas.
Las empresas también se convierten a lo digital. Órdenes de pedido, facturas, seguimientos de envíos, archivos legal y contable, documentación de los productos, relación con el cliente, el ciclo de vida del documento administrativo se transforma, en la mayoría de los casos, en un circuito informático. Un proceso análogo sucede en las empresas trasnacionales, quienes cuentan con su propia red corporativa que permite integrar a todo su personal geográficamente distribuido; centralizar el acceso y el flujo de la información; informatizar los circuitos de trabajo; monitorear desde niveles jerárquicos a sectores productivos u operativos on-line, sin importar su ubicación física. Tales transformaciones apuntan a facilitar el intercambio de información, agilizar los tiempos de respuesta internos, mejorar la toma de decisión, etc.

Las redes de comunicaciones, son las estructuras subyacentes que posibilitan todo este intercambio de información a través de las computadoras.

En la actualidad, las mismas constituyen un factor clave en el desarrollo del mundo contemporáneo: ciudades, países, continentes interconectados intercambiando información de todo tipo en todo momento. Puesto que, debido al grado de dependencia creciente, la criticidad en aumento de las diversas aplicaciones involucradas, y la incorporación de cables de fibra óptica que concentran cientos de miles de comunicaciones, se requiere diseñar ***redes seguras***, que tengan la capacitad de seguir transmitiendo información a pesar de que falle alguno de sus enlaces en la transmisión. Dicha virtud se la conoce como ***supervivencia*** (en inglés ***survivability***). Muchas de las posibles causas se encuentran asociadas a fenómenos naturales o simplemente a fallas técnicas.

1.1 Objetivo

Nuestro trabajo se centrará en redes de comunicación conocidas con el nombre de ***self healing network***. Éstas poseen la virtud de seguir transmitiendo información a pesar de tener fallas en algunos de sus enlaces. Lo que se pretende con estas redes es unir diferentes ciudades de uno o varios países.
Nuestro objetivo es, dado una *red* física y un tamaño fijo *K*, construir pequeñas subredes *ad hoc* con forma de *ciclo* que no superen dicho *tamaño K*, de manera que la unión de todas ellas, cubran todos los *enlaces* de la *red* original minimizando los costos involucrados. Esto se modela con el problema del **Cubrimiento de Redes por Ciclos Acotados** (***Bounded Cycle Cover Problem***) ***(BCCP)*** y nuestra meta será encontrar una solución adecuada al mismo.

Se dice que un ciclo es ***self healing ring***, porque si se cae un enlace, permite que los paquetes que circulan de un nodo a otro, puedan ser redireccionados en forma automática, tomando el camino alternativo en el ciclo.

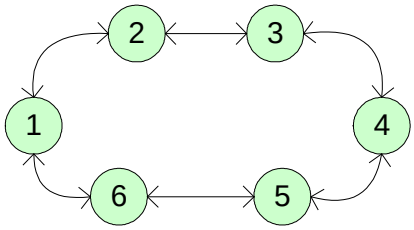


Figura 1.1

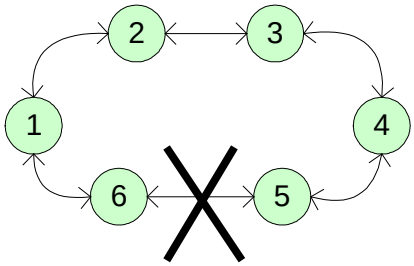


Figura 1.2

En la figura 1.1 observamos un ciclo con seis nodos y sus respectivos enlaces. En la figura 1.2 observamos cómo es interrumpido el enlace que va del nodo 5 al 6 y, a pesar de ello, todos los nodos cuentan con un camino alternativo por el que pueden enviar información a cualquier otro nodo. En la práctica se utilizan ciclos que no suelen superar los 10 enlaces. De esta manera, la probabilidad de que se caigan dos de ellos, dentro de un mismo ciclo es baja, de manera que este enfoque resulta adecuado para mitigar las fallas más comunes. Además, según los técnicos, esto posibilita una mejor calidad en las comunicaciones.
A menudo ocurre que, cuando se cae un enlace, éste puede ser reestablecido, reparado o sustituido en poco tiempo. Por tanto, no necesitaríamos más redundancia que la provista por el ciclo. Sin embargo, puede ser que para otro tipo de topologías se requiera mayor redundancia, pero eso elevará los costos de la red.

Cada enlace tiene un costo asociado. El costo cada ciclo generado ad hoc para cubrir la red, está dado por la suma de los costos de los enlaces que lo componen. En tanto, el costo de la solución empleada esta dado la suma de los costos de los ciclos que la conforman y esto es lo que queremos minimizar.

Este problema pertenece a la categoría de ***intratables***, más específicamente ***NP-Hard*** [8]. El problema de decisión asociado es NP-Completo $\forall K \varepsilon O(1)$. La cantidad de soluciones posibles crece exponencialmente respecto del tamaño de la red original y del *K* preestablecido. Por tanto, si queremos explorar todas ellas para encontrar la mejor, nos puede llevar cientos de años!
Por esta razón proponemos como solución implementar la heurística de ***Tabú Search***, adaptada a las particularidades de este problema. La misma no garantizará que tengamos la mejor solución posible, pero sí buenas soluciones en tiempos razonables.

1.2 Organización de este trabajo

- En la sección 2 daremos una descripción detallada y formal del problema a tratar. Asimismo veremos algunos de los trabajos relacionados y abordaremos el Problema del Cartero Chino y la estrecha relación que este guarda con nuestro problema.
- En la sección 3 explicaremos en qué consiste la heurística Tabú Search, cuáles son sus principales características y aspectos relevantes.
- En la sección 4 expondremos el algoritmo propuesto y en la sección 5, otras variantes que fueron inicialmente abordadas sin éxito, aunque contribuyeron de manera indirecta a la solución definitiva.
- En la sección 6 analizaremos los resultados obtenidos a partir de los ensayos realizados.
- Finalmente, en las secciones 7 y 8 presentaremos las conclusiones obtenidas y los trabajos a futuro, respectivamente.

2 Descripción del problema

2.1 The Bounded Cycle-Cover Problem - BCCP

En las redes o grafos con que trabajaremos los nodos representan ciudades y los ejes representan los enlaces de fibra óptica que unen a dichas ciudades.
Tal como antes se había mencionado, una consideración a tener en cuenta a la hora de diseñar redes de comunicación es su alta capacidad de supervivencia a las fallas.

Una grafo donde existe al menos un camino entre cualquier par de nodos se considera ***conexa***, ***1-conexa*** o ***1-connected***. Nuestro problema parte del supuesto de que las redes con las que trabajaremos son ***2-conexa***. Esto significa que, para todo par de nodos, existen al menos dos caminos simples distintos entre ellos, garantizando así una mayor redundancia y posibilitando la construcción de ***redes seguras***.

Los ciclos con los que trabajaremos no pueden superar cierto tamaño K previamente establecido. Esto se debe a razones técnicas, dado que no se quiere generar recorridos demasiado largos dentro de los ciclos. Un ciclo está compuesto por nodos y enlaces en similares cantidades. Los enlaces tienen un costo asociado que puede representar el costo económico, la distancia física entre un nodo y otro, el ancho de banda del enlace, un promedio de todos los anteriores, etc.

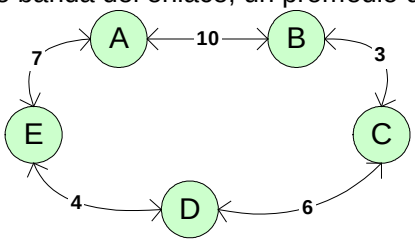


Figura 2.1.1 – Ciclo de tamaño 5 y costo 30

En la figura 2.1.1 podemos observar un ciclo de tamaño 5, es decir que cuenta con 5 nodos A, B, C, D y E, y además 5 enlaces con sus respectivos costos asociados: (A, B, 10), (B, C, 3), (C, D, 6), (D, E, 4) y (E, A, 7). El costo total del ciclo es la suma de los costos de sus enlaces, en este caso es de 30.
Los ciclos con los que vamos a trabajar deberán ser de tamaño mayor o igual a 3, porque los de tamaño 2 se los considera ciclos degenerados y quedan fuera del alcance de éste problema. Por las razones antes mencionadas estos no superaran el tamaño 8.

Conceptualmente, **el problema del Cubrimiento de Redes por Ciclos Acotados ó *Bounded Cycle Cover Problem*** (al que nos referiremos como ***BCCP***), consiste en encontrar un cubrimiento de todos los ejes del grafo con ciclos que no superen el tamaño K, de modo de minimizar los costos involucrados. Por ejemplo, su pongamos que tenemos la siguiente red[1] y nuestro K es 4:

[1] Esta red está fue obtenida de un ejemplo real donde sus nodos son ciudades de Estados Unidos y el costo de sus enlaces representa la distancia entre dichas ciudades medida en millas. Notar que por sus dimensiones, la misma podría cubrir por ejemplo a 10 ciudades cualesquiera de un país como Argentina.

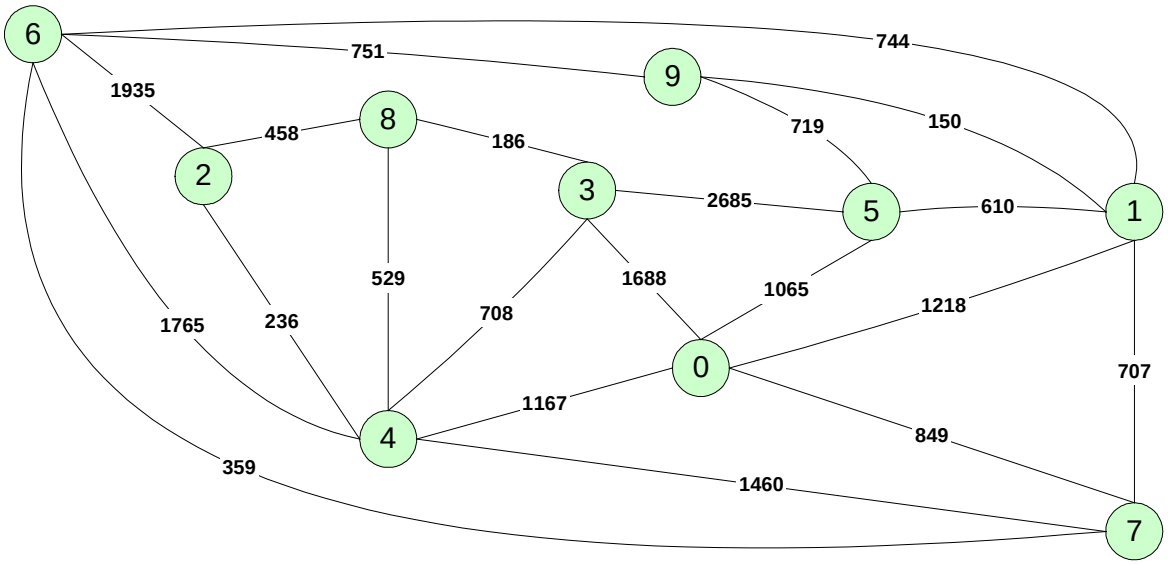
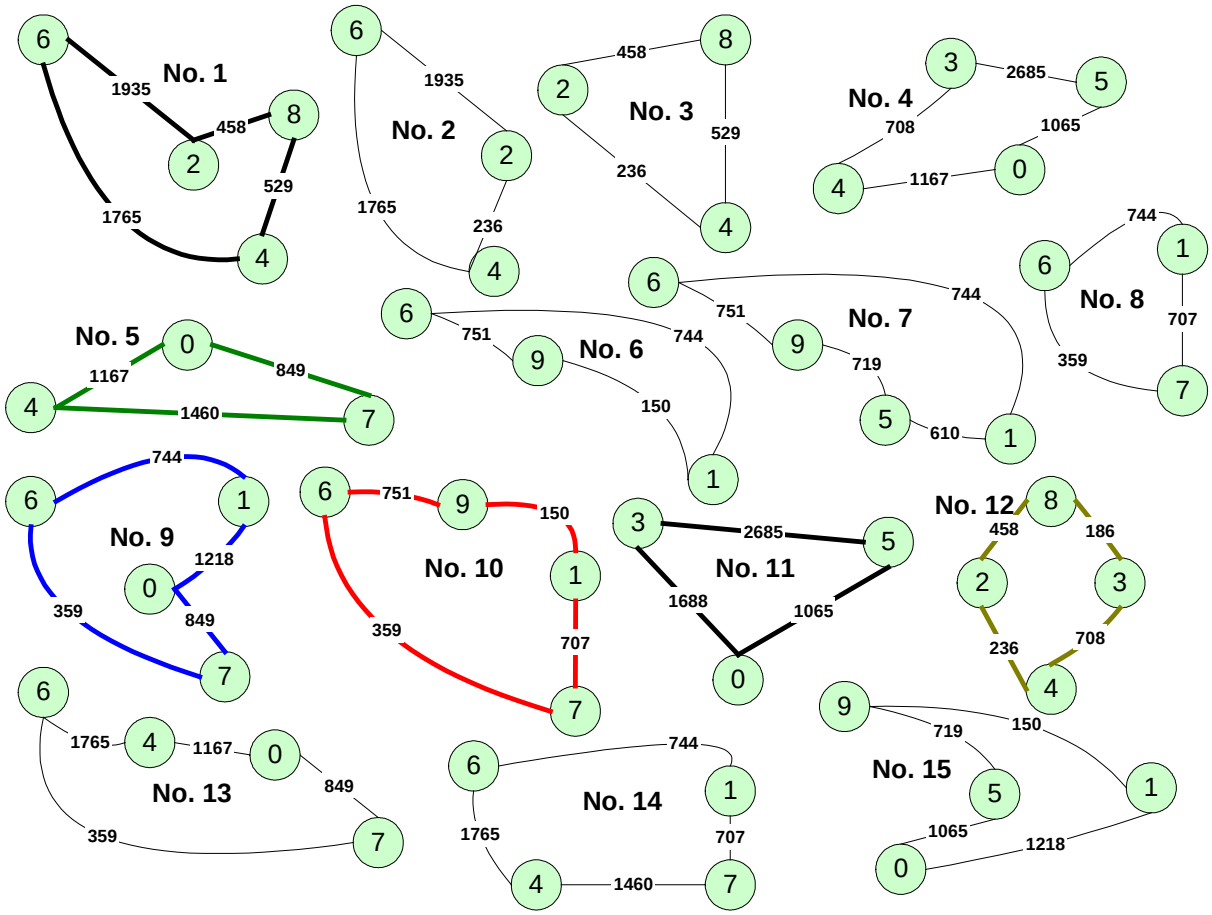


Figura 2.1.2 – Grafo de una red real de 10 nodos

Nuestro problema consiste en encontrar un cubrimiento con ciclos generados a partir de esta red, de tamaño 3 y 4. Si K fuese 5, éstos serían de tamaño 3, 4 y 5; si fuese 6, entonces dichos ciclos serían 3, 4, 5 y 6; y así. En este ejemplo, los ciclos posibles para K = 4 son:



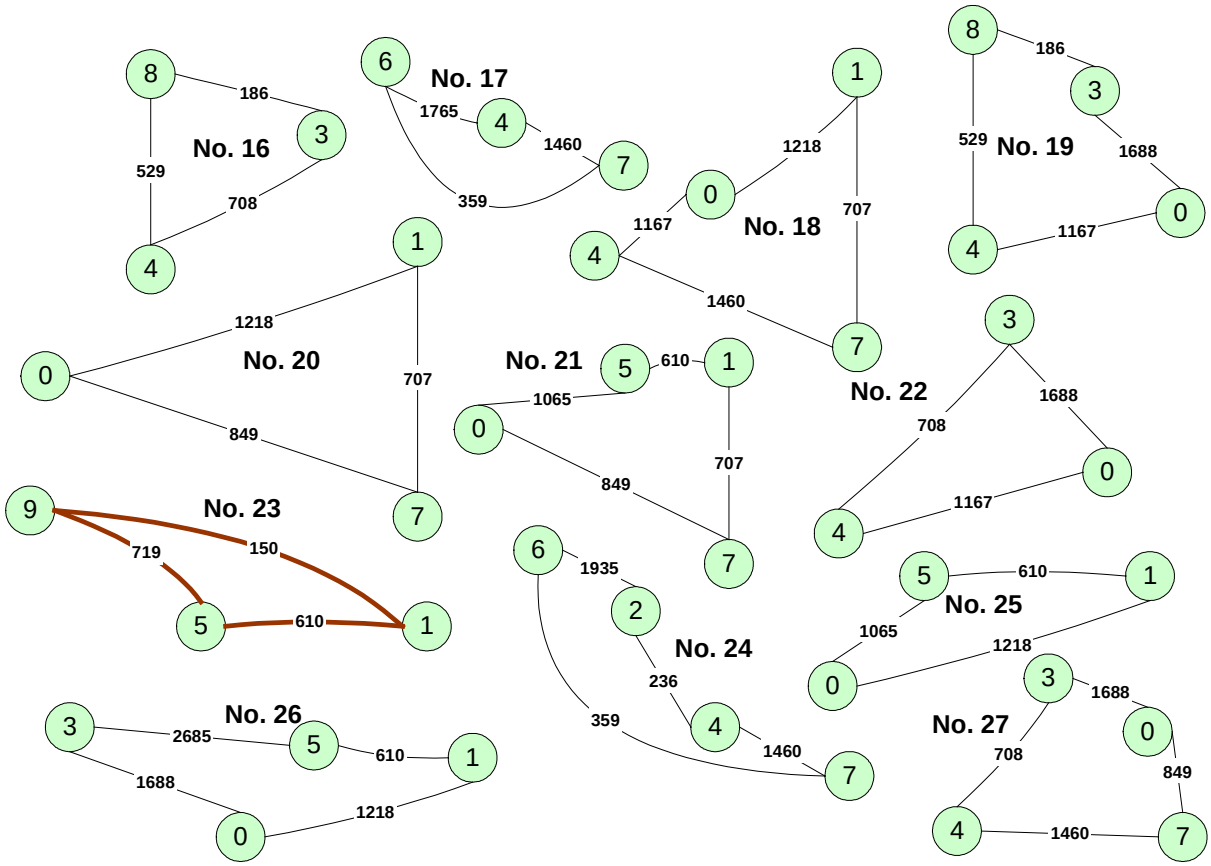


Figura 2.1.3 – Ciclos de tamaño 3 y 4 del grafo de la figura 2.1.2

Luego, en la figura 2.1.4 se observa un cubrimiento posible. Los ciclos 1, 5, 9, 10, 11 y 23, resaltados en diferentes colores, conformar la solución.

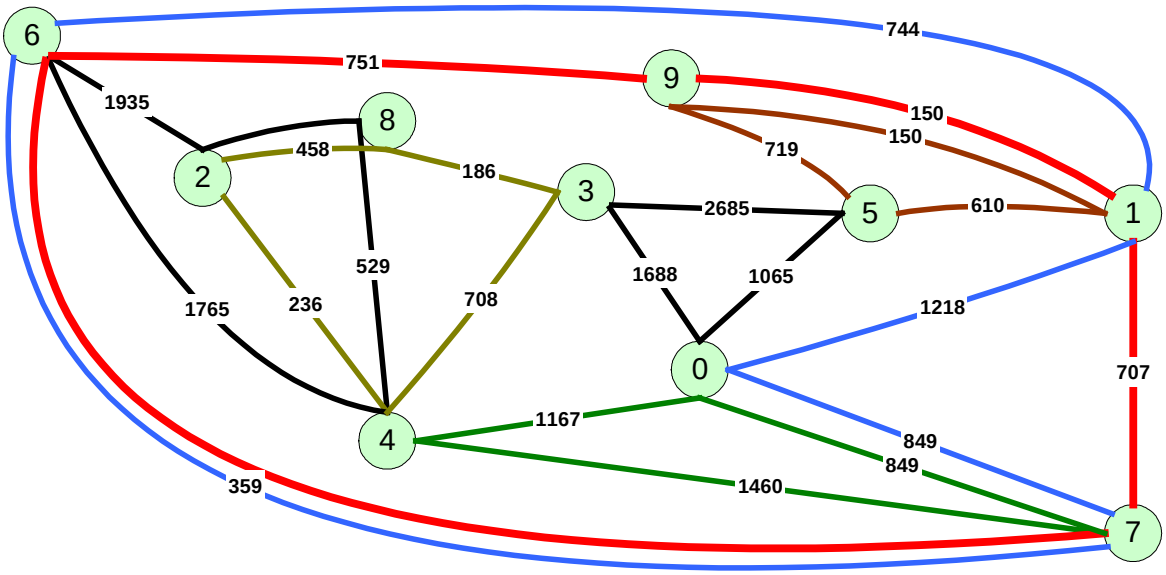


Figura 2.1.4 – Cubrimiento del grafo de la figura 2.1.2

Más adelante veremos que éste resultado es el mejor posible para esta red con K=4.

2.2 Definición formal

A continuación daremos algunas definiciones y posteriormente procedemos a describir el problema.

Ejes adyacentes: sean e_1 y e_2 dos ejes cualesquiera, se dicen que e_1 y e_2 son adyacentes si un extremo de e_1 coincide con uno de e_2, es decir, ambos tienen un mismo nodo en común.

Camino: un camino en un grafo es una sucesión de ejes e_1 e_2.....e_n tal que un extremo de e_i coincide con uno de e_{i-1} y el otro extremo con uno de e_{i+1}. La notación empleada P_{ij} indica un *camino* que va del nodo *i* al nodo *j*.

Camino simple: es un camino que no pasa dos veces por el mismo nodo.

Circuito: es un camino que empieza y termina en el mismo nodo.

Ciclo / Circuito simple: un ciclo es un circuito de 3 o más nodos que no pasa dos veces por el mismo nodo. Un ciclo con nodos i y j, es un camino simple P_{ij} sin nodos repetidos, seguido del eje (i, j). Para denotarlo utilizamos C_{ij}. El costo de un ciclo es la suma de los costos de los ejes que contiene C_{ij}. La longitud del ciclo $|C_{ij}|$, es la cantidad de ejes que éste tiene, que es la misma cantidad de nodos.

K-path: un $K\text{-}path$ es un camino simple de a lo sumo longitud K.

K-Ciclo: un $K\text{-}ciclo$ es un ciclo de longitud mayor o igual a tres y de longitud menor o igual a K.

Grafo conexo: un grafo es *conexo* o *1-conexo* si cada par de nodos está conectado por un camino; es decir, si para cualquier par de vértices *(a, b)*, existe al menos un camino posible desde *a* hacia *b*.

Grafo 2 - conexo: un grafo G es *2-conexo* si existen al menos dos caminos simples alternativos distintos entre sí para todo par de nodos.

Cycle Cover (CC): decimos que un ciclo C cubre al eje e, cuando C contiene al eje e del grafo. Un *Cycle Cover (CC)* es un conjunto de ciclos que cubren todos los ejes del grafo.

Bounded Cycle Cover (BCC): es un conjunto de *K-ciclos* que cubren todos los ejes del grafo. A partir del *K* surge la palabra *limitado* o *bounded.*

Datos de entrada del problema

Sea $G = (V, E)$ un grafo bidireccional / no orientado y 2-conexo.

Sea *K* la longitud máxima permitida de los ciclos de *G* para nuestro problema. El rango de valores con los que trabajaremos es entre 4 y 8, por disponer de resultados en la literatura para estos valores, además de las razones técnicas antes mencionadas.

Las soluciones con las que trabajaremos son minimales; de manera que si eliminásemos un ciclo de una solución S, haríamos que al menos un eje del grafo quede descubierto, y por lo tanto S dejaría de ser una solución o un BCC válido. Esto es porque estamos trabajando con costos positivos.

El costo de un eje e es c_e y la variable x_e indica la cantidad de ciclos que cubren a e que participan del cubrimiento. Por otro lado, $E_{k\text{-}ciclo}$ representa el conjunto de ejes E que tiene asociado el $K\text{-}ciclo$.

Este problema pertenece a la categoría de intratables, en particular es NP – Hard [8], y su definición formal es:

Problema: *Bounded Cycle-Cover Problem*

Instancia: *sea* $G=(V,E)$ *un grafo 2 - conexo, donde cada eje* $e \in E$ *tiene un costo asociado* c_e *y sea* K *un número natural*

Problema a Optimizar: *encontrar un conjunto de K-ciclos* $C_1,C_2,...,C_n$ *tal que minimicen* $\sum_{e \in E} c_e x_e$ tal que $E = \bigcup_{e \in K-Ciclo} E_{K-Ciclo}$ *, es decir, E puede ser particionado en K-ciclos de G, o mostrar que el problema no tiene solución para el valor de* K *dado.*

2.3 The Chinese Postman Problem – CPP

Dado que no conocemos un algoritmo que en tiempo polimomial explore todas las soluciones posibles de nuestro problema, no estamos en condiciones de saber que tan buena es una solución hallada. Por esa razón necesitamos encontrar una cota que nos permita estimar la bondad de la solución. A priori, se nos ocurre la suma de todos los ejes del grafo como cota inferior, dado que cada uno de ellos debe ser cubierto por al menos un ciclo.

Para nuestro ejemplo expuesto en la figura 2.1.2, esta cota inferior es 19989. La solución obtenida de la figura 2.1.4, nos da un total de 21805. De esta manera podemos comparar qué tan buena fue esa solución, y qué tan cerca estamos de la cota inferior. Pero, ¿no existirá una mejor cota inferior? ¿Cómo podemos saber si el resultado de 2.1.4 es la mejor solución posible?
La respuesta a la primer pregunta es afirmativa. Podemos encontrar una cota inferior mejor, como veremos a continuación, que corresponde a la solución del ***Problema del Cartero Chino*** o ***Chinese Postman Problem*** (CPP) [8]. Respecto de la segunda pregunta, podemos aseverar que, si la solución hallada tiene el mismo valor que la encontrada por el Cartero Chino, sí es la mejor de todas.

Definición
Dado un grafo G, el objetivo del CPP es encontrar un circuito que pase al menos una vez por todos los ejes de G, minimizando los costos involucrados. A este circuito se lo conoce como *Postman Tour* o *Recorrido del Cartero.*

A modo de ilustración, dicho problema puede ser aplicado, tal como su nombre lo indica, a un cartero que debe realizar un recorrido para entregar la correspondencia. Otro ejemplo del mismo problema, es un camión recolector de residuos cuya función es recolectar la basura de algunas calles de la ciudad. Cabe destacar que en este ejemplo hay que adicionar el sentido de las calles a los ejes del grafo que la representan. Además, para que el problema se ajuste a nuestro ejemplo, hay que omitir algunos factores como la capacidad máxima de recolección de basura, el combustible limitado y las restricciones del horario entre otros. En ambos casos, el objetivo es encontrar el recorrido mínimo tal que cubra todas las calles deseadas.

Existen algoritmos que pueden resolver este problema en forma exacta en tiempo polinomial. Afortunadamente nosotros disponemos de una implementación que utilizamos en los ensayos [3].

En el siguiente ejemplo se observa un grafo y a su lado el *Recorrido del Cartero* hallado. Éste último esta dado por: 4 → 3 → 1→ 2 → 3 → 6 → 3→ 5 → 2 → 4 → 5. El nodo inicial es 4, está resaltado en rojo. Luego el número de cada eje indica el orden en que se efectúa el recorrido.

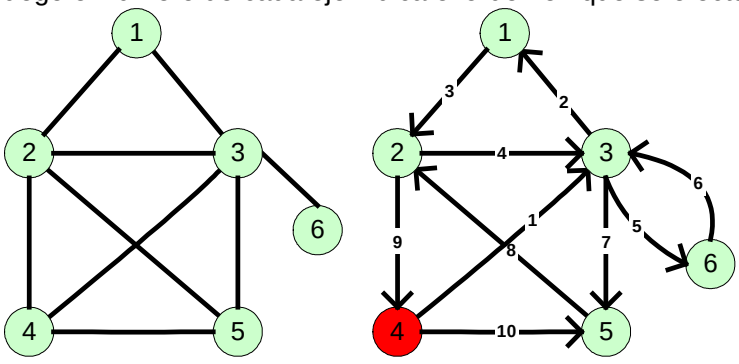


Figura 2.4.1 – Ejemplo de un Recorrido del Cartero Chino.
El nodo 4 resaltado en 4 es el inicial. El valor de cada eje indica el orden en que se efectúa el recorrido.

Notar que el eje (3,6) se repite en los pasos 5 y 6. La idea es que en un grafo mucho más complejo, se repitan los ejes de menor costo y/o la menor cantidad posible de ellos, de manera de

obtener el *Recorrido* más eficiente. Cabe destacar que cuando un grafo tiene un circuito Euleriano, éste es la solución del CPP.

Volviendo a nuestro ejemplo de la figura 2.1.2, el *Recorrido* mínimo obtenido es el siguiente:

Paso	Eje	Costo
1	0 → 1	1218
2	1 → 9	150
3	9 → 6	751
4	6 → 4	1765
5	4 → 7	1460
6	7 → 6	359
7	6 → 2	1935
8	2 → 8	458

Paso	Eje	Costo
10	3 → 5	2685
11	5 → 9	719
12	9 → 1	150
13	1 → 5	610
14	5 → 0	1065
15	0 → 7	849
16	7 → 6	359
17	6 → 1	744

Paso	Eje	Costo
19	7 → 0	849
20	0 → 4	1167
21	4 → 2	236
22	2 → 8	458
23	8 → 4	529
24	4 → 3	708
25	3 → 0	1688
	Total	**21805**

Tabla 2.4.3 – Recorrido del Carteo para la figura 2.1.2

Para hallar este recorrido el algoritmo generó los siguientes ciclos:

Nro.	Ciclo	Costo
1	[0, 1, 7, 0]	2774
2	[3, 0, 4, 3]	3563
3	[5, 0, 7, 6, 1, 5]	3761
4	[6, 4, 7, 6]	3584

Nro.	Ciclo	Costo
5	[9, 1, 9]	300
6	[4, 2, 8, 4]	1223
7	[6, 2, 8, 3, 5, 9, 6]	6734

Tabla 2.4.4 – Ciclos generados por el Cartero Chino para la figura 2.1.2

Notar que el costo del *Recorrido* mínimo obtenido es el mismo de nuestra solución: *21805*, es decir, la solución encontrada en 2.1.4 es la mejor posible por igualar a la cota inferior. Sin bien la solución para nuestro problema (BCCP) difiere de la del cartero chino (CPP), esta última podría ser solución de BCCP, aunque no en este ejemplo. Esto sucede porque en nuestra solución de 2.1.4, el grafo está cubierto con ciclos de tamaño 3 y 4. En cambio, en la tabla 2.4.4 podemos observar que los ciclos del cartero chino nro. 3 y 7, cuyos tamaños son 5 y 6 respectivamente, superan a K = 4, invalidando la solución. Podríamos afirmar que es un BCC válido para $K \geq 6$, pero esto no es así porque el ciclo nro. 5 es un ciclo degenerado, es decir, de tamaño 2.
Gráficamente, los ejes cubiertos por el *Recorrido* mínimo de nuestro ejemplo quedan conformados de la siguiente manera:

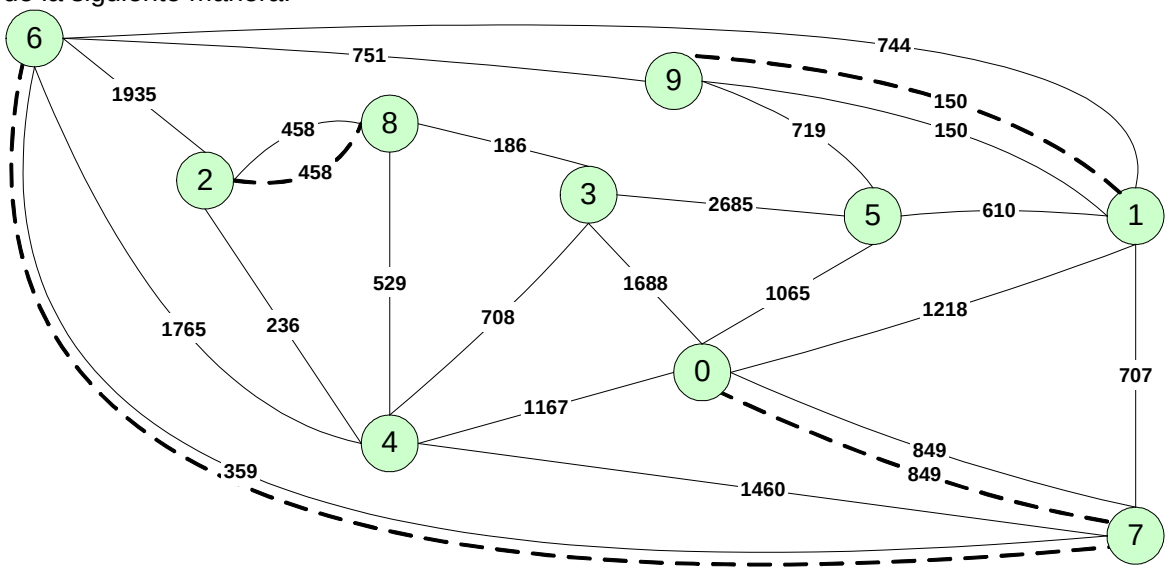


Figura 2.4.5 – Ejes adicionales agregados por el cartero chino a la figura 2.1.2

Las líneas punteadas indican los ejes que se repiten. Notar que estos cuatro ejes: $\{(0,7),(1,9),(2,8),(6,7)\}$, son los mismos que se repiten en nuestra solución en la figura 2.1.4, y, como antes habíamos señalado, al sumar el costo de todos los ejes, más los cuatro que se repiten, el costo total del *recorrido del cartero* es el mismo de nuestra solución: *21805*.

Hasta aquí tenemos que CPP es un problema polinomial donde para encontrar el *Recorrido del Cartero*, genera ciclos de tamaño mayor o igual a dos (admite ciclos degenerados).
Ahora introducimos al problema MCCP (Minimun-cost Cycle Cover Problem), que es similar a BCCP, sólo que no tiene restricciones con el tamaño de los ciclos. Esto significa que un cubrimiento con ciclos de tamaño mayor o igual a tres es solución de MCCP. Notar que no se admiten ciclos degenerados. Este problema es NP-Completo y es una instancia particular de BCCP para $K=n$. De aquí se deduce que BCCP es NP-Hard y que MCCP ≤ BCCP. Las soluciones de BCCP son soluciones para MCCP.
MCCP es equivalente a CPP para grafos planares [8]. Esto se debe a que CPP genera un *Recorrido del Cartero* libre de ciclos degenerados en grafos planares, conformando así una solución para CPP y para MCCP. En el caso de grafos no planares, CPP actúa como cota inferior de MCCP. Entonces tenemos que CPP ≤ MCCP y que MCCP ≤ BCCP. Luego, CPP ≤ BCCP.

Si nos preguntamos: CPP resuelve a BCCP?
La respuesta inmediata sería que no, porque la solución de CPP admite ciclos degenerados ó de tamaño mayor a K. Por otro lado, si la respuesta fuese afirmativa, implicaría P=NP. La conclusión de este punto es que CPP es una cota inferior de BCCP.

2.4 Estado del Arte

Entre los trabajos relacionados con el BCCP, podemos mencionar los siguientes:

- D. Hochbaum y E. Olinick [8], proponen las siguientes 3 heurísticas para BCCP:
  1. Cycle-Basis: construye una base de *K-Ciclos* (a lo sumo $n \times m$), y luego aplica un algoritmo goloso para construir un *BCC*.
  2. Augmented Cycle-Basis: aplica la heurística anterior y, en caso de no hallar un *BCC*, genera y agrega los *K-Ciclos* necesarios para obtener un BCC.
  3. Greedy Postman: parte de un *Recorrido del Cartero* y luego genera los *K-Ciclos* necesarios para transformar este r*ecorrido* en un BCC. En otras palabras, reemplaza los ciclos degenerados de tamaño 2, y los que superan el tamaño *K*, por otros *K-Ciclos* válidos.

  Los autores de este trabajo formulan un modelo de programación lineal del BCCP como problema de cubrimiento de conjuntos (Set Covering Problem). Cada columna de la matriz del modelo es un ciclo acotado (K-Ciclo).
  En lugar de resolver el problema con todas las columnas posibles, reducen la formulación usando sólo los ciclos obtenidos por las heurísticas. Luego utilizan el paquete CPLEX para resolverlo. Los resultados de este trabajo nos servirán de referencia para comparar con los nuestros.

- B. Fortz y M. Labbé [6], proponen un algoritmo de Tabú Search para resolver el problema Two-Connected Network with Bounded Rings (2CNBR). Este problema difiere del nuestro, respecto de que se pretende encontrar un cubrimiento de *K-Ciclos* (*K-Cycle Cover*), que minimice los costos, a un subconjunto de ejes tal que cubran todos los nodos del grafo. En otras palabras, dado $G = (V, E)$, el objetivo es encontrar un $F \subseteq E$ tal que F incluya todos los nodos $n \in V$, F sea 2-conexo, y que cada eje $e \in F$ pertenezca a un ciclo de tamaño menor o igual a K.
  Este trabajo pretende ser una cota superior con relación a otros trabajos desarrollados por los autores, donde implementan algoritmos de Branch & Cut a problemas de la vida real, aunque limitado a instancias pequeñas.
  Si bien este problema difiere del nuestro, tiene características estructurales que lo asimilan en varios aspectos, y por esa razón se le prestó especial atención.

- En otro trabajo de B. Fortz y M. Labbé [7], formulan y demuestran una cota inferior para el problema anteriormente citado 2CNBR. Muestran que el problema de decisión asociado es NP completo y finalmente implementan un algoritmo de Branch & Cut.

- C. Ribeiro e I. Rosseti [13] proponen un algoritmo GRASP paralelo que utiliza una heurística de path-relinking para el problema 2-path network design (2PNDP). Sea $G = (V, E)$, el problema consiste en encontrar un subconjunto de ejes $F \subseteq E$ tal que F minimice los costos, y contenga un *camino* para cualquier par de nodos [esto es lo mismo que encontrar un arbol generador mínimo]. La idea de path relinking es combinar atributos en común de antiguas soluciones para luego intentar encontrar mejores soluciones. El algoritmo de GRASP consiste en la distribución de las iteraciones de través de los diversos procesadores involucrados. Los ensayos fueron realizados con 1, 2, 4, 8, 16 y 32 procesadores.

3 Tabú Search

Introducción
Tabú Search es una metaheurística muy utilizada en problemas de optimización combinatoria. Muchos de estos problemas se caracterizan por tener muchas variables involucradas, ser complejos de analizar y visualizar, no conocérseles algoritmos exactos que los resuelvan en un tiempo razonable, etc.

Algunos ejemplos de esta clase de problemas son:

- Diseño de redes de comunicación (BCCP, 2CNBR, 2PNDP)
- Ruteo en redes de comunicación
- Coloreo de grafos
- Ruteo de vehículos. (Viajante de comercio)
- Clique máximo
- Max Cut
- Asignación de recursos y horarios en instituciones educativas
- Minimizaron de desperdicios en el corte de materiales
- Localización de plantas
- Planificación financiera
- Problemas de energía

Tabu Search fue presentado por primera vez en 1986 por Glover.

Se trata de un procedimiento iterativo que aplica una exploración guiada en el espacio de soluciones a tratar. Si bien aún no se conoce una demostración formal que explique su buen comportamiento, empíricamente se puede observar el exitoso desempeño en los problemas antes mencionados. Dicho procedimiento intenta mejorar la búsqueda local en el espacio de soluciones.

Los rasgos más relevantes de esta metaheurística son:

- Parte de una única solución inicial, que luego se va modificando hasta obtener el resultado
- Acepta peores soluciones que la mejor encontrada hasta el momento
- Utiliza una lista tabú con el objeto de forzar al algoritmo a explorar nuevas soluciones, y evitar de esta manera que el algoritmo caiga en un ciclo repetitivo
- Puede utilizar una función de aspiración que permite poder elegir uno o varios elementos prohibidos por la lista tabú de una solución en particular

A continuación daremos una breve descripción de las características que forman parte de Tabú Search, y luego detallaremos cómo se articulan todas ellas.

Solución Inicial
Esta solución es utilizada como base por el algoritmo Tabú Search. La misma es generada previamente por un algoritmo goloso, aleatorio o alguna otra heurística.

Función Objetivo
Constituye la función de evaluación aplicada que es aplicada a las soluciones del espacio. El objetivo de Tabú Search es minimizar / maximizar esta función.

Vecindad
Es el espacio de soluciones posibles del algoritmo en un contexto dado, dentro del espacio global de soluciones.

Mínimo Local
También llamado mínimo relativo, es un mínimo que no necesita ser (pero puede ser) un mínimo global. Formalmente diremos que una función ***f(x)*** alcanza un mínimo local en un punto ***a*** de su dominio si existe un entorno de ***a*** en el que los valores de ***f*** son mayores que ***f(a)***. Si la función es derivable en ***a***, entonces ***f'(a)=0***.

Movimiento
Un *movimiento* o *tabú move* es una variación a la solución actual, que nos posibilita cambiar de una vecindad a otra, es decir, un espacio de soluciones por otro.

Lista Tabú
La lista tabú sirve para prohibir soluciones o parte de las mismas durante un número de iteraciones, con el objeto de evitar que el algoritmo repita soluciones. Lo que busca el algoritmo de Tabú Search, es escapar de un mínimo local permitiéndole desplazarse de esta forma hacia una nueva vecindad.

Función de Aspiración
Hay veces que necesitamos relajar las decisiones cuando una función tabú resulta atractiva. En ese caso la función de aspiración nos permite realizar movimientos prohibidos que se encuentren en la lista tabú, bajo un criterio o umbral establecido; por ejemplo cuando estos permiten alcanzar una mejor solución respecto de la mejor encontrada hasta el momento.

Criterio de parada
Para este tipo de algoritmo, suele estar dado por la realización de una determinada cantidad de iteraciones en las que no se encuentre una mejora en la solución. Otros criterios pueden ser la utilización de una cota de tiempo o un valor de umbral en el que se considere que la solución obtenida es suficientemente buena para la aplicación en cuestión.

Intensificación
La intensificación es una estrategia de búsqueda en el marco de tabú search, que tiene por objeto explorar en mayor detalle una determinada zona del espacio de soluciones o ***vecindad***, con la expectativa de encontrar una solución mejor a las conocidas. La intensificación puede estar motivada por alguna característica sobresaliente de una solución o conjunto de soluciones, sea en su calidad o basada en algún conjunto de atributos que estén asociados a buenas soluciones.
Existen diversas maneras de implementar esta técnica, la más simple consiste en disminuir el tamaño de la lista tabú, de modo que se permitan más movimientos de retroceso que, a riesgo de generar la ocurrencia de ciclos, permitan una exploración más detallada de una determinada zona.
Otra alternativa consiste en modificar el criterio de selección de movimientos a fin de privilegiar aquellos que estén asociados a atributos que aparecen frecuentemente en soluciones buenas.

Diversificación
Contraria a la estrategia de intensificación, la diversificación pretende analizar regiones o ***vecindades*** lejanas de la actual, en el espacio global de soluciones, con la intención de escapar a ***mínimos locales*** y evitar que queden ***vecindades*** sin visitar.
Una forma simple de implementar este comportamiento consiste en aumentar el tamaño de la lista tabú. De esta forma, al aumentar la cantidad de soluciones prohibidas en una zona dada, se fuerza al algoritmo a orientar la búsqueda a otras áreas lejanas a la actual.
Otra opción es modificar las reglas de selección de movimientos, a fin de privilegiar la utilización de atributos que no fueron empleados frecuentemente en las soluciones analizadas hasta el momento.

Introducir movimientos o variaciones de la solución actual al azar contribuye también a la diversificación, y además permite romper con el determinismo inherente que un algoritmo posee.

Tamaño de la lista tabú
La variación dinámica del tamaño de la lista tabú, permite almacenar una cantidad mayor o menor de movimientos o soluciones prohibidas.
Un aumento en el tamaño de la lista tabú puede verse como una diversificación, mientras que una disminución puede considerarse como una intensificación. La variación en el tamaño de la lista podría estar motivada en base a la frecuencia de aparición de determinados atributos en las soluciones que se visitaron recientemente, o bien en un componente aleatorio. La combinación de intensificación y diversificación permite una exploración amplia y detallada del espacio de soluciones aumentando las posibilidades de explorar nuevas ***vecindades*** para luego intentar encontrar mejores ***mínimos locales***.

Soluciones Elite
Se llaman soluciones elite a aquellas buenas soluciones que fueron exploradas durante el proceso de búsqueda y que por algún motivo se destacan del resto, ya sea por alguna característica en sus atributos o por tener valores de la función objetivo cercanos al mejor valor conocido. En general estas soluciones se las almacena en una lista para ser exploradas mediante un proceso de intensificación en momentos posteriores de la búsqueda.

Frecuencia
La idea de este punto es medir atributos, movimientos ó soluciones completas que se repiten en las soluciones buenas, con el objeto de establecer un patrón, para luego ayudar a tomar una decisión. Por ejemplo, un atributo que debe establecerse en nuestra solución, o un patrón en el comportamiento del algoritmo que debe ser modificado para explorar nuevas vecindades.

Algoritmo de Tabú Search
Una vez expuestas las características más relevantes, procedemos a describir en términos generales al algoritmo:

1. Determinar una solución inicial $i \in V$, donde V conforma el conjunto global de soluciones
2. Generar un subconjunto V^* de soluciones tal que $V^* \subset V$ que no sean Tabú o que cumplan con el criterio de Aspiración.
3. Encontrar la mejor solución $j \in V^*$, respecto la función objetivo $f / f(j) \leq f(k) \ \forall k \in V^*$
4. Hacer $i = j$
5. Actualizar la lista tabú y la función de Aspiración
6. Si cumple con el criterio de parada, entonces parar. Sino, volver al paso 2

Figura 3.1 – Esquema conceptual del algoritmo de Tabú Search

[Entonces el óptimo de un vencindario no es un mínimo local. mínimo local sería punto s tq s es mínimo en V(S).]

4 Solución Propuesta

La solución que proponemos establece articular al algoritmo en las siguientes dos etapas:

- Etapa 1: se centra en la generación de ciclos de tamaño menor o igual K *(K-ciclos)*. Para ello expondremos dos alternativas para la generación de ciclos.
- Etapa 2: el algoritmo de Tabú Search propuesto tiene por objeto decidir con qué ciclos nos quedamos a partir de los generados en la etapa anterior, de manera de minimizar los costos.

Se decidió estructurar la solución en estas dos etapas por las siguientes razones:

- Factorizar el problema principal en dos subproblemas algo más sencillos, con el objeto de analizar y resolver cada uno por separado.
- Analizar las dependencias e implicancias que tiene un subproblemas por sobre el otra.
- Partir del supuesto de que iba a ser muy beneficioso que el algoritmo de Tabú Search pueda disponer del espacio completo de soluciones. Aplicado a este problema, significa que el Tabú tenga a mano todos los *K-ciclos* posibles para un grafo dado. Luego, como se verá más adelante, esto no necesariamente es así.

4.1 Fase 1: Generador de Ciclos

4.1.1 Primera aproximación

Intuitivamente, pensamos en un algoritmo que, a partir de cada nodo inicial i, recorra todos los caminos posibles de tamaño menor o igual a K, sin repetir nodos. Si en esa búsqueda nos topamos con el nodo inicial i, significa que hemos encontrado un ciclo. Si el nodo i fue hallado en el paso 3, el ciclo encontrado tiene tamaño 3. Si fue en el paso K, tiene tamaño K, y así sucesivamente. Si fue encontrado en el paso 2, éste es un ciclo degenerado y por lo tanto hay que descartarlo. El problema que tiene éste algoritmo sencillo es que genera varias veces un mismo ciclo.

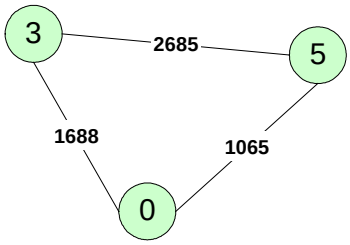


Figura 4.1.1.1- Ejemplo de un ciclo de tamaño 3

En la figura anterior se observa un ciclo de 3 nodos. Al aplicar el algoritmo, ese mismo ciclo se genera de seis formas distintas.

Nro.	Ciclo	Costo
1	[3, 5, 0 ,3]	5438
2	[5, 0, 3, 5]	5438
3	**[0, 3, 5, 0]**	**5438**

Nro.	Ciclo	Costo
4	[3, 0, 5, 3]	5438
5	[5, 3, 0, 5]	5438
6	[0, 5, 3, 0]	5438

Gráficamente, esto es:

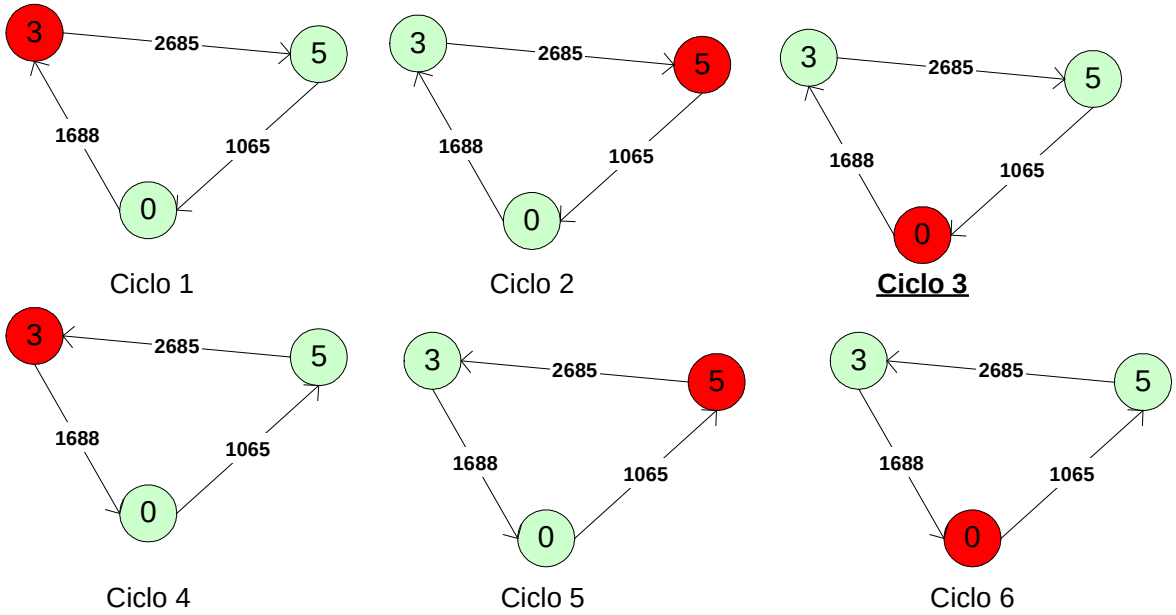


Figura 4.1.1.2 – Ciclos repetidos para la figura 4.1.1.1

El nodo inicial i, es el nodo rojo; las direcciones de los ejes indican el orden en el cuál se fue generando el ciclo. Por ejemplo, el ciclo 1 se creó en el orden 3,5 y 0. Cabe destacar que por cada ciclo de tamaño K, el algoritmo genera $2 \times K$ ciclos iguales. En este caso donde $K = 3$, se generaron 6 ciclos. Esto representa un problema pues, por un lado, se está gastando recursos en generar varias veces un mismo ciclo, y por el otro, se gastarán más recursos en elegir tan solo uno de ellos.

4.1.2 Normalización de ciclos

Para resolver este problema, lo primero que se hizo fue normalizar los ciclos. Esto es, establecer una convención que nos permita identificar unívocamente cualquier ciclo.

La convención adoptada fue la siguiente:

1. Se elige como nodo inicial, el de menor valor posible. A este nodo lo llamaremos cabeza o head.
2. El segundo nodo elegido, adyacente al nodo inicial, deberá ser el de menor índice entre los dos nodos adyacentes

Para el ejemplo anterior, figura 4.1.1.2, al aplicar la primer condición, nos quedaron el ciclo 3 y 6, dado que el nodo cero es el menor posible. Entonces, al aplicar la segunda condición, nos quedó como resultado el ciclo 3, de modo que el mínimo entre el nodo 3 y el nodo 5, es 3. Por lo tanto, el ciclo se genera en el orden 0,3,5 y 0. De ese modo, para referenciar unívocamente éste ciclo diremos que es el (0,3,5). Omitimos la repetición del nodo inicial al final para simplificar.
Con esta definición, el listado de los ciclos de la figura 2.1.3 está dado por:

Nro.	Ciclo	Costo
1	**[2, 6, 4, 8,]**	**4687**
2	[2, 4, 6,]	3936
3	[2, 4, 8,]	1223
4	[0, 4, 3, 5,]	5625
5	**[0, 4, 7,]**	**3476**
6	[1, 6, 9,]	1645
7	[1, 5, 9, 6,]	2824
8	[1, 6, 7,]	1810
9	**[0, 1, 6, 7,]**	**3170**

Nro.	Ciclo	Costo
10	**[1, 7, 6, 9,]**	**1967**
11	**[0, 3, 5,]**	**5438**
12	**[2, 4, 3, 8,]**	**1588**
13	[0, 4, 6, 7,]	4140
14	[1, 6, 4, 7,]	4676
15	[0, 1, 9, 5,]	3152
16	[3, 4, 8,]	1423
17	[4, 6, 7,]	3584
18	[0, 1, 7, 4,]	4552

Nro.	Ciclo	Costo
19	[0, 3, 8, 4,]	3570
20	[0, 1, 7,]	2774
21	[0, 5, 1, 7,]	3231
22	[0, 3, 4,]	3563
23	**[1, 5, 9,]**	**1479**
24	[2, 4, 7, 6,]	3990
25	[0, 1, 5,]	2893
26	[0, 1, 5, 3,]	6201
27	[0, 3, 4, 7,]	4705

Tabla 4.1.2.3 – Ciclos de la figura 2.1.3
con sus costos asociados

Notar que los ciclos nro. 1,5,9,10,11,12 y 23, subrayados y resaltados en colores en la tabla anterior 4.1.2.3, conforman el BCC que se muestra en la figura 2.1.4 de la sección 2.2. Como corolario de este punto, diremos primeramente que, la primer condición establece el *nodo inicial* y, la segunda condición *la dirección* en la que se genera el ciclo.

4.1.3 Alternativa 1: Generando todos los ciclos

El próximo paso fue proponer un algoritmo que construya los ciclos en forma normalizada, evitando así generar ciclos repetidos que luego hubiera que eliminar. Para ello nos basamos en el algoritmo propuesto por Liu H. y Wang J. [11], cuya idea central es la siguiente: sea $G=(V,E)$ un grafo, donde V son sus nodos y E sus ejes. Por cada nodo $v \in V$, generamos caminos de longitud K, donde el nodo v conforma la cabecera (head), y la cola (tail) va a ir variando en función de los distintos nodos que se van agregando a este camino. La condición que se tiene que cumplir, es que el nodo que se va agregando a la cola del camino, siempre debe ser mayor a la cabeza. Este algoritmo se expone en la figura 4.1.3.4, y esta condición está remarcada con **(***)**.

En el ejemplo de la figura 4.1.1.2, vemos cómo los ciclos 1,2,4 y 5 no cumplen con esta condición, y los ciclos 3 y 6 sí la cumplen. Por ejemplo, al intentar construir el ciclo 1: [3,5,0], donde 3 es la cabecera, ocurrió lo siguiente: primeramente agregamos el nodo 5, conformando el camino [3,5]. Ahora la cola es el nodo 5 y cumple con la condición de que la cabecera es menor. Luego, al agregar el nodo 0, vemos que éste es menor que la cabecera, y esto no puede ocurrir. De esta forma, esta opción queda descardada.
Para el caso del ciclo 3: [0,3,5], tenemos como cabecera el nodo 0. Entonces agregamos los nodos 3 y 5 que cumplen con ser mayores que la cabecera. Luego, nos volvemos a topar con el nodo 0, cerrando de esta manera el ciclo.

Este algoritmo fue tomado de Liu H. y Wang J. [11], y nosotros implementamos una versión recursiva del mismo que se puede ver a continuación:

```
AlgoritmoGeneradorDeCiclos( K )
   global ConjCiclo = ∅, nodeIni , K
   camino = ∅

   For each v ∈ V
      nodeIni = v
      GeneradorCiclos(v,0,null)
   end for

return ConjCiclo
```

```
GeneradorCiclos(nodoU, nivel, camino)
   if(nodoU.estado ≠ VISITADO) ∧ (nivel<K) then
      Camino.addNode(nodoU)
      nodoU.estado = VISITADO
      nivel = nivel + 1

      for each (nodoSuc ∈ nodoU.Vecinos V)
         if(nodeIni.valor ≤ nodoSuc.valor) then   (***)
            GeneradorCiclos(nodoSuc, nivel, camino)
         end if
      end for

      nodoU.estado = NO_VISITADO
      nivel = nivel - 1
      camino.removeLast()
   else
      if(nodoU = nodeIni) ∧ (3 ≤ nivel ≤ K) then
         camino.addNode(nodeIni);
         ConjCiclo.add(CrearCiclo(camino))
         Camino.removeLast()
      end if
   end if
```

Figura 4.1.3.4 – Este algoritmo genera todos los *K-ciclos*

Con este algoritmo, logramos generar todos los ciclos posibles de tamaño menor o igual que *K*, normalizados. Su complejidad en el peor caso es $O\left(m \cdot d(V)_{\max}^{K-1}\right)$, donde $d(V)_{\max}$ es el mayor grado del nodo del grafo.
Cabe destacar que este algoritmo genera los ciclos sin tener en cuenta la dirección en la que los construye, es decir, omite la condición 2 expuesta en la sección 4.1.1.2. Para el ejemplo 4.1.1.1, el algoritmo genera los ciclos 3 y 6, reduciendo la cantidad de $2 \times K$ a 2. Como mejora, se puede implementar la condición 2 en el algoritmo para generar sólo el ciclo normalizado (en este ejemplo el ciclo 3). De momento, nuestras estructuras de datos se ocuparan de filtrar al ciclo 6, aplicando la condición 2.

4.1.4 Alternativa 2: Generando una base de ciclos

Hasta aquí logramos construir de manera eficiente todos los ciclos de tamaño menor o igual a K. En esta sección presentamos una propuesta de D. Hochbaum y E. Olinick, cuyo objeto es armar una cantidad limitada de ciclos estratégicos.

Ellos aplican el algoritmo de Horton's modificado [8], donde crean una base de ciclos de tamaño a lo sumo $n \times m$. La forma de hacerlo es la siguiente:

```
GeneradorDe_K-CycleBasis( K )
  ConjCiclo = ∅

  for each  (i,j) ∈ E
    for each  v ∈ V \{i,j}
      l = ∞
      for x = 1 to K-1
        Encontrar el camino mínimo entre v e i,  P_vi  y de tamaño x
        Encontrar el camino mínimo entre j y v,  P_jv  y de tamaño (k-1)-x
        If ∃P_vi P_jv then
           C = P_vi ∪ {(i, j)} ∪ P_jv
           If Σ_{e∈C} c(e) < l then
              C_vij = C
              l = Σ_{e∈C} c(e)
           end if
        end if
      end for
      if l ≠ ∞ then // es decir, existe un K-path entre i y j contiene a v
        ConjCiclo=ConjCiclo ∪ C_vij
      end if
    end for
  end for

return ConjCiclo
```

Figura 4.1.4.5 – Este algoritmo construye una base de K-ciclos

Esto es, para todo eje (i, j) y nodo v, crean ciclos de manera que exista un camino mínimo entre i y v, y otro entre v y j. El ciclo formado por la unión de estos dos caminos más el eje (i, j), cumple con ciertas condiciones que hacen que sea un ciclo de mínimo costo (MCCB. minimum-cost cycle basis [8]). Este procedimiento encuentra el *K-Cycle* de mínimo costo o determina que no existe, para el eje (i, j) y el nodo v. Su complejidad en el peor caso es $O\left(m \cdot n \cdot (k-1) \cdot d(V)_{\max}^{K-1}\right)$, donde $d(V)_{\max}$ es el mayor grado del nodo del grafo.

El objetivo es formar la base de ciclos, que tienen la propiedad de ser mínimos, y de poder generar a los otros restantes. De esta manera, obtenemos un conjunto acotado de a lo sumo $n \times m$ ciclos, donde todos ellos conforman una base. Además, reducimos el espacio global de soluciones, a nuestro entender de manera estratégica. Como se verá en la sección 6, esto tiene un impacto positivo en los casos medianos y grandes, y un impacto negativo en los casos chicos.
Por último, hay que remarcar que esta solución no genera los ciclos normalizados, sino que genera a cada uno con sus $2 \times K$ repeticiones, como se ve en el ejemplo 4.1.1.2. Sin embargo, las estructuras de almacenamiento implementadas, que veremos en la sección 10.1, serán las encargadas de filtrar los ciclos redundantes en forma eficiente.

4.2 Fase 2: Tabu Search BCCP

En esta sección detallaremos el algoritmo Tabú Search propuesto. Antes de embarcarnos en dicha tarea, daremos algunas definiciones preliminares y conceptos de Tabú Search aplicados a nuestro problema específico.

4.2.1 Definiciones Tabú Search

Región

Es el conjunto de soluciones posibles y la integran cubrimientos formados por ciclos de tamaño menor o igual a K, generados en la primer etapa por los algoritmos de las secciones 4.1.3 y 4.1.4. En esta región tendremos la posibilidad de prohibir ciclos en forma temporal, definitiva, o establecerlos como parte de la solución buscada.

Función objetivo

Es la suma de los costos de los ciclos que forman el cubrimiento. Éstos a su vez están compuestos por la suma de los costos de sus respectivos ejes.
Sea $G=(V,E)$ un grafo, y S una solución BCC de G, y C un ciclo cualquiera que pertenece a la solución S; luego tenemos que:

$$f=\sum_{C\in S} S(C)\text{, donde } S(C) \text{ es el costo del ciclo } C\in S$$

$$C=\sum_{e\in C} C(e)\text{, donde } C(e) \text{ es el costo del eje } e\in C$$

Solución inicial

Se construye a partir de un algoritmo goloso que va agregando ciclos de menor a mayor costo, siempre y cuando, cada nuevo ciclo que se agrega cubre al menos un eje del grafo. Las soluciones con las que trabajaremos son siempre minimales, es decir, al eliminar un ciclo, siempre se va a descubrir al menos un eje diferente.

```
 1 ConjCiclos algoritmoGoloso(Region kCiclos)
 2
 3   ConjCiclos solInicial = ∅
 4
 5   While(Not solInicial.EsBCC())
 6
 7      tmpCiclo=kCiclos.sacarMínimo()  //Obtiene el ciclo de menor costo
 8                                      //y lo elimina del conjunto kCiclos
 9      If( Grafo.CuantosEjesCubre(solInicial,tmpCiclo)>0) then
10         solInicial.Add(tmpCiclo)
11      End if
12
13   End while
14
15 return tmpCiclo
```

Figura 4.2.1.1 – Algoritmo Goloso empleado en la construcción de la solución inicial

Movimiento tabú

Dada una solución factible para un grafo G, obtenemos un eje e_1 cualquiera de G, y eliminamos todos los ciclos de la solución que atraviesan e_1. Como la solución es minimal, por cada ciclo eliminado, se descubre al menos un eje.

Por ejemplo, si tomamos la solución propuesta en la figura 2.1.4, y eliminásemos el eje $(1,9)$, obtendríamos el siguiente grafo:

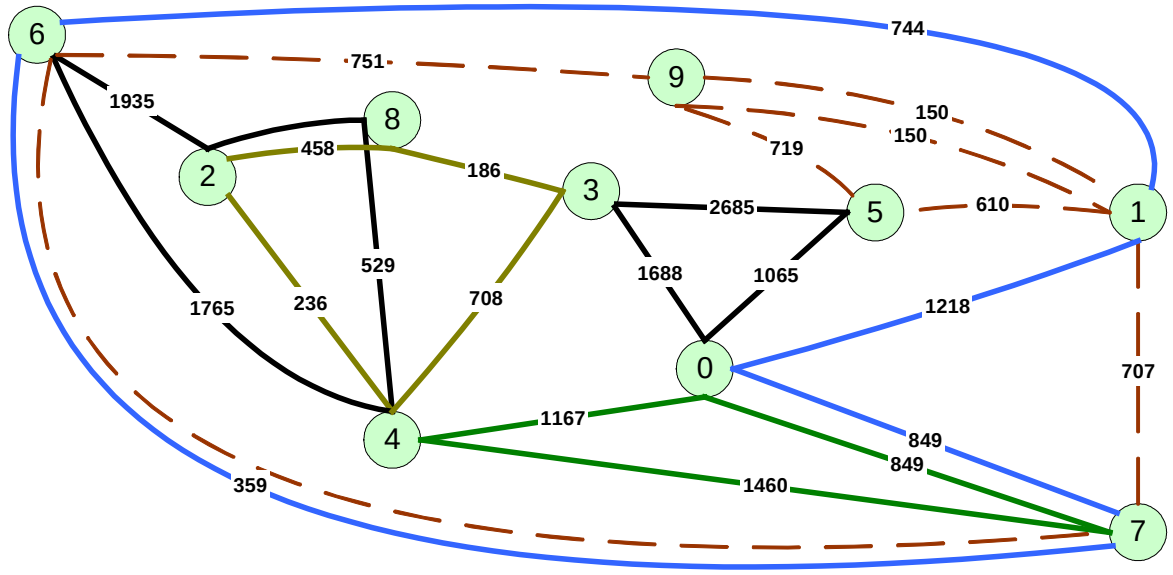


Figura 4.2.1.1 – Cubrimiento de la figura 2.1.4 sin los ciclos 10 y 23

Los ciclos **10** **[1, 5, 9,]** y **23** **[1, 7, 6, 9,]** han sido eliminados, dejando al descubierto los ejes $\{(1,5)(1,7)(1,9)(5,9)(6,9)\}$, donde $(1,9)$ es el eje elegido por el Movimiento.
Luego, la reconstrucción de esa solución se hace mediante un algoritmo de backtracking, que explorará todos los ciclos que contengan a los ejes descubiertos. Estos ciclos se detallarán a continuación. El algoritmo de backtracking tiene limitada la cantidad de iteraciones, rompiendo así con la complejidad exponencial que posee. En la sección 5.1 explicaremos más en detalle su funcionamiento.

Sea R la región compuesta por todos los ciclos generados en la primer etapa del algoritmo, $R(e_{i,j})$ son los ciclos de la región que contienen al eje (i, j).
En nuestro ejemplo, tenemos:

Nro.	$R(e_{1,5})$	Costo
7	[1, 5, 9, 6,]	2824
21	[0, 5, 1, 7,]	3231
23	**[1, 5, 9,]**	**1479**
25	[0, 1, 5,]	2893
26	[0, 1, 5, 3,]	6201

Nro.	$R(e_{1,7})$	Costo
8	[1, 6, 7,]	1810
10	**[1, 7, 6, 9,]**	**1967**
14	[1, 6, 4, 7,]	4676
18	[0, 1, 7, 4,]	4552
20	[0, 1, 7,]	2774
21	[0, 5, 1, 7,]	3231

Nro.	$R(e_{5,9})$	Costo
7	[1, 5, 9, 6,]	2824
15	[0, 1, 9, 5,]	3152
23	**[1, 5, 9,]**	**1479**

Nro.	$R(e_{1,9})$	Costo
6	[1, 6, 9,]	1645
10	**[1, 7, 6, 9,]**	**1967**
15	[0, 1, 9, 5,]	3152
23	**[1, 5, 9,]**	**1479**

Nro.	$R(e_{6,9})$	Costo
6	[1, 6, 9,]	1645
7	[1, 5, 9, 6,]	2824
10	**[1, 7, 6, 9,]**	**1967**

Figura 4.2.1.2 – Ciclos que contienen
a los ejes $\{(1,5)(1,7)(1,9)(5,9)(6,9)\}$

Donde $\#(R(1,5))=5$, $\#(R(1,7))=6$, $\#(R(1,9))=4$, $\#(R(5,9))=3$ y $\#(R(6,9))=3$

Al efectuar el **Movimiento** propuesto, la cantidad de soluciones vecinas potenciales, estará dada por:

$$\#(R(1,5))\times\#(R(1,7))\times\#(R(1,9))\times\#(R(5,9))\times\#(R(6,9))=5\times6\times4\times3\times3=1080$$

O sea que tenemos un **vecindario** de 1080 soluciones potenciales, y nuestro algoritmo de backtracking explora a todas ellas para reconstruir la solución, siempre y cuando no supere la cota máxima establecida. Cabe destacar que este cálculo no es correcto, porque presume que los ciclos de cada subconjunto son distintos entre sí, cuando en realidad algunos de ellos no lo son. Por simplicidad se adoptó este mecanismo para establecer una cota máxima.
Otro punto interesante, es que este movimiento puede llegar a descubrir los ejes que están a $K/2$ de distancia respecto del eje seleccionado $(1,9)$, es decir, tenemos un impacto de $K/2$ ejes adyacentes a la redonda. Por ser un impacto acotado en una zona determinada a partir de un eje seleccionado, lo denominamos ***movimiento en profundidad***.

Lista Tabú
En la lista tabú vamos a prohibir *ciclos*. La idea es sacar un ciclo de la solución en construcción, o de los candidatos disponibles en una vecindad, y prohibirlo por una cantidad arbitraria de iteraciones.
Esta lista funciona como una cola FIFO de longitud acotada.
Con los ensayos realizados en la sección 6.2, se determinó que 100 es la longitud apropiada para grafos y K grandes.

Criterio de parada
El criterio de parada entra en vigencia cuando el algoritmo no mejora la solución obtenida en 100 iteraciones de un total máximo de 300.

Soluciones Elite
Tienen características que hacen que se destaquen de otras soluciones. Este punto no fue implementado.

Intensificación
Para la intensificación, se implementó una función de backtracking, donde sus iteraciones están limitadas a un tamaño fijo. La misma se describirá en detalle en la sección 5.1. La idea central es explorar la mayor cantidad de vecinos posibles, dentro de las iteraciones permitidas, y elegir al mejor.

Los movimientos realizados, para el cambio de vecindad, apuntan a habilitar una zona, que luego será aprovechada por la intensificación. Debido a que muchas zonas descubiertas admiten una cantidad elevada de solución posibles, la cantidad máxima de iteraciones apunta a limitar esa búsqueda, para que no se prolongue en el tiempo.

Diversificación
La elección de ciclos al azar de la solución actual para su prohibición cuando se cambia de vecindad, contribuye a la diversificación del algoritmo. La lista tabú y la definición de vecino también sirven para diversificar.
En los ensayos se observa que hay muy pocas soluciones repetidas a lo largo de la ejecución del algoritmo en los grafos grandes y K grandes. No obstante, para los grafos pequeños con K chicos, se advierte una significativa cantidad de soluciones repetidas. A pesar de ello, tal como se verá más adelante, esto no se refleja en el resultado. Si bien en escenarios pequeños y/o medianos se producen soluciones repetidas, se cuentan con muy buenos resultados.

4.2.2 Algoritmo Tabú

A partir de las características de Tabú Search antes mencionadas, proponemos el siguiente algoritmo:

```
 1 ConjCiclos TabuSearchBCCP(K)
 2    Region R = Generador_De_Ciclos(K)
 3    SolActual = algoritmoGoloso(R)
 4    i=0
 5    //Pasos de Tabú Search
 6    While(mejorSol no mejore por X_ITER iteraciones and i<MAX_ITER)
 7
 8       e = grafo.getIterNextEje()
 9
10       solActualMv = Movimiento(solActual,e)  //Elimina todos los ciclos que
                                                //contienen el eje e en la sol. actual
11       //Prohíbo un ciclo de la solución Actual, de los que fueron
12       //eliminados en el movimiento tabú
         ListaTabu.add(solActual.getCicloAlAzar(solActual-solActualMv))
13
14       solActual =Intensificar(solActualMv,R)  aplico el backtracking  limitado
15
16       If(solActual < mejorSol) then
17          mejorSol = solActual
18       End If
19       i=i+1
20    End while
21    Return mejorSol
```

Figura 4.2.2.1 – Algoritmo Tabú Search propuesto

En una primera etapa, generamos los ciclos de tamaño menor a igual a K (línea 2 de la figura 4.2.2.1). Esto lo hacemos con uno de los dos algoritmos propuestos en 4.1.3 y 4.1.4.
Luego, construimos una solución inicial con un algoritmo goloso, que agrega ciclos de menor a mayor costo, siempre que cada ciclo a agregar cubra al menos un eje del grafo (línea 3).
En la línea 6 comienzan las iteraciones del Tabú Search. Estas se interrumpirán si la mejor solución *MejorSol* no mejora en una cantidad determinada de iteraciones (*X_ITER*). Además la cantidad máxima de iteraciones no podrá superar el valor *MAX_ITER*. El mejor valor hallado en nuestros ensayos para *X_ITER y MAX_ITER* es de 100 y 300 respectivamente.

En cada iteración realizamos los siguientes pasos:

- Obtenemos un eje **e** del grafo, es decir, en cada nueva iteración del Tabú obtenemos un eje nuevo del grafo (línea 8). Cuando hubimos recorrido todos los ejes en las sucesivas iteraciones, volvemos a obtener el eje inicial.
- Aplicamos un movimiento tabú (línea 10). Esto es, eliminamos todos los ciclos que atraviesan el eje **e**, de manera que éste y eventualmente otros ejes, quedasen al descubierto en la solución actual.
- Prohibimos un ciclo al azar de la solución actual *solActual*, de los eliminados en el movimiento tabú (línea 12).
- Ejecutamos la función de intensificación (línea 14), la cual consiste en reconstruir la solución actual *solActual*, con un algoritmo de backtracking limitado en sus iteraciones, que detallaremos en 5.1.
- Almacenamos *solActual* sólo si ésta mejoraba respecto de nuestra mejor solución mejor*Sol* (líneas 16,17 y 18).
- Volvemos al inicio de otra iteración tabú.

5 Otras variantes

En la presente sección se explican algunas variantes probadas para el mismo problema que no tuvieron resultados adecuados. A pesar de ello, las mismas contribuyeron directa e indirectamente al desarrollo de nuestro algoritmo Tabú Search.
La primera variante conforma un algoritmo de backtracking personalizado, cuyo objetivo es poder emplearlo en casos chicos, y como función de intensificación en el Tabú Search. Como veremos más adelante, también es utilizado por el algoritmo de Búsqueda Local.
La segunda de ellas, es un algoritmo de Búsqueda Local que se implementó previo al Tabú Search, a modo de prueba. Con algunas mejoras y correcciones introducidas, y como se verá en la próxima sección 6, éste obtuvo resultados respetables, y en algunos casos hasta sorprendentes.

5.1 Backtracking

El doble propósito de este algoritmo es emplearlo para casos chicos con el objeto de encontrar el mínimo global, y de ese modo poder analizar dichos resultados respecto de los generados por nuestro tabú search. Además se pretende poder compararlo con la cota mínima encontrada por la implementación del cartero chino. Asimismo, se pretende implementar una función de intensificación que encuentre muy buenos resultados (no estrictamente el óptimo) en una vecindad dada, en un tiempo adecuado. Dado que, las funciones de intensificación y exploración de vecindades que probamos no arrojaron los resultados esperados, consideramos que una función de backtracking, adaptada a las necesidades puntuales de nuestro Tabú Search, y cumpliendo con los requisitos de tiempos estipulados, es lo más provechoso para el algoritmo.

Por dichas razones, se implementó el siguiente algoritmo recursivo.

```
1  ConjCiclos Intensificar(ConjCiclos solBase, Region R)
2
3     pilaEjes = ∅
4
5     for each  e ∈ E
6        If not(solBase.IsCover(e)) then
7           PilaEjes.Push(e)
8        End if
9
10    BackTracking(PilaEjes, SolBase, R)
11
12    Return SolBase
```

```
13 BackTracking(PilaEjes PEjes, ConjCiclos solBase, Region R)
14
15    ConjCiclos CiclosE  = ∅, MejorSol = ∞
16
17    Edge e = PEjes.Pop()
18
19    If not(solBase.IsCover(e)) then  //No es redundante preguntarlo porque
20                                     //Un ciclo puede cubrir más de un eje no cubierto
21       CiclosE = R.getCiclos(e)
22
23       for each c ∈ CiclosE and I<MAXITER   //Cota máxima de valor 1.000.000
24          I++     //en la cual contabilizo cada ciclo que agrego a la solución.
25          SolBase.add(c)
```

```
26
27
28          If(SolBase < MejorSol ) then
29            If(SolBase.IsBcc())then
30              If(SolBase < MejorSol) then
31                MejorSol = SolBase
32              End if
33            Else
34              BackTracking(PEjes, solBase, R)
35            End if
36          Else
37            CortoRama++
38          End if
39
40          SolBase.remove(c) I++
41
42        End for
43      Else
44        BackTracking(PEjes, solBase, R)
45      End if
46
47      PEjes.Push(e)
```

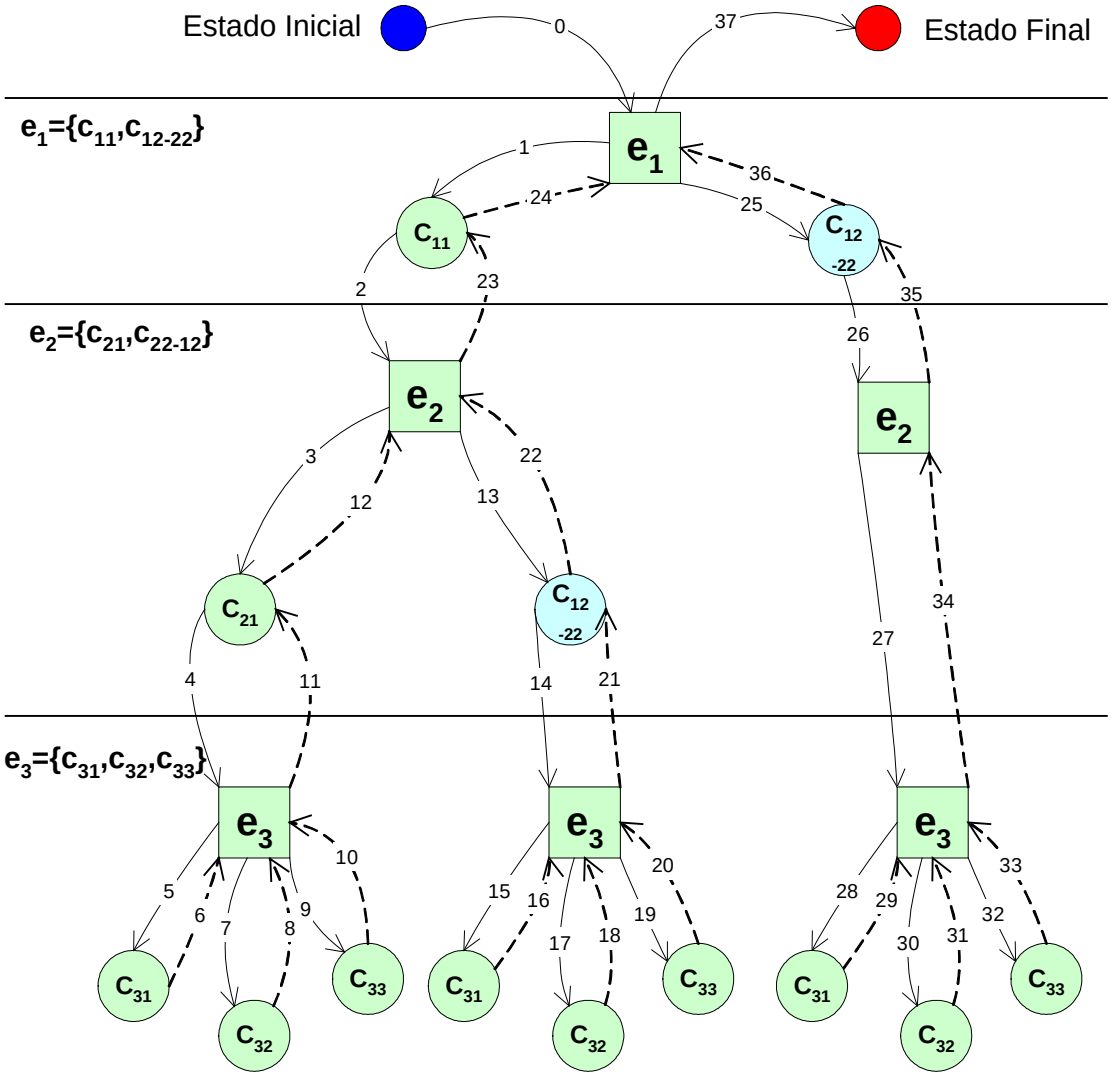
Figura 5.1.1 – Algoritmo de Backtracking

Conceptualmente, el algoritmo opera de la siguiente forma:

1. Obtiene los ejes *descubiertos* de la solución inicial `solBase`, que pueden ser todos, en el caso de una solución vacía (búsqueda exhaustiva); o sólo algunos, en el caso de una solución parcial (intensificación en tabú search).
2. Por cada eje descubierto, obtiene los respectivos ciclos que contienen a dichos ejes y efectúa el producto cartesiano entre los ciclos de cada uno de ellos. Como se verá en el ejemplo siguiente, se observa que los ejes e_1, e_2 y e_3 quedan descubiertos. Cada uno de ellos tiene asociados los siguientes ciclos: $e_1=\{c_{11},c_{12\text{-}22}\}$, $e_2=\{c_{21},c_{12\text{-}22}\}$ y $e3=\{c_{31},c_{32},c_{32}\}$. Entre estos tres conjuntos, se realizan todas las combinaciones posibles. En el caso de la función de intensificación, la cantidad de combinaciones posibles está limitada por una cota máxima.
3. Puede ocurrir que un ciclo cubra más de un eje, es decir, que el ciclo esté en más de un conjunto. En el ejemplo, esto le ocurre al ciclo $c_{12\text{-}22}$, y como se observa en el paso 26 y 27 de la siguiente figura 5.1.2, al detectar que el eje e_2 ya fue cubierto por el ciclo $c_{12\text{-}22}$, saltea deliberadamente todos los ciclos de e_2, y continúa operando para cubrir e_3. Por esta razón, cada vez que se quiere cubrir un nuevo eje, se pregunta si antes no fue cubierto por otro ciclo. (Algoritmo 5.1.1, línea 19).
4. Cada vez que el backtracking se ejecuta en el contexto de la intensificación del tabú, la mejor solución establecida en `MejorSol`, inicialmente es vacía y su valor es infinito. Esto hace que el algoritmo encuentre la mejor solución en la vecindad aplicada, sin saber que ocurrió en otras intensificaciones, en otras vecindades. De este modo, permite al Tabú Search aceptar soluciones peores. Cabe destacar que, cuando nos referimos a la mejor solución de la vecindad, esto es siempre y cuando no supere la cota máxima establecida para la cantidad de combinaciones permitidas. En caso de realizar todas las combinaciones permitidas, no se garantiza que la solución encontrada sea la mejor de la vecindad.
   Para la búsqueda exhaustiva, se ejecuta el backtracking con una solución inicial vacía, y con `MejorSol` vacía y su valor es infinito. Al final de la búsqueda, se logra la mejor de todas las soluciones.
5. El algoritmo va almacenando la mejor solución encontrada en `MejorSol`. Entonces, al ir construyendo una nueva solución, si detecta que esta supera en costo de `MejorSol`,

elimina el último ciclo agregado y pasa al siguiente. En otras palabras, corta la rama del árbol de exploración. Ver Algoritmo 5.1.1, líneas 28, 37 y 40.

En el siguiente ejemplo, se observa el comportamiento del algoritmo. Las flechas delineadas indican el avance del mismo, y las punteadas el retroceso. La numeración de las flechas indica el orden en que opera. Se parte del estado inicial en 0 y luego se avanza según los pasos 1, 2, 3, 4, etc.



Ejemplo 5.1.2 – Comportamiento del algoritmo Backtracking de 5.1.1

Como antes se mencionó, el algoritmo en el paso 25 adiciona el ciclo $c_{12\text{-}22}$, - resaltado en color celeste -, y éste cubre los ejes e_1 y e_2. Luego, en el paso 26 al detectar que el eje e_2 está cubierto, en este caso por el ciclo $c_{12\text{-}22}$, salta al siguiente eje e3, mediante el paso 27.

Las características sobresalientes de dicho algoritmo son:

- Se puede partir de una solución vacía o parcial. El primer caso sirve para realizar ensayos en grafos chicos (hasta n=15 y K=4). El segundo es para aplicar la función intensificación en el algoritmo de tabú search, a una solución en construcción con una vecindad por explorar. Entonces tenemos que una solución es un conjunto de K-ciclos tal que cubren

todos los ejes del grafo, es decir, conforman un BCC. Una solución vacía es cuando el conjunto de ciclos es vacío, y por lo tanto, ninguno de los ejes del grafo es cubierto por ningún ciclo. Una solución parcial es cuando ya contamos con algunos ciclos tal que cubren algunos ejes del grafo, pero no todos.

- El algoritmo está limitado por una cota máxima en las iteraciones que realiza. Dichas iteraciones se miden en función de los ciclos que se agregan a la solución que se va construyendo (Algoritmo 5.1.1, línea 24). La cota que obtuvo mejores resultados es de 1.000.000 (ver sección 6.2). Esta cantidad suele ser aprovechada por la intensificación del Tabú en grafos y K grandes. Si observamos el Ejemplo 5.1.2, la cantidad de iteraciones efectuadas por el backtracking es 13.
  En principio se limitaron las iteraciones respecto la cantidad de soluciones generadas (un máximo de 100). Pero una vez hallado el mínimo local, se imposibilita construir una nueva solución, dado que la poda de ramas del backtracking es aplicada cuando la solución en construcción supera el costo de la mejor encontrada hasta ese momento. Entonces, en ese caso la condición de corte no se efectuaba y el algoritmo nunca concluía en grafos y K grandes (por quedarse explorando todos los casos posibles). Por esta razón, se adoptó una cota en función de la cantidad de ciclos empleados en la construcción de soluciones y no en la cantidad de soluciones construidas.

5.2 Algoritmo de Búsqueda Local BCCP

Se implementó un algoritmo de búsqueda local, con algunos puntos de coincidencia con el algoritmo de Tabú Search, pero con características propias.

Los principales puntos a destacar son:

- Inicialmente emplea el mismo generador de ciclos (líneas 3).
- Aplica el mismo algoritmo greedy para la construcción de una solución inicial (línea 4).
- Emplea el mismo criterio para las iteraciones que tabú search (línea 6).

Dentro de cada iteración efectúa los siguientes pasos:

- Obtiene un eje al azar, garantizando que se van a invocar a todos. Esto difiere de Tabú Search, que obtiene los ejes en forma secuencial entre una iteración tabú y otra (línea 7).
- Utiliza la misma función de movimiento que Tabú Search. Esto es, para el eje obtenido elimina todos los ciclos de la solución actual que lo contienen (línea 9).
- Para reconstruir la solución, aplica el algoritmo de backtracking expuesto en la sección 5.1, aunque con una gran diferencia respecto de Tabú Search: toma en cuenta la mejor solución histórica almacenada en el 80% de los casos. Dicho procedimiento es realizado al azar y con probabilidad 4/5 elige la mejor solución histórica, mientras que con probabilidad 1/5 elige una solución vacía cuyo valor es infinito. De esta manera, el backtracking puede cortar muchas más ramas en el árbol de búsqueda, y además permite que la solución actual pueda empeorar en el 20% de los casos restantes (línea 11).

A continuación exponemos el algoritmo de búsqueda local:

```
 1 ConjCiclos BusqLocal(Grafo g, int K)
 2
 3   Region R = Generador_De_Ciclos(K)
 4   solActual = generarSolución(R)
 5   i=0

 6   While(mejorSol no mejore por X_ITER iteraciones and i<MAXITER)
 7     Edge e = g.getEjeAlAzar()
 8
 9     solActual = Movimiento(solActual,e)
10
11     solActual = IntensificarBusqLocal(solActual,R)
12
13     If(solActual < MejorSol) then
14          MejorSol = solActual
15     End if
16     i=i+1
17   End while
18
19   Return solActual
20
21
22
```

Algoritmo 5.2.1 – Algoritmo de Búsqueda Local

6 Resultados Computacionales

En esta sección, como primer punto, propondremos dos variantes del algoritmo Tabú Search y dos del algoritmo de Búsqueda Local. Elegiremos una de cada una de ellas, en función de la calidad de las soluciones obtenidas, tiempos de ejecución empleados y otros parámetros. Y finalmente, analizaremos los tiempos totales y los tiempos parciales respecto de la generación de ciclos.
Seguidamente, realizaremos varios ensayos con diversos parámetros para el algoritmo de Tabú Search elegido, y analizaremos los resultados obtenidos y el comportamiento de éste en diversas situaciones.
Para concluir, en el último punto, compararemos los resultados obtenidos con nuestros algoritmos previamente elegidos, respecto del trabajo de D. Hochbaum y E. Olinick [8]. El eje central de este punto es ver qué tan bueno fue nuestro algoritmo respecto de otros que resuelven el mismo problema.

Con el objeto de recrear las mismas condiciones de los ensayos realizados por D. Hochbaum y E. Olinick [8], utilizamos los mismos grafos que nos fueron facilitados por dichos autores. Estos grafos fueron generados usando la rutina *plane_miles* de *Stanford GrahpBase*, que crea grafos planares y euclideanos seleccionando ciudades al azar de la guía de *McNally & Company's Standard Highway Mileage*. Los mismos tienen 10, 15, 20, 25, 30 y 35 nodos, donde el costo de sus ejes representa la distancia en millas entre las ciudades. Nuestros algoritmos fueron probados con valores de K = 4, 5, 6, 7 y 8.
Las pruebas fueron realizadas en una PC con un procesador Intel Core 2 Duo de 2.20 Ghz, y 2Gb de memoria Ram, sistema operativo Windows XP service pack 2. La implementación del algoritmo fue realizada en el lenguaje de programación Java.
Para calcular el CPP se utilizó una implementación en Visual C++ 6.0, cuyo autor es Dominic Battré [3]. Para más detalles de la implementación, consultar el Apéndice A del presente trabajo.

6.1 Variantes del algoritmo

Los grafos utilizados en los ensayos son planares y euclideanos. La familia de grafos planares y de grafos euclideanos cuentan con 28 y 30 grafos respectivamente, que se dividen en grupos de 10, 15, 20, 25, 30 y 35 nodos. Cada grupo cuenta con 5 grafos distintos aprox.
Nuestros algoritmos fueron probados con valores de K = 4, 5, 6, 7 y 8, lo que da un total de 150 corridas por cada algoritmo, por cada familia aproximadamente.

La solución al problema del cartero chino es una cota inferior para nuestro problema. Entonces, para establecer que tan buena es una solución hallada para un grafo G, calculamos el desvío relativo respecto de la solución del cartero chino para dicho grafo G.
Sea G un grafo cualquiera, $AlgoritmoBCCP$ es una función que aplica nuestro algoritmo y devuelve el valor total de la solución encontrada, y CPP un algoritmo que calcula el cartero chino de G y devuelve el valor de la solución encontrada para CPP; luego tenemos que el desvío relativo es:

$$\frac{A\lg oritmoBCCP(G) - CPP(G)}{CPP(G)}$$

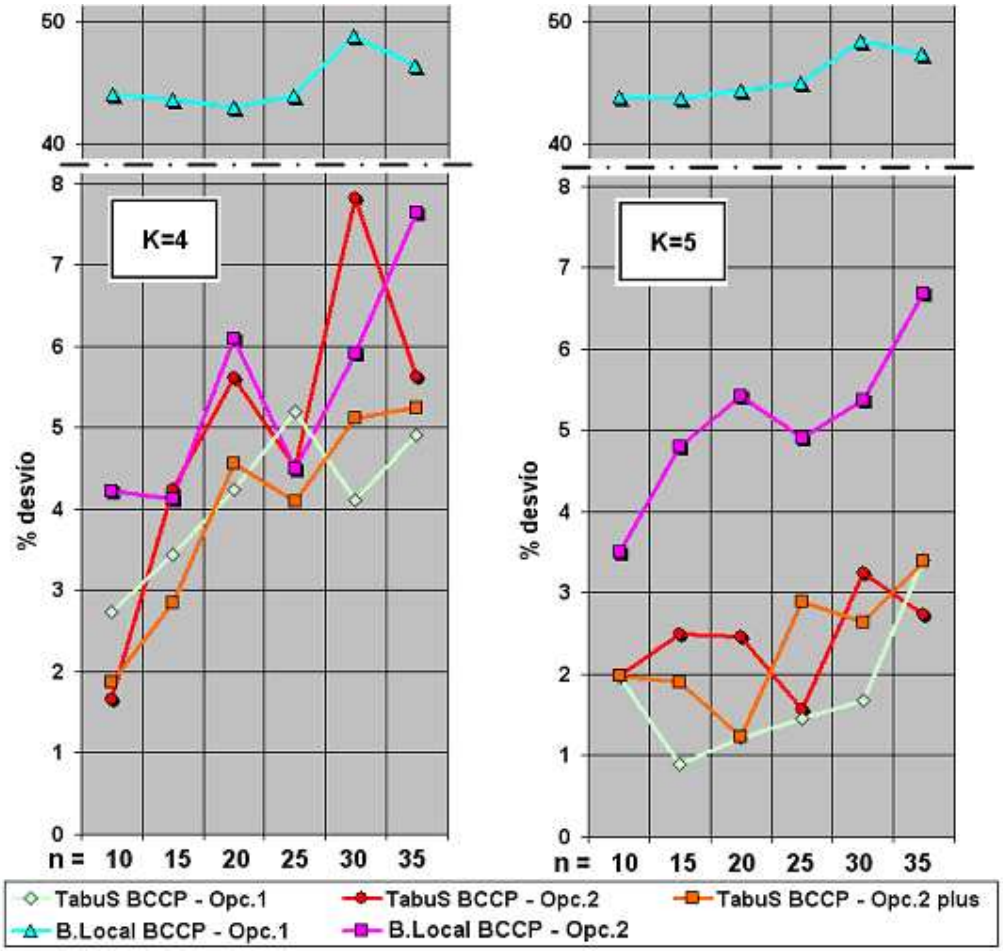
donde cero es el mejor resultado posible, dado que en ese caso se está igualando la cota inferior.
Los resultados fueron agrupados por K y por la cantidad de nodos.
En el siguiente recuadro se observan las variantes de los dos algoritmos usados en los experimentos.

Tabu Search BCCP - Opción 1: corresponde al algoritmo Tabú Search detallado en la sección 4.2.2. Opción 1 significa que utiliza la generación de ciclos de la sección 4.1.3 [11], es decir, la alternativa 1.

Tabu Search BCCP - Opción 2 / Opción 2 Plus: aplica el mismo algoritmo tabú, pero para la generación de ciclos utiliza el algoritmo de la sección 4.1.4 (alternativa 2). Ésta propone generar una base de ciclos similar a la del trabajo de D. Hochbaum y E. Olinick [8]. La variante plus significa que el backtracking que opera en la intensificación del Tabú está habilitado para realizar 25 veces más iteraciones que la opción 2.

Búsqueda Local BCCP - Opción 1: utiliza el algoritmo de búsqueda local expuesto en la sección 5.2. El generador de ciclos empleado corresponde a la sección 4.1.3 (alternativa 1).

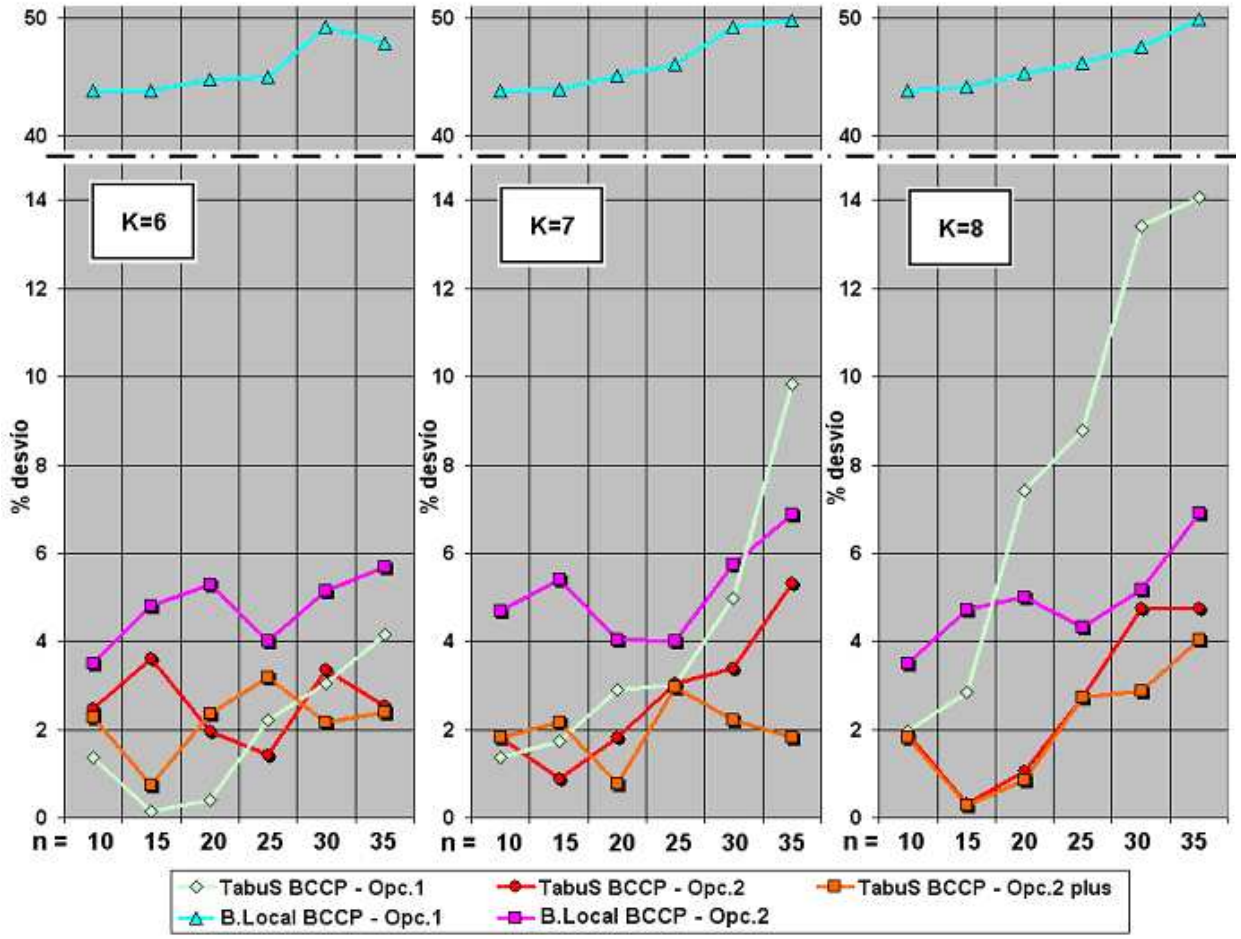
Búsqueda Local BCCP - Opción 2: ídem anterior, solo que la generación de ciclos es realizada por el algoritmo de la sección 4.1.4 (alternativa 2).



Gráficos 6.1.1 – Ensayos realizados con grafos euclideanos para K=4 y K=5

En el gráfico 6.1.1 se pueden observar dos gráficos correspondientes a ensayos hechos con K = 4 y 5. En el 6.1.2, tres gráficos correspondientes a ensayos realizados con K = 6, 7 y 8. El eje X es una variable discreta que indica la cantidad de nodos del grafo, siendo sus valores 10, 15,

20, 25, 30 y 35 nodos. El eje Y indica el porcentaje de desvío de la solución hallada respecto CPP. Cada punto del gráfico es el promedio de 5 soluciones, correspondientes a 5 grafos distintos para n=10, luego para n=15 y así; es decir, 5 grafos que tienen la misma cantidad de nodos[2].
La interpolación lineal entre los diversos puntos es sólo a fines ilustrativos, dado que como se dijo antes, X es una variable discreta. Esto nos permite visualizar mejor el comportamiento y la tendencia de los algoritmos.



Gráficos 6.1.2 – Ensayos realizados con grafos euclideanos para K=6, K=7 y K=8

En un primer enfoque, al observar la calidad de las soluciones, vemos cómo prevalece en la mayoría de los casos, y como era de esperar, el algoritmo de Tabú Search y sus variantes versus las de B. Local. Por ejemplo **B. Local BCCP - Opción 1** es el algoritmo que peores resultados tiene, y sus porcentajes de desvío respecto CPP oscilan entre el 40 y 50%, cuando el resto no supera el 15%. El algoritmo **B. Local BCCP - Opción 2**, tiene resultados mucho mejores que su predecesor (oscilan entre un 3 y 8%), aunque no logra acercarse al resto de los algoritmos de Tabú Search, a excepción de **TabúS BCCP – Opc.1**.

Se observa en los resultados que, Tabú Search BCCP Opc. 1 obtiene mejores resultados respecto de su par B. Local BCCP Opc.1; y Tabú Search BCCP Opc. 2 respecto de su homólogo B. Local Opc.2; aunque, como se observa en la tabla 6.1.3, al precio de un mayor tiempo de procesamiento invertido.

[2] Los resultados completos se encuentran en el cd adjunto. Ver Apéndice B.

Grafos			(%) Porcentajes de desvío respecto CPP					Tiempo medido en segundos				
K	# Nodos n	# Ejes Promedio	% TabuS Opc.1	% TabuS Opc.2	%TabuS Opc.2 plus	% B.Local Opc.1	% B.Local Opc.2	TabuS Opc.1	TabuS Opc.2	TabuS Opc.2 plus	B.Local Opc.1	B.Local Opc.2
4	**10**	20	2,73	**1,66**	1,87	43,98	**4,22**	0,19	**0,12**	0,16	0,06	**0,14**
4	**15**	35	3,44	**4,23**	2,85	43,54	**4,12**	0,24	**0,23**	0,18	0,07	**0,17**
4	**20**	50	4,23	**5,61**	4,55	42,92	**6,09**	2,31	**2,13**	45,73	0,08	**0,23**
4	**25**	63	5,20	**4,51**	4,10	43,92	**4,50**	2,12	**0,37**	24,66	0,11	**0,27**
4	**30**	77	4,11	**7,81**	5,11	48,77	**5,91**	0,50	**0,40**	0,39	0,12	**0,34**
4	**35**	93	4,90	**5,62**	5,25	46,32	**7,64**	0,61	**0,48**	0,43	0,16	**0,44**
5	**10**	20	1,97	**1,98**	1,98	43,84	**3,50**	3,52	**0,64**	0,94	0,08	**0,22**
5	**15**	35	0,90	**2,50**	1,90	43,67	**4,79**	18,35	**7,75**	4,35	0,13	**0,42**
5	**20**	50	1,22	**2,46**	1,23	44,32	**5,42**	20,10	**7,00**	4,73	0,22	**0,66**
5	**25**	63	1,46	**1,56**	2,90	44,94	**4,90**	14,50	**7,14**	8,10	0,26	**0,68**
5	**30**	77	1,68	**3,24**	2,63	48,40	**5,37**	19,99	**8,04**	10,72	0,31	**0,78**
5	**35**	93	3,41	**2,74**	3,39	47,30	**6,68**	20,77	**14,70**	13,67	0,39	**1,14**
6	**10**	20	1,35	**2,46**	2,26	43,84	**3,50**	18,54	**1,55**	2,24	0,16	**0,33**
6	**15**	35	0,13	**3,60**	0,75	43,83	**4,79**	133,23	**37,94**	27,92	0,44	**1,06**
6	**20**	50	0,40	**1,95**	2,37	44,74	**5,28**	181,25	**36,25**	58,31	0,63	**1,52**
6	**25**	63	2,21	**1,41**	3,17	44,98	**4,01**	137,20	**42,94**	64,78	0,74	**2,23**
6	**30**	77	3,04	**3,34**	2,15	49,26	**5,13**	151,78	**52,20**	104,69	0,93	**1,87**
6	**35**	93	4,14	**2,53**	2,39	47,81	**5,69**	193,90	**66,94**	197,79	1,29	**2,86**
7	**10**	20	1,35	**1,83**	1,83	43,84	**4,69**	118,63	**2,96**	2,15	0,32	**0,42**
7	**15**	35	1,72	**0,88**	2,15	43,86	**5,39**	450,02	**55,18**	179,10	1,59	**1,56**
7	**20**	50	2,89	**1,83**	0,77	45,12	**4,03**	457,92	**138,82**	399,41	2,34	**3,57**
7	**25**	63	3,01	**3,03**	2,96	46,03	**4,02**	420,17	**97,79**	365,00	2,82	**3,35**
7	**30**	77	4,97	**3,39**	2,23	49,22	**5,74**	410,67	**126,03**	405,22	4,04	**4,84**
7	**35**	93	9,84	**5,32**	1,82	49,73	**6,89**	488,29	**194,27**	1045,63	5,94	**9,30**
8	**10**	20	1,97	**1,90**	1,83	43,84	**3,50**	186,00	**3,13**	2,70	0,56	**0,37**
8	**15**	35	2,83	**0,31**	0,27	44,15	**4,73**	713,53	**62,97**	74,73	6,15	**1,99**
8	**20**	50	7,41	**1,04**	0,85	45,34	**4,99**	802,56	**140,22**	453,76	9,90	**5,25**
8	**25**	63	8,77	**2,74**	2,72	46,10	**4,31**	775,49	**176,64**	768,02	11,33	**8,73**
8	**30**	77	13,42	**4,74**	2,87	47,58	**5,18**	864,62	**184,74**	917,61	15,54	**8,56**
8	**35**	93	14,06	**4,76**	4,04	49,83	**6,92**	961,65	**262,24**	1503,12	22,62	**54,31**
Promedio			3,96	**3,03**	2,51	45,70	**5,06**	252,29	**57,73**	222,87	2,98	**3,92**
Total			118,76	**90,98**	75,19	1371,0	**151,93**	7568,63	**1731,81**	6686,24	89,31	**117,61**

Tabla 6.1.3 – Resultados detallados de los gráficos 6.1.1 y 6.1.2

En la tabla 6.1.3 se muestran los porcentajes de desvío de las soluciones encontradas respecto de CPP, correspondientes a los gráficos de 6.1.1 y 6.1.2. También se incluyen los tiempos promedio para dichos ensayos, medidos en segundos.
En la anteúltima fila denominada **promedio**, podemos observar el comportamiento y tiempo promedio de los algoritmos para esta familia de grafos, y para K = 4, 5, 6, 7 y 8. Por ejemplo **Tabú Search BCCP – Opc.2** tiene en promedio 3.03% de desvío respecto del cartero chino, y 57.73 segundos de tiempo promedio. En la última fila, que figura con el nombre **Total**, es similar a la anterior, sólo que se calcula la suma total de los porcentajes de desvío y tiempos involucrados. Este total sería algo así como la superficie debajo de la curva que trazan los algoritmos en los

gráficos 6.1.1 y 6.1.2, y nuestro objetivo es minimizarlo. El resultado de los 150 ensayos que se realizaron para cada algoritmo, lo podemos cuantificar, para luego poder compararlo, en tan sólo dos valores: porcentaje total y tiempo total empleado.

En un segundo enfoque, analizamos el impacto que tiene el generador de ciclos en los algoritmos propuestos. Al comparar algoritmos idénticos, como ser **Tabú Search BCCP – Opc.1** versus **Tabú Search BCCP – Opc.2**, y **B. Local BCCP – Opc.1** versus **B. Local BCCP – Opc.2**; la única diferencia radica en el conjunto de ciclos de entrada (input). Luego, al observar los resultados, vemos que éstos son muy disímiles.
Para el caso de **B. Local BCCP – Opc. 1** y **B. Local BCCP – Opc. 2**, se observa una diferencia abismal en la calidad de la solución (5,06% en promedio contra 45,7%), aunque los tiempos de ejecución son bastantes similares (3,92 segundos en promedio vs 2,98). Sin embargo, en ambos casos se observan cierta estabilidad en la calidad de las soluciones, aunque con una leve declinación a medida que la cantidad de nodos y K aumentan.

En cuanto a **Tabú Search BCCP – Opc. 1**, vemos cómo se degrada significativamente la calidad de las soluciones y los tiempos de ejecución empleados, a medida que la cantidad de nodos y el K aumentan. En particular, esto se observa cuando K=7, a partir de n= 25; y cuando K=8, a partir de n=20. Para K=4, 5 y varios casos de 6.
TabúS BCCP – Opc. 1 obtiene los mejores resultados, pero para K=7 y 8 esto se revierte drásticamente, y la tendencia indica que podría seguir empeorando mucho más. Notar que el porcentaje promedio tiene muy poca diferencia (3,96% vs. 3,03%), pero el tiempo empleado en **TabúS BCCP – Opc. 1** es 5 veces más tiempo que su par **TabúS BCCP – Opc. 2**, haciendo que la opción 1 sea prohibitiva.

Reflexionando sobre este punto, la opción 1 propone un algoritmo que genera todos los ciclos de tamaño menor o igual a K posibles [11]. Como se observa en esta familia de grafos en particular, y en los grafos *2 – conexos* en general, el crecimiento de la cantidad de ciclos es exponencial a medida que el K y el tamaño del grafo aumenta. Esto produce una degradación muy acentuada en la calidad de las soluciones, y un aumento considerable en los tiempos de ejecución empleados para el algoritmo **TabúS BCCP – Opc. 1**.
Respecto de **B. Local BCCP – Opc. 1**, se observan muy malos resultados, independientemente del tamaño del grafo y del K empleado.
La opción 2 genera una cantidad considerablemente menor de ciclos (a lo sumo $n \times m$ ciclos), aunque con ciertas propiedades que garantizan el hallazgo de buenas soluciones[3]. Esto además hace que se pueda explorar más en profundidad y en menos tiempo el espacio de soluciones resultante.

En la siguiente tabla 6.1.4, se observan los tiempos totales de los distintos generadores de ciclos, y los porcentajes relativos de los mismos, y asimismo de los algoritmos de tabú search y/o búsqueda local según corresponda. En términos absolutos, la diferencia entre un generador de ciclos y otro es de tan sólo algunos segundos, en el peor de los casos (ejemplo K=8 y #nodos=35). Si bien la alternativa 1 genera más ciclos que la alternativa 2, esta última construye una base de ciclos, y por esta razón es un tanto más costosa que la primera (ver sección 4.1.3 y 4.1.4).
Lo que podemos destacar de la tabla 6.1.4, es la incidencia del generador de ciclos en la resolución completa del problema. Para el caso del Tabú Search, la alternativa 1 incide en un 0,3% en promedio vs. 2,91% de la alternativa 2. En la búsqueda local, tenemos un 4,92% de la opción 1, contra 15,12% de la opción 2. Estas diferencias se debe a que el tiempo total empleado por el algoritmo Tabú Search es ampliamente mayor al empleado en la búsqueda local, y por esta razón, difieren los porcentajes entre uno y otro.

[3] Para más detalles, ver el trabajo [7]

Grafos			Porcentaje: Generación de ciclos vs. Algoritmo Tabú Search						Porcentaje: Generación de ciclos vs. Algoritmo Greedy				Tiempos en segundos	
K	# Nodos n	# Ejes Promedio	Generador de Ciclos Opc.1	Algoritmo TabuS Opc.1	Generador de Ciclos Opc.2	Algoritmo TabuS Opc.2	Generador de Ciclos Opc.2	Algoritmo TabuS Opc.2 plus	Generador de Ciclos Opc.1	Algoritmo B.Local Opc.1	Generador de Ciclos Opc.2	Algoritmo B.Local Opc.2	Generador de Ciclos Opción 1	Generador de Ciclos Opción 2
4	10	20	4,9%	95,1%	8,0%	92,0%	5,9%	94,1%	10,2%	89,8%	10,9%	89,1%	0,0078	0,0114
4	15	35	1,2%	98,8%	5,4%	94,6%	6,9%	93,1%	4,5%	95,5%	9,0%	91,0%	0,0030	0,0135
4	20	50	0,0%	100,0%	1,2%	98,8%	0,1%	99,9%	3,6%	96,4%	10,8%	89,2%	0,0015	0,0261
4	25	63	0,0%	100,0%	13,6%	86,4%	0,2%	99,8%	5,5%	94,5%	14,9%	85,1%	0,0030	0,0427
4	30	77	0,6%	99,4%	15,7%	84,3%	16,2%	83,8%	5,1%	94,9%	21,1%	78,9%	0,0047	0,0659
4	35	93	1,0%	99,0%	20,3%	79,7%	21,6%	78,4%	7,8%	92,2%	22,7%	77,3%	0,0093	0,0969
5	10	20	0,0%	100,0%	1,4%	98,6%	0,7%	99,3%	0,0%	100,0%	1,5%	98,5%	0,0000	0,0062
5	15	35	0,0%	100,0%	0,4%	99,6%	0,6%	99,4%	4,6%	95,4%	7,4%	92,6%	0,0061	0,0282
5	20	50	0,0%	100,0%	0,9%	99,1%	1,3%	98,7%	5,6%	94,4%	9,5%	90,5%	0,0108	0,0625
5	25	63	0,1%	99,9%	1,4%	98,6%	1,3%	98,7%	6,1%	93,9%	14,7%	85,3%	0,0158	0,1008
5	30	77	0,1%	99,9%	2,0%	98,0%	1,5%	98,5%	4,0%	96,0%	20,9%	79,1%	0,0124	0,1605
5	35	93	0,1%	99,9%	1,7%	98,3%	1,8%	98,2%	4,8%	95,2%	21,4%	78,6%	0,0188	0,2469
6	10	20	0,0%	100,0%	0,8%	99,2%	0,5%	99,5%	0,0%	100,0%	4,6%	95,4%	0,0000	0,0133
6	15	35	0,0%	100,0%	0,2%	99,8%	0,2%	99,8%	5,1%	94,9%	5,6%	94,4%	0,0158	0,0605
6	20	50	0,0%	100,0%	0,4%	99,6%	0,3%	99,7%	4,5%	95,5%	10,3%	89,7%	0,0250	0,1521
6	25	63	0,0%	100,0%	0,7%	99,3%	0,4%	99,6%	4,6%	95,4%	12,2%	87,8%	0,0360	0,2749
6	30	77	0,0%	100,0%	0,8%	99,2%	0,4%	99,6%	5,0%	95,0%	22,8%	77,2%	0,0499	0,4239
6	35	93	0,0%	100,0%	1,0%	99,0%	0,3%	99,7%	5,6%	94,4%	23,9%	76,1%	0,0733	0,6823
7	10	20	0,0%	100,0%	0,5%	99,5%	1,0%	99,0%	2,9%	97,1%	6,1%	93,9%	0,0062	0,0211
7	15	35	0,0%	100,0%	0,2%	99,8%	0,1%	99,9%	3,3%	96,7%	8,2%	91,8%	0,0515	0,1303
7	20	50	0,0%	100,0%	0,3%	99,7%	0,1%	99,9%	4,1%	95,9%	10,6%	89,4%	0,0969	0,3761
7	25	63	0,0%	100,0%	0,7%	99,3%	0,2%	99,8%	5,1%	94,9%	21,3%	78,7%	0,1438	0,7169
7	30	77	0,1%	99,9%	0,9%	99,1%	0,3%	99,7%	5,5%	94,5%	24,8%	75,2%	0,2203	1,2042
7	35	93	0,1%	99,9%	1,1%	98,9%	0,2%	99,8%	5,7%	94,3%	22,2%	77,8%	0,3422	2,0669
8	10	20	0,0%	100,0%	1,0%	99,0%	1,1%	98,9%	1,1%	98,9%	9,3%	90,7%	0,0064	0,0314
8	15	35	0,0%	100,0%	0,4%	99,6%	0,4%	99,6%	4,1%	95,9%	13,0%	87,0%	0,2530	0,2616
8	20	50	0,1%	99,9%	0,7%	99,3%	0,2%	99,8%	5,5%	94,5%	17,6%	82,4%	0,5470	0,9439
8	25	63	0,1%	99,9%	1,1%	98,9%	0,3%	99,7%	6,5%	93,5%	22,7%	77,3%	0,7421	1,9958
8	30	77	0,1%	99,9%	2,0%	98,0%	0,4%	99,6%	7,2%	92,8%	41,6%	58,4%	1,1251	3,5791
8	35	93	0,2%	99,8%	2,6%	97,4%	0,4%	99,6%	9,7%	90,3%	12,1%	87,9%	2,2030	6,6751
Promedio			0,30%	99,70%	2,91%	97,09%	2,16%	97,84%	4,92%	95,08%	15,12%	84,88%		

Tabla 6.1.4 – Tiempos porcentuales de las dos etapas del algoritmo
Se corresponden con los resultados de la tabla 6.1.3

En resumen, **TabúS BCCP – Opc. 1** obtiene prácticamente los mejores resultados para K = 4, 5 y varios casos de 6. Esto sucede por la misma razón que hace que para grafos y K grandes, un espacio de soluciones más reducido obtenga mejores resultados. Para grafos y K más chicos, un espacio de soluciones más amplio (o en este caso el espacio completo de soluciones, por generarse todos los ciclos posibles), obtiene mejores resultados. A modo de ejemplo, si observamos el algoritmo **TabúS BCCP – Opc. 2** para K = 4 en el gráficos 6.1.1, vemos cómo hay resultados muy dispares. En particular, n = 30, es el peor resultado obtenido por este algoritmo. Esto se debe a que se está trabajando con un espacio de soluciones más reducido, y además al

utilizar una lista tabú que tiende a restringirlo más aun. Luego, para K = 5, 6, 7 y 8, el algoritmo tiene un mejor y más estable comportamiento.

A modo de conclusión, por las razones aquí detalladas, decidimos en adelante seguir trabajando con la opción 2 del generador de ciclos, aunque no descartamos en un futuro, utilizar la opción 1 para K chicos ó grafos más pequeños.
Finalmente, al comparar **TabúS BCCP – Opc. 2** contra **TabúS BCCP – Opc. 2 plus**, notamos una sensible mejora en las soluciones por parte de este último, no obstante, un fuerte incremento en los tiempos de ejecución empleados (4 veces más), directamente relacionados con el aumento en la máxima cantidad de iteraciones permitidas en la etapa de intensificación del Tabú Search. Por dicha razón descartamos la opción plus, aunque nos quedamos con la idea de que para mejorar una solución, tenemos que estar dispuestos a invertir más tiempo.
Como corolario de este punto, por todo lo expuesto anteriormente, los algoritmos que proponemos son **Tabú Search BCCP – Opción. 2** y **B. Local BCCP – Opción. 2.**

6.2 Ensayos con diversos parámetros

El algoritmo Tabú elegido en la sección 6.1 es **Tabú Search BCCP – Opción. 2**. Éste ofrece una serie de parámetros que se establecen de antemano y permiten modificar sustancialmente el comportamiento a lo largo de la ejecución. En esta sección nos centraremos en experimentar con el algoritmo, aplicando diversos valores para los parámetros involucrados. Luego, analizaremos los resultados y el comportamiento del mismo. Finalmente elegiremos los parámetros que obtengan mejores resultados en la mayoría de los ensayos.

Los datos del problema son:

Grafo: representa la red de comunicaciones que queremos diseñar y/o configurar.

K: tamaño máximo permitido que puede tener un ciclo en la red. El mínimo es siempre 3. Nuestros ensayos comprenden valores de 4 a 8 inclusive.

Los parámetros involucrados son:

Generador de Ciclos: permite elegir la alternativa 1 que genera todos los ciclos posibles, o la alternativa 2 que genera una base de ciclos. Nosotros utilizaremos únicamente la alternativa 2.

Iteraciones Tabú: establece la cantidad máxima de iteraciones que puede efectuar el algoritmo tabú search.

Iteraciones sin mejorar: establece la máxima cantidad de iteraciones que el algoritmo puede realizar sin mejorar la solución.

Longitud lista tabú: permite establecer la longitud de la lista tabú. La misma almacena ciclos que estarán temporalmente prohibidos, en la medida en que permanezcan en dicha lista. Cabe destacar que ésta es estática, en el sentido que una vez que su longitud fue establecida, perdura a lo largo de toda la ejecución. La longitud empleada por defecto es de 100.

Iteraciones Intensificación: la intensificación del Tabú Search utiliza una función de backtracking que tiene limitada la cantidad de iteraciones. Como se vio en la sección 5.1, este algoritmo es exponencial, y para romper con ello se limitó la máxima cantidad de iteraciones que éste puede realizar.

Algoritmo de Tabú Search propuesto, con diferentes parametros						
Parámetros	TabuS Opc.2	TabuS Opc.2 plus	TabuS Opc.2 - A	TabuS Opc.2 - B	TabuS Opc.2 - C	TabuS Opc.2 - D
Iteraciones Tabú	300	300	1000	2000	200	400
Iteraciones sin mejorar	_	_	300	400	150	150
Longitud lista tabú	100	100	400	100	100	100
Iteraciones Intensificación	1.000.000	25.000.000	1.000.000	100.000	2.000.000	5.000.000

Tabla 6.2.1 - Valores establecidos para los distintos parámetros del Tabú Search.
En todos los casos de utiliza el generador de ciclos que construye una base (opción 2).

El criterio de selección de dichos valores se obtuvo a partir de ensayos preliminares en casos más complejos. Éstos están compuestos por K grandes, grafos grandes, y/o cuya densidad de ejes es superior respecto de sus pares de igual cantidad de nodos.
En los siguientes gráficos 6.2.2 y 6.2.3 se observan los ensayos realizados para la familia de grafos euclideana, para K = 4 y 5 y para K = 6, 7 y 8 respectivamente.

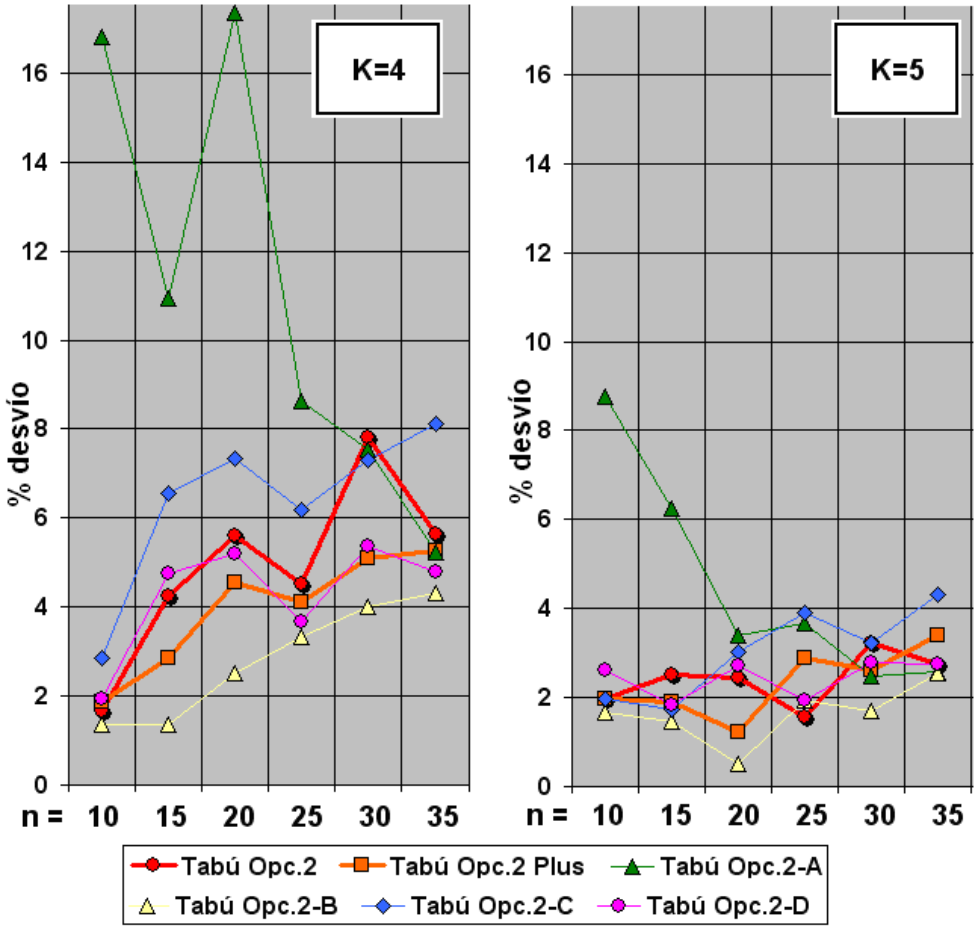


Gráfico 6.2.2 – Ensayos realizados con grafos euclideanos para K=4 y K=5

Nuestro algoritmo **Tabú Opc.2** es el elegido, y es el mismo expuesto en las secciones anteriores con el nombre **TabúS BCCP Opc.2**. Éste será el patrón de referencia respecto de los otros algoritmos, cuando establezcamos una comparación. Como se observa en la tabla 6.2.1, utiliza 300 Iteraciones Tabú, 1.000.000 Iteraciones Intensificación y una lista tabú de longitud 100.

Como se vio en la sección 6.1, **Tabú Opc.2 Plus** ofrece mejores resultados que nuestro algoritmo, aunque el tiempo adicional empleado hace que éste quede descartado. La clave de éste, radica que en emplea 25 veces más iteraciones para la intensificación que **Tabú Opc.2**.

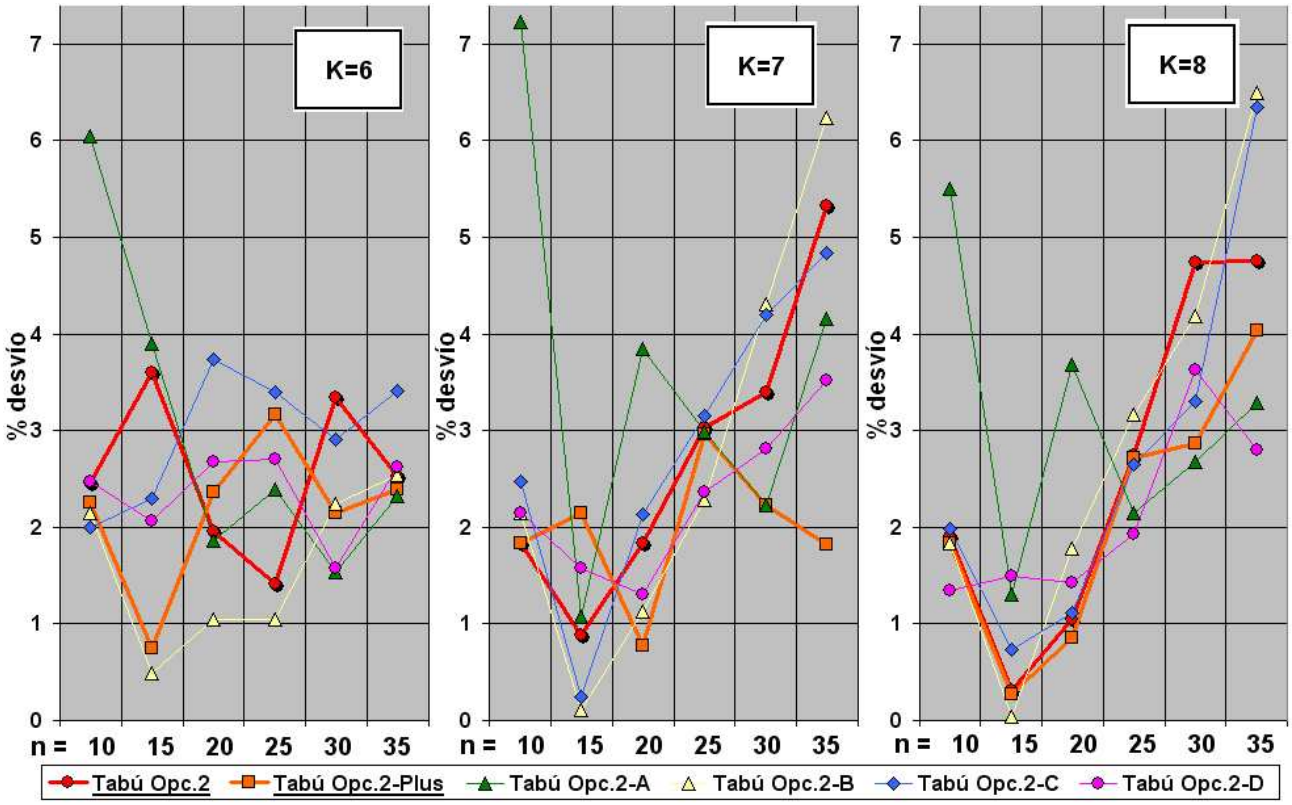


Gráfico 6.2.3 – Ensayos realizados con grafos euclideanos para K=6, K=7 y K=8

Tabú Opc.2-A se caracteriza por tener una lista tabú excesivamente grande y muchas más iteraciones tabú. Esto es 4 veces más que la lista estándar, lo que da una longitud total de 400 y 1000 iteraciones tabú (3,3 veces más). La premisa por la cual fue concebido, tiene por objeto funcionar bien en grafos grandes para K grandes. En particular, nos interesan los grafos cuyo n = 30 y 35, y K = 7 y 8. El resultado es positivo, aunque si se observa la tabla 6.2.4, el tiempo empleado es 3 veces más que **Tabú Opc.2**.

La siguiente variante **Tabú Opc.2-B**, sorprende por los resultados logrados para la mayoría de los casos, y por superar el porcentaje promedio y disminuir los tiempos empleados respecto **Tabú Opc.2** (ver tabla 6.2.4). Aunque, al analizar el comportamiento, se observa un muy buen desempeño en los casos chicos y medianos, específicamente en K = 4, 5, 6 y K = 7 y 8 para n=10,15 y quizás 20. Para los casos restantes K = 7 y 8 con n=25, 30 y 35, se observa un rápido crecimiento en el porcentaje de desvío (prácticamente lineal) resultando un deterioro significativo en las soluciones. Si bien a **Tabú Opc.2** le ocurre algo parecido en los casos antes mencionados, el deterioro de las soluciones es más atenuado para dichos casos.

Grafos			(%) Porcentajes de error respecto CPP						Tiempo medido en segundos					
K	# Nodos n	# Ejes Promedio	% TabuS Opc.2	%TabuS Opc.2 plus	%TabuS Opc.2 - A	%TabuS Opc.2 - B	%TabuS Opc.2 - C	%TabuS Opc.2 - D	TabuS Opc.2	TabuS Opc.2 plus	TabuS Opc.2 - A	TabuS Opc.2 - B	TabuS Opc.2 - C	TabuS Opc.2 - D
4	10	20	**1,66**	1,87	16,81	1,35	2,85	1,95	**0,12**	0,16	0,06	0,32	0,10	0,12
4	15	35	**4,23**	2,85	10,93	1,36	6,57	4,75	**0,23**	0,18	0,15	0,62	0,18	0,29
4	20	50	**5,61**	4,55	17,35	2,51	7,34	5,21	**2,13**	45,73	0,27	1,48	4,00	14,09
4	25	63	**4,51**	4,10	8,64	3,32	6,19	3,67	**0,37**	24,66	0,40	1,03	0,24	0,39
4	30	77	**7,81**	5,11	7,55	4,00	7,29	5,35	**0,40**	0,39	0,82	1,04	2,13	0,53
4	35	93	**5,62**	5,25	5,23	4,30	8,10	4,78	**0,48**	0,43	1,09	1,65	2,26	0,64
5	10	20	**1,98**	1,98	8,75	1,67	1,98	2,60	**0,64**	0,94	0,93	1,72	0,56	0,72
5	15	35	**2,50**	1,90	6,25	1,45	1,74	1,83	**7,75**	4,35	13,54	9,11	2,83	12,33
5	20	50	**2,46**	1,23	3,40	0,50	3,03	2,70	**7,00**	4,73	5,71	14,04	4,30	10,09
5	25	63	**1,56**	2,90	3,66	1,92	3,92	1,95	**7,14**	8,10	19,68	17,45	3,69	7,27
5	30	77	**3,24**	2,63	2,48	1,71	3,24	2,78	**8,04**	10,72	29,61	20,43	3,62	12,48
5	35	93	**2,74**	3,39	2,59	2,54	4,31	2,74	**14,70**	13,67	31,93	20,30	10,37	17,88
6	10	20	**2,46**	2,26	6,05	2,15	1,99	2,47	**1,55**	2,24	1,49	4,33	1,34	1,36
6	15	35	**3,60**	0,75	3,90	0,49	2,29	2,06	**37,94**	27,92	60,62	42,16	46,46	51,35
6	20	50	**1,95**	2,37	1,86	1,05	3,74	2,67	**36,25**	58,31	110,66	50,30	24,53	52,44
6	25	63	**1,41**	3,17	2,39	1,04	3,40	2,70	**42,94**	64,78	103,52	50,99	35,84	86,79
6	30	77	**3,34**	2,15	1,54	2,24	2,90	1,58	**52,20**	104,69	135,14	48,26	34,06	129,93
6	35	93	**2,53**	2,39	2,32	2,54	3,41	2,62	**66,94**	197,79	268,60	77,82	70,84	261,64
7	10	20	**1,83**	1,83	7,23	2,15	2,47	2,15	**2,96**	2,15	1,92	5,52	2,12	2,56
7	15	35	**0,88**	2,15	1,07	0,11	0,25	1,58	**55,18**	179,10	96,10	63,23	40,83	56,59
7	20	50	**1,83**	0,77	3,85	1,13	2,13	1,31	**138,82**	399,41	133,03	79,42	83,66	199,36
7	25	63	**3,03**	2,96	2,99	2,28	3,15	2,36	**97,79**	365,00	229,75	108,32	93,13	244,16
7	30	77	**3,39**	2,23	2,23	4,30	4,20	2,81	**126,03**	405,22	291,33	70,60	114,42	200,58
7	35	93	**5,32**	1,82	4,16	6,23	4,84	3,52	**194,27**	1045,6	470,85	122,54	199,10	619,57
8	10	20	**1,90**	1,83	5,50	1,83	1,98	1,35	**3,13**	2,70	2,22	7,97	2,03	2,35
8	15	35	**0,31**	0,27	1,30	0,04	0,73	1,49	**62,97**	74,73	77,39	63,18	61,08	52,72
8	20	50	**1,04**	0,85	3,68	1,78	1,11	1,43	**140,22**	453,76	176,58	85,75	118,35	245,07
8	25	63	**2,74**	2,72	2,14	3,16	2,65	1,93	**176,64**	768,02	305,79	120,76	121,53	385,49
8	30	77	**4,74**	2,87	2,68	4,18	3,30	3,63	**184,74**	917,61	640,00	134,59	187,60	560,53
8	35	93	**4,76**	4,04	3,29	6,49	6,34	2,80	**262,24**	1503,1	740,93	186,20	280,40	843,76
(%) Promedio y Total para K= 6, 7 y 8									Tiempo Promedio y Total para K= 6, 7 y 8					
Promedio			**2,61**	2,08	3,23	2,40	2,83	2,25	**93,5**	365,1	213,7	73,4	84,3	222,0
Total			**47,06**	37,43	58,18	43,19	50,88	40,46	**1682,8**	6572,2	3845,9	1321,9	1517,3	3996,3
(%) de error Prom. y Total para K=4, 5, 6, 7 y 8									Tiempo Prom. y Total para K=4, 5, 6, 7 y 8					
Promedio			**3,03**	2,51	5,06	2,33	3,58	2,69	**57,7**	222,9	131,7	47,0	51,7	135,8
Total			**90,98**	75,19	151,8	69,82	107,4	80,77	**1731,8**	6686,2	3950,1	1411,1	1551,6	4073,1

Tabla 6.2.4 – Resultados detallados de los gráficos 6.2.2 y 6.2.3

Tabú Opc.2-C emplea un 33% menos iteraciones tabú respecto **Tabú Opc.2** y duplica las iteraciones en la intensificación. Esto es 200 y 2.000.000 respectivamente. Los resultados no mejoran los actuales, y los tiempos empleados son muy similares.

Tabú Opc.2-D utiliza 400 iteraciones tabú y 5.000.000 iteraciones intensificación. El resultado es una mejora global en todos los ensayos respecto **Tabú Opc.2**, y un deterioro significativo en los tiempos empleados. El promedio para los ensayos de K = 6, 7 y 8 es de 2,25% contra 2,61% de **Tabú Opc.2** y 1,98% y 2,13% de **Greedy + AB** y **Greedy + CB** respectivamente, propuestos en [8] (ver sección 6.2), es decir, esta muy cerca de los mejores resultados globales. La desventaja de esta variante es el tiempo empleado, cuyo promedio es de 222 segundos contra 93,5; 0,10 y 14,14 de **Tabú Opc.2**, **Greedy + AB** y **Greedy + CB** respectivamente. Aunque la proporción calidad y tiempo empleado mejoró bastante respecto **Tabú Opc.2 Plus** (ver tabla 6.2.6).

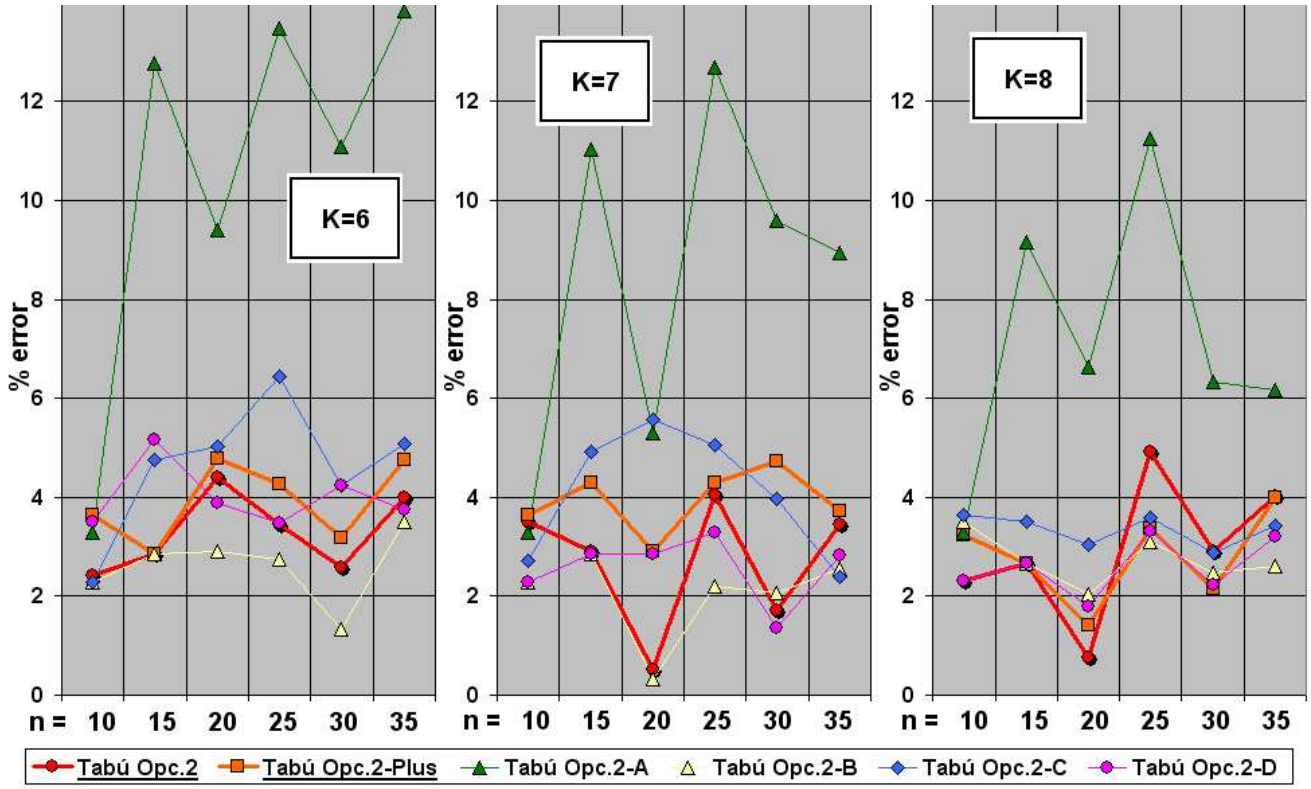


Gráfico 6.2.5 – Ensayos realizados con grafos ralos y/o sparser para K=6, K=7 y K=5

Hasta aquí fueron los resultados para la familia de grafos euclideanos. A continuación analizaremos los resultados para la familia de grafos ralos o sparser. Dichos ensayos fueron hechos para K = 4, 5, 6, 7 y 8. En el siguiente gráfico sólo se muestran los ensayos para K=6, 7 y 8. Notar que para K=4 no se encontraron resultados. En comparación con la familia de grafos euclideanos, se preservan en la mayoría de los casos las mismas proporciones y/o tendencias en cuanto a calidad de la solución y tiempo empleado.

La primer diferencia importante, es que el promedio y/o total de nuestro algoritmo **Tabú Opc.2**, aventaja a **Tabú Opc.2 Plus** y **Tabú Opc.2-D**, y además se mantienen la proporción de los tiempos empleados (ver tabla 6.2.6).
El otro hecho importante a destacar, es que **Tabú Opc.2-B** mejora respecto de **Tabú Opc.2** para K y grafos grandes, contrariamente a lo ocurrido en la familia de grafos euclideana. Además sigue preservando el mejor promedio en la calidad de la soluciones y tiempo empleado respecto de nuestro algoritmo **Tabú Opc.2**, lo que lo convierte en el mejor de todos para esta familia, tanto para las variantes analizadas en esta sección, como para los algoritmos propuestos en el trabajo [8] .

Grafos			(%) Porcentajes de desvío respecto CPP						Tiempo medido en segundos					
K	# Nodos n	# Ejes Promedio	% TabuS Opc.2	%TabuS Opc.2 plus	%TabuS Opc.2 - A	%TabuS Opc.2 - B	%TabuS Opc.2 - C	%TabuS Opc.2 - D	TabuS Opc.2	TabuS Opc.2 plus	TabuS Opc.2 - A	TabuS Opc.2 - B	TabuS Opc.2 - C	TabuS Opc.2 - D
4	10	16	**4,66**	4,66	16,64	4,66	4,66	4,66	**0,05**	0,03	0,03	0,08	0,03	0,03
4	15	25	–	–	–	–	–	–	–	–	–	–	–	–
4	20	36	–	–	–	–	–	–	–	–	–	–	–	–
4	25	47	–	–	–	–	–	–	–	–	–	–	–	–
4	30	60	–	–	–	–	–	–	–	–	–	–	–	–
4	35	66	–	–	–	–	–	–	–	–	–	–	–	–
5	10	16	**5,60**	4,46	8,74	4,46	4,46	5,42	**0,07**	0,12	0,08	0,23	0,06	0,07
5	15	25	**8,29**	9,33	23,92	9,59	11,31	11,00	**0,47**	0,18	0,05	0,91	0,06	0,41
5	20	36	**3,58**	7,62	9,56	5,52	8,86	9,56	**1,45**	2,36	0,08	1,69	0,75	1,64
5	25	47	**8,39**	7,24	12,40	8,72	8,47	8,47	**18,55**	107,42	0,13	5,20	11,72	38,09
5	30	60	**9,81**	6,31	34,43	7,59	11,36	10,86	**14,88**	344,63	0,20	5,44	17,90	85,19
5	35	66	**15,55**	9,90	24,99	9,77	14,49	9,01	**9,73**	323,82	0,20	5,06	6,41	38,58
6	10	16	**2,41**	3,64	3,28	2,28	2,28	3,50	**0,44**	0,20	0,42	0,53	0,30	0,20
6	15	25	**2,86**	2,86	12,76	2,86	4,75	5,15	**0,58**	0,42	0,55	1,11	0,61	0,79
6	20	36	**4,40**	4,79	9,39	2,91	5,03	3,89	**12,17**	164,64	0,56	8,28	12,55	113,32
6	25	47	**3,45**	4,27	13,47	2,75	6,44	3,49	**4,37**	2,88	2,10	8,46	1,56	3,71
6	30	60	**2,58**	3,18	11,09	1,33	4,23	4,25	**11,82**	92,26	1,91	7,55	5,21	40,92
6	35	66	**3,98**	4,75	13,82	3,50	5,08	3,76	**3,83**	47,79	0,82	4,04	5,18	11,04
7	10	16	**3,50**	3,64	3,28	2,28	2,73	2,28	**0,32**	0,31	0,35	1,16	0,27	0,28
7	15	25	**2,92**	4,30	11,02	2,84	4,92	2,84	**2,34**	1,77	1,70	3,17	1,27	1,28
7	20	36	**0,51**	2,92	5,29	0,32	5,56	2,86	**33,93**	71,79	32,90	14,93	22,85	33,00
7	25	47	**4,04**	4,30	12,68	2,19	5,04	3,29	**22,94**	13,96	12,53	17,78	15,57	14,64
7	30	60	**1,71**	4,74	9,59	2,07	3,97	1,35	**24,56**	20,89	20,66	29,09	13,83	13,87
7	35	66	**3,46**	3,71	8,93	2,55	2,39	2,83	**14,38**	127,15	19,77	25,62	12,64	41,48
8	10	16	**2,32**	3,23	3,28	3,50	3,64	2,32	**0,63**	0,31	0,61	0,79	0,23	0,53
8	15	25	**2,67**	2,67	9,15	2,67	3,50	2,67	**1,71**	3,59	4,04	5,53	1,88	5,17
8	20	36	**0,75**	1,41	6,64	2,03	3,05	1,78	**21,66**	83,01	24,21	35,06	20,02	48,36
8	25	47	**4,91**	3,38	11,25	3,09	3,58	3,32	**40,15**	187,93	42,49	21,26	27,87	49,25
8	30	60	**2,91**	2,15	6,34	2,48	2,87	2,23	**50,41**	138,08	155,80	48,69	51,65	112,03
8	35	66	**4,02**	3,98	6,17	2,62	3,42	3,20	**55,80**	139,50	112,62	64,46	41,83	110,05
(%) Promedio y Total para K= 6, 7 y 8									Tiempo Promedio y Total para K= 6, 7 y 8					
Promedio			**2,97**	3,55	8,75	2,46	4,03	3,06	**16,8**	60,9	24,1	16,5	13,1	33,3
Total			**53,40**	63,92	157,4	44,27	72,48	55,01	**302,0**	1096,5	434,0	297,5	235,3	599,9
(%) de error Prom. y Total para K=4, 5, 6, 7 y 8									Tiempo Prom. y Total para K=4, 5, 6, 7 y 8					
Promedio			**4,37**	4,54	11,52	3,78	5,44	4,56	**13,9**	75,0	17,4	12,6	10,9	30,6
Total			**109,3**	113,4	288,1	94,58	136,1	114,0	**347,2**	1875,0	434,8	316,1	272,3	764,0

Tabla 6.2.6 – Resultados detallados del gráfico 6.2.5

A continuación analizaremos el comportamiento de las distintas variantes del algoritmo para un caso testigo. La idea de los siguientes párrafos es entender el comportamiento del algoritmo, para poder determinar cuáles son los parámetros más acertados.

El grafo elegido es euclideano, tiene 35 nodos y 95 ejes (grafo 35.5). La cota inferior de este grafo, es decir, solución del cartero chino, tiene un costo de 53.661. Cabe destacar que el posterior análisis no pretende establecer reglas universales a partir de este ejemplo, sino observar el desempeño del algoritmo, y por tanto esbozar algunas conjeturas.

Los siguientes gráficos muestran la radiografía completa del comportamiento del algoritmo. En el eje **x** se observan las iteraciones, y en el eje **y** el costo total de la solución encontrada para una iteración dada. Para simplificar, asumimos que dos soluciones son distintas si tienen distinto costo. Esta simplificación es muy acertada para los grafos con los que estamos trabajando, por ser el costo de sus ejes muy disímiles entre sí, y por lo tanto, la probabilidad de que dos soluciones distintas tengan el mismo costo, es baja.

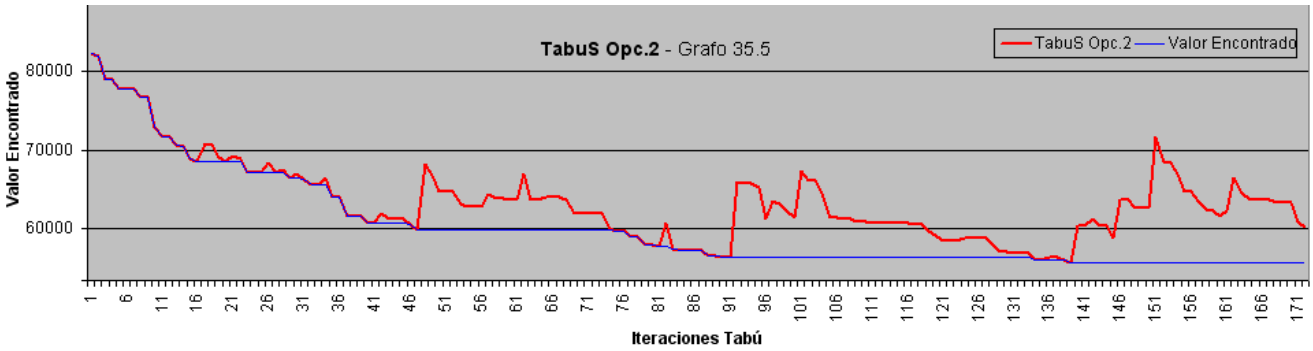


Gráfico 6.2.7 – Comportamiento del Tabú Search Opc.2 para un grafo de 35 nodos

La línea azul representa la mejor solución encontrada hasta el momento, en este caso por **TabúS Opc.2**. La línea roja muestra el comportamiento del algoritmo a lo largo de las sucesivas iteraciones. En ellas se observa cómo éste sale de un mínimo local una vez hallado. Al principio da saltos pequeños respecto de la mejor solución encontrada, debido a que el algoritmo está mejorando permanentemente. Luego, los saltos son más pronunciados, aunque en términos absolutos éstos se encuentran dentro de un cierto rango.
Notar que el algoritmo describe distintas figuras donde a su vez el costo de los puntos que la componen es también distinto. Esto se interpreta como que una figura es un área dentro del espacio global de soluciones, y un punto de esa figura es una solución cuya iteración está dada por el eje X, y su costo está dado por el eje Y. Para este caso, se puede observar cómo el algoritmo explora distintas áreas, dado que se observan figuras disímiles.

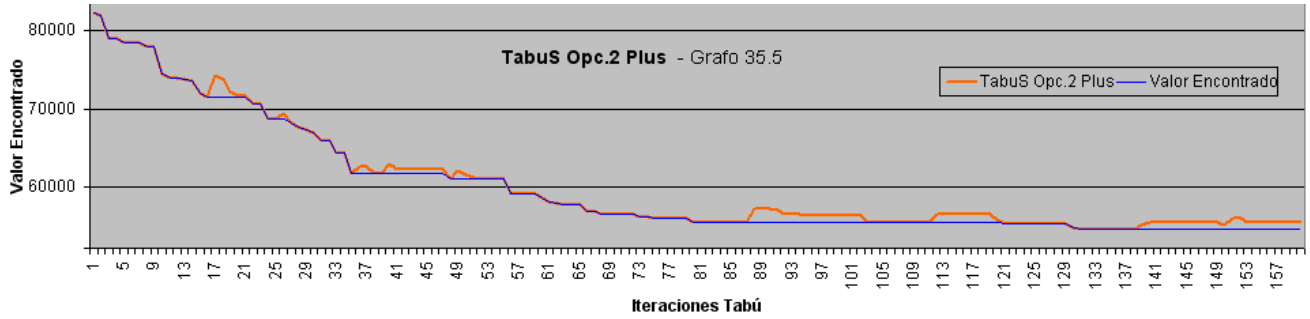


Gráfico 6.2.8 – Comportamiento del Tabú Search Opc.2 Plus para un grafo de 35 nodos

En esta variante **TabúS Opc.2 Plus**, se observan saltos muy pequeños en términos absolutos y relativos. Esto se debe a que esta opción tiene una función de intensificación muy potente (recordar que emplea 25 veces más iteraciones que **TabúS Opc.2**),y por lo tanto cada vez que el algoritmo se diversifica, la función de intensificación rápidamente regresa en las próximas iteraciones a valores cercanos al mejor encontrado al momento.

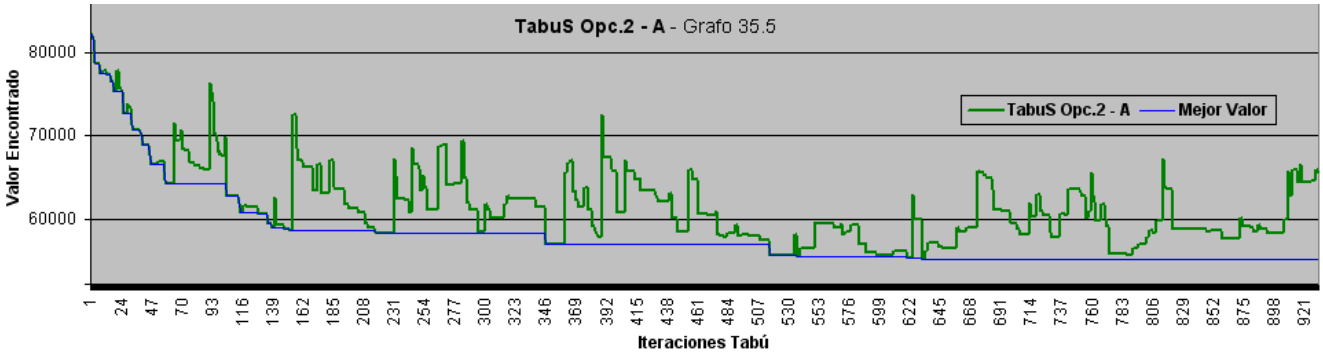


Gráfico 6.2.9 – Comportamiento del Tabú Search Opc.2-A para un grafo de 35 nodos

Tanto esta variante **TabúS Opc.2-A** como la siguiente **TabúS Opc.2-B**, parecen tener un comportamiento similar, si bien los parámetros empleados son conceptualmente opuestos (ver 6.2.1). A diferencia de **TabúS Opc.2**, éstos dan saltos más bruscos, y si bien parecen oscilar mucho más, hay que tener en cuenta que emplean muchas más iteraciones y esto les permite delinear más figuras y/o explorar más áreas.

Ambas variantes tienen en común el empleo de muchas iteraciones tabú, pero contrastan en cuanto a la longitud de la lista tabú y la cantidad de iteraciones en la intensificación. En la primera, los saltos bruscos se dan por la longitud de la lista tabú (prohíbe hasta 400 ciclos), creando de esta forma una fuerte restricción. En la segunda, se explica por tener una función suave de intensificación (esto es 10 veces menos que **TabúS Opc.2**). La lista tabú le impone una restricción a **TabúS Opc.2-B**, y éste no tiene fuerza suficiente para volver rápidamente a una solución cercana a la mejor (ver gráfico 6.2.10). Sin embargo, el ahorro de tiempo que esta variante hace en cuanto a la intensificación, le permite realizar más iteraciones tabú, y por lo tanto explorar nuevas áreas. Éste es el factor principal que explica su éxito en los casos medianos y/o chicos, aunque para los casos grandes constituye la principal desventaja. Contrariamente a esta variante, **TabúS Opc.2-A**, luego de pegar un salto, rápidamente vuelve a posicionarse gracias al empleo de una mayor cantidad de iteraciones en la intensificación. En cuanto a los resultados, contrariamente a **TabúS Opc.2-B, TabúS Opc.2-A** obtiene malos resultados para grafos medianos y chicos, y buenos resultados para grafos grandes. (ver gráficos 6.2.2 y 6.2.3)

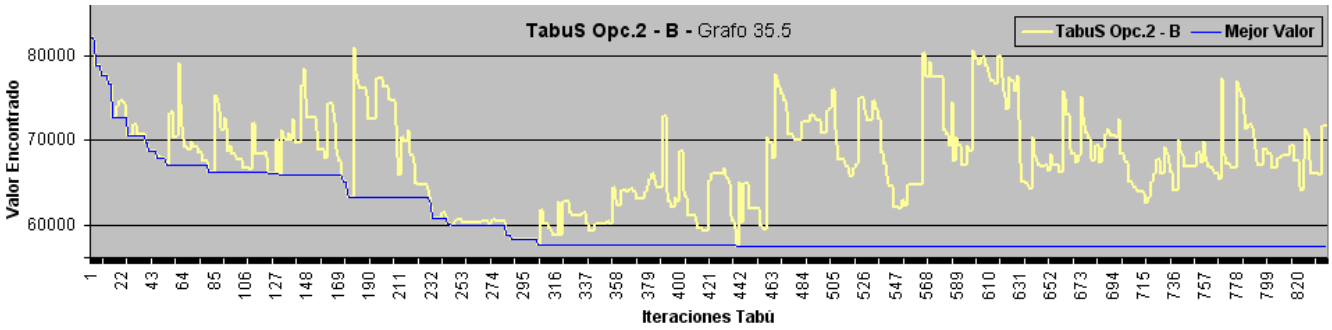


Gráfico 6.2.10 – Comportamiento del Tabú Search Opc.2-B para un grafo de 35 nodos

En la opción **TabúS Opc.2-C**, se observa un comportamiento intermedio entre **TabúS Opc.2** y **TabúS Opc.2 Plus**, es decir, saltos más acentuados respecto el primero, y más bruscos respecto el segundo. Si observamos los resultados en los gráficos 6.2.2 y 6.2.3, y su desempeño en el siguiente gráfico 6.2.11, concluimos con que esta variante tiene un desempeño coherente respeto de los parámetros recibidos (ver tabla 6.2.1).

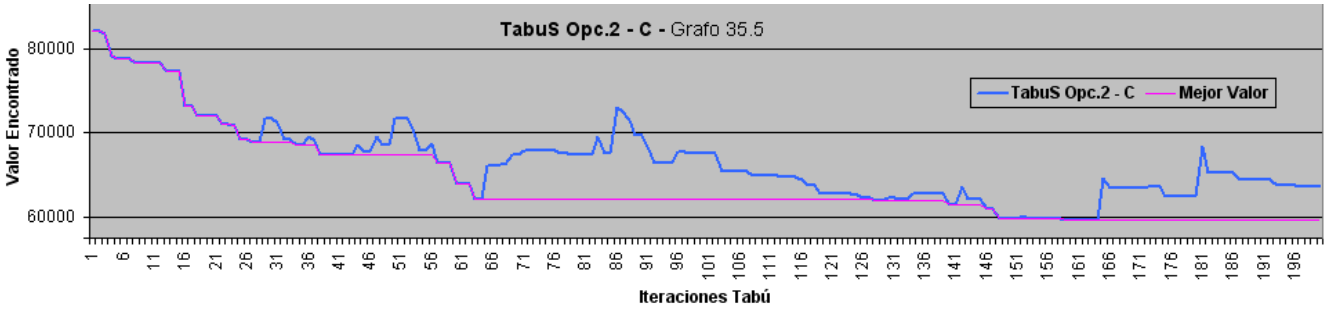


Gráfico 6.2.11 – Comportamiento del Tabú Search Opc.2-C para un grafo de 35 nodos

La última variante **TabúS Opc.2-D**, tiene saltos limitados y muy pronunciados. Esta variante emplea 5 veces más iteraciones en la intensificación que **TabúS Opc.2**, lo cual hace que su comportamiento tienda a parecerse a **TabúS Opc.2 Plus**.

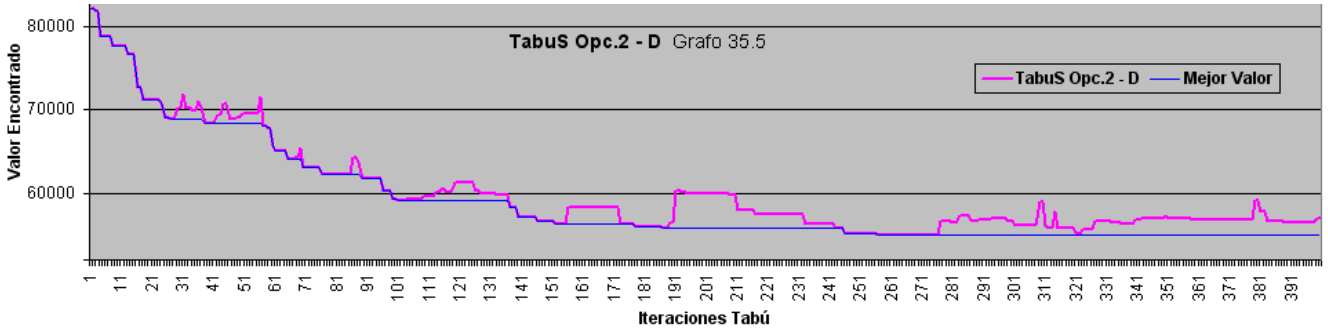


Gráfico 6.2.12 – Comportamiento del Tabú Search Opc.2-D para un grafo de 35 nodos

Luego de este análisis, se empiezan a ver ciertos patrones de conducta directamente vinculado con determinados parámetros del algoritmo.

6.3 Análisis de los resultados

Tras haber propuesto los algoritmos **Tabú Search BCCP – Opción 2** y **B. Local BCCP – Opción 2**, procederemos a comparar los resultados obtenidos con los del trabajo de D. Hochbaum y E. Olinick [8]. La idea aquí es, establecer un marco de referencia con otros algoritmos que resuelvan BCCP, para ver qué tan buenos fueron nuestros resultados, al margen de la cota inferior del Cartero Chino (CPP).

La comparación de tiempos expuesta a lo largo de esta sección es meramente informativa, dado que ambos experimentos fueron realizados en diferentes condiciones. El objetivo es ver que tan cerca estamos de los tiempos empleados en términos absolutos. Los ensayos del trabajo [8] fueron realizados en una computadora Sun Microsystems SPARCstation 20 Modelo 16 workstation, con un procesador SuperSparc de 60 Mhz y 32 MB de memoria RAM. Nuestros ensayos fueron realizados en una PC con un procesador Intel Core 2 Duo de 2.20 Ghz, y 2Gb de memoria Ram.

Como ya mencionamos, Hochbaum y E. Olinick proponen tres heurísticas. Ellos plantean el problema de elegir ciclos, como instancias de un problema de cubrimientos de ejes con ciclos, en este caso, Set - Covering Problem. Posteriormente, al resultado de cada una aplican programación lineal para la selección de ciclos, utilizando el algoritmo de CPLEX. Debido a la relajación lineal, se acepta valores decimales, que luego son redondeados, logrando de esta forma buenas soluciones, aunque no necesariamente la optima.
A continuación daremos un breve resumen de las heurísticas expuestas en el trabajo [8]:

1. Greedy Postman Heuristic: parte de un cubrimiento generado por el Chinese Postman Algorithm, y seguidamente genera los *K-Ciclos* necesarios para construir un BCC. En otras palabras, reemplaza los ciclos degenerados de tamaño dos, y los que superan el tamaño K, por otros *K-Ciclos* válidos.

2. Cycle Basis (CB): construye una cantidad limitada de *K-Ciclos* (a lo sumo $n \times m$) y luego aplica un algoritmo goloso para construir un *BCC*.

3. Augmented Cycle Basis (AB): aplica la heurística anterior y en caso de no hallar un *BCC*, genera y agrega los *K-Ciclos* necesarios para poder hacerlo.

Cabe destacar que en nuestros ensayos utilizamos la misma familia de grafos. Asimismo, la forma de medir los resultados será la misma que en la sección anterior, es decir, calculando el desvío relativo respecto CPP. Los tiempos empleados se medirán en segundos. Como novedad, incorporaremos el desvío estándar calculado en función de los desvíos relativos.

En los siguientes tres gráficos que se muestran a continuación, se exponen ensayos realizados para K=6, 7 y 8, para la familia de grafos euclideanos. Al igual que los gráficos de la sección anterior, el eje X es una variable discreta que indica la cantidad de nodos del grafo, siendo sus valores 10, 15, 20, 25, 30 y 35 nodos. El eje Y indica el porcentaje de desvío de la solución hallada respecto CPP. Cada valor corresponde al promedio de 5 grafos distintos que tienen la misma cantidad de nodos, por ejemplo n = 10, n = 15 y así.

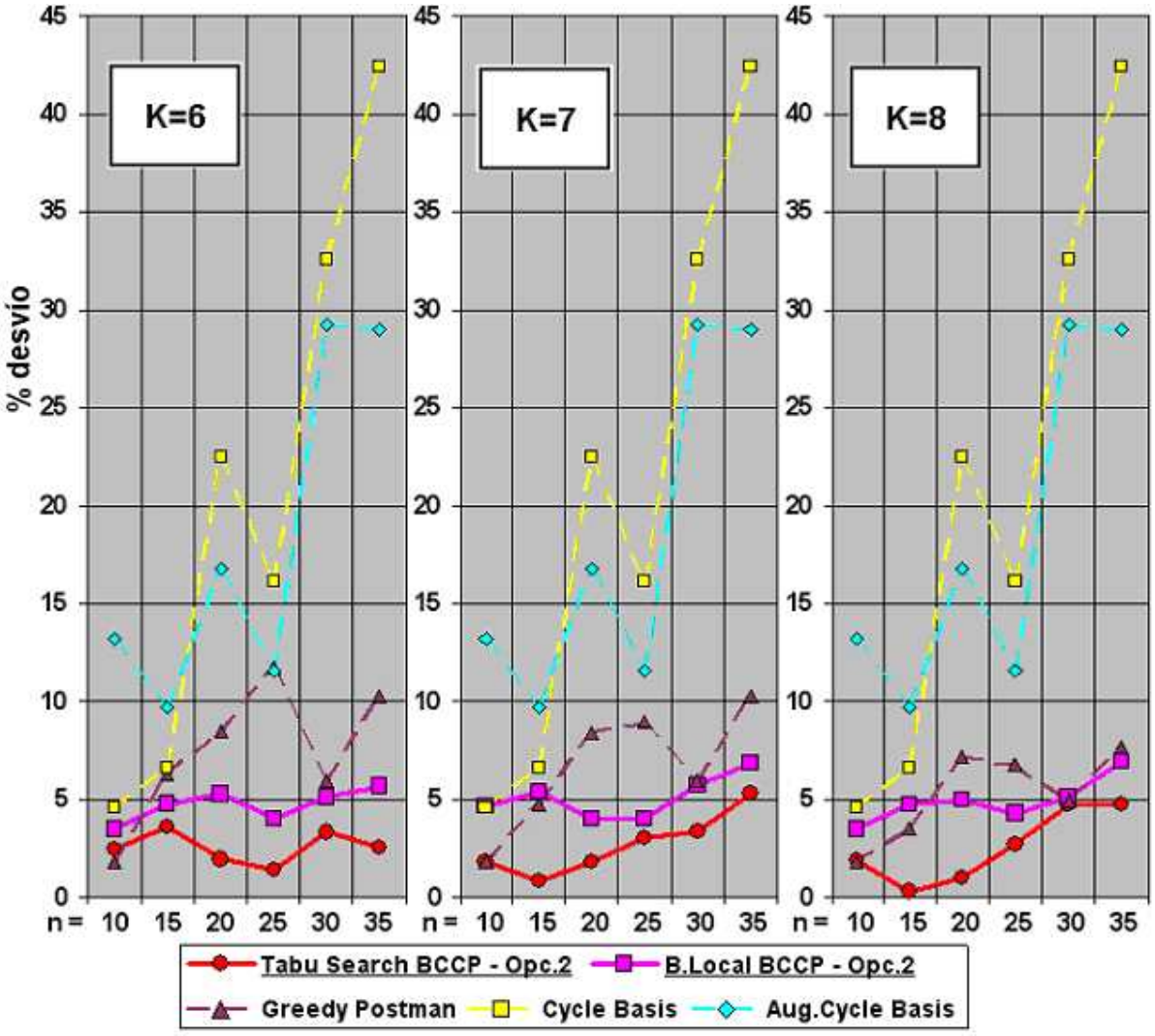


Gráfico 6.3.1– Comparación de las tres heurísticas a las cuales se les aplica CPLEX
Ensayos realizados con grafos euclideanos para K=6, K=7 y K=5

A simple vista se observa cómo nuestros algoritmos obtienen mejores resultados en la mayoría de los ensayos. En el caso de **Tabú Search BCCP**, vemos como aventaja al resto de los algoritmos en casi todos los casos. En promedio, el desvío relativo es de 2.61% con un desvío estándar promedio de 1,72%, mientras que para **B. Local BCCP**, **Greedy Postamn**, **Cycle Basis** y **Aug. Cycle Basis**, es de 4,93%, 6,49%, 20,80% y 18,27% respectivamente. Siguiendo el mismo orden, para el área debajo de la curva, es decir, la suma de todos los resultados, tenemos que para nuestro algoritmo Tabú es de 47,06%, mientras que para el resto es de 88,79%, 16,73%, 374,43% y 328,86% respectivamente.

En contrapartida, el tiempo empleado por el algoritmo, es mayor al resto. Por ejemplo, nuestro algoritmo tabú, el tiempo promedio es de 93.49 segundos, mientras que para **B. Local BCCP**, **Greedy Postamn**, **Cycle Basis** y **Aug. Cycle Basis** es 6,23, 0,15, 10,97 y 11,10 segundos, respectivamente. Para el total de tiempo empleado en todas las pruebas, tenemos 1682,81 segundos vs. 112,12, 2,69, 197,97 y 199,73 segundos. Esto se puede ver en detalle en la siguiente tabla 6.3.2.

Grafos		(%) Porcentajes de desvío respecto CPP					Desviación estándar		Tiempo medido en segundos				
K	# Nodos n	% Greedy Heuristic	% Cycle Basis	% Cycle Basis Augmented	% TabuS BCCP Opc.2	% B.Local BCCP Opc.2	Desviación estándar TabuS BCCP Opc.2	Desviación estándar B.Local BCCP Opc.2	Greedy Heuristic	Cycle Basis	Cycle Basis Augmented	TabuS BCCP Opc.2	B.Local BCCP Opc.2
6	10	1,82	4,57	13,23	**2,46**	**3,50**	**2,51**	**2,42**	0,04	0,12	0,14	**1,55**	**0,33**
6	15	6,28	6,63	9,72	**3,60**	**4,79**	**2,99**	**2,04**	0,06	0,85	0,89	**37,94**	**1,06**
6	20	8,52	22,47	16,82	**1,95**	**5,28**	**3,00**	**4,74**	0,09	3,44	3,48	**36,25**	**1,52**
6	25	11,77	16,12	11,55	**1,41**	**4,01**	**0,52**	**1,74**	0,14	7,64	7,73	**42,94**	**2,23**
6	30	5,94	32,58	29,28	**3,34**	**5,13**	**1,14**	**2,65**	0,23	16,46	16,63	**52,20**	**1,87**
6	35	10,29	42,44	29,02	**2,53**	**5,69**	**1,83**	**2,22**	0,33	37,30	37,71	**66,94**	**2,86**
7	10	1,82	4,57	13,23	**1,83**	**4,69**	**2,57**	**4,19**	0,04	0,12	0,14	**2,96**	**0,42**
7	15	4,75	6,63	9,72	**0,88**	**5,39**	**0,90**	**4,06**	0,06	0,85	0,89	**55,18**	**1,56**
7	20	8,43	22,47	16,82	**1,83**	**4,03**	**0,91**	**4,02**	0,09	3,44	3,48	**138,82**	**3,57**
7	25	8,95	16,12	11,55	**3,03**	**4,02**	**2,77**	**1,33**	0,14	7,64	7,73	**97,79**	**3,35**
7	30	5,94	32,58	29,28	**3,39**	**5,74**	**1,20**	**3,12**	0,23	16,46	16,63	**126,03**	**4,84**
7	35	10,28	42,44	29,02	**5,32**	**6,89**	**2,79**	**0,63**	0,33	37,30	37,71	**194,27**	**9,30**
8	10	1,82	4,57	13,23	**1,90**	**3,50**	**2,41**	**2,42**	0,04	0,12	0,14	**3,13**	**0,37**
8	15	3,56	6,63	9,72	**0,31**	**4,73**	**0,30**	**3,91**	0,06	0,85	0,89	**62,97**	**1,99**
8	20	7,18	22,47	16,82	**1,04**	**4,99**	**1,18**	**4,06**	0,09	3,44	3,48	**140,22**	**5,25**
8	25	6,76	16,12	11,55	**2,74**	**4,31**	**1,13**	**1,45**	0,14	7,64	7,73	**176,64**	**8,73**
8	30	4,95	32,58	29,28	**4,74**	**5,18**	**2,07**	**2,44**	0,23	16,46	16,63	**184,74**	**8,56**
8	35	7,67	42,44	29,02	**4,76**	**6,92**	**0,66**	**1,08**	0,33	37,30	37,71	**262,24**	**54,31**
Prom.		6,49	20,80	18,27	**2,61**	**4,93**	**1,72**	**2,69**	0,15	10,97	11,10	**93,49**	**6,23**
Total		116,73	374,43	328,86	**47,06**	**88,79**			2,69	197,47	199,73	**1682,81**	**112,12**

Tabla 6.3.2 – Resultados detallados del gráfico 6.3.2

En el caso de la heurística **B. Local BCCP**, se observa un buen desempeño respecto del resto de los algoritmos, aventajando claramente a dos de ellos: **Cycle Basis** y **Augmented Cycle Basis**. En relación a **Greedy Postman**, se observa cómo **B. Local BCCP** prevalece en más de la mitad de los casos. En términos absolutos, tenemos 88,79% vs. 116,73%, o en promedio, 4,93% vs. 6,49%.

Al observar los tiempos empleados en **B. Local BCCP**, notamos un drástico descenso respecto **Tabú Search BCCP** (6,93 en promedio vs. 93,49 segundos), y un importante descenso respecto **Cycle Basis** y **Augmented Cycle Basis** (6,93 en promedio vs. 10,97 y 11,10 segundos), aunque un aumento respecto de **Greedy Postman**. El tiempo promedio es de 6,23 segundos contra 0,15 de **Greedy Postman**.

Hasta aquí se realizó una comparación con los resultados de algunas de las heurísticas mencionadas en [8]. En dicho trabajo se propone además una heurística usando técnicas de programación lineal entera a una formulación del problema basada en las heurísticas anteriores. Asimismo, proponen dos heurísticas adicionales que pasaremos a detallar:

4. Greedy + CB: une los conjuntos de ciclos candidatos de G*reedy-Postman* y de *Cycle-Basis.* Luego, aplica a dicho conjunto un procedimiento de Branch & Bound. Como resultado, obtiene una mejora significa en la calidad de las soluciones respecto de las tres heurísticas anteriores, al precio de un mayor tiempo de procesamiento.

5. Greedy + AB: es similar a *Greedy + CB*, sólo que difiere en que combina los ciclos candidatos de las heurísticas de G*reedy-Postman* y de *Augmented- Basis.* Esta estrategia permite ampliar el espacio de soluciones, y por lo tanto, encontrar mejores resultados. Al igual que la heurística *Greedy + CB*, los resultados son sorprendentes.

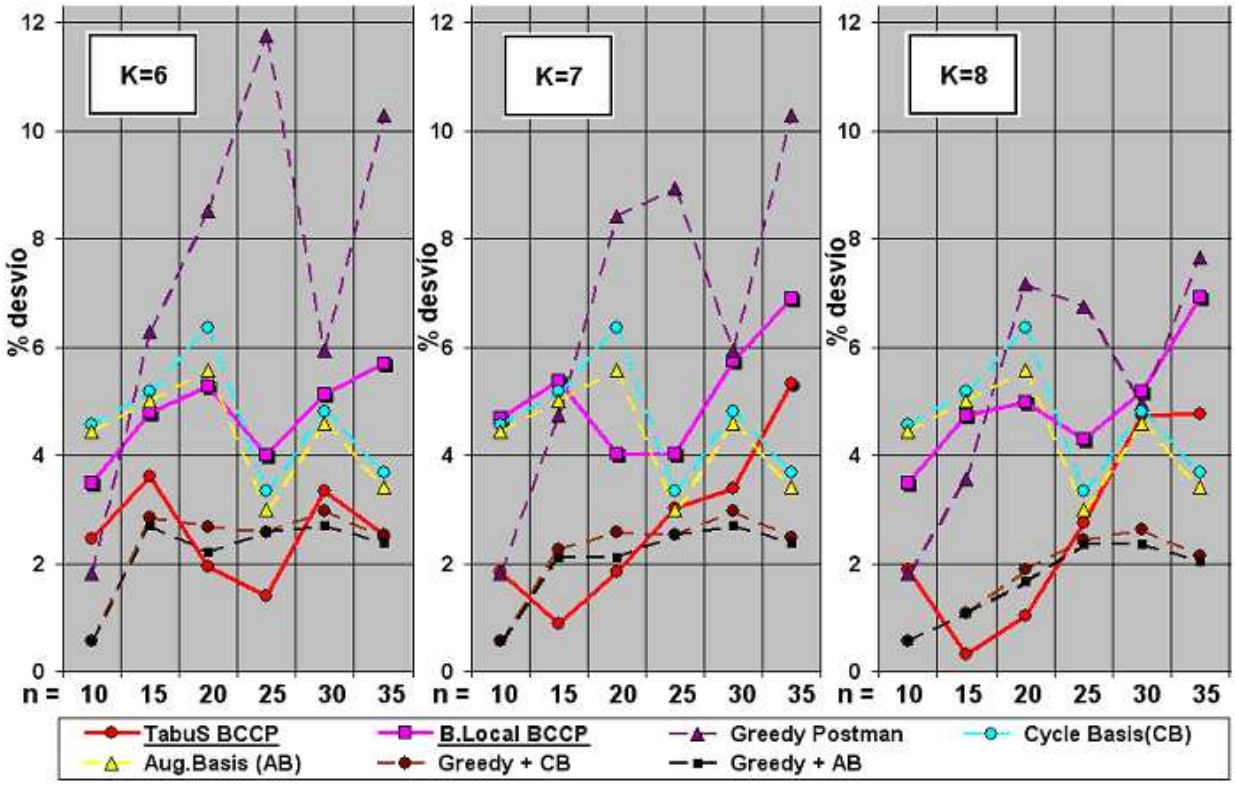


Gráfico 6.3.3 – Comparación de las cinco heurísticas a las cuales se les aplica Branch & Bound
Ensayos realizados con grafos euclideanos para K=6, K=7 y K=5

En estos ensayos, se puede observar a nuestro algoritmo de **Tabú Search** aventajando claramente a las tres heurísticas **Greedy Postman**, **Cycle Basis** y **Aug. Cycle Basis**, en la mayoría de los casos. El promedio total es de 6,49%, 4,66% y 4,34% de desvío respectivamente, contra 2,61%. Aunque al compararlo con los otros dos algoritmos, vemos que en algunos casos sobresale y en otros no. Si miramos los promedios totales, tenemos a **Greedy + CB** y **Greedy + AB** con 2,13% y 1,98%, contra 2,61%.

Para el algoritmo **B. Local BCCP**, vemos cómo se mantiene por debajo del resto, en términos de calidad de la solución. Aunque el promedio general es tan sólo 4,93%, o sea, está muy cerca de **Cycle Basis Aug.** y **Cycle Basis** (4,66% y 4,34%), y supera a **Greedy Postman** (6,49%), pero, como se observa en el gráfico 6.3.3, se encuentra alejado de **Greedy + CB** y **Greedy + AB** (2,13% y 1,98%) y de **Tabú Search BCCP** (2,61%).

El desvío estándar de **Tabú Search BCCP** oscila entre 0,3% y 3%, mientras que para **B. Local BCCP** va de 0,63% a 4,74%. En líneas generales se puede observar que a medida que el K y el tamaño del grafo aumenta, la calidad de la solución se degradar y el desvío estándar disminuye levemente.

La siguiente tabla muestra los porcentajes para los diversos ensayos con K = 6, 7 y 8, que dan origen a los gráficos anteriores, correspondientes a las figura 6.3.3.

Grafos		(%) Porcentajes de desvío respecto CPP							Desviación estándar	
K	# Nodos n	Greedy	% Cycle Basis (CB)	% Augmented Basis (AB)	% Greedy + CB	% Greedy + AB	% TabuS BCCP Opc.2	% B.Local BCCP Opc.2	Desviación estándar TabuS BCCP Opc.2	Desviación estándar B.Local BCCP Opc.2
6	10	1,82	4,57	4,44	0,56	0,56	**2,46**	**3,50**	**2,51**	**2,42**
6	15	6,28	5,18	5,02	2,85	2,71	**3,60**	**4,79**	**2,99**	**2,04**
6	20	8,52	6,35	5,58	2,67	2,20	**1,95**	**5,28**	**3,00**	**4,74**
6	25	11,77	3,34	2,99	2,59	2,59	**1,41**	**4,01**	**0,52**	**1,74**
6	30	5,94	4,82	4,60	2,98	2,71	**3,34**	**5,13**	**1,14**	**2,65**
6	35	10,29	3,69	3,41	2,49	2,38	**2,53**	**5,69**	**1,83**	**2,22**
7	10	1,82	4,57	4,44	0,56	0,56	**1,83**	**4,69**	**2,57**	**4,19**
7	15	4,75	5,18	5,02	2,26	2,12	**0,88**	**5,39**	**0,90**	**4,06**
7	20	8,43	6,35	5,58	2,58	2,11	**1,83**	**4,03**	**0,91**	**4,02**
7	25	8,95	3,34	2,99	2,53	2,53	**3,03**	**4,02**	**2,77**	**1,33**
7	30	5,94	4,82	4,60	2,98	2,71	**3,39**	**5,74**	**1,20**	**3,12**
7	35	10,28	3,69	3,41	2,49	2,38	**5,32**	**6,89**	**2,79**	**0,63**
8	10	1,82	4,57	4,44	0,56	0,56	**1,90**	**3,50**	**2,41**	**2,42**
8	15	3,56	5,18	5,02	1,07	1,07	**0,31**	**4,73**	**0,30**	**3,91**
8	20	7,18	6,35	5,58	1,90	1,67	**1,04**	**4,99**	**1,18**	**4,06**
8	25	6,76	3,34	2,99	2,42	2,37	**2,74**	**4,31**	**1,13**	**1,45**
8	30	4,95	4,82	4,60	2,63	2,37	**4,74**	**5,18**	**2,07**	**2,44**
8	35	7,67	3,69	3,41	2,14	2,03	**4,76**	**6,92**	**0,66**	**1,08**
Prom.		6,49	4,66	4,34	2,13	1,98	**2,61**	**4,93**	**1,72**	**2,69**
Total		116,73	83,85	78,12	38,26	35,63	**47,06**	**88,79**		

Tabla 6.3.4 – Resultados detallados del gráfico 6.3.3

Al comparar los tiempos empleados, como se observa en la siguiente tabla 6.3.5, para **B. Local BCCP**, notamos una mejora significativa respecto **Tabú Search**. Esto es un promedio total de 6,23 segundos contra 93,49, respectivamente. En términos absolutos, **B. Local BCCP** se encuentra muy cerca del resto de los algoritmos, que no superan el segundo. Aunque supera a **Greedy + AB**, que en promedio tarda 14,14 segundos.

En cuanto a **Tabú Search**, tenemos un tiempo promedio de 93,49 segundos respecto del resto que no supera el segundo. Para el caso del algoritmo **Greedy + AB**, en promedio tarda 14,14 segundos y nuestro algoritmo tabú insume 6.5 veces más tiempo que éste. Sin embargo, hay que destacar que el algoritmo de Branch & bound aplicado en estas heurísticas, es exponencial respecto la cantidad de ciclos.

En la siguiente tabla se observan los tiempos totales empleados por los diversos algoritmos. Al igual que en otras tablas, la anteúltima fila corresponde al promedio general de los tiempos

empleados estos ensayos (K = 6, 7 y 8), y la última fila corresponde a la suma total del tiempo empleado.

Grafos		Tiempo medido en segundos						
K	# Nodos n	Greedy	Cycle Basis (CB)	Augmented Basis (AB)	Greedy + CB	Greedy + AB	TabuS BCCP Opc.2	B.Local BCCP Opc.2
6	**10**	0,008	0,008	0,044	0,018	0,044	**1,547**	**0,334**
6	**15**	0,000	0,004	0,066	0,016	0,092	**37,941**	**1,059**
6	**20**	0,004	0,016	0,060	0,026	0,186	**36,250**	**1,525**
6	**25**	0,012	0,018	0,374	0,028	1,210	**42,941**	**2,234**
6	**30**	0,004	0,028	0,570	0,058	5,546	**52,197**	**1,866**
6	**35**	0,006	0,032	1,846	0,258	104,984	**66,941**	**2,856**
7	**10**	0,006	0,006	0,040	0,012	0,046	**2,956**	**0,416**
7	**15**	0,004	0,008	0,054	0,020	0,074	**55,181**	**1,563**
7	**20**	0,002	0,010	0,070	0,034	0,198	**138,816**	**3,569**
7	**25**	0,004	0,014	0,404	0,058	1,018	**97,791**	**3,347**
7	**30**	0,002	0,028	0,596	0,056	5,452	**126,034**	**4,837**
7	**35**	0,008	0,032	1,838	0,614	62,634	**194,269**	**9,297**
8	**10**	0,006	0,006	0,036	0,016	0,046	**3,134**	**0,372**
8	**15**	0,004	0,010	0,056	0,022	0,046	**62,975**	**1,994**
8	**20**	0,008	0,022	0,064	0,024	0,112	**140,216**	**5,253**
8	**25**	0,004	0,016	0,398	0,070	0,948	**176,637**	**8,731**
8	**30**	0,012	0,022	0,608	0,050	7,018	**184,744**	**8,557**
8	**35**	0,004	0,036	1,880	0,480	64,850	**262,237**	**54,310**
Prom.		0,01	0,02	0,50	0,10	14,14	**93,49**	**6,23**
Total		0,10	0,32	9,00	1,86	254,50	**1682,81**	**112,12**

Tabla 6.3.5 – Tiempos de los ensayos expuestos en 6.3.4

En el siguiente gráfico se observan ensayos realizados para K=6, 7 y 8, para la familia de grafos planares o ralos. Éstos son referenciados en el trabajo [8] con el nombre de Sparser Graphs. Por otro lado, fueron probados con los algoritmos **Greedy Postman**, **Augmented Cycle Basis** y **Greedy + AB**.
La principal diferencia de estos grafos respecto los euclidianos, es que cuentan con menos ejes, y esto hace que tengan espacios de soluciones más reducidos. Tal es así, que para el caso de K=4 no se han encontrado soluciones a excepción de un solo grafo. Por esta razón, y por no contar con resultados en el trabajo [8] para K = 4 y 5, excluimos a éstos del análisis.

A continuación, como se observa en gráfico 6.3.6, vemos una mejora significativa de nuestro algoritmo **Tabú Search BCCP** respecto de los otros algoritmos. En términos cualitativos, esto es un 2,97% de desvío en promedio, respecto de 9,58%, 19,57% y 5,70% para **Greedy Postman**, **Augmented Cycle Basis** y **Greedy + AB** respectivamente (ver la próxima tabla 6.3.7). En el caso de nuestro algoritmo **B. Local BCCP**, tenemos un promedio de 6,59%, es decir, levemente peor que **Greedy + AB**.

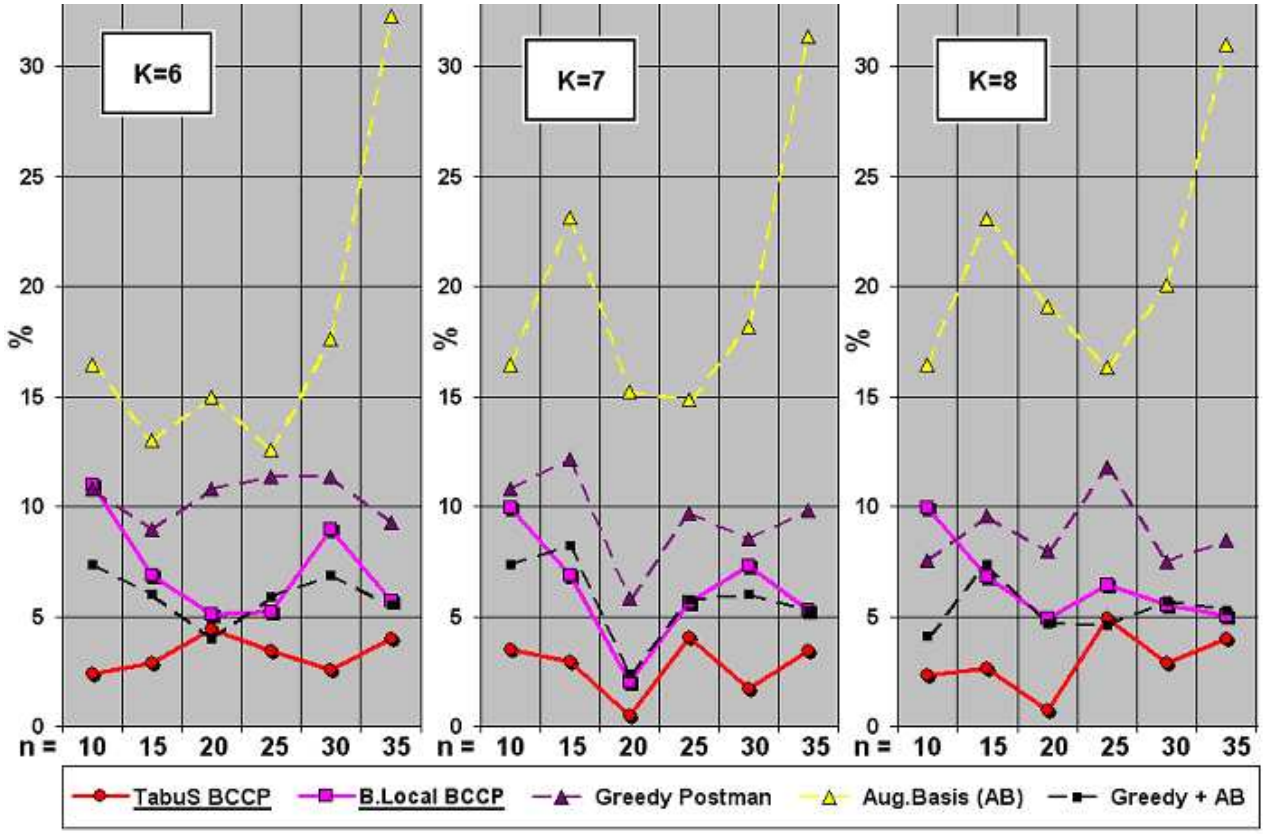


Gráfico 6.3.6 – Comparación de las cinco heurísticas a las cuales se les aplica Branch & Bound
Ensayos realizados con grafos ralos y/o sparser para K=6, K=7 y K=5

Al observar los tiempos empleados, tenemos que **Tabú Search BCCP** insume en promedio 16,78 segundos, aventajando claramente a **Greedy + AB** que tarda 49,19 segundos. En cambió, **B. Local BCCP** tarda 1,34 segundos, y esto lo ubica muy cerca de **Greedy Postman** y **Augmented Cycle Basis**, que consumen 0,06 y 0,9 segundos respectivamente.
Independientemente del K y del tamaño del grafo, el algoritmo **Tabú Search BCCP** obtiene resultados más o menos parecidos en términos de desvío respecto CPP, y estos no superan el 5%.
En el caso de **B. Local BCCP** vemos como mejoran los resultados a medida que K y el tamaño del grafo aumentan, aunque, a excepción de n=20, para K=7, éstos parecen tener un piso del 5%.
Por último, su desempeño tiene una similitud con **Greedy + AB**, tanto la curva que describen como el valor total y promedio alcanzados (118, 62% vs. 102,52% y 6,59% vs. 5,70%); aunque hay que recordar que el tiempo empleado es mucho menor (1,34 segundos contra 49,19).

En estos ensayos, el desvío estándar de **Tabú Search BCCP** oscila entre 0,45% y 5,23%, mientras que para **B. Local BCCP** va de 0,09% a 7,54%. **Tabú Search BCCP** tiene un desvío estándar promedio mayor en esta familia de grafos que en la euclideana: 2,85% contra 1,72% de la famita euclideana. Para el caso de **B. Local BCCP** ocurre algo siminar: 3,60 % contra 2,69%.

Grafos		(%) Porcentajes de desvío respecto CPP					Desviación estándar		Tiempo medido en segundos				
K	# Nodos n	% Greedy	% Augmented Basis (AB)	% Greedy + AB	% TabuS BCCP Opc.2	% B.Local BCCP Opc.2	Desviación estándar TabuS BCCP Opc.2	Desviación estándar Greedy BCCP Opc.2	Greedy	Augmented Basis (AB)	Greedy + AB	TabuS BCCP Opc.2	B.Local BCCP Opc.2
6	10	10,78	16,48	7,34	**2,41**	**11,02**	**4,54**	**4,93**	0,010	0,070	0,054	**0,437**	**0,141**
6	15	8,99	13,04	6,02	**2,86**	**6,85**	**3,87**	**2,94**	0,014	0,088	0,136	**0,583**	**0,172**
6	20	10,82	15,00	4,01	**4,40**	**5,07**	**5,06**	**0,09**	0,036	0,248	0,431	**12,172**	**0,383**
6	25	11,37	12,59	5,89	**3,45**	**5,23**	**2,25**	**0,61**	0,050	0,494	1,286	**4,371**	**0,516**
6	30	11,34	17,60	6,90	**2,58**	**8,98**	**2,36**	**3,04**	0,070	0,638	1,424	**11,823**	**0,531**
6	35	9,29	32,28	5,51	**3,98**	**5,70**	**2,67**	**2,69**	0,114	1,124	135,55	**3,833**	**0,636**
7	10	10,78	16,48	7,34	**3,50**	**9,97**	**4,05**	**5,93**	0,014	0,072	0,054	**0,318**	**0,156**
7	15	12,18	23,12	8,24	**2,92**	**6,86**	**5,23**	**7,54**	0,016	0,144	0,217	**2,336**	**0,293**
7	20	5,85	15,25	2,37	**0,51**	**2,05**	**0,45**	**2,72**	0,040	0,440	0,699	**33,927**	**1,120**
7	25	9,68	14,83	5,78	**4,04**	**5,62**	**2,79**	**0,91**	0,078	0,902	1,311	**22,938**	**0,975**
7	30	8,51	18,16	6,04	**1,71**	**7,28**	**2,07**	**2,77**	0,106	1,778	2,696	**24,562**	**1,113**
7	35	9,83	31,35	5,27	**3,46**	**5,29**	**1,36**	**2,75**	0,144	2,538	569,58	**14,379**	**1,297**
8	10	7,53	16,48	4,09	**2,32**	**9,97**	**4,74**	**5,93**	0,014	0,062	0,048	**0,630**	**0,266**
8	15	9,60	23,11	7,39	**2,67**	**6,83**	**3,79**	**7,44**	0,024	0,156	0,210	**1,712**	**0,538**
8	20	7,98	19,07	4,68	**0,75**	**4,90**	**0,57**	**4,94**	0,042	0,506	0,694	**21,664**	**1,160**
8	25	11,82	16,33	4,63	**4,91**	**6,44**	**1,84**	**1,74**	0,086	1,096	1,354	**40,146**	**2,255**
8	30	7,50	20,08	5,73	**2,91**	**5,51**	**1,73**	**4,62**	0,128	2,332	4,626	**50,415**	**2,706**
8	35	8,50	31,00	5,29	**4,02**	**5,05**	**1,96**	**3,16**	0,176	3,424	164,96	**55,797**	**9,884**
Prom.		9,58	19,57	5,70	**2,97**	**6,59**	**2,85**	**3,60**	0,06	0,90	49,19	**16,78**	**1,34**
Total		172,35	352,25	102,52	**53,40**	**118,6**			1,16	16,11	885,34	**302,04**	**24,14**

Tabla 6.3.7 – Resultados detallados del gráfico 6.3.6

Para finalizar, realizaremos una comparación de nuestras heurísticas con un algoritmo exacto al que denominaremos **UBColGen**. A su vez, dicho algoritmo funciona como heurística cuando no alcanza a procesar todas las soluciones posibles en el tiempo designado. Los ensayos fueron realizados para cinco casos puntuales de esta misma familia de grafos.

Grafo			Costo total de BCCP			(%) Porcentajes de desvío respecto CPP			Tiempo medido en hh:mm:ss		
K	# Nodos n	Costo CPP	UBColGen (algoritmo exacto)	TabuS BCCP Opc.2	B. Local BCCP Opc.2	% UBColGen (algoritmo exacto)	% TabuS BCCP Opc.2	% B. Local BCCP Opc.2	UBColGen	TabuS BCCP Opc.2	B. Local BCCP Opc.2
10	**20.1**	**1286**	1349	1297	1352	4,90%	0,86%	5,13%	00:20:06	00:00:54	00:00:01
5	**30.1**	**1514**	1626	1632	1656	7,40%	7,79%	9,38%	01:00:04	00:00:06	00:00:00
20	**30.1**	**1514**	1637	1522	1656	8,12%	0,53%	9,38%	02:00:04	00:04:27	00:00:06
10	**35.1**	**1501**	1650	1507	1556	9,93%	0,40%	3,66%	01:59:34	00:02:36	00:00:04
15	**35.1**	**1501**	1717	1953	1559	14,39%	30,11%	3,86%	04:13:07	00:32:07	00:00:11
Prom.		**1463**	**1595,80**	**1582,20**	**1555,80**	**8,95%**	**7,94%**	**6,28%**	**01:54:35**	**00:08:02**	**00:00:04**

Tabla 6.3.8 – Comparación del Algoritmo Exacto con Tabú Search y B. Local

A diferencia de los otros ensayos, en estos la medición de los tiempos excluye lo que tarda el algoritmo generador de ciclos.

Un aspecto a destacar es diferencia significativa que hay en los tiempos de cada heurística. Para **UBColGen** estamos hablando casi 2 horas en promedio, mientras que para **Tabú Search** sólo 8 minutos y para **B. Local** apenas unos 4 segundos.

En relación a la calidad de las soluciones, podemos observar cómo **Tabú Search** y **B. Local** en promedio aventajan levemente a **UBColGen**. Lo llamativo de estos ejemplos es que por primera vez **B. Local** supera en promedio a **Tabú Search**. Esto se debe simplemente a que en el grafo 35.1 para K=15, **Tabú Search** obtiene un pésimo resultado. Si consideramos a este caso como un outlier y lo excluimos de los resultados, los promedios de los algoritmos serían **7,59%**, **2,39%** y **6,89%** para **UBColGen**, **Tabú Search** y **B. Local** respectivamente. En ese mismo orden, los tiempos promedio asociados serían **01:19:57**, **00:02:01** y **00:00:03** respectivamente.
De esta manera **Tabú Search** tendría el porcentaje de desvío promedio y el tiempo promedio más cercano a los obtenidos en los ensayos anteriores. Para **UBColGen** y **B. Local** no varía mucho dicho porcentaje aunque disminuye significativamente el tiempo para el primero.

En líneas generales, se puede concluir que nuestros algoritmos ofrecen resultados competitivos, en tiempos razonables respecto los otros algoritmos expuestos.

7 Conclusiones

Como se ha podido observar en el desarrollo del presente trabajo, en este tipo de problemas se da siempre una estrecha relación entre el tiempo/esfuerzo empleado y los resultados obtenidos. Es decir, a mayor tiempo empleado, se obtienen mejores soluciones, no obstante ésta no es una relación lineal. Inicialmente se va mejorando en forma continua la solución buscada, para luego atenuarse y finalmente plancharse, describiendo de esta manera una asíntota. A partir de un cierto punto, es necesario invertir grandes cantidades de tiempo y/o esfuerzo para lograr una muy leve o nula mejora, y es en este punto donde queremos que nuestro algoritmo se detenga.

En nuestro problema, esta disyuntiva se vio reflejada en la intensificación del Tabú Search, donde se hizo hincapié en reforzar la búsqueda en una vecindad dada. Esto se logró creando una función de backtracking acotada en la cantidad de vecinos a explorar, que permitió lograr un equilibrio entre el tiempo empleado y la calidad de las soluciones encontradas para la mayoría de los casos. De este punto se desprende que explorar superficialmente una vecindad no conduce a encontrar buenos resultados, y explorarla en exceso, deteriora de manera significativa los tiempos empleados.

Asimismo, se redujo el espacio de soluciones de manera estratégica, lo que permitió un ahorro significativo en el tiempo empleado y una mejora sustancial en las soluciones encontradas. Contar con el especio completo de soluciones no es necesariamente bueno en este problema, aunque hay que destacar que se observó un comportamiento inestable y deficiente en grafos y K muy pequeños, de la misma forma que cuando la lista tabú es demasiado grande. Además, en este último caso, ocurrió frecuentemente que el algoritmo tabú empeoró en exceso la solución temporal, y luego se le dificultó volver a acercarse a la mejor encontrada, perdiendo tiempo y esfuerzo.

Afortunadamente, se pudo contar con grafos reales, y resultados sobre éstos [8], lo cual nos permitió establecer un marco referencia respecto a los nuestros. Además, el disponer de la implementación del Cartero Chino (CPP) [3], que constituye la cota inferior de nuestro problema, nos permitió saber el grado de desvío cometido en las soluciones encontradas respecto de ésta.

La conclusión tal vez más relevante de este trabajo, es que contamos con un algoritmo de Tabú Search y un algoritmo de Búsqueda Local, que ofrecen resultados competitivos para este problema, en tiempos razonables respecto de otros algoritmos. Asimismo, estos resultados están en promedio cercanos a la cota inferior.

Otro aspecto sumamente importante, fue la automatización de los ensayos, y la centralización de la información en una base de datos (Apéndice 10.2). Este complemento nos permitió procesar en reiteradas ocasiones todo el conjunto de datos, y llegar rápidamente a conclusiones generales o globales que nos fueron ayudando a reorientar la estrategia. Gracias a este soporte y a los resultados del trabajo [8], pudimos saber en todo momento donde estábamos parados y hacia donde nos dirigíamos.

Finalmente, algo que parece trivial, pero que en su momento no lo fue, es la elección del lenguaje de programación para la implementación del algoritmo. Inicialmente se había establecido como posibilidad C++, por contar con experiencia en el mismo, ser un lenguaje eficiente, etc. Pero luego de un breve análisis, se decidió que éste fuese Java, dado que tiene un uso más amigable, esencialmente en el uso de sus librerías, y fundamentalmente el manejo automático de memoria. A nuestro entender, esta elección nos permitió canalizar todo nuestro esfuerzo en el problema en sí, y no en la implementación del mismo. Parte del éxito del presente trabajo radicó en contar con una buena y eficiente implementación que permitió introducir fácilmente cambios significativos, extender el programa, reutilizar y encapsular funcionalidad, dándonos valiosos grados de libertad en todo el proceso de desarrollo.

8 Mejoras y Trabajo Futuro

Elección del generador de ciclos
El algoritmo y/o el usuario podrá optar qué generador de ciclos utilizar a partir de las dos variantes propuestas en este trabajo. El algoritmo lo hará en función del *K* y del tamaño de grafo. La idea es utilizar la alternativa de la sección 4.1.3 para *K* y grafos chicos y 4.1.4 para K y grafos medianos / grandes.

Generación de ciclos bajo demanda
Se podría adoptar una estrategia integral que genere ciclos bajo demanda a medida que el algoritmo de Tabú Search los requiera. Esta alternativa ahorra el tiempo que emplea el algoritmo generador de ciclos, dado que éste ya no sería necesario.

Criterio de parada
Si bien el criterio de parada es configurable, se puede implementar un time-out que pueda ser establecido por el usuario. La cantidad de iteraciones del backtracking puede ser también definida por el usuario y/o establecerse en función el time-out estipulado. Asimismo, se puede introducir la cota inferior como parámetro y en caso de ser ésta alcanzada por una solución, interrumpir la ejecución del algoritmo.

Movimiento transversal
Constituye una alternativa de movimiento tabú complementaria a la existente. Dada una solución *BCC* para un grafo *G*, obtenemos un eje e_1 y aplicamos el siguiente procedimiento:

```
e_x = e_1
for i =0 to #(BCC(e_x)) // donde #(BCC(e_x)) es la cantidad de
                        // ciclos de la solución BCC que atraviesan e_1
      Eliminamos un ciclo C_i cualquiera de BCC, que atraviese e_x
      Obtenemos un eje e_i de C_i cualquiera / e_x ≠ e_i
      e_x = e_i
End for
```

La idea es eliminar ciclos con ejes en común a lo largo de todo el grafo, intentando buscar impactar transversalmente, pero de manera superficial en la solución actual.
El objetivo es construir una vecindad que abarque soluciones de varias zonas distintas a las generadas por el actual *movimiento en profundidad*, para luego combinar ciclos de distintos extremos del grafo que cooperen para cubrir los ejes en común recientemente descubiertos.

Movimiento tabú
Dado que el actual moviendo tabú (*Movimiento en profundidad*) emplea un criterio un tanto fuerte (eliminar todos los ciclos del eje que lo atraviesan), se podría hacer un movimiento más suave en las primeras iteraciones del tabú, que es cuando éste mejora de manera progresiva. Luego, cuando los resultados se estancan, hacer movimientos bruscos con la esperanza encontrar un espacio de soluciones no explorado. Por ejemplo, al inicio del tabú se puede eliminar la mitad de los ciclos que atraviesan el eje seleccionado, y al final elegir dos o tres ejes al azar para luego eliminar todos ciclos asociados al BCC actual.

Intensificación
Una vez efectuado el Movimiento, se puede generar una solución utilizando el algoritmo goloso para que esta sirva de base al algoritmo de Backtracking que emplea la intensificación del Tabú.

Backtracking
Se puede hacer podas más sofisticadas contabilizando los ciclos que quedan por usar en un heap, para de esa manera determinar si los mismos superan el costo de la mejor solución encontrada por el backtracking hasta ese momento. El paso previo es definir una relación de orden en los ciclos para poder almacenarlos en el heap.

Lista tabú
A futuro, se puede hacer la lista tabú dinámica. De esta manera se busca que el algoritmo favorezca la intensificación cuando éste disminuya la lista. Por otro lado, puede favorecer la diversificación, cuando la lista es aumentada en tamaño. Del mismo modo, el algoritmo podría tener en cuenta el *K*, el tamaño del grafo, y la cantidad de generada de ciclos para dimensionarla en función de estos valores.
Las mejoras planteadas para la implementación pueden ser:

Modificación manual de soluciones
Se debe brindar al usuario la opción de poder seleccionar ciclos para la solución. En este contexto, el algoritmo deberá resolver el problema a partir de una solución parcial, respetando los ciclos establecidos de la misma. También se puede disponer de una opción que permita interrumpir y/o reanudar la ejecución del algoritmo. La interrupción arrojará la mejor solución encontrada hasta ese momento. La reanudación deberá admitir cualquier solución como base, siendo esta nula, parcial o completa.

Parámetros del algoritmo
Se implementó el procesamiento de grafos por lotes, donde en un archivo se define el listado de grafos a procesar. Como trabajo futuro, se puede incorporar a este mismo archivo, para cada grafo en particular o para el lote en general, los parámetros detallados en la sección 6.2.

Interfaz gráfica
Se puede construir una interfaz gráfica que permita visualizar en distintas capas el mapa geográfico donde se va a diseñar y/o construir la red, el grafo de representación de la misma, el cubrimiento BCC obtenido por el algoritmo y la solución del cartero chino. Además se podrá editar grafos, cubrimientos, mapas, entre otros. También se deberá contar con reportes que sinteticen el comportamiento del algoritmo, los parámetros empleados y otros datos de interés.

2CNBR
De igual modo, se puede desarrollar, en otro trabajo aparte, una resolución al problema *Two-Connected Network with Bounded Rings (2CNBR)*[7]. Dicho problema es estructuralmente muy parecido a BCCP, dado que tiene por objeto encontrar un cubrimiento de todos los nodos del grafo y no de todos los ejes como en nuestro problema. Por esa razón se puede aprovechar nuestra implementación, la automatización de pruebas, y todo el conocimiento desarrollado a lo largo de este trabajo para resolverlo.

9 Bibliografía

[1] AARTS E. – LENSTRA J. Local Search in Combinatorial Optimization. Wiley, (1998), pp 121-137.

[2] BALAKRISHNAN A. – ALTINKEMER K. Using a hop-constrained model to generate alternative communication network designs. ORSA Journal of Computing, Vol. 4, No. 2, (1998), pp. 192–205.

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

[4] COOK W. – ROCHE, A. Computing Minimum-Weight Perfect Matchings. INFORMS Journal on Computing, Vol. 11, No. 2, (1999), pp.138-148.

[5] FORTZ B. – SORIANO P. - WYNANTS, C. A tabu search algorithm for self-healing ring network design. European Journal of Operational Research, Vol. 151, No. 2, (2003), pp. 280-295.

[6] FORTZ B. – LABBÉ, M. A tabu search heuristic for the design of two-connected networks with bounded rings. Working Paper 98/3, Universidad Católica de Louvain, (2002).

[7] FORTZ B. – LABBÉ, M. Two-connected networks with rings of bounded cardinality. Computacional Optimization and Applications, Vol. 27, No. 2, (2004), pp. 123-148.

[8] HOCHBAUM D. – OLINICK E. The Bounded Cycle-Cover Problem. INFORMS Journal on Computing, Vol. 13, No. 2, (2001), pp. 104–119.

[9] LAGUNA M. Clustering for the Design of SONET Rings in Interoffice Telecommunications. Management Science, Vol. 40, No. 11, (1994), pp. 1533-1541.

[10] LAGUNA M. A Guide to Implementing Tabu Search. Investigación Operativa, Vol. 4, No. 1, (1994), pp.5-25

[11] LIU H. – WANG J. 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, (AICT/ICIW 2006).

[12] LOISEAU I. – MACULAN N. Ring networks design. Tesis de Doctorado, Universidad de Paris 13, (2005).

[13] RIBEIRO C. – ROSSETI I. A parallel GRASP heuristic for the 2-path network design problem. Lectura note in computer sience, Vol. 2400, (2002), pp. 922 a 926.

[14] THIMBLEBY H. The directed Chinese Postman Problem. Practices and Experience, Vol.33, No. 10, (2003), pp. 1081-1096.

10 Apéndice A: Detalles de Implementación

En esta sección se destacan los aspectos más relevantes de la implementación de las tres componentes centrales:

La primera y la más importante, el algoritmo de Tabú Search junto con sus clases soporte, implementado en Java.

La segunda y central para el análisis y la organización de la información generada por los ensayos: es la automatización de todos los resultados, utilizando una base de datos de Sql Server.

La tercera componente es un aplicativo que resuelve el Problema del Cartero Chino [14], de código fuente abierto, implementada en Visual C++.

10.1 Algoritmo Tabú Search

Se realizó una implementación en Java orientada a objetos, que se divide conceptualmente en tres partes:

1. **Manejo de entrada y salida**: se ocupa de la importación de grafos desde archivos de texto, recepción de parámetros para los ensayos y para el algoritmo, exportación de los resultados obtenidos, creación de un log con el detalle del comportamiento del algoritmo empleado en la corrida, datos en la pantalla para medir el progreso del algoritmo.

2. **Grafos y otros**: constituyen el soporte en todo lo relacionado a grafos, operaciones básicas, operaciones más sofisticadas como BFS, DFS, etc., y objetos de grafos relacionados con nuestro problema, como ser caminos / paths, ciclos, etc.

3. **Tabú Search BCC y complementarias**: se centra en lo referente al problema BCCP, y el algoritmo de tabú search que aplica para su resolución. Además, incluyen las clases complementarias que el tabú utiliza directa e indirectamente. Éstas comprenden los algoritmos generadores de ciclos, regiones de ciclos, objetos BCC específicos para el problema, etc.

En el siguiente diagrama 10.1.1, se observan los objetos más relevantes del segundo punto.

Los puntos a destacar son: se cuenta con una capa física de grafos y otra lógica. Ésta última permite ocultar y desocultar nodos y ejes en forma transparente a todas las operaciones que los acceden. Los algoritmos BFS, DFS y otros, utilizan la capa lógica de grafos.

El almacenamiento interno de los ciclos está normalizado, prohibiendo de esta forma los ciclos redundantes, tal como se expuso en la sección 4.1.2. A su vez, estos son almacenados en un hashset, y se le definió una función de hashcode. Esto hace que la comparación por la igualdad de ciclos sea O(1) en la gran mayoría de los casos (cuando el hash code es distinto), y $O(C_k)$ en el peor caso (cuando coinciden ambos hashcode que se están comparando), donde C_k es la cantidad de ejes que tiene el ciclo.

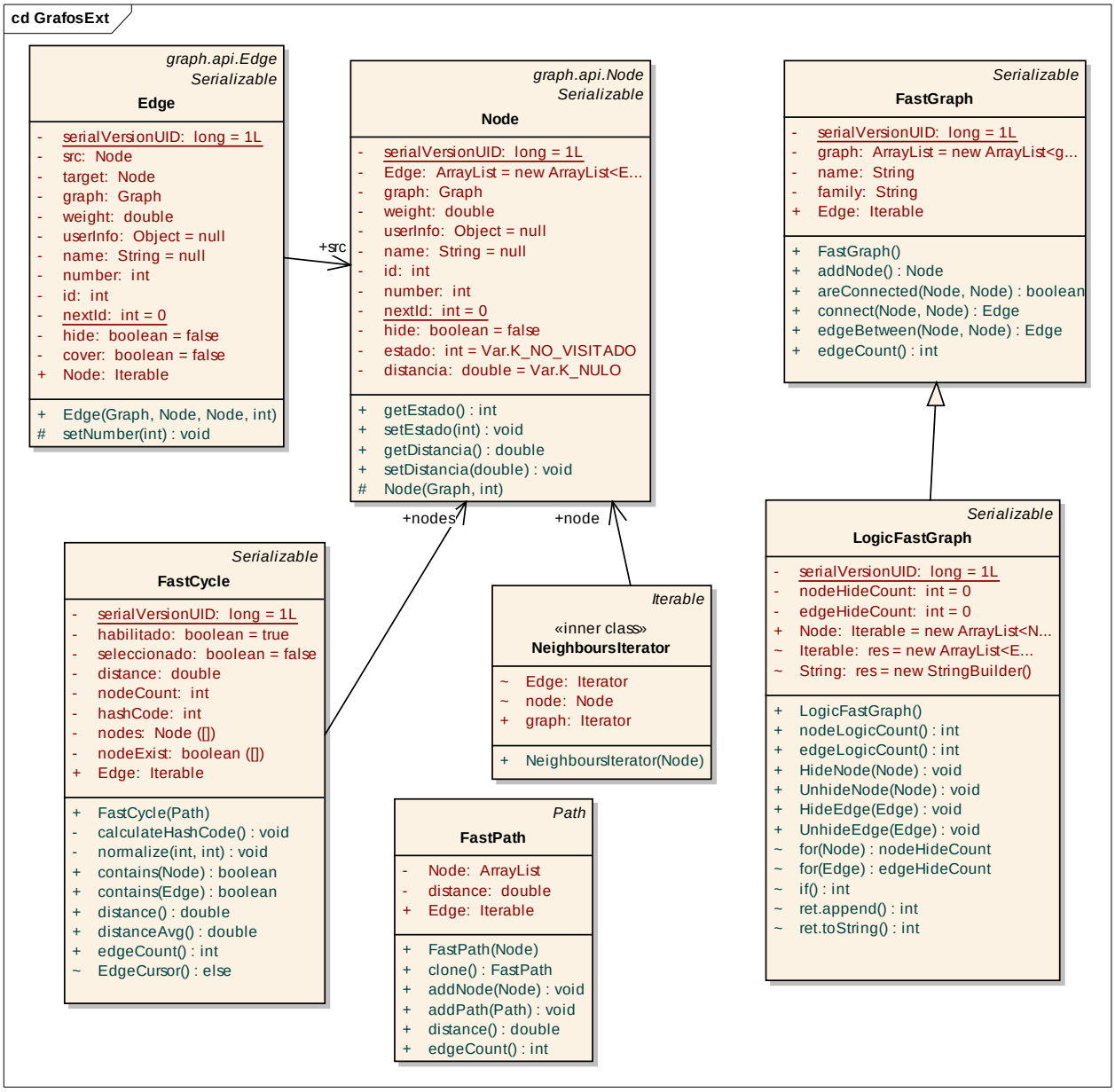


Diagrama 10.1.1 – Implementación de la clase grafo

El siguiente diagrama corresponde a lo expuesto en el punto 3:

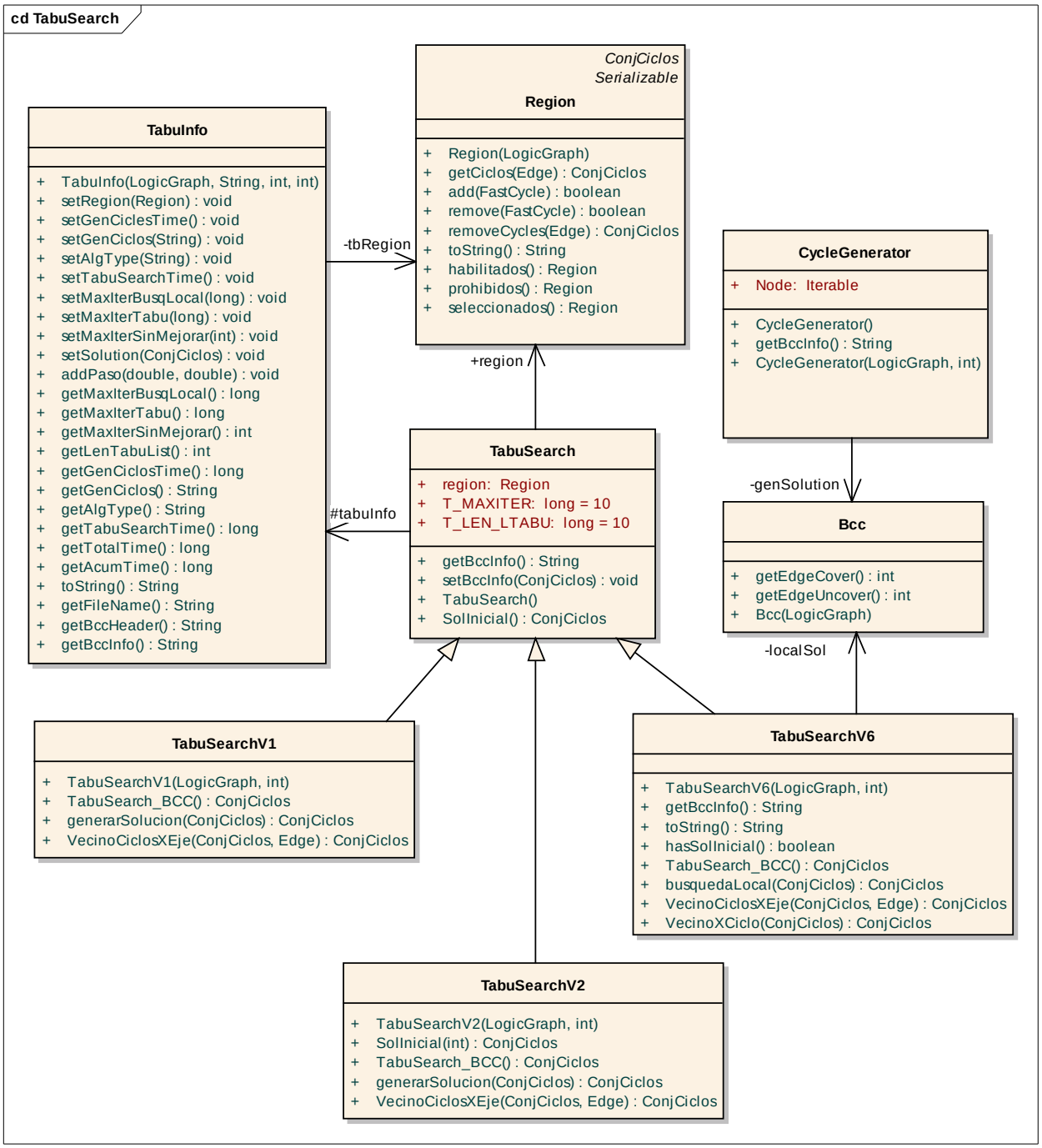


Diagrama 10.1.2 – Implementación del Tabú Search y sus clases soporte

En esencia, se cuenta con una clase tabú search general y diversas instancias de ésta, que se corresponden con las distintas versiones del algoritmo tabú que se fueron implementando y evolucionando, a lo largo de este trabajo. Cada nueva versión, re-implementa el algoritmo de tabú search propiamente dicho, y reutiliza objetos, propiedades y métodos comunes a las distintas versiones.

Entre los más destacados, podemos mencionar:

- Algoritmo generador de ciclos
- TabúInfo: permite administrar los parámetros de entrada del tabú search que, tal como se vio en la sección 6.2, que habilitan modificar sustancialmente el comportamiento del algoritmo. Por otro lado, permite contabilizar el tiempo transcurrido del algoritmo en sus dos fases. Toda esta información, parámetros, resultados, y tiempos, fue expuesta a lo largo de la sección 6.

Luego, los principales objetos que son utilizados por el tabú search, ya sea a nivel general, como en las instancias en particular, son:

- Región: administra un conjunto de ciclos, y además permite habilitar y/o prohibir determinados ciclos.
- BCC: permite una fácil manipulación de una solución parcial y/o total. Esta clase modela un cubrimiento de ciclos y/o BCC, y todos sus métodos y atributos están enfocados en brindar, incorporar y eliminar información de manera eficiente. Por ejemplo, saber si se ha formado un BCC es O(1). El costo de un cubrimiento, sea éste parcial o total, (formando este último BCC), tiene O(1).
- CycleGenerator: se ocupa de generar ciclos. Provee los algoritmos presentados en las secciones 4.1.3 y 4.1.4.

10.2 Automatización de pruebas

Se automatizó la información generada en los múltiples ensayos, en sus diversas etapas. La idea principal es estructurar y centralizar la información surgida a partir de los miles de ensayos realizados, para poder fácilmente consultarla, y de esta forma facilitar y enriquecer el análisis.
Las cuatro etapas que involucran el flujo de la información son:

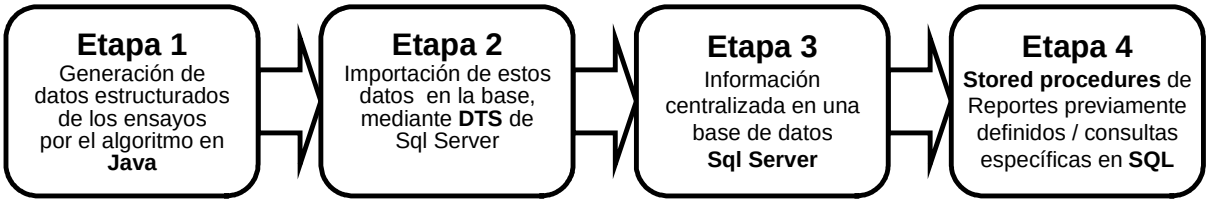


Diagrama de flujo 10.2.1

Para la base de datos, se diseño un modelo estrella, como los empleados en datawarehouse, cubos de información y/o tableros de control. Este modelo consta de una tabla dinámica de hechos (Fact Table) y distintas tablas estáticas y/o semiestáticas denominadas dimensiones.

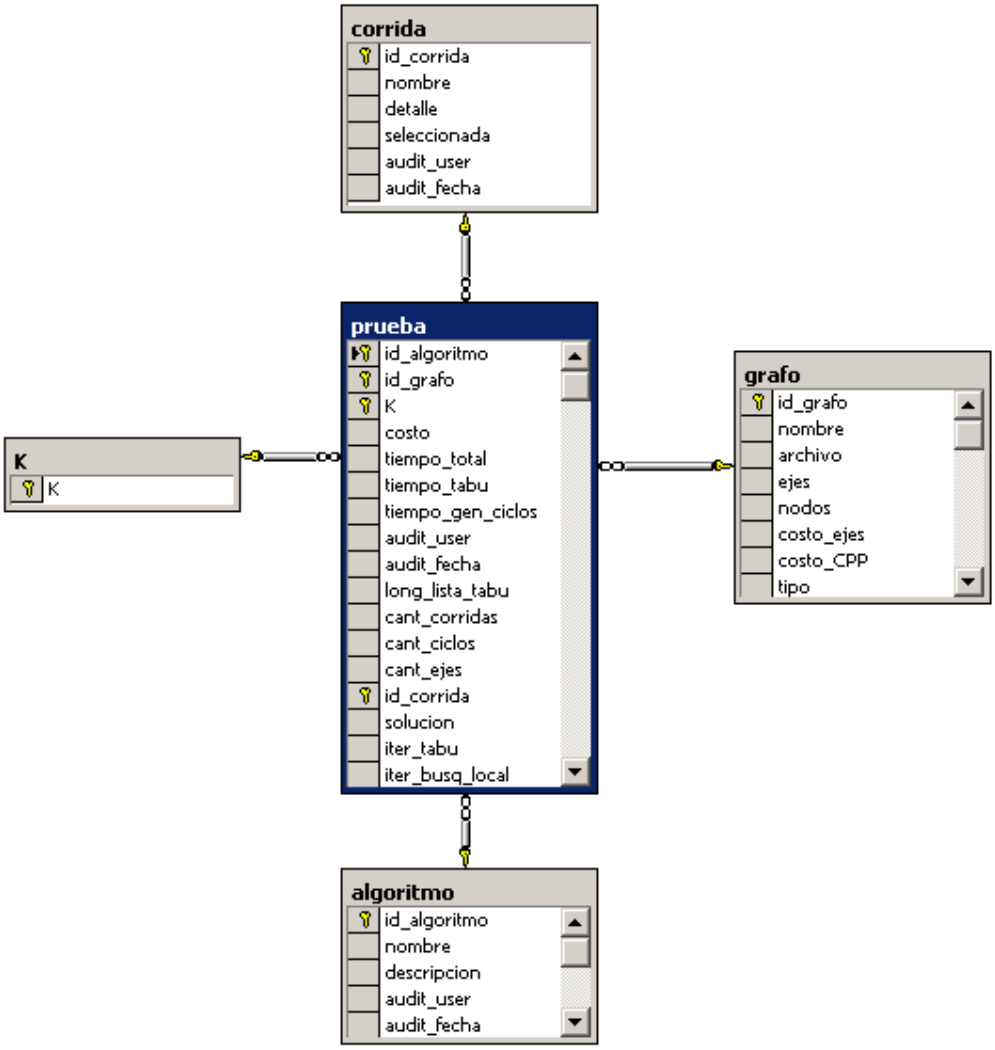


Diagrama 10.2.2- Modelo de datos de la base que almacena los resultados

En nuestro modelo, las dimensiones están constituidas por:

- **K**: son todos los valores posibles de K en los que se pueden realizar los ensayos.

- **Algoritmo**: cada algoritmo, sean estos búsqueda local, backtracking, tabú search; ó cada nueva versión de un algoritmo en particular, como las diversas versiones de tabú search, son dados de alta en esta tabla con un nombre que lo identifique y una breve descripción.

- **Grafo**: almacena información relevante para nuestro problema, de los distintos grafos utilizados en los ensayos. La información está dada por: tipo de grafo (actualmente DENSE o EUCLIDEAN), cantidad de nodos, cantidad de ejes, costo total de todos sus ejes y costo total del cartero chino (el cálculo de este último valor, se verá en la siguiente sección 10.3).

- **Corrida**: a las tres dimensiones anteriores, las podemos caracterizar como estáticas con algún grado de variación. En cambio, a ésta se le agrega un registro cada vez que se procesa un lote de grafos, diferenciándola del resto. El objetivo es analizar el comportamiento de un algoritmo para una y/o varias *corridas* en particular. Cada una de ellas cuenta con una descripción, fecha y hora en que fue realizada, y finalmente un valor que indica si va ser incorporada a los resultados obtenidos, o simplemente desechada (como en el caso de pruebas experimentales).

Luego, la tabla de hechos es **Prueba**. Los datos generados en la generación de ciclos tienen por objeto poblar a ésta. Inicialmente, el algoritmo en JAVA procesa un lote de grafos para múltiples valores de K, generando como resultado el archivo *ensayos.txt.* Este archivo es tomado por el DTS, e importado en la base (Etapa 2). Primero se crea un registro en la tabla **Corrida**, y luego se importan los valores de la tabla **Prueba**, que luego se vinculan a la última corrida ingresada. De esta manera, ya contamos con la información correctamente almacenada (Etapa 3). Finalmente, se ejecuta el stored procedure **sp_error_relativo**, que permite obtener un reporte con los promedios del desvío relativo respecto CPP, agrupado por K y n, tal como se calcula en el trabajo [7], en las páginas 114, 115, 116, 117 y 118. También permite obtener los tiempos promedios empleados en las respectivas fase 1 y 2, e información complementaria como los parámetros utilizados, cantidad de ciclos promedio de las soluciones, etc.

La forma de invocar a este stored es la siguiente:
exec sp_error_relativo @modo, @id_corrida, @id_algoritmo, @grafo, @k, @n

Cuyos parámetros significan:

- @modo: puede ser 'AGRUPADO' o 'DETALLE'. El primero promedia los resultados obtenidos, % desvío relativo, tiempos, etc., agrupados por K y n; mientras que el segundo muestra todos los resultados, agregando una columna con el detalle de la solución encontrada.
- @id_corrida: filtra por una corrida en particular. Si el valor ingresado es null, entonces obtiene automáticamente *la última corrida ingresada*.
- @id_algoritmo: filtra por un algoritmo en particular. Si el valor es null, involucra a todos los algoritmos.
- @grafo: permite filtrar por un grafo en particular.
- @k: permite filtrar por un K en particular. Si el valor es null, abarca a todos los K para los que se encuentren resultados.
- @n: permite filtrar por la cantidad de nodos de un grafo, siendo estos: 10, 15, 20, 25, 30, 35 y 40. Al igual que en otros caso, si éste valor es null, no aplica ningún filtro.

El ejemplo más utilizado, en el cual nos interesa obtener los porcentajes de desvío respecto CPP, y los tiempos promedios, para luego comparar con el trabajo [7], es:

```
exec sp_error_relativo 'AGRUPADO',null, null, null, null, null
```

Luego, si queremos ver el detalle de los ensayos para grafos de n=35, y K=8, para la corrida 113, tenemos que ejecutar el siguiente comando de SQL:

```
exec sp_error_relativo 'DETALLE',113, null, null, 8, 35
```

Por último, hay que destacar en la etapa 1 que, mientras se genera el archivo *ensayos.txt* con todos los resultados de la corrida, por cada grafo procesado, se crea en un directorio específico, un archivo con todos los datos relacionados a ese ensayo. Ejemplo:

10.1_stV6_K4.st
Algoritmo: TabuSearchClasicV6
Grafo: EUCLIDEAN_10_1
Region: 27 Ciclos
K: 4 Long. Lista Tabu: 100
Iter.Tabu: 2000 Iter.Busq.Local: 100000
Tiempo Gen.Ciclos: 46
Tiempo Tabu Search: 329
Tiempo Total: 375

Nodos: 10 Ejes:21
Ciclos: 7 Costo:21805.0
1 4687.0 [2, 6, 4, 8,]
2 1967.0 [1, 7, 6, 9,]
3 5438.0 [0, 3, 5,]
4 1588.0 [2, 4, 3, 8,]
5 3170.0 [0, 1, 6, 7,]
6 1479.0 [1, 5, 9,]
7 3476.0 [0, 4, 7,]

Paso Sol. Actual Mejor Sol.
1 34333.0 34333.0
2 30770.0 30770.0
3 30770.0 30770.0
4 27626.0 27626.0
5 27626.0 27626.0
6 28971.0 27626.0
7 27161.0 27161.0
8 27161.0 27161.0
9 27483.0 27161.0
10 27483.0 27161.0
11 27097.0 27097.0
...Continúa...

Además de la información que se agrega a *ensayos.txt*, a este archivo se agrega los valores que el algoritmo obtiene en cada iteración tabú, junto con el mejor valor actual encontrado. Luego, esta información permite analizar el comportamiento del algoritmo a lo largo de toda la corrida, tal como se vio en la sección 6.3.

De esta forma quedaría constituida la automatización del flujo de datos.

Como trabajo futuro, se puede pensar en extender la base de datos, incorporando la corrida completa realizada por cada ensayo. Esto es, la información recién antes expuesta.

También se podría incorporar los grafos con sus nodos y ejes en el modelo relacional de la base; y de la mano de esto, todos los ciclos posibles de K=3, 4, 5, 6, etc., para cada grafo. La idea de esto, es ver qué ciclos obtiene el algoritmo generador de ciclos respecto del total, y luego ver qué ciclos son incluidos en la solución encontrada, para luego intentar encontrar algún patrón. Quizás, podríamos encontrar que en las soluciones prevalecen los ciclos con K grande, o los que su costo promedio por eje (costo total del ciclo / cantidad de ejes) es menor que el resto.
Finalmente, se puede agregar la solución completa del problema del cartero chino para cada grafo.

10.3 Cartero Chino

Para el problema del cartero chino (CPP), contamos con una aplicación que lo resuelve [14].
Esta aplicación, junto con toda la documentación, se encuentra en el cd adjunto, en la carpeta CPP.

De fácil manejo, consiste en editar el grafo requerido, y luego oprimir el botón señalado por el círculo rojo. A continuación mostramos el mismo ejemplo de la sección 2.1, figura 2.1.1.

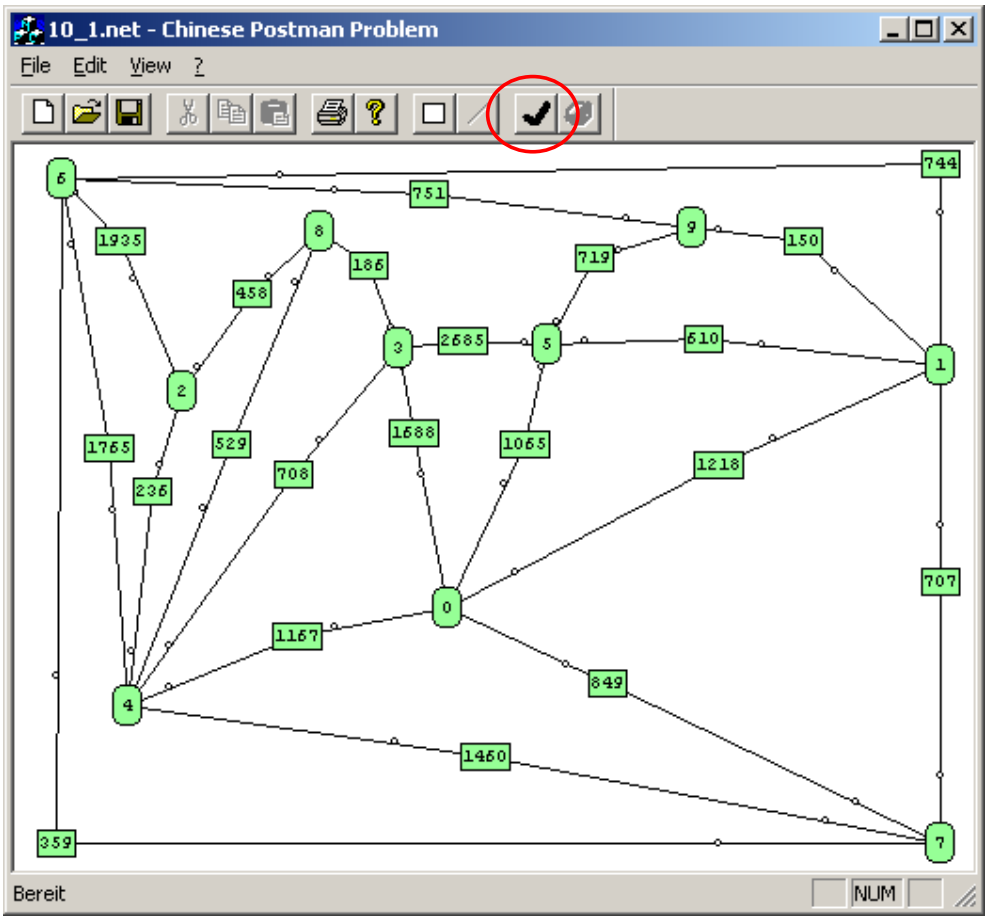


Figura 10.3.1 – Aplicativo que resuelve el Cartero Chino.

En el resultado se muestra los ejes que se agregan, repitiéndose tal como muestra la figura 2.4.5. Estos son $\{(0,7),(1,9),(2,8),(6,7)\}$:

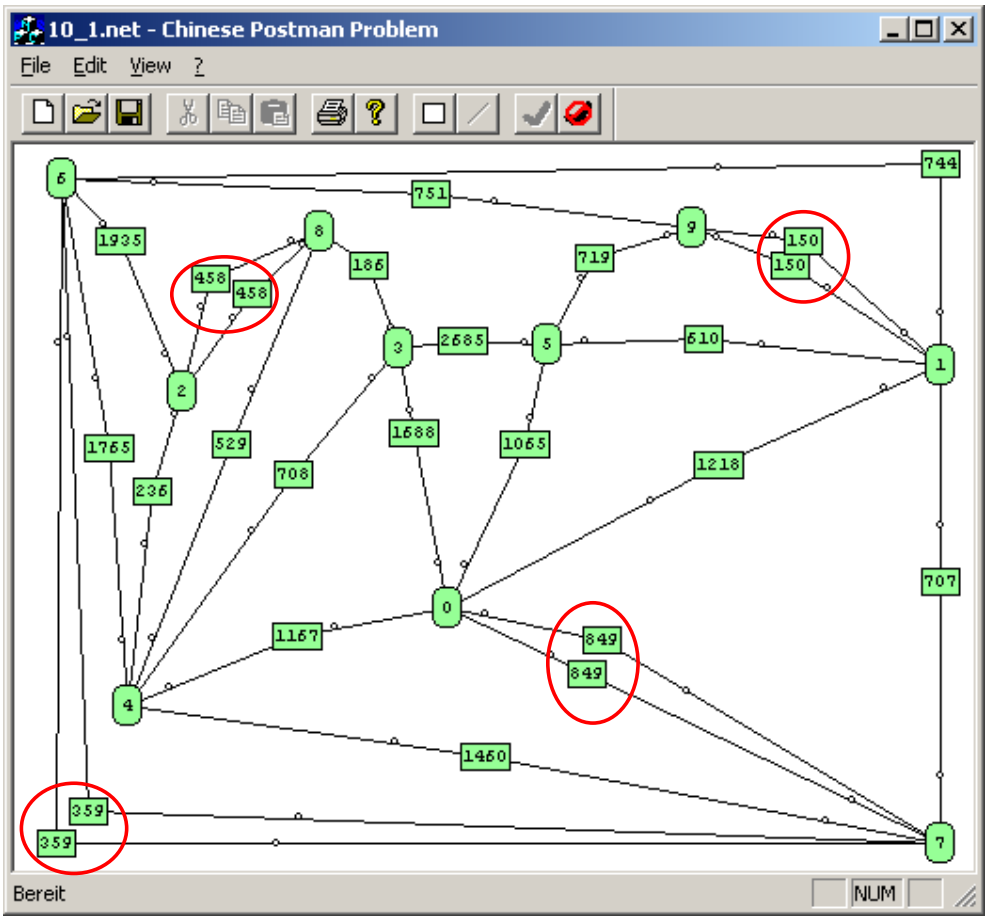


Figura 10.3.2 – Resolución en el aplicativo para el graf 2.1.2.
Este resultado es similar al expuesto en 2.4.5

Finalmente, la herramienta genera en el archivo **eulTrails.txt**, el postman tour hallado y ciclos encontrados, similar al expuesto en las tablas 2.4.3 y 2.4.4 respectivamente.
A continuación se muestra el archivo **eulTrails.txt**:

```
This is a list of small eularian trails
(8,0,9)   0->1
(9,7,5)   1->7
(5,4,8)   7->0

(0,1,8)   3->0
(8,2,2)   0->4
(2,12,0)  4->3

(3,3,8)   5->0
(8,22,5)  0->7
(5,19,4)  7->6
(4,6,9)   6->1
(9,5,3)   1->5

(7,8,9)   9->1
(9,24,7)  1->9
```

```
(2,9,1)   4->2
(1,11,6)  2->8
(6,17,2)  8->4

(4,10,1)  6->2
(1,21,6)  2->8
(6,14,0)  8->3
(0,13,3)  3->5
(3,18,7)  5->9
(7,20,4)  9->6

(4,15,2)  6->4
(2,16,5)  4->7
(5,23,4)  7->6

The Trail
(8,0,9)   0->1,1218
(9,24,7)  1->9,150
(7,20,4)  9->6,751
(4,15,2)  6->4,1765
(2,16,5)  4->7,1460
(5,23,4)  7->6,359
(4,10,1)  6->2,1935
(1,21,6)  2->8,458
(6,14,0)  8->3,186
(0,13,3)  3->5,2685
(3,18,7)  5->9,719
(7,8,9)   9->1,150
(9,5,3)   1->5,610
(3,3,8)   5->0,1065
(8,22,5)  0->7,849
(5,19,4)  7->6,359
(4,6,9)   6->1,744
(9,7,5)   1->7,707
(5,4,8)   7->0,849
(8,2,2)   0->4,1167
(2,9,1)   4->2,236
(1,11,6)  2->8,458
(6,17,2)  8->4,529
(2,12,0)  4->3,708
(0,1,8)   3->0,1688

TOTAL:21805
TOTAL Ejes:19989
```

Archivo de texto 10.3.3

Por tratarse de una aplicación de código fuente abierto, hecha en Visual C++, para hacer más amigable la operatoria, se incorporó dentro de esta herramienta un parser para nuestros grafos, que están almacenados en archivos de texto plano. Además, se le agregaron al archivo de salida las últimas dos líneas resaltadas en negrita: TOTAL y TOTAL Ejes: la primera calcula el costo total del cartero chino, y la segunda, la suma total de los ejes del grafo. El primer valor es almacenado en la base de datos, en la tabla grafo, en el campo costo_CPP al grafo correspondiente. El segundo lo almacenamos en el campo costo_ejes, y sirve como control para saber que el grafo fue correctamente incorporado y procesado por la herramienta.

Si se quiere importar nuestro grafo de ejemplo en la aplicación a partir de un archivo plano, se deberá cumplir con la siguiente sintaxis:

```
999
10
0
```

```
1
2
3
4
5
6
7
8
9
0 1 1218
0 3 1688
0 4 1167
0 5 1065
0 7 849
1 5 610
1 6 744
1 7 707
1 9 150
2 4 236
2 6 1935
2 8 458
3 4 708
3 5 2685
3 8 186
4 6 1765
4 7 1460
4 8 529
5 9 719
6 7 359
6 9 751
```

Archivo de texto 10.3.4

Este formato es el mismo utilizado por nuestro algoritmo en Java, donde los nodos se enumeran de 0 a n-1, a excepción del primer número 999 que nuestra aplicación no lo necesita.

Finalmente, hay que tener en cuenta que al importar el grafo a la herramienta, la construcción visual se hace de una manera estándar no tan legible; aunque, dado que el resultado detallado es un archivo de texto, poco nos interesa. En nuestro ejemplo tenemos:

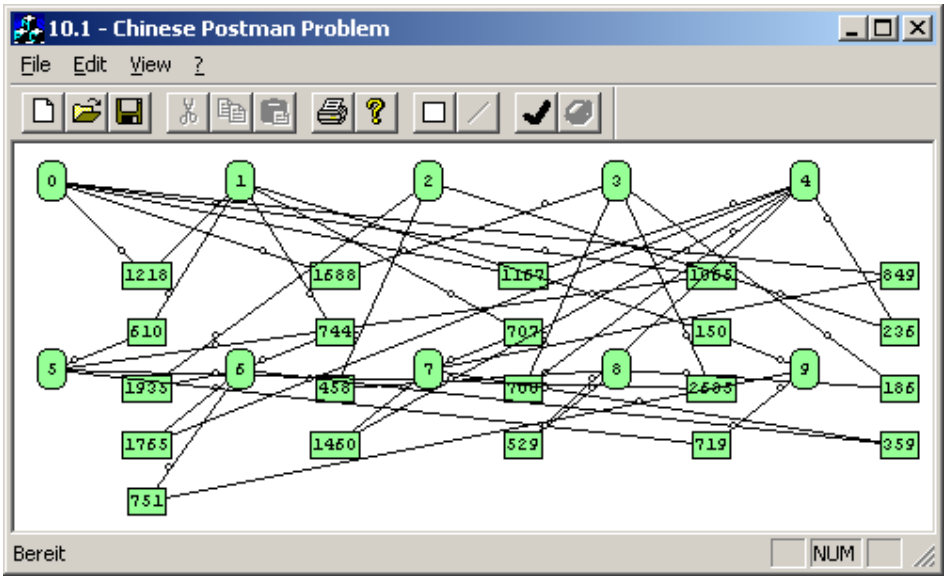


Figura 10.3.5 – Importación del grafo 2.1.2 en el aplicativo

11 Apéndice B: Contenido digital del CD

11.1 Papers

En la carpeta 01_papers se encuentran los papers [2], [5], [6], [7], [8], [9], [11] y [13].

11.2 Fuentes

Implementación Tabú Search

En la carpeta 02_fuentes\01_java_bccp se encuentra el proyecto de Java que implementa al los algoritmos Tabú Search y Búsqueda Local. Este proyecto fue desarrollado en Eclipse v3.2.
Los dos archivos más importantes que el usuario necesita configurar son:

- Var.Java: se encuentra dentro del package src\graph\api y contiene los paths de los directorios donde se exportan los resultados de los ensayos.
- Prueba.Java: se encuentra dentro del package scr\test y corresponde al menú principal de la implementación. Dentro de este archivo, la función GraphBuilder.cargar_archivos llama a un archivo de texto (grafos_dense.txt en el código) cuyo contenido es un listado de los grafos que se van a procesar tipo batch. Al finalizar la ejecución, la aplicación genera el archivo ensayosTBSC.txt, cuyos resultados serán luego tomados por el DTS que los importa a la base de datos.

Base de datos

En 02_fuentes\02_base_de_datos**,** se encuentra un backup completo de la base de datos detallada en 10.2. Dicha base de datos alberga a todos los ensayos realizados a lo largo del proyecto. También hay un DTS el cuál es el encargado de importar los datos a la base, cuyo origen es el archivo ensayosTBSC.txt.

Cartero Chino

Dentro de la carpeta 03_cpp podemos encontrar las siguientes carpetas con sus respectivos contenidos:

- release: se encuentra el propragma que resuelve el cartero chino
- ejemplos_cpp: ejemplos de grafos utilizados en los ensayos que fueron creados y editados con esta implementación
- documentacion: a partir del archivo index.htm de esta carpeta se puede acceder a la documentación de diversos algoritmos que el autor [3] nos provee. En particular, se encuentra el del CPP, disponible tanto en inglés como en aleman.
- Fuentes: por último, en esta carpeta se encuentra el código fuente de la implementación. Es requisito contar con Visual C++ correspondiente al Visual Studio 6.0, para poder trabajar con dicho código. Con estos requisitos, sólo resta acceder al archivo del proyecto cpp.dsw.

11.3 Ensayos

El archivo ensayos.xls que se encuentra en la carpeta 03_ensayos, muestra los resultados de cada variante de algoritmo Tabú Search o Búsqueda Local, donde cada corrida involucra a 150 ensayos aproximadamente. En esta planilla se puede ver cada uno de ellos, pero la solución, es decir, el cubrimiento encontrado se encuentra truncado por las limitaciones físicas del Excel (cada celda no puede superar los 2000 caracteres). Por esa razón, cada solución correspondiente a cada ensayo fue almacenada en un archivo de texto cuya extensión es .out. En las siguientes tablas se muestra la ubicación de los ensayos realizados para cada algoritmo

ID_CORRIDA en la Base	VARIANTE	GRAFICOS donde se encuentra	TABLAS donde se encuentra	Ubicación en el CD partiendo de la carpeta: 03_ensayos\03_TabuSearch\V1\EUCLIDEAN
111	TabuS Opc.2	6.1.1; 6.1.2; 6.2.2; 6.2.3; 6.3.1 y 6.3.3	6.1.3; 6.1.4; 6.2.1; 6.2.4; 6.3.2; 6.3.4; 6.3.5	2008-09-28_1700_V6-TabuS Op2
113	B.Local Opc.2	6.1.1; 6.1.2; 6.3.1 y 6.3.3	6.1.3; 6.1.4; 6.3.2; 6.3.4 y 6.3.5	2008-10-05_2130_V9-Greedy Opc2
116	TabuS Opc.1	6.1.1 y 6.1.2	6.1.3 y 6.1.4	2008-11-09_0400_V6-GenCycle
117	B.Local Opc.1	6.1.1 y 6.1.2	6.1.3 y 6.1.4	2008-11-09_1520_V6-Greedy-GenCycle
110	TabuS Opc.2 Plus	6.1.1; 6.1.2;6.2.2 y 6.2.3	6.1.3; 6.1.4; 6.2.1 y 6.2.4	2008-09-26_0900_V6-TabuS Op2 plus
128	TabuS Opc.2 - A	6.2.2 y 6.2.3	6.2.1 y 6.2.4	2008-12-14_1130_V6-TabuS Op2-A
123	TabuS Opc.2 - B	6.2.2 y 6.2.3	6.2.1 y 6.2.4	2008-12-07_2300_V6-TabuS Op2-B
125	TabuS Opc.2 - C	6.2.2 y 6.2.3	6.2.1 y 6.2.4	2008-12-11_0020_V6-TabuS Op2-C
127	TabuS Opc.2 - D	6.2.2 y 6.2.3	6.2.1 y 6.2.4	2008-12-14_0330_V6-TabuS Op2-D

Tabla 11.3.1 – Indica las carpetas que albergan los resultados para la familia de grafos densos

ID_CORRIDA	VARIANTE	GRAFICOS donde se encuentra	TABLAS donde se encuentra	Ubicación en el CD partiendo de la carpeta: 03_ensayos\03_TabuSearch\V1\DENSE
130	TabuS Opc.2	6.2.5 y 6.3.6	6.2.6 y 6.3.7	2008-12-14_2300_V6-Iter300-150out
139	B. Local Opc.2	6.3.6	6.3.7	2008-12-28_1750_V9-Greedy Op2
136	TabuS Opc.2 Plus	6.2.5	6.2.6	2008-12-24_1730_V6-TabuS Op2 Plus
132	TabuS Opc.2 - A	6.2.5	6.2.6	2008-12-24_1320_V6-TabuS Op2-A
133	TabuS Opc.2 - B	6.2.5	6.2.6	2008-12-24_1350_V6-TabuS Op2-B
134	TabuS Opc.2 - C	6.2.5	6.2.6	2008-12-24_1420_V6-TabuS Op2-C
135	TabuS Opc.2 - D	6.2.5	6.2.6	2008-12-24_1540_V6-TabuS Op2-D

Tabla 11.3.2 – Indica las carpetas que albergan los resultados para la familia de grafos ralos y/o sparser

Cada grafo procesado cuenta con dos archivos:

- 10.1_BCCv6_K4.out: archivo correspondiente al grafo 10.1 que fue procesado para K=4. Este archivo contiene el detalle del cubrimiento hallado.
- 10.1_BCCv6_K4.st: idem anterior, solo que este archive contiene la solución detallada, los parámetros involucrados, y los costos totales hallados en cada iteración realizada por el algoritmo. Con estos últimos fueron realizados los gráficos 6.2.7, 6.2.8, 6.2.9, 6.2.10, 6.2.11 y 6.2.12.

Por último, en el archivo EUCLIDEAN-OTROS.ZIP de la carpeta 03_ensayos\03_TabuSearch\V1\EUCLIDEAN se encuentran el resto de los ensayos realizados que fueron descartados.

11.4 Tesis

En la carpeta 04_tesis se encuentra el presente documento en formato Word y PDF.

11.5 Programas

En la carpeta 05_Programas figuran las siguientes aplicaciones empleadas para el desarrollo de este trabajo:

- **Instalador de Java Virtual Machine V6 Actualización 5**
- **Eclipse 3.2**: no es necesario instalarlo. Sólo hay que descomprimir el archivo eclipse-SDK-3.2-win32.zip y luego ejecutar el archivo eclipse.exe. Es requesito contar con la Java Virtual Machina instalada.