INSTITUTO TECNOLÓGICO DE CIUDAD MADERO

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

TESIS

SOLUCIÓN DEL PROBLEMA ANTIBANDWIDTH CÍCLICO USANDO MÉTODOS EXACTOS

Para obtener el grado de:

Maestro en Ciencias de la Computación

Presenta:

Ing. Fanny Gabriela Maldonado Nava

Director de tesis:

Dra. Claudia Guadalupe Gómez Santillán

Co-director de tesis:

Dra. Laura Cruz Reyes

Ciudad Madero, Tamaulipas Noviembre 2014

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO
Instituto Tecnológico de Ciudad Madero

"2014, Año de Octavio Paz"

Cd. Madero, Tamps; a 7 de Octubre de 2014.

OFICIO No.: U5.234/14
AREA: DIVISIÓN DE ESTUDIOS
DE POSGRADO E INVESTIGACIÓN
ASUNTO: AUTORIZACIÓN DE IMPRESIÓN DE TESIS

ING. FANNY GABRIELA MALDONADO NAVA
NO. DE CONTROL G12072014
P R E S E N T E

Me es grato comunicarle que después de la revisión realizada por el Jurado designado para su examen de grado de Maestría en Ciencias de la Computación, el cual está integrado por los siguientes catedráticos:

PRESIDENTE :	DRA. LAURA CRUZ REYES
SECRETARIO :	DRA. GUADALUPE CASTILLA VALDEZ
VOCAL :	DRA. CLAUDIA GUADALUPE GÓMEZ SANTILLÁN
SUPLENTE	DR. JUAN JAVIER GONZÁLEZ BARBOSA
DIRECTOR DE TESIS :	DRA. CLAUDIA GUADALUPE GÓMEZ SANTILLÁN
CO-DIRECTOR DE TESIS:	DRA. LAURA CRUZ REYES

Se acordó autorizar la impresión de su tesis titulada:

"SOLUCIÓN DEL PROBLEMA ANTIBANDWIDTH CÍCLICO USANDO MÉTODOS EXACTOS"

Es muy satisfactorio para la División de Estudios de Posgrado e Investigación compartir con Usted el logro de esta meta.

Espero que continúe con éxito su desarrollo profesional y dedique su experiencia e inteligencia en beneficio de México.

A T E N T A M E N T E
"POR MI PATRIA Y POR MI BIEN"®

M. P. MARÍA YOLANDA CHÁVEZ CINCO
JEFA DE LA DIVISIÓN

S.E.P.
DIVISIÓN DE ESTUDIOS
DE POSGRADO E
INVESTIGACION
I T C M

c.c.p.- Archivo
Minuta
MYCHC 'NLCO ' jar

Ave. 1° de Mayo y Sor Juana I. de la Cruz, Col. Los Mangos, CP. 89440 Cd. Madero, Tam.
Tel. (833) 357 48 20, Fax, Ext. 1002, e-mail: itcm@itcm.edu.mx
www.itcm.edu.mx

Agradecimientos

Quiero agradecer principalmente al comité tutorial, en especial a la Doctora Claudia Guadalupe Gómez Santillán por sus consejos, su apoyo, comprensión y paciencia durante el tiempo que me asesoró, así mismo a la Doctora Laura Cruz Reyes, a la Doctora Guadalupe Castilla Valdez y al Doctor Juan Javier González Barbosa por sus comentarios y sugerencias en la realización de esta tesis. También quiero dar un agradecimiento al Doctor Jesús David Terán Villanueva quien me ha brindado su apoyo y su amistad.

Un agradecimiento muy especial a mi familia, a mis padres por apoyarme en mis decisiones, a mis hermanas que siempre ha estado ahí cuando las he necesitado y a mis sobrinos que hacen mis días siempre muy especiales.

También quiero agradecer a mis compañeros de maestría Javier Alberto Rangel González, Yazmin Gómez Rojas, Rafael Ortega Cortes y Oscar Mellado Camacho, con los que conviví estos dos años y con quienes pasé momentos muy agradables y memorables.

Declaración de Originalidad

Declaro y prometo que este documento de tesis es producto de mi trabajo original y que no infringe los derechos a terceros, tales como derechos de publicación, derechos de autor, patentes y similares.

Además, declaro que en las citas textuales que he incluido (las citas aparecen entre comillas) y en los resúmenes que he realizado de publicaciones ajenas, indico explícitamente los datos de los autores y publicaciones.

Además, en caso de infracción de los derechos de terceros derivados de este documento de tesis acepto la responsabilidad de la información y relevo de ésta área a mi director y codirectores de tesis, así como al Instituto Tecnológico de Ciudad Madero y sus autoridades.

Fanny Gabriela Maldonado Nava

Contenido

Índice de Figuras

Índice de Tablas

Resumen

En este trabajo de investigación se aborda el problema Antibandwidth Cíclico (CAB) por medio de métodos exactos. El problema CAB pertenece a la clase NP-completos [Leung, 1984]. Este problema consiste en encontrar un etiquetado para los vértices de un grafo, de modo tal que se maximice la mínima diferencia absoluta entre las etiquetas de vértices adyacentes cuando estos son dispuestos sobre un grafo tipo ciclo C_n [Bansal y Srivastava, 2011].

De acuerdo a la revisión de la literatura especializada, solo se encontraron dos trabajos que resuelven el problema CAB, uno mediante un algoritmo memético y otro mediante un algoritmo híbrido. También se han presentado algunos resultados teóricos para grafos con topologías de tipo mallas, caminos y ciclos, entre otros.

La aportación de este trabajo consiste en cuatro implementaciones del método *branch and bound* para la solución del problema CAB y la implementación de un modelo de programación lineal entera mediante una herramienta de optimización, resultando éste de la modificación del modelo propuesto por [Duarte, 2010] para el problema AB y ajustándolo al problema CAB.

Capítulo 1. Introducción

El objetivo de este trabajo es el estudio y solución del problema del Antibandwidth Cíclico (CAB), el cual forma parte de la familia de etiquetado de grafos y es una variante del problema Antibandwidth (AB) [Calamoneli et al, 2006]; los cuales pertenecen a la clase NP-completos [Leung, 1984]. Estos problemas consisten en encontrar un etiquetado para los vértices de un grafo, de modo tal que se maximice la mínima diferencia absoluta entre las etiquetas de vértices adyacentes cuando estos son dispuestos sobre un grafo tipo camino P_n (para el problema AB) o tipo ciclo C_n (para el CAB) [Leung, 1984][Bansal y Srivastava, 2011].

El problema de interés tiene aplicaciones reales muy importantes, entre ellas podemos encontrar los problemas relacionados con asignación de frecuencias de comunicación [Hale, 1980], los cuales se pueden modelar mediante un grafo donde los nodos representan el conjunto de n emisoras y las aristas conectan las emisoras que pueden interferir entre ellas y el etiquetado de los nodos indica las frecuencias asignadas a cada emisora. La distancia entre frecuencias corresponde a la mínima diferencia en valor absoluto entre etiquetas de nodos adyacentes. Por lo tanto para resolver el problema de asignación de frecuencias se debe encontrar un etiquetado que maximice la mínima diferencia absoluta entre etiquetas de nodos adyacentes.

En la literatura se han presentado algunos resultados teóricos para grafos con topologías de tipo mallas, caminos y ciclos, entre otros; pero muy poco algoritmos metaheurísticos que los solucionan [Lozano, 2012] [Raspaud, 2009] [Sykora, 2005].

En este trabajo se desarrolló un algoritmo exacto basado en el método *branch and bound* para resolver el problema del Antibandwidth Cíclico. También se implementó un modelo de programación lineal entera para formular el problema de Antibandwidth Cíclico el cual está basado en un modelo propuesto por Duarte et al. [Duarte, 2010] para el problema de Antibandwidth.

1.1 Objetivos del Proyecto

1.1.1 Objetivo general

Diseñar un método de solución exacta para el problema Antibandwidth Cíclico que incluya un modelo de programación lineal entera y un método de tipo ramificación y acotamiento (*branch and bound*).

1.1.2 Objetivos particulares

1. Diseñar una tabla con las características de las diferentes propuestas de solución encontradas en la literatura para el problema CAB.
2. Implementar la solución de un modelo de programación lineal entera utilizando una herramienta científica de optimización.
3. Diseñar e implementar un método de solución exacto basado en ramificación y acotamiento.
4. Identificar y documentar los bancos de prueba que ya existen en la literatura.

1.2 Justificación

El problema CAB es importante debido a que existen problemas reales que pueden ser modelados mediante el uso de grafos y cuya solución implica encontrar un etiquetado para los vértices.

Debido a que la versión de decisión del problema CAB es NP-completo [Leung, 1984], el tamaño del espacio de búsqueda crece de forma factorial en función del tamaño del problema. Por ello es necesario desarrollar algoritmos que sean eficientes y que reduzcan considerablemente los altos tiempos de cómputo registrados en el estado del arte [Lozano, 2012].

De acuerdo a la revisión realizada en el estado del arte, sólo se encontraron reportes de óptimos teóricos de algunos grafos y un reducido número de algoritmos aproximados reportados para resolver el problema CAB. Por lo tanto, existe aún la necesidad de desarrollar algoritmos que permitan resolver de manera competitiva dicho problema con mayor variedad de instancias de gran escala.

Debido a las aplicaciones que se pueden resolver con este problema y a la poca información que se identificó en el estado del arte, se concluye que el tema que se aborda es relevante, complejo y poco estudiado.

1.3 Alcances y Limitaciones

1.3.1 Alcances

1. Este proyecto de investigación solo se enfocará en desarrollar una estrategia de solución exacta basada en *branch and bound*.
2. Implementar la solución de un modelo de programación lineal entera utilizando la herramienta de optimización CPLEX.

1.3.2 Limitaciones

1. El algoritmo solo resolverá instancias con grafos que tiene entre 20 y 100 vértices.
2. El algoritmo solo resolverá instancias de las cuales se conoce su óptimo, ya sea por medio de una formula o por la solución exacta del modelo de programación lineal.

Capítulo 2. Marco Teórico

En esta sección se presentan de forma resumida algunos de los conceptos básicos que conciernen a los problemas de optimización, la teoría de grafos y una introducción a la descripción de dos problemas de etiquetado de grafos estrechamente relacionados: el Antibandwidth (AB) y el Antibandwidth Cíclico (CAB).

2.1 Problemas de Optimización

Se puede decir que un problema de optimización es simplemente un problema en el que hay varias (en general muchas) posibles soluciones y alguna forma clara de comparación entre ellas, de manera que éste existe si y sólo si se dispone un conjunto de soluciones candidatas diferentes que pueden ser comparadas [Duarte, 2007].

Desde un punto de vista matemático, un problema de optimización P se puede formular como una 3-tupla $P = (f, SS, F)$, definida como sigue:

$$P = \begin{cases} opt: f(x), & Función\ Objetivo \\ s.a., & \\ x \in F \subset SS & Restricciones \end{cases}$$

Donde f es la función a optimizar (maximizar o minimizar), F es el conjunto de soluciones factibles y SS es el espacio de soluciones.

Los problemas de optimización se dividen en dos categorías [Blum y Roli, 2003]: aquéllos en los que la solución está codificada mediante valores reales y aquéllos cuyas soluciones están codificadas por valores enteros. Dentro de la segunda categoría se encuentran un tipo particular de problemas denominados problemas de optimización combinatoria.

Un Problema de Optimización Combinatoria consiste en encontrar entre un conjunto finito de soluciones potenciales (llamado espacio de búsqueda) la o las mejores soluciones que verifiquen un criterio particular [Duarte, 2007].

Para resolver los problemas de optimización combinatoria se pueden usar métodos aproximados o exactos. Los algoritmos exactos permiten explorar sistemáticamente un conjunto de soluciones potenciales por lo que siempre proporcionan la mejor solución (óptima) al problema en cuestión. Los métodos aproximados a su vez se puede dividir en dos categorías: algoritmos a la medida y algoritmos generales. Los algoritmos a la medida son aquellos que fueron diseñados para resolver un problema específico. Los algoritmos generales, en cambio, son fácilmente adaptables a la solución de una amplia variedad de problemas de optimización [Duarte, 2007].

2.2 Complejidad Computacional

La teoría de la complejidad computacional es la rama de la teoría de la computación que estudia, de manera teórica, la complejidad inherente a la resolución de un problema computable. Los recursos comúnmente estudiados son el tiempo (mediante una aproximación al número y tipo de pasos de ejecución de un algoritmo para resolver un problema) y el espacio (mediante una aproximación a la cantidad de memoria utilizada para resolver un problema). La mayor parte de los problemas en teoría de la complejidad tienen que ver con los problemas de decisión.

Se define un problema de decisión $\prod$ como un conjunto $D_{\prod}$ de instancias y un subconjunto $Y_{\prod} \subseteq D_{\prod}$ de si-instancias. Donde una instancia pertenece a $D_{\prod}$ si y solo si puede obtenerse a partir de una instancia genérica mediante la sustitución de objetos particulares de los tipos específicos para todos los componentes genéricos, así mismo una instancia pertenece a $Y_{\prod}$ si y solo si la respuesta a la cuestión del problema es sí para esa instancia [Garey, 1979].

Los problemas de decisión se clasifican en conjuntos de complejidad comparable llamados clases de complejidad [Duarte, 2007] [Garey, 1979].

- La clase de complejidad P es el conjunto de los problemas de decisión que pueden ser resueltos en una máquina determinista en tiempo polinomial.
- La clase de complejidad NP es el conjunto de los problemas de decisión que pueden ser resueltos por una máquina no determinista en tiempo polinomial.
- Complejidad NP−Completo, son los problemas que no tienen un algoritmo en tiempo polinomial que los resuelva. Este tipo de problemas es un subconjunto de los problemas NP.
- Complejidad NP−Duro. Son otro tipo de problemas al menos tan difíciles de resolver como los problemas NP−Completos, aunque es posible que sean incluso más difíciles. Para los problemas NP−Duros no existe un algoritmo polinomial que nos permita verificar una solución. Para que un problema sea considerado NP−Duro, debe existir un problema NP−Completo que sea reducible a ese problema. Es decir, debe existir algún algoritmo polinomial que pueda usar una caja negra que resuelva el problema NP−Duro para resolver un problema NP−Completo.

2.3 Métodos Heurísticos

Para muchos problemas no se conoce un algoritmo exacto con complejidad polinomial que encuentre la solución óptima a dicho problema. Debido a esto se deben utilizar métodos aproximados o heurísticas que permitan obtener una solución de calidad en un tiempo razonable a estos problemas [Duarte, 2007]. Una de las definiciones más claras del término heurística es la presentada en [Zanakis, 1981] y que las define como:

"Procedimientos simples, a menudo basados en el sentido común, que se supone obtendrán una buena solución (no necesariamente óptima) a problemas difíciles de un modo sencillo y rápido".

Los métodos heurísticos tienen su principal limitación en su incapacidad para escapar de óptimos locales. Esto se debe fundamentalmente, a que estos algoritmos no utilizan ningún mecanismo que les permita proseguir la búsqueda del óptimo en el caso de encontrar en un óptimo local.

Para solventar este problema, se introducen otros algoritmos de búsqueda inteligentes (denominados metaheurísticas) que evitan, en la medida de lo posible, este problema. Este tipo de algoritmos son procedimientos de alto nivel que guían a métodos heurísticos conocidos, evitando que éstos queden atrapados en óptimos locales.

2.4 Métodos Metaheurísticos

El término Metaheurística o Meta-heurística fue acuñado por F. Glover en el año 1986 [Duarte, 2007]. Glover pretendía definir un procedimiento maestro de alto nivel que guiara y modificara otras heurísticas para explorar soluciones más allá de la simple optimalidad local. Una de las definiciones más descriptivas del término metaheurística es la presentada en [Kelly and Osman, 1996], que se puede enunciar del siguiente modo:

"Las metaheurísticas son una clase de métodos aproximados que están diseñados para resolver problemas difíciles de optimización combinatoria en los que los heurísticos clásicos no son efectivos. Las metaheurísticas proporcionan un marco general para crear nuevos algoritmos híbridos combinando diferentes conceptos derivados de la inteligencia artificial, la evolución biológica y los mecanismos estadísticos".

Podemos encontrar dos tipos de metaheurísticas [Duarte, 2007] [Laguna and Delgado, 2007]:

- Metaheurísticas trayectoriales. Se utiliza el término trayectoria porque el proceso de búsqueda que desarrollan estos métodos se caracteriza por una trayectoria en el espacio de soluciones. Es decir, que partiendo de una solución inicial, son capaces de generar un camino o trayectoria en el espacio de búsqueda a través de operaciones de movimiento.

- Metaheurísticas poblacionales: Las metaheurísticas basadas en poblaciones o metaheurísticas poblacionales son aquéllas que emplean un conjunto de soluciones (población) en cada iteración del algoritmo, en lugar de utilizar una única solución como las metaheurísticas trayectoriales.

2.5 Métodos Exactos

Los problemas combinatorios presentan como particularidad que siempre existe un algoritmo exacto que permite obtener la solución óptima. Este método consiste en la exploración exhaustiva del conjunto de soluciones (enumeración). Los métodos exactos son aquellos que parten de una formulación como modelos de programación lineal (enteros) o similares, y llegan a una solución factible (entera) gracias a algoritmos de acotamiento del conjunto de soluciones factibles [Laguna and Delgado, 2007].

Algunos métodos exactos son:

- *Branch and bound*. Es un enfoque casi enumerativo para resolver una variedad de problemas combinatorios. Esto es bastante eficiente para problemas de tamaño modesto, y la metodología general forma una parte importante del conjunto de métodos de solución para la clase general de problemas de programación lineal entera. Su idea básica es particionar un problema dado en un número de subproblemas (*branching*). Propone establecer subproblemas que son más fáciles de resolver que el problema original, por su tamaño menor o estructura susceptible [Laguna and Delgado, 2007].

- Enumeración Implícita. Es una técnica usualmente aplicada a problemas de programación entera cero-uno. Es similar al *branch and bound*; sin embargo, las reglas de ramificación, restricción y acotamiento son simplificadas y refinadas porque cada variable entera puede tomar solo valores cero o uno. Los problemas de programación entera cero-uno tienen un número finito de puntos factibles, 2^n puntos enteros factibles, donde n es el número de variables cero-uno [Ignizio and Cavalier, 1994].

- Planos de corte. El objetivo de este método es iterativamente construir el casco convexo (*convex hull*), es la mínima región de soluciones factibles en la cual se puede encontrar el valor óptimo, en la vecindad de la solución óptima entera. La solución óptima de un problema de programación entera puede ser determinado resolviendo un problema de programación lineal única en la que el casco convexo es usado como la región factible. Esto es porque los puntos de los extremos del casco convexo corresponden a las soluciones enteras. Se hace de una manera sistemática introduciendo restricciones adicionales que cortan porciones de la región factible excluyendo algunos puntos enteros factibles [Ignizio and Cavalier, 1994].

2.6 Introducción a la Teoría de Grafos

Se considera que la Teoría de Grafos (TG), comenzó en el siglo XVIII, cuando Leonhard Euler demostró que no era posible atravesar los siete puentes de la ciudad rusa de Königsberg (situada actualmente en Kaliningrado) exactamente una vez y volver al punto de partida [Hibbard and Yazlle, 2009]. El grafo que modela este problema se muestra en la figura 2.1.

En 1936 salió publicado en Alemania el primer texto de la teoría de grafos: Theorie der endlichen und unendlichen grapen, escrito por el húngaro Denes König. Este

libro puede considerarse como el principio de la teoría moderna de grafos [Hibbard and Yazlle, 2009].

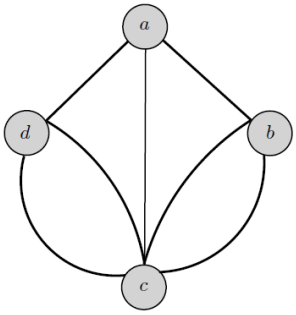


Figura 2. 1 Grafo que modela el problema de los puentes de Königsberg.

La TG es una importante área de las matemáticas, que también se aplica en otras disciplinas como la biología (obtener una clasificación jerárquica del conjunto de todas las proteínas conocidas), la química (modelado de las estructuras, donde los átomos son representados por nodos y los enlaces covalentes por los aristas), ciencias sociales (modelado de relaciones, donde se sustituye a los nodos por los actores sociales y verifica la posición, centralidad e importancia de cada actor dentro del grafo) o aplicaciones industriales (diseño de circuitos VLSI). Además, es considerada como uno de los instrumentos más eficaces para resolver problemas discretos que plantea la Investigación de Operaciones [Hibbard and Yazlle, 2009].

En general, un grafo se puede utilizar para modelar muchas situaciones prácticas donde los objetos involucrados tienen interacción. Por ejemplo, los componentes de un circuito, las vías del ferrocarril, árboles genealógicos, etc.

A continuación se presentan una serie de definiciones que conciernen a la teoría de grafos y que se utilizarán para introducir formalmente el problema de estudio.

Grafo no dirigido. Un grafo no dirigido $G = (V, E)$ es una estructura que consta de un conjunto de elementos llamados vértices $V(G)$ y un conjunto de pares de vértices que son llamados aristas o arcos $E(G)$ [Hibbard and Yazlle, 2009].

Orden de un grafo. El orden de un grafo $G = (V, E)$ es el número de aristas que componen el grafo y se denota como: $ord(G) = |V| = n$ [Hibbard and Yazlle, 2009].

Nodos adyacentes. Dos vértices *u* y *v* de un grafo *G* se llaman adyacentes si $\{u, v\}$ es un arista de *G*. Si todos lo pares de vértices de un grafo *G* son adyacentes entonces es un grafo completo [Hibbard and Yazlle, 2009].

Camino. Un camino (P_n) entre dos vértices u y v de un grafo $G = (V, E)$ es una secuencia $\{\{v_1, v_2\}, \{v_2, v_3\}, \dots, \{v_{k-1}, v_k\}\}$ de aristas de E tal que $u = v_1$ y $v = v_k$. El número de aristas del camino se conoce como longitud del camino [Bansal y Srivastava, 2011].

Ciclo. Un grafo $G = (V, E)$ de tipo Ciclo (C_n), es un conjunto de aristas que forman un camino en el cual el primer vértice del camino corresponde también al último [Bansal y Srivastava, 2011].

Malla. Una malla bidimensional está definida por el producto cartesiano de dos trayectorias (caminos) de tamaños $n_1 y\ n_2$: $P_{n1} \times P_{n2}$ [Raspaud, 2009].

Cuboidal. Una malla tridimensional (cuboidal) está definida por el producto cartesiano de tres caminos $P_{n1} \times P_{n2} \times P_{n3}$ con $n = n_1 \times n_2 \times n_3$ [Bansal y Srivastava, 2011].

Toroide. Una malla toroidal bidimensional $C_n \times C_n$ está definida por el producto cartesiano de dos ciclos [Raspaud, 2009].

Hypercubo. Un grafo hypercubo (Q_n) de dimensión d es un grafo d-regular con 2^d vértices y $d \times 2^{d-1}$ aristas [Bansal y Srivastava, 2011].

Grafo Hamiltoniano. Un grafo Hamiltoniano d-dimensional $\prod_{k=1}^{d} K_{nk}$ esta definido como el producto cartesiano de d grafos completos K_{nk} , para $k = 1,2, \dots, d$. Los vértices de $\prod_{k=1}^{d} K_{nk}$ son *d*-tuplas $(i_1, i_2, \dots, i_d)$, donde $i_k \in \{0,1,2, \dots, n_k - 1\}$. Dos vértices $(i_1, i_2, \dots, i_d)$ y $(j_1, j_2, \dots, j_d)$ son adyacentes si y solo si dos tuplas difieren en precisamente una coordenada. En caso de que $n_k = n$, para toda *k*, el grafo se denota como K_n^d. Y se define el valor de N_k de la siguiente forma. Sea $N_0 = 1$ y para $k = 1,2, \dots, d$, se denota $N_k = n_1 n_2 \dots n_k$ [Dobrev, 2009].

2.7 Problemas de Etiquetado de Grafos

Etiquetado de grafos son una clase particular de problemas de optimización combinatoria, cuya meta consiste en encontrar una distribución lineal para un grafo dado, de tal forma que un cierto objetivo sea optimizado [Petit, 2011]. Un etiquetado de grafos es la asignación de etiquetas, por lo general representadas por números enteros, a las aristas o vértices, o ambos, de un grafo determinado.

Una **distribución lineal** de un grafo no dirigido $G = (V, E)$ de orden n es una función biyectiva de la forma $\varphi: V \rightarrow [n] = \{1,2, \dots, n\}$, que asocia a cada vértice del grafo asocia un número entero distinto entre 1 y n. También se le puede llamar *ordenamiento lineal*, *arreglo lineal* o *etiquetado* de los vértices de un grafo [Petit, 2011] [Duarte, 2010]. Para designar la etiqueta de un vértice *u* utilizaremos la notación $\varphi(u)$.

En la tabla 2.1 se muestran algunos problemas que forman parte de este tipo de problemas con su respectivo nombre.

Tabla 2. 1 Problemas de etiquetado de grafos.

Problema	**Nombre**
Bandwidth	Bandwidth
Min Linear Arrangement	MinLA
Cutwidth	Cutwidht
Modified Cut	ModCut
Vertex Separation	VertSep
Sum Cut Profile	SumCut Profile
Edge Bisection	EdgeBis
Vertex Bisection	VertBis

Al igual que el problema Bandwidth; donde se busca encontrar un etiquetado para los vértices de un grafo, de tal forma que se minimice la máxima diferencia entre los vértices adyacentes [Chinn et al., 1982], encontramos el problema **Antibandwidth** (AB), el cual consiste en encontrar un etiquetado para los vértices de un grafo, de modo tal que se maximice la mínima diferencia absoluta entre las etiquetas de vértices adyacentes cuando estos son dispuestos sobre un grafo tipo camino P_n [Calamoneli et al, 2006].

Del problema AB se extiende el problema **Antibandwidth Cíclico** (CAB, Ciclic Antibandwidth) [Bansal y Srivastava, 2011], que es un problema de disposición sobre un grafo tipo ciclo C_n, donde los vértices de un grafo $G = (V, E)$ de orden n son mapeados biyectivamente en C_n de forma tal que la mínima distancia (medida en el ciclo) entre etiquetas de vértices adyacentes sea maximizada.

2.8 Programación Lineal

Programación lineal es una técnica que consiste en una serie de métodos y procedimientos que permiten resolver problemas de optimización. La programación lineal utiliza un modelo matemático para describir el problema de interés.

El adjetivo “lineal” significa que todas las funciones matemáticas en el modelo deben ser funciones lineales. La palabra “programación” no se refiere al término programación en el ámbito de computación; sino, como sinónimo de la palabra planificación. Entonces, programación lineal significa la planificación de actividades con el fin de obtener el resultado óptimo. Es decir, un resultado que satisfaga la meta especificada sobre todas las posibles soluciones (factibles).

Muchos problemas de interés pueden ser sistemas muy grandes y complejos. Difícilmente se podría pensar en evaluar estos sistemas, en lugar de esto, un modelo que represente este sistema es usado normalmente para dicha experimentación. Si el modelo es “bueno”, la solución puede representar una buena aproximación a la solución del

sistema. Modelar un problema usando programación lineal entera involucra escribir en un lenguaje de programación lineal.

Los principales elementos de un problema de optimización modelado son: las variables de decisión, restricciones y función objetivo.

1. Variables (también llamadas variables de decisión): los valores de estas son desconocidas en un principio. Usualmente las variables representan cosas que pueden ser ajustadas o controladas. El objetivo es encontrar los valores de las variables que dé como resultado el mejor valor de la función objetivo.
2. Restricciones: estas, son expresiones matemáticas que combinan las variables para expresar límites de las posibles soluciones.
3. Función objetivo: esta es una expresión matemática que combina las variables para expresar el objetivo. La función objetivo debe ser maximizada o minimizada.

En el siguiente ejemplo se muestran estos elementos:

Variables de decisión: $x_1 \ y \ x_2$

Función objetivo: $Maximizar\ Z = 3x_1 + 5x_2$

Sujeto a las siguientes restricciones:

$$x_1 \leq 4$$
$$2x_2 \leq 12$$
$$3x_1 + 2x_2 \leq 18$$

Capítulo 3. Descripción del Problema

En esta sección se presenta la descripción y formulación del problema Antibandwidth Cíclico, que es el problema de estudio de este trabajo de investigación, pero antes se presenta una descripción del problema Antibandwidth Lineal, ya que de este se extiende el problema Antibandwidth Cíclico.

3.1 Problema Antibandwidth

El problema AB se define formalmente de la siguiente manera:

Sea $G = (V, E)$ un grafo no dirigido de orden n, con un conjunto de vértices V y un conjunto de aristas E, y $\varphi: V \to \{1,2, \dots, n\}$ un etiquetado para los vértices de G. El Antibandwidth del grafo G para el etiquetado φ está definido con la ecuación (3.1) [Duarte, 2010]:

$$ab(G, \varphi) = \min_{\{u,v\} \in E} \{|\varphi(u) - \varphi(v)|\} \qquad (3.1)$$

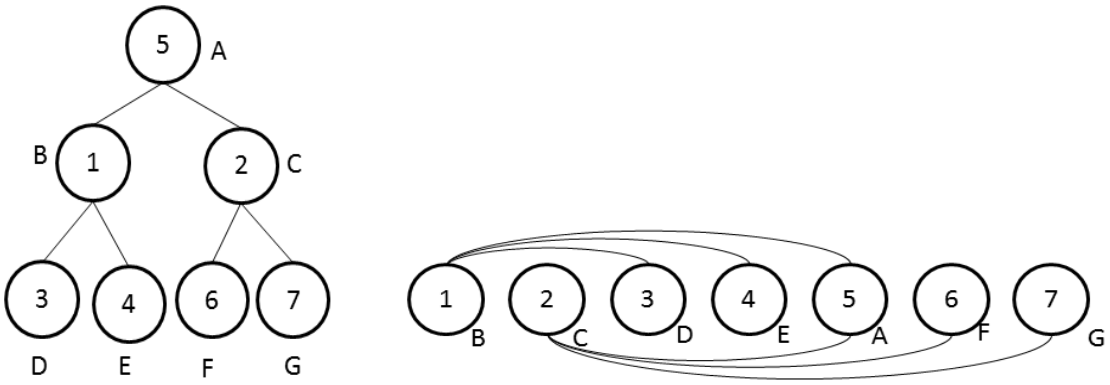
Entonces el problema del Antibandwidth consiste en encontrar un etiquetado óptimo φ^* para el cual el valor del Antibandwidth del grafo G, sea máximo:

$$ab(G, \varphi^*) = \max_{\varphi \in L} \{ab(G, \varphi)\} \qquad (3.2)$$

Donde L es el conjunto de todos los etiquetados posibles.

En la figura 3.1 se muestra un ejemplo de un etiquetado para un grafo de orden $n = 7$ con un etiquetado φ, las etiquetas están anotadas en el interior de cada vértice del

grafo original. En la tabla 3.1 se muestran los cálculos correspondientes a las distancias entre cada par de vértices adyacentes para dicho etiquetado.



(a) Grafo original (b) Grafo dispuesto en un grafo tipo camino

Figura 3. 1 Ejemplo de un grafo dispuesto en un grafo tipo camino $\mathbf{P_7}$.

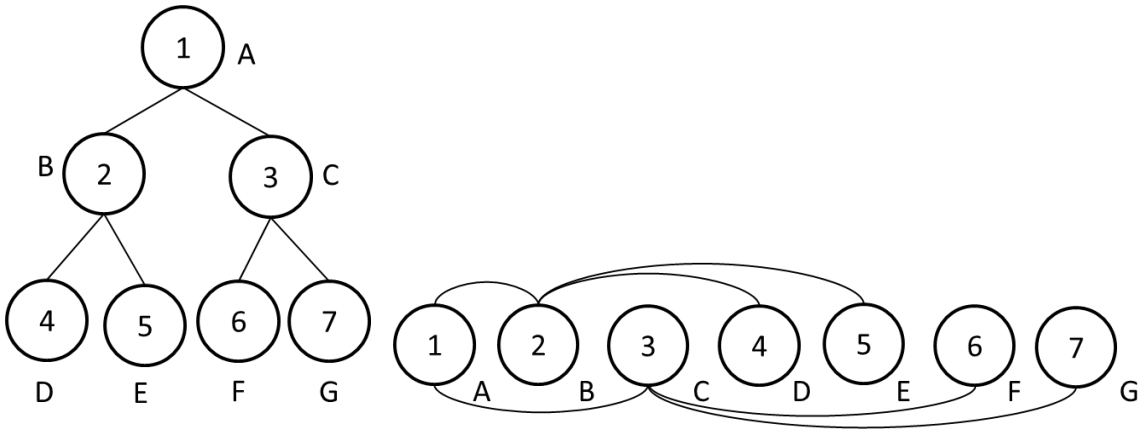
Por ejemplo, la distancia entre los vértices adyacentes A y B se calcula como: $|\varphi(A) - \varphi(B)| = |5 - 1| = 4$. De igual forma se deben calcular estas diferencias para cada par de vértices adyacentes, como se muestra en la tabla 3.1.

Tabla 3. 1Cálculos de las distancias para el grafo de la figura 3.1.

Aristas	**Cálculos**	**Resultados**
AB	|5-1|	4
AC	|5-2|	3
BD	|1-3|	**2**
BE	|1-4|	3
CF	|2-6|	4
CG	|2-7|	5

Entonces, para el etiquetado φ del grafo de la figura 3.1, el valor del Antibandwidth es: $ab(G, \varphi) = 2$; que es la mínima de todas las diferencias obtenidas.

En contraste tenemos la figura 3.2 en la que se muestra el mismo grafo de la figura 3.1, pero con un etiquetado diferente φ', es decir, en este etiquetado la distancia entre los vértices adyacentes A y B es $|\varphi(A) - \varphi(B)| = |1 - 2| = 1$, y en la tabla 3.2 se tienen los cálculos para el resto de los vértices adyacentes.



(a) Grafo original (b) Grafo dispuesto en un grafo tipo camino

Figura 3. 2 Ejemplo de un grafo dispuesto en un grafo tipo camino $\mathbf{P_7}$.

Tabla 3. 2 Cálculos de las distancias para el grafo de la figura 3.2.

Aristas	**Cálculos**	**Resultados**
AB	|1-2|	1
AC	|1-3|	2
BD	|2-4|	2
BE	|2-5|	3
CF	|3-6|	3
CG	|3-7|	4

Como se puede observar en la tabla 3.2, la mínima de las diferencias obtenidas es 1, entonces, el valor del Antibandwidth para el etiquetado φ' del grafo de la figura 3.2 es $ab(G, \varphi') = 1$.

Debido a que queremos maximizar, el etiquetado φ para el grafo de la figura 3.1 cuyo Antibandwidth es $ab(G, \varphi') = 2$ es una mejor solución que el etiquetado φ' del grafo de la figura 3.2 donde el Antibandwidth es $ab(G, \varphi) = 1$.

3.2 Modelo de Programación Lineal Entera para el Problema Antibandwidth

Duarte [Duarte, 2010] resuelve el problema Antibandwidth por medio de GRASP con Path Relinking, y a su vez propone la formulación del modelo de programación lineal entera para dicho problema.

El modelo propuesto se plantea de la siguiente forma:

Sea x_{ik} una variable binaria que toma el valor de 1, si y solo si $\varphi(i) = k$, es decir, si al vértice i se le asigna la etiqueta k. Se define la variable entera l_i como la etiqueta del vértice i, es decir, $l_i = \varphi(i) \in \{1, 2, \dots, n\}$ y sea $b = ab(G, \varphi) = \min_{\{u,v\} \in E}\{|\varphi(u) - \varphi(v)|\}$.

La formulación del modelo para el problema Antibandwidth es el siguiente:

$$\max b \qquad (3.5)$$

Sujeto a:

(3.6) $$\sum_{i=1}^{n} x_{ik} = 1\,, \forall k = 1, \dots, n$$

Asegura que se asigne sólo una etiqueta a cada vértice.

(3.7) $$\sum_{k=1}^{n} x_{ik} = 1\,, \forall i = 1, \dots, n$$

Asegura que se asigne sólo un vértice a cada etiqueta.

(3.8) $$\sum_{k=1}^{n} k \cdot x_{ik} = l_i\,, \forall i = 1, \dots, n$$

Calculan el valor de las etiquetas de las variables x_{ik}.

(3.9) $$b - \left(l_i - l_j\right) \leq 2y_{ij}(n-1), \forall (i,j) \in E$$

Implica que:

$b = \min_{(i,j)\in E}\{\left|l_i - l_j\right|\}$.

(3.10) $$b + \left(l_i - l_j\right) \leq 2z_{ij}(n-1), \forall (i,j) \in E$$

Implica que:

$b = \min_{(i,j)\in E}\{\left|l_i - l_j\right|\}$.

(3.11) $$y_{ij} + z_{ij} = 1, \forall (i,j) \in E$$

Solo una de las alternativas; ya sea que $l_i \geq l_j$ o $l_i < l_j$.

(3.12) $$b \geq 1$$

Garantiza que el valor de b sea positivo.

(3.13) $$x_{ik} \in \{0,1\}, \forall i,k = 1, \dots, n$$

Dominio de las variables x_{ik}.

(3.14) $$y_{ij} \in \{0,1\}, \forall (i,j) \in E$$

Dominio de las variables y_{ij}.

(3.15) $$z_{ij} \in \{0,1\}, \forall (i,j) \in E$$

Dominio de las variables z_{ij}.

(3.16) $$1 \leq l_i \leq n, \forall i = 1, \dots, n$$

Dominio de las variables l_i.

3.3 Problema Antibandwidth Cíclico

Como se mencionó anteriormente, el problema de interés de este trabajo es el problema Antibandwidth Cíclico, que es un problema de disposición sobre un grafo tipo ciclo C_n (grafo anfitrión), donde los vértices de otro grafo $G = (V, E)$ de orden n (grafo invitado) son mapeados biyectivamente en C_n de forma tal que la mínima distancia (medida en el ciclo) entre etiquetas de vértices adyacentes sea maximizada [Bansal y Srivastava, 2011].

El CAB se describe formalmente como:

Sea $G = (V, E)$ un grafo no dirigido de orden n, y $\varphi: V \to \{1,2, \dots, n\}$ un etiquetado para G. El Antibandwidth Cíclico del grafo G para el etiquetado φ esa definido con la ecuación (3.17):

$$cab(G, \varphi) = \min_{\{u,v\} \in E} \{|\varphi(u) - \varphi(v)|, n - |\varphi(u) - \varphi(v)|\} \qquad (3.17)$$

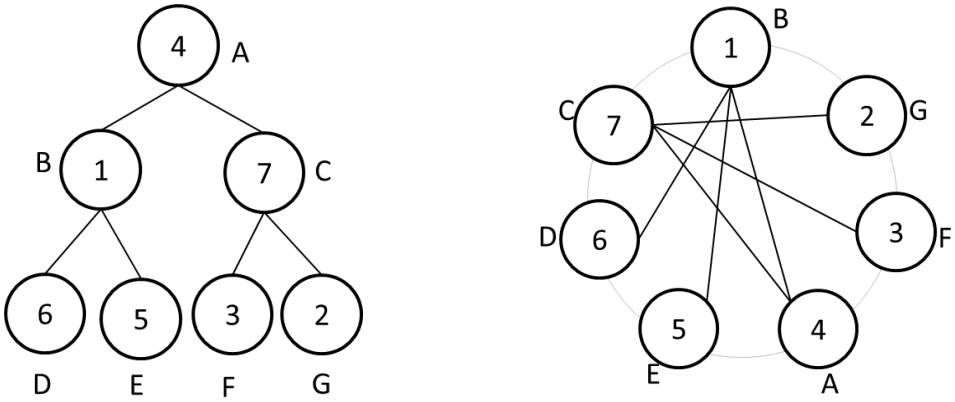
De acuerdo al concepto anterior, podemos definir formalmente el problema CAB de la siguiente manera [Leung, 1984]:

El problema CAB consiste en encontrar un etiquetado óptimo φ^* para el cual el valor del Antibandwidth Cíclico del grafo G, sea máximo [Sykora, 2005]:

$$cab(G, \varphi^*) = \max_{\varphi \in L} \{\text{cab}(\text{G}, \varphi)\} \qquad (3.18)$$

Donde L es el conjunto de todos los etiquetados posibles.

En la figura 3.3, se muestra un ejemplo de un grafo con $n = 7$ vértices designados con letras, $m = 6$ aristas y con un etiquetado φ. En la tabla 3.3 se observan los cálculos correspondientes a las distancias entre cada par de vértices adyacentes y las distancias dado un etiquetado circular.



(a) Grafo original (b) Grafo dispuesto en un grafo tipo ciclo

Figura 3. 3 Ejemplo de un grafo dispuesto en un grafo tipo ciclo $\mathbf{C_7}$.

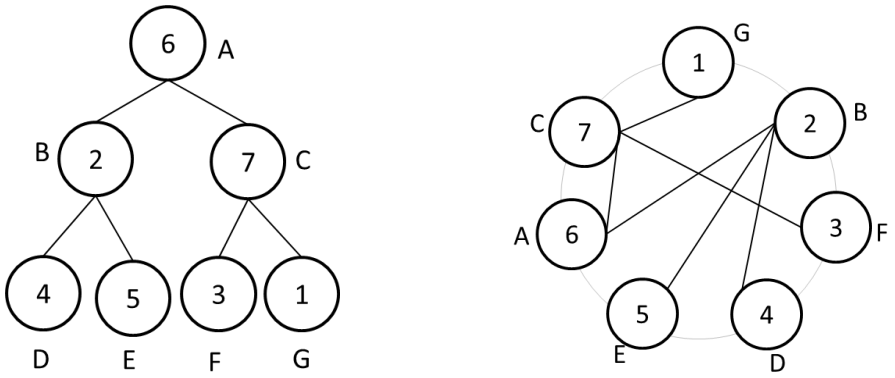
Por ejemplo, la distancia entre los vértices adyacentes A y B es $|\varphi(A) - \varphi(B)| = |4 - 1| = 3$. Y para la disposición circular se calcula como $n - |\varphi(A) - \varphi(B)| = 7 - |4 - 1| = 4$. Se deben calcular estas mismas diferencias para las aristas restantes (como se muestra en la tabla 3.3).

Tabla 3. 3 Cálculos de las distancias para el grafo de la figura 3.3.

Aristas	**Cálculos**	**Resultados**
AB	|4-1|, 7-|4-1|	3, 4
AC	|4-7|, 7-|4-7|	3, 4
BD	|1-6|, 7-|1-6|	5, 2
BE	|1-5|, 7-|1-5|	4, 3
CF	|7-3|, 7-|7-3|	4, 3
CG	|7-2|, 7-|7-2|	5, 2

Quedando como resultado para el etiquetado φ la mínima de todas las diferencias obtenidas, entonces para este grafo el valor del $cab(G, \varphi) = 2$.

En la figura 3.4 se muestra un etiquetado diferente φ' y en la tabla 3.4 se tienen los cálculos para encontrar el valor del CAB. Donde podemos verificar la mínima de las diferencias obtenidas es 1. Entonces, el CAB para este etiquetado es $cab(G, \varphi') = 1$.



(a) Grafo original (b) Grafo dispuesto en un grafo tipo ciclo

Figura 3. 4 Ejemplo de un grafo dispuesto en un grafo tipo ciclo $\mathbf{C_7}$.

Tabla 3. 4 Cálculos de las distancias para el grafo de la figura 3.4.

Aristas	**Cálculos**	**Resultados**
AB	|6-2|, 7-|6-2|	4, 3
AC	|6-7|, 7-|6-7|	**1**, 6
BD	|2-4|, 7-|2-4|	2, 5
BE	|2-5|, 7-|2-5|	3, 4
CF	|7-3|, 7-|7-3|	4, 3
CG	|7-1|, 7-|7-1|	6, **1**

Debido a que el problema CAB es de maximización, el etiquetado φ' para el grafo de la figura 3.2 cuyo $cab(G, \varphi') = 2$ es una mejor solución que el etiquetado φ de la figura 3.1 donde el $cab(G, \varphi) = 1$.

Capítulo 4. Estado del Arte

En esta sección se describen algunos métodos reportados en la literatura para resolver el problema Antibandwidth Cíclico.

El problema Antibandwidth fue originalmente introducido por Leung *et al* [Leung, 1984] en relación con problemas de calendarización de tareas en varios procesadores. El término Antibandwidth se introdujo por Raspaud *et al* [Raspaud, 2009]. Del problema Antibandwidth se extiende de manera natural el problema Antibandwidth Cíclico el cual fue presentado originalmente por Leung *et al* [Leung, 1984], quienes demostraron que la versión de decisión del CAB pertenece a la clase de problemas NP-completos.

Los métodos que se han propuesto para resolver el problema CAB se pueden clasificar de dos maneras: en algoritmos metaheurísticos y acercamientos matemáticos.

4.1 Acercamientos Matemáticos al Problema

Muchos de los trabajos que se encuentran en la literatura sobre el problema CAB se enfocan principalmente en el estudio teórico para encontrar soluciones óptimas para algunos casos de grafos.

Por ejemplo Raspaud *et al* [Raspaud, 2009] muestran resultados óptimos para tres tipos de grafos: mallas bidimensionales, toroidales e hypercubos. Dobrev *et al* [Dobrev, 2009] resuelven instancias del problema CAB y AB para grafos de tipo hamiltoniano. Para grafos tipo cuboidal y doble estrella las conjeturas del CAB fueron obtenidas por medio del análisis experimental de un algoritmo memético propuesto por Bansal y Srivastava [Bansal y Srivastava, 2011], el cual se describe más adelante. En la

tabla 4.1 se muestran las conjeturas sobre límites teóricos del problema CAB para algunos tipos de grafos.

Tabla 4. 1 Conjeturas del CAB para algunos tipos de grafos.

Instancias	**Conjeturas del CAB para algunos tipos de grafos**
Caminos	$cab(P_n) = \left\lceil \frac{n}{2} \right\rceil - 1$
Ciclos	$cab(C_n) = \left\lceil \frac{n}{2} \right\rceil - 1$
Mallas	$cab(P_{n1} \times P_{n2}) = \begin{cases} \left\lfloor \frac{n_2(n_1-1)}{2} \right\rfloor \leq cab(G) \leq \left\lceil \frac{n_2(n_1-1)}{2} \right\rceil si\ n_1\ es\ par \\ \quad y\ n_1\ impar (n_1 \geq n_2) \\ \frac{n_2(n_1-1)}{2} \quad de\ otra\ manera \end{cases}$
Toroides	$cab(C_n \times C_n) = \begin{cases} \frac{n(n-2)}{2} & si\ n\ es\ par \\ \left(\frac{(n-2)(n+1)}{2}\right) & si\ n\ es\ impar \end{cases}$
Hypercubos	$cab(Q_n) = 2^{n-1} - \frac{2^n}{\sqrt{2\pi n}}(1 + o(1))$
Cuboidal	$cab(P_{n1} \times P_{n2} \times P_{n3}) = \begin{cases} 2(n_3 - 1)\ si\ n_1 = 2, n_2 = 2, n_3 = 3 - 500 \\ \frac{9n_3 - 8}{2} si\ n_1 = 3, n_2 = 3, n_3 = 3 - 400\ y\ n_3\ es\ par \\ \frac{9n_3 - 1}{2} si\ n_1 = 3, n_2 = 3, n_3 = 3 - 400\ y\ n_3\ es\ impar \\ 8(n_3 - 1)\ si\ n_1 = 4, n_2 = 4, n_3 = 5 - 200 \\ \frac{25n_3 - 26}{2}\ si\ n_1 = 5, n_2 = 5, n_3 = 7 - 100\ y\ n_3\ es\ par \\ \frac{25n_3 - 27}{2} si\ n_1 = 5, n_2 = 5, n_3 = 7 - 100\ y\ n_3\ es\ impar \\ 18n_3 - 19\ \ si\ n_1 = 2, n_2 = 2, n_3 = 3 - 500 \end{cases}$
Doble estrella	$cab\big(s(n_1, n_2)\big) = \begin{cases} \left\lfloor \frac{n_1}{2} \right\rfloor & si\ n_1 = n_2 \\ \left\lceil \frac{\min(n_1, n_2)}{2} \right\rceil & de\ otra\ manera \end{cases}$
Hamming	$cab\left(\prod_{k=1}^{d} K_{nk}\right) = \begin{cases} n_1 n_2 \dots n_{d-1} & si\ n_{d-1} \neq n_d, d \geq 2 \\ n_1 n_2 \dots n_{d-1} - 1 & si\ n_{d-1} = n_d\ y\ n_{d-2} \neq n_{d-1}, d \geq 3 \end{cases}$

4.2 Algoritmos Metaheurísticos

De acuerdo a la revisión en la literatura, se encontraron dos trabajos reportados con algoritmos metaheurísticos, el primero es un algoritmo memético hecho por Bansal y Srivastava [Bansal y Srivastava, 2011] y el segundo es un algoritmo híbrido desarrollado por Lozano *et al* [Lozano, 2012].

Bansal y Srivastava [Bansal y Srivastava, 2011] desarrollaron un algoritmo memético para el problema CAB, llamado MACAB en donde emplean la técnica Búsqueda Primero en Anchura (Breadth First Search, BFS). Esta técnica es aplicada sobre el grafo para generar una estructura de niveles con lo cual asegura una mayor diversidad a la población. Posteriormente emplean una nueva heurística para etiquetar los vértices.

En cada generación, para la selección se aplica un operador de torneo a la población de los padres, para seleccionar los mejores para la reproducción y así eliminar a las peores soluciones. Después de la selección, en la fase de reproducción se aplica un intermedio BFS para generar nuevas soluciones a partir de los padres, después para generar un nuevo etiquetado es aplicada la heurística exhaustiva para asignación de etiquetas.

El operador de mutación basado en Hill Climbing es aplicado para asegurar que después de la heurística las mejores soluciones entren a la población. Esta mutación se basa en intercambio de etiquetas de dos vértices para obtener la solución mutada la cual reemplaza la original si es mejor. Después de la mutación la población es actualizada, cada hijo compite con sus respectivos padres y el mejor de ellos se convierte en padre de la siguiente generación. Este proceso se repite hasta que no haya mejora para un número especificado de iteraciones.

Lozano *et al* [Lozano, 2012] propusieron un algoritmo hibrido para resolver el problema CAB, llamado HABC-CAB. En este trabajo, proponen un enfoque basado en la metodología colonia artificial de abejas (ABC, artificial bee colony).

El enfoque comienza con una población de soluciones (fuentes de alimentación) generado por la función que implementa de manera aleatoria una Búsqueda Primero en Anchura (Breadth First Search, BFS), método propuesto por Bansal y Srivastava [Bansal y Srivastava, 2011] para construir soluciones para el problema CAB. Después de la generación de las soluciones, se repiten los siguientes pasos hasta que se cumpla un criterio de paro:

- Fase de abejas empleadas. En esta fase se producen nuevas soluciones para las abejas empleadas utilizando un operador de vecindad. El operador de vecindad implementa cadenas de eyección. Cada abeja empleada es relacionada a una fuente de comida vecina (actualizando su estado cada vez que encuentran una mejor) y al finalizar la información es compartida con las abejas observadoras.
- Fase de las abejas espectadoras. Produce nuevas soluciones para los espectadores por medio de un operador de torneo. Una abeja espectadora selecciona la mejor fuente de comida de entre dos fuentes seleccionadas aleatoriamente por medio de dicho operador. Cada espectadora determina una fuente de comida en la vecindad.
- Fase de abejas exploradoras. Si las abejas asociadas con una fuente de comida no encuentran una mejor fuente de alimento vecina en determinado número de iteraciones, dicha fuente es abandonada y la abeja exploradora comienza a buscar una nueva fuente de alimento mediante la función de construcción BFS y una nueva iteración empieza.

Con el fin de mejorar la capacidad de explotación del HABC-CAB, un método de búsqueda local rápida (FLS) se incluye en el algoritmo. Específicamente, después de generar nuevas soluciones (en la inicialización y la fase de exploración) y en la producción de fuentes de alimentos vecinos (en la fase de empleadas y espectadoras). Esta búsqueda se aplica (con una probabilidad) para mejorar aún más la calidad de las soluciones. Además, en cada iteración, después de que todas las abejas terminan sus búsquedas, el algoritmo ABC memoriza la mejor fuente de alimento encontrado.

Es importante destacar que de acuerdo con la investigación realizada en la literatura especializada, no existen reportados hasta el momento algoritmos exactos para resolver el problema CAB, al igual que no hay un modelo de programación lineal propuesto para este problema.

Se encontró un modelo de programación lineal entera para el problema Antibandwidth Lineal implementado por [Duarte, 2010] y reportado en un trabajo en el cual resuelve este problema mediante GRASP. Este modelo se describe en la sección 3.2, donde se hace una descripción más amplia de ambos problemas.

Capítulo 5. Diseño e Implementación

En este capítulo se describe con detalle el diseño e implementación de los algoritmos basados en *branch and bound* y el modelo de programación lineal entera para resolver el problema del CAB, así como los componentes que integran cada uno de ellos.

5.1 Modelo de Programación Lineal Entera Propuesto para el Problema CAB

En este trabajo se propone un modelo de programación lineal entera para el problema Antibandwidth Cíclico, basado en el modelo presentado por [Duarte, 2010] que se describe en la sección 3.2.

Como se define en la sección 3.3, el problema del antibandwidth cíclico implica la verificación de distancias entre dos etiquetas, una en el sentido de las manecillas del reloj tal como se calcula en el antibandwidth y otra en sentido contrario haciéndolo cíclico. Quedando así para este problema, la siguiente función objetivo:

$$b = \min_{\{u,v\} \in E} \{|\varphi(u) - \varphi(v)|, n - |\varphi(u) - \varphi(v)|\} \qquad (5.1)$$

Cabe destacar que la función objetivo del antibandwidth cíclico al implementarla como modelo de programación lineal entera se separa en una nueva función objetivo (ecuación 5.2) y un conjunto de restricciones para incorporar el cálculo de los dos valores absolutos. La linealización del primer valor absoluto es idéntica al modelo de AB (ecuaciones 5.6 y 5.7). Para la linealización de la segunda diferencia de la función

objetivo se utilizó la misma forma de linealización del modelo propuesto en [Duarte, 2010], dando como resultado las restricciones de las ecuaciones 5.8 y 5.9.

La formulación del modelo de programación lineal entera para el CAB es:

(5.2) $\max b$

Sujeto a:

(5.3) $\sum_{i=1}^{n} x_{ik} = 1\,, \forall k = 1, \dots, n$ — Asegura que se asigne sólo una etiqueta a cada vértice.

(5.4) $\sum_{k=1}^{n} x_{ik} = 1\,, \forall i = 1, \dots, n$ — Asegura que se asigne sólo un vértice a cada etiqueta.

(5.5) $\sum_{k=1}^{n} k \cdot x_{ik} = l_i\,, \forall i = 1, \dots, n$ — Calculan el valor de las etiquetas de las variables x_{ik}.

(5.6) $b - (l_i - l_j) \leq 2y_{ij}(n-1), \forall (i,j) \in E$ — Implica que: $b = \min_{(i,j)\in E}\{|l_i - l_j|\}$.

(5.7) $b + (l_i - l_j) \leq 2z_{ij}(n-1), \forall (i,j) \in E$ — Implica que: $b = \min_{(i,j)\in E}\{|l_i - l_j|\}$.

(5.8) $b + (l_i - l_j) \leq 2y_{ij}(n-1) + n, \forall (i,j) \in E$ — Implica que: $b = \min_{(i,j)\in E}\{n - |l_i - l_j|\}$.

(5.9) $b - (l_i - l_j) \leq 2z_{ij}(n-1) + n, \forall (i,j) \in E$ — Implica que: $b = \min_{(i,j)\in E}\{n - |l_i - l_j|\}$.

(5.10) $y_{ij} + z_{ij} = 1, \forall (i,j) \in E$ — Solo una de las alternativas; ya sea que $l_i \geq l_j$ o $l_i < l_j$.

(5.11)	$b \geq 1$	Garantiza que el valor de b sea positivo.
(5.12)	$x_{ik} \in \{0,1\}, \forall i, k = 1, \dots, n$	Dominio de las variables x_{ik}.
(5.13)	$y_{ij} \in \{0,1\}, \forall (i,j) \in E$	Dominio de las variables y_{ij}.
(5.14)	$z_{ij} \in \{0,1\}, \forall (i,j) \in E$	Dominio de las variables z_{ij}.
(5.15)	$1 \leq l_i \leq n, \forall i = 1, \dots, n$	Dominio de las variables l_i.

5.2 Implementación del Método *Branch and Bound*

5.2.1 Algoritmo BBCAB

La primer propuesta de solución para el problema Antibandwidth Cíclico mediante la implementación del método *branch and bound*, es un algoritmo en el cual se recorre el árbol de exploración de manera completa, dicho algoritmo se ejecuta mientras haya etiquetas que asignar, es decir mientras no llegue a la última hoja de la rama, como se muestra en la figura 5.1. Cada que se agrega una etiqueta se hace la evaluación parcial de la permutación y se van comparando las diferencias obtenidas con las anteriores con el fin de conservar el menor valor de cada permutación y estos valores mínimos se comparan entre sí para obtener la mayor de estas diferencias, lo cual es nuestro objetivo.

A continuación se presenta el algoritmo 5.1 del método *branch and bound* implementado como una primera versión del método de solución para el problema Antibandwidth Cíclico.

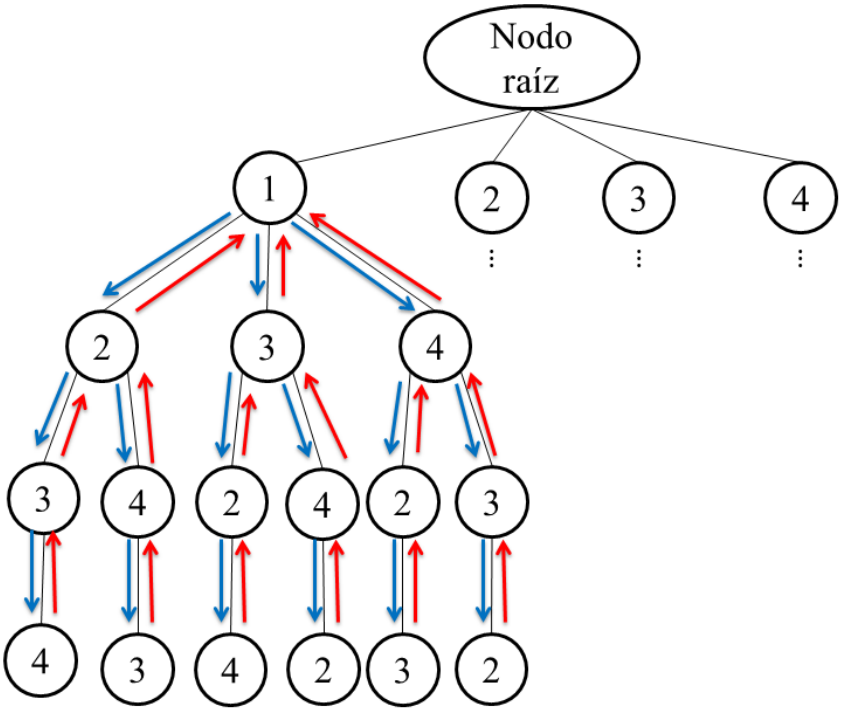


Figura 5. 1 Recorrido del árbol de exploración.

Algoritmo 5.1 Algoritmo BBCAB

Entrada: $S \leftarrow 1 \ldots n$

```
1.  for (j = 0; j < n - i; j++)
2.      P[i] = tomarElemdeCola(S);
3.      costo = evaluacionParcial(P,i);
4.      if (costo[i] > mejorSol)
5.          if (i < n - 1)
6.              BBCAB(i + 1);
7.          else
8.              mejorSol = costo[i];
9.          end if
10.     end if
11.     insertarElemenCola(P[i]);
12.     menorLocal[i] = Huge.Value;
13. end for
```

En el algoritmo 5.2 se presenta el método mediante el cual se hace la evaluación parcial de la solución que se llama cuando se van agregando etiquetas a la solución.

Algoritmo 5.2 Evaluación parcial para el algoritmo BBCAB

Entrada: $ady[][]$

```
1.  for (j = i-1; j >= 0; j--)
2.      if(ady[i][j] == 1)
3.          minTemporal = |P[i]-P[j]|
4.          if(menorLocal[i-1] < minTemporal)
5.              menorLocal[i]=MIN(menorLocal[i-1], menorLocal[i])
6.          else
7.              menorLocal[i]=MIN(minTemporal, menorLocal[i])
8.          end if
9.      else
10.         menorLocal[i] = MIN(menorLocal[i-1], menorLocal[i])
11.     end if
12. end for
```

5.2.2 Algoritmo BBCAB1

Después de un conjunto de pruebas se analizaron los resultados se observó que consumía mucho tiempo inclusive para resolver instancias pequeñas. Considerando que la causa de este problema se debió a que la estrategia implementada era exhaustiva, por lo cual se decidió realizar una nueva versión del algoritmo BBCAB. La estrategia abordada en la primera versión implementada describe que, para poder evaluar una solución, es necesario llegar a un nodo hoja en el árbol de exploración. Es decir se recorre todo el árbol para poder encontrar la solución óptima.

Las mejoras que se incorporaron a la primera versión del B&B son: la generación de una solución inicial y cálculo del CAB parcial para poder aplicar el criterio de poda.

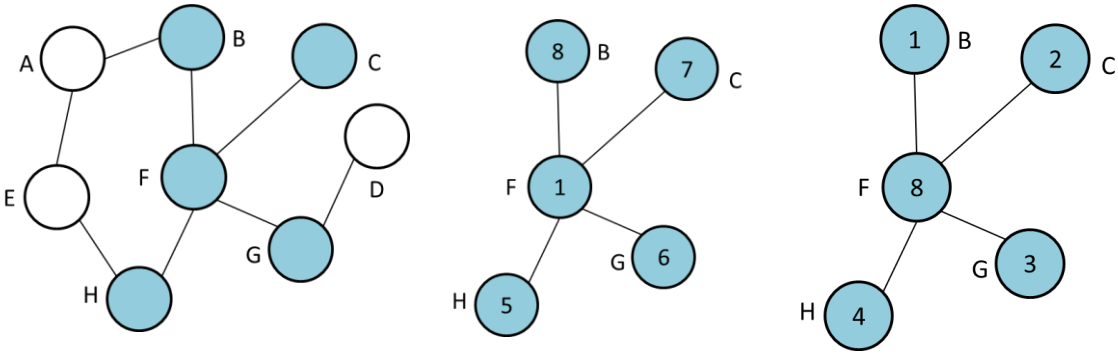
La solución inicial se utilizó para tener un valor inicial de referencia a la hora de decidir si la nueva solución encontrada por el algoritmo es mejor que la encontrada hasta el momento. Para generar esta solución se desarrolló un método heurístico, que consta de dos pasos:

1. Etiquetado del vértice de mayor grado: Se asignan etiquetas tanto al vértice de mayor grado como a los vértices adyacentes a éste, maximizando la diferencia mínima de etiquetas.
2. Etiquetado aleatorio de los vértices restantes: El resto de las etiquetas son asignadas de forma aleatoria a los vértices que no tengan ninguna etiqueta asignada. Este proceso se realiza una determinada cantidad de veces y se elige el etiquetado que genere un CAB mayor de entre todos.

Una vez obtenida una solución inicial se calcula el valor del CAB para dicho etiquetado y se asigna como mejor solución encontrada hasta el momento.

Para etiquetar el vértice de mayor grado y sus adyacentes se probaron dos métodos heurísticos: a) etiquetado de los extremos y b) etiquetado del valor medio.

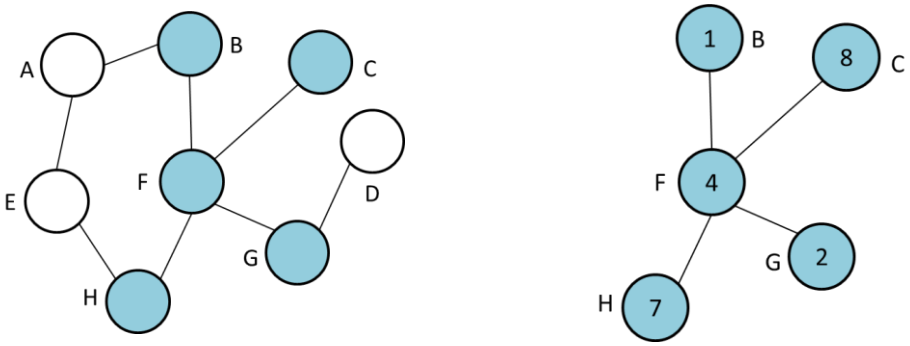
a) Etiquetado de los extremos. Al vértice de mayor grado se le asigna la etiqueta más baja disponible (por ejemplo 1) y los vértices adyacentes a éste, son etiquetados con los valores más altos que haya disponibles, por ejemplo: $n, n-1, n-2, ...$, y así sucesivamente hasta que todos los vértices adyacentes sean etiquetados. Como se puede observar, esta técnica tiene dos variantes posibles: etiquetar el vértice principal con el valor más pequeño posible y los adyacentes con las etiquetas más altas disponibles, o bien, etiquetar el vértice principal con el valor más alto posible y los adyacentes con las etiquetas más pequeñas que haya disponibles. En la figura 5.2 se muestra un ejemplo de este etiquetado, en el cual se etiqueta el vértice F, que es el vértice de mayor grado.



a) Grafo original b) Etiquetado de los extremos inferior c) Etiquetado de los extremos superior

Figura 5. 2 Ejemplo de un vértice etiquetado por los extremos.

b) Etiquetado del valor medio. El vértice de mayor grado es etiquetado con la etiqueta de valor medio en el rango $[1 \ldots n]$, es decir con $n/2$, siendo ésta una división entera. Los vértices adyacentes al de mayor grado son etiquetados con los valores alternos: $1, n, 2, n-1, \ldots$, y sucesivamente hasta que todos los vértices adyacentes sean etiquetados. Una vez completado el proceso, se calcula el valor del CAB. En la figura 5.3 se muestra un ejemplo del etiquetado del valor medio.



a) Grafo original b) Vértice de mayor grado etiquetado

Figura 5. 3 Ejemplo de un etiquetado del valor medio.

Una vez hecha la experimentación con los métodos para generar la solución inicial, se optó por trabajar con la heurística del valor medio (algoritmo 5.3), ya que los

resultados mostraron que el valor del CAB de las soluciones hechas mediante este método es mejor que el obtenido por medio del método de los extremos. Entonces se utilizó dicha técnica en el algoritmo B&B.

Algoritmo 5.3 Heurística para obtener la solución inicial

Entrada: $\boldsymbol{G = (V, E)}$
Salida: $(\boldsymbol{mejorSolucion, CAB})$

```
1: etiquetas[1, ..., n]
2: lectura_instancia() //búsqueda del vértice del mayor grado
3: permut[mayorGrado] = n/2 //se le asigna la etiqueta del valor medio
4: for i = 0 to n do
5:     if(matAdy[mayorGrado] == 1) //se buscan los vértices adyacentes
6:         contador++;
7:         if(contador es par)
8:                 permut[i] = contador;  //se asigna una etiqueta del extremo
inferior
9:         else
10:            permut[i] = fin  //se asigna una etiqueta del extremo superior
11:        end if
12:    end if
13: end for
14: for i = 0 to n do
15:     if(permut[i] == 0) //revisar los vértices que no tienen etiqueta
16:         apermut[i] ← generarAleat(); //asignar etiqueta de forma aleatoria
17: end for
18: return  (mejorSolucion ← valorCAB(permut))
```

En cuanto a la implementación del criterio de poda; cada vez que un vértice es etiquetado, el valor del CAB parcial de los vértices se actualiza y se puede obtener un CAB parcial del grafo solamente con calcular el mínimo de los CAB parciales de los vértices. Con esto podemos saber si la rama que se está explorando, después de asignar una etiqueta a un vértice, sigue siendo prometedora o no.

La forma de podar ramas es la siguiente: cada vez que una etiqueta es asignada a un vértice, se calcula el CAB parcial del grafo y, si ese CAB parcial es menor o igual al valor de la mejor solución encontrada hasta el momento, se deja de explorar el resto de la rama (se poda), y se da marcha atrás para empezar a explorar otra rama que aún siga siendo prometedora (figura 5.4). El algoritmo 5.4 muestra esta implementación.

Algoritmo 5.4 Algoritmo BBCAB1

Entrada: $mejorSolucion$ // solución inicial
$etiquetas \leftarrow 1 \ldots n$ //etiquetas disponibles
$solucion = \emptyset$ //etiquetas asignadas a la solución
Salida: $(\varphi *, \text{CAB})$
1: if($etiquetas = \emptyset$)
2: minimoCAB($solucion$) //si llegó hasta la hoja se hace el cálculo
2: if($valorCAB(solucion) > valorCAB(mejorSolucion)$)
3: $mejorsolucion \leftarrow solucion$ //si el valor del CAB es mejor, se asigna como
4: end if nueva mejor solución encontrada
5: else
6: for $i = 0$ to n do
7: $l \leftarrow etiquetas[i]$
8: $solucion \leftarrow l$ //se agrega nueva etiqueta a la solución
9: CalcularCAB($ultimoVerticeEtiquetado(solucion)$)
10: if ($valorParcialCAB(solucion) > valorCAB(mejorSolucion)$)
11: CAB($solucion, etiquetas$) //se continua con el etiquetado
12: end if
13: quitar($solucion, l$)
14: $etiquetas[i] \leftarrow l$ //se regresa, es decir, se poda la rama
15: end for
16: end if
17: return $(solucion, CAB)$

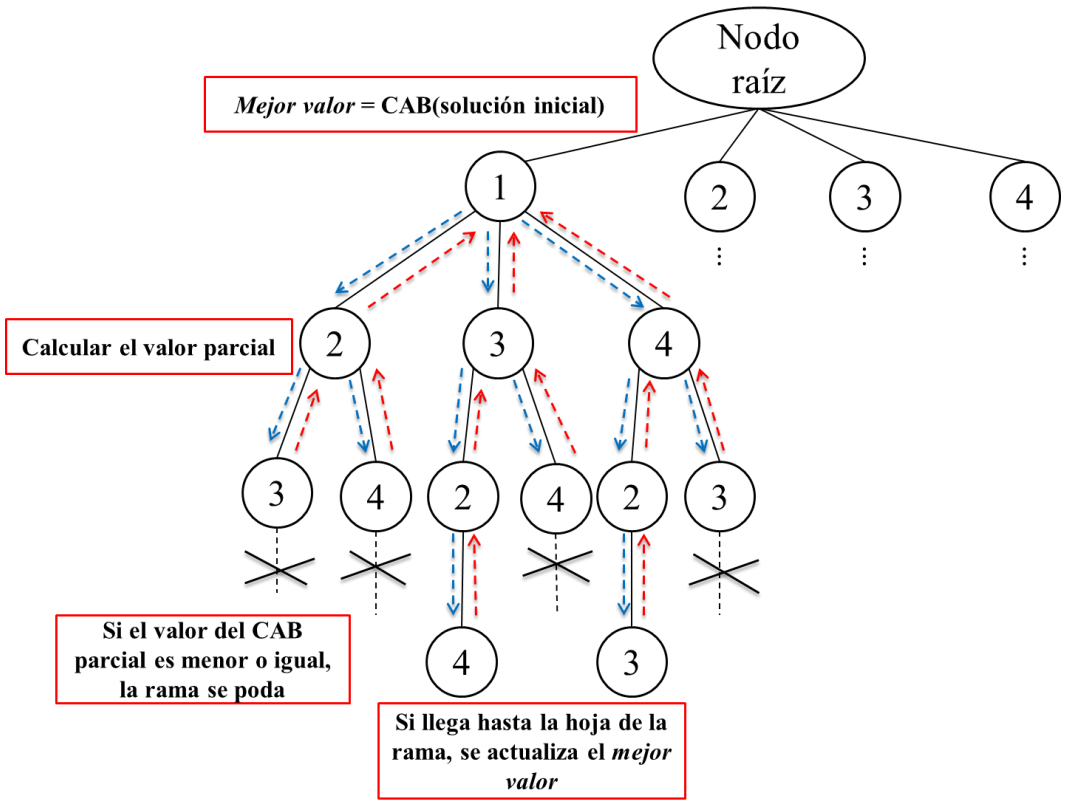


Figura 5. 4 Proceso de poda.

5.2.3 Algoritmo BBCAB2

En la primera implementación el tiempo promedio que se registró es muy grande, es decir consumió todo el tiempo asignado y solo resolvió el 6.72% (8 de 119) de las instancias. Por lo cual se decidió seguir mejorando las estrategias y así lograr mejorar el desempeño del algoritmo *branch and bound*, conservando el método heurístico para la generación de una solución inicial, así como el cálculo del CAB parcial.

Debido al bajo desempeño de algoritmo BBCAB1 se propuso nueva implementación que utiliza una lista de prioridad de acuerdo al valor de la solución parcial de cada nodo. Cuando el algoritmo accede a la lista para recuperar el nodo más prometedor, es el nodo de mayor prioridad (mayor valor CAB parcial), el que es recuperado, para expandirlo. El objetivo de este algoritmo es poder elegir en cada momento la mejor rama a explorar.

El algoritmo se divide en dos fases, en la primera se generan todos los nodos de profundidad dos, y se añaden a la lista de prioridad de acuerdo al valor de su CAB parcial ordenados de menor a mayor, como se muestra en la tabla 5.1. El pseudocódigo de este algoritmo puede verse en el algoritmo 5.5.

Algoritmo 5.5 Primera fase algoritmo BBCAB2

Entrada: $mejorSolucion$ // solución inicial
Salida: $(\varphi *, CAB)$

```
1: for(i = 0 to n) //generación de los nodos profundidad dos
2:     for(j = 0 to n)
2:         if(j! = i)
3:             cabParcial = calculaCabParcial(i, j)
4:             agregarNodo(i, j, cabParcial) //agregar el nodo a la lista de prioridad de
5:         end if                              // acuerdo al valor de su cabParcial
6:     end for
7: end for
```

Tabla 5. 1 Lista de prioridad en la que se añaden los nodos.

cab[1]	12, 23, 34, …
cab[2]	13, 24, 35, …
cab[3]	14, 25, 36, …
cab[4]	15, 26, 37, …
⋮	⋮

En la segunda fase se recupera el nodo más prometedor de la lista de prioridad, y se evalúa éste con todas las etiquetas disponibles, y si la evaluación del mínimo entre este CAB parcial y la cota es mayor o igual que el mejor valor global, se agrega esa etiqueta y el nodo con el nuevo valor CAB parcial es agregado a lista de prioridad. Este

proceso se realiza hasta que la solución final es generada. El pseudocódigo de este algoritmo puede verse en el algoritmo 5.6.

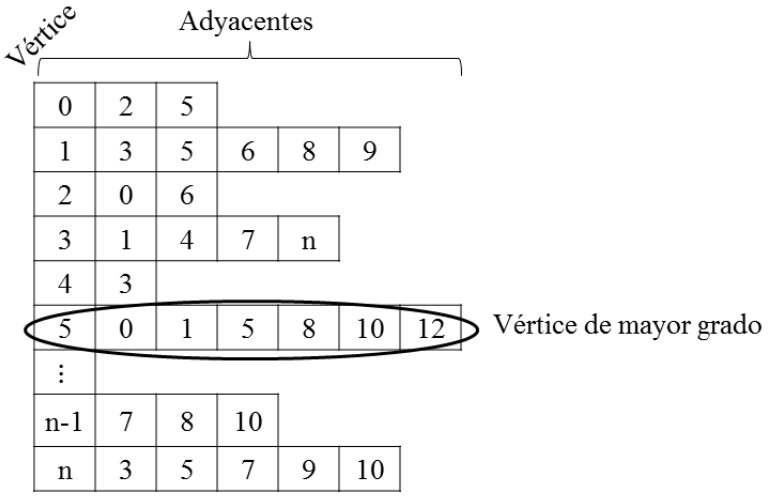
Algoritmo 5.6 Segunda fase algoritmo BBCAB2

Entrada: $mejorSolucion$ // solución inicial
Salida: $(\varphi *, CAB)$

```
1: while listaPrioridad ≠ ∅ do
2:       for(i = n/2 to 0) //se busca el elemento por prioridad en la lista
3:          if(listaPrioridad[i] ≠ null  &&  mejorVal ≤ i)
4:             break
5:          end if
6:       end for
7:       if(i == −1) //ya no hay elementos
8:          break;
9:       end if
10:       nodoTemp = tomarNodo(i)  //se toma el nodo más prometedor de la lista
11:       for(i = cantElemEtiq to n)
12:            cabParcial = calculaCabParcial(nodoTemp, i)
13:            if(cantElemEtiq + 1 == n) //si ya se asignaron todas las etiquetas
14:               if(cabParcial > mejorVal) //actualizar el mejor valor (sol. final)
15:                     mejorVal = cabParcial //arrElem tiene la solución completa
16:               end if
17:            else
18:                cota = calculaCota(nodoTemp, i) //calcular cota
19:                if(min(cabParcial, cota) > mejorVal)
20:                   agregaNodo(nodoTemp, cabParcial, cota, i) //agregar nodo a
21:               end if                                          // la lista de prioridad
22:            end if
23:       end for
24: end while
```

El cálculo del CAB parcial se basa en las etiquetas que han sido asignadas a los vértices del grafo por el algoritmo. Otro criterio que se implementó en este algoritmo, se denomina cota. Este criterio permite estimar el mejor valor del CAB que una solución parcial es capaz de obtener en función de las etiquetas que aún no han sido asignadas.

Para el cálculo de la cota se toma el subgrafo de vértice de mayor grado que no haya sido etiquetado por completo y se evalúan los vértices no etiquetados con las etiquetas disponibles. Para esto se tiene una estructura como la de la tabla 5.2 que contiene los vértices del grafo y su respectivo grado, ordenados de mayor a menor. Cuando se identifica el vértice de mayor grado se verifica si éste y si sus vértices adyacentes han sido etiquetados o no. Para esto se verifica en una estructura como la imagen 5.5, la cual contienen los vértices adyacentes a cada uno de los vértices del grafo.



Vértice	Adyacentes					
0	2	5				
1	3	5	6	8	9	
2	0	6				
3	1	4	7	n		
4	3					
5	0	1	5	8	10	12
⋮						
n-1	7	8	10			
n	3	5	7	9	10	

Figura 5. 5 Estructura que guarda los vértices adyacentes.

Por ejemplo, en el grafo de la figura 5.6 se puede notar que el vértice de mayor grado es el marcado con el número 6, es decir a este subgrafo es al que se le hará la evaluación para así obtener el valor de la cota.

Tabla 5. 2 Estructura en la que se guardan el grado de cada vértice.

Vértice	Grado
4	7
1	5
3	3
0	3
2	2

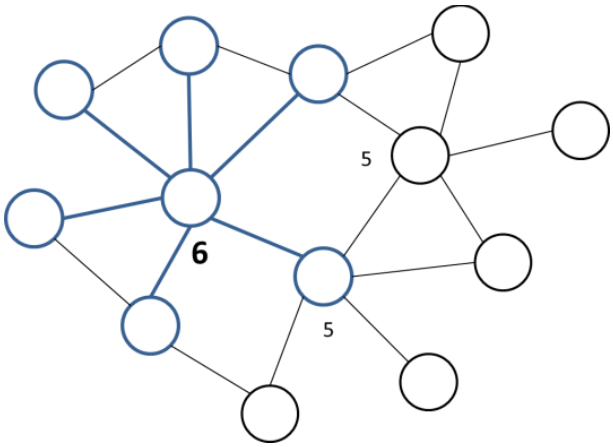


Figura 5. 6 Grafo con el vértice de mayor grado 6.

Para este subgrafo se pueden presentar dos casos; que el vértice de mayor grado esté etiquetado y que no esté etiquetado, como lo muestra la figura 5.7. En el primer caso, se hace una evaluación de las etiquetas disponibles de los vértices no etiquetados con respecto al vértice de mayor grado y se tomarán las que generen los mayores valores mínimos. En el segundo caso, se realiza el procedimiento anterior, pero ahora con la evaluación de las etiquetas disponibles para el vértice de mayor grado y en caso de que se genere un valor mayor o igual que el CAB parcial, el procedimiento se detiene.

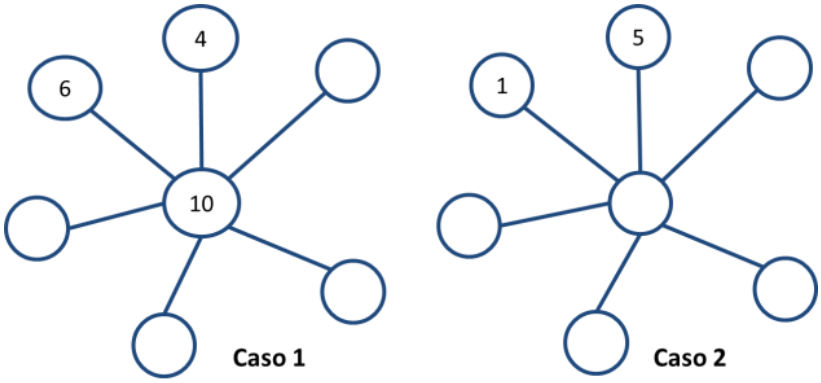


Figura 5. 7 Casos para el subgrafo del vértice de mayor grado.

Si el subgrafo de mayor grado ya ha sido etiquetado por completo, se procede a encontrar el siguiente vértice de mayor grado y verificar si los vértices han sido o no etiquetados. Por ejemplo, en el grafo de la figura 5.8 en dado caso que el subgrafo del vértice de grado 6 ya haya sido etiquetado se hace el cálculo de la cota con el siguiente vértice de mayor grado, que en este caso es el de grado 5. Al haber dos vértices de grado 5, se toma uno de los dos de manera arbitraria y se hace el cálculo de la cota, según sea el caso de nodo (caso 1 o caso 2).

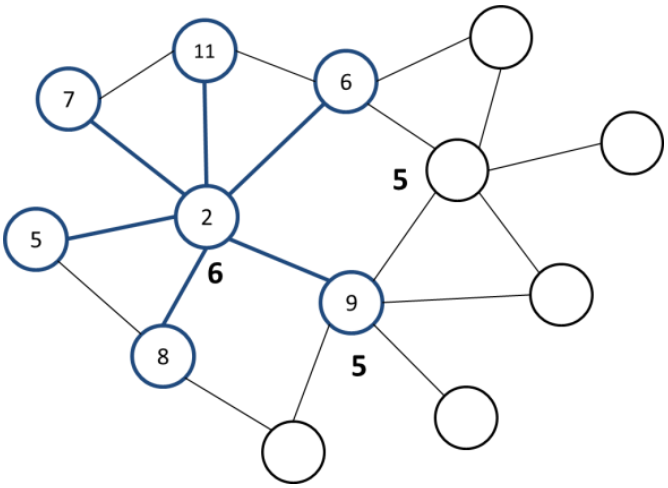


Figura 5. 8 Subgrafo del vértice de mayor grado ya etiquetado.

5.2.4 Algoritmo BBCAB3

Con la finalidad de mejorar la cantidad de óptimos encontrados y reducir el tiempo de ejecución del algoritmo, se realizó una mejora a la versión del algoritmo (BBCAB2), la cual consistió en cambiar la heurística para la construcción de la solución inicial (BBCAB3).

El método para construir la solución inicial se describe de manera general a continuación [Bansal y Srivastava, 2011]:

La solución construye una estructura basada en niveles por medio de una búsqueda en amplitud construida de manera aleatoria (RBFS) en donde los vértices pertenecientes a los niveles alternados no son adyacentes. Esta estructura asegura que los vértices pertenecientes a niveles alternados no sean vértices adyacentes. Dependiendo del vértice raíz se pueden obtener diferentes niveles del árbol [Bansal y Srivastava, 2011].

El etiquetado de los vértices para generar la solución inicial se hace de la siguiente forma:

Se etiquetan todos los vértices no adyacentes de manera subsecuente (en el orden en que fueron visitados en el árbol) en los niveles impares (o pares) y después se continua con el etiquetado de los vértices restantes usando una heurística voraz (GLAH). Este etiquetado tiende a incrementar el valor de la diferencia de las etiquetas adyacentes, lo cual resulta en un mayor valor del antibandwidth.

La heurística de etiquetado consiste en los siguientes pasos [Bansal y Srivastava, 2011]:

- Inicializar en ceros todos los vértices en la estructura de nivel.
- Elegir de manera aleatoria un nivel par o impar. Si es impar entonces, todos los vértices no adyacentes de niveles impares son etiquetados primero, de otra forma, los vértices no adyacentes de niveles pares son

etiquetados. Empezando con la etiqueta 1, el etiquetado es hecho de forma creciente de acuerdo a la numeración del árbol RBFS.

- Si al asignar las etiquetas uno o más generan el mismo valor mayor de CAB, la etiqueta a asignar se escoge de manera aleatoria.

Algoritmo 5.7 Generación de la solución inicial 2

```
1: while(no_visitado ≠ ∅)
2:     for i = 1 to n
3:         for j = 1 to n
4:             for k = 1 to adyac
5:                 temp[j] = min(|et_asig[i] − etiqueta[k]|)
6:             end for
7:             val = val_max (temp)
8:             etiqueta[j] = et_asig[i]
9:         end for
10:    end for
11: end while
```

Por ejemplo [Bansal y Srivastava, 2011], para el grafo de la figura 5.11 (a) se genera un estructura por niveles por medio de RBFS como lo muestra la figura 5.11 (b), donde los números a un costado de los vértices muestran el orden en el que fueron visitados los vértices del grafo (la línea punteada indica que los vértices 5 y 1 son adyacentes), de igual forma se muestra en la estructura de la figura 5.9.

R =	**6**	**8**	**3**	**4**	**2**	**9**	**5**	**7**	**1**	**10**

Figura 5. 9 Orden de visita de los vértices.

En la estructura de niveles L (figura 5.10) se muestra que los niveles 1, 2, 3 y 6, contienen un vértice; los niveles 4 y 5, contienen 2 y 4 vértices respectivamente. El orden de visita para los vértices del grafo por medio de RBFS se muestra en la estructura R.

L =	1	1	1	2	4	1

Figura 5. 10 Estructura por niveles.

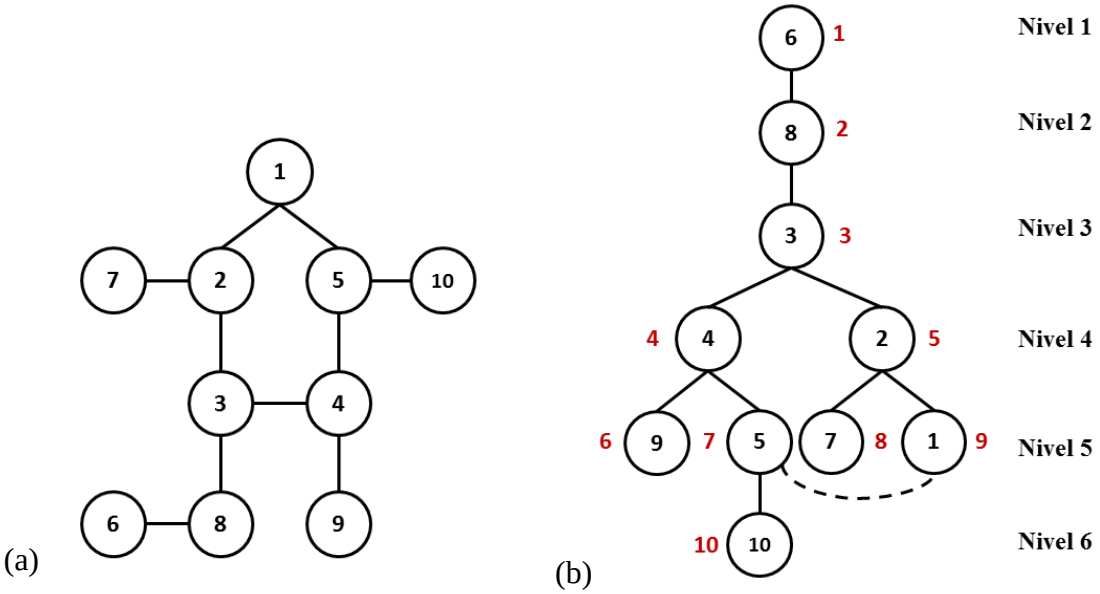


Figura 5. 11 Ejemplo de la estructura por niveles.
(a) Grafo original. (b) Árbol generado por búsqueda en anchura aleatoria (RBFS).

Empezando con el etiquetado desde el nivel 1 (impar), el vértice 6 recibe la etiqueta 1, el vértice 3 del nivel 3 recibe la etiqueta 2, al igual que los vértices 9, 5 y 7 del nivel 5, reciben las etiquetas 3, 4 y 5, respectivamente. El vértice 1 se mantiene sin etiquetar debido a que es adyacente al vértice 5.

Entonces hasta el momento las etiquetas quedan asignadas como se muestra en la figura 5.12,

Vértices	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**cab**
Etiquetas	**0**	**0**	**2**	**0**	**4**	**1**	**5**	**0**	**3**	**0**	

Figura 5. 12 Etiquetado de los niveles impares.

quedando las siguientes etiquetas disponibles: {6,7,8,9,10}.

Ahora la siguiente etiqueta a asignar es 6; para lo cual los vértices 1, 2, 4, 8, y 10 (los no visitados), son los candidatos. Para seleccionar el vértice al cual se le asignar se hacen los siguientes cálculos:

$temp[1] = \min\{|6 - etiqueta[2]|, |6 - etiqueta[5]|\}$
$temp[1] = \min\{|6 - 0|, |6 - 4|\}$
$temp[1] = \{6,2\}$
$temp[1] = 2$

$temp[2] = \min\{|6 - etiqueta[1]|, |6 - etiqueta[3]|, |6 - etiqueta[7]|\}$
$temp[2] = \min\{|6 - 0|, |6 - 2|, |6 - 5|\}$
$temp[2] = \{6,4,1\}$
$temp[2] = 1$

$temp[4] = \min\{|6 - etiqueta[3]|, |6 - etiqueta[5]|, |6 - etiqueta[9]|\}$
$temp[4] = \min\{|6 - 2|, |6 - 4|, |6 - 3|\}$
$temp[4] = \{4,2,3\}$
$temp[4] = 2$

$temp[8] = \min\{|6 - etiqueta[3]|, |6 - etiqueta[6]|\}$
$temp[8] = \min\{|6 - 2|, |6 - 1|\}$
$temp[8] = \{4,5\}$
$\boldsymbol{temp[8] = 4}$

$temp[10] = \min\{|6 - etiqueta[5]|\}$
$temp[10] = \min\{|6 - 4|\}$
$temp[10] = \{2\}$
$temp[10] = 2$

Como se puede observar en $temp[8]$ se genera el mayor valor CAB, por lo tanto la etiqueta 6 será asignada al vértice 8. La siguiente etiqueta a ser asignada es 7, y los vértices candidatos son 1, 2, 4 y 10. Los cálculos generan los siguientes valores: $temp[1] = temp[4] = temp[10] = 3$ y $temp[2] = 2$, de estos, el mayor valor es 3, y el vértice al cual se le asignará la etiqueta 7 se escoge de manera aleatoria.

Y así se continúa con el procedimiento hasta terminar con el etiquetado de todos los vértices del grafo. Quedando el etiquetado completo como se muestra en la estructura de la figura 5.13 y en la figura 5.14, con un valor $cab = 2$:

Vértices	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**cab**
Etiquetas	**7**	**10**	**2**	**8**	**4**	**1**	**5**	**6**	**3**	**9**	**2**

Figura 5. 13 Etiquetado final del grafo de la figura 5.11 (a).

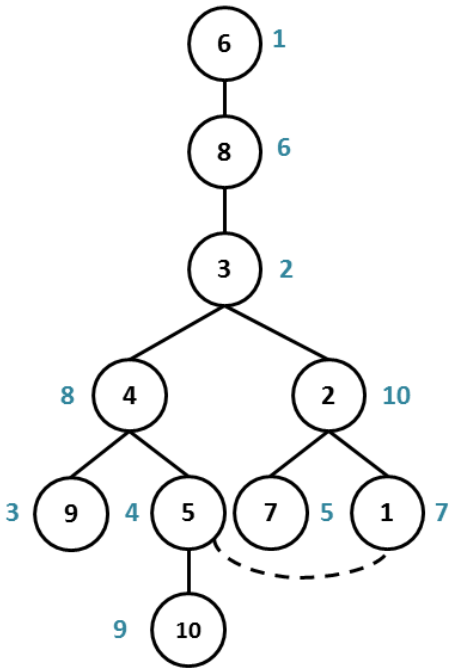


Figura 5. 14 Etiquetado completo del grafo de la figura 5.11 (a).

El algoritmo principal no fue modificado, quedando con la misma estructura anteriormente propuesta. El algoritmo se divide en dos fases, en la primera se generan todos los nodos de profundidad dos, y se añaden a la lista de prioridad de acuerdo al valor de su CAB parcial ordenados de menor a mayor. El pseudocódigo de este algoritmo puede verse en el algoritmo 5.8.

Algoritmo 5.8 Primera fase algoritmo BBCAB3

Entrada: $mejorSolucion$ // solución inicial
Salida: $(\varphi *, CAB)$

```
1: for(i = 0 to n) //generación de los nodos profundidad dos
2:     for(j = 0 to n)
2:         if(j! = i)
3:             cabParcial = calculaCabParcial(i, j)
4:             agregarNodo(i, j, cabParcial) //agregar el nodo a la lista de prioridad de
5:         end if                                  // acuerdo al valor de su cabParcial
6:     end for
7: end for
```

En la segunda fase, se recupera el nodo más prometedor de la lista de prioridad, y se evalúa éste con todas las etiquetas disponibles. Si la evaluación del mínimo entre este CAB parcial y la cota es mayor o igual que el mejor valor global, se agrega esa etiqueta y el nodo con el nuevo valor CAB parcial es agregado a lista de prioridad. Este proceso se realiza hasta que la solución final es generada. El pseudocódigo de este algoritmo puede verse en el algoritmo 5.9.

Algoritmo 5.9 Segunda fase algoritmo BBCAB3

Entrada: $mejorSolucion$ // solución inicial
Salida: $(\varphi *, CAB)$

1: while $listaPrioridad \neq \emptyset$ do
2: for(i $= \frac{n}{2}$ to 0) //se busca el elemento por prioridad en la lista
3: if($listaPrioridad[i] \neq$ null && $mejorVal \leq i$)
4: break
5: end if
6: end for
7: if($i == -1$) //ya no hay elementos
8: break;
9: end if
10: $nodoTemp =$ tomarNodo(i) //se toma el nodo más prometedor de la lista
11: for($i = cantElemEtiq$ to n)
12: $cabParcial =$ calculaCabParcial$(nodoTemp, i)$
13: if($cantElemEtiq + 1 == n$) //si ya se asignaron todas las etiquetas
14: if($cabParcial > mejorVal$) //actualizar el mejor valor (sol. final)
15: $mejorVal = cabParcial$ //arrElem tiene la solución completa
16: end if
17: else
18: $cota =$ calculaCota$(nodoTemp, i)$ //calcular cota
19: if(min$(cabParcial, cota) > mejorVal$)
20: agregaNodo$(nodoTemp, cabParcial, cota, i)$ //agregar nodo a
21: end if // la lista de prioridad
22: end if
23: end for
24: end while

Capítulo 6. Resultados

En este capítulo se describirán las instancias utilizadas en la fase de experimentación de cada uno de los algoritmos propuestos y en la implementación del modelo de programación lineal entera en la herramienta CPLEX, así como los resultados obtenidos en dicha experimentación.

6.1 Conjunto de Instancias para Evaluación

Para medir tanto el rendimiento de cada una de las versiones finales de los distintos algoritmos basados en *branch and bound* como el desempeño de la implementación en CPLEX del modelo de programación lineal entero propuesto, se utilizaron las instancias descritas a continuación.

6.1.1 Instancias con valor óptimo conocido

Path: este conjunto contiene 24 grafos construidos como un arreglo lineal de vértices de forma que cada vértice tiene un grado de dos, excepto el primer y el último vértice que tiene grado uno. El tamaño de estas instancias está en un rango de 50 a 1000. Para los grafos de tipo camino se conoce el valor óptimo CAB como:

$$cab(P_n) = \left\lceil \frac{n}{2} \right\rceil - 1$$

Cycle: este conjunto contiene 24 grafos construidos como un arreglo circular de vértices de forma tal que cada vértice tiene un grado dos. El tamaño de estas instancias está en un rango de 50 a 1000. Para los grafos de tipo ciclo se conoce el valor óptimo como:

$$cab(C_n) = \left\lceil \frac{n}{2} \right\rceil - 1$$

Toroidalmesh. Son un conjunto de grafos de tipo toroide. Donde un toroide se define como una malla toroidal bidimensional $C_n \times C_n$, definida por el producto cartesiano de dos ciclos [7]. El valor óptimo teórico se puede calcular como:

$$cab(C_n \times C_n) = \begin{cases} \dfrac{n(n-2)}{2} & si\ n\ es\ par \\ \left(\dfrac{(n-2)(n+1)}{2}\right) & si\ n\ es\ impar \end{cases}$$

6.1.2 Instancias con valor óptimo desconocido

Small: Este conjunto de pruebas está formado por 84 grafos, cuyo número de vértices, va desde los 16 vértices los grafos más pequeño hasta los 24 vértices los más grandes, y el rango del número de aristas está en un rango de 18 a 49.

Caterpillar: Este conjunto de instancias contiene 40 grafos. Cada grafo, P_{n1n2} está construido usando el camino P_{n1} y $n1$ copias del camino P_{n2}, donde cada vértice i en P_{n1} está conectado al primer vértice de la $i-ésima$ copia del camino P_{n2}.

Harwell-Boeing: Este grupo de instancias consiste en un conjunto de matrices de prueba estándar $M = (M_{ij})$ derivadas de problemas en sistemas lineales, mínimos cuadrados y cálculos de valores propios de una amplia variedad de disciplinas científicas de ingeniería. Los grafos son derivados de estas matrices considerando un arco (i, j) para cada elemento $M_{ij} \neq 0$. Se puede dividir en dos subconjuntos, el primero con las instancias más pequeñas de 30 a 100 vértices y el segundo subconjunto de 400 a 900 vértices.

6.2 Condiciones Experimentales

La experimentación fue hecha en un equipo con las siguientes características:

- Sistema operativo Windows 7.
- Procesador IntelCore i5-3210M CPU @ 2.50 GHz.
- Memoria RAM 6.0 GB.
- Entorno de Desarrollo: Para el modelo de Programación Lineal se utilizó NetBeans, Java y se utilizó la librería CPLEX 10.9. Para la implementación *branch and bound* se utilizó Visual Studio 2010, C#.

6.3 Resultados Obtenidos

Para comprobar que los resultados obtenidos mediante la implementación en CPLEX del modelo propuesto son los esperados, se tomaron las instancias con valores óptimos conocidos y se hizo una comparativa entre el valor obtenido teóricamente y el valor obtenido mediante la implementación. Las instancias utilizadas son instancias que tienen alrededor de 100 vértices. Se tomaron 12 instancias de cada uno de los siguientes conjuntos: Path y Cycle, y del conjunto Toroidalmesh se tomaron 8.

También se utilizaron los tres conjuntos cuyo valor óptimo es desconocido, para tener un marco comparativo de los resultados obtenidos con el método *branch and bound*. Estas instancias son los conjuntos Small, Caterpillar y HB; de los que se usaron 84, 15 y 12 instancias respectivamente.

Como se muestra en la sección 5.1 una de las propuestas de este trabajo es el modelo de programación lineal entera para el problema Antibandwidth Cíclico. Para poder experimentar con dicho modelo, se hizo una implementación de dicho modelo con una herramienta de optimización llamada CPLEX. Este software forma parte de una serie de programas denominados *General Problem Solver* (GPS) que son capaces de

resolver cualquier tipo de problema que previamente se haya definido mediante una formulación matemática, como es el caso de problema Antibandwidth y su modelo presentado en la sección 3.2.

6.3.1 Resultados de la implementación en CPLEX del modelo de programación lineal entera

La primera experimentación realizada fue la evaluación de la implementación en CPLEX del modelo propuesto en la sección 5.1.

En las tablas 6.1 y 6.2 se muestran los resultados obtenidos por CPLEX para el modelo de programación lineal entera. En la primera columna se observa el nombre del conjunto de instancias utilizadas con el total de instancias entre paréntesis, en la segunda columna se muestra el tiempo promedio de ejecución para cada instancia (medido en segundos), enseguida se muestra el total de óptimos encontrados para dicho conjunto, terminando con el porcentaje de óptimos. El tiempo de ejecución para cada instancia se limitó a 3600 segundos.

Tabla 6.1 Resultados de la implementación en CPLEX del modelo propuesto con instancias con valor óptimo conocido.

Instancias	**Tiempo promedio (s)**	**Óptimos encontrados**	**Porcentaje de óptimos**
Path (12)	676.50	12	100%
Cycle (12)	674.45	12	100%
Toroidalmesh (8)	307.51	8	100%
Total = 32	552.82	32	100%

Una vez que se hizo la experimentación con el algoritmo del modelo propuesto, se procedió a hacer las comparaciones de los resultados obtenidos de la implementación del modelo de programación lineal entera contra los valores generados mediante la formulación matemática de cada instancia utilizada (sección 6.1). Como se puede observar en la tabla 6.1, se obtuvo el 100% de óptimos en los tres grupos de instancias, en un tiempo promedio de 552.82 segundos.

En la siguiente experimentación se trabajó con instancias de los grupos de valor óptimo desconocido, para tener un marco comparativo de los resultados obtenidos con los algoritmos del método *branch and bound.* A estas instancias también se les asignó un tiempo límite de 3600 segundos.

Tabla 6.2 Resultados de la implementación en CPLEX del modelo propuesto con instancias con valor óptimo desconocido.

Instancias	**Tiempo promedio (s)**	**Óptimos encontrados**	**Porcentaje de óptimos**
Small (84)	3.57	84	100%
Caterpillar (15)	156.09	15	100%
HB (12)	1142.94	12	100%
Total = 111	434.2	111	100%

Al analizar los resultados se observó que en todas las instancias utilizadas les tomó un tiempo menor al asignado encontrar el valor objetivo y se lograron resolver todas las instancias propuestas. Entonces en esta experimentación se tomaron los valores encontrados por la implementación del modelo propuesto en CPLEX como los valores óptimos para hacer las comparaciones con los siguientes algoritmos desarrollados.

6.3.2 Resultados de los métodos branch and bound

Para comparar los resultados de la implementación del método *branch and bound* se utilizaron las instancias de los conjuntos Small, Caterpillar, Toroidalmesh y Harwell-Boeing con la misma cantidad de instancias de cada conjunto que para la experimentación del modelo.

Los resultados de las cuatro implementaciones se pueden observar en las tablas que se presentan a continuación; donde cada tabla contiene en la primera columna el nombre del conjunto de instancias con el total de instancias utilizadas entre paréntesis, en la segunda columna se muestra el tiempo promedio de ejecución (medido en segundos), se muestra el total de óptimos encontrados para dicho conjunto y por último el porcentaje de óptimos encontrados.

La primera versión del método *branch and bound,* se diseñó con recorridos en el árbol de exploración de manera completa, dicho algoritmo se ejecuta mientras haya etiquetas que asignar, es decir mientras no llegue a la última hoja de la rama.

Tabla 6.3 Resultados obtenidos por el algoritmo BBCAB.

Instancias	**Tiempo promedio (s)**	**Óptimos encontrados**	**Porcentaje de óptimos**
Small (84)	292.30	6	7.14%
Caterpillar (15)	3390.86	1	6.66%
Toroidalmesh (8)	3483.02	1	12.5%
HB (12)	3604.4	0	0%
Total = 119	2692.64	8	6.72%

Los resultados de la experimentación se pueden observar en la tabla 6.3. Como se observa solo se encontraron 8 valores óptimos de las 119 instancias utilizadas, es decir solo el 6.72% de ella fueron resueltas en un tiempo promedio de 2692.64 segundos.

Debido a que, para poder dar una solución completa, el algoritmo tiene que evaluar todo el árbol, es decir, hace una búsqueda exhaustiva, el tiempo de ejecución promedio de las instancias es casi igual al tiempo límite del algoritmo, mostrando así que le toma mucho tiempo al algoritmo poder encontrar una solución.

Enseguida se buscó una mejora en la estrategia presentada en la primera versión del método *branch and bound*, y se implementó la generación de una solución inicial y cálculo del CAB parcial para poder aplicar el criterio de poda.

Tabla 6.4 Resultados obtenidos por el algoritmo BBCAB1.

Instancias	**Tiempo promedio (s)**	**Óptimos encontrados**	**Porcentaje de óptimos**
Small (84)	256.81	25	29.76%
Caterpillar (15)	3254.91	4	26.66%
Toroidalmesh (8)	3220.85	3	37.5%
HB (12)	3602.62	1	8.33%
Total = 119	2583.79	33	27.73%

Logrando los resultados que se observan en la tabla 6.4, en la cual se puede observar que con la nueva implementación se aumentó a 27.73% de instancias resueltas, en un tiempo de 2583.79 segundos.

Como se observa, los resultados todavía están lejos de ser competitivos, motivo por el cual se presenta una tercera implementación del método *branch and bound*. En esta nueva implementación se hizo uso de una solución inicial y una lista de prioridad. Con estas estrategias se logró disminuir el tiempo promedio y aumentar el porcentaje de soluciones encontradas. Los resultados se pueden observar en la tabla 6.5.

Tabla 6.5 Resultados obtenidos por el algoritmo BBCAB2.

Instancias	**Tiempo promedio (s)**	**Óptimos encontrados**	**Porcentaje de óptimos**
Small (84)	249.26	27	32.14%
Caterpillar (15)	3248.20	4	26.66%
Toroidalmesh (8)	3168.04	3	37.5%
HB (12)	3116.93	1	8.33%
Total = 119	2445.6	35	29.41%

Los resultados muestran una mejoría, logrando aumentar el porcentaje de óptimos encontrados a 29.41% y disminuyendo el tiempo promedio a 2445.6 segundos.

Por último se realizó una implementación del método *branch and bound*, en la cual se hizo uso de una solución inicial por medio de una heurística propuesta por [Bansal y Srivastava, 2011], dejando la lista de prioridad del algoritmo anterior. Los resultados de la experimentación se pueden observar en la tabla 6.6.

Tabla 6.6 Resultados obtenidos por el algoritmo BBCAB3.

Instancias	**Tiempo promedio (s)**	**Óptimos encontrados**	**Porcentaje de óptimos**
Small (84)	195.76	47	55.95%
Caterpillar (15)	2143.69	9	60%
Toroidalmesh (8)	2318.76	5	62.5%
HB (12)	2601.96	4	33.33%
Total = 119	1815.04	65	54.62%

Con esta nueva implementación se logró disminuir de una manera considerable el tiempo promedio y se aumentó el porcentaje de soluciones encontradas a 54.62% en un tiempo promedio de 1815.04 segundos. Esta última implementación fue la que mejores resultados reportó del conjunto de implementaciones realizadas.

Capítulo 7. Conclusiones y Trabajo Futuro

En esta sección se presentan las conclusiones de este trabajo, así como también sugerencias para el desarrollo de los trabajos futuros en esta área de conocimiento.

7.1 Conclusiones

En este trabajo se desarrolló un enfoque exacto para la solución del problema Antibandwidth Cíclico (CAB), apoyado por algunos métodos heurísticos.

De acuerdo con la literatura especializada, se han propuesto algunas estrategias para la solución de este problema. Estas se pueden dividir en algoritmos metaheurísticos y acercamientos matemáticos. Hasta este momento no se identificaron trabajos que aborden el problema CAB mediante un método de solución exacta. Así mismo no se identificaron modelos de programación lineal entera para el problema CAB, razón por la cual se propuso en este trabajo.

La primer estrategia implementada consistió en una estrategia exhaustiva y logró resolver 8 instancias de 119 con un 6.72% de efectividad. Motivo por el cual se replanteó la estrategia en una segunda versión la cual se caracteriza por incluir una solución inicial y un cálculo parcial del CAB, logró resolver 33 instancias de 119 con un 27.73% de efectividad.

En la búsqueda de mejorar la estrategia *branch and bound* se propuso una tercera estrategia a la cual se le integró una lista de prioridad y una estrategia de poda, esta versión logró resolver 35 instancias de 119, es decir, tuvo una efectividad de 29.41%. Además a esta versión se le cambió la heurística para generar la solución inicial, esta modificación logró mejorar la calidad del algoritmo alcanzando una efectividad del 54.62% es decir alcanzó a resolver 65 instancias de un total de 119.

7.2 Aportaciones de la Investigación

Este proyecto de investigación cumplió satisfactoriamente con los objetivos planteados inicialmente, proponiendo lo siguiente:

- Implementación de un modelo de programación lineal entera para el problema Antibandwidth Cíclico en la herramienta de optimización CPLEX, basado en el modelo propuesto por [Duarte, 2010] para el problema AB.
- Una implementación basada en *branch and bound* en la cual se recorre el árbol de exploración de manera completa.
- Una implementación basada en *branch and bound* integrando una solución inicial y un cálculo parcial para poder implementar el criterio de poda.
- Una implementación basada en *branch and bound* que integra una solución inicial, una lista de prioridad y una cota para aplicar el criterio de poda.
- Una implementación basada en *branch and bound* con una metaheurística propuesta por [Bansal y Srivastava, 2011] para obtener una solución inicial, la lista de prioridad y la cota.

7.3 Trabajos Futuros

Para dar continuidad a este trabajo de investigación, a partir de la experiencia obtenida se propone diseñar un algoritmo metaheurístico con el cual se pueda hibridar las estrategias exactas aquí planteadas, con el objetivo de seguir mejorando la calidad de los resultados obtenidos por este conjunto de estrategias.

A demás se puede probar con diversos valores en la solución inicial, tomando como referencia valores cercanos al valor óptimo conocido.

Bibliografía

[Bansal y Srivastava, 2011] Richa Bansal and Kamal Srivastava. A memetic algorithm for the cyclic antibandwidth maximization problem. Soft Computing - A Fusion of Foundations, Methodologies and Applications, 15:397–412, 2011.

[Blum y Roli, 2003] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[Calamoneli et al, 2006] T. Calamoneri, A. Missini, L. Török, and I. Vrtó. Antibandwidth of complete k-ary trees. Electronic Notes in Discrete Mathematics, 24:259–266, 2006.

[Chinn et al., 1982] Chinn, P., Chvátalová, J., Dewdney, A., and Gibbs, N. (1982). The bandwidth problem for graphs and matrices—A survey. Journal of Graph Theory, 6:223–254.

[Dobrev, 2009] S. Dobrev, R. Královič, D. Pardubská, L. Török, and I. Vrtó. Antibandwidth and cyclic antibandwidth of hamming graphs. Electronic Notes in Discrete Mathematics, 34(0):295– 300, 2009.

[Duarte, 2007] Duarte Muñoz, A., Pantrigo Fernández, J., and Gallego Carrillo, M. Metaheurísticas. 2007.

[Duarte, 2010] Duarte A. Duarte, R. Martí, M. G. C. Resende, and R. M. A. Silva. GRASP with path relinking heuristics for the antibandwidth problem. Technical report, AT&T Labs Research Technical Report, 2009.

[Garey, 1979] Garey, M. R., Johnson, D. S., et al. *Computers and Intractability: A Guide to the Theory of NP-completeness*, 1979.

[Hale, 1980] W. K. Hale. Frequency assignment: Theory and applications. Proceedings of the IEEE, 68 (12):1497–1514, 1980.

[Hibbard and Yazlle, 2009] Hibbard Thomas N., Yazlle Jorge F. Teoría de grafos. Matemáticas discretas. 2009.

[Ignizio and Cavalier, 1994] James P. Ignizio and Tom M. Cavalier, Linear Programming, Edit. Prentice Hall, 1994.

[Kelly and Osman, 1996] J.P. Kelly and I.H. Osman. *Meta-Heuristic: Theory and Applications*. Kluwer Academic Publisher, 1996.

[Laguna and Delgado, 2007] M. Laguna, C. Delgado. Introducción a los metaheurísticos. Monográfico 3. 2007.

[Leung, 1984] J. Leung, O. Vornberger, and J. Witthoff. On some variants of the bandwidth minimization problem. SIAM Journal on Computing, 13(3):650–667, 1984.

[Lozano, 2012] Lozano, M., J., A. Duarte, F. Gortázar, R. Martí. A Hybrid Metaheuristic for the Cyclic Antibandwidth Problem. Submitted to Knowledge-Based Systems, 2012.

[Petit, 2011] Jordi Petit. Addenda to the Survey of Layout Problems, Bulletin of the EATCS, No.105, 2011.

[Raspaud, 2009] A. Raspaud, H. Schröder, O. Sykora, L. Török y I. Vrt'o. Antibandwidth and cyclic antibandwidth of meshes and hypercubes. Discrete Mathematics, 309(2009) 3541–3552, June 2009.

[Sykora, 2005] O. Sykora, L. Török, and I. Vrtó. The cyclic antibandwidth problem. Electronic Notes in Discrete Mathematics, 22:223–227, 2005.

[Zanakis, 1981] H. Zanakis, J.R. Stelios, and A. Evans. Heuristic optimization: Why, when and how to use it. *Interfaces*, 5(11):84–90, 1981.