UNIVERSIDAD
REY JUAN CARLOS

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

Curso Académico 2009/2010

Proyecto Fin de Carrera

Algoritmos exactos para el problema del Antibandwidth

Autor: Cristian Jaramillo Cáceres

Tutores: Abraham Duarte Muñoz
Eduardo García Pardo

Agradecimientos

Quiero agradecer a mis tutores, Abraham y Edu, toda la ayuda y el apoyo que me han ofrecido durante este largo periodo de tiempo y por la oportunidad de trabajar y aprender con ellos. Especialmente a Edu por la infinita paciencia que ha demostrado durante las innumerables tutorías que hemos tenido. También quiero mostrar mi agradecimiento a toda la gente del GAVAB que me ha ayudado, en especial a Juanjo y a Mica.

También quiero agradecer a mis padres el apoyo y la paciencia que han tenido conmigo durante estos años y que no me dieran por un caso perdido. Gracias a ellos he conseguido seguir adelante.

Por último, y no por ello menos importante, quiero agradecer a mis amigos el interés mostrado en todo momento y, por supuesto, a Amparo por haber estado a mi lado en todo momento apoyándome durante estos años.

Resumen

Los problemas de optimización se caracterizan por la búsqueda de un valor óptimo dentro del espacio de soluciones factibles, es decir, el valor de la mejor solución de entre todas las posibles. Para resolver este tipo de problemas existen programas genéricos (*solvers*) que, mediante una formulación matemática, son capaces de encontrar la solución óptima. También es posible la creación de programas específicos para resolver el problema. Dentro de los métodos específicos podemos distinguir entre técnicas aproximadas y exactas. Las técnicas aproximadas están basadas en heurísticas y son capaces de obtener soluciones de buena calidad en un tiempo limitado, pero sin poder certificar que dichas soluciones sean soluciones óptimas. Las técnicas exactas son aquellas que garantizan que la solución encontrada es la mejor posible, es decir, la solución óptima, aunque tienen el inconveniente de que invierten mucho tiempo en la ejecución.

Dentro de los problemas de optimización, el problema del Antibandwidth, es un problema de maximización y, en base a la estructura de sus soluciones, un problema de permutaciones. El problema consiste en etiquetar los vértices de un grafo no dirigido y no ponderado, de forma que se maximice la diferencia mínima de etiquetas entre los vértices adyacentes. Las etiquetas se corresponden con el rango de valores [1..N], siendo N el número de vértices del grafo. Un vértice sólo puede tener una etiqueta y, una etiqueta sólo puede estar asignada a un único vértice. Entre las distintas aplicaciones del problema del Antibandwidth, una de las aplicaciones más relevantes es el reparto de frecuencias de radio a distintas emisoras. Las emisoras que están cerca geográficamente no pueden tener frecuencias muy parecidas, ya que esto generaría interferencias, por lo que se busca asignar frecuencias lo más distantes posibles a emisoras cercanas y frecuencias parecidas a emisoras que estén alejadas entre sí.

El objetivo de este Proyecto de Fin de Carrera consiste en desarrollar un algoritmo exacto capaz de resolver el problema del Antibandwidth en un tiempo razonable, cuando el tamaño del mismo sea pequeño y, de dar cotas inferiores y superiores lo más ajustadas posibles, para problemas de mayor tamaño. Para ello se han propuesto diversos esquemas algorítmicos cuyo rendimiento será evaluado, comparando sus resultados con *software* comercial capaz de resolver este tipo de problemas, en concreto el *solver* CPLEX.

Índice general

Índice de figuras

Índice de acrónimos

- **AB**: *Antibandwidth*
- **GPS**: *General Problem Solver*
- **GRASP**: *Greedy Randomized Adaptive Search Procedure*
- **IDE**: Entorno de Desarrollo Integrado
- **ILP** o **IP**: *Integer Linear Programming*
- **LB**: *Lower Bound*
- **LP**: *Linear Programming*
- **LPPL**: LaTeX Project Public License
- **PFC**: Proyecto de Fin de Carrera
- **POO**: Programación Orientada a Objetos
- **PUD**: Proceso Unificado de Desarrollo
- **TSP**: *Travelling Salesman Problem*
- **UB**: *Upper Bound*
- **VRP**: *Vehicle Routing Problem*
- **XP**: *eXtreme Programming* - Programación Extrema

Capítulo 1

Introducción

En este capítulo se hará una breve introducción al problema que se pretende resolver en este Proyecto de Fin de Carrera (PFC). Para ello se describirá el mismo tanto de manera prosaica como matemática.

1.1. Marco de trabajo

La temática que se trata en este PFC pertenece a la rama de problemas de optimización. Estos problemas se caracterizan por la búsqueda de un valor óptimo para el problema en cuestión. Este valor óptimo se define como el valor de la mejor solución de entre todas las posibles. Algunos ejemplos de problemas de optimización conocidos son: el problema de la mochila (*Rucksack problem*), el problema del viajante (*Travelling Salesman Problem - TSP*), o el problema de rutas de vehículos (*Vehicle Routing Problem - VRP*).

Los problemas de optimización pueden ser problemas de maximización o problemas de minimización. En los problemas de maximización, como en el de la mochila, se quiere que el valor de la solución sea lo mayor posible y, en los de minimización, como en el del viajante, se desea que dicho valor sea lo menor posible.

Existen distintos tipos de problemas de optimización: binarios, de permutaciones, de optimización global, etc.

1.2. Descripción del problema del Antibandwidth

El problema del Antibandwidth (AB) es un problema de maximización y, particularmente, un problema de permutaciones, que consiste en etiquetar los vértices de un grafo. En concreto, se emplearán grafos conexos no dirigidos y no ponderados. El objetivo del problema es asignar las etiquetas del rango [1..N] a los N vértices del grafo, de forma que se maximice la diferencia mínima de etiquetas entre vértices adyacentes. Se entiende por vértices adyacentes, a aquellos vértices de un grafo que están unidos mediante una arista.

No existen restricciones sobre el etiquetado de los vértices, es decir, todos los vértices pueden ser etiquetados con cualquier etiqueta siempre y cuando no haya etiquetas repetidas, ni tampoco haya un vértice con más de una etiqueta. Así, cualquier etiquetado se convierte en una solución al problema, teniendo un total de N! soluciones posibles.

El problema del AB es un problema NP-Completo [6]. Para garantizar que la solución encontrada es la solución óptima habría que explorar las N! soluciones posibles, obtenidas al permutar las etiquetas sobre los vértices.

1.2.1. Cálculo del Antibandwidth

El cálculo del valor del AB de un grafo se realiza en dos fases:

- Cálculo del AB de cada vértice.
- Obtención del AB general del grafo.

Antibandwidth de un vértice

Para calcular el valor del AB de un vértice se realiza la diferencia, en valor absoluto, de la etiqueta del vértice sobre el cual se está calculando el AB y cada una de las etiquetas de los vértices adyacentes. Una vez se han obtenido todas las diferencias de etiquetas, se elige la diferencia mínima, y se asigna como valor del AB del vértice. En la Figura 1.1 se muestra el cálculo del valor del AB para el vértice F. Este vértice tiene asignada la etiqueta 8 y sus adyacentes, los vértices B, C, G y H, las etiquetas 3, 5, 2 y 4 respectivamente. Una vez se ha calculado la diferencia de etiquetas entre la etiqueta del vértice F y las etiquetas de sus adyacentes se elige la menor de todas. En este caso el valor del AB del vértice F es 3.

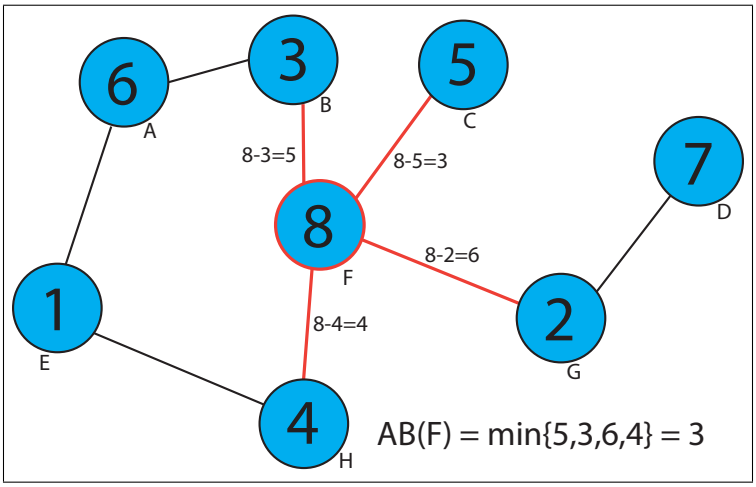

Figura 1.1: Cálculo del AB de un vértice

Antibandwidth del grafo

Una vez obtenido el valor del AB para cada vértice del grafo, se elige el valor mínimo de entre todos los vértices y se asigna como el valor del AB del grafo. La Figura 1.2 ilustra el cálculo del AB de un grafo. En este caso el valor del AB para el grafo mostrado es 3.

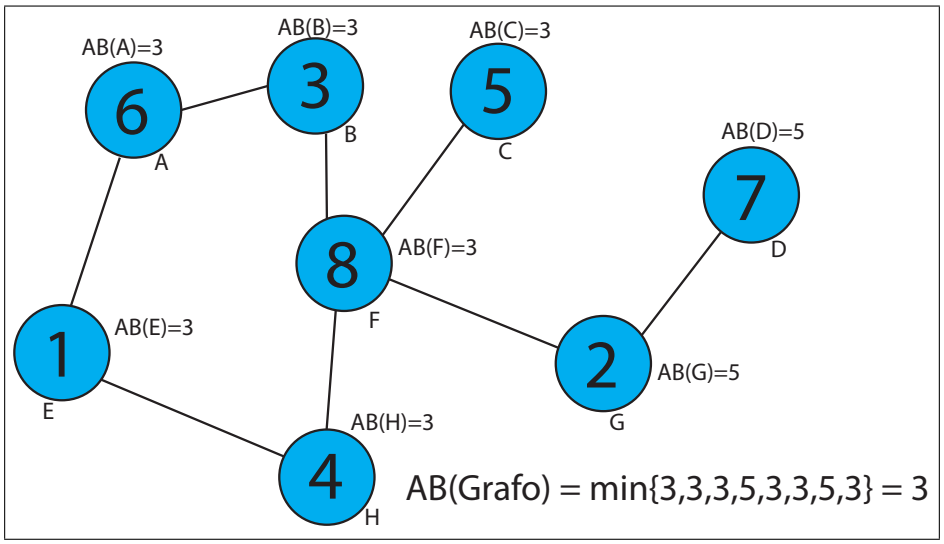

Figura 1.2: Cálculo del AB general de un grafo

1.3. Aplicaciones del problema

Entre las distintas aplicaciones del problema del AB, una de las aplicaciones más relevantes es el reparto de frecuencias de radio a distintas emisoras. Las emisoras que están cerca geográficamente no pueden tener frecuencias muy parecidas, ya que esto generaría interferencias. Para ello se modela como un grafo, que representa la red de interferencias entre emisoras cercanas y se asigna una frecuencia a cada emisora. El objetivo es asignar frecuencias lo más dispares posible a emisoras que estén cercanas en la red de interferencias, con el fin de reducir las interferencias al mínimo.

1.4. Formulación matemática del problema

La programación lineal (LP) es un método matemático de optimización mediante el cual se resuelven problemas formulados a través de ecuaciones lineales. Aquí, la función objetivo consiste en maximizar o minimizar el valor que se obtiene de la resolución del problema en cuestión. En concreto la programación entera lineal (ILP o IP) es aquella en la cual las variables de las ecuaciones lineales que definen el problema son de tipo entero. A su vez la IP puede clasificarse en tres tipos: IP Pura, IP Mixta (MILP) y *zero-one* IP.

- IP Pura: Es aquella en la cual todas las variables son de tipo entero.
- IP Mixta: Es aquella en la cual algunas las variables son de tipo entero.
- *Zero-one* IP: Es aquella en la cual las variables son binarias y sólo pueden tomar los valores: 0 (*false*) ó 1 (*true*).

La mayoría de los problemas de tipo IP son muy difíciles de resolver, por lo que se suelen utilizar algoritmos específicos para problemas concretos frente a técnicas genéricas. Además, su uso se puede combinar con heurísticas capaces de generar soluciones muy cercanas al valor óptimo de forma relativamente rápida y sencilla. En este caso, se ha intentado resolver el problema del AB mediante un algoritmo exacto específico para el problema.

Existe 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. Para los experimentos se utilizará uno de los *solvers* comerciales más utilizados, CPLEX, al cual se le pasará la formulación matemática del problema propuesta en [10]. A continuación se describe la formulación matemática empleada para resolver el problema mediante el uso de CPLEX.

1.4.1. Definición de las variables enteras

- Sea x_{ik} una variable binaria que toma valor 1 si y solo si $f(i) = k$, por ejemplo: Al nodo i se le asigna la etiqueta k.
- Se define $l_i = f(i) \in \{1, 2, .., n\}$ como la etiqueta que se le asigna al nodo i.
- Sea $b = AB_f(G) = min\{|f(u) - f(v)| : (u, v) \in E\}$ el AB correspondiente al grafo f.
- El objetivo del AB consiste en encontrar un etiquetado de f^* que maximice el valor de b.

1.4.2. Restricciones

- Cada etiqueta deber ser asignada a un único nodo:

$$\sum_{i=1}^{n} x_{ik} = 1, \forall k = 1..n$$

- Cada nodo sólo puede tener una etiqueta asignada:

$$\sum_{k=1}^{n} x_{ik} = 1, \forall i = 1..n$$

- Cada etiqueta l_i es una función sobre las variables binarias x_{ik}:

$$\sum_{k=1}^{n} k \cdot x_{ik} = l_i, \forall i = 1..n$$

- b tiene que ser mínimo ($b = min\{|l_i - l_j| : (i,j) \in E\}$):

$$b \leq |l_i - l_j|,\ (i,j) \in E$$

- Las variables binarias x_{ik} solo pueden tomar valor 0 ó 1:

$$x_{ik} \in \{0,1\}, \forall\ i,k = 1..n$$

- Las etiquetas l_i toman valores en el rango $\{1..n\}$:

$$l_i \in \{1..n\}, \forall\ i = 1..n$$

- Las restricciones $b \leq |l_i - l_j|,\ (i,j) \in E$ son no lineales:
  - Si $l_i \geq l_j$ entonces $b \leq l_i - l_j$
  - Si no, $b \leq -(l_i - l_j)$
  - Introducir dos variables binarias para indicar el caso:
    - Si $l_i \geq l_j$ entonces $y_{ij} = 0\ \wedge\ z_{ij} = 1$
    - Si no, $y_{ij} = 1\ \wedge\ z_{ij} = 0$

- Las restricciones $b \leq |l_i - l_j|,\ (i,j) \in E$ se convierten en:

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

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

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

$$b \geq 1$$

Resumen:

$$\textbf{\textit{maximizar } b} \qquad \textbf{b} \geq \textbf{1}$$

$$\sum_{i=1}^{n} x_{ik} = 1, \forall k = 1..n \qquad x_{ik} \in \{0,1\},\ \forall\ (i,k) \in E$$

$$\sum_{k=1}^{n} x_{ik} = 1, \forall i = 1..n$$

$$\sum_{k=1}^{n} k \cdot x_{ik} = l_i, \forall i = 1..n \qquad l_i \in \{1..n\},\ \forall\ i = 1..n$$

$$b - (l_i - l_j) \leq 2y_{ij}(n-1),\ \forall\ (i,j) \in E \qquad y_{ik} \in \{0,1\},\ \forall\ (i,k) \in E$$

$$b + (l_i - l_j) \leq 2z_{ij}(n-1),\ \forall\ (i,j) \in E \qquad z_{ik} \in \{0,1\},\ \forall\ (i,k) \in E$$

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

1.5. Otras técnicas de resolución

Como se ha mencionado anteriormente, además de las técnicas genéricas basadas en el uso de *solvers* mediante la aplicación de una formulación matemática, existen otra serie de técnicas específicas para resolver este tipo de problemas.
Estás técnicas pueden clasificarse en aproximadas y exactas.

- **Aproximadas**: Las técnicas aproximadas se basan en la obtención de soluciones de buena calidad, pero sin poder certificar que sean soluciones óptimas. La principal ventaja que tienen este tipo de técnicas es que reducen significativamente el tiempo que se tarda en encontrar una solución de calidad. En definitiva, buscan un balance entre la calidad de la solución generada, y el tiempo de ejecución necesario para alcanzarla. Estas técnicas aproximadas se basan en heurísticas, que son ideas intuitivas que permiten encontrar una buena solución a un problema. Dichas heurísticas se embeben en metaheurísticas, que son esquemas algorítmicos que implementan una heurística para resolver un problema concreto. Las metaheurísticas son empleadas con éxito en la resolución de problemas de optimización. Un ejemplo de algoritmo basado en metaheurísticas para resolver el problema del Antibandwidth es el *Greedy Randomized Adaptive Search Procedure* (*GRASP*) propuesto en [10].

- **Exactas**: Las técnicas exactas son aquellas que garantizan que la solución encontrada es la mejor posible, es decir, la solución óptima. El gran inconveniente de estas técnicas es que para poder encontrar el óptimo invierten mucho tiempo en la ejecución. Tal es la cantidad de tiempo invertido, que para problemas complejos, o muy grandes, el tiempo de ejecución se aproxima a infinito. Vuelta atrás (*backtracking*),

y ramificación y acotación (*branch and bound*), son algunos ejemplos de esquemas algorítmicos para la resolución exacta de problemas de optimización. En los casos en los que estas técnicas no pueden garantizar el óptimo en un tiempo determinado, garantizan entre qué valores se encuentra dicho óptimo. Estos valores se denominan *Lower Bound* (LB) y *Upper Bound* (UB). Para un problema de maximización, el LB es el mejor valor encontrado por el algoritmo exacto en un tiempo determinado y, el UB es el mejor valor que el algoritmo estima que es posible encontrar en el espacio de soluciones que le queda por explorar. La diferencia entre el UB y el LB, se conoce como *gap* y, se utiliza para medir cómo de bueno es un algoritmo exacto. Si dos algoritmos exactos no son capaces de encontrar el óptimo en un tiempo determinado, el algoritmo que tenga un *gap* menor se considera mejor.

- **Vueltra atrás**: Estos algoritmos por lo general utilizan una implementación recursiva y, en cada paso de la recursividad, generan una parte de la solución. Si en algún momento detectan que la solución parcial generada no es viable, dan marcha atrás en la recursividad, y regresan a un punto desde el cual la solución parcial todavía seguía siendo viable, explorando a continuación otras alternativas. Si en algún momento llegan a una solución completa, la evalúan y la comparan con la mejor solución encontrada hasta el momento, quedándose con la mejor de ellas.
- **Ramificación y acotación**: Estos algoritmos se basan en la generación de una serie de soluciones parciales y, en cada paso, exploran la solución parcial más prometedora hasta dar con la solución óptima. Por lo general, estos algoritmos son implementados iterativamente, y se apoyan en una cola de prioridad para almacenar soluciones parciales que todavía son susceptibles de ser exploradas.

Para la resolución del problema planteado en este PFC se utilizarán las técnicas exactas descritas anteriormente.

En lo que resta de memoria, se describirán los objetivos que se desean alcanzar mediante el desarrollo de este PFC (Capítulo 2). Se explicarán las distintas técnicas algorítmicas empleadas para la resolución del problema (Capítulo 3) y, durante el Capítulo 4, se realizará una descripción de la metodología de desarrollo empleada en la elaboración del proyecto. En el Capítulo 5 se presentarán los resultados experimentales obtenidos durante la aplicación de los algoritmos propuestos a un conjunto de problemas estándar y, por último, en el Capítulo 6 se expondrán las conclusiones.

Capítulo 2

Objetivos

En este capítulo se describen los distintos objetivos que se pretenden alcanzar durante el desarrollo de este PFC.

2.1. Objetivo principal

La finalidad de este PFC es implementar un algoritmo exacto capaz de resolver el problema del Antibandwidth en un tiempo razonable.

Para lograr este objetivo principal se crearon subobjetivos incrementales más pequeños. Estos subobjetivos se definen en función de la complejidad de la técnica algoritmica empleada, partiendo de un algoritmo simple hasta llegar a un algoritmo más elaborado y eficiente.

2.2. Objetivos parciales

Los objetivos parciales que se propusieron serán descritos a continuación y permiten la aplicación de una metodología de desarrollo basada en protrotipos o versiones como la programación extrema.

Versión inicial

El primer objetivo que se propone es la implementación de una versión lo más básica y sencilla posible del algoritmo capaz de resolver el problema del AB, que permita tener una referencia con la cual medir el impacto de las distintas mejoras que puedan ser realizadas a lo largo del desarrollo del proyecto. Esta versión estará basada en un algoritmo exacto de vuelta atrás.

Proposición de cotas

Una vez terminada la versión inicial se plantearán y desarrollarán una serie de cotas para poder desarrollar un algoritmo exacto de ramificación y acotación.

Versión mejorada

Una vez las cotas estén desarrolladas se comenzará el desarrollo de una versión más elaborada que haga uso de técnicas algorítmicas que sean capaces de mejorar la calidad de las soluciones obtenidas. Esta nueva versión implementará un algoritmo de ramificación y acotación recursivo basado en la versión inicial.

Recorridos del árbol

Todos las versiones que se crearán hasta este punto estarán basadas en algoritmos recursivos, por lo que el siguiente hito a conseguir será implementar un algoritmo iterativo exacto basado en un algoritmo de ramificación y acotación. Este algoritmo utilizará una cola de prioridad capaz de organizar los nodos del árbol de exploración de manera que se haga un recorrido inteligente y utilizará las cotas descritas en el punto anterior. Todo esto permitirá obtener soluciones de mayor calidad.

Evaluación de los resultados

Una vez todos los algoritmos exactos estén desarrollados por completo se realizará una implementación de la formulación matemática descrita en la Sección 1.4 mediante el *solver* CPLEX. Esto permitirá tener un marco de referencia con el cual comparar los algoritmos exactos desarrollados durante este proyecto y medir su calidad.

Capítulo 3

Descripción algorítmica

En este capítulo se hará una descripción de las diversas técnicas algorítmicas empleadas para la resolución del problema del AB, comenzando por las técnicas más elementales y llegando a técnicas más complejas. Inicialmente se definirán algunos conceptos básicos, necesarios para la compresión del capítulo.

3.1. Conceptos básicos

El problema del AB es un problema de permutaciones. Estas permutaciones son los distintos etiquetados de los vértices del grafo que representa el modelado del problema y que dan lugar a las distintas soluciones del mismo. Dos permutaciones distintas que generan el mismo valor del AB son igualmente válidas y, a priori, no se puede tener una preferencia clara por ninguna de ellas. El hecho de que en las N! soluciones posibles haya un subconjunto de ellas en el cual, todas y cada una de las soluciones generen el valor óptimo del AB, nos permite tener distintos caminos para llegar a la solución de valor óptimo. En el caso de que sólo exista una única solución óptima el problema se complica, puesto que hay que encontrar el único camino existente hasta la permutación óptima de entre las N! soluciones.

Los algoritmos empleados para la resolución del problema son algoritmos de exploración que, como su nombre indica, exploran todos los posibles caminos hasta encontrar uno que conduzca hasta una solución de valor óptimo.

Cuando se describen algoritmos de exploración, es común encontrar términos como "árbol de exploración", "nodo", "nodo raíz", "nodo hoja", "rama del árbol de exploración", "camino", o "poda", cuya definición debe conocerse previamente:

- Los **nodos** representan cada una de las etapas necesarias para poder generar una solución al problema. Esta etapa se compone de la solución parcial construida hasta el momento y de los datos necesarios para poder llegar a una solución completa. Es-

tos nodos pueden ser implementados mediante estructuras de datos o, dependiendo del tipo del algoritmo que se utilice, de una manera implícita, como por ejemplo, en un algoritmo recursivo, donde el nodo está descrito en cada paso de la recursividad que lleva hasta una solución completa.

- El **árbol de exploración** es una representación imaginaria de todos los posibles nodos que se generarían para un problema determinado. En la parte de arriba del árbol de exploración se encuentra la solución vacía o nodo raíz, mientras que en la parte más baja del árbol se encuentran los nodos hoja, representando soluciones completas. Entre el nodo raíz y los nodos hojas se encuentran el resto de nodos que representan las soluciones parciales.

- Las **ramas del árbol** de exploración representan a cada una de las ramificaciones que se producen a partir de un nodo determinado.

- Un **camino** es un recorrido que parte del nodo raíz y llega a un único nodo hoja, pasando por una serie de nodos intermedios.

- Se denomina **podar** una **rama del árbol** de exploración a desechar caminos que, mediante un pequeño estudio, se han considerado no viables para llegar a una solución mejor que la que ya se tiene. La forma de ver si una rama es susceptible de ser podada o no, consiste en calcular el valor una **función de cota** que estima cómo de buena sería la mejor solución alcanzable por esa rama, en caso de recorrer todos los posibles caminos que hay en ella.

En la Figura 3.1 podemos observar una representación de un árbol de exploración, en el cual se señalan las partes anteriormente descritas.

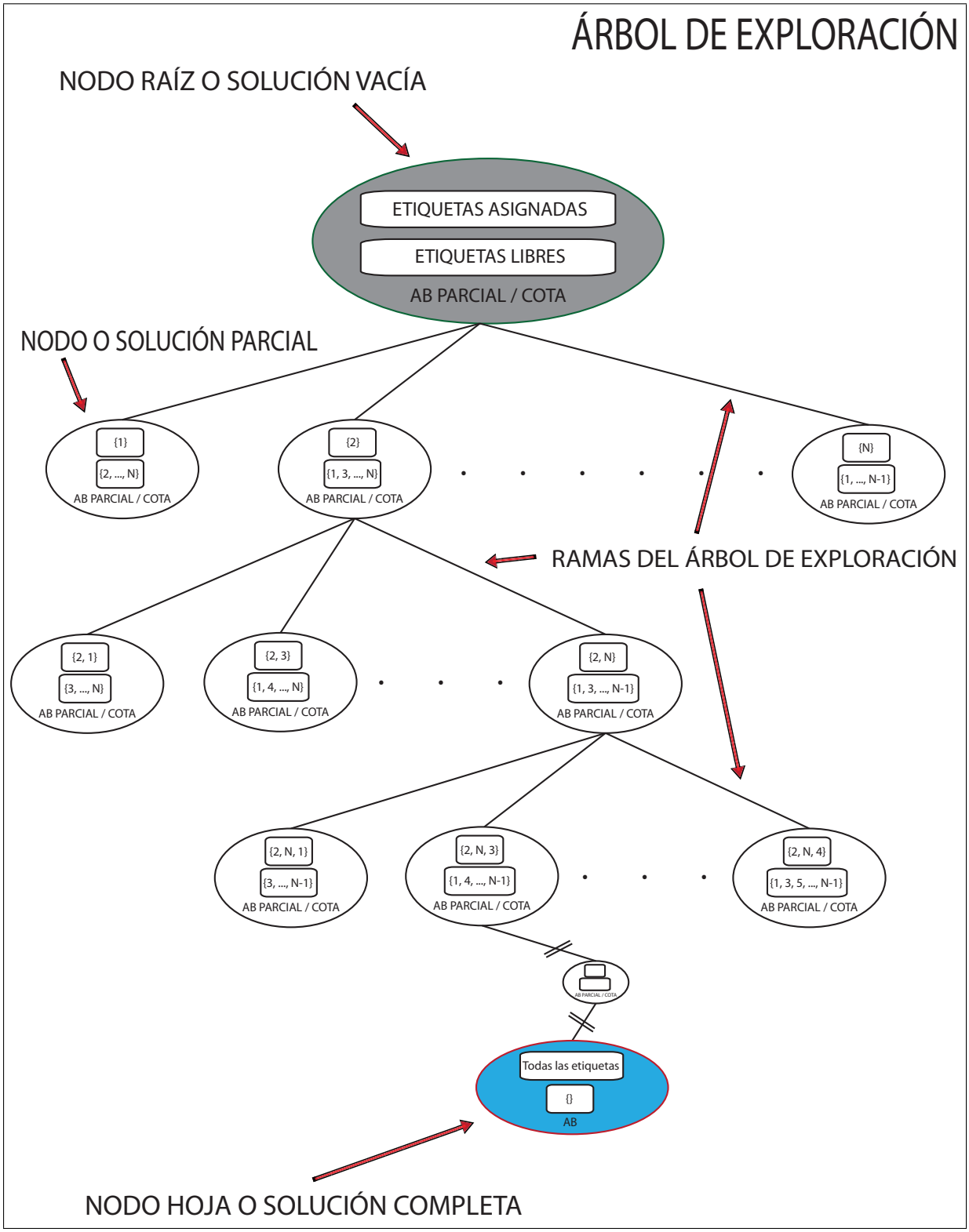

Figura 3.1: Representación de un árbol de exploración

3.2. Breve descripción de las técnicas empleadas

La descripción de las distintas técnicas algorítmicas empleadas para la resolución del problema se hará en orden cronológico según fueron desarrolladas. Sin embargo, hay algunos puntos de interés que deben ser brevemente expuestos antes de describirlos a fondo.

- **Recorridos del árbol de exploración**: Durante el desarrollo de este PFC se implementarán distintos algoritmos de exploración. Cada uno de estos algoritmos recorre el árbol de exploración de forma distinta, por lo que las soluciones encontradas por cada uno de ellos, aunque tengan el mismo valor del AB, puede que sean permutaciones distintas. A continuación se hará una breve descripción de los recorridos que realizan los distintos algoritmos:

  - **Vuelta atrás**: Esta técnica recorre el árbol de exploración en profundidad y siguiendo un orden lexicográfico.
  - **Ramificación y poda**: Esta técnica algorítmica recorre el árbol de exploración siguiendo el nodo más prometedor encontrado hasta el momento y que no haya sido previamente expandido. Para poder decidir qué nodo es el más prometedor en cada momento, se utiliza el valor de la cota del nodo, y se ordenan los distintos nodos dentro de una cola de prioridad. Esta cola dará mayor prioridad a los nodos con menor cota, en caso de ser un problema de minimización, y en caso de ser un problema de maximización, se priorizarán los nodos con una cota más alta.

- **Cotas propuestas**: Como se ha contado anteriormente, para poder podar las ramas del árbol de exploración es necesario una función de cota que evalúe lo prometedora que es una rama del árbol.
  Un cota es una estimación que se realiza sobre un nodo del árbol de exploración. Para poder realizar esta estimación se tiene en cuenta la solución parcial que contiene el nodo, así como los datos restantes que no han sido añadidos a la solución parcial. Mediante este conjunto de valores se estima cuál es el mejor valor que podría alcanzar dicho nodo. Esto es lo que se denomina cota de un nodo.
  Las cotas propuestas para el problema del AB se basan en buscar el mejor etiquetado posible del vértice de mayor grado del grafo. Para realizar este etiquetado se tienen en cuenta las etiquetas que puedan haber sido asignadas previamente, tanto al vértice de mayor grado, como a sus adyacentes. Para etiquetar los vértices que estén libres se usan las etiquetas que todavía no han sido asignadas a ningún vértice.

- **Factorización**: En matemáticas la factorización hace referencia al hecho de expresar un objeto como producto de otros objetos más pequeños (factores). Sin embargo, este término también tiene su descripción en el contexto de la informática.
  Durante la ejecución de un algoritmo de exploración, cada nodo se crea a partir de otro nodo del árbol de exploración, al que se le añade una nueva parte de la solución. Cuando esta nueva parte es añadida, es posible que no todos los elementos que componen la solución se vean afectados, dependiendo de las restricciones de cada problema. Por lo tanto es posible reducir la evaluación de la solución, teniendo en cuenta únicamente a los elementos que se han visto afectados al añadir la nueva parte a la solución. Esto permite ir evaluando poco a poco cada nodo del árbol de exploración y, una vez llegado a un nodo hoja, tener su función objetivo completamente calculada.

3.3. Vuelta atrás (Backtracking)

La primera técnica empleada que se utilizó para resolver el problema es la técnica de vuelta atrás o *backtracking*. Se optó por comenzar el desarrollo del problema con esta técnica porque es la más sencilla de implementar, lo que permite obtener una primera versión funcional en poco tiempo que sirva como *baseline*. Además, sobre esta técnica se probaron diversas optimizaciones para medir su impacto en la solución obtenida.

Todas las versiones de vuelta atrás se han implementado mediante algoritmos recursivos. Éstos, necesitan que se construyan una serie de parámetros antes de proceder a su ejecución. En este caso los parámetros necesarios son tres: una lista de etiquetas, una solución vacía sobre la que trabajar y una solución inicial.

- **Lista de etiquetas**: Es necesario construir una lista con todas las etiquetas que se van a a asignar a los distintos vértices del grafo. Esta lista debe permitir el acceso individual a las distintas etiquetas mediante su posición dentro de la lista. Por ejemplo, en la lista [1,2,5,7] la etiqueta 7 debe de ser accesible mediante la posición 4 de la lista.

- **Solución vacía**: Sobre esta solución se irán añadiendo y eliminando las distintas etiquetas para generar las permutaciones que dan lugar a cada una de las soluciones.

- **Solución inicial**: Esta solución se utilizará para tener una solución de referencia a la hora de decidir si la nueva solución encontrada por el algoritmo es mejor que la mejor solución encontrada hasta el momento. Para generar esta solución se utilizan

tres métodos: un etiquetado lexicográfico del grafo, un etiquetado aleatorio y un etiquetado mediante una técnica heurística.

- **Etiquetado lexicográfico**: Es aquel etiquetado en el cual se le asigna al vértice V_i del grafo la etiqueta i. Por ejemplo: $V_1 \leftarrow 1, V_2 \leftarrow 2, \ldots, V_N \leftarrow N$
- **Etiquetado aleatorio**: Mediante este método se van seleccionando etiquetas aleatoriamente del conjunto de etiquetas disponibles y se asignan a los distintos vértices del grafo en orden lexicográfico. Es decir se selecciona una etiqueta aleatoriamente y se asigna al vértice V_1. A continuación, se obtiene otra etiqueta aleatoriamente y se asigna al vértice V_2, y así sucesivamente hasta que todos los vértices son etiquetados.
- **Etiquetado heurístico**: Este método consta de dos partes:
  1. Etiquetado del vértice de mayor grado: Se asignan etiquetas tanto al vértice de mayor grado como a los vértices adyacentes a este maximizando la diferencia mínima de etiquetas.
  2. Etiquetado aleatorio de los vértices restantes: El resto de etiquetas son asignadas de forma aleatoria a los vértices que no tengan ninguna etiqueta asignada. Este proceso se realiza diez mil veces y se elige el que genere un AB mayor de entre todos.

Una vez obtenida una solución inicial se calcula el valor del AB y se asigna como mejor solución encontrada hasta el momento.

A continuación se expondrán las distintas versiones del algoritmo de vuelta atrás que se han implementado para la resolución del problema.

3.3.1. Vuelta atrás básico

Esta técnica, parte de una solución vacía y añade etiquetas a la solución hasta que no haya más etiquetas disponibles, momento en el que se ha alcanzado un nodo hoja. En los nodos hoja se calcula el valor del AB del grafo. Si el valor calculado es mejor que el valor de la mejor solución encontrada hasta el momento, se actualiza el valor de la mejor solución. Una vez que se ha terminado de evaluar la solución, se vuelve en la recursividad para generar las permutaciones restantes y evaluar las nuevas soluciones encontradas. Ver Algoritmo 3.1.

```
Require: solucion = ∅ {Etiquetas asignadas a la solución}
Require: etiquetas ← 1..N {Lista de etiquetas disponibles}
Require: mejorSolucion ← 1, 2, .., N {Solución inicial trivial}
  procedure AB(solucion, etiquetas)
  begin
  if etiquetas = ∅ then
    calcularAB(solucion)
    if valorAB(solucion) > valorAB(mejorSolucion) then
      mejorSolucion ← solucion
    end if
  else
    FIN = numeroElementos(etiquetas)
    for i = 1 to FIN do
      l ← etiquetas[i]
      solucion ← l
      AB(solucion, etiquetas) {LLamada recursiva}
      quitar(solucion, l)
      etiquetas[i] ← l
    end for
  end if
  end procedure
```

Algoritmo 3.1: Backtracking básico

Optimizaciones

Al ser la primera implementación realizada, las estructuras de datos utilizadas fueron las menos costosas de implementar y, por ello, se optó por utilizar una matriz de adyacencia para modelar el grafo. Esta matriz tiene tantas filas y columnas como vértices tiene el grafo que modela. Almacena un 1 en la coordenada (A,B) si el vértice A es adyacente al vértice B y, por tratarse de grafos no dirigidos, también almacena un 1 en la coordenada (B,A). En aquellas coordenadas que representan vértices no adyacentes se almacena un 0. En la Figura 3.2 se puede observar la matriz de adyacencia correspondiente al grafo de la Figura 1.2.

Vértice \ Ady	A	B	C	D	E	F	G	H
A	0	1	0	0	1	0	0	0
B	1	0	0	0	0	1	0	0
C	0	0	0	0	0	1	0	0
D	0	0	0	0	0	0	1	0
E	1	0	0	0	0	0	0	1
F	0	1	1	0	0	0	1	1
G	0	0	0	1	0	1	0	0
H	0	0	0	1	1	0	0	0

Figura 3.2: Matriz de adyacencia del grafo de la Figura 1.2

Este tipo de estructura es eficiente para decidir si dos vértices son adyacentes entre sí, pero deficiente para generar una lista de adyacentes a un grafo dado, ya que es necesario recorrer completamente la fila correspondiente al vértice del cual se desean obtener los adyacentes, e ir añadiéndolos a una lista. Además se desaprovecha memoria.

Para calcular el AB de un vértice, es necesario obtener la lista de adyacentes a éste, para poder realizar la diferencia entre la etiqueta del vértice evaluado y las etiquetas de los vértices adyacentes al mismo. El tener que generar una lista de adyacentes a partir de la matriz de adyacencia repercute negativamente en el tiempo de cálculo del valor del AB, puesto que hay que recorrer la fila correspondiente al vértice del cual se desean obtener los adyacentes y añadirlos a una lista. Tener que generar la misma lista cada vez que se calcula el valor de la solución es malgastar tiempo de procesamiento.

Por ello, en la siguiente versión, se reemplazó la matriz de adyacencia por una tabla de listas de adyacencia (Figura 3.3) en la cual el índice de la tabla representa un vértice, y la lista asociada a ese índice los adyacentes a ese vértice. Esto aceleró el proceso de cálculo del valor del AB, ya que permitió acceder rápidamente a la lista de adyacentes de un vértice sin tener que generarla.

A	$\longrightarrow B \longrightarrow E$
B	$\longrightarrow B \longrightarrow F$
C	$\longrightarrow F$
D	$\longrightarrow G$
E	$\longrightarrow A \longrightarrow H$
F	$\longrightarrow B \longrightarrow C \longrightarrow G \longrightarrow H$
G	$\longrightarrow D \longrightarrow F$
H	$\longrightarrow E \longrightarrow F$

Figura 3.3: Tabla de listas de adyacencias del grafo de la Figura 1.2

Como medida adicional se mejoró la función encargada de calcular el AB de una solución completa. En un primer momento esta función calculaba todas las diferencias de etiquetas de cada vértice con sus adyacentes, elegía el valor más pequeño y lo asignaba como AB del vértice. Una vez calculado el AB de todos los vértices elegía el menor y lo asignaba como AB del grafo. Sin embargo, ésta no es la mejor solución. El valor mínimo del AB que puede obtener un vértice es 1, puesto que no se puede asignar la misma etiqueta a dos vértices distintos y, sin embargo, sí se puede asignar a dos vértices adyacentes dos etiquetas consecutivas y, por tanto, el valor del AB del grafo sería 1. Si durante el cálculo del valor del AB de un vértice alguna diferencia de etiquetas es igual a 1, se puede parar el proceso de cálculo del AB del grafo y asignar el valor 1 a esa solución.

Este mecanismo se puede mejorar un poco más. Para ello es necesario conocer el valor del AB de la mejor solución encontrada hasta el momento. En este caso, si mientras se

calcula el AB de un vértice, alguna diferencia de etiquetas es menor o igual que el valor del AB de la mejor solución encontrada hasta el momento, se puede parar el proceso de cálculo, puesto que el valor del AB de esa solución nunca sería mejor que el valor del AB de la mejor solución encontrada hasta el momento y, por lo tanto, continuar con el cálculo del AB para esa solución sería malgastar tiempo.

3.3.2. Vuelta atrás con poda

La versión básica del algoritmo de vuelta atrás tiene el inconveniente de que, para poder evaluar una solución, es necesario llegar a un nodo hoja en el árbol de exploración. El problema surge cuando dos vértices adyacentes que se etiquetan en la parte alta del árbol de exploración, generan un AB menor que el de la mejor solución encontrada hasta el momento. Al evaluar el valor del AB sólo en los nodos hoja del árbol de exploración es condición necesaria el etiquetar el resto de vértices del grafo, perdiendo el tiempo en generar una solución que nunca, por tratarse de un problema de maximización de mínimos, va a ser mejor que la mejor solución encontrada hasta el momento.

Para paliar el problema anteriormente descrito se introdujeron dos mejoras: factorizar el cálculo de la solución y aplicar un criterio de poda.

- **Factorización del cálculo de la solución**: Como se describió en la Sección 3.2, mediante la factorización se puede ir evaluando una solución a medida que va siendo construida. En el caso del cálculo del valor del AB de una solución, como ya se comentó en la sección 1.2.1, son necesarios dos pasos para calcularlo: primero se calcula el AB de cada vértice y luego se elige el valor mínimo de éstos. Así, se puede ir calculando el AB de los distintos vértices a medida que se va construyendo la solución y, una vez construida una solución completa, elegir el valor mínimo.
  El modo de factorizar el cálculo del AB es el siguiente: cuando una etiqueta es asignada a un vértice, sólo un número limitado de vértices (los adyacentes al vértice etiquetado) se ven afectados por este cambio. Puesto que esta solución no está completa (no todos los vértices tienen una etiqueta asignada) para poder calcular el AB del vértice etiquetado, sólo se tienen en cuenta aquellos adyacentes que ya hayan sido etiquetados previamente. Este valor es el AB parcial de un vértice, siendo necesario volver a calcular el AB parcial del sus vértices adyacentes, puesto que el valor del AB de cada uno de ellos se ve afectado por la inclusión de la nueva etiqueta.

- **Aplicación de un criterio de poda**: Mediante la factorización anterior, se puede observar que cada vez que un vértice es etiquetado, el valor del AB parcial de los vértices se actualiza y se puede obtener un AB parcial del grafo solamente con

calcular el mínimo de los AB parciales de los vértices. Esto es muy útil cuando se aplica un criterio de poda, ya que permite 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 haciendo uso de la factorización es la siguiente: cada vez que una etiqueta es asignada a un vértice, se calcula el AB parcial del grafo y, si ese AB 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. Ver algoritmo 3.2.

```
Require: solucion = ∅ {Etiquetas asignadas a la solución}
Require: etiquetas ← 1..N {Lista de etiquetas disponibles}
Require: mejorSolucion ← 1, 2, .., N {Solución inicial trivial}
  procedure AB(solucion, etiquetas)
  begin
  if etiquetas = ∅ then
    minimoAB(solucion)
    if valorAB(solucion) > valorAB(mejorSolucion) then
      mejorSolucion ← solucion
    end if
  else
    FIN = numeroElementos(etiquetas)
    for i = 1 to FIN do
      l ← etiquetas[i]
      solucion ← l
      calcABVertice(ultimoVerticeEtiq(solucion))
      if valorParcialAB(solucion) > valorAB(mejorSolucion) then
        AB(solucion, etiquetas) {LLamada recursiva}
      end if
      quitar(solucion, l)
      etiquetas[i] ← l
    end for
  end if
  end procedure
```

Algoritmo 3.2: Backtracking con poda

Cotas propuestas

Hasta el momento, el único criterio empleado para podar ramas del árbol de exploración era el uso del AB parcial obtenido mediante la factorización del cálculo de la solución. El AB parcial se basa en las etiquetas que han sido asignadas a los vértices del grafo por el algoritmo. Sin embargo, existen otros criterios que permiten estimar el mejor valor del

AB que una solución parcial es capaz de obtener en función de las etiquetas que aún no han sido asignadas. Esta estimación, del mejor valor que un nodo del árbol de exploración puede obtener, se denomina cota.

Para calcular la cota de una solución parcial se calculará el mínimo entre el AB parcial de esa solución y la estimación del mejor valor del AB que esa solución es capaz de obtener:

$$cota = min\{AB_{parcial},\ estimación\}$$

Como se describió en la Sección 3.2, las cotas generadas se basan en el etiquetado del vértice de mayor grado. Previamente se analizaron las soluciones óptimas que los algoritmos anteriores encontraban y, en todas ellas, la tendencia que se observaba era que, el valor del AB del vértice o vértices de mayor grado, marcaban el valor del AB del grafo completo. Por ello se planteó una cota basada en el etiquetado del vértice de mayor grado.

Para poder aplicar este método de estimación, es necesario tratar a los grafos de una manera especial. Se debe tratar tanto al vértice de mayor grado, como a los vértices adyacentes a éste, como si fueran un grafo independiente, desestimando las adyacencias que pudieran tener entre sí los vértices adyacentes al de mayor grado. En estos vértices se conservan las etiquetas que hubieran sido previamente asignadas por el algoritmo. En la Figura 3.4 se observa un ejemplo de la selección de los vértices que intervienen en el cálculo de la cota para el grafo de la Figura 1.2.

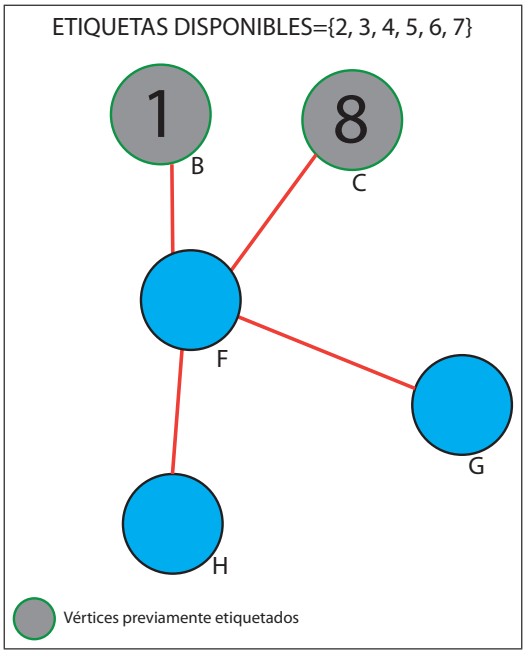

Figura 3.4: Vértices que calculan la cota de grafo de la Figura 1.2

Una vez aislados los vértices que intervienen en el cálculo de la cota, se asignan, a los vértices que estén libres, las etiquetas disponibles que maximicen el valor del AB

del vértice de mayor grado. Este etiquetado no es realmente efectivo y tanto la solución parcial, como las etiquetas disponibles, no se ven modificadas por el cálculo de la cota. Si hubiera varios vértices con el grado máximo se repetiría el proceso para cada uno de ellos y se asignaría como cota el menor valor obtenido.
El hecho de que las cotas generadas se basen en el etiquetado del vértice de mayor grado hace que éstas sean optimistas. Esto quiere decir que el valor del AB de una solución final, será siempre menor o igual al valor de la cota en cualquier nodo de la solución parcial.

El objetivo de la función de cota es estimar el mejor etiquetado posible sobre el vértice de mayor grado del grafo. Por ello hay que tener en cuenta las distintas situaciones en las que podrían encontrarse los vértices implicados en el cálculo de la cota. Para poder aplicar un método que maximice el valor de la estimación, hay que hacer un pequeño estudio de las etiquetas que ya han sido asignadas. Esto significa que no todos los métodos obtienen el mejor AB en todos los casos, y que hay que elegir el correcto para poder calcular la cota.

Los métodos que se utilizan para realizar el etiquetado de este conjunto de vértices son los tres siguientes: etiquetado de extremos, etiquetado del valor medio y mejor AB posible.

1. **Etiquetado de extremos**: Este etiquetado se basa en la distribución de etiquetas lo más dispares posible y se realiza de la siguiente forma: 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: $N, N-1, N-2, \ldots$, 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 3.5 se puede observar un ejemplo de etiquetado por extremos sobre el vértice de mayor grado del grafo de la Figura 1.2.

   Aunque en un principio pueda parecer que se puede obtener el mismo AB etiquetando el vértice de mayor grado con el valor más pequeño y el resto con los más grandes, y que mediante el etiquetado inverso, en ciertas ocasiones no es así. Esto ocurre cuando no todas las etiquetas están disponibles. En el ejemplo de la Figura 3.5, si la etiqueta 7 no estuviera disponible, por estar previamente asignada a otro vértice, se tendrían que usar las siguientes etiquetas: 1 para el vértice de mayor grado y $\{4, 5, 6, 8\}$ para los adyacentes obteniendo un AB de 3. Sin embargo, si se

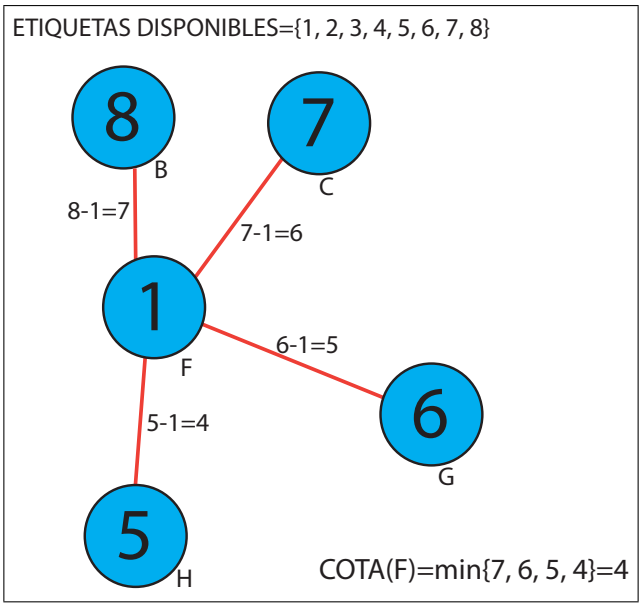

Figura 3.5: Cálculo de cota por etiquetado de extremos

realizase el etiquetado inverso como se muestra en la Figura 3.6 se obtendría un AB de 4.

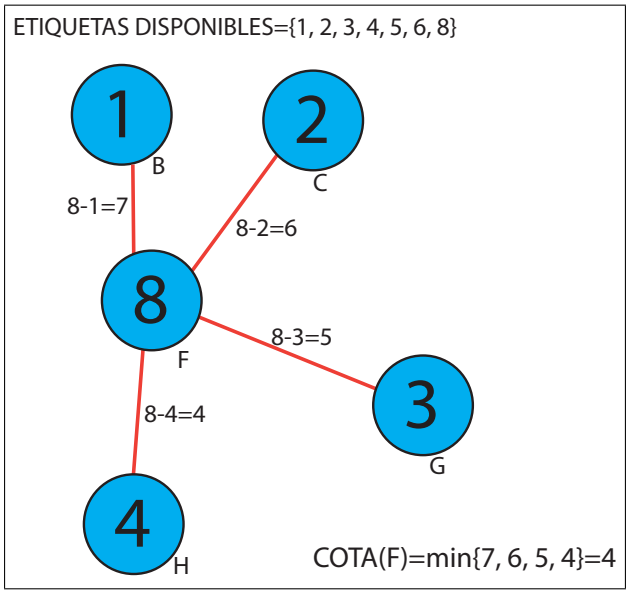

Figura 3.6: Cálculo de cota por etiquetado de extremos

2. **Etiquetado del valor medio**: Este etiquetado se basa en la distribución de las etiquetas de la siguiente forma: el vértice de mayor grado es etiquetado con la etiqueta de valor medio en el rango $[1..N]$, es decir con $\frac{N}{2}$, 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 AB y se

asigna como cota. En la Figura 3.7 se muestra un ejemplo del etiquetado del valor medio en cual las etiquetas 1 y 8 habían sido previamente asignadas a los vértices B y C respectivamente.

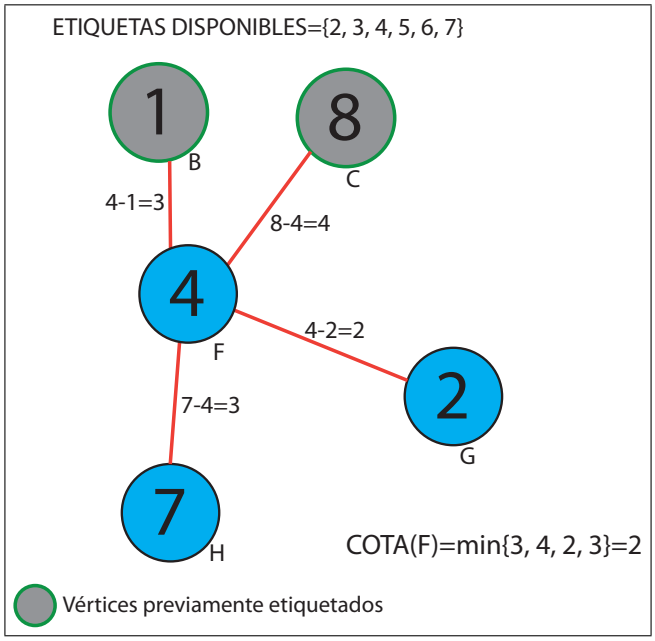

Figura 3.7: Cálculo de cota por etiquetado del valor medio

3. **Selección del mejor AB posible**: Este método necesita que el vértice de mayor grado tenga una etiqueta previamente asignada por el algoritmo. Consta de tres etapas.

   - En primer lugar se obtiene el AB parcial del vértice de mayor grado. Se calcula mediante las etiquetas previamente asignadas por el algoritmo.
   - En la segunda etapa se etiquetan los vértices adyacentes al de mayor grado con aquellas etiquetas que generen un AB igual o mayor al que se obtuvo en el primer paso. Básicamente consiste en etiquetar los vértices adyacentes de manera que no se empeore el AB parcial del vértice.
   - Por último, en caso de que aún queden vértices sin etiquetar, porque no hubiera suficientes etiquetas que no empeorasen el AB parcial del vértice, se eligen aquellas etiquetas que menos empeoren el AB parcial del vértice de mayor grado.

   Tras estos tres pasos se puede garantizar que el vértice de mayor grado ha obtenido el mejor AB posible con las etiquetas que estaban disponibles.

A la hora de realizar la estimación del mejor valor del AB que podría obtener una solución parcial, las distintas situaciones que pueden darse son las siguientes:

- **Todos los vértices etiquetados**: En el supuesto de que todos los vértices implicados en el cálculo de la cota hubieran sido previamente etiquetados por el algoritmo, se procederá al cálculo del AB de ese vértice y se asignará como cota. En este caso dicho valor sería menor o igual que el AB parcial.

- **Ningún vértice etiquetado**: Cuando tanto el vértice de mayor grado, como los vértices adyacentes a éste no tienen ninguna etiqueta asignada, el mejor etiquetado posible se puede realizar mediante la técnica del etiquetado de extremos. En este caso hay que realizar las dos variantes de este método, puesto que dependiendo de las etiquetas disponibles, pueden obtener valores distintos del AB.

- **Algunos vértices etiquetados**: Cuando entre los vértices sobre los cuales se calcula la cota, hay algunos que han sido previamente etiquetados por el algoritmo, hay que analizar el tipo de etiquetas que han sido asignadas para poder elegir el mejor método de los anteriormente descritos.

  Para clasificarlas, se comparan las etiquetas asignadas con la etiqueta de valor medio, es decir, con la etiqueta $\frac{N}{2}$ (siendo una división entera). Las etiquetas asignadas se dividen en aquellas que son mayores estrictamente que el valor medio (etiquetas altas) y las que son menores o iguales al valor medio (etiquetas bajas). En caso de que el vértice de mayor grado esté etiquetado, no se tiene en cuenta a la hora de contar las etiquetas altas y bajas.

  Una vez tenemos clasificadas las etiquetas, se elige el mejor modo de etiquetar los vértices restantes. Por lo general, el mejor modo de etiquetar los vértice es mediante el etiquetado del mejor AB posible. Sin embargo hay ocasiones en las cuales esta técnica no es la mejor.

  En el caso de que el único vértice etiquetado sea el de mayor grado, se utilizará el método de extremos, eligiendo una variante u otra en función de la etiqueta asignada a este vértice. Si el vértice central no está etiquetado y hay, tanto etiquetas altas, como bajas asignadas a los adyacentes, la mejor técnica es la del etiquetado del valor medio.

En el Algoritmo 3.3 puede verse el uso del cálculo de la cota para realizar podas en el árbol de exploración.

```
Require: solucion = ∅ {Etiquetas asignadas a la solución}
Require: etiquetas ← 1..N {Lista de etiquetas disponibles}
Require: mejorSolucion ← 1, 2, .., N {Solución inicial trivial}
  procedure AB(solucion, etiquetas)
  begin
  if etiquetas = ∅ then
    minimoAB(solucion)
    if valorAB(solucion) > valorAB(mejorSolucion) then
      mejorSolucion ← solucion
    end if
  else
    FIN = numeroElementos(etiquetas)
    for i = 1 to FIN do
      l ← etiquetas[i]
      solucion ← l
      calcABVertice(ultimoVerticeEtiq(solucion))
      calcularCota(solucion)
      mejorValor = valorAB(mejorSolucion)
      if (valorParcialAB(solucion) > mejorValor) and
         (valorCota(solucion) > mejorValor)
       then
         AB(solucion, etiquetas) {LLamada recursiva}
      end if
      quitar(solucion, l)
      etiquetas[i] ← l
    end for
  end if
  end procedure
```

Algoritmo 3.3: Backtracking con poda y estimación

3.3.3. Orden del etiquetado

En los algoritmos descritos hasta el momento, el etiquetado de un grafo se hace por orden lexicográfico, es decir, el primer vértice en etiquetarse es el número 1, luego se etiqueta el número 2, y así sucesivamente hasta etiquetar los N vértices que componen el grafo. Esta forma de etiquetar trata a todos los vértices por igual, sin embargo, aunque en principio no haya diferencias aparentes entre el AB de dos vértices distintos, el hecho de que un vértice tenga mayor grado que otro, lo hace más susceptible a generar un AB menor que un vértice de menor grado, simplemente por el hecho de que en él intervienen más etiquetas en el cálculo de su AB.

Por ello se realizó un etiquetado alternativo, que asignaba etiquetas a los vértices en función del grado de éstos. Para poder ordenar los vértices en base a su adyacencia, se implementó un algoritmo de ordenación, basado en el algoritmo de burbuja, llamado

shakesort. Se probó a ordenar tanto descendentemente, como ascendentemente los vértices del grafo. El orden descendente da prioridad a los vértices de mayor grado, mientras que el orden ascendente primero asigna etiquetas a los vértices de menor grado. En caso de que dos vértices tengan el mismo grado, se mantiene el orden lexicográfico.

Los experimentos realizados demostraron que la versión que etiquetaba los nodos de forma descendente encontraba la solución óptima de forma más rápida que la versión que empleaba el orden lexicográfico.

3.4. Ramificación y acotación (Branch and Bound)

Todos los algoritmos propuestos hasta el momento están basados en búsquedas en profundidad, e implementados recursivamente. Esto implica que el árbol de exploración es recorrido en función de cómo se etiqueten los vértices, ya sea mediante el orden lexicográfico, o en base a la adyacencia de los vértices tal y como se ha descrito en la Sección 3.3.3.

El hecho de que la exploración se haga de forma recursiva implica que para pasar de una rama del árbol de exploración a otra, es necesario deshacer todas las llamadas recursivas y, por lo tanto, esa rama debe haber sido explorada por completo.

Con el fin de realizar una exploración más eficiente del espacio de soluciones, se desarrolló un algoritmo iterativo capaz de ramificar, en cada paso, el nodo más prometedor del árbol de exploración. La forma de decidir qué nodo es el más prometedor en cada momento se realiza mediante la cota expuesta en la sección 3.3.2, el valor del AB de una solución parcial y la profundidad (número de vértices etiquetados) del nodo. Mediante estos tres valores, se genera una prioridad para cada nodo susceptible de ser ramificado. La prioridad se organiza en dos niveles:

- **Primer nivel**: Para este nivel se calcula el valor mínimo entre la cota y el valor del AB. Cuanto mayor sea este valor, mayor prioridad tendrá el nodo. Este es el valor de la prioridad.

- **Segundo nivel**: La prioridad de segundo nivel se utiliza entre nodos que tengan la misma prioridad de primer nivel. A igual valor tendrá mayor prioridad el nodo que tenga más vértices etiquetados, es decir, aquel nodo más profundo en el árbol.

Una vez calculada la prioridad de un nodo es necesario decidir si ese nodo es prometedor o no. Para ello se compara el valor de la prioridad con el valor del AB de la mejor solución encontrada hasta el momento. Sólo si la prioridad es estrictamente mayor, se almacena para su posterior ramificación.

Los nodos que se almacenan deben contener todos los datos necesarios para poder ser ramificados una vez sean elegidos como nodo más prometedor. Para poder continuar con la ejecución, un nodo debe de almacenar la lista de etiquetas que no han sido asignadas a ningún vértice y la solución parcial construida hasta el momento. A partir de estos datos se puede ramificar y calcular, tanto las prioridades, como la estimación del mejor valor del AB del nodo.

3.4.1. Cola de prioridad multinivel

Para poder almacenar los nodos, es necesario una estructura de datos capaz de aplicar los dos criterios de prioridad. Esta estructura es una cola de prioridad multinivel. En vez de almacenar todos los nodos en una única cola de prioridad, éstos se almacenan en distintas colas. Estas colas son accesibles a través de una tabla, cuyo índice representa el valor de la prioridad del nodo. Una vez dentro de la cola específica de cada prioridad, los nodos son ordenados en función del número de vértices etiquetados. Si dos nodos tienen la misma prioridad y el mismo número de vértices etiquetados, se utiliza el orden *FIFO*.

Cuando el algoritmo accede a la cola para recuperar el nodo más prometedor, es el nodo de mayor prioridad y con mayor número de vértices etiquetados, el que es recuperado. En el ejemplo de la Figura 3.8, el nodo de mayor prioridad de todos sería el *nodo 9*.

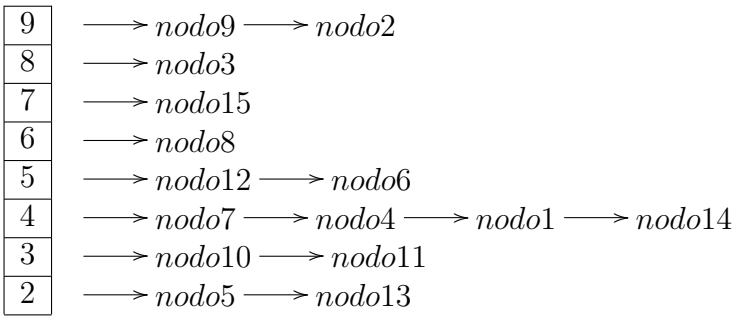

Figura 3.8: Cola de prioridad multinivel

3.4.2. Implementación iterativa

El objetivo de este algoritmo es poder elegir en cada momento la mejor rama a explorar, por lo que el algoritmo que se utiliza debe ser implementado de forma iterativa. Este algoritmo iterativo utiliza la cola de prioridad descrita en la Sección 3.4.1, para elegir, en cada iteración, cuál es el nodo más prometedor a expandir, si no se trata de una solución completa al problema.

Para poder ejecutar el algoritmo, primero deben generarse todos los nodos de profundidad dos, y añadirlos a la cola de prioridad. Esto es necesario porque el algoritmo

detiene su ejecución cuando ya no quedan elementos en la cola de prioridad, dando por sentado que ha terminado de procesar todos los nodos prometedores. El pseudocódigo de este algoritmo puede verse en el Algoritmo 3.4.

3.4.3. Variación en el recorrido del árbol de exploración en algoritmo de ramificación y acotación

Aunque a priori el recorrido lógico del árbol de exploración es mediante la exploración en cada paso del nodo más prometedor, esto puede derivar en una serie de problemas. Uno de los problemas que pueden darse es que el tamaño del árbol de exploración puede desbordar la pila de ejecución del programa como se describe en la Sección 3.5.1.
Otro de los problemas derivados de este recorrido es que todo el tiempo se invierta en ramificar los nodos más prometedores, y nunca se llegue a alcanzar los nodos hoja. Esto hace que, al no generar nuevas soluciones, nunca se actualice la mejor solución temporal, y se siga realizando las podas mediante el valor de la solución inicial.

Para solucionar esto, se puede realizar un recorrido inverso del árbol de exploración. Este recorrido prioriza los nodos de menor prioridad (normalmente más profundos en el árbol de exploración), haciendo que se alcance un mayor número de nodos hoja, y obteniendo soluciones mejores rápidamente. Esto conlleva que constantemente se descarten nodos y se realicen muchas podas manteniendo una cola de tamaño reducido que permite ahorrar tiempo en el volcado a disco. Este recorrido inverso también tiene sus inconvenientes. Puede darse el caso de que, llegado un punto, el algoritmo sólo se dedique a podar los nodos de menor prioridad, y nunca llegue a ramificar los nodos más prometedores, haciendo que la diferencia entre el valor de la mejor solución obtenida y el valor del nodo de mayor prioridad de la cola sea demasiado alta.

3.5. Problemas encontrados

Durante el desarrollo de los distintos algoritmos exactos para la resolución del problema del AB, se encontraron una serie de problemas que serán descritos a continuación.

3.5.1. Tamaño del árbol de exploración

Dadas las características del problema: existen N! etiquetados posibles para un grafo, siendo N el número de vértices del mismo. A este número, hay que añadir que la cantidad de nodos que componen las capas intermedias del árbol de exploración es muy grande. El número de nodos que debe explorar un algoritmo exacto para resolver este problema se puede calcular mediante la siguiente expresión matemática:

$$\sum_{i=0}^{N} \frac{N!}{(N-i)!}$$

A pesar de las podas de las distintas ramas del árbol de exploración, y el purgado de nodos que dejan de ser prometedores cada vez que se encuentra una solución que mejora el valor del AB de la mejor solución actual, la cola de prioridad necesita almacenar una gran cantidad de nodos en memoria. Tal es la cantidad que puede llegar a desbordar la pila de ejecución del programa. Por ese motivo es necesario un mecanismo que sea capaz de trasvasar información desde la memoria principal hasta las unidades de almacenamiento secundario para su posterior recuperación.

Para este fin se implementó, en la cola de prioridad, un mecanismo de volcado a disco. Este volcado se realiza cuando la ocupación de pila de memoria del programa alcanza más del 80 % de su capacidad. En ese momento, la cola vuelca automáticamente el 20 % de su contenido. Este 20 % se compone de los nodos menos prometedores que, en el momento del volcado, pueblan la cola de prioridad. Estos nodos no pueden ser desechados, ya que en algún momento pueden llegar a ser los nodos más prometedores y necesitar de su recuperación desde el almacenamiento secundario, aunque también es probable que estos nodos almacenados sean los primeros en ser podados por su condición de ser los menos prometedores de todos. Por este motivo se mantiene una lista, que contiene los nodos que siguen siendo objeto de una posible recuperación por parte de el algoritmo. Si los nodos que se encuentran almacenados son podados, serán eliminados de esta lista de posibles recuperaciones. Cuando el programa finaliza su ejecución, se eliminan, del almacenamiento secundario, todos aquellos nodos que finalmente no fueron recuperados por el algoritmo.

3.5.2. Mantenimiento de la cola

Dado que el manejo de la cola es muy complejo y la capacidad limitada, es necesario mantener en memoria de ejecución sólo los nodos necesarios. Esto se realiza mediante una purga constante de los nodos que todavía son susceptibles de ser ramificados, tanto en memoria, como en almacenamiento secundario. Cada vez que una nueva solución genera un AB que es mayor que el de la mejor solución encontrada hasta el momento, se purgan todos aquellos elementos que tienen una proyección menor o igual al valor del AB de la nueva mejor solución encontrada. Sólo se dejan en la cola aquellos elementos que puedan mejorar el AB actual, porque no merece la pena explorar nodos que vayan a obtener un valor igual al mejor encontrado hasta el momento, puesto que todas las soluciones son igualmente válidas.

```
Require: cola = ∅
Require: mejorValor = valorAB(solucionTrivial)
  {Generación de los nodos de profundidad 2}
  N = numeroVertices(Grafo)
  for i = 1 to N do
    tempSol = ∅
    tempSol ← i
    etiquetasLibres ← {1..N} − {i}
    for j = 1 to N − 1 do
      auxSol = tempSol
      l ← etiquetasLibres[j]
      auxSol ← l
      auxLista = etiquetasLibres − {l}
      cola ← nuevoNodo(auxSol, auxLista)
    end for
  end for

  {Exploración de los nodos más prometedores}
  while cola ≠ ∅ do
    nodoAux = primero(cola)
    if prioridad(nodoAux) > mejorValor then
      tempSol = obtenerSolucion(nodoAux)
      etiquetasLibres = obtenerEtiquetas(nodoAux)
      N = numElementos(etiquetasLibres)
      for t = 1 to N do
        auxSol = tempSol
        l ← etiquetasLibres[t]
        auxSol ← l
        auxLista = etiquetasLibres − {l}
        if auxLista == ∅ then
          calcularAB(auxSol)
          if valorAB(auxSol) > mejorValor then
            mejorSolucion = auxSol
            mejorValor = valorAB(auxSol)
          end if
        else
          calcABVertice(ultimoVerticeEtiq(auxSol))
          calcularCota(auxSol)
          if (valorParcialAB(auxSol) > mejorValor) and
             (valorCota(auxSol) > mejorValor)
           then
            cola ← nuevoNodo(auxSol, auxLista)
          end if
        end if
      end for
    end if
  end while
  {Fin del algoritmo}
  return mejorSolucion
```

Algoritmo 3.4: Ramificación y acotación iterativo

Capítulo 4

Descripción Informática

Cuando se desarrolla un proyecto informático de cierta envergadura, es necesario utilizar un patrón que proporcione al programador una guía adecuada para estructurar, planear y controlar el proceso de desarrollo. Esto es lo que se conoce como una metodología de desarrollo *software*. Durante este capítulo se hará una pequeña introducción a las metodologías *software* más relevantes y se describirá en detalle la empleada para la elaboración de este PFC, así como su adaptación al mismo.

4.1. Introducción a las metodologías software más importantes

Entre las diversas metodologías de desarrollo *software* existentes, algunas de las más conocidas son las siguientes:

- **Incremental [12]**: Esta metodología provee una estrategia para controlar la complejidad y los riesgos, desarrollando una parte del producto *software* y reservando el resto de aspectos para el futuro.

- **Espiral [12]**: El modelo en espiral tiene en cuenta fuertemente el riesgo que aparece a la hora de desarrollar *software*. Para ello, se comienza mirando las posibles alternativas de desarrollo, se opta por la de riesgo más asumible y se realiza un ciclo de la espiral. En cada ciclo, si el cliente quiere seguir haciendo mejoras en el *software*, se vuelven a evaluar las distintas nuevas alternativas y riesgos y se realiza otra vuelta de la espiral, así hasta que llegue un momento en el que el producto *software* desarrollado sea aceptado y no necesite seguir mejorándose con otro nuevo ciclo.

- **Proceso Unificado de Desarrollo (PUD) [11]**: Es un marco de desarrollo de *software* que se caracteriza por estar dirigido por casos de uso, centrado en la arqui-

tectura y por ser iterativo e incremental, que puede ser adaptado a organizaciones o proyectos específicos. El PUD está compuesto de cuatro fases denominadas Inicio, Elaboración, Construcción y Transición. Cada una de estas fases es a su vez dividida en una serie de iteraciones. Estas iteraciones ofrecen como resultado un incremento del producto desarrollado que añade o mejora las funcionalidades del sistema en desarrollo.

- **Programación Extrema (eXtreme Programming - XP) [3]**: Es el más destacado de los procesos ágiles de desarrollo de *software*. Esta metodología promueve el desarrollo de código durante todo el ciclo del vida del proyecto, siendo capaz de hacer frente a los imprevistos que puedan surgir durante el mismo.

Las metodologías clásicas como el modelo incremental, el modelo en espiral o el PUD intentan definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos, mientras que las metodologías ágiles como XP ponen más énfasis en la adaptabilidad frente a los cambios que se producen durante el proceso de desarrollo del proyecto.

La metodología de XP considera que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos y que, ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que el enfoque de la metodologías clásicas.

4.2. Metodología empleada: Programación Extrema

Para el desarrollo de este PFC se ha utilizado la metodología de la Programación Extrema. Se optó por este enfoque por ser una metodología capaz de adaptarse fácilmente a los distintos imprevistos que pudieran surgir durante el desarrollo del mismo. A continuación se realizará una descripción de la misma.

4.2.1. Breve introducción a la Programación Extrema

La idea de la XP consiste en trabajar estrechamente con el cliente, realizando pequeñas versiones del proyecto con mucha frecuencia (cada dos semanas). En cada pequeña versión se debe hacer el mínimo de código y lo más simple posible para que funcione correctamente. El diseño se hace sobre la marcha, haciendo un pequeño diseño para la primera versión y luego modificándolo en las siguientes versiones. Además, no hay que hacer una documentación para el diseño, no hay mejor documentación que el mismo código. El código, por tanto, también se modifica continuamente de versión en versión, añadiéndole

funcionalidad y extrayendo sus partes comunes.

4.2.2. Principios de la Programación Extrema

La XP esta basada en los siguientes principios:

- **Simplicidad**: Se realizará sólo lo necesario y nada más. Se realizarán pequeños incrementos en el desarrollo y se solventarán los fallos según vayan apareciendo. Esto facilitará el mantenimiento a largo plazo.
- **Comunicación**: Todo el mundo es parte del equipo y es necesaria una comunicación diaria entre todos los miembros. Todo el equipo trabajará junto desde los requisitos iniciales hasta la codificación, en busca de la mejor solución posible.
- **Retroalimentación**: Al estar el cliente integrado en el proyecto, su opinión sobre el estado del proyecto se conoce en tiempo real. Al realizarse ciclos muy cortos tras los cuales se muestran resultados, se minimiza el tener que rehacer partes que no cumplen con los requisitos y ayuda a los programadores a centrarse en lo que es más importante.
- **Respeto**: Todo el mundo debe sentirse valorado como un miembro del equipo. Los desarrolladores deben valorar la experiencia del cliente y viceversa.
- **Coraje**: Es necesario ser sincero con el progreso realizado y con las estimaciones de tiempo. Hay que adaptarse a los cambios cuando estos aparezcan.

4.2.3. Prácticas básicas de la Programación Extrema

Para que la XP funcione, se describen trece prácticas basadas en los principios anteriores:

- **Versiones pequeñas**: Las versiones deben ser lo suficientemente pequeñas como para poder hacer una cada pocas semanas. Deben ser versiones que ofrezcan algo útil al usuario final y no trozos de código que no pueda ver funcionando.
- **Equipo completo**: Forman parte del equipo todas las personas que tienen algo que ver con el proyecto, incluido el cliente y el responsable del proyecto.
- **Planificación**: Se hacen las historias de usuario y se planifica en qué orden se van a hacer, así como las versiones. La planificación se revisa continuamente.
- **Test del cliente**: El cliente, con la ayuda de los desarrolladores, propone sus propias pruebas para validar las versiones.

- **Diseño simple**: Hacer siempre lo mínimo imprescindible de la forma más sencilla posible. Mantener siempre sencillo el código.

- **Pareja de programadores**: Los programadores trabajan por parejas (dos delante del mismo ordenador) y se intercambian las parejas con frecuencia (un cambio diario).

- **Desarrollo guiado por las pruebas automáticas**: Se deben realizar programas de prueba automática y deben ejecutarse con mucha frecuencia. Cuantas más pruebas se hagan, mejor.

- **Mejora del diseño**: Mientras se codifica, debe mejorarse el código ya hecho con el que nos crucemos y que sea susceptible de ser mejorado. Extraer funcionalidades comunes, eliminar líneas de código innecesarias, etc.

- **Integración continua**: Debe tenerse siempre un ejecutable del proyecto que funcione y, en cuanto se tenga una nueva pequeña funcionalidad, debe recompilarse y probarse. Es un error mantener una versión congelada dos meses mientras se hacen mejoras y luego integrarlas todas de golpe. Cuando falle algo, no se sabe qué es lo que falla de todo lo que se ha introducido.

- **El código es de todos**: Cualquiera puede y debe tocar y conocer cualquier parte del código. Para eso se hacen las pruebas automáticas.

- **Normas de codificación**: Debe haber un estilo común de codificación (no importa cuál) de forma que parezca que ha sido realizado por una única persona.

- **Metáforas**: Hay que buscar unas frases o nombres que definan cómo funcionan las distintas partes del programa, de forma que sólo con los nombres se pueda uno hacer una idea de qué es lo que hace cada parte del programa. Un ejemplo claro es el recolector de basura de Java. Ayuda a que todos los programadores (y el cliente) sepan de qué estamos hablando y a que no haya malentendidos.

- **Ritmo sostenible**: Se debe trabajar a un ritmo que se pueda mantener indefinidamente. Esto quiere decir que no debe haber días muertos en que no se sabe qué hacer y que no se deben hacer un exceso de horas otros días. Al tener claro semana a semana lo que debe hacerse, hay que trabajar duro en ello para conseguir el objetivo cercano de terminar una historia de usuario o versión.

4.3. Adaptación de la metodología empleada en este PFC

A continuación se expondrán algunas adaptaciones de la metodología de XP a este PFC:

- **Equipo completo**: En este caso el equipo se ve formado tanto por los tutores del proyecto como por el alumno responsable de su realización.
- **Planificación**: En las distintas reuniones se proponían metas que marcaban cada versión y, una vez alcanzada la nueva versión, se volvía a reunir el equipo para planificar la siguiente etapa. Las metas más importantes que se marcaron fueron la implementación de distintas técnicas algorítmicas capaces de resolver el problema del AB tales como: vuelta atrás, vuelta atrás con poda y ramificación y acotación.
- **Test del cliente**: En este caso el cliente está representado por los tutores del proyecto. Estos propusieron conjunto de problemas, denominado *SmallInstances* [9], que los distintos algoritmos tenían que conseguir resolver.
- **Pareja de programadores**: En este caso, al ser un proyecto individual, no es aplicable.
- **Desarrollo guiado por las pruebas automáticas**: Cada vez que una versión se termina se ejecuta sobre un conjunto de pruebas, que permite medir el rendimiento de dicha versión.
- **Mejora del diseño**: Durante el desarrollo de este PFC este punto fue una constante. Continuamente se probaban nuevas soluciones y se medía el impacto en el rendimiento que ofrecían.
- **Metáforas**: Para el desarrollo de este PFC se utilizaron diversas metáforas para referirse a las distintas formas de etiquetado de los grafos tal y como se mostró en la Sección 3.3.2.

4.4. Planificación y seguimiento

A pesar de que en la XP no existe una planificación a largo plazo del proyecto y, cada versión tiene su propia planificación, se optó por realizar una primera planificación de cómo sería el desarrollo del proyecto a lo largo de todo su ciclo de vida, para poder compararlo con los plazos reales en la realización de este PFC.

En primer lugar se planearon las distintas etapas por las cuales debería pasar el proyecto, así como los distintos hitos a conseguir para que el proyecto estuviera terminado. Estas etapas son las descritas en el Capítulo 2. También se adaptó un diagrama de Gantt a los términos de la XP con el fin de tener una referencia de tiempo en la cual ubicar los distintos hitos a conseguir y tener un seguimiento del trabajo realizado y el tiempo empleado en el desarrollo de cada versión.

4.4.1. Diagrama de Gantt de planificación

El diagrama o gráfico Gantt es un herramienta útil en la gestión de proyectos, creada por Henry L. Gantt en 1917. Consiste en confeccionar un cuadro con todas las actividades o tareas, por orden de inicio, con los respectivos tiempos previstos para su realización e identificación de la actividad precedente, a partir del cual se calculan las fechas de inicio y finalización, y se realiza una representación gráfica horizontal del comienzo y duración de todas las tareas del proyecto.
Con este método se consigue una mayor eficiencia en la ejecución de proyectos, porque permite conseguir los siguientes objetivos:

- Minimizar los tiempos de espera, lo que implica una utilización óptima de los recursos.
- Reducir al máximo el incumplimiento de los plazos.
- Acortar el tiempo empleado en la ejecución global del proyecto.

En la Figura 4.1 puede verse el diagrama de Gantt utilizado para la planificación de este proyecto.

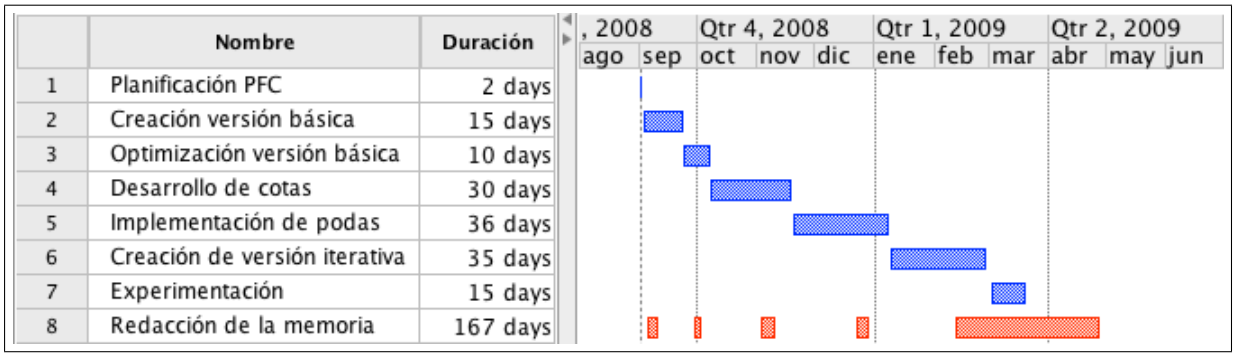

Figura 4.1: Diagrama de Gantt de planificación

4.4.2. Diagrama de Gantt de seguimiento

Este otro diagrama de Gantt es similar en concepto al anteriormente expuesto, es decir, una representación gráfica de la duración de las distintas tareas que componen el proyecto. Sin embargo, en este diagrama se registran los datos reales del tiempo invertido en cada tarea. Este tipo de gráficos ayuda a evaluar si la planificación inicial fue correcta o no, así como las tareas que han sido más difíciles de completar o que han llevado más tiempo del previsto inicialmente.

En la Figura 4.2 puede verse el diagrama de Gantt utilizado para el seguimiento de este proyecto.

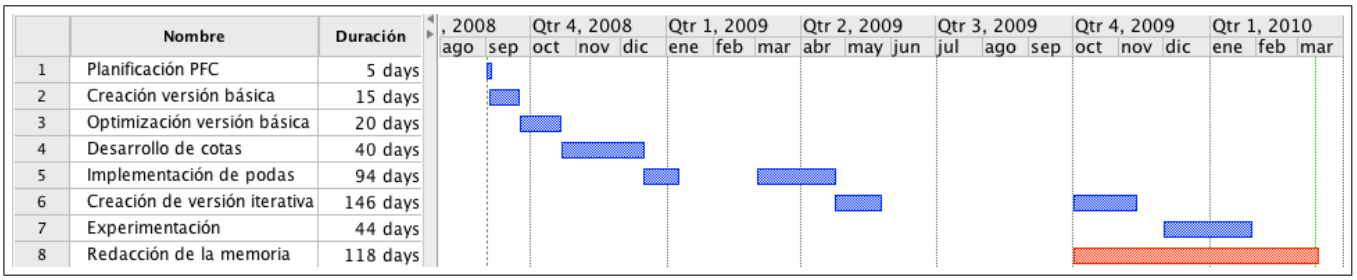

Figura 4.2: Diagrama de Gantt de seguimiento

4.5. Diagrama de clases

Aunque en los principios de la XP se expresa que el propio código debe ser toda la documentación necesaria, un diagrama de clases permite mostrar la estructura del proyecto y cómo los distintos módulos están interconectados. Esto es útil a la hora de añadir funcionalidades adicionales en futuros trabajos. En la Figura 4.3 se puede observar el diagrama de clases asociado a este proyecto.

4.5.1. Clases principales

A continuación se describirán las clases más importantes del proyecto.

- Clase **Instance**: Esta clase es la encargada de modelar los grafos usados para representar los problemas.
- Clase **Solution**: Esta clase contiene una solución (una permutación) de un problema concreto. Esta solución puede ser parcial o completa.
- Clase **Antibandwidth**: Esta clase es la encargada de implementar todos los algoritmos capaces de resolver el problema del AB.

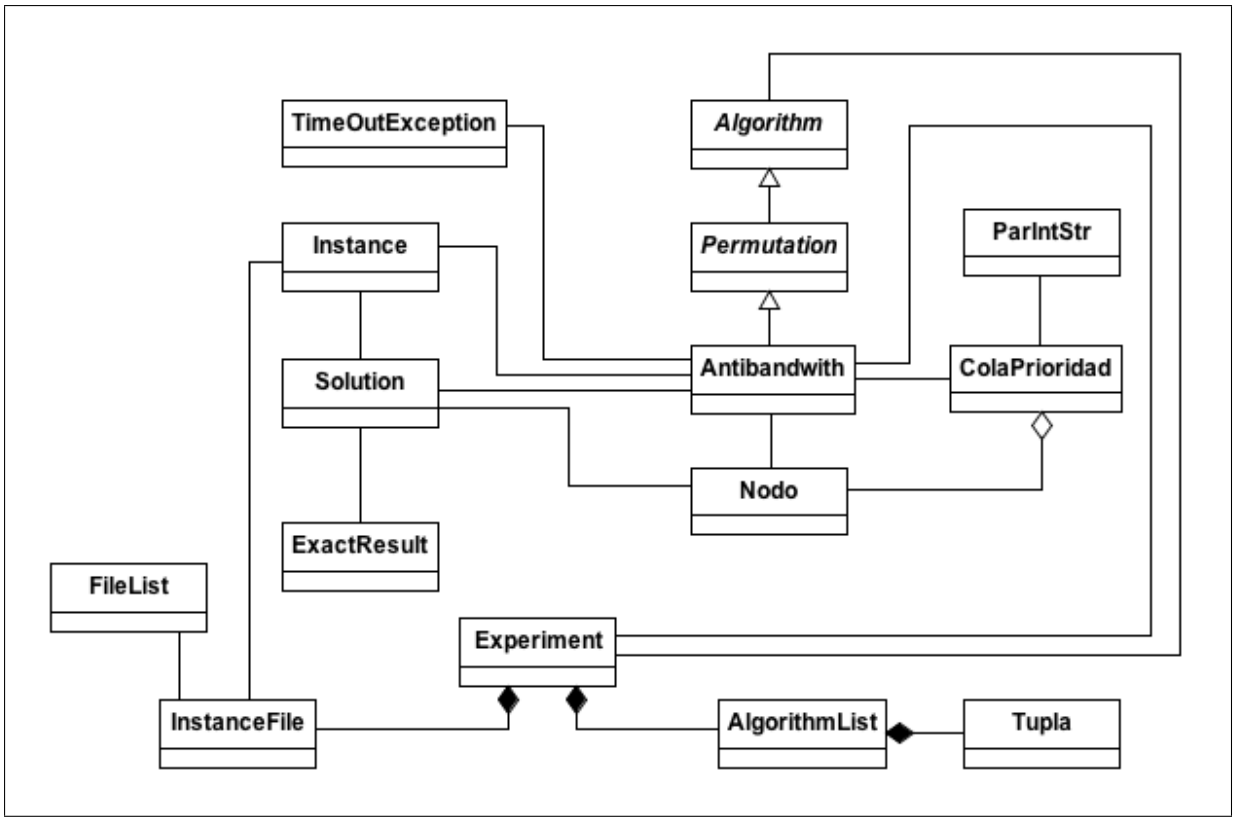

Figura 4.3: Diagrama de clases del proyecto

- Clase **ColaPrioridad**: Esta clase modela una cola de prioridad capaz de volcar y recuperar de almacenamiento secundario parte de su contenido.

- Clase **Nodo**: Esta clase modela los nodos del árbol de exploración. Es usada por la clase ColaPrioridad.

4.6. Paradigma de programación

Para el correcto desarrollo de este proyecto fue preciso elegir un lenguaje de programación cuyas características permitieran un desarrollo ágil e intuitivo. Por ello se eligió un lenguaje de programación basado en el Paradigma de Orientación a Objetos. Mediante este paradigma se puede realizar una descripción de las distintas partes del proyecto lo más cercana posible a la realidad y permite aplicar la práctica de las metáforas de la XP (ver Sección 4.2.2) de manera más sencilla que un lenguaje basado en el paradigma estructural.

A continuación se describirán los principios de la Programación Orientada a Objetos.

4.6.1. Programación Orientada a Objetos

La Programación Orientada a Objetos (POO) es un paradigma de programación que, como su nombre indica, modela los datos mediante objetos, intentando imitar la realidad. Estos objetos son entidades que combinan estado, comportamiento e identidad.

- El **estado** de un objeto son los valores de los atributos que representan las características básicas de un objeto y que conforman su estructura interna en un momento dado. Por ejemplo un objeto que represente una caja podría tener atributos tales como anchura, altura y profundidad, y unos valores determinados para esos atributos.

- El **comportamiento** son las operaciones que un objeto puede realizar, estas operaciones son denominadas métodos. En el ejemplo del objeto caja, un método podría, por ejemplo, calcular el volumen de dicha caja.

- La **identidad** de un objeto es una propiedad mediante la cual se puede identificar cualquier objeto unívocamente, incluso cuando dos objetos de la misma clase poseen los mismos valores para sus atributos.

La POO expresa un programa como un conjunto de objetos que colaboran entre sí para realizar tareas. Esto permite hacer los programas y módulos más fáciles de escribir, mantener, y reutilizar.

En todo lenguaje que se considere como orientado a objetos se debe de poder reconocer cada una de estas características:

- **Abstracción**: Denota las características esenciales de un objeto, donde se captura su comportamiento. Los procesos, las funciones o los métodos pueden también ser fruto de la abstracción.

- **Encapsulamiento**: Significa reunir a todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción. Esto permite aumentar la cohesión de los componentes del sistema.

- **Herencia**: Es la facultad mediante la cual una clase de objeto, a la que llamaremos clase hija, hereda en ella los atributos y operaciones de otra clase de objeto, llamada clase padre, como si esos atributos y operaciones hubiesen sido definidos en la propia clase hija. Esta característica permite organizar jerárquicamente los distintos objetos que forman parte de un programa. La herencia permite crear especializaciones de objetos sin tener que volver a definir las partes comunes.

- **Polimorfismo**: Es la capacidad de asociar, a variables de una clase padre, objetos de clases que hereden de esta última (clases hijas). Esto permite que se produzcan comportamientos distintos para los mismos métodos, dependiendo del objeto de la clase hija que se asigne a la variable de la clase padre.

4.7. Tecnologías empleadas

Durante el desarrollo de este PFC se emplearon diversas tecnologías que serán expuestas a continuación.

4.7.1. Lenguaje de programación

Para el desarrollo de este proyecto se optó por utilizar el lenguaje de programación Java de Sun Microsystems. En concreto se utilizó la versión Java SE 6. Este lenguaje permite emplear el paradigma de la POO. A continuación se describirán las principales características del lenguaje de programación Java.

Lenguaje de programación Java

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.

Las aplicaciones Java están típicamente compiladas en un *bytecode*, aunque la compilación en código máquina nativo también es posible. En tiempo de ejecución, el *bytecode* es normalmente interpretado y compilado a código nativo para la ejecución, aunque la ejecución directa por *hardware* del *bytecode* por un procesador Java también es posible. El lenguaje Java se creó con cinco objetivos principales:

- Debía usar la metodología de la Programación Orientada a Objetos.
- Debía permitir la ejecución de un mismo programa en múltiples sistemas operativos.
- Debía incluir, por defecto, soporte para trabajo en red.
- Debía diseñarse para ejecutar código en sistemas remotos de forma segura.
- Debía ser fácil de usar y tomar lo mejor de otros lenguajes orientados a objetos, como C++.

4.7.2. Entorno de desarrollo

A continuación se describirá el entorno de desarrollo utilizado para la elaboración del este PFC.

IDE Eclipse [5]

Eclipse es un Entorno de Desarrollo Integrado (IDE, de sus siglas en inglés, *Integrated Development Environment*), es decir, un programa informático compuesto por un conjunto de herramientas que facilitan la tarea de programación y depuración de aplicaciones *software*. Los IDE proveen un marco de trabajo amigable para los programadores, siendo posible que un mismo IDE pueda utilizar varios lenguajes de programación. Este es el caso de Eclipse, al que mediante *plugins* se le puede añadir soporte para distintos lenguajes. También es posible añadir funcionalidades adicionales que no se encuentran disponibles de serie, mediante el uso de estos *plugins*, como por ejemplo: editores de texto LaTeX, integración de sistemas de control de versiones, etc.

4.7.3. GNU/Linux [4][7]

Para la experimentación llevada a cabo en este proyecto fue necesario utilizar el sistema operativo Linux. En concreto se utilizó la distribución Ubuntu Linux 64 bits [8]. El uso de un sistema de 64 bits es necesario porque, dadas las características de los algoritmos, se requiere una gran cantidad de memoria para su correcta ejecución y sólo un sistema de 64 bits es capaz de direccionar mas de 3,2 GB de memoria.

La elección de Ubuntu Linux como sistema operativo se debe a su sencillez de instalación y, dado el gran número de usuarios del mismo, la información necesaria para solucionar posibles problemas técnicos está prácticamente garantiza sea cual sea la naturaleza del mismo.

4.7.4. Sistema de documentación LaTeX

LaTeX [2] es un sistema de composición de textos escrito por Leslie Lamport en 1984, con la intención de facilitar el uso del lenguaje de composición tipográfica TeX, un lenguaje «de bajo nivel» en el sentido de que sus acciones últimas son muy elementales, creado por Donald Knuth. LaTeX está formado mayoritariamente por órdenes (macros) construidas a partir de comandos de TeX. Es muy utilizado para la composición de artículos académicos, tesis y libros técnicos, dado que la calidad tipográfica de los documentos realizados con LaTeX es comparable a la de una editorial científica de primera línea. LaTeX es *software* libre bajo licencia LPPL (LaTeX Project Public License).

LaTeX presupone una filosofía de trabajo diferente a la de los procesadores de texto habituales y se basa en comandos. Tradicionalmente, este aspecto se ha considerado una desventaja, sin embargo, LaTeX, a diferencia de los procesadores de texto tradicionales, permite a quien escribe un documento centrarse exclusivamente en el contenido, sin tener que preocuparse de los detalles del formato. Además de sus capacidades gráficas para representar ecuaciones, fórmulas complicadas, notación científica e incluso musical, permite estructurar fácilmente el documento (con capítulos, secciones, notas, bibliografía, índices analíticos, etc.) lo cual brinda comodidad y lo hace útil para artículos académicos y libros técnicos.

4.7.5. Solver CPLEX

CPLEX es un *software* de optimización desarrollado por la compañía ILOG [1]. Su nombre viene dado por el lenguaje de programación C y el método simplex, a pesar de que hoy en día implementa el método de puntos interiores y tiene interfaces para los lenguajes de programación C++ , C# y Java.

CPLEX fue originalmente desarrollado por Robert E. Bixby y comercializado por CPLEX Optimization Inc., la cual fue adquirida por ILOG en 1997. ILOG fue recientemente comprada por IBM en Enero de 2009.

Capítulo 5

Resultados Experimentales

En este capítulo se describirán los experimentos realizados para comprobar la calidad de las soluciones que los distintos algoritmos ofrecen.

5.1. Conjunto de instancias de evaluación

Para las pruebas se utilizaron dos conjuntos de problemas: uno aleatorio generado mediante una pequeña aplicación desarrollada para tal fin y otro conjunto llamado "Smallinstances" que fueron introducidas en [9]. Cada uno de estos problemas recibe el nombre de instancia.

- **Instancias aleatorias**: Este conjunto de pruebas esta formado por 13 grafos, cuyo número de vértices, va desde 12 hasta 24. Este grupo de problemas se creó para ser el banco de pruebas durante el desarrollo del PFC. Cada vez que un algoritmo era implementado, o se alcanzaba una nueva versión, se utilizaba este banco de pruebas para comprobar tanto la corrección de la ejecución, como el rendimiento de dicha versión.

- **SmallInstances**: Este conjunto de pruebas esta formado por 84 grafos, cuyo número de vértices, va desde los 16 del más pequeño hasta los 24 del más grande. Este conjunto es el empleado para medir, tanto el rendimiento de cada una de las versiones finales de las distintas técnicas algorítmicas, como del *solver* CPLEX, permitiendo así tener un marco comparativo.

5.2. Resultados experimentales

A continuación se expondrán los resultados experimentales en dos tablas resumen que permitirán comparar el rendimiento de los distintos algoritmos. En la primera de ellas la solución inicial será lexicográfica, mientras que en la segunda se empleará un algoritmo

heurístico para alcanzar la solución inicial. Los datos que pueden verse en las Tablas 5.1 y 5.2 son la media aritmética del conjunto total de problemas. Este estudio estadístico de los resultados permite que las anomalías que puedan darse en los resultados sean filtradas y los datos muestren la tendencia general de los distintos algoritmos. Los datos individuales de las distintas instancias pueden consultarse en el Anexo A. Para este experimento se ha empleado el conjunto de instancias SmallInstances con un tiempo máximo de una hora.

Small Instances	BT	BT*	$B\&B_1$	$B\&B_2$	CPLEX
# Óptimos	0	56	69	62	62
Gap Absoluto	16,833	5,083	1,238	3,405	0,369
Gap Relativo	0,867	0,235	0,100	0,166	0,035
CPU Time (s)	3600	1526,389	825,180	1138,925	1194,320

Tabla 5.1: Algoritmos con solución inicial lexicográfica

Small Instances	BTheur	BTheur*	$B\&B_1heur$	$B\&B_2heur$	CPLEX
# Óptimos	0	62	71	63	62
Gap Absoluto	15,191	3,917	0,786	3,321	0,369
Gap Relativo	0,782	0,178	0,061	0,160	0,035
CPU Time (s)	3600	1199,460	733,952	1161,214	1194,320

Tabla 5.2: Algoritmos con solución inicial heurística

La nomenclatura de los algoritmos es la siguiente:

- **BT**: Vuelta atrás básico con solución inicial lexicográfica.
- **BTheur**: Vuelta atrás básico con solución inicial heurística.
- **BT***: Vuelta atrás con poda, estimación y solución inicial lexicográfica.
- **BTheur***: Vuelta atrás con poda, estimación y solución inicial heurística.
- $\boldsymbol{B\&B_1}$: Ramificación y acotación con recorrido en orden y solución inicial lexicográfica.
- $\boldsymbol{B\&B_1}$**heur**: Ramificación y acotación con recorrido en orden y solución inicial heurística.
- $\boldsymbol{B\&B_2}$: Ramificación y acotación con recorrido inverso y solución inicial lexicográfica.
- $\boldsymbol{B\&B_2}$**heur**: Ramificación y acotación con recorrido inverso y solución inicial heurística.
- **CPLEX**: Uso de la formulación matemática mediante el *solver* CPLEX.

Los valores de las tablas deben ser interpretados de la siguiente manera:

- #**Óptimos**: Es el número total de soluciones exactas que el algoritmo ha sido capaz de obtener en el tiempo máximo de 1 hora.

- **Gap Absoluto**: Es la diferencia en valor absoluto entre el valor de la mejor solución que el algoritmo ha sido capaz de encontrar en el tiempo máximo de 1 hora (cota inferior) y el mejor valor que el algoritmo estima (cota superior) podría ser el valor de la mejor solución, es decir, la diferencia en valor absoluto entre la cota inferior y la cota superior. Cuando una solución es óptima este valor es 0.

- **Gap Relativo**: Matemáticamente es el valor del *gap* absoluto dividido entre la cota superior. Esto representa un porcentaje de cuanto se acerca la cota inferior obtenida a la solución óptima estimada. En una solución óptima este valor es 0. Por ejemplo, para una instancia cuya cota inferior es 7000 y cuya cota superior es 8000, su *gap* absoluto es 1000. Este valor puede parecer muy alto, sin embargo si lo medimos en términos de *gap* relativo obtenemos un valor de 0,14. Teniendo en cuenta que el *gap* relativo de una solución óptima es 0, podemos observar que el valor 7000 se encuentra mucho más cerca del valor óptimo de lo que se podría pensar al observar el *gap* absoluto.

- **CPU Time (s)**: Es el tiempo medio que ha empleado el algoritmo para la ejecución de los problemas planteados.

5.3. Análisis de los resultados

En este apartado se realizará un análisis de los resultados obtenidos utilizando como marco de referencia el *solver* CPLEX.

5.3.1. Descripción de las tablas de datos

Los datos que se muestran en las tablas de la sección anterior corresponden a la experimentación realizada usando el conjunto de problemas denominado SmallInstances. En ambas tablas se muestran los resultados obtenidos por el *solver* CPLEX para poder realizar una comparación.

En la Tabla 5.1 se muestran los resultados de los algoritmos que utilizaban una solución inicial lexicográfica, mientras que en la Tabla 5.2 están los datos de los algoritmos que utilizaban una solución inicial heurística.

5.3.2. Comparación de los algoritmos desarrollados

Como se pueden observar en las tablas resumen, aquellos algoritmos que utilizan una solución inicial heurística ven sustancialmente reducido su tiempo de ejecución así como los valores del *gap* tanto absoluto como relativo, a la vez que obtienen una mayor cantidad de soluciones óptimas. Esto se debe a que la solución inicial heurística permite a los algoritmos empezar a podar utilizando un cota inferior más alta que la que aporta una solución inicial lexicográfica.

Una vez centrados en los algoritmos que usan soluciones heurísticas observamos que el mejor algoritmo de los implementados es el algoritmo de ramificación y acotación que hace uso de la cola de prioridad con recorrido en orden y volcado a disco. Este método es capaz de resolver el 85 % de las instancias en un tiempo medio de 12 minutos y 15 segundos, además de tener el *gap* absoluto y relativo más bajo de todos los algoritmos desarrollados.

5.3.3. Comparación con CPLEX

Como se expuso en el apartado anterior el mejor algoritmo de los desarrollados es el que usa la técnica de ramificación y acotación haciendo uso de la cola de prioridad con recorrido en orden, $B\&B_2heur$ siguiendo con la nomenclatura de la Tabla 5.2.

Al compararlo con el *solver* CPLEX se puede observar que el algoritmo $B\&B_2heur$ es capaz de resolver un número mayor de instancias. CPLEX consigue resolver el 74 % de las instancias, mientras que el algoritmo $B\&B_2heur$ consigue resolver el 85 % de estas instancias. También se puede destacar que el tiempo de ejecución medio es sustancialmente menor: 7 minutos 40 segundos menos.

Sin embargo se puede apreciar que tanto el *gap* absoluto como el relativo es menor utilizando el *solver* CPLEX. Esto significa que, en las soluciones que no consigue alcanzar el óptimo, la solución que proporciona CPLEX se acerca más al resultado óptimo que la solución que proporcionaría el algoritmo $B\&B_2heur$.

Capítulo 6

Conclusiones y trabajos futuros

A continuación se expondrán los hitos conseguido así como las propuestas para trabajos futuros.

6.1. Objetivos alcanzados

Como resultado de la elaboración de este PFC se han alcanzado los siguientes hitos:

- **Desarrollo de un algorítmo básico**: Se ha implementado un versión inicial con las funcionalidades más básicas, a partir de la cual se ha desarrollado el proyecto.
- **Proposición de cotas**: Se han propuesto una serie de cotas para el problema del AB.
- **Desarrollo de un algorítmo mejorado**: Mediante la aplicación de las cotas se consiguió desarrollar un algoritmo recursivo mejorado de ramificación y acotación.
- **Desarrollo de un algorítmo iterativo**: Se ha logrado desarrollar un algoritmo iterativo de ramificación y acotación que, mediante el uso de las cotas, permite un recorrido más eficiente del árbol de exploración.
- **CPLEX**: Se ha utilizado el *solver* CPLEX para implementar la formulación matemática descrita en la Sección 1.4.

Con todo esto se puede concluir que se ha logrado el objetivo principal de desarrollar diferentes versiones de algoritmos exactos capaces de resolver el problema del AB de manera satisfactoria.

6.2. Resultados

Mediante la realización de este PFC se ha realizado un estudio de distintas técnicas algorítmicas capaces de resolver el problema del AB de manera exacta. La posterior ex-

perimentación con un conjunto de instancias de pruebas estándar ha conseguido resolver instancias de hasta veintiún vértices para el problema del AB de manera óptima.

6.3. Trabajos futuros

En está sección se expondrán algunas ideas sobre posibles trabajos futuros de este PFC.

6.3.1. Procesamiento en paralelo

En este PFC se han desarrollado algoritmos monohilo, es decir, algoritmos que sólo ejecutan sobre un proceso y consecuentemente sobre un único procesador. Sin embargo sería factible y muy interesante implementar estos algoritmos para que hagan uso de las técnicas de computación distribuida para mejorar su rendimiento.

Hasta hace poco, un ordenador no albergaba más de un procesador en su interior y los centros de cálculo utilizaban *clusters* de ordenadores interconectados en red. Sin embargo, actualmente la mayoría de ordenadores que se comercializan poseen dos o más procesadores y se puede tener acceso a un sistema de computación distribuida de manera relativamente económica.

Mediante una implementación distribuida de los distintos algoritmos, se podría multiplicar, teóricamente, la capacidad de exploración de los algoritmos por el número de procesadores que la máquina poseyese. Esto permitiría ,en los casos en los cuales los algoritmos monohilo son capaces de encontrar la solución óptima, reducir considerablemente el tiempo de ejecución, o, en aquellos problemas en los cuales el algoritmo no consigue una solución óptima, reducir el espacio de búsqueda explorado.

Anexo A

Tablas de datos completas

A.1. Vuelta atrás básico

A.1.1. Solución inicial lexicográfica (BT)

Small Instances	Cota Inferior	Cota Superior	Gap Absoluto	Gap Relativo	Tiempo (s)
p17_16_24	4	15	11	0,733	3600,000
p18_16_21	1	15	14	0,933	3600,000
p19_16_19	6	15	9	0,600	3600,000
p20_16_18	5	15	10	0,667	3600,000
p21_17_20	1	16	15	0,938	3600,000
p22_17_19	6	16	10	0,625	3600,000
p23_17_23	5	16	11	0,688	3600,000
p24_17_29	1	16	15	0,938	3600,000
p25_17_20	1	16	15	0,938	3600,000
p26_17_19	4	16	12	0,750	3600,000
p27_17_19	1	16	15	0,938	3600,000
p28_17_18	4	16	12	0,750	3600,000
p29_17_18	1	16	15	0,938	3600,000
p30_17_19	3	16	13	0,813	3600,000
p31_18_21	6	17	11	0,647	3600,000
p32_18_20	1	17	16	0,941	3600,000
p33_18_21	1	17	16	0,941	3600,000
p34_18_21	6	17	11	0,647	3600,000
p35_18_19	5	17	12	0,706	3600,000
p36_18_20	2	17	15	0,882	3600,000
p37_18_20	5	17	12	0,706	3600,000
p38_18_19	6	17	11	0,647	3600,000
p39_18_19	6	17	11	0,647	3600,000
p40_18_32	2	17	15	0,882	3600,000
p41_19_20	6	18	12	0,667	3600,000
p42_19_24	1	18	17	0,944	3600,000
p43_19_22	2	18	16	0,889	3600,000
p44_19_25	1	18	17	0,944	3600,000
p45_19_25	4	18	14	0,778	3600,000
p46_19_20	1	18	17	0,944	3600,000
p47_19_21	1	18	17	0,944	3600,000
p48_19_21	6	18	12	0,667	3600,000
p49_19_22	6	18	12	0,667	3600,000
p50_19_25	2	18	16	0,889	3600,000
p51_20_28	1	19	18	0,947	3600,000
p52_20_27	1	19	18	0,947	3600,000
p53_20_22	1	19	18	0,947	3600,000
p54_20_28	5	19	14	0,737	3600,000
p55_20_24	1	19	18	0,947	3600,000
p56_20_23	2	19	17	0,895	3600,000
p57_20_24	1	19	18	0,947	3600,000
p58_20_21	1	19	18	0,947	3600,000
p59_20_23	1	19	18	0,947	3600,000
p60_20_22	1	19	18	0,947	3600,000
p61_21_22	1	20	19	0,950	3600,000

Small Instances	Cota Inferior	Cota Superior	Gap Absoluto	Gap Relativo	Tiempo (s)
p63_21_42	2	20	18	0,900	3600,000
p64_21_22	6	20	14	0,700	3600,000
p65_21_24	1	20	19	0,950	3600,000
p66_21_28	3	20	17	0,850	3600,000
p67_21_22	2	20	18	0,900	3600,000
p68_21_27	1	20	19	0,950	3600,000
p69_21_23	5	20	15	0,750	3600,000
p70_21_25	1	20	19	0,950	3600,000
p71_22_29	1	21	20	0,952	3600,000
p72_22_49	1	21	20	0,952	3600,000
p73_22_29	1	21	20	0,952	3600,000
p74_22_30	2	21	19	0,905	3600,000
p75_22_25	2	21	19	0,905	3600,000
p76_22_30	1	21	20	0,952	3600,000
p77_22_37	1	21	20	0,952	3600,000
p78_22_31	1	21	20	0,952	3600,000
p79_22_29	3	21	18	0,857	3600,000
p80_22_30	2	21	19	0,905	3600,000
p81_23_46	1	22	21	0,955	3600,000
p82_23_24	6	22	16	0,727	3600,000
p83_23_24	2	22	20	0,909	3600,000
p84_23_26	4	22	18	0,818	3600,000
p85_23_26	1	22	21	0,955	3600,000
p86_23_24	1	22	21	0,955	3600,000
p87_23_30	1	22	21	0,955	3600,000
p88_23_26	2	22	20	0,909	3600,000
p89_23_27	1	22	21	0,955	3600,000
p90_23_35	2	22	20	0,909	3600,000
p91_24_33	4	23	19	0,826	3600,000
p92_24_26	3	23	20	0,870	3600,000
p93_24_27	2	23	21	0,913	3600,000
p94_24_31	1	23	22	0,957	3600,000
p95_24_27	2	23	21	0,913	3600,000
p96_24_27	2	23	21	0,913	3600,000
p97_24_26	1	23	22	0,957	3600,000
p98_24_29	1	23	22	0,957	3600,000
p99_24_27	2	23	21	0,913	3600,000
p100_24_34	1	23	22	0,957	3600,000

A.1.2. Solución inicial heurística (BTheur)

Small Instances	Cota Inferior	Cota Superior	Gap Absoluto	Gap Relativo	Tiempo (s)
p17_16_24	3	15	12	0,800	3600,000
p18_16_21	5	15	10	0,667	3600,000
p19_16_19	5	15	10	0,667	3600,000
p20_16_18	4	15	11	0,733	3600,000
p21_17_20	5	16	11	0,688	3600,000
p22_17_19	6	16	10	0,625	3600,000
p23_17_23	5	16	11	0,688	3600,000
p24_17_29	1	16	15	0,938	3600,000
p25_17_20	2	16	14	0,875	3600,000
p26_17_19	4	16	12	0,750	3600,000
p27_17_19	5	16	11	0,688	3600,000
p28_17_18	5	16	11	0,688	3600,000
p29_17_18	5	16	11	0,688	3600,000
p30_17_19	4	16	12	0,750	3600,000
p31_18_21	5	17	12	0,706	3600,000
p32_18_20	6	17	11	0,647	3600,000
p33_18_21	5	17	12	0,706	3600,000
p34_18_21	6	17	11	0,647	3600,000
p35_18_19	5	17	12	0,706	3600,000

Small Instances	Cota Inferior	Cota Superior	Gap Absoluto	Gap Relativo	Tiempo (s)
p36_18_20	4	17	13	0,765	3600,000
p37_18_20	5	17	12	0,706	3600,000
p38_18_19	5	17	12	0,706	3600,000
p39_18_19	5	17	12	0,706	3600,000
p40_18_32	3	17	14	0,824	3600,000
p41_19_20	5	18	13	0,722	3600,000
p42_19_24	4	18	14	0,778	3600,000
p43_19_22	2	18	16	0,889	3600,000
p44_19_25	4	18	14	0,778	3600,000
p45_19_25	4	18	14	0,778	3600,000
p46_19_20	4	18	14	0,778	3600,000
p47_19_21	5	18	13	0,722	3600,000
p48_19_21	6	18	12	0,667	3600,000
p49_19_22	5	18	13	0,722	3600,000
p50_19_25	2	18	16	0,889	3600,000
p51_20_28	4	19	15	0,789	3600,000
p52_20_27	4	19	15	0,789	3600,000
p53_20_22	5	19	14	0,737	3600,000
p54_20_28	5	19	14	0,737	3600,000
p55_20_24	5	19	14	0,737	3600,000
p56_20_23	4	19	15	0,789	3600,000
p57_20_24	5	19	14	0,737	3600,000
p58_20_21	5	19	14	0,737	3600,000
p59_20_23	5	19	14	0,737	3600,000
p60_20_22	5	19	14	0,737	3600,000
p61_21_22	5	20	15	0,750	3600,000
p62_21_30	4	20	16	0,800	3600,000
p63_21_42	2	20	18	0,900	3600,000
p64_21_22	5	20	15	0,750	3600,000
p65_21_24	5	20	15	0,750	3600,000
p66_21_28	4	20	16	0,800	3600,000
p67_21_22	4	20	16	0,800	3600,000
p68_21_27	4	20	16	0,800	3600,000
p69_21_23	5	20	15	0,750	3600,000
p70_21_25	4	20	16	0,800	3600,000
p71_22_29	4	21	17	0,810	3600,000
p72_22_49	1	21	20	0,952	3600,000
p73_22_29	4	21	17	0,810	3600,000
p74_22_30	2	21	19	0,905	3600,000
p75_22_25	2	21	19	0,905	3600,000
p76_22_30	1	21	20	0,952	3600,000
p77_22_37	3	21	18	0,857	3600,000
p78_22_31	1	21	20	0,952	3600,000
p79_22_29	5	21	16	0,762	3600,000
p80_22_30	3	21	18	0,857	3600,000
p81_23_46	1	22	21	0,955	3600,000
p82_23_24	6	22	16	0,727	3600,000
p83_23_24	2	22	20	0,909	3600,000
p84_23_26	3	22	19	0,864	3600,000
p85_23_26	5	22	17	0,773	3600,000
p86_23_24	6	22	16	0,727	3600,000
p87_23_30	4	22	18	0,818	3600,000
p88_23_26	5	22	17	0,773	3600,000
p89_23_27	4	22	18	0,818	3600,000
p90_23_35	4	22	18	0,818	3600,000
p91_24_33	3	23	20	0,870	3600,000
p92_24_26	4	23	19	0,826	3600,000
p93_24_27	5	23	18	0,783	3600,000
p94_24_31	5	23	18	0,783	3600,000
p95_24_27	6	23	17	0,739	3600,000
p96_24_27	4	23	19	0,826	3600,000
p97_24_26	5	23	18	0,783	3600,000
p98_24_29	1	23	22	0,957	3600,000
p99_24_27	5	23	18	0,783	3600,000
p100_24_34	2	23	21	0,913	3600,000

A.2. Vuelta atrás con poda

A.2.1. Solución inicial lexicográfica (BT*)

Small Instances	Cota Inferior	Cota Superior	Gap Absoluto	Gap Relativo	Tiempo (s)
p17_16_24	5	5	0	0,000	4,048
p18_16_21	6	6	0	0,000	2,328
p19_16_19	7	7	0	0,000	1,204
p20_16_18	6	6	0	0,000	9,220
p21_17_20	7	7	0	0,000	127,767
p22_17_19	7	7	0	0,000	12,548
p23_17_23	6	6	0	0,000	64,368
p24_17_29	5	5	0	0,000	8,408
p25_17_20	7	7	0	0,000	0,110
p26_17_19	7	7	0	0,000	16,721
p27_17_19	8	8	0	0,000	3,360
p28_17_18	8	8	0	0,000	16,049
p29_17_18	7	7	0	0,000	262,270
p30_17_19	7	7	0	0,000	36,927
p31_18_21	7	7	0	0,000	9,923
p32_18_20	8	8	0	0,000	35,865
p33_18_21	8	8	0	0,000	105,202
p34_18_21	7	7	0	0,000	29,660
p35_18_19	7	7	0	0,000	88,168
p36_18_20	8	8	0	0,000	286,366
p37_18_20	8	8	0	0,000	30,520
p38_18_19	8	8	0	0,000	48,272
p39_18_19	8	8	0	0,000	34,208
p40_18_32	6	6	0	0,000	0,782
p41_19_20	8	8	0	0,000	786,700
p42_19_24	7	7	0	0,000	805,358
p43_19_22	7	7	0	0,000	30,208
p44_19_25	8	8	0	0,000	2629,994
p45_19_25	7	7	0	0,000	11,767
p46_19_20	9	9	0	0,000	255,503
p47_19_21	8	8	0	0,000	80,246
p48_19_21	8	8	0	0,000	132,877
p49_19_22	8	8	0	0,000	60,367
p50_19_25	7	7	0	0,000	30,426
p51_20_28	8	8	0	0,000	778,089
p52_20_27	8	8	0	0,000	905,700
p53_20_22	9	9	0	0,000	786,169
p54_20_28	8	8	0	0,000	55,320
p55_20_24	9	9	0	0,000	5,532
p56_20_23	9	9	0	0,000	33,474
p57_20_24	8	8	0	0,000	189,932
p58_20_21	8	19	11	0,579	3600,000
p59_20_23	8	19	11	0,579	3600,000
p60_20_22	4	19	15	0,789	3600,000
p61_21_22	4	20	16	0,800	3600,000
p62_21_30	9	9	0	0,000	812,032
p63_21_42	6	6	0	0,000	1062,187
p64_21_22	9	20	11	0,550	3600,000
p65_21_24	9	20	11	0,550	3600,000
p66_21_28	8	8	0	0,000	163,578
p67_21_22	3	20	17	0,850	3600,000
p68_21_27	8	8	0	0,000	656,078
p69_21_23	9	9	0	0,000	598,703
p70_21_25	9	9	0	0,000	2262,922
p71_22_29	9	20	11	0,550	3600,000
p72_22_49	6	21	15	0,714	3600,000
p73_22_29	9	9	0	0,000	2506,422
p74_22_30	8	8	0	0,000	1191,156
p75_22_25	9	9	0	0,000	1366,343
p76_22_30	8	8	0	0,000	624,687
p77_22_37	8	8	0	0,000	488,093
p78_22_31	8	8	0	0,000	1394,359
p79_22_29	9	9	0	0,000	2953,563
p80_22_30	8	21	13	0,619	3600,000
p81_23_46	7	22	15	0,682	3600,000
p82_23_24	3	22	19	0,864	3600,000
p83_23_24	8	22	14	0,636	3600,000
p84_23_26	4	22	18	0,818	3600,000

Small Instances	Cota Inferior	Cota Superior	Gap Absoluto	Gap Relativo	Tiempo (s)
p85_23_26	3	22	19	0,864	3600,000
p86_23_24	6	22	16	0,727	3600,000
p87_23_30	8	22	14	0,636	3600,000
p88_23_26	6	22	16	0,727	3600,000
p89_23_27	5	22	17	0,773	3600,000
p90_23_35	9	9	0	0,000	901,234
p91_24_33	9	23	14	0,609	3600,000
p92_24_26	7	23	16	0,696	3600,000
p93_24_27	7	23	16	0,696	3600,000
p94_24_31	9	23	14	0,609	3600,000
p95_24_27	3	23	20	0,870	3600,000
p96_24_27	2	23	21	0,913	3600,000
p97_24_26	6	23	17	0,739	3600,000
p98_24_29	10	10	0	0,000	1623,328
p99_24_27	10	23	13	0,565	3600,000
p100_24_34	6	23	17	0,739	3600,000

A.2.2. Solución inicial heurística (BTheur*)

Small Instances	Cota Inferior	Cota Superior	Gap Absoluto	Gap Relativo	Tiempo (s)
p17_16_24	5	5	0	0,000	1,481
p18_16_21	6	6	0	0,000	0,814
p19_16_19	7	7	0	0,000	0,374
p20_16_18	6	6	0	0,000	2,815
p21_17_20	7	7	0	0,000	32,770
p22_17_19	7	7	0	0,000	4,021
p23_17_23	6	6	0	0,000	19,734
p24_17_29	5	5	0	0,000	2,636
p25_17_20	7	7	0	0,000	0,056
p26_17_19	7	7	0	0,000	12,704
p27_17_19	8	8	0	0,000	1,092
p28_17_18	8	8	0	0,000	5,143
p29_17_18	7	7	0	0,000	9,277
p30_17_19	7	7	0	0,000	13,237
p31_18_21	7	7	0	0,000	3,195
p32_18_20	8	8	0	0,000	10,907
p33_18_21	8	8	0	0,000	35,750
p34_18_21	7	7	0	0,000	9,868
p35_18_19	7	7	0	0,000	27,921
p36_18_20	8	8	0	0,000	90,234
p37_18_20	8	8	0	0,000	3,726
p38_18_19	8	8	0	0,000	15,207
p39_18_19	8	8	0	0,000	11,234
p40_18_32	6	6	0	0,000	0,274
p41_19_20	8	8	0	0,000	259,427
p42_19_24	7	7	0	0,000	275,284
p43_19_22	7	7	0	0,000	10,151
p44_19_25	8	8	0	0,000	849,280
p45_19_25	7	7	0	0,000	3,481
p46_19_20	9	9	0	0,000	115,384
p47_19_21	8	8	0	0,000	1,074
p48_19_21	8	8	0	0,000	26,638
p49_19_22	8	8	0	0,000	20,578
p50_19_25	7	7	0	0,000	10,181
p51_20_28	8	8	0	0,000	261,119
p52_20_27	8	8	0	0,000	174,804
p53_20_22	9	9	0	0,000	265,129
p54_20_28	8	8	0	0,000	19,332
p55_20_24	9	9	0	0,000	1,854
p56_20_23	9	9	0	0,000	11,212
p57_20_24	8	8	0	0,000	54,131
p58_20_21	8	8	0	0,000	1762,795
p59_20_23	8	8	0	0,000	1487,978
p60_20_22	9	9	0	0,000	656,685

Small Instances	Cota Inferior	Cota Superior	Gap Absoluto	Gap Relativo	Tiempo (s)
p61_21_22	10	20	10	0,500	3600,000
p62_21_30	4	20	16	0,800	3600,000
p63_21_42	6	6	0	0,000	365,544
p64_21_22	9	20	11	0,550	3600,000
p65_21_24	9	9	0	0,000	1880,620
p66_21_28	8	8	0	0,000	50,614
p67_21_22	7	20	13	0,650	3600,000
p68_21_27	8	8	0	0,000	209,308
p69_21_23	9	9	0	0,000	186,940
p70_21_25	9	9	0	0,000	747,557
p71_22_29	9	9	0	0,000	1486,558
p72_22_49	6	21	15	0,714	3600,000
p73_22_29	9	9	0	0,000	854,379
p74_22_30	8	8	0	0,000	424,888
p75_22_25	9	9	0	0,000	474,857
p76_22_30	8	8	0	0,000	205,873
p77_22_37	8	8	0	0,000	163,392
p78_22_31	8	8	0	0,000	488,370
p79_22_29	9	9	0	0,000	993,727
p80_22_30	3	21	18	0,857	3600,000
p81_23_46	7	7	0	0,000	1978,541
p82_23_24	6	22	16	0,727	3600,000
p83_23_24	8	22	14	0,636	3600,000
p84_23_26	9	9	0	0,000	3595,776
p85_23_26	5	22	17	0,773	3600,000
p86_23_24	8	22	14	0,636	3600,000
p87_23_30	9	22	13	0,591	3600,000
p88_23_26	9	22	13	0,591	3600,000
p89_23_27	5	22	17	0,773	3600,000
p90_23_35	9	9	0	0,000	307,309
p91_24_33	9	23	14	0,609	3600,000
p92_24_26	7	23	16	0,696	3600,000
p93_24_27	7	23	16	0,696	3600,000
p94_24_31	10	23	13	0,565	3600,000
p95_24_27	5	23	18	0,783	3600,000
p96_24_27	4	23	19	0,826	3600,000
p97_24_26	6	23	17	0,739	3600,000
p98_24_29	10	10	0	0,000	559,363
p99_24_27	10	23	13	0,565	3600,000
p100_24_34	7	23	16	0,696	3600,000

A.3. Ramificación y Acotación

A.3.1. Solución inicial lexicográfica ($B\&B_1$)

Small Instances	Cota Inferior	Cota Superior	Gap Absoluto	Gap Relativo	Tiempo (s)
p17_16_24	5	5	0	0,000	1,682
p18_16_21	6	6	0	0,000	0,284
p19_16_19	7	7	0	0,000	0,205
p20_16_18	6	6	0	0,000	7,003
p21_17_20	7	7	0	0,000	14,070
p22_17_19	7	7	0	0,000	4,993
p23_17_23	6	6	0	0,000	4,085
p24_17_29	5	5	0	0,000	1,116
p25_17_20	7	7	0	0,000	2,331
p26_17_19	7	7	0	0,000	3,681
p27_17_19	8	8	0	0,000	1,460
p28_17_18	8	8	0	0,000	6,823
p29_17_18	7	7	0	0,000	3,501
p30_17_19	7	7	0	0,000	1,048
p31_18_21	7	7	0	0,000	2,060
p32_18_20	8	8	0	0,000	2,629
p33_18_21	8	8	0	0,000	23,102
p34_18_21	7	7	0	0,000	10,294
p35_18_19	7	7	0	0,000	8,881

Small Instances	Cota Inferior	Cota Superior	Gap Absoluto	Gap Relativo	Tiempo (s)
p36_18_20	8	8	0	0,000	12,871
p37_18_20	8	8	0	0,000	0,465
p38_18_19	8	8	0	0,000	19,768
p39_18_19	8	8	0	0,000	0,272
p40_18_32	6	6	0	0,000	0,149
p41_19_20	8	8	0	0,000	23,006
p42_19_24	8	8	0	0,000	91,643
p43_19_22	7	7	0	0,000	43,545
p44_19_25	8	8	0	0,000	106,945
p45_19_25	7	7	0	0,000	2,380
p46_19_20	9	9	0	0,000	22,603
p47_19_21	8	8	0	0,000	0,735
p48_19_21	8	8	0	0,000	23,217
p49_19_22	8	8	0	0,000	99,774
p50_19_25	7	7	0	0,000	11,797
p51_20_28	8	8	0	0,000	87,706
p52_20_27	8	8	0	0,000	191,401
p53_20_22	9	9	0	0,000	6,889
p54_20_28	8	8	0	0,000	10,105
p55_20_24	9	9	0	0,000	13,997
p56_20_23	9	9	0	0,000	256,234
p57_20_24	8	8	0	0,000	8,495
p58_20_21	8	8	0	0,000	1215,872
p59_20_23	8	8	0	0,000	155,797
p60_20_22	9	9	0	0,000	31,666
p61_21_22	10	10	0	0,000	181,198
p62_21_30	9	9	0	0,000	68,661
p63_21_42	6	6	0	0,000	114,338
p64_21_22	9	9	0	0,000	27,869
p65_21_24	9	9	0	0,000	539,406
p66_21_28	8	8	0	0,000	39,692
p67_21_22	8	9	1	0,111	3600,000
p68_21_27	8	8	0	0,000	65,251
p69_21_23	9	9	0	0,000	21,277
p70_21_25	9	9	0	0,000	939,552
p71_22_29	1	12	11	0,917	3600,000
p72_22_49	1	8	7	0,875	3600,000
p73_22_29	1	12	11	0,917	3600,000
p74_22_30	8	8	0	0,000	398,180
p75_22_25	9	9	0	0,000	36,575
p76_22_30	8	8	0	0,000	34,209
p77_22_37	8	8	0	0,000	1798,318
p78_22_31	1	11	10	0,909	3600,000
p79_22_29	9	9	0	0,000	824,619
p80_22_30	8	8	0	0,000	38,776
p81_23_46	7	7	0	0,000	399,691
p82_23_24	10	10	0	0,000	839,039
p83_23_24	9	10	1	0,100	3600,000
p84_23_26	9	9	0	0,000	322,235
p85_23_26	9	11	2	0,182	3600,000
p86_23_24	11	13	2	0,154	3600,000
p87_23_30	9	9	0	0,000	1384,611
p88_23_26	9	10	1	0,100	3600,000
p89_23_27	1	12	11	0,917	3600,000
p90_23_35	9	9	0	0,000	931,779
p91_24_33	1	12	11	0,917	3600,000
p92_24_26	10	10	0	0,000	16,414
p93_24_27	12	14	2	0,143	3600,000
p94_24_31	10	10	0	0,000	326,866
p95_24_27	10	10	0	0,000	2946,105
p96_24_27	1	17	16	0,941	3600,000
p97_24_26	1	15	14	0,933	3600,000
p98_24_29	10	10	0	0,000	286,781
p99_24_27	10	14	4	0,286	3600,000
p100_24_34	10	10	0	0,000	197,094

A.3.2. Solución inicial heurística ($B\&B_1 heur$)

Small Instances	Cota Inferior	Cota Superior	Gap Absoluto	Gap Relativo	Tiempo (s)
p17_16_24	5	5	0	0,000	1,607
p18_16_21	6	6	0	0,000	3,446
p19_16_19	7	7	0	0,000	0,186
p20_16_18	6	6	0	0,000	61,525
p21_17_20	7	7	0	0,000	13,585
p22_17_19	7	7	0	0,000	2,932
p23_17_23	6	6	0	0,000	3,301
p24_17_29	5	5	0	0,000	1,165
p25_17_20	7	7	0	0,000	0,117
p26_17_19	7	7	0	0,000	3,312
p27_17_19	8	8	0	0,000	0,331
p28_17_18	8	8	0	0,000	4,811
p29_17_18	7	7	0	0,000	3,196
p30_17_19	7	7	0	0,000	6,989
p31_18_21	7	7	0	0,000	1,344
p32_18_20	8	8	0	0,000	4,607
p33_18_21	8	8	0	0,000	22,203
p34_18_21	7	7	0	0,000	8,769
p35_18_19	7	7	0	0,000	6,129
p36_18_20	8	8	0	0,000	27,925
p37_18_20	8	8	0	0,000	0,321
p38_18_19	8	8	0	0,000	16,014
p39_18_19	8	8	0	0,000	42,494
p40_18_32	6	6	0	0,000	3,100
p41_19_20	8	8	0	0,000	21,989
p42_19_24	8	8	0	0,000	83,341
p43_19_22	7	7	0	0,000	84,723
p44_19_25	8	8	0	0,000	121,025
p45_19_25	7	7	0	0,000	2,021
p46_19_20	9	9	0	0,000	19,991
p47_19_21	8	8	0	0,000	0,524
p48_19_21	8	8	0	0,000	16,433
p49_19_22	8	8	0	0,000	5,936
p50_19_25	7	7	0	0,000	8,125
p51_20_28	8	8	0	0,000	78,186
p52_20_27	8	8	0	0,000	2,669
p53_20_22	9	9	0	0,000	0,959
p54_20_28	8	8	0	0,000	7,773
p55_20_24	9	9	0	0,000	1,857
p56_20_23	9	9	0	0,000	2,605
p57_20_24	8	8	0	0,000	24,197
p58_20_21	8	8	0	0,000	1174,276
p59_20_23	8	8	0	0,000	148,057
p60_20_22	9	9	0	0,000	24,164
p61_21_22	10	10	0	0,000	204,715
p62_21_30	9	9	0	0,000	54,625
p63_21_42	6	6	0	0,000	106,662
p64_21_22	9	9	0	0,000	10,262
p65_21_24	9	9	0	0,000	634,228
p66_21_28	8	8	0	0,000	41,025
p67_21_22	8	9	1	0,111	3600,000
p68_21_27	8	8	0	0,000	55,326
p69_21_23	9	9	0	0,000	18,324
p70_21_25	9	9	0	0,000	656,015
p71_22_29	4	11	7	0,636	3600,000
p72_22_49	6	6	0	0,000	154,031
p73_22_29	9	9	0	0,000	2206,956
p74_22_30	8	8	0	0,000	19,809
p75_22_25	9	9	0	0,000	34,543
p76_22_30	8	8	0	0,000	35,260
p77_22_37	8	8	0	0,000	1106,589
p78_22_31	1	11	10	0,909	3600,000
p79_22_29	9	9	0	0,000	569,916
p80_22_30	8	8	0	0,000	34,362
p81_23_46	7	7	0	0,000	102,944
p82_23_24	10	10	0	0,000	815,540
p83_23_24	9	10	1	0,100	3600,000
p84_23_26	9	9	0	0,000	394,950

Small Instances	Cota Inferior	Cota Superior	Gap Absoluto	Gap Relativo	Tiempo (s)
p85_23_26	9	11	2	0,182	3600,000
p86_23_24	11	13	2	0,154	3600,000
p87_23_30	9	9	0	0,000	1024,374
p88_23_26	9	12	3	0,250	3600,000
p89_23_27	4	13	9	0,692	3600,000
p90_23_35	9	9	0	0,000	349,731
p91_24_33	5	11	6	0,545	3600,000
p92_24_26	10	10	0	0,000	15,951
p93_24_27	11	15	4	0,267	3600,000
p94_24_31	10	10	0	0,000	879,025
p95_24_27	10	10	0	0,000	2569,378
p96_24_27	6	17	11	0,647	3600,000
p97_24_26	10	15	5	0,333	3600,000
p98_24_29	10	10	0	0,000	391,950
p99_24_27	10	15	5	0,333	3600,000
p100_24_34	10	10	0	0,000	297,207

A.4. Ramificación y Acotación con recorrido inverso del árbol de exploración

A.4.1. Solución inicial lexicográfica ($B\&B_2$)

Small Instances	Cota Inferior	Cota Superior	Gap Absoluto	Gap Relativo	Tiempo (s)
p17_16_24	5	5	0	0,000	2,840
p18_16_21	6	6	0	0,000	0,815
p19_16_19	7	7	0	0,000	0,438
p20_16_18	6	6	0	0,000	0,769
p21_17_20	7	7	0	0,000	40,826
p22_17_19	7	7	0	0,000	6,494
p23_17_23	6	6	0	0,000	15,413
p24_17_29	5	5	0	0,000	7,280
p25_17_20	7	7	0	0,000	0,062
p26_17_19	7	7	0	0,000	18,151
p27_17_19	8	8	0	0,000	1,769
p28_17_18	8	8	0	0,000	8,634
p29_17_18	7	7	0	0,000	6,245
p30_17_19	7	7	0	0,000	7,782
p31_18_21	7	7	0	0,000	4,396
p32_18_20	8	8	0	0,000	0,713
p33_18_21	8	8	0	0,000	142,951
p34_18_21	7	7	0	0,000	20,504
p35_18_19	7	7	0	0,000	90,328
p36_18_20	8	8	0	0,000	193,906
p37_18_20	8	8	0	0,000	28,156
p38_18_19	8	8	0	0,000	46,450
p39_18_19	8	8	0	0,000	3,018
p40_18_32	6	6	0	0,000	2,207
p41_19_20	8	8	0	0,000	148,874
p42_19_24	8	8	0	0,000	33,987
p43_19_22	7	7	0	0,000	12,431
p44_19_25	4	17	13	0,765	3600,000
p45_19_25	7	7	0	0,000	2,996
p46_19_20	9	9	0	0,000	81,249
p47_19_21	8	8	0	0,000	0,189
p48_19_21	8	8	0	0,000	55,317
p49_19_22	8	8	0	0,000	31,828
p50_19_25	7	7	0	0,000	25,810
p51_20_28	8	8	0	0,000	126,082
p52_20_27	5	18	13	0,722	3600,000
p53_20_22	9	9	0	0,000	2,067
p54_20_28	8	8	0	0,000	67,422
p55_20_24	9	9	0	0,000	8,168

Small Instances	Cota Inferior	Cota Superior	Gap Absoluto	Gap Relativo	Tiempo (s)
p56_20_23	9	9	0	0,000	50,571
p57_20_24	5	16	11	0,688	3600,000
p58_20_21	8	18	10	0,556	3600,000
p59_20_23	8	8	0	0,000	286,024
p60_20_22	9	9	0	0,000	218,537
p61_21_22	7	19	12	0,632	3600,000
p62_21_30	9	9	0	0,000	124,143
p63_21_42	6	6	0	0,000	75,102
p64_21_22	9	9	0	0,000	34,347
p65_21_24	9	9	0	0,000	460,260
p66_21_28	8	8	0	0,000	90,625
p67_21_22	9	9	0	0,000	556,377
p68_21_27	8	8	0	0,000	153,769
p69_21_23	9	9	0	0,000	114,813
p70_21_25	9	9	0	0,000	977,899
p71_22_29	9	9	0	0,000	253,754
p72_22_49	5	20	15	0,750	3600,000
p73_22_29	7	20	13	0,650	3600,000
p74_22_30	8	8	0	0,000	96,152
p75_22_25	9	9	0	0,000	116,627
p76_22_30	7	20	13	0,650	3600,000
p77_22_37	5	20	15	0,750	3600,000
p78_22_31	8	8	0	0,000	144,052
p79_22_29	9	9	0	0,000	1292,501
p80_22_30	8	8	0	0,000	157,148
p81_23_46	7	7	0	0,000	1086,609
p82_23_24	10	10	0	0,000	3417,679
p83_23_24	8	22	14	0,636	3600,000
p84_23_26	9	9	0	0,000	714,413
p85_23_26	10	22	12	0,545	3600,000
p86_23_24	9	21	12	0,571	3600,000
p87_23_30	8	20	12	0,600	3600,000
p88_23_26	8	21	13	0,619	3600,000
p89_23_27	9	22	13	0,591	3600,000
p90_23_35	9	9	0	0,000	67,214
p91_24_33	10	10	0	0,000	2087,141
p92_24_26	10	10	0	0,000	173,887
p93_24_27	12	22	10	0,455	3600,000
p94_24_31	10	10	0	0,000	2473,496
p95_24_27	10	22	12	0,545	3600,000
p96_24_27	7	23	16	0,696	3600,000
p97_24_26	7	22	15	0,682	3600,000
p98_24_29	9	23	14	0,609	3600,000
p99_24_27	10	23	13	0,565	3600,000
p100_24_34	7	22	15	0,682	3600,000

A.4.2. Solución inicial heurística ($B\&B_2heur$)

Small Instances	Cota Inferior	Cota Superior	Gap Absoluto	Gap Relativo	Tiempo (s)
p17_16_24	5	5	0	0,000	2,814
p18_16_21	6	6	0	0,000	0,838
p19_16_19	7	7	0	0,000	0,480
p20_16_18	6	6	0	0,000	0,794
p21_17_20	7	7	0	0,000	41,556
p22_17_19	7	7	0	0,000	6,544
p23_17_23	6	6	0	0,000	15,701
p24_17_29	5	5	0	0,000	7,496
p25_17_20	7	7	0	0,000	0,074
p26_17_19	7	7	0	0,000	18,554
p27_17_19	8	8	0	0,000	1,911
p28_17_18	8	8	0	0,000	8,902
p29_17_18	7	7	0	0,000	6,327
p30_17_19	7	7	0	0,000	7,930
p31_18_21	7	7	0	0,000	4,492
p32_18_20	8	8	0	0,000	0,745
p33_18_21	8	8	0	0,000	146,285
p34_18_21	7	7	0	0,000	20,231
p35_18_19	7	7	0	0,000	83,038

Small Instances	Cota Inferior	Cota Superior	Gap Absoluto	Gap Relativo	Tiempo (s)
p36_18_20	8	8	0	0,000	223,924
p37_18_20	8	8	0	0,000	30,362
p38_18_19	8	8	0	0,000	47,314
p39_18_19	8	8	0	0,000	3,068
p40_18_32	6	6	0	0,000	2,359
p41_19_20	8	8	0	0,000	149,081
p42_19_24	8	8	0	0,000	32,628
p43_19_22	7	7	0	0,000	12,676
p44_19_25	4	17	13	0,765	3600,000
p45_19_25	7	7	0	0,000	3,065
p46_19_20	9	9	0	0,000	82,751
p47_19_21	8	8	0	0,000	1,252
p48_19_21	8	8	0	0,000	83,442
p49_19_22	8	8	0	0,000	32,033
p50_19_25	7	7	0	0,000	26,038
p51_20_28	8	8	0	0,000	122,606
p52_20_27	4	18	14	0,778	3600,000
p53_20_22	9	9	0	0,000	2,110
p54_20_28	8	8	0	0,000	61,666
p55_20_24	9	9	0	0,000	3,404
p56_20_23	9	9	0	0,000	48,296
p57_20_24	8	8	0	0,000	3146,428
p58_20_21	8	18	10	0,556	3600,000
p59_20_23	8	8	0	0,000	288,690
p60_20_22	9	9	0	0,000	220,308
p61_21_22	7	19	12	0,632	3600,000
p62_21_30	9	9	0	0,000	126,635
p63_21_42	6	6	0	0,000	76,268
p64_21_22	9	9	0	0,000	34,985
p65_21_24	9	9	0	0,000	522,024
p66_21_28	8	8	0	0,000	90,354
p67_21_22	9	9	0	0,000	642,831
p68_21_27	8	8	0	0,000	155,896
p69_21_23	9	9	0	0,000	115,027
p70_21_25	9	9	0	0,000	824,647
p71_22_29	4	20	16	0,800	3600,000
p72_22_49	5	20	15	0,750	3600,000
p73_22_29	7	20	13	0,650	3600,000
p74_22_30	8	8	0	0,000	50,180
p75_22_25	9	9	0	0,000	119,223
p76_22_30	7	20	13	0,650	3600,000
p77_22_37	8	8	0	0,000	3585,657
p78_22_31	8	8	0	0,000	146,700
p79_22_29	9	9	0	0,000	590,325
p80_22_30	8	8	0	0,000	159,427
p81_23_46	7	7	0	0,000	1123,761
p82_23_24	10	10	0	0,000	3522,377
p83_23_24	8	22	14	0,636	3600,000
p84_23_26	9	9	0	0,000	728,299
p85_23_26	10	22	12	0,545	3600,000
p86_23_24	6	21	15	0,714	3600,000
p87_23_30	8	20	12	0,600	3600,000
p88_23_26	8	21	13	0,619	3600,000
p89_23_27	9	22	13	0,591	3600,000
p90_23_35	9	9	0	0,000	68,952
p91_24_33	10	10	0	0,000	2107,051
p92_24_26	10	10	0	0,000	30,340
p93_24_27	12	22	10	0,455	3600,000
p94_24_31	10	10	0	0,000	2122,802
p95_24_27	10	22	12	0,545	3600,000
p96_24_27	7	23	16	0,696	3600,000
p97_24_26	8	22	14	0,636	3600,000
p98_24_29	9	23	14	0,609	3600,000
p99_24_27	10	23	13	0,565	3600,000
p100_24_34	7	22	15	0,682	3600,000

A.5. CPLEX

Small Instances	Cota Inferior	Cota Superior	Gap Absoluto	Gap Relativo	Tiempo (s)
p17_16_24	5	5	0	0	235,201
p18_16_21	6	6	0	0	13,620
p19_16_19	7	7	0	0	0,281
p20_16_18	6	6	0	0	4,228
p21_17_20	7	7	0	0	55,927
p22_17_19	7	7	0	0	248,338
p23_17_23	6	6	0	0	209,603
p24_17_24	5	5	0	0	3,230
p25_17_24	7	7	0	0	0,641
p26_17_24	7	7	0	0	61,417
p27_17_24	8	8	0	0	810,500
p28_17_24	8	8	0	0	2,402
p29_17_24	7	7	0	0	33,993
p30_17_24	7	7	0	0	61,637
p31_18_24	7	7	0	0	66,114
p32_18_24	8	8	0	0	4,104
p33_18_24	8	8	0	0	704,591
p34_18_24	7	7	0	0	171,070
p35_18_24	7	7	0	0	3513,642
p36_18_24	8	8	0	0	1101,097
p37_18_24	8	8	0	0	2,372
p38_18_24	8	8	0	0	1762,336
p39_18_24	8	9	1	0,111	3600
p40_18_24	6	6	0	0	2,524
p41_19_24	8	8	0	0	1218,748
p42_19_24	8	8	0	0	31,267
p43_19_24	7	7	0	0	126,864
p44_19_24	8	8	0	0	15,580
p45_19_24	7	7	0	0	1834,944
p46_19_24	9	9	0	0	2,781
p47_19_24	8	8	0	0	1,027
p48_19_24	8	8	0	0	350,700
p49_19_24	8	8	0	0	3,702
p50_19_24	7	7	0	0	17,795
p51_16_24	8	8	0	0	1346,619
p52_16_24	8	8	0	0	148,736
p53_16_24	9	10	1	0,100	3600
p54_16_24	8	8	0	0	25,801
p55_16_24	9	9	0	0	1,502
p56_16_24	9	9	0	0	213,918
p57_16_24	8	8	0	0	34,445
p58_16_24	8	9	1	0,111	3600
p59_16_24	8	8	0	0	698,734
p60_16_24	9	10	1	0,100	3600
p61_16_24	10	10	0	0	2124,210
p62_16_24	9	10	1	0,100	3600
p63_16_24	6	7	1	0,143	3600
p64_16_24	9	9	0	0	81,605
p65_16_24	9	9	0	0	153,630
p66_16_24	8	8	0	0	520,791
p67_16_24	8	10	2	0,200	3600
p68_16_24	8	8	0	0	2248,152
p69_16_24	9	9	0	0	5,912
p70_16_24	9	10	1	0,100	3600
p71_16_24	9	9	0	0	35,834
p72_16_24	6	7	1	0,143	3600
p73_16_24	9	9	0	0	6,288
p74_16_24	8	8	0	0	7,473
p75_16_24	9	9	0	0	3,948
p76_16_24	8	8	0	0	84,366
p77_16_24	8	8	0	0	18,019
p78_16_24	8	8	0	0	86,814
p79_16_24	9	9	0	0	28,081
p80_16_24	8	8	0	0	152,101
p81_16_24	7	9	2	0,222	3600
p82_16_24	10	11	1	0,091	3600
p83_16_24	9	10	1	0,100	3600
p84_16_24	9	10	1	0,100	3600

Small Instances	Cota Inferior	Cota Superior	Gap Absoluto	Gap Relativo	Tiempo (s)
p85_16_24	10	11	1	0,091	3600
p86_16_24	11	12	1	0,083	3600
p87_16_24	9	9	0	0	251,363
p88_16_24	9	12	3	0,250	3600
p89_16_24	9	10	1	0,100	3600
p90_16_24	9	9	0	0	17,503
p91_16_24	10	10	0	0	74,006
p92_16_24	10	12	2	0,167	3600
p93_16_24	12	12	0	0	48,848
p94_16_24	10	10	0	0	16,624
p95_16_24	10	12	2	0,167	3600
p96_16_24	10	11	1	0,091	3600
p97_16_24	10	12	2	0,167	3600
p98_16_24	10	10	0	0	4,646
p99_16_24	10	13	3	0,231	3600
p100_16_24	10	10	0	0	10,624

Anexo B

Contenido del CD

- Código fuente
- Documentación
  - Memoria
  - Presentación
- Resultados experimentales
  - Ejecución de los algoritmos
  - Datos tabulados
- Conjunto de instancias de prueba
  - SmallInstances
  - Instancias aleatorias

Bibliografía

[1] Solver CPLEX. `http://www.ilog.com/products/cplex/`.

[2] Sistema de documentación LaTeX. `http://www.latex-project.org/`.

[3] M. Fowler and K. Beck. *Planning extreme programming.* Pearson Addisson-Wesley, 2001.

[4] GNU. `http://www.gnu.org`.

[5] Eclipse IDE. `http://www.eclipse.org`.

[6] J. Leung, O. Vornberger, and J. Witthoff. Some variants of the bandwidth minimization problem. *SIAM Journal on Computing*, 13:650–667, 1984.

[7] Sistema Operativo Linux. `http://www.linux.org`.

[8] Ubuntu Linux. `http://www.ubuntu.com`.

[9] R. Martí, V. Campos, and E. Piñana. Branch and bound for the matrix bandwidth minimization. *European Journal of Operational Research*, 186:513–528, 2008.

[10] M.G.C. Resende, A. Duarte, R. Martí, and R. Silva. Grasp with evolutionary path-relinking for the antibandwidth problem, July 13-16 2009. VIII Metaheuristics International Conference (MIC 2009), Hamburg.

[11] J. Rumbaugh, I. Jacobson, and G. Booch. *El proceso unificado de desarrollo de software.* Pearson Addisson-Wesley, 2000.

[12] M.G. Piattini Velthuis. *Análisis y diseño de aplicaciones informáticas de gestión. Una perspectiva de ingeniería del software.* Editorial Ra-Ma, 2003.