INSTITUTO TECNOLÓGICO DE CIUDAD MADERO

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

TESIS

METODOS EXACTOS PARA EL PROBLEMA DEL SUMCUT DE UN GRAFO CONEXO NO DIRIGIDO

Para obtener el grado de:
Maestro en Ciencias Computacionales

Presenta:
Yazmin Gómez Rojas

Director de tesis:
Dr. Juan Javier González Barbosa

Co-Director de tesis:
Dr. Héctor J. Fraire Huacuja

Ciudad Madero, Tamaulipas Octubre 2014

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO
Instituto Tecnológico de Ciudad Madero

"2014, Año de Octavio Paz"

Cd. Madero, Tamps; a 6 de Octubre de 2014.

OFICIO No.: U5.230/14
AREA: DIVISIÓN DE ESTUDIOS
DE POSGRADO E INVESTIGACIÓN
ASUNTO: AUTORIZACIÓN DE IMPRESIÓN DE TESIS

ING. YAZMIN GÓMEZ ROJAS
NO. DE CONTROL G07070938
P R E S E N T E

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

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

Se acordó autorizar la impresión de su tesis titulada:

"MÉTODOS EXACTOS PARA EL PROBLEMA DE SUMCUT DE UN GRAFO CONEXO NO DIRIGIDO"

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

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

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

Ma. Yolanda Chávez Cinco

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

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

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

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

Dedicatoria

Dedico este trabajo a las personas más importantes en mi vida: a mis padres.

A mi papá *porque es mi ejemplo de vida, mi súper héroe, por todo el apoyo, soporte y consejos que me ha dado.*

A mi mamá *porque siempre me ha apoyado en cada momento de mi vida, por cada consejo que me dio,*

Espero algún día regresar aunque sea un poco de lo mucho que me han dado.

Agradecimiento

Agradezco a toda mi familia por todo el apoyo que me dieron durante esta etapa de mi vida, a mis padres porque me apoyaron en cada una de las decisiones que he tomado y a mis hermanas que siempre han estado ahí cuando las he necesitado. Nunca podrs describir con palabras lo mucho que les debo.

Gracias al Dr. Juan Javier González Barbosa por su apoyo y consejos que me dio durante este proceso que me permitieron seguir adelante. Mi agradecimiento más sincero para usted por todo.

Gracias al Dr. Héctor J. Fraire Huacuja por todo su apoyo, consejos y guiarme durante este proceso. Mi más sentido aprecio y agradecimiento por todo.

Gracias a mis compañeros Fanny Maldonado, Rafael Ortega, Eduardo del Ángel, Oscar Camacho y Javier Rangel por acompañarme durante este proceso y por lo momentos tan divertidos que pase a su lado.

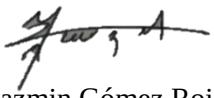
Sobre todo gracias a Dios por todo lo que me ha dado hasta el día de hoy y porque en ningún momento de dejaste sola a pesar de cualquier adversidad.

Declaración de originalidad

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

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

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



(Ing. Yazmin Gómez Rojas)

INDICE GENERAL

Índice de Tablas

Índice de Figuras

Resumen

En este trabajo de investigación aborda el problema de SUMCUT por medio de métodos exactos. El SUMCUT es un problema NP-Completo [Yuan, 1998], el cual consiste en minimizar la suma de los cortes de ancho de un grafo conexo; este problema tiene aplicaciones en la genética y en la arqueología [Karp, 1993] [Kendall, 1993].

La aportación de este trabajo consiste en el desarrollo de 4 modelos de programación lineal entera y 2 métodos exactos basados en ramificación y acotamiento, ya que hasta el momento no se han publicado ningún método exacto que lo resuelva.

Además se realizó una heurística para crear una solución inicial, la cual permitiera trabajar con las instancias con un gran número de nodos. Para validar la eficiencia de la heurística se realizó una metaheurística GRASP.

CAPITULO 1 Introducción

En el presente trabajo de investigación se aborda el Problema de Suma de Cortes (SUMCUT Problem, SCP por sus siglas en inglés), el cual es un problema de optimización combinatoria que consiste en minimizar la suma de los cortes de las aristas de un grafo ordenado en forma lineal. Demostrando en [Yuan, 1998] que es un problema NP-completo.

Este problema tiene aplicación en la genética, específicamente en el Proyecto del Genoma Humano [Karp, 1993]. El objetivo de dicho proyecto es la secuencia del ADN de los humanos así como el de otras especies con el objetivo de obtener la información genética que lo contiene. Otra aplicación que tiene es en la arqueología donde es necesario organizar diferentes artefactos (fósiles, herramientas, joyas, etc.) según un orden determinado [Kendall, 1993]. A este proceso se le denomina seriación, y consiste en situar en orden cronológico diferentes artefactos de la misma cultura utilizando un método de establecimiento de la fecha relativo.

En este trabajo se desarrollaron métodos exactos para resolver el problema del SUMCUT. Estos métodos constan de 4 modelos de programación lineal entera, los cuales son: un modelo compacto, un modelo aplicando la técnica de linealización tradicional de [Fortet, 1959], un modelo utilizando la técnica de linealización propuesta por Leo Lieberti [Liberti, 2007] y un modelo basado en conjuntos. Además de realizar 2 métodos basados en ramificación y acotamiento.

1.1 Objetivos de proyecto.

1.1.1 Objetivo General.

Desarrollar un método de solución exacta para el problema SUMCUT basado en un modelo de programación lineal entera y un método de tipo ramificación y acotamiento, que aproveche la estructura del problema.

1.1.2 Objetivos específicos.

- Revisar la literatura relacionada con los métodos de solución del problema SUMCUT.
- Revisar el marco teórico relacionado con el problema y los métodos de solución exactos.
- Diseñar e implementar un método de solución basado en un modelo de programación lineal entera utilizando CPLEX.
- Diseñar e implementar un método de solución basado en ramificación y acotamiento.
- Evaluar experimentalmente los métodos desarrollados.

1.2 Justificación.

Como se señaló previamente el problema del SUMCUT tiene importantes aplicaciones en las áreas de la genética y arqueología [Karp, 1993], [Kendall, 1993].

El problema pertenece a la categoría de los problemas de complejidad NP-completos [Yuan, 1998], lo que significa que no se conoce un algoritmo determinista de tiempo polinomial que lo resuelva de manera exacta. La única posibilidad de resolver exactamente las instancias de tamaño medio y grande del problema es utilizar un algoritmo aleatorio de tiempo polinomial o un algoritmo aproximado.

Al revisar el estado del arte se encontró que para este problema no existe una formulación de programación lineal entera, ni métodos exactos basados en el enfoque de ramificación y acotamiento. Hasta el momento solo se han reportado un reducido número de algoritmos aproximados para este problema.

Debido a lo anterior en este trabajo se propone desarrollar métodos exactos de solución del problema del SUMCUT.

1.3 Alcances.

- Se desarrollará una formulación de programación lineal entera del problema.
- Se desarrollará un método exacto basado en la formulación de programación lineal entera.
- Se desarrollará un método exacto basado en el enfoque de ramificación y acotamiento.

1.4 Limitaciones.

- Para la solución del modelo de programación lineal entera se utilizará el optimizador CPLEX.
- El desempeño de los métodos de solución se evaluará experimentalmente utilizando instancias asociadas a redes de tipo *scale free* e instancias estándar.
- El proyecto se desarrollará utilizando el lenguaje de programación Java.

1.5 Organización del documento.

En el Capítulo 2 se muestra información teórica para mayor comprensión de la complejidad del problema, así como información del método por el que se va a resolver el problema abordado en esta tesis.

En el Capítulo 3 se muestran los diferentes trabajos que han realizado otros autores para resolver el problema del SUMCUT.

En el Capítulo 4 se ve una descripción del problema del SUMCUT, así como el tipo de complejidad que tiene.

En el Capítulo 5 se muestran los diferentes métodos que se desarrollaron en esta tesis para resolver el problema del SUMCUT.

En el Capítulo 6 se muestran los resultados obtenidos con los métodos desarrollados que se vieron en el Capítulo 5.

Ultimo en el Capítulo 7 se encuentra las conclusiones obtenidas sobre este trabajo, además de los trabajos a futuro.

CAPITULO 2 Marco teórico

A continuación se definirán una serie de conceptos empleados en las secciones posteriores de esta tesis.

2.1 Conceptos de Optimización.

En los siguientes subtema se verán algunos de los conceptos de optimización más relevantes para esta tesis.

2.1.1 Problema de Optimización

Un problema de optimización es un problema en el que hay varias posibles soluciones y alguna forma clara de comparación entre ellas, de manera que existe si y solo si se dispone un conjunto de soluciones candidatas diferentes que pueden ser comparadas. "Desde un punto de vista matemático, un problema de optimización P se formula como un 3-tupla $P = (f, SS, F)$, definida de la siguiente manera:

$$P = \begin{array}{l} opt: \ f\ x\ , \quad Función\ Objetivo \\ s.a., \\ \quad x \in F \subset SS \quad Restricciones \end{array}$$

donde f es la función a optimizar (maximizar o minimizar), F es el conjunto de soluciones factible y SS es el espacio de soluciones" [Duarte, 2007]..

Este tipo de problemas se pueden dividir en dos categorías: los que la solución está codificada mediante valores reales y aquéllos cuyas soluciones están codificadas por valores enteros. En esta segunda categoría se encuentran un tipo particular de problemas denominados *problemas de optimización combinatoria.*

"El ***problema de optimización combinatoria*** consiste en encontrar un objeto entre un conjunto fijo de posibilidades. Este objeto suele ser un número natural, una permutación o una estructura de grafo" [Papadimitriou, 1998].

Los problemas combinatorios presentan como particularidad que siempre existe un algoritmo exacto que permite obtener la solución óptima. Este método consiste en la exploración de forma exhaustiva del conjunto de soluciones (enumeraciones). Este tipo de problemas se caracterizan por tener espacios de soluciones con una cardinalidad muy elevada [Duarte, 2007].

Existen tres conceptos básicos que se pueden encontrar en la resolución algorítmica de problemas de optimización combinatoria. Independientemente de la técnica que se utilice, se debe especificar:

- **Representación:** se encarga de codificar las soluciones factibles para su manipulación. Determina el tamaño del espacio de búsqueda (*SS*) de cada problema.
- **Objetivo:** es un predicado matemático que expresa la tarea que se tiene que realizar.
- **Función de evaluación:** permite asociar a cada solución factible un valor que determina su calidad.

2.1.2 Vecindad y Óptimos Locales

"Dada una solución $x \in SS$, la vecindad $N(x)$ de la solución es un subconjunto del espacio de soluciones que contiene soluciones que están *"próximas"* de la solución considerada" [Duarte, 2007]. En la figura 2.1 se muestra una representación gráfica de la vecindad de una determinada solución x [Duarte, 2007].

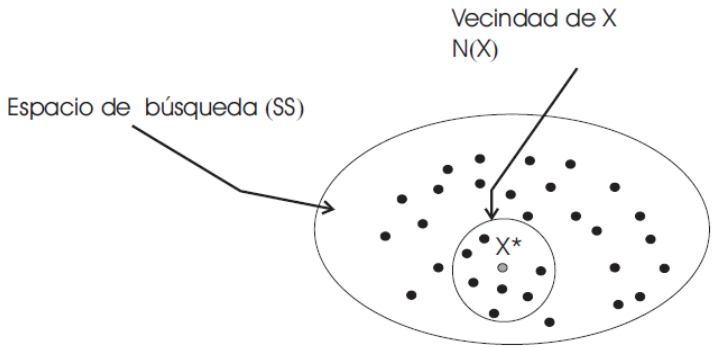


Figura 2.1 Vecindad de una solución x.

La cercanía de dos soluciones del espacio de búsqueda se puede definir de varias formas [Michalewicz, 2000]:

- Dado un espacio de búsqueda SS para un problema concreto, se puede definir una función distancia $dist(x, y)$ entre cualquier par de puntos $x, y \in SS$ como sigue:

$$dist: SS \times SS \rightarrow \mathbb{R}$$

  A partir de esta función se puede definir la vecindad $N\ x\ \subseteq SS$ de la solución x como:

$$N\ x\ =\ y \in SS: \qquad dist\ x, y\ \leq \varepsilon$$

  para algún $\varepsilon \geq 0$. La función distancia descrita en la aplicación anterior depende del problema que se quiera resolver.

- Dado un espacio de búsqueda SS para un problema concreto, se puede definir una función N del espacio de soluciones como sigue:

$$N: SS \rightarrow 2^{SS}$$

  donde esta función define una vecindad $N(x)$ para cada punto $x \in SS$.

En la vecindad de una solución se encuentran todas aquellas soluciones *"cercanas"* de forma que, dada una solución $x \in N(x)$, cada solución de su vecindad $y \in N(x)$ puede alcanzarse directamente desde x mediante una sola operación o movimiento.

De acuerdo a [Duarte, 2007] se puede definir respectivamente el *óptimo global* y el *óptimo local* de la siguiente manera:

- **Óptimo Global:**

  Dado un problema de optimización f, SS, F se dice que una solución factible $x \in F \subseteq SS$ es un óptimo (máximo) global si:

$$\forall y \in F \quad f(x) \geq f(y)$$

- **Óptimo Local:**

  Dado un problema de optimización $P = (f, SS, F)$ y una estructura de vecindad *N*, se dice que una solución factible $x \in F \subseteq SS$ es un óptimo (máximo) local con respecto a *N* si:

$$\forall y \in N(x) \quad f(x) \geq f(y)$$

Estas definiciones de pueden particularizar para problemas de minimización cambiando "≥" por "≤". En la figura 2.2 se muestra gráficamente el máximo global, el máximo local y la vecindad, para una función objetivo dada.

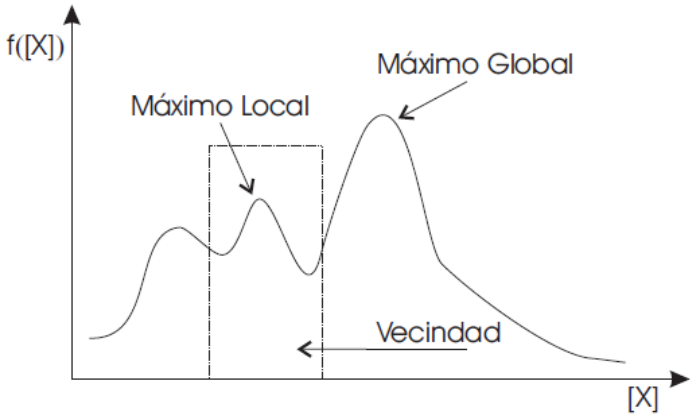


Figura 2.2 Función objetivo con el máximo global, máximo local y vecindad [Duarte, 2007].

2.3 Problema de Decisión.

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

2.4 Complejidad Computacional.

Los problemas pueden caracterizarse dependiendo de la dificultad de su resolución por un ordenador. Se ha definido varias clases de problemas, las cuales son las clases P, NP, NP-Completo y NP-duro [Duarte, 2007].

- **Clase P:**

  Se dice que un problema se puede resolver en tiempo polinomial cuando el tiempo de ejecución de un algoritmo que lo resuelve se puede relacionar con el tamaño de la entrada con una formula polinomial. A estos problemas se les denominan P. Se considera que los problemas P se pueden resolver en un tiempo de ejecución razonable para la información actual [Duarte, 2007].

- **Clase NP:**

  Es el conjunto de todos los problemas de decisión que se pueden verificar por algoritmos no deterministas en tiempo polinomial. Por verificar se entiende que se puede generar en tiempo polinomial una solución candidata con un algoritmo no determinista y que se puede verificar su factibilidad en tiempo polinomial con un algoritmo determinista.

  Para para probar que un problema de decisión pertenece a la clase NP se debe:

- Definir una estructura de datos para representar las soluciones candidatas.
- Construir un algoritmo aleatorio para generar una solución candidata.
- Construir un algoritmo determinista para verificar que una solución candidata cumple las condiciones especificadas en el problema.

Como cada problema de decisión resuelto por un algoritmo determinista de tiempo polinomial es también resuelto por un algoritmo no determinista de tiempo polinomial, se tiene entonces que $P \subseteq NP$ [Garey, 19779].

- **Clase NP-completo:**

Esta clase es un subconjunto de la clase NP de tal forma que todo problema NP se pueda transformar a uno de la clase NP-completo [Garey, 19779]. Para probar que un problema de decisión π pertenece a la clase NP-completo se demuestra de la siguiente manera:

- Mostrar que $\pi \in NP$
- $\forall \pi' \in NPC$
- $\pi' \leq_p \pi$

- **Clase NP-duro:**

Sea un problema de decisión π, sea NP o no, que podamos transformar en un problema NP-completo y tenga la propiedad de que no se pueda resolver en tiempo polinomial, podremos decir que el problema es NP-duro.

En la figura 2.3 se muestra la relación entre los problemas P, NP, NP-completo y NP-duro [Duarte, 2007].

Figura 2.3 Relación entre los problemas P, NP, NP-completo y NP-duro.

2.5 Métodos de Optimización

Existen diferentes métodos de optimización para la resolución de problemas, a continuación se definen.

2.5.1 Métodos Exactos

Los métodos exactos para la resolución de problemas se han aplicado con éxito a una cantidad elevada de problemas. Estos métodos resuelven problemas que pertenecen a la clase P de forma óptima y en tiempo razonable. En cambio para los problemas de la clase NP no se conocen algoritmos exactos con tiempos de convergencia en tiempo polinomial, esto quiere decir, aunque existe un algoritmo que encuentre la solución exacta al problema, tardaría tanto tiempo en encontrarla [Duarte, 2007].

Algunos de estos métodos son:

- **Ramificación y acotamiento (*BAB por sus siglas en Ingles*):**

  "Este método es un enfoque casi enumerativo para resolver el problema que será aplicada para una variedad de problemas combinatorios. Esto es bastante eficiente para problemas de tamaño modesto, y la metodología general forma una parte importante del conjunto de métodos de solución para la clase general de problemas de programación lineal entera" [Ignizio, 1994].

  La idea básica de este método es particionar un problema dado en un número de subproblemas. Esta idea propone establecer subproblemas que son más fáciles de resolver que el problema original, por su tamaño menor o una estructura más susceptible. La figura 2.4 muestra la estructura del método basado en ramificación y acotamiento [Ignizio, 1994].

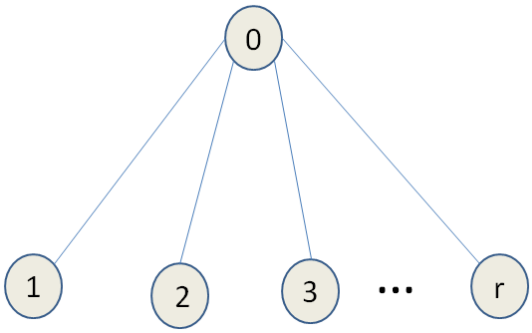


Figura 2.4 Ramificación y acotamiento.

- **Enumeración Implícita:**

Es una técnica que es usualmente aplicada a problemas de programación entera cero-uno. Esta técnica es similar al método basado en ramificación y acotamiento; sin embargo, las reglas de ramificación, restricción y acotamiento son simplificadas y refinadas porque cada variable entera puede tomar solo valores cero o uno.

Los problemas de programación entera cero-uno tienen un número finito de puntos factibles, 2^n puntos enteros factibles, donde n es el número de variables cero-uno. En la figura 2.5 muestra un estructura del método de enumeración implícita [Ignizio, 1994].

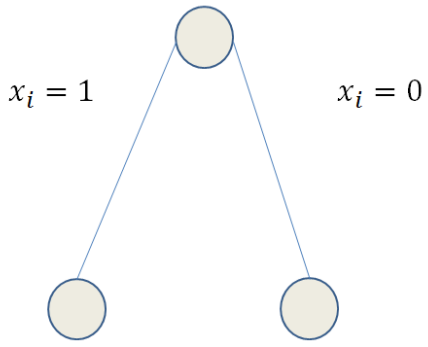


Figura 2.5 Ramificación con variables binarias.

- **Planos de corte:**

  El objetivo de este método es construir iterativamente el casco convexo (es la región del conjunto de las soluciones factibles) en la vecindad de la solución óptima entera. La solución óptima de un problema de programación entera se puede determinar resolviendo un problema de programación lineal única en la que el casco convexo es usado como región factible.

  Ya que los puntos de los extremos del casco convexo corresponden a las soluciones enteras. Se hace de una manera sistemática introduciendo restricciones adicionales que cortan porciones de la región factible excluyendo algunos puntos enteros factibles [Ignizio, 1994].

2.5.2 Métodos Heurísticos.

Actualmente existen dos interpretaciones posibles para el término heurística. La primera de ellas concibe las heurísticas como un procedimiento para resolver problemas. La segunda interpretación de heurística entiende que éstas son una función que permite evaluar la bondad de un movimiento, estado, elemento o solución [Duarte, 2007].

A continuación se muestra una clasificación de las heurísticas de acuerdo a [Duarte, 2007]:

1. **Métodos constructivos:** Procedimientos que son capaces de construir una solución a un problema dado. La forma de construir la solución depende de la estrategia seguida. Las más comunes son las siguientes:

   - *Estrategia voraz:* Partiendo de una semilla, se va construyendo paso a paso una solución factible.

   - *Estrategia de descomposición:* Se divide sistemáticamente el problema en subproblemas más pequeños.

- *Métodos de reducción:* Identifican características que contienen las soluciones buenas conocidas y se asume que la solución óptima también las tendrá.

- *Métodos de manipulación del modelo:* Consiste en simplificar el modelo del problema original para obtener una solución al problema simplificado.

2. **Métodos de búsqueda:** Parten de una solución factible dada y a partir de ella intentan mejorarla.

    - *Estrategia de búsqueda local 1:* Parte de una solución factible y la mejora progresivamente. Para ello examina su vecindad y selecciona el primer movimiento que produce una mejora en la solución actual (*first improvement*).

    - *Estrategia de búsqueda local 2:* Parte de una solución factible que la mejora progresivamente. Par ello examina su vecindad y todos los posibles movimientos seleccionando el mejor movimiento de todos los posibles (*best improvement*)

    - *Estrategia aleatorizada:* Para una solución factible dada y una vecindad asociada a esa solución, se seleccionan aleatoriamente soluciones vecinas de esa vecindad.

2.5.3 Métodos Metaheurísticos

Las siguientes definiciones aportadas por diferentes investigadores describen sus principales propiedades.

- El termino metaheurística o meta-heurística fue acuñado por F. Glover en el año 1986. Con este término, pretendía definir un "*procedimiento maestro de alto nivel*

que guía y modifica otras heurísticas para explorar soluciones más allá de la simple optimalidad local" [Glover et al, 1979].

- Las metaheurísticas son una clase de métodos aproximados que están diseñados para resolver problemas difíciles de optimización combinatoria, en los que los heurísticos clásicos no son efectivos. Las metaheurísticas proporcionan un marco general para crear nuevos algoritmos híbridos combinando diferentes conceptos derivados de la inteligencia artificial, la evolución biológica y los procedimientos estadísticos [Kelly,1996].

A continuación se muestra una clasificación de las metaheurísticas de [Blum, 2003] .

- **Metaheurísticas Trayectoriales:**
  - *Búsqueda Local:* la búsqueda local consiste en buscar una mejor solución que la solución actual.
  - *Recocido Simulado:* la idea principal de esta metaheurística consiste en permitir soluciones de peor calidad que la actual con el fin de escapar de los mínimos locales.
  - *Búsqueda Tabú:* usa una búsqueda histórica la cual consiste en no estancarse en un mínimo local, además de implementar en estrategia de búsqueda.
  - *GRASP:* está compuesta de dos fases: construcción de la solución y mejora de la solución.
  - *Búsqueda Local en Vecindad Variable:* aplica de forma explícita una estrategia basada en un cambio dinámico de las estructuras de la vecindad.
  - *Búsqueda Local Iterada:* aplica una búsqueda local a la solución inicial hasta encontrar un óptimo local, entonces perturba la solución y reinicia la búsqueda local.
- **Metaheurísticas Poblacionales:**
  - *Computación Evolutiva:* son algoritmos inspirados en la naturaleza. Estos algoritmos utilizan operadores llamados cruza y mutación.

- *Scatter Search y Path Relinking:* es una estrategia que genera un conjunto de soluciones de un conjunto de referencia de soluciones el cual contiene soluciones factibles.
- *Colonia de Hormigas:* esta estrategia esa basada en como las hormigas utilizan la feromona para guiar a las demás hormigas y encontrar en nuestro caso una solución óptima.

2.6 CPLEX

CPLEX proporciona algoritmos matemáticos robustos, permite resolver problemas con millones de variables y restricciones. Este optimizador resuelve problemas de programación lineal entera utilizando variables ya se primaras o duales del método simplex, problemas de programación cuadrática convexos o no convexos

Sus características:

- **Algoritmo automático y dinámico de control de parámetros**
- **Rápido, reinicios automáticos**
- **Una variedad de opciones de modificaciones del problema**
- **Una amplia variedad de opciones de entrada/salida**
- **Mensaje de la información y análisis de la solución**

2.7 Modelo de Programación Lineal

Programación lineal es un término usado para indicar un método de solución para modelos matemáticos lineales o simplemente describe la práctica de modelado y solución.

Un modelo de programación lineal es representado por una función que es determinada como función objetivo, el resto del modelo consiste de funciones las cuales representan las restricciones del problema [Ignizio, 1994]. Como se muestra en el siguiente ejemplo:

$$\max z = 7x_1 + 10x_2$$

Sujeto a:

$$x_1 + x_2 \leq 10$$
$$3x_1 - x_2 \geq 4$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$

Figura 2.6 Modelo de Programación Lineal

Existen modelos de programación lineal que tienen más de una función objetivo a ellos se les llama modelos de programación lineal multiobjetivo [Ignizio, 1994].

CAPITULO 3 Trabajos relacionados

En este capítulo se describen los trabajos más relevantes y recientes relacionados con el problema de SUMCUT.

[Lewis, 1993] desarrolla la metaheurística de Recocido Simulado mediante movimientos de intercambios, para resolver el problema de Profile(el cual es equivalente al SUMCUT deacuerdo con [Ravi, 1991]). Primero reordena la matriz con el algoritmo de Cuthill-McKee con el que se marcan los vértices de la matriz. El movimiento de intercambio consta en tomar dos posiciones, i y j e intercambiar las filas i y las columnas j de la matriz, recalculando todos el costo del Profile enfatizando que si la distancia entre i y j es muy grande puede tomar bastante tiempo computacional en realizar los movimientos de intercambio.

[Sánchez-Oro, 2011] propone el uso de la metaheurística GRASP para resolver el problema del SUMCUT, evaluando dos tipos de construcciones: una implementa la construcción típica del GRASP donde se evalúa inicialmente a cada candidato en base a una función voraz para construir la lista de candidatos restringida (RCL), seleccionando un elemento de forma aleatoria de la RCL, y la segunda se basa en una estrategia donde la selección aleatoria y voraz se intercambian. Su búsqueda local tiene asociado dos tipos de movimientos: intercambio e inserción. La debilidad de este trabajo es que recalcula el costo completo del SUMCUT cada vez que se construye una nueva solución.

[Sánchez-Oro, 2012] en este artículo se desarrollan dos versiones de la metaheurística *Path Relinking*: *Static Path Relinking* (SPR) y *Dynamic Path Relinking* (DRP) para resolver el problema del SUMCUT. El primero crea un conjunto de referencia (*RefSet*), para ello se ejecuta el procedimiento GRASP de [Sánchez-Oro, 2011] para generar un conjunto de p soluciones. Del conjunto de p se seleccionan b soluciones donde $b/2$ se seleccionan de acuerdo a su calidad y $b/2$ se seleccionan de acuerdo a su diversidad para el *RefSet.* Una vez generado el *RefSet* el procedimiento del SPR genera un camino para

cada par de soluciones del *RefSet.* El algoritmo termina cuando todos los pares de soluciones se han combinado.

El segundo genera el *RefSet* ejecutando el procedimiento GRASP y seleccionando del mismo las primeras *b* soluciones diferentes. Se selecciona una solución de forma aleatoria del *RefSet*, que será utilizada como solución guía. El algoritmo genera un camino desde la solución inicial a la solución guía, almacenado la mejor solución encontrada en el camino. Finalmente, el algoritmo DPR comprueba si la nueva solución cumple una serie de características que le permitan entrar a formar parte de *RefSet*, si la solución es admitida en el *RefSet*, sustituirá a la solución más parecida del subconjunto de soluciones peores.

[Zamarron, 2013] en este trabajo se desarrollaron tres estrategias de búsqueda local para resolver el problema del SUMCUT, las cuales son: first, best y optima, estas se implementaron en un algoritmo de Búsqueda Local Iterada y una Búsqueda Local Iterada Celular

Del análisis realizado en el estado del arte se concluye que para el problema de SUMCUT solo se han realizado trabajos basados en metaheurísticas, por lo tanto de decidió aportar en este trabajo métodos exactos.

En la siguiente tabla muestra los elementos que contienen los algoritmos mencionados anteriormente.

Tabla 3.1 Algoritmos del Estado del Arte.

Autor	Metodología	Año
Lewis	Recosido Simulado	1993
Sánchez-Oro	GRASP	2011
Sánchez-Oro	GRASP con Path Relinking (*Static Path Relinking* (SPR) y *Dynamic Path Relinking* (DRP))	2012
Zamarron	Búsqueda Local Iterada y Búsqueda Local Iterada Celular	2013

CAPITULO 4 Descripción del problema de Investigación

4.1 Problema del SUMCUT

El problema del SUMCUT consiste en la suma de todos los CutWidth, debido a esto se describe primero el problema del CutWidth.

Dado $G = (V, E)$ con $n = |V|$ y $m = |E|$, un etiquetado o arreglo lineal π de G asigna los números enteros $1,2,\dots,n$ a los vértices del conjunto V, donde cada vértice recibe una etiqueta distinta.

El CutWidth (CW) de un vértice v con respecto al arreglo lineal π $CW_\pi\ v$, es el número de aristas $u,w \in E$ que satisfacen la condición $\pi\ u \leq \pi\ v < \pi\ w$ [Lopez, 2012].

El CW del grafo G con respecto al arreglo lineal π, $CW_\pi\ G$, es el máximo CW de todos sus vértices:

$$CW_\pi = max_{v \in V} CW_\pi\ v \tag{4.1}$$

Dado un grafo no dirigido, a cada nodo del grafo se le etiqueta con un número identificador como se muestra en el grafo de la figura 4.1

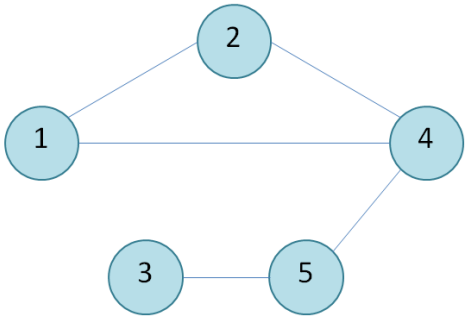


Figura 4.1 Grafo.

Se ordenan los nodos del grafo de forma horizontal basándose en la etiqueta de los nodos. Una vez hecho esto, se coloca una línea punteada entre dos nodos, como se muestra en la figura 4.2.

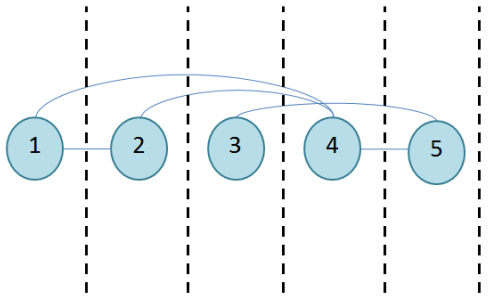


Figura 4.2 Etiquetado o arreglo lineal del grafo.

El arreglo puede ser representado por su matriz de adyacencia, mostrada en la figura 4.3, la cual es simétrica, debido que se trata de un grafo no dirigido.

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

Figura 4.3 Matriz de adyacencia del grafo.

El valor de la función objetivo del CW de v_1, donde v_1 es la primera posición de la permutación, está dado por:

$$CW\ v_1 = \sum_{j=2}^{n} a\ v_1\ v_j \qquad (4.2)$$

El valor del CW en v_1 puede ser obtenido con la ecuación de la siguiente manera:

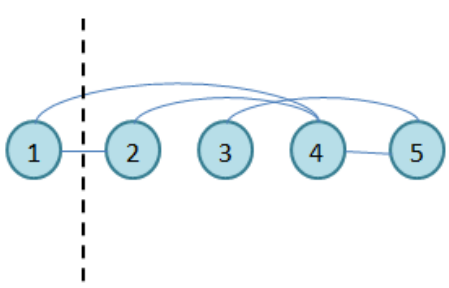
$$CW\ v_1 = \sum_{j=2}^{n} a\ v_1\ v_j$$

$$= a\ v_1\ v_2 + a\ v_1\ v_3 + a\ v_1\ v_4 + a\ v_1\ v_5$$

$$= 1 + 0 + 1 + 0$$

$$= 2$$

Tal como se puede ver en la figura 4.4, en que al efectuar el corte en la primera posición, se afecta en dos arcos del grafo.



	1	2	3	4	5
1	0	1	0	1	0
2	1	0	0	1	0
3	0	0	0	0	1
4	1	1	0	0	1
5	0	0	1	1	0

Figura 4.4 CutWidth del nodo $\boldsymbol{v_1}$.

El valor de la función objetivo del CW de v_k, donde $1 < k < n$, de la permutación, puede ser calculada por:

$$CW\ v_k = CW\ v_{k-1} + \sum_{j=k+1}^{n} a\ v_k\ v_j - \sum_{j=1}^{k-1} a\ v_j\ v_k \qquad (4.3)$$

El valor de CW puede ser calculado con la ecuación 4.3 de la siguiente manera:

$$CW\ v_2 = CW\ v_1 + \sum_{j=3}^{n} a\ v_2\ v_j - \sum_{j=1}^{1} a\ v_j\ v_2$$

$$= CW\ v_1 + a\ v_2\ v_3 + a\ v_2\ v_4 + a\ v_2\ v_5 - a\ v_1\ v_2$$

$$= 2 + 0 + 1 + 0 - \{1\}$$

$$= 2$$

$$CW\ v_3 = CW\ v_2 + \sum_{j=4}^{n} a\ v_3\ v_j - \sum_{j=1}^{2} a\ v_j\ v_3$$

$$= CW\ v_2 + a\ v_3\ v_4 + a\ v_3\ v_5 - a\ v_1\ v_3 + a\ v_2\ v_3$$

$$= 2 + 0 + 1 - \{0 + 0\}$$

$$= 3$$

$$CW\ v_4 = CW\ v_3 + \sum_{j=5}^{n} a\ v_4\ v_j - \sum_{j=1}^{3} a\ v_j\ v_4$$

$$= CW\ v_3 + a\ v_4\ v_5 - a\ v_1\ v_4 + a\ v_2\ v_4 + a\ v_3\ v_4$$

$$= 3 + 1 - 1 + 1 + 0$$

$$= 2$$

En la figura 4.5 muestra cómo se calcula el valor del CW para los siguientes cortes:

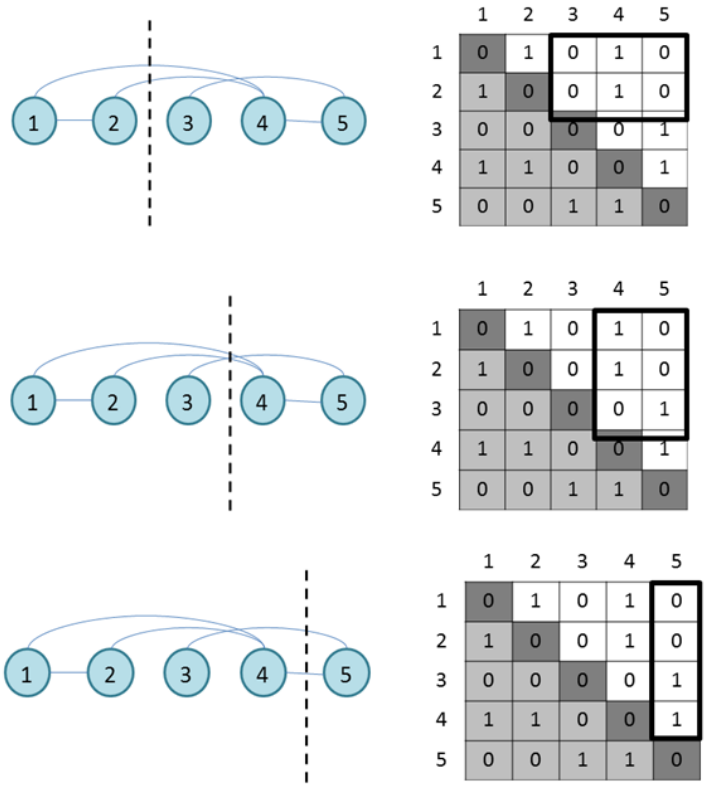


Figura 4.5 Cutwith del nodo $\boldsymbol{v_k}$.

El valor de la función objetivo del CW para cualquier permutación en la última posición, debido a que ningún arco pasa sobre este punto de corte, está dado por:

$$CW\ v_n\ = 0 \tag{4.4}$$

La definición del SUMCUT (SC) es dado un grafo no dirigido $G = (V, E)$ con $n = |V|$ y $m = |E|$, un etiquetado o arreglo lineal π de G asigna los números enteros $1,2,\dots,n$ a los vértices del conjunto V, donde cada vértice recibe una etiqueta distinta [Petit, 2011].

El SC del grafo G con respecto al arreglo lineal π, $SC_\pi\ G$, es la suma de todos sus CW:

$$SC_\pi = \sum_{v \in V} CW_\pi\ v \tag{4.5}$$

De esta forma el SC de define como:

$$SC(G) = SC(\pi^*) \tag{4.6}$$

dónde:

$$\pi^* = argmin_{\pi \in P}\ SC\ \pi \tag{4.7}$$

De acuerdo al grafo de la figura 4.1 la tabla 4.1 muestra los CW de la permutación $\pi =$ 1,2,3,4,5 :

Tabla 4.1 CutWidth de la permutación $\boldsymbol{\pi}$.

$\boldsymbol{\pi}$	1	2	3	4	5
CW	2	2	3	2	0

Calculado todos lo CW para cada nodo del grafo se prosigue al cálculo de SC, este valor de obtiene de la suma de todos los CW del grafo, el resultado seria 9.

$$SC\ G\ = CW\ 1\ + CW\ 2\ + CW\ 3\ + CW\ 4\ + CW\ 5$$

$$= 2 + 2 + 3 + 2 + 0$$

$$= 9$$

4.1.2 Solución exhaustiva del SUMCUT

Considerando el grafo de la figura 4.1 se programó un exhaustivo que a continuación se muestra para resolver el SUMCUT.

Algoritmo1. Exhaustivo

```
minCosto = ∞
P = GenerarPermutaciones
Do {
    perm = i|i ∈ P
    costo = calculaSC(perm)
    if (costo < minCosto)
    {
        minCosto = costo
    }
    P = P − 1

} until(P = ∅)
```

La solución del problema se muestra en la tabla 4.2:

Tabla 4.2 Algunas soluciones del Exhaustivo

Permutación ∏	SC(∏)
(1,2,4,5,3)	6
(1,2,4,3,5)	7
(3,5,1,2,4)	8
(5,3,2,1,4)	9
(1,2,3,5,4)	10
(3,4,1,2,5)	11
(4,2,3,5,1)	12
(5,2,1,3,4)	13

Dando como solución óptima la permutación $\pi =$ 1,2,4,5,3 y el valor de la función objetivo del SC es 6, donde es el valor mínimo de todos los SC.

4.1.2 Estructura del Problema

Al hacer este programa se notó que el valor de la solución objetivo se repite en varias permutaciones como se muestra en la gráfica, donde el rango del valor objetivo es de 6 a 13.

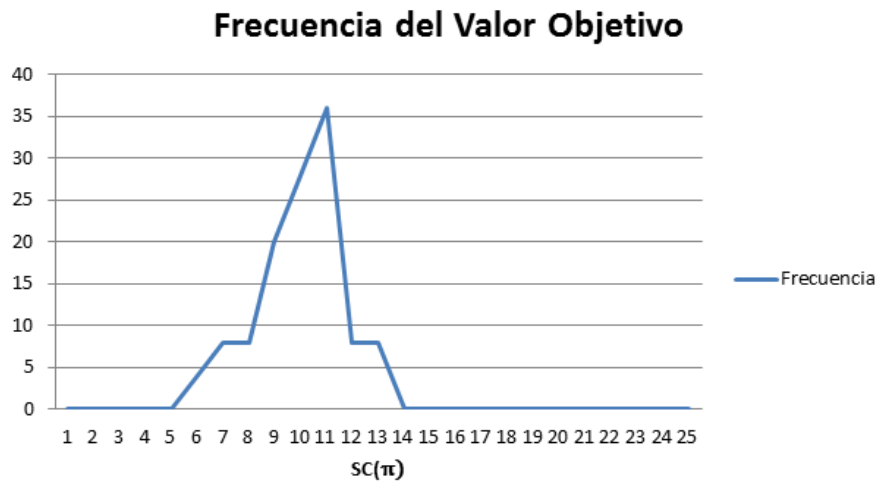


Figura 4.6 Frecuencia del valor Objetivo.

El problema de SUMCUT, como otros similares, tiene la propiedad de que existen múltiples soluciones del espacio de búsqueda que tiene el mismo valor objetivo. Este comportamiento se puede observar en la gráfica anterior.

Otra propiedad de estos problemas es que los valores parciales de la función objetivo también presentan frecuentemente repeticiones. Esto se puede observar en la tabla 4.1.

Estas dos propiedades pueden ser utilizadas para el diseño de nuevos métodos de solución eficientes del problema SUMCUT. El enfoque general de este trabajo consiste en utilizar estas propiedades en el diseño del algoritmo de tipo ramificación y acotamiento.

4.2 Complejidad del problema

[Ravi, 1991] se demuestra que el problema del SUMCUT es equivalente al problema de minimización Profile, cuya diferencia es que para dicho problema sería necesario proporcionar la solución inversa a la obtenida. En [Yuan, 1998] se demuestra que el problema del Profile es NP-completo para grafos bipartidos completos, por lo tanto el problema del SUMCUT también es NP-completo, ya que estos problemas son equivalentes.

CAPITULO 5 Métodos Exactos de Solución Propuestos

En este capítulo se describirá el diseño e implementación de los métodos exactos de solución propuestos para resolver el problema del SUMCUT.

5.1 Modelo de programación lineal entera

A continuación se muestran los modelos de programación lineal entera desarrollados en esta tesis para resolver el problema del SUMCUT.

5.1.1 Modelo cuadrático.

Para modelar el SUMCUT se utilizan dos variables binarias la variable x_u^p es usada para representar el ordenamiento lineal y se define como:

$$x_u^p = \begin{matrix} 1 & p = \varphi(u) \\ 0 & otherwise \end{matrix} \tag{5.1}$$

La variable $y_{u,v}^{p,q}$ permite modelar la conectividad del grafo, y se define como $y_{u,v}^{p,q} = x_u^p \wedge x_v^q$, es decir, si existe una arista que una a ese par de nodos en la permutación actual, se puede también definir a estas variables de la siguiente forma:

$$y_{u,v}^{p,q} = \begin{matrix} 1, & x_u^p = 1 \wedge x_v^q = 1 \\ 0, & otherwise \end{matrix} \tag{5.2}$$

A continuación se muestra el modelo:

$$min \sum_{p \le c < q \ \vee \ q \le c < p} y_{u,v}^{p,q} \tag{5.3}$$

Sujeto a:

$$\sum_{p=1}^{n} x_u^p = 1, \forall u = 1,2,\dots,n \tag{5.4}$$

$$\sum_{u=1}^{n} x_u^p = 1, \forall p = 1,2,\dots,n \tag{5.5}$$

$$y_{u,v}^{p,q} = x_u^p x_v^q, \forall \ u,v \ \in E \vee \ v,u \ \in E, \forall p,q = 1,2,\dots,n \tag{5.6}$$

Las restricciones de asignación (5.4) y (5.5) aseguran que el SUMCUT sea determinado únicamente para ordenamientos lineales factibles. La restricción (5.4) establece que cada vértice del grafo debe ser asignado a una posición. La restricción (5.5) especifica que únicamente un vértice puede ser ubicado en cada posición. La restricción (5.6) sirve para especificar el conjunto de aristas en función de las posiciones relativas de los vértices $u, v \ \in E$ en el ordenamiento lineal.

5.1.2 Modelo lineal ILP1.

Este modelo ILP1 utiliza las mismas variables y restricciones del modelo anterior, solo que para la restricción (5.6) se sustituye por las restricciones (5.10), (5.11) y (5.12), las cuales son el resultado de aplicar la técnica de linealización tradicional [Fortet, 1959].

Se puede ver a continuación el modelo matemático ILP1:

$$\min \sum_{p \le c < q \ \vee \ q \le c < p} y_{u,v}^{p,q} \tag{5.7}$$

Sujeto a:

$$\sum_{p=1}^{n} x_u^p = 1, \forall u = 1,2,\dots,n \tag{5.8}$$

$$\sum_{u=1}^{n} x_u^p = 1, \forall p = 1,2,\dots,n \tag{5.9}$$

$$y_{u,v}^{p,q} \le x_u^p, \forall\ u,v\ \in E \vee\ v,u\ \in E, p,q = 1,2,\dots,n \tag{5.10}$$

$$y_{u,v}^{p,q} \le x_v^q, \forall\ u,v\ \in E \vee\ v,u\ \in E, p,q = 1,2,\dots,n \tag{5.11}$$

$$x_u^p + x_v^q \le y_{u,v}^{p,q} + 1, \forall\ u,v\ \in E \vee\ v,u\ \in E, p,q = 1,2,\dots,n \tag{5.12}$$

5.1.3 Modelo compacto ILP2.

El modelo matemático ILP2 se define utilizando las mismas variables y restricciones utilizadas en el modelo matemático ILP1, con la diferencia que se sustituyen las restricciones (5.10), (5.11) y (5.12) por las restricciones (5.16) y (5.17). Esto se realiza utilizando la técnica de linealización propuesta por Leo Lieberti [Liberti, 2007], como se muestra a continuación:

$$\min \sum_{p \le c < q \ \vee \ q \le c < p} y_{u,v}^{p,q} \tag{5.13}$$

Sujeto a:

$$\sum_{p=1}^{n} x_u^p = 1, \forall u = 1,2,\dots,n \tag{5.14}$$

$$\sum_{u=1}^{n} x_u^p = 1, \forall p = 1,2,\dots,n \tag{5.15}$$

$$\sum_{p=1}^{n} y_{u,v}^{p,q} = x_v^q, \forall\ u,v\ \in E \vee\ v,u\ \in E, q = 1,2,\dots,n \tag{5.16}$$

$$y_{u,v}^{p,q} = y_{v,u}^{q,p}, \forall\ u,v\ \in E \vee\ v,u\ \in E, p,q = 1,2,\dots,n \tag{5.17}$$

5.1.4 Modelo basado en conjuntos ILP3.

De acuerdo a [Cohen, 2014] se decidió realizar un modelo de programación lineal basado en conjuntos, el cual para modelar el SUMCUT se consideró que una secuencia $v_1, v_2, \dots, v_n$ que se puede representar como una secuencia de conjuntos v_1 , v_1, v_2 , v_1, v_2, v_3 , … , $v_1, v_2, \dots, v_n$. Debido a ello se construye la secuencia S_t de conjuntos que para un ordenamiento $v_1, v_2, \dots, v_n$ de los vértices se representa como: $S_0 = v_1$, $S_1 = v_1, v_2$,…, $S_{n-1} = v_1, v_2, \dots, v_n$.

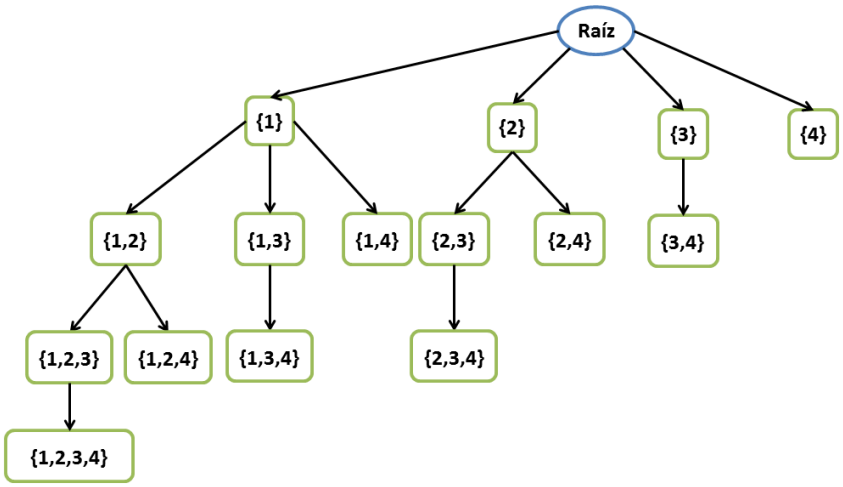


Figura 5.1 Árbol de conjuntos.

De acuerdo al árbol al ir creando la secuencia de conjuntos para la solución se va agregando el nodo, por ejemplo:

Secuencia de Conjuntos	Permutación
$S_0 = \{3\}$	3 | | | |
$S_1 = \{3,4\}$	3 | 4 | | |
$S_2 = \{1,3,4\}$	3 | 4 | 1 | |
$S_3 = \{1,3,4,5\}$	3 | 4 | 1 | 5 |
$S_4 = \{1,2,3,4,5\}$	3 | 4 | 1 | 5 | 2

Figura 5.2 Secuencia de Conjuntos.

Para este modelo se utilizan tres variables binarias, la variable y_u^t es usada para representar los conjuntos y se define como:

$$y_u^t = \begin{matrix} 1 & si\ u \in S_t \\ 0 & otherwise \end{matrix} \tag{5.18}$$

La variable $R_{u,w}^t$ revisa si el par de nodos se encuentra en el conjunto S_t:

$$R_{u,w}^t = \begin{matrix} 1 & si\ y_u^t \wedge y_w^t \\ 0 & otherwise \end{matrix} \tag{5.19}$$

La tercera variable $U_{u,w}^t$ permite modelar la conectividad del grafo, además va revisando si uno de los nodos pertenece al conjunto S_t, es decir, si existe una arista entre este para de nodo y además uno de ellos pertenece al conjunto St. Se puede también definir esta variable de la siguiente forma:

$$U_{u,w}^t = \begin{matrix} 1 & si\ u \in St \wedge \exists\ u,w\ \in E : w \notin St \\ 0 & otherwise \end{matrix} \tag{5.20}$$

Se puede ver a continuación el modelo de programación lineal ILP3 aplicando la técnica de linealización tradicional [Fortet, 1959]:

$$\min \sum_{t=1}^{n} \sum_{(u,w)\in E} U_{u,w}^{t} \tag{5.21}$$

Sujeto a:

$$y_u^t = y_u^{t+1} \qquad \forall u \in V, t = 1,2,\dots,n-1 \tag{5.22}$$

$$\sum_{u\in V} y_u^t = t \qquad t = 1,2,\dots,n \tag{5.23}$$

$$R_{u,w}^t \leq y_u^t \qquad \forall\ u,w\ \in E, t = 1,2,\dots,n \tag{5.24}$$

$$R_{u,w}^t \leq y_w^t \qquad \forall\ u,w\ \in E, t = 1,2,\dots,n \tag{5.25}$$

$$y_u^t + y_w^t \leq R_{u,w}^t + 1 \qquad \forall\ u,w\ \in E, t = 1,2,\dots,n \tag{5.26}$$

$$U_{u,w}^t = y_u^t + y_w^t - 2R_{u,w}^t \qquad \forall\ u,w\ \in E, t = 1,2,\dots,n \tag{5.27}$$

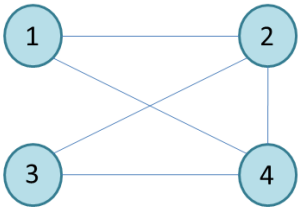
La restricción (5.22) verifica que todo nodo que se encuentre en el conjunto S_t se encuentre en el conjunto S_{t+1}. La restricción (5.23) revisa que cada conjunto t contenga t elementos. Las restricciones (5.24), (5.25) y (5.26) controlan los elementos que se encuentran dentro del conjunto S_t. La restricción (5.27) sirven para especificar el conjunto de aristas en función de las posiciones relativas de los vértices $u, w \in E$ que se encuentran en el conjunto S_t.

5.2 Métodos Basados en ramificación y acotamiento

En los siguientes subtemas se muestra los métodos basados en ramificación y acotamiento que se desarrollaron para resolver el problema del SUMCUT.

5.2.1 Ramificación y acotamiento SCBAB1.

En esta sección se describe un método exacto propuesto que aplica la metodología de ramificación y acotamiento. El algoritmo SCBAB1 hace una búsqueda en profundidad revisando todas las ramas generadas, para obtener el límite superior (UB) se utiliza una heurística que obtiene una solución próxima al óptimo para acortar la búsqueda. Esta heurística escoge el primer nodo con el menor grado y los siguientes nodos con el menor corte. La suma de todos estos cortes es el valor que se le asigna al UB. En el siguiente ejemplo se muestra su funcionamiento tomando el siguiente grafo:



	1	2	3	4
1	0	1	0	1
2	1	0	1	1
3	0	1	0	1
4	1	1	1	0

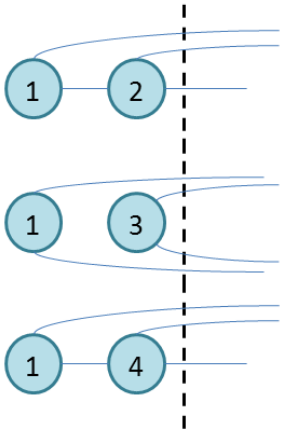
Figura 5.3 Grafo y matriz de adyacencia.

Para crear la solución toma el primer nodo con el menor grado:

Tabla 5.1 Grado correspondiente a cada nodo.

Nodo	1	2	3	4
Grado	2	3	2	3

Se toma el nodo 1 y se agrega a la solución {1}. Después para los siguientes nodos se va a tomar el primer nodo con el menor corte, como se muestra a continuación:



Solución Parcial	CW
{1,2}	3
{1,3}	4
{1,4}	3

Figura 5.4 Selección del segundo nodo.

El procedimiento de la figura 5.4 continua hasta crear la solución completa, donde obteniendo la solución se calcula el SUMCUT y ese valor es asignado al UB para el algoritmo.

Tabla 5.2 CutWith de la solución.

Solución	1	2	4	3
CW	2	3	2	0

La suma de todos los CutWith es 7, por lo tanto ese valor es asignado al UB. En el siguiente algoritmo se muestra el pseudocódigo heurística.

Algoritmo 5.1 **Pseudocódigo de la Heurística para obtener el UB**

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

Salida: $\boldsymbol{UB}$

1: $\boldsymbol{CW[1, \ldots, n] \leftarrow 0}$

2: $\boldsymbol{leer\_instancia()}$

3: $\boldsymbol{sol\ 1\ = nodo\ con\ menos\ grado\ con\ menor\ grado}$

4: for $\boldsymbol{i = 2}$ **to** $\boldsymbol{n}$ **do**

5: $\boldsymbol{Mgrado = \infty}$

6: $\boldsymbol{Actualizar\_Stack()}$

7: **for** $j = 1$ **to** $Tamaño\_Stack$ **do**

8: $sol\ i \leftarrow Pop(Stack)$

9: $cw \leftarrow Calculo\_CW()$

10: **if** $cw < Mgrado$ **then**

11: $Mgrado \leftarrow CW$

12: $Nodo \leftarrow sol[i]$

13: **end if**

14: **end for**

15: $sol[i] \leftarrow nodo$

16: $CW\ i \leftarrow Mgrado$

17: end for

18: return $UB \leftarrow Calculo\_SC()$

Después de obtener el valor de UB empieza la búsqueda en el árbol haciendo la ramificación hasta llegar a un porcentaje establecido donde se realiza un cálculo parcial del valor objetivo (PO), al hacer esto se compara el valor obtenido con el UB, si es mayor que el UB entonces se corta la rama y ya no continua la búsqueda, en caso contrario si en menor que el UB se continua la búsqueda generando más ramas hasta llegar a ultima hoja donde se actualiza el UB con el valor objetivo encontrado, como se muestra en el siguiente ejemplo.

Suponiendo que el UB es 8 el árbol con n = 4, se va creando el árbol como se muestra en la figura 5.5.

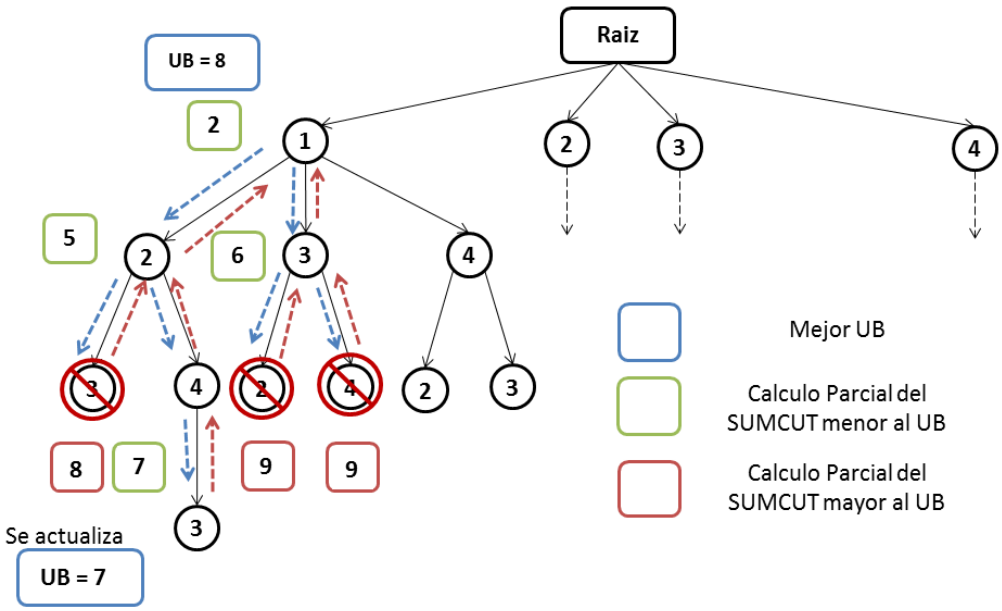


Figura 5.5 Árbol del SCBAB1.

Algoritmo 5.2 **Pseudocódigo del algoritmo SCBAB1**

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

Salida: $(\boldsymbol{\pi^*, UB})$

1: $\boldsymbol{Stack \leftarrow \{n, n-1, \ldots, 1\}}$

2: $\boldsymbol{Levels \leftarrow \bigcup_{i=1}^{n} \{1\}}$

3: $\boldsymbol{CW[1, \ldots, n] \leftarrow 0}$

4: $\boldsymbol{c \leftarrow 1}$

5: $\boldsymbol{UB \leftarrow 0}$

6: While $\boldsymbol{Stack \neq \{\emptyset\}}$ **do**

8: $\boldsymbol{c \leftarrow Pop(Levels)}$

9: $\boldsymbol{sol[c] \leftarrow Pop(Stack)}$

10: if $\boldsymbol{c = n}$ **then**

11: SC($\boldsymbol{sol}$) = Sum(CW)

12: if $\boldsymbol{SC(sol) < SC(Mejor\_sol)}$ **then**

13: $\boldsymbol{Mejor\_sol \leftarrow sol}$

14: $\boldsymbol{SC(Mejor\_sol) \leftarrow SC(sol)}$

15: $\boldsymbol{UB \leftarrow SC\ Mejor\_sol}$

16: end if

17: **else**
18: **if** $porcentaje < 40$ **then**
19: $PO \leftarrow CalculoPacial(c)$
20: **if** $PO < UB$ **then**
21: $Ramificar(sol\ c\ , Stack, Levels)$
22: **else**
23: $Cortar(sol\ c\ , Stack, Levels)$
24: **end if**
25: **else**
26: $Ramificar(sol\ c\ , Stack, Levels)$
27: **end if**
28: **end if**
29: **end While**
30: **return** $(Mejor\_sol, UB)$

5.2.2 Ramificación y acotamiento basado en conjuntos SCBAB2.

El algoritmo de ramificación y acotamiento SCBAB2 crea un árbol con conjuntos S_t de tamaño $t = 1,2,\dots,n$ como se muestra en la figura 5.6.

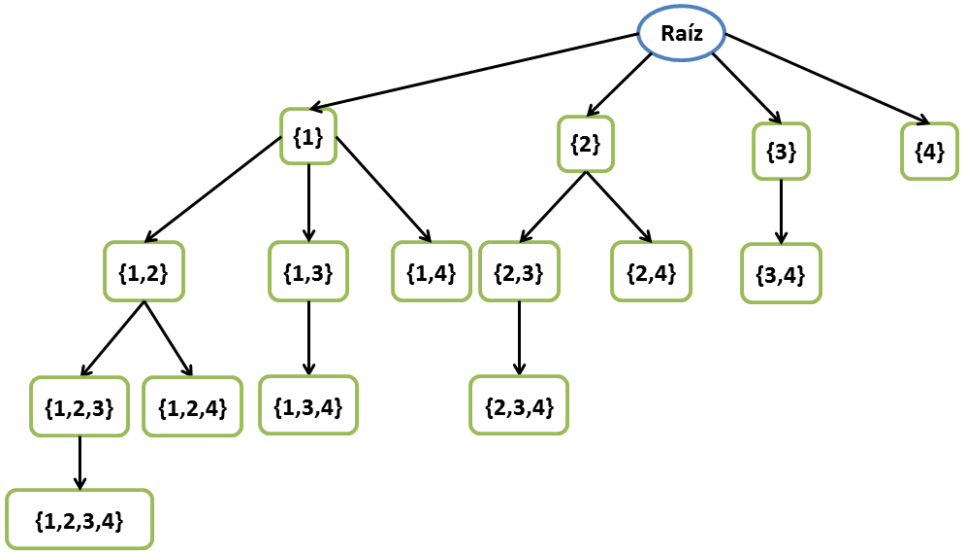


Figura 5.6 Árbol de conjuntos.

Para realizar la búsqueda en el árbol se necesita un límite superior (UB) este se consigue realizando una heurística en la cual se obtiene una solución que se acerque al óptimo para acortar la búsqueda, la cual se utilizó en el anterior algoritmo de ramificación y acotamiento. El algoritmo 5.1 muestra el funcionamiento de esta heurística.

Después de obtener el valor de UB_H empieza la búsqueda en el árbol donde comenzamos con un $UB = 0$, se calcula el valor parcial del conjunto si al menos un conjunto es menor que el UB esté se mantiene, pero si ningún conjunto es menor que el UB se actualiza con la siguiente formula:

$$UB_{nuevo} = \frac{UB_H + UB_{actual}}{2} \tag{5.28}$$

Al generar los conjuntos se calcula su valor parcial si el conjunto es menor que el UB se corta la rama, si no continúa la búsqueda, como se muestra en la figura 5.7.

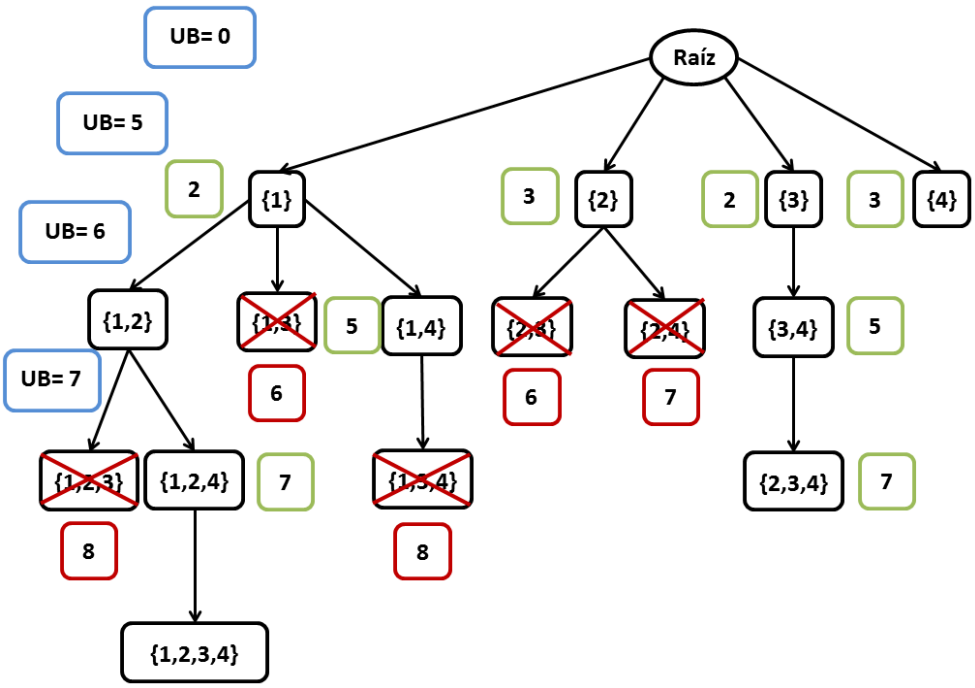


Figura 5.7 Árbol del SCBAB2.

Algoritmo 5.3 **Pseudocódigo del algoritmo SCBAB2**

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

Salida: $(\boldsymbol{\pi^*, UB})$

1: $\boldsymbol{Conjuntos \leftarrow \emptyset}$

2: $\boldsymbol{UB\_H \leftarrow Heuristica()}$

3: $\boldsymbol{CW[1, \ldots, n] \leftarrow 0}$

4: $\boldsymbol{UB \leftarrow 0}$

5:seguir = false

6: While $\boldsymbol{t < n - 1}$ **do**

7: $\boldsymbol{sol \leftarrow GeneraConjunto(t)}$

8: $\boldsymbol{CP \leftarrow CalculoParcial(t)}$

9: if $\boldsymbol{CP < UB}$ **then**

10: $\boldsymbol{Conjuntos \leftarrow sol}$

11: $\boldsymbol{seguir \leftarrow true}$

12: end if

13: if seguir then

14: t++

15: else

16: $\boldsymbol{UB \leftarrow Actualiza()}$

17: end if

18: end While

19: $\boldsymbol{Mejor\_sol \leftarrow GenerarPemutacion(sol)}$

20: return (*Mejor_sol,UB*)

5.3 GRASP.

Para realizar comparar el desempeño de los métodos exactos propuestos contra un método metaheurístico se construyó el algoritmo de solución de tipo GRASP que se describe en esta sección.

Algoritmo 5.4 **Pseudocódigo del GRASP**

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

Salida: $\boldsymbol{SCGlobal}$

1: $\boldsymbol{leer\_instancia}()$

3: iteraciones = 15

4: for $\boldsymbol{i = 0}$ **to** $\boldsymbol{iteraciones}$ **do**

5: $\boldsymbol{generar\_sol}()$

6: $\boldsymbol{Busqueda\_Local}()$

7: if $\boldsymbol{SCLocal < SCGlobal}$ **then**

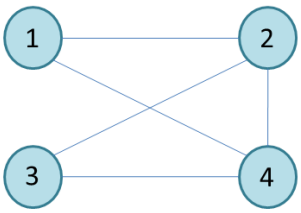
11: $\boldsymbol{SCGlobal \leftarrow SCLocal}$

12: end if

14: end for

15: return $\boldsymbol{SCGlobal}$

El método para generar la solución inicial es un algoritmo H1 que construye una solución inicial seleccionando los nodos con el menor grado y de ese grupo se escoge de forma aleatoria un nodo, para los siguientes nodos de la permutación que se van a seleccionar se elige el nodo con el menor corte. Una vez hecho se calcula el valor del SUMCUT de esta permutación. Después se localiza el nodo con mayor grado y lo ubica a la mitad de la permutación, si al hacer esto el valor del SUMCUT de la nueva permutación es menos al anterior se queda con esta nueva permutación para la solución inicial. En el siguiente ejemplo se muestra su funcionamiento tomando el siguiente grafo:



	1	2	3	4
1	0	1	0	1
2	1	0	1	1
3	0	1	0	1
4	1	1	1	0

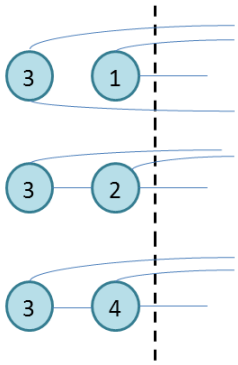
Figura 5.8 Grafo y Matriz de Adyacencia

Se seleccionan los nodos con el menor grado:

Nodo	1	3
Grado	2	2

Figura 5.9 Nodos con menor grado.

De los nodos con menor grado se elige aleatoriamente el primer nodo de la permutación, en este caso es el nodo 3 . Después para los siguientes nodos se va a tomar el primer nodo con el menor corte, como se muestra a continuación:



Solución Parcial	CW
{3,1}	4
{3,2}	3
{3,4}	3

Figura 5.10 Selección del segundo nodo.

El procedimiento de la figura 5.10 continua hasta crear la solución completa, donde obteniendo la solución se calcula el SUMCUT.

Tabla 5.3 CutWith de la solución.

Solución	3	2	4	1
CW	2	3	2	0

Para esta permutación el SUMCUT es de 7. Una vez hecho esto se localiza el primer nodo con el mayor grado y le ubica a la mitad de la permutación $\frac{n}{2}$. En este caso es el nodo 2 si este ya se encuentra en la posición de en medio de la permutación ya no se realiza este paso como se muestra en este caso.

Algoritmo 5.5 **Pseudocódigo para obtener la solución inicial H1**

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

Salida: $\boldsymbol{UB}$

1: $\boldsymbol{CW[1,\ldots,n] \leftarrow 0}$

2: $\boldsymbol{leer\_instancia()}$

3: $\boldsymbol{sol\ 1\ = seleccio\_aleatria()}$

4: for $\boldsymbol{i = 2}$ **to** $\boldsymbol{n}$ **do**

5: $\boldsymbol{Mgrado = \infty}$

6: $\boldsymbol{Actualizar\_Stack()}$

7: for $\boldsymbol{j = 1}$ **to** $\boldsymbol{Tamaño\_Stack}$ **do**

8: $\boldsymbol{sol\ i\ \leftarrow Pop(Stack)}$

9: $\boldsymbol{cw \leftarrow Calculo\_CW()}$

10: if $\boldsymbol{cw < Mgrado}$ **then**

11: $\boldsymbol{Mgrado \leftarrow CW}$

12: $\boldsymbol{Nodo \leftarrow sol[i]}$

13: end if

14: end for

15: $\boldsymbol{sol[i] \leftarrow nodo}$

16: $\boldsymbol{CW\ i\ \leftarrow Mgrado}$

17: end for

18: $\boldsymbol{intercambio\_mayorGrado()}$

18: return $\boldsymbol{sol}$

Para el método de búsqueda local se decidió utilizar el propuesto en [Zamarron, 2012], que es una búsqueda optima que evalúa toda la vecindad de inserción hasta encontrar el mejor movimiento que mejore el valor objetivo, si existiera una mejora, el proceso se vuelve a repetir.

En la Figura 5.11, se muestra la permutación $\pi =$ 1,2,3,4,5 con un SC(π) = 15, donde se explora todo el vecindario hasta encontrar el mejor movimiento resultando en la permutación $\pi =$ 1,2,4,5,3 , como se encontró una mejora se vuelve a explorar todo el vecindario por un mejor movimiento, el cual resulta en la permutación $\pi =$ 1,4,2,5,3 , se vuelve a reiniciar este proceso de búsqueda pero debido a que ningún movimiento mejora el valor objetivo actual esta termina regresando π.

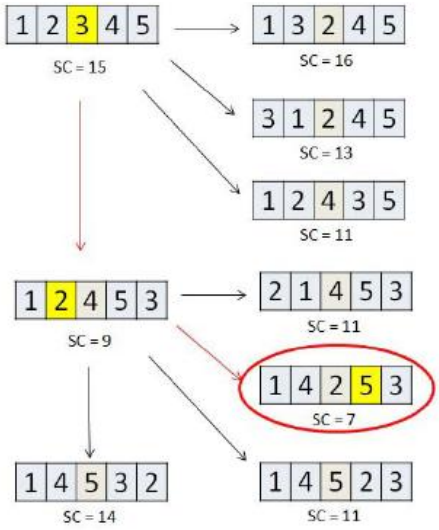


Figura 5.11 Búsqueda Local.

CAPITULO 6 Resultados experimentales

En este capítulo se mostrara los resultados obtenidos, así como bajo qué condiciones se realizaron lo experimentos y las instancias utilizadas.

6.1 Instancias

Los grupos de instancias utilizados son los siguientes:

- **Small:** En un conjunto de datos de 84 grafos introducidos en el contexto del problema de reducción de bandwith introducidos en [Martih, 2008]. El rango del número de vértices va de 16 a 24, y el rango del número de arcos se de 18 a 49.

- **Grids:** Este conjunto de datos se compone de 81 matrices construidas como el producto cartesiano de dos caminos [Raspaud, 2009]. Tambien fueron llamadas dos mallas dimensionales, y fueron documentadas en [Raspaud, 2009]. Para este conjunto de instancias, los vértices están colocados en un malla donde su dimensión es de anchura x altura, esta dos medidas son seleccionadas entre el conjunto {3, 6, 9, 12, 15, 18, 21, 24, 27}.

- **Harwell-Boeing:** Este grupo de instancias consiste en un conjunto de matrices de prueba estándar $M = (M_{ij})$ derivadas de problemas en sistemas lineales, minimos cuadrados y cálculos de valores propios de un amplia variedad de disciplinas científicas de ingeniería. Los grafos son derivados de estas matrices considerando un arco (i, j) para cada elemento $M_{ij} \neq 0$.

- **Tree:** Conjunto de instancias que consisten en arboles con un mínimo número de nodos y separación de vértices igual a λ. En general, para la construcción de un árbol con la separación vértice $\lambda + 1$ es necesario para seleccionar cualquiera de los tres miembros de T (λ) y vincular cualquier nodo de cada uno de estos a un nuevo nodo que actúa como la raíz del nuevo árbol. El número de nodos, n (λ), de un árbol en T (λ) se puede obtener usando la relación de recurrencia n (λ) = 3n (λ - 1) + 1, donde n y (1) = 2 [Ellis, 1994].

Las instancias empleadas para la experimentación fueron 5 de las *Grid*, 84 de las *Small* , 15 de las *Tree* y 3 de las *Harwell-Boeing* (HB). Las cuales son las que se muestran en la siguiente tabla:

Tabla 6.1 Instancias empleadas para la experimentación.

Grupo de Instancias	Instancia	
Grid	Grid3x3.txt	grid_6.mtx.rnd
	Grid4x4.txt	grid_7.mtx.rnd
	grid_5.mtx.rnd	
Small	p17_16_24_n.txt	p59_20_23_n.txt
	p18_16_21_n.txt	p60_20_22_n.txt
	p19_16_19_n.txt	p61_21_22_n.txt
	p20_16_18_n.txt	p62_21_30_n.txt
	p21_17_20_n.txt	p63_21_42_n.txt
	p22_17_19_n.txt	p64_21_22_n.txt
	p23_17_23_n.txt	p65_21_24_n.txt
	p24_17_29_n.txt	p66_21_28_n.txt
	p25_17_20_n.txt	p67_21_22_n.txt
	p26_17_19_n.txt	p68_21_27_n.txt
	p27_17_19_n.txt	p69_21_23_n.txt
	p28_17_18_n.txt	p70_21_25_n.txt
	p29_17_18_n.txt	p71_22_29_n.txt
	p30_17_19_n.txt	p72_22_49_n.txt
	p31_18_21_n.txt	p73_22_29_n.txt
	p32_18_20_n.txt	p74_22_30_n.txt
	p33_18_21_n.txt	p75_22_25_n.txt
	p34_18_21_n.txt	p76_22_30_n.txt
	p35_18_19_n.txt	p77_22_37_n.txt
	p36_18_20_n.txt	p78_22_31_n.txt
	p37_18_20_n.txt	p79_22_29_n.txt
	p38_18_19_n.txt	p80_22_30_n.txt
	p39_18_19_n.txt	p81_23_46_n.txt
	p40_18_32_n.txt	p82_23_24_n.txt
	p41_19_20_n.txt	p83_23_24_n.txt
	p42_19_24_n.txt	p84_23_26_n.txt
	p43_19_22_n.txt	p85_23_26_n.txt
	p44_19_25_n.txt	p86_23_24_n.txt
	p45_19_25_n.txt	p87_23_30_n.txt
	p46_19_20_n.txt	p88_23_26_n.txt
	p47_19_21_n.txt	p89_23_27_n.txt
	p48_19_21_n.txt	p90_23_35_n.txt
	p49_19_22_n.txt	p91_24_33_n.txt
	p50_19_25_n.txt	p92_24_26_n.txt
	p51_20_28_n.txt	p93_24_27_n.txt
	p52_20_27_n.txt	p94_24_31_n.txt
	p53_20_22_n.txt	p95_24_27_n.txt
	p54_20_28_n.txt	p96_24_27_n.txt

	p55_20_24_n.txt p56_20_23_n.txt p57_20_24_n.txt p58_20_21_n.txt	p97_24_26_n.txt p98_24_29_n.txt p99_24_27_n.txt p100_24_34_n.txt
Tree	1rot	TREE_22_3_rot1.mtx.rnd TREE_22_3_rot2.mtx.rnd TREE_22_3_rot3.mtx.rnd TREE_22_3_rot4.mtx.rnd TREE_22_3_rot5.mtx.rnd
	2rot	TREE_22_3_rot1.mtx.rnd TREE_22_3_rot2.mtx.rnd TREE_22_3_rot3.mtx.rnd TREE_22_3_rot4.mtx.rnd TREE_22_3_rot5.mtx.rnd
	3rot	TREE_22_3_rot1.mtx.rnd TREE_22_3_rot2.mtx.rnd TREE_22_3_rot3.mtx.rnd TREE_22_3_rot4.mtx.rnd TREE_22_3_rot5.mtx.rnd
HB	bcspwr01.mtx.rnd bcspwr02.mtx.rnd can___24.mtx.rnd	

6.2 Condiciones Experimentales

Todos los experimentos están realizados bajo estos entornos de prueba:

- **Procesador:** Intel Pentium de 2.00 GHz.
- **Memoria RAM:** 4 GB.
- **Sistema Operativo:** Windows 7.
- **Entorno de Desarrollo:** NetBeans.
- **Lenguaje de Programación:** Java.
- Para los modelos de Programación Lineal se utilizó la librería CPLEX 10.9.

6.3 Experimentos

Las instancias de los grupos *Grid*, *Small* y *Tree* fueron resueltas con un tiempo límite de 300s de CPU utilizando el optimizador CPLEX, en las tablas 6.2 y 6.3 se muestran los resultados de los modelos de programación lineal entera para estas instancias. La primera columna de la tabla contiene el nombre del grupo y la cantidad de instancias utilizadas. Las siguientes tres columnas contienen los resultados que se obtienen con el modelo ILP1. El contenido de estas columnas es el promedio de los mejores valores objetivos encontrados (Prom. Mejor), el número de óptimos encontrados (#Opt) y el promedio del tiempo de solución de las instancias del grupo correspondiente en segundos de CPU.(Prom. Tiempo). Las últimas 3 columnas contienen los resultados correspondientes con el modelo ILP2. En la última fila de la tabla se muestran los acumulados de los valores observados en los experimentos.

Tabla 6.2 Resultados de la solución de las instancias *Grid*, *Small* y *Tree* con los modelo de programación lineal entera propuestos (ILP1 y ILP2)

	ILP1			**ILP2**		
Grupo	**Prom. Mejor**	**#Opt**	**Prom. Tiempo**	**Prom. Mejor**	**#Opt**	**Prom. Tiempo**
Grid(5)	490.6	1	192.2	377	1	240.9522
Tree(15)	54.93	0	280.1189	35.47	0	300.3935
Small(84)	86.23	0	300.1038	63.78	23	231.527619
Total	631.76	1	855.3729	476.25	24	772.873319

Tabla 6.3 Resultados de la solución de las instancias *Grid*, *Small* y *Tree* con los modelo de programación lineal entera propuesta en este documento (ILP3)

	ILP3		
Grupo	**Prom. Mejor**	**#Opt**	**Prom. Tiempo**
Grid(5)	154.6	2	198.2734
Tree(15)	35.2	0	300.03
Small(84)	62.06	59	125.60397
Total	251.86	61	623.90737

En las siguientes tablas muestra los resultados obtenidas para el grupo de instancias HB con un tiempo límite de 3600s de CPU utilizando también para esta instacias el optimizador CPLEX.

Tabla 6.4 Resultados de la solución de las instancias *HB* con los modelos de programación lineal entera propuestos (ILP1 y ILP2)

	ILP1			ILP2		
Grupo	**Prom. Mejor**	**#Opt**	**Prom. Tiempo**	**Prom. Mejor**	**#Opt**	**Prom. Tiempo**
HB(3)	289.66	0	3600.643	200.66	0	3600.75

Tabla 6.5 Resultados de la solución de las instancias *HB* con el modelo de programación lineal entera propuesto en este documento (ILP3)

	ILP3		
Grupo	**Prom. Mejor**	**#Opt**	**Prom. Tiempo**
HB(3)	165.33	0	3600.625

De acuerdo con los resultados obtenidos que se muestran en las tablas 6.2, 6.3, 6.4 y 6.5, se observó que el modelo ILP3 obtiene los mejores resultados al obtener mayor número de óptimos encontrados, además de la disminución del tiempo de ejecución.

La instancias utilizadas para el método basado en ramificación y acotamiento son las *Grid*, las *Small*, las *Tree* y las HB, el tiempo límite para las Grid, las *Small* y las *Tree* es de 300s de CPU y los resultados se muestran en la tabla 6.6.

Tabla 6.6 Resultados de la solución de las instancias *Grid*, *Small* y *Tree* con el método basado en ramificación y acotamiento.

	SCBAB1			SCBAB2		
Grupo	**Prom. Mejor**	**#Opt**	**Prom. Tiempo**	**Prom. Mejor**	**#Opt**	**Prom. Tiempo**
Grid(5)	147	1	240.138	147	2	196
Tree(15)	37.07	0	300	35.47	0	300
Small(84)	84.023	0	300	81.774	6	295.61
Total	268.09	1	840.138	264.244	8	791.61

Para las instancias HB los resultados se muestran en la tabla 6.7, estas se corrieron con un tiempo límite de 3600s de CPU.

Tabla 6.7 Resultados de la solución de las instancias *HB* con el método basado en ramificación y acotamiento.

	SCBAB1			SCBAB2		
Grupo	**Prom. Mejor**	**#Opt**	**Prom. Tiempo**	**Prom. Mejor**	**#Opt**	**Prom. Tiempo**
HB(3)	215	0	3600	215	0	3600

Para el método basado en ramificación y acotamiento el algoritmo que obtuvo mejor resultados es el SCBAB2 al tener un mayor número de óptimos encontrados en un menor tiempo de ejecución.

Se utilizaron las mismas instancias para los experimentos con el algoritmo GRASP.

Tabla 6. 8 Resultados de la solución de las instancias Grid, Small, Tree y HB con el GRASP.

	GRASP		
Grupo	**Prom. Mejor**	**#Opt**	**Prom. Tiempo**
Grid(5)	148.6	2	0.185
Tree(15)	35.133	0	0.357
Small(84)	75.4881	2	1.067
HB(3)	204.666	0	6.281
Total	463.8871	4	7.89

En las tablas 6.9 y 6.10 se muestran los resultados de los métodos más sobresalientes en este trabajo.

Tabla 6.9 Resultados de la solución de las instancias Grid, Small, Tree y HB con el ILP3 y SCBAB2.

	ILP3			SCBAB2		
Grupo	**Prom. Mejor**	**#Opt**	**Prom. Tiempo**	**Prom. Mejor**	**#Opt**	**Prom. Tiempo**
Grid(5)	154.6	2	198.273	147	2	196
Tree(15)	35.2	0	300.03	35.47	0	300
Small(84)	62.06	59	125.604	81.774	6	295.61
HB(3)	165.33	0	3600.625	204.666	0	3600
Total	417.73	61	4224.532	479.244	8	4391.61

Tabla 6.10 Resultados de la solución de las instancias Grid, Small, Tree y HB con el Heurística solución inicial H1 y GRASP.

	H1			GRASP		
Grupo	**Prom. Mejor**	**#Opt**	**Prom. Tiempo**	**Prom. Mejor**	**#Opt**	**Prom. Tiempo**
Grid(5)	150	2	0.0194	148.6	2	0.185
Tree(15)	37.067	0	0.0108	35.133	0	0.357
Small(84)	82.905	0	0.0055	75.4881	2	1.067
HB(3)	208.333	0	0.0123	215	0	6.281
Total	478.305	2	0.048	463.8871	4	7.89

CAPITULO 7 Conclusiones y Trabajos Futuro

El proyecto de investigación presentado en este trabajo cumplió satisfactoriamente con los objetivos planteados inicialmente, los cuales son la realización de un modelo de programación lineal entera y un método de tipo ramificación y acotamiento. En este capítulo se describen las aportaciones más relevantes del trabajo y los trabajos futuros. A continuación se muestras estas aportaciones:

- El modelo de programación lineal entera basada en conjuntos ILP3 (véase capítulo 5 sección 1.4) obtuvo los mejores resultados comparado con los otros modelos de programación lineal entera, ya obtuvo mayor número de óptimos encontrados en menor tiempo (véase capítulo 6).
- La heurística H1(véase capítulo 5 sección 3) que se utiliza para generar la solución inicial, la cual es de complejidad $O(n^2)$ lo que le permite resolver instancias con un gran número de nodos en poco tiempo. Sus resultados se pueden ver en el capítulo 6.

También se participó en dos congresos, a continuación se enlista los congresos junto con el artículo que se presentó:

- Primer Congreso Internacional de Investigación en Enseñanza de las Ciencias, *Modelos de Programación Lineal Entera Mixta para Minimizar la Suma de los Anchos de Corte en un grafo conexo no dirigido (SUMCUT)*,2013.
- VII Encuentro de Investigadores, *Métodos Exactos para Minimizar la Suma de los Anchos de Corte en un Grafo Conexo no Dirigido*, 2013.

Para los trabajos futuros gracias a la experiencia obtenida se propone seguir abordando este tema pero utilizando algoritmos metaheurísticos para obtener resultados con mayor calidad.

Anexo A

En este anexo se muestra una tabla de con los resultados obtenidos por el modelo de programación lineal entera basado en conjuntos, ya que es el que nos dio mejores resultados.

Tabla A.1 Resultados de los resultados por instancias.

Grupo de Instancias	Instancia	SC	Optimo
Grid	Grid3x3.txt	24	SI
	Grid4x4.txt	60	SI
	grid_5.mtx.rnd	120	NO
	grid_6.mtx.rnd	209	NO
	grid_7.mtx.rnd	360	NO
Small	p17_16_24_n.txt	70	SI
	p18_16_21_n.txt	48	SI
	p19_16_19_n.txt	39	SI
	p20_16_18_n.txt	36	SI
	p21_17_20_n.txt	44	SI
	p22_17_19_n.txt	37	SI
	p23_17_23_n.txt	51	SI
	p24_17_29_n.txt	82	SI
	p25_17_20_n.txt	40	SI
	p26_17_19_n.txt	40	SI
	p27_17_19_n.txt	39	SI
	p28_17_18_n.txt	32	SI
	p29_17_18_n.txt	29	SI
	p30_17_19_n.txt	39	SI
	p31_18_21_n.txt	39	SI
	p32_18_20_n.txt	46	SI
	p33_18_21_n.txt	47	SI
	p34_18_21_n.txt	39	SI
	p35_18_19_n.txt	35	SI
	p36_18_20_n.txt	44	SI
	p37_18_20_n.txt	48	SI

	p38_18_19_n.txt	32	SI
	p39_18_19_n.txt	36	SI
	p40_18_32_n.txt	94	NO
	p41_19_20_n.txt	36	SI
	p42_19_24_n.txt	57	SI
	p43_19_22_n.txt	42	SI
	p44_19_25_n.txt	70	NO
	p45_19_25_n.txt	56	SI
	p46_19_20_n.txt	37	SI
	p47_19_21_n.txt	39	SI
	p48_19_21_n.txt	41	SI
	p49_19_22_n.txt	45	SI
	p50_19_25_n.txt	51	SI
	p51_20_28_n.txt	73	SI
	p52_20_27_n.txt	70	SI
	p53_20_22_n.txt	47	SI
	p54_20_28_n.txt	68	SI
	p55_20_24_n.txt	51	SI
	p56_20_23_n.txt	54	NO
	p57_20_24_n.txt	56	SI
	p58_20_21_n.txt	41	SI
	p59_20_23_n.txt	50	SI
	p60_20_22_n.txt	47	SI
	p61_21_22_n.txt	44	SI
	p62_21_30_n.txt	91	NO
	p63_21_42_n.txt	154	NO
	p64_21_22_n.txt	45	SI
	p65_21_24_n.txt	51	SI
	p66_21_28_n.txt	77	NO
	p67_21_22_n.txt	40	SI
	p68_21_27_n.txt	68	SI
	p69_21_23_n.txt	53	SI
	p70_21_25_n.txt	57	SI
	p71_22_29_n.txt	69	NO
	p72_22_49_n.txt	210	NO
	p73_22_29_n.txt	65	SI
	p74_22_30_n.txt	67	SI

	p75_22_25_n.txt	55	SI
	p76_22_30_n.txt	73	NO
	p77_22_37_n.txt	127	NO
	p78_22_31_n.txt	79	NO
	p79_22_29_n.txt	67	SI
	p80_22_30_n.txt	74	NO
	p81_23_46_n.txt	190	NO
	p82_23_24_n.txt	56	SI
	p83_23_24_n.txt	48	SI
	p84_23_26_n.txt	52	SI
	p85_23_26_n.txt	50	SI
	p86_23_24_n.txt	43	SI
	p87_23_30_n.txt	87	NO
	p88_23_26_n.txt	58	NO
	p89_23_27_n.txt	71	NO
	p90_23_35_n.txt	101	NO
	p91_24_33_n.txt	96	NO
	p92_24_26_n.txt	58	NO
	p93_24_27_n.txt	55	SI
	p94_24_31_n.txt	85	NO
	p95_24_27_n.txt	59	NO
	p96_24_27_n.txt	54	NO
	p97_24_26_n.txt	57	NO
	p98_24_29_n.txt	71	SI
	p99_24_27_n.txt	62	NO
	p100_24_34_n.txt	98	NO
Tree	1rot/TREE_22_3_rot1.mtx.rnd	35	NO
	1rot/TREE_22_3_rot2.mtx.rnd	35	NO
	1rot/TREE_22_3_rot3.mtx.rnd	34	NO
	1rot/TREE_22_3_rot4.mtx.rnd	35	NO
	1rot/TREE_22_3_rot5.mtx.rnd	34	NO
	2rot/TREE_22_3_rot1.mtx.rnd	36	NO
	2rot/TREE_22_3_rot2.mtx.rnd	34	NO
	2rot/TREE_22_3_rot3.mtx.rnd	36	NO
	2rot/TREE_22_3_rot4.mtx.rnd	36	NO
	2rot/TREE_22_3_rot5.mtx.rnd	34	NO
	3rot/TREE_22_3_rot1.mtx.rnd	37	NO

	3rot/TREE_22_3_rot2.mtx.rnd	34	NO
	3rot/TREE_22_3_rot3.mtx.rnd	37	NO
	3rot/TREE_22_3_rot4.mtx.rnd	34	NO
	3rot/TREE_22_3_rot5.mtx.rnd	37	NO
HB	bcspwr01.mtx.rnd	100	NO
	bcspwr02.mtx.rnd	161	NO
	can___24.mtx.rnd	193	NO

Las instancias que no alcanzan el óptimo es porque no logran revisar todas las soluciones antes de que se termina el tiempo límite.

REFERENCIAS

[Blum, 2003] Christian Blum and Andrea Roli. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. ACM Computing Surveys (CSUR). Volume 35 Issue 3, Pages 268-308 , September 2003.

[Castilla, 2013] Castilla Valdez Guadalupe. Metaheuristicas aplicadas a la solución del problema de ordenamiento lineal. Tésis doctoral. Instituto Tecnológico de Tijuana. Junio 2013

[Cohen, 2014] Cohen Nathann and David Coudert, Sage, *Vertex Separation*, http://www.sagemath.org/doc/reference/graphs/sage/graphs/graph_decompositions/vertex_separation.html, 2014

[Crespelle, 1964] Crespelle, C., Todinca, I.: An $o(n^2)$ algorithm for the minimal interval completion problem (1964)

[Duarte, 2007] Duarte, A., Pantrigo, J., Gallego, M.: Metaheuristicas, vol. 1, 1a edn. Dykinson , 2007.

[Ellis, 1994] Ellis J.A., Sudborough I.H., Turner J.S.. The vertex separation and search number of a graph. *Journal Information and Computation*, 1994; 113:50–79.

[Fortet, 1959] R. Fortet, *"Applications de l'algébre de boole en recherche opérationnelle"*, Revue Francaise de Recherche Opérationelle 4 (1960) 17–26, 1959.

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

[Glover et al, 1986] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13:533 549, 1986.

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

[Karp, 1993] Richard M. Karp, "*Mapping the genome: some combinatorial problems arising in molecular biology*," in Proceedings of the twenty-fifth annual ACM symposium on Theory of computing, New York, NY, USA, 1993, STOC '93, pp. 278-285, ACM.

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

[Kendall, 1993]R. Kendall, "*Incidence matrices, interval graphs and seriation in archaeology*," STOC'93 Proceedings of the twenty-fifth annual ACM symposium on Theory of computing, 1993.

[Lewis, 1993] Lewis, R.: Simulated annealing for profile and fill reduction of sparse matrices. Master's thesis, University of British Columbia (1993)

[Liberti, 2007] Liberti, L., 2007. "*Compact linearization for binary quadratic problems"*, 4OR: A Quarterly Journal of Operations Research 5 (3), 231-245.

[López, 2012] López Loces Mario. Estrategias de búsqueda local para el problema del CUTWIDTH. Instituto Tecnológico de Cd. Madero. Diciembre 2012

[Martin, 2008] Martí, R., V. Campos and E. Piñana, Branch and Bound for the Matrix Bandwidth Minimization.*European Journal of Operational Research*, 186:513-528, 2008.

[Martí, 2012] Martí, Rafael, Juan J. Pantrigo, Abraham Duarte, Eduardo G. Pardo, *Branch and bound for the Cutwidth Minimization Problem*, Computers and Operations Research, 2012.

[Michalewicz, 2000] Z. Michalewicz and D.B. Fogel. *How to Solve it: Modern Heuristic*. Springer-Verlag, Berlin, Heidelberg, Germany, 2000.

[Papadimitriou, 1998] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization*. Dover Publications, INC, 1998.

[Petit, 2011] Petit, Jordi, *"Addenda to the Survey of Layout Problems"*. European Association for Theoretical Computer Science, pp. 177-201, Bulletin of the EATCS no 105, 2011.

[Raspaud et al., 2009] Raspaud, A., H. Schröder, O. Sýkora, L. Török, and I. Vrt'o. Antibandwidth and cyclic antibandwidth of meshes and hypercubes. *Discrete Mathematics.* 309:3541-3552, 2009.

[Ravi, 1991] R. Ravi, Ajit Agrawal, and Philip Klein, *"Ordering problems approximated: single-processor scheduling and interval graph completion,"* in Automata, Languages and Programming, Javier Albert, Burkhard Monien, and Mario Artalejo, Eds., vol.

510 of Lecture Notes in Computer Science, pp. 751-762. Springer Berlin / Heidelberg, 1991, 10.1007/3-540-54233-7 180.

[Sánchez-Oro, 2011] Sanchez-Oro, J., Duarte, A. *"Grasp for the sumcut problem."* 17th International Congress on Computer Science Research (2011)

[Sánchez-Oro, 2012] Sánchez-Oro, J., Duarte, A., *"GRASP with Path Relinking for the SumCut Problem"*, International Journal of Combinatorial Optimization Problems and Informatics, Vol. 3, pp. 3-11, 2012.

[Terán, 2013] Terán Villanueva David. Metaheuristicas aplicadas a la solución del problema de ordenamiento lineal con costos acumulados. Tésis doctoral. Instituto Tecnológico de Tijuana. Junio 2013

[Yuan, 1998] J. Yuan, Y. Lin, Y. Liu, and S. Wang, "*Np-completeness of the profile problem and the fill-in problem on cobipartite graphs,"* Journal of Mathematical Study, 1998.

[Zamarrón, 2013] Zamarrón Escobar Daniel. Estrategias de búsuqeda local para el problema del SUMCUT. Instituto Tecnológico de Cd. Madero. Junio 2013.