Instituto Tecnológico de Ciudad Madero

División de Estudios de Posgrado e Investigación

TESIS

Estrategias de Búsqueda Local Para el Problema del Ancho de Corte

PARA OBTENER EL GRADO DE:
MAESTRO EN CIENCIAS EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA:
ING. MARIO CÉSAR LÓPEZ-LOCÉS

DIRECTOR DE TESIS:
DR. HECTOR JOAQUÍN FRAIRE HUACUJA

Cd. Madero, Tamaulipas Octubre 2012

[redacted]

SUBSECRETARÍA DE EDUCACIÓN SUPERIOR
DIRECCIÓN GENERAL DE EDUCACIÓN SUPERIOR TECNOLÓGICA
INSTITUTO TECNOLÓGICO DE CIUDAD MADERO

SECRETARÍA DE EDUCACIÓN PÚBLICA

Cd. Madero, Tamps; a 10 de Octubre de 2012.

OFICIO No.: U5.213/12
AREA: DIVISIÓN DE ESTUDIOS
DE POSGRADO E INVESTIGACIÓN
ASUNTO: AUTORIZACIÓN DE IMPRESIÓN
DE TESIS

C. ING. MARIO CÉSAR LÓPEZ LOCÉS
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 en Ciencias de la Computación, se acordó autorizar la impresión de su tesis titulada:

"ESTRATEGIAS DE BÚSQUEDA LOCAL PARA EL PROBLEMA DEL ANCHO CORTE"

Es muy satisfactorio para la División de Estudios de Posgrado e Investigación compartir con Usted el logro de esta meta. Espero que continúe con éxito su desarrollo profesional y dedique su experiencia e inteligencia en beneficio de México.

A T E N T A M E N T E
"Por mi patria y por mi bien"

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

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

c.c.p.- Archivo
Minuta

MYCHC 'NLCO *arpc*

Ave. 1°. de Mayo y Sor Juana I. de la Cruz, Col. Los Mangos, C.P. 89440 Cd. Madero, Tam.
Tels. (833) 3 57 48 20, Fax: (833) 357 48 20, Ext. 1002, email: itcm@itcm.edu.mx
www.itcm.edu.mx

ISO 9001 CERTIFIED COMPANY
ISO 14001 CERTIFIED COMPANY

A mi familia.

DECLARACIÓN DE ORIGINALIDAD

Declaro y prometo que este documento de tesis es producto de mi trabajo original y que no infringe los derechos de 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 comillas) y en los resúmenes que he realizado de publicaciones ajenas, indico explícitamente los datos de los autores y publicaciones.

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

27 de Agosto del 2012, Cd. Madero, Tam.

Ing. Mario César López Locés

AGRADECIMIENTOS

A mi familia. Siempre. Por ser la principal motivación que tengo para emprender todo lo que hago y mi apoyo para poder lograrlo.

A mi director de tesis, Dr. Hector Joaquín Fraire Huacuja y a mi Codirector no oficial, M.C. David Terán Villanueva. Por la dedicación y apoyo brindados a este trabajo, por la confianza otorgada y por ser parte en mi formación como investigador y profesional en el área de las Ciencias de la Computación.

A la Dra. Claudia Guadalupe Gómez Santillán, Dr. Juan Javier González Barbosa y Dr. Arturo Hernández, por la revisión cuidadosa que han realizado de este texto y por sus valiosas sugerencias.

A mis profesores y compañeros que han contribuido con su ayuda en este trabajo de investigación.

Resumen

El presente trabajo de investigación se enfoca en el análisis de funciones de búsquedas locales, implementado diferentes configuraciones de los distintos componentes que las integran, para resolver instancias del Problema del Ancho de Corte.

La evaluación del desempeño de las funciones desarrolladas se realizó tanto con los métodos de búsqueda local en solitario, como embebidos en las metaheurísticas GRASP y el Algoritmo de Procesamiento Celular, para determinar cual de estas estrategias obtienen mejores resultados solucionando los conjuntos de instancias considerados.

Con los estudios experimentales realizados, se concluyó que es la sinergía entre las funciones de búsqueda local y las metaheurísticas las que son capaces de brindar soluciones de mayor calidad, con resultados comparables a los del estado del arte. Estos resultados fueron avalados por la prueba estadística de Friedman dado un nivel de significación α de 0.05.

Abstract

The present research focuses on the analysis of local search functions, implementig different configurations of the various components within them, for solving instances from the Cutwidth Problem.

Performance evaluation of the developed functions was conducted with the local search methods alone and embedded in the metaheuristics GRASP and Cellular Processing Algorithm to determine which of these strategies outperform solving the sets of instances considered.

After the completion of the experimental studies, it was concluded that it is the synergy between the local search functions and the metaheuristics which is capable of providing high quality solutions, with results comparable to those in of the state of the art. Results were assessed by the Friedman test statistic given a significance level *alpha* of 0.05.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE ALGORITMOS

ÍNDICE DE TABLAS

CAPÍTULO 1

INTRODUCCIÓN

> "The process of preparing programs for a digital computer is especially attractive, not only because it can be economically and scientifically rewarding, but also because it can be an aesthetic experience much like composing poetry or music."
>
> *The Art of Computer Programming*
> Donald Knuth

En la actualidad, numerosos problemas cuya complejidad de solución es sumamente alta son encontrados con frecuencia en áreas críticas de la industria. La complejidad de este tipo de problemas radica en la imposibilidad de encontrar su solución óptima en un tiempo considerado razonable para la tarea que se quiere resolver.

Sin embargo, con ciertas herramientas es posible encontrar soluciones que si bien no son las óptimas, puedan ser satisfactorias en el contexto en el que se lleva a cabo la actividad en el cual es encontrado el problema.

Entre estas herramientas se encuentran las metaheurísticas y los algoritmos de búsqueda local, los cuales si son diseñados eficientemente, pueden ser capaces de encontrar soluciones de muy alta calidad, en un menor tiempo que el empleado para conseguir una solución exacta.

Es por ello que este trabajo de tesis se centra en un problema de optimización, que es encontrado en áreas críticas tales como las telecomunicaciones y la electrónica, denominado Problema del Ancho de Corte o CWP por su nombre en inglés CutWidth Problem.

El Problema del Ancho de Corte se define formalmente de la siguiente manera:

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

El CutWidht (CW) de un vértice v con respecto a π $(CW_\pi(v))$, es el número de aristas $(u, w) \in E$ que satisfacen $\pi(u) \leq \pi(v) < \pi(w)$.

El CW del grafo G con respecto a π $(CW_\pi(G))$, es el máximo CW de todos sus vértices:

$$CW_\pi(G) = max_{v \in V} CW_\pi(v)$$

El objetivo del Problema del Ancho de Corte, consiste en encontrar el arreglo lineal π, tal que su CW sea el mínimo de todas las permutaciones posibles π^*:

$$CWP(G) = min_{\pi \in \pi *} CW_\pi(G)$$

Este trabajo de tesis muestra como herramientas de búsqueda local, así como sus respectivas funciones de cálculo de la función objetivo, si se encuentran altamente depuradas, son capaces de lograr un desempeño alto con respecto a los valores objetivos de las instancias consideradas en la experimentación de acuerdo a sus valores óptimos o mejores conocidos, ello en un

tiempo menor que el obtenido por aquellas del estado del arte.

Para ello, en este documento de tesis se muestra como fueron diseñadas e implementadas las funciones de calculo de la función objetivo, los mecanismos de búsqueda local y las metaheurísticas que hacen uso de ellas, así como los estudios comparativos entre la mejor configuración obtenida y aquellas del estado del arte.

1.1. Objetivos del Proyecto

1.1.1. Objetivo General

Diseñar e implementar búsquedas locales eficientes para resolver CWP.

1.1.2. Objetivos Específicos

- Implementar la función objetivo.
- Implementar la vecindad de inserción.
- Implementar la función de predicción del costo de una inserción.
- Diseñar e implementar funciones de búsqueda local
- Evaluar diferentes estrategias de búsqueda local.

1.2. Justificación

El Problema del Ancho de Corte se encuentra en la categoría de los problemas de complejidad NP-Duros [Garey and Johnson, 1976], lo que significa que para obtener la solución de una instancia del problema se llega a un punto en que si el tamaño de esta supera un cierto orden de magnitud, el obtener su solución óptima mediante algún algoritmo determinista es prácticamente imposible. Además, siendo un problema que es encontrado en áreas tan

importantes como las telecomunicaciones y el diseño de circuitos integrados, es primordial encontrar alguna estrategia que ayude a su resolución en un espacio de tiempo adecuado para la tarea que se realiza.

Debido a ello, el desarrollo de una metaheurística, que haga uso de mecanismos locales eficientes para que resulte escalable en concordancia con el tamaño de la instancia que se desea resolver, es realmente importante, pues aunque esta no garantice la obtención de su solución óptima, encuentra una alternativa razonable en el contexto en el que se encuentra el problema.

1.3. Hipótesis

El desarrollo de mecanismos de búsqueda local eficientes y de metaheurísticas que hagan uso de ellos, hacen posible obtener soluciones equiparables en calidad a aquellas del estado del arte.

1.4. Antecedentes del Proyecto

Hasta el año 2010, de acuerdo con el sitio del proyecto OPTSICOM de la Universidad de Valencia, España [Martí R. and E.G., 2010]. Los métodos heurísticos desarrollados para resolver CWP más relevantes son los siguientes:

GRASP+Reencadenamiento Trayectorial Es un método híbrido desarrollado por [Andrade and Resende, 2007b], el cual combina GRASP con Reencadenamiento Trayectorial.

GRASP+Reencadenamiento Trayectorial Evolutivo Método híbrido desarrollado por [Andrade and Resende, 2007a], el cual combina la metaherística GRASP con Reencadenamiento Trayectorial Evolutivo.

Recocido Simulado Implementa el Recocido Simulado empleando diferentes métodos constructivos y mecanismos de búsqueda local tal como es descrito en [Cohoon and Sahni, 1983].

Búsqueda Dispersa Este desarrollo consiste en un Algoritmo de Búsqueda Dispersa que emplea a GRASP como método constructivo y una estrategia de búsqueda local basada en movimientos de inserción y combinación basada en votación [Pantrigo et al., 2010].

1.5. Alcances y Limitaciones

Entre los alcances del proyecto se encuentran los siguientes puntos:

- El presente trabajo de investigación tiene como propósito el diseño e implementación de mecanismos de búsqueda local que resulten eficientes en la búsqueda de óptimos locales de soluciones de problemas de ordenamiento lineal.
- El problema de ordenamiento lineal particular que será objeto de estudio es CWP.
- Los conjuntos de instancias de CWP empleados para verificar la factibilidad y desempeño de los métodos de búsqueda local desarrollados incluyen a Grids, Harwellboeing y Small.

CAPÍTULO 2

MARCO TEÓRICO Y TRABAJOS RELACIONADOS

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

2.1. Problema de Decisión

En [Garey et al., 1979] se define un problema de decisión como una tripleta (L, U, Σ), donde Σ es un alfabeto y $L \subseteq U \subseteq \Sigma^*$. Un algoritmo A resuelve (decide) el problema de decisión (L, U, Σ) si para cada $x \in U$,

1. $A(x) = 1$ si $x \in L$, y
2. $A(x) = 0$ si $x \in U - L (x \notin L)$

En el problema de decisión se intenta verificar si una solución candidata resuelve o no la instancia que se evalúa. El problema de optimización, por

otra parte, busca la mejor solución posible para el problema que se intenta resolver.

2.2. Problema de Optimización

De acuerdo con [Garey et al., 1979] un problema de optimización es una 7-tupla $U = (\Sigma_I, \Sigma_O, L, L_I, M, cost, goal)$ donde,

1. Σ_I es un alfabeto, denominado el alfabeto de entrada de U
2. Σ_O es un alfabeto, llamado el alfabeto de salida de U
3. $L \subseteq \Sigma_I^*$ es el lenguaje de instancias factibles del problema
4. $L_I \subseteq L$ es el lenguaje de la actual instancia del problema de U
5. $M(x)$ es llamado el conjunto de soluciones factibles para x
6. *cost* es la función de costo que para cada par (u, x) donde $u \in M(x)$ para alguna $x \in L$, asigna un número real positivo $cost(u, x)$
7. $goal \in \{minimum, maximum\}$

Dado que para resolver un problema de optimización, la solución candidata explora a través del espacio de búsqueda de la instancia a evaluar, es posible que, en la búsqueda de la mejor solución al problema u óptimo global, la solución quede atrapada en regiones que sean óptimas localmente dado el estado actual en el que se encuentra. Estos conceptos serán definidos en las subsecciones siguientes.

2.2.1. Óptimo global de un problema de minimización

Dado un problema de optimización, en el que se pretende minimizar el valor de la función objetivo f, se dice que una solución factible $s_x \in S' \subseteq S$

(S' es el espacio de soluciones factibles, y S es el espacio de búsqueda) es un óptimo global si $\forall s_y \in S$, $f(s_x) \leq f(s_y)$ [Duarte et al., 2006].

2.2.2. Óptimo local de un problema de minimización

Dado un problema de optimización en el que se pretende minimizar la función objetivo y, una estructura de vecindad N, se dice que una solución factible $s_x \in S' \subseteq S$, es un óptimo local si $\forall s_y \in N(s_x)$, $f(s_x) \leq f(s_y)$ [Duarte et al., 2006].

2.3. Complejidad Computacional

La teoría de la complejidad computacional es la rama de la teoría de la computación que estudia los problemas de decisión, así como los recursos requeridos por los algoritmos que los resuelven. Los problemas pueden clasificarse como P, NP, NP-completos y NP-Duros[Garey et al., 1979].

Clase P Es el conjunto de todos los problemas de decisión que pueden ser resueltos en tiempo polinomial por un algoritmo determinista. A los problemas que pertenecen a esta clase se les conoce por el nombre de tratables[Garey et al., 1979].

Clase NP La clase NP es el conjunto de todos los problemas de decisión que se pueden verificar en tiempo polinomial con un algoritmo no determinista. 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 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 toda máquina determinista es un caso particular de una máquina no determinista, se tiene entonces que $P \subseteq NP$[Garey et al., 1979].

Problema NP-Duro Numerosos problemas derivados de aplicaciones prácticas pertenecen a la clase NP-Duro que incluye a los problemas para los que no existe actualmente algún algoritmo de tiempo polinomial capaz de resolverlos hasta la optimalidad. La dificultad para resolver estos problemas mediante algoritmos de tiempo polinomial ha impulsado el desarrollo de métodos (tales como los métodos heurísticos y metaheurísticos) que mejoren la solución generada por un algoritmo no determinista, los cuales ofrecen soluciones aproximadas en tiempos razonables con bajos requerimientos computacionales[Garey et al., 1979].

Para la resolución de las distintas clases de problemas presentados en esta sección, se han desarrollado con los años, métodos que brinden soluciones exactas o cuando menos de una calidad adecuada para la tarea que se necesita resolver. A estos métodos de solución, se les denomina algoritmos, lo cuales pueden ser clasificados como deterministas o no deterministas, que serán definidos a continuación.

2.4. Algoritmos Deterministas

Un algoritmo determinista es aquel que se se comporta de manera predecible, es decir, que dada una entrada particular, siempre devuelve la misma

salida, siguiendo en todos los casos el mismo proceso para conseguirlo. Sin embargo, este tipo de algoritmos no son eficientes en la solución de problemas NP-Duros, pues hasta el momento no ha sido posible encontrar un algoritmo determinista que resuelva alguno de ellos en tiempo polinomial [Duarte Muñoz et al., 2007].

2.5. Algoritmo No Determinista

Para la resolución de problemas NP-Duros, que necesitan ser solucionados en un tiempo polinomial, se emplean Algoritmos No Deterministas.

Los Algoritmos No Deterministas se diferencian de su contraparte determinista, en el hecho de que implementan una etapa adicional, denominada de adivinación, la cual genera una solución aleatoria para una determinada instancia del problema, que es luego revisada por otra etapa, denominada de verificación, cuya función es decidir si la solución propuesta resuelve o no a la instancia del problema presentado [Duarte Muñoz et al., 2007].

Dentro de esta categoría caen los métodos heurísticos y metaheurísticos.

2.5.1. Métodos Heurísticos

Las heurísticas utilizan reglas, estrategias o programas basados generalmente en conocimientos empíricos para guiar el conocimiento. La palabra heurística procede del griego y significa «hallar, inventar» (etimología que comparte con eureka) [Polya, 1971]. Nicholson y Reeves ofrecen las siguientes definiciones de heurística.

- "Un procedimiento para resolver problemas por medio de un método intuitivo en el que la estructura del problema puede interpretarse y explotarse inteligentemente para obtener una solución razonable" tal como es descrito en [Nicholson, 2007].

- "Una técnica que busca buenas soluciones con un tiempo de computación razonable sin garantizar la optimalidad" [Reeves, 1993].

2.5.2. Métodos Metaheurísticos

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

- "Los procedimientos Metaheuristicos son una clase de métodos aproximados que están diseñados para resolver problemas difíciles de optimización combinatoria, en los que los heurísticos clásicos no son eficientes" [Duarte et al., 2006].

- "Una metaheurística es un proceso iterativo que combina conceptos derivados de las ciencias naturales, la biología, la matemáticas, la física y la inteligencia artificial, para guiar de forma inteligente una heurística subordinada durante la exploración y explotación del espacio de búsqueda con el fin de encontrar soluciones cercanas al óptimo de forma eficiente" [Kelly and Osman, 1996].

2.6. Búsqueda Local

Los enfoques computacionales para resolver problemas de optimización combinatoria consisten básicamente en métodos de búsqueda. Estos métodos funcionan generando y evaluando iterativamente soluciones candidatas. De estos dos procesos , aún para problemas combinatorios NP-Duros que presenten complejidad de crecimiento exponencial, la evaluación de las soluciones generalmente puede realizarse de manera eficiente en tiempo polinomial, sin embargo, es el proceso de generación de soluciones el que tiene un mayor impacto sobre las propiedades teóricas y la eficiencia práctica de los algoritmos de búsqueda.

Los algoritmos de búsqueda pueden clasificarse en sistemáticos, y locales. Los primeros atraviesan el espacio de búsqueda de forma sistemática, garantizando que encontrarán la solución óptima si existe.

Los algoritmos de búsqueda local también llamados de mejora iterativa, parten de una solución inicial ubicada en cualquier punto del espacio de búsqueda y se mueven a una solución vecina mediante un movimiento. Cada solución tiene un número relativamente bajo de vecinos y cada movimiento esta determinado por una decisión basada únicamente en información local. Estos algoritmos no garantizan que se alcance el óptimo y además pueden visitar la misma solución varias veces durante la exploración del espacio de búsqueda. Entre los algoritmos de búsqueda local están la caminata aleatoria no informada (Uninformed Random Walk), remontando la colina (Hill Climbing), 2-Opt, entre otros [Schiavinotto and Stützle, 2004].

De acuerdo con [Schiavinotto and Stützle, 2004], dado un problema Π, un algoritmo de búsqueda local que resuelve una instancia $\pi \in \Pi$, esta definido por los siguientes componentes:

- El espacio de búsqueda $S(\pi)$ de la instancia π, el cual es un conjunto finito de soluciones candidatas $s \in S$ conocidas también como permutaciones, configuraciones o estados.
- Un conjunto de soluciones factibles $S'(\pi) \subseteq S(\pi)$
- Una relación de vecindad en $S(\pi)$, $N(\pi) \subseteq S(\pi) \times S(\pi)$
- Un conjunto finito de estados memoria $M(\pi)$, o dado el caso de que el algoritmo de búsqueda local no implemente una estructura de memoria, su tamaño será de uno.
- Una función de inicialización $init(\pi) : \emptyset \mapsto D(S(\pi) \times M(\pi))$, la cual especifica una distribución de probabilidad sobre las posiciones iniciales de búsqueda y estados de memoria.

- Una función de movimiento $step(\pi) : S(\pi) \times M(\pi) \mapsto D(S(\pi) \times M(\pi))$ la cual asigna a cada posición y estado de memoria una distribución de probabilidad sobre las posiciones y estados de memoria de su vecindario
- Una condición de parada $terminate(\pi) : S(\pi) \times M(\pi) \mapsto D(\{\top, \bot\})$ que asigna a cada posición y estado de memoria una distribución de probabilidad sobre los valores de verdad $\top = True, \bot = False$, los cuales indican la probabilidad con la cual la búsqueda local terminará al alcanzar un punto específico en el espacio de búsqueda o los estados de memoria.

En el Algoritmo 1, se puede observar la estructura básica de un Método de Búsqueda Local.

Algoritmo 1 *busquedaLocal*

Require: V Arreglo lineal del grafo.
Require: E Arreglo con elementos a reposicionar.

1: $E = seleccionaElementos()$
2: **while** $E = \emptyset$ **do**
3: $i \in E$
4: $j = \arg\max_{k \in \{1,2,\ldots,n\}} \{ganancia\{predInsert(V, k, i)\}\}$
5: $V = insert(V, j, i)$
6: $E = E - \{i\}$
7: **end while**

2.6.1. Permutación

La búsqueda local opera sobre una solución candidata, la cual, es codificada en una estructura de datos en una representación dada, para ser manipulada por esta. A esta representación de una solución dada, se le denomina permutación y es definida de la siguiente manera.

Sea n un entero positivo, $\pi = \{a_1, a_2, \ldots, a_n\}$ y sea a un conjunto de n objetos o elementos. Una permutación de n objetos $a_1, \ldots, a_n$ es un arreglo

ordenado de los objetos en π [Hromkovic and Oliva, 2001].

La exploración que realiza la función de búsqueda local es realizada a partir del estado actual de la permutación de la solución candidata que se desea mejorar, por medio de movimientos que pueden realizarse dentro del espacio de búsqueda, los cuales se denominan vecindarios. A continuación se describirán algunas de las funciones de vecindad más importantes.

2.6.2. Vecindad de Inserción

Los movimientos de inserción son uno de los mecanismos básicos para moverse de una solución a otra en un espacio de búsqueda. Un movimiento de inserción en la permutación π, $insert(\pi, j, i)$ consiste en eliminar el elemento π_j de su posición actual j para ser insertado en la posición i.

Esta operación transforma la permutación π en π':

Si $i < j$ entonces

$$\pi' = (\pi_1, \pi_2, \ldots, \pi_{i-1}, \pi_j, \pi_i, \ldots, \pi_{j-1}, \pi_{j+1}, \ldots, \pi_n)$$

Si $i > j$ entonces

$$\pi' = (\pi_1, \pi_2, \ldots, \pi_{j-1}, \pi_{j+1}, \ldots, \pi_i, \pi_j, \pi_{i+1}, \ldots, \pi_n)$$

El mecanismo de acción de la función de inserción se ejemplifica el la Figura 4.3.

Vecindad de Inserción Consecutiva

La inserción consecutiva, puede verse como un caso particular de la función de inserción, en el que los movimientos que realiza el elemento π_j hasta llegar a la posición i, se hacen a través de intercambios con su vecino inmediato, como puede verse en la Figura 4.4.

El tamaño de la estructura de vecindario generada por la función de inserción es de $|N(\pi)_I| = (n-1)^2$ [Schiavinotto and Stützle, 2004].

2.6.3. Vecindad de Intercambio

Otro mecanismo para la construcción de una estructura de vecindad lo constituye la función de intercambio. Un movimiento de intercambio en la permutación π, $interchange(\pi, j, i)$ consiste en intercambiar el elemento π_j de su posición actual j con el elemento en la posición i.

Esta operación transforma la permutación π en π':

$$\pi' = (\pi_1, \pi_2, \ldots, \pi_{i-1}, \pi_j, \pi_{i+1}, \ldots, \pi_{j-1}, \pi_i, \pi_{j+1}, \ldots, \pi_n)$$

Un ejemplo de como la función de intercambio actúa sobre una permutación, puede observarse en la Figura 4.2.

El tamaño de la estructura de vecindario generada por la función de intercambio es de $|N_X| = n(n-1)/2$ [Schiavinotto and Stützle, 2004].

2.7. Grafos

Algunos problemas de optimización, incluido el que trata el presente trabajo de investigación, pueden ser representados por medio de estructuras matemáticas denominadas grafos.

Un grafo no dirigido G es un par (V, E), donde V es un conjunto finito denominado conjunto de vértices de G y E es un subconjunto de $\{\{u, v\}|v, u \in Vyv \neq u\}$, llamado conjunto de aristas de G [Hromkovic and Oliva, 2001].

2.7.1. Grado de un Grafo

El grado de un vértice v de G, $deg_G(v)$, es el número de arcos incidentes en v. El grado de un grafo $G = (V, E)$ es

$$deg(G) = max\{deg_G(v)|v \in V\}$$

2.7.2. Grafo Completo

Para cualquier $n \in N$, $K_n = (\{v_1, \ldots, v_n\}, \{\{v_i, v_j\} | i, j \in \{1, \ldots, n\}, i \neq j\})$ es el grafo completo de n vértices.

2.7.3. Grafo Conexo

Un grafo $G = (V, E)$ se denomina conexo si para toda $x, y \in V, x \neq y$, existe un camino entre x e y en G

2.8. Matriz de Adyacencia

Una manera de representar un grafo en forma matricial, de modo que pueda ser tratado por una computadora, es por medio de su matriz de adyacencia[Kreyszig, 2007], la cual se define de la siguiente manera:

$$a_{i,j} = \begin{cases} 1 & \text{si G tiene una arista } (i,j) \\ 0 & \text{en caso contrario} \end{cases}$$

2.9. Trabajos Relacionados

En esta sección se presentarán aquellos métodos de búsqueda local que se han implementado en metaheurísticas del estado del arte que resuelven problemas con una estructura de similar al de aquellas de CWP.

En [Andrade and Resende, 2007b] se describe el diseño de una metaheurística GRASP en combinación con Path-relinking, para resolver el problema de Network Migration Scheduling (Equivalente a CWP). El mecanismo de búsqueda local implementado en la metaheurística tal como es descríto en su artículo consiste en examinar todos los vértices en el arreglo del ordenamiento lineal por medio de movimientos de swap (caso particular del movimiento de inserción), determinando como afecta el intercambio los valores en los anchos de corte de la permutación y llevando a cabo el intercambio si existe

una mejora en la función objetivo. Este enfoque tiene un costo asintótico de $O(n^2)$.

En el artículo presentado por [Pantrigo et al., 2010], se describe el diseño de un algoritmo metahehurístico de Busqueda Dispersa que resuelve CWP. En el, se describe como la búsqueda local implementada se realiza a través de una estructura de vecindad de inserción, deteniéndose al momento de lograr la primer mejora en el valor de la función objetivo. Posteriormente, aplica un Path-relinking desde la solución actual, hasta la obtenida por la búsqueda local, generando paso a paso todos los posibles movimientos de una solución a la otra, deteniéndose cuando se alcanza la solución final y regresando la mejor solución encontrada en la trayectoria que se siguió. Hasta el momento, este enfoque es el que reporta los mejores resultados al resolver las instancias descrítas en esta propuesta.

Para el Problema de Ordenamiento Lineal con Costos Acumulados, descrito en [Duarte et al., 2006] se implementa un algoritmo metaheurístico de búsqueda Tabú, el cual contiene un mecanismo de búsqueda local empleado. Dicho algoritmo explora a través de una estructura de vecindad de inserción y consiste en que dada una permutación de solución del problema, se selecciona aleatoriamente uno de sus nodos, en donde la probabilidad de selección de cualesquiera de ellos es proporcional a una medida de influencia establecida. Posteriormente, se inicia la búsqueda del primer movimiento que genere una mejora en el valor objetivo de la solución o en caso de que ningún movimiento produzca mejora alguna, se selecciona el que empeore en menor medida el valor objetivo de la solución.

En el artículo publicado por [Schiavinotto and Stützle, 2004] se describe el desarrollo de la mejor alternativa de solución al Problema de Ordenamiento Lineal hasta la fecha. Esta consiste en un Algoritmo Memético que como método de búsqueda local implementa una búsqueda First, la cual se mueve a lo largo de una vecindad de intercambio.

En el artículo publicado por [Campos et al., 2006] se detalla el desarrollo

de una metaheurística híbrida de Tabu Search y Scatter Search como solución al problema Matrix Bandwidth Minimization. La solución propuesta implementa un método de búsqueda local de tipo First, el cual se mueve a través de una estructura de vecindades de intercambio e intercambio consecutivo y seleccionando los elementos a ser probados por medio de los denominados de una lista de vectores críticos y vectores críticos cercanos, los cuales son aquellos en los que, en el caso particular del problema Matrix Bandwidth Minimization son los elementos cuyo valor objetivo es igual al valor objetivo del grafo completo y los vectores críticos cercanos, los que aún sin determinar el valor objetivo del grafo, tienen un valor cercano a este.

CAPÍTULO 3

DESCRIPCIÓN DEL PROBLEMA

3.1. Problema de Ancho de Corte (CWP)

El Problema de Ancho de Corte es un problema de minimización que pertenece a la clase NP-Duro, tal como es definido en [Garey and Johnson, 1976], al mostrar la transformación polinomial de CWP al Problema de Partición. CWP es definido por [Pantrigo et al., 2010] de la siguiente manera:

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

El CutWidht (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 $\pi(u) \leq \pi(v) < \pi(w)$.

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

$$CW_\pi(G) = max_{v \in V} CW_\pi(v)$$

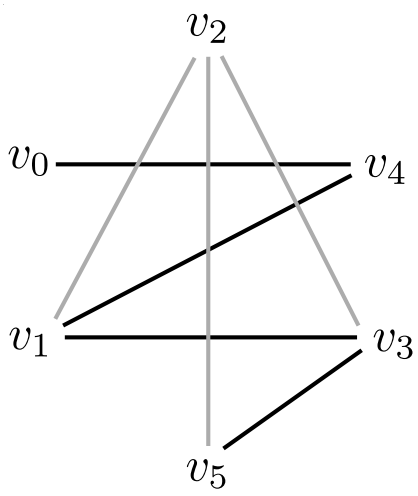


Figura 3.1: Grafo no dirigido de una instancia de CWP

El objetivo de CWP consiste en encontrar el arreglo lineal que minimize el CW del grafo.

$$CWP(\pi^*, G) = min_{\pi \in \pi^*} CW_\pi(G)$$

3.1.1. Cálculo Inicial de la Función Objetivo

Para un grafo no dirigido, $G = (V, E)$, como el mostrado en la Figura 3.1

Al cual se le asigna una etiqueta numérica por cada uno de sus nodos, donde:

$$\begin{aligned} v_1 &= 0 \\ v_2 &= 1 \\ v_3 &= 2 \\ v_4 &= 3 \\ v_5 &= 4 \\ v_6 &= 5 \end{aligned}$$

Y dispuesto en un arreglo lineal como se ilustra en la Figura 3.2, los

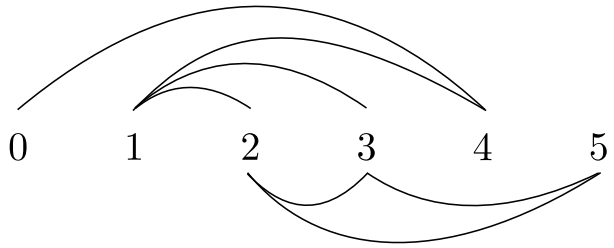


Figura 3.2: Arreglo Lineal de Grafo

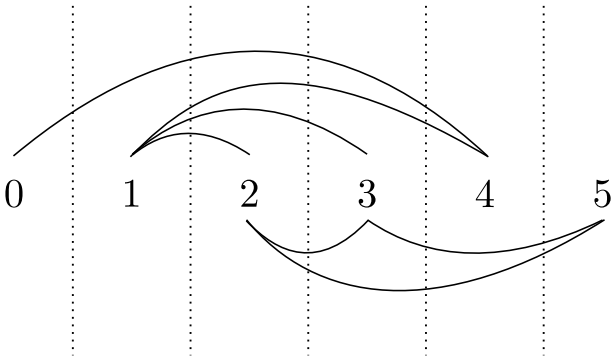


Figura 3.3: Anchos de Corte de Grafo (CutWidth)

anchos de corte para esta permutación son los que se pueden observar en la Figura 3.3.

Gráficamente puede verse que el CW de un vértice dado un orden lineal π corresponde al número de arcos que lo cruzan, más el número de vértices que surgen de él.

Para ejemplificar la definición anterior, obtendremos $CW(2)$, dado el arreglo lineal π de la Figura 3.2:

$$\begin{aligned} CW_\pi(2) &= \{\text{Número de arcos que cruzan 2}\} + \{\text{Número de arcos que surgen de 2}\} \\ &= \{e_{0,4} + e_{1,4} + e_{1,3}\} + \{e_{2,3} + e_{2,5}\} \\ &= 3 + 2 \\ &= 5 \end{aligned}$$

Como puede comprobarse gráficamente en $CW_\pi(2)$ de la Figura 3.3, en donde 5 arcos son cortados en ese punto.

El arreglo lineal puede ser representado por su matriz de adyacencia mostrada en la Figura 3.4, la cual es simétrica, debido a que se trata de un grafo no dirigido.

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \left[\begin{array}{cccccc} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{array}\right] \end{array}$$

Figura 3.4: Matriz de Adyacencia de Grafo.

Sea $\pi = v_1, v_2, \ldots, v_n$ una permutación dada por el arreglo lineal del grafo, y a la matriz de adyacencia de G entonces:

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

$$CW(v_1) = \sum_{j=2}^{n} a[v_1][v_j] \tag{3.1}$$

Una vez más, para ejemplificar, tomaremos el grafo de la Figura 3.1 dispuesto en el arreglo lineal de la Figura 3.2 y con

$$a = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \left[\begin{array}{cccccc} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{array}\right] \end{array}$$

El valor de CW en v_1 puede ser obtenido con la Ecuación 3.1:

$$\begin{aligned} CW(v_1) &= \sum_{j=2}^{n} a[v_1][v_j] \\ &= a[v_1][v_2] + a[v_1][v_3] + a[v_1][v_4] + a[v_1][v_5] + a[v_1][v_6] \\ &= 0 + 0 + 0 + 1 + 0 \\ &= 1 \end{aligned}$$

Tal como se puede comprobar en la Figura 3.3, en el que al efectuar un corte en la primera posición, se afecta solo un arco del grafo.

Remarcados en el recuadro color negro, se observan los elementos que componen el valor del *cutwidth* del nodo v_1 del arreglo lineal, en la Figura 3.5.

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

$$CW(v_k) = CW(v_{k-1}) + \sum_{j=k+1}^{n} a[v_k][v_j] - \sum_{j=1}^{k-1} a[v_j][v_k] \qquad (3.2)$$

Por ejemplo, dado el grafo de la Figura 3.1 dispuesto en el arreglo lineal

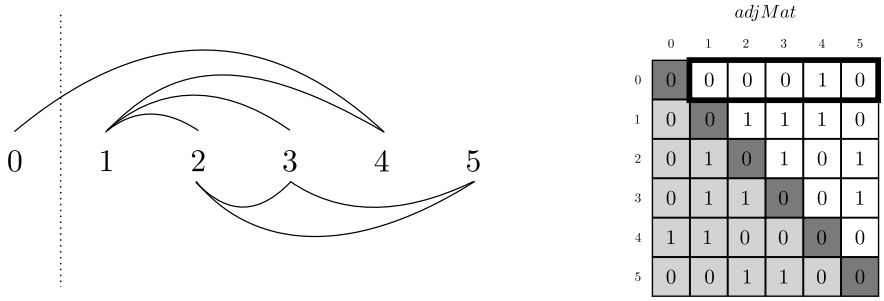


Figura 3.5: Cutwidth del Nodo v_1

de la Figura 3.2 y con

$$a = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

El valor de CW en v_5 puede ser calculado con la Ecuación 3.2:

$$
\begin{aligned}
CW(v_5) &= CW(v_4) + \sum_{j=6}^{6} a[v_5][v_j] - \sum_{j=1}^{4} a[v_j][v_5] \\
&= CW(v_4) + \{a[v_5][v_6]\} - \{a[v_1][v_5] + a[v_2][v_5] + a[v_3][v_5] + a[v_4][v_5]\} \\
&= CW(v_4) + \{0\} - \{1 + 1 + 0 + 0\} \\
&= CW(v_4) - 2 \\
CW(v_4) &= CW(v_3) + \sum_{j=5}^{6} a[v_4][v_j] - \sum_{j=1}^{3} a[v_j][v_4] \\
&= CW(v_3) + \{a[v_4][v_5] + a[v_4][v_6]\} - \{a[v_1][v_4] + a[v_2][v_4] + a[v_3][v_4]\} \\
&= CW(v_3) + \{0 + 1\} - \{0 + 1 + 1\} \\
&= CW(v_3) - 1 \\
CW(v_3) &= CW(v_2) + \sum_{j=4}^{6} a[v_3][v_j] - \sum_{j=1}^{2} a[v_j][v_3] \\
&= CW(v_2) + \{a[v_3][v_4] + a[v_3][v_5] + a[v_3][v_6]\} - \{a[v_1][v_3] + a[v_2][v_3]\} \\
&= CW(v_2) + \{1 + 0 + 1\} - \{0 + 1\} \\
&= CW(v_2) + 1 \\
CW(v_2) &= CW(v_1) + \sum_{j=3}^{6} a[v_2][v_j] - \sum_{j=1}^{1} a[v_j][v_2] \\
&= CW(v_1) + \{a[v_2][v_3] + a[v_2][v_4] + a[v_2][v_5] + a[v_2][v_6]\} - \{a[v_1][v_2]\} \\
&= CW(v_1) + \{1 + 1 + 1 + 0\} - \{0\} \\
&= CW(v_1) + 3 \\
CW(v_1) &= 1
\end{aligned}
$$

$$\begin{aligned} CW(v_2) &= 1+3 \\ &= 4 \\ CW(v_3) &= 4+1 \\ &= 5 \\ CW(v_4) &= 5-1 \\ &= 4 \\ CW(v_5) &= 4-2 \\ CW(v_5) &= 2 \end{aligned}$$

Tal como se puede comprobar en la Figura 3.3, en donde al realizar los cortes en los puntos 2, 3, 4 y 5 son afectados, 4, 5, 4 y 2 arcos respectivamente.

De la misma forma, en la Figura 3.6 se observan los elementos que sumados conforman el *cutwdith* del elemento v_k, cuando $1 < k < n$.

El valor de la función objetivo del ancho de corte (CW) de v_n, donde v_n es la última posición de la permutación, está dado en todos los casos por:

$$CW(v_n) = 0 \tag{3.3}$$

Puesto que ningún arco pasa sobre este o de este a nodos posteriores, ya que no existen.

Finalmente, el valor de la función objetivo del ancho de corte (CW) de la permutación π,es decir, del grafo completo, esta dado por:

$$CW_\pi(G) = max\{CW(v_1), CW(v_2), \ldots, CW(v_n)\} \tag{3.4}$$

Dado el mismo grafo de los ejemplos anteriores, mostrado en la Figura 3.3, del cual ya se obtuvieron los valores del CW en cada punto de corte, el CW del grafo completo se puede obtener a partir de la Ecuación 3.4:

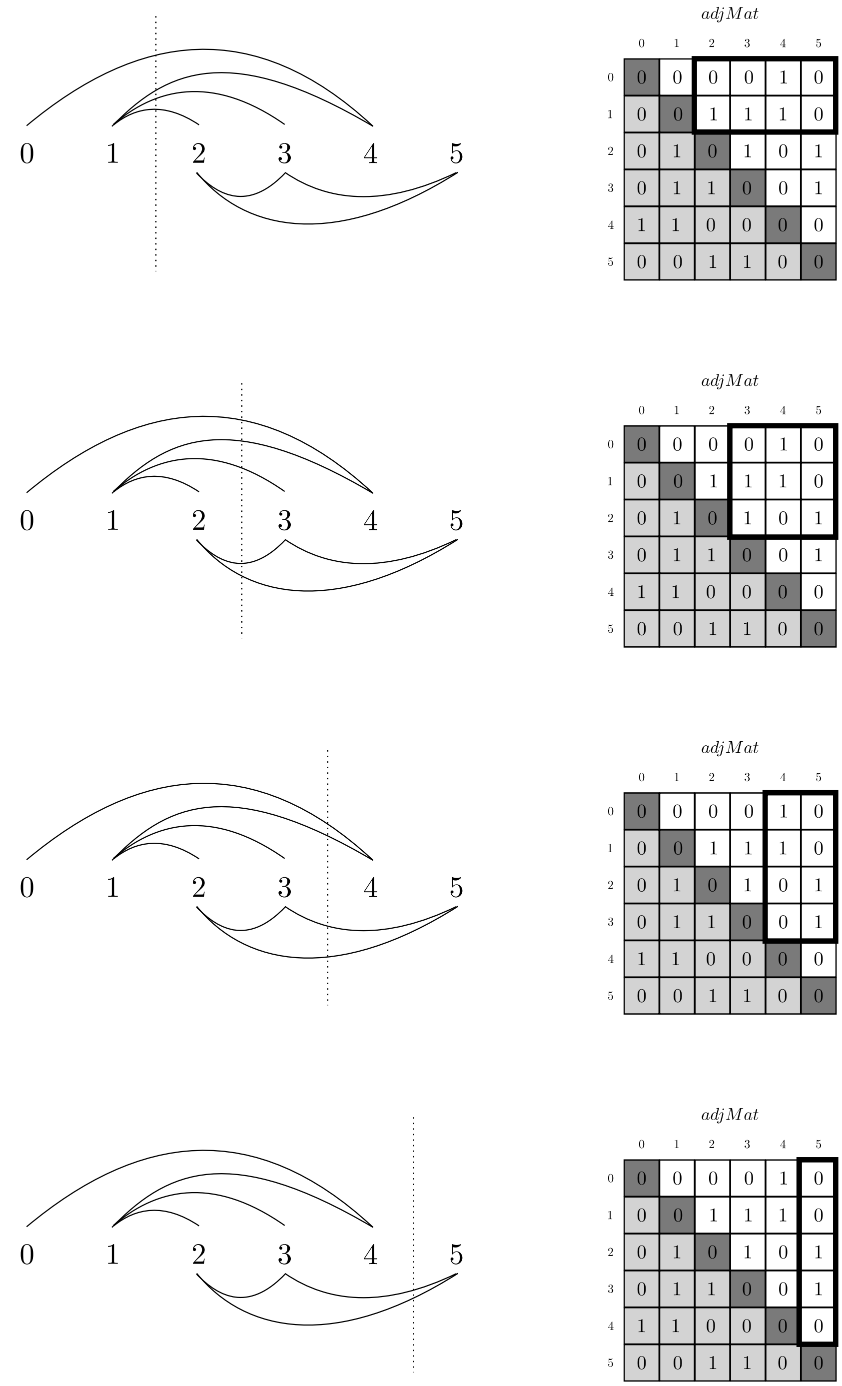


Figura 3.6: Cutwidth del Nodo v_k

$$\begin{aligned} CW_\pi(G) &= max\{CW(v_1), CW(v_2), CW(v_3), CW(v_4), CW(v_5), CW(v_6)\} \\ &= max\{1, 4, 5, 4, 2, 0\} \\ &= 5 \end{aligned}$$

3.2. Complejidad del Problema

En [Garey and Johnson, 1976] se demuestra que el Problema de Ancho de Corte (Cut Width), denominado Simple Max Cut por Gary y Johnson, pertenece a la clase NP-Completo, tal como fue probado por Karp al realizar su transformación al Problema de Partición.

CAPÍTULO 4

DISEÑO E IMPLEMENTACIÓN DE BÚSQUEDAS LOCALES

Dado que el enfoque principal del presente trabajo de tesis se centra en los algoritmos de búsqueda local, en este capítulo se mostrarán las diferentes configuraciones, así como resultados experimentales previos en los cuales se evalúa el desempeño de las búsquedas locales implementadas, y que servirá como núcleo de las metaheurísticas que resuelven el problema del ancho de corte, descritas en los capítulos siguientes.

La importancia del desarrollo de mecanismos de búsqueda local eficientes, radica en el hecho de que dentro del conjunto de los problemas de ordenamiento lineal, del cual forma parte el problema de ancho de corte, las metaheurísticas más exitosas en su resolución fueron aquellas que implementaban dentro de su proceso, algún tipo de búsqueda local.

En general, la estructura de un algoritmo de búsqueda local es relativamente sencillo, tal como puede verse en el Algoritmo 2, por lo que el que una implementación particular sea exitosa o no depende de como son configurados

sus componentes.

Algoritmo 2 *LS*

Require: *Instancia instancia*
Require: *Solucion solucion*

1: $V = EC(solucion)$
2: **repeat**
3: $v\_j \in V$
4: $i = BP(instancia, solucion, v\_j)$
5: $solucion = insert(solucion, i, v\_j)$;
6: $V \setminus \{v\_j\}$
7: **until** $V \neq \emptyset$
8: **return** *solucion*

Entre los componentes que se ajustaron, de acuerdo al análisis realizado por [Schiavinotto and Stützle, 2004], para obtener las diversas configuraciones se encuentra la función de movimiento que en el caso del problema que se aborda se subdivide en la selección de elementos críticos y el mecanismo de reposicionamiento de la lista de elementos críticos seleccionados y la función de inicialización, para reducir el tiempo de búsqueda al partir de soluciones de calidad.

El criterio de parada de la búsqueda local es determinado por estancamiento de la solución desde la cual se intenta obtener el óptimo a partir de la instancia que se intenta resolver, es decir, cuando no es posible mejorar la solución actual dado un número establecido de iteraciones sin un aumento en la calidad.

El vecindario elegido fue el inserción consecutiva, esto es debido a que por las características propias del problema, la búsqueda local se beneficia al implementar esta forma de vecindad al operar sobre funciones de cálculo de la función objetivo optimizadas, las cuales se detallarán en mayor detalle en este capítulo.

4.1. Funciones de Cálculo de la Función Objetivo de CWP

El primer paso para el diseño de funciones de búsqueda local eficientes, comienza con el desarrollo de funciones que calculen la función objetivo de una permutación del problema de manera empleando la menor cantidad de recursos de computo que pueda ser posible.

Por ello analizando la función original que calcula el valor objetivo de CWP, fue posible simplificar este proceso como se describirá a continuación en las siguientes subsecciones.

4.1.1. Función de Cálculo Inicial de la Función Objetivo

La reducción de costo de la complejidad computacional de la función que calcula el valor objetivo de la permutación inicial de una instancia del problema que se intenta resolver, es importante debido a que permite que la fase de optimización tenga una ventana de tiempo mucho mayor para operar sobre la solución que se intenta mejorar. Además, en la medida de lo posible, debe ser capaz de generar información relativa a la permutación actual para que sea explotada por las funciones de calculo del valor objetivo en subsecuentes iteraciones.

En el caso particular de CWP, partiendo del calculo gráfico original de los anchos de corte del grafo, el cual se puede consultar en las Figuras 3.5 y 3.6, se obtuvieron algunas observaciones.

En este caso con una permutación distinta a la del ejemplo anterior, el calculo del cutwidth del nodo que se encuentra en la primera posición del arreglo lineal mostrado en la Figura 4.6, cuyos elementos que conforman el cutwidth, se encuentran enmarcados en el recuadro en color negro coinciden con aquellos que componen la fila del triángulo superior de la matriz de adyacencia del nodo en cuestión. Por lo que se puede deducir que el cutwidth del primer elemento de la permutación coincide con el de la suma de los

elementos de su fila en el triángulo superior de la matriz de adyacencia de la matriz de la instancia del grafo.

Para los k elementos siguientes de la permutación, donde $1 < k < n$ y donde n es el índice del último elemento en la permutación, el cálculo del cutwidth es análogo al mostrado en la Figura 3.6, pero mostrado con una permutación distinta, específicamente la que se puede ver en la Figura 4.7.

En esta figura se observa que el cutwidth del elemento a evaluar contiene a su vez, una porción del cutwidth del nodo que le precede en la permutación, contenido en el recuadro gris, en la matriz de adyacencia del grafo. La porción contenida en el cutwitdth anterior, pero que no esta presente en el actual, corresponde a la suma de la columna del triángulo superior de la matriz de adyacencia del nodo a evaluar. Y finalmente la porción del cutwidth actual que no esta contenida en el cutwidth del elemento anterior, corresponde a la suma de la fila de triángulo de la matriz superior del nodo actual en la matriz de adyacencia del grafo.

Generalizando, el cálculo para cualquier elemento del arreglo lineal se puede ver de forma gráfica en la Figura 4.1.

Con estas observaciones en mente, se diseño el Algoritmo 3 el cual permite calcular el cutwidth de todos los elementos de un grafo de instancia, dado un arreglo lineal π.

Algoritmo 3 *InitialEvaluationCW*

Require: *Instance ins*
Require: *Solution sol*

1: $v_0 = sol.permutation.front()$
2: $v_n = sol.permutation.back()$
3: $v_k = sol.permutation[1]$
4: $sol.cutwidth[v_0] = sol.sumC[v_0]$
5: **for** v_k **to** v_n **do**
6: $\quad sol.cutwidth[v_k] = sol.cutwidth[v_k - 1] - sol.sumR[v_k] + sol.sumC[v_k]$
7: **end for**
8: $sol.cutwidth[v_n] = 0$

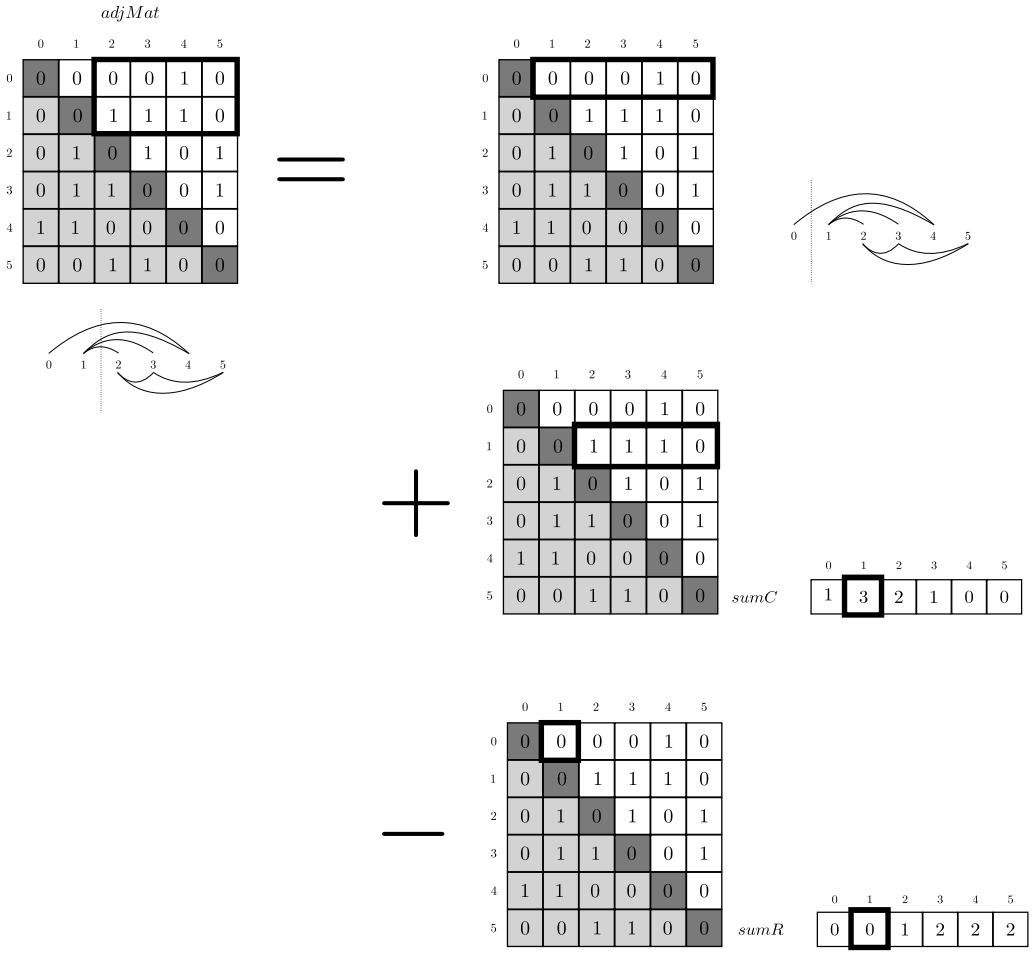


Figura 4.1: Cálculo del cutwidth de un nodo en un arreglo lineal π

4.1.2. Funciónes de Cálculo Inicial de la Suma de las Filas y Columnas del Triángulo Superior de la Matriz de Adyacencia

Sin embargo, el principal problema de este algoritmo es que cada permutación evaluada necesita de su propia matriz de adyacencia, que permita obtener las sumas de las filas y columnas de su triángulo superior. Esto lo hace altamente costoso en complejidad tanto espacial como computacional. Por lo que se hace necesario el diseño de una función capaz de calcular estos valores, sin necesidad de modificar la matriz de adyacencia.

Teniendo esto en consideración, fueron diseñados los Algoritmos 5 y 4 que permiten calcular la suma de las columnas y filas del triángulo superior de la matriz de adyacencia, respectivamente.

Algoritmo 4 *SumRows*

Require: $Instance\ ins$
Require: $Solution\ sol$

1: $sol.sumR[sol.permutation.front()] = 0$
2: **for** $i = 1$ **to** $ins.size$ **do**
3: $\quad sol.sumR[sol.permutation[i]] = 0$
4: $\quad$ **for** $j = 0$ **to** i **do**
5: $\quad\quad valAdjMat = ins.adjMat[sol.permutation[j]][sol.permutation[i]]$
6: $\quad\quad sol.sumR[sol.permutation[i]] += valAdjMat$
7: $\quad$ **end for**
8: **end for**

4.1.3. Función de Cálculo de la Función Objetivo

Una vez que se calcularon por primera vez los valores de cutwidth para cada uno de los nodos del arreglo lineal inicial, es necesario determinar los valores de la permutación cuando se realizan movimientos de exploración en el vecindario elegido, en este caso de inserciones consecutivas o swap.

Algoritmo 5 *SumCols*

Require: *Instance ins*
Require: *Solution sol*

```
1: sol.sumC[sol.permutation.back()] = 0
2: for i = 0 to ins.size − 1 do
3:    sol.sumC[sol.permutation[i]] = 0
4:    for j = i + 1 to ins.size do
5:       valAdjMat = ins.adjMat[sol.permutation[i]][sol.permutation[j]]
6:       sol.sumC[sol.permutation[i]]+ = valAdjMat
7:    end for
8: end for
```

La justificación de la elección de este tipo de vecindario para la exploración de nuevas soluciones se debió fundamentalmente a dos razones, el tamaño del vecindario y la reducción en la complejidad de las funciones de cálculo de la función objetivo.

La primera, el tamaño del vecindario. Dentro de las alternativas consideradas, el vecindario por intercambio y el vecindario por inserción, el que cuenta con un mayor espacio de búsqueda es el de inserción, como puede verse en las Figuras 4.2 y 4.3, en los cuales se observa que en el caso de la vecindad de intercambio, los únicos elementos que resultan afectados, son aquellos que participan en el movimiento. Mientras que, en la función de inserción lo son también los nodos intermedios entre el elemento a reposicionar y el lugar de inserción.

Por ello, la vecindad de inserción es capaz de explorar dentro de un espacio de búsqueda mucho mayor. Sin embargo, el principal problema al cual se enfrenta es que solo evalúa en una sola ocasión, al momento de posicionarse en el lugar de inserción. Así que teniendo en consideración, que la mejor opción dentro de las vecindades de intercambio e inserción es esta última, se decidió explorar variantes que permitieran evaluar las soluciones intermedias al realizar la inserción.

La variante elegida por ser análoga a la función de inserción, permitiendo

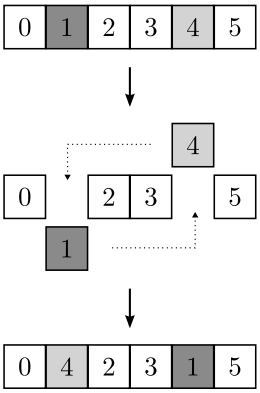


Figura 4.2: Movimiento de Intercambio

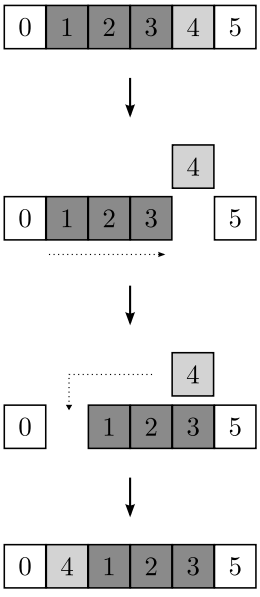


Figura 4.3: Movimiento de Inserción

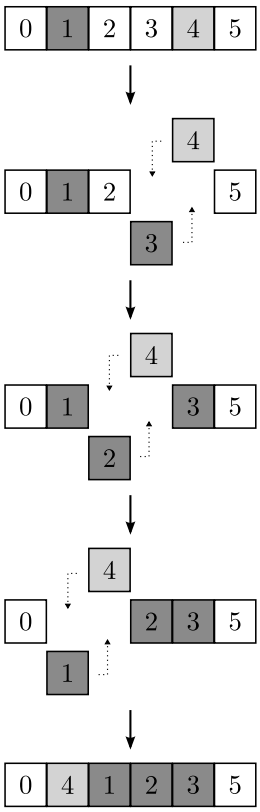


Figura 4.4: Movimiento de Inserción Consecutiva

ademas evaluar cada uno de los movimientos intermedios antes de llegar al punto de inserción, fue la función de inserciones consecutivas o swap. Como puede observarse en la Figura 4.4, a través de movimientos de intercambios consecutivos con su vecino inmediato, la función swap, es capaz de formar el mismo arreglo lineal que la función de inserción, con la ventaja adicional de explorar una mayor cantidad de posibles soluciones candidatas.

La segunda razón, es debido a la facilidad con la que es posible actualizar y determinar los nuevos valores del cutwidth de cada uno de los nodos y de actualizar los valores de las sumas de filas y columnas del triángulo superior de la matriz del grafo si se realiza a través de movimientos consecutivos con su vecino inmediato. Por lo que es posible reducir aún más la complejidad de

las funciones que sirvieron como base para las funciones de búsqueda local implementadas.

Dado que en calculo inicial del valor objetivo de CWP se logró obtener información acerca de la permutación, esta podrá ser aprovechada para reducir la complejidad computacional al determinar los nuevos valores de cutwidth de los nodos cuando sean reacomodados dentro del arreglo lineal, por lo que la evaluación del vaor del cutwidth de un nodo en el arreglo lineal, se reduce a la función mostrada en el Algoritmo 6.

Algoritmo 6 *EvaluationCW*

Require: *Instance ins*
Require: *Solution sol*
Require: *int pos*

1: $updateRows(ins, sol, pos)$
2: $updateCols(ins, sol, pos)$
3: $next = sol.permutation[pos + 1]$
4: $current = sol.permutation[pos]$
5: $previous = sol.permutation[pos - 1]$
6: $sol.cutwidth[next] = sol.cutwidth[current]$
7: $sol.cutwidth[current] = sol.cutwidth[previous] - sol.sumR[current] + sol.sumC[current]$

4.1.4. Funciónes de Actualización de la Suma de las Filas y Columnas del Triángulo Superior de la Matriz de Adyacencia

El costo computacional sería constante, si no fuese por las funciones *updateSumRows* y *updateSumCols*, las cuales, a pesar de tener una complejidad polinomial, adecuada para el calculo inicial de la permutación, no lo es tanto para la evaluación constante en la búsqueda de nuevas regiones por medio de movimientos de inserción.

Es por ello que fue necesario su rediseño, por medio de la observación de como eran alterados los valores de las sumas cuando se hacia un movimiento de inserción consecutiva.

En la Figura 4.5 se muestra un ejemplo de este análisis.

Una vez más se realiza la inserción del nodo el la posición 4 del arreglo lineal, a la posición 1 del mismo, pero esta vez observando como es que las estructuras que contienen las sumas de filas y columnas del triángulo superior de la matriz de adyacencia son alteradas por los movimientos de inserción consecutiva. Como se observa en la figura, los únicos elementos en los cuales ocurren modificaciones en las estructuras de suma, son aquellos directamente involucrados en la inserción, específicamente en el elemento adyacente a la diagonal principal del elemento que se esta reposicionando. Como se observa, cuando se mueve hacia la izquierda, el elemento adyacente se pierde en el caso del arreglo $sumR$, y su elemento opuesto, adyacente a la diagonal principal en el triángulo inferior, resaltado en color gris, se agrega al arreglo $sumC$ del elemento que se está insertando. También se observa que en el caso del elemento con el cual se esta realizando el intercambio, ocurre exactamente lo opuesto. Al moverse a la izquierda, gana el elemento perdido por el elemento a mover en $sumaR$ y pierde el que es sumado al nodo a reposicionar en el arreglo $sumC$.

Gracias a estas observaciones fue posible el diseño de los Algoritmos 7 y 8 los cuales tienen una complejidad computacional constante, debido a que en cualquier caso se realizan sólo las operaciones descritas en los algoritmos, por tanto, hacen que la complejidad de calcular el valor del *cutwidth* de cada nodo al evaluar un movimiento de inserción consecutiva sea asimismo constante.

4.1.5. Costo de la Inserción

Debido a que los movimientos de exploración no son todos iguales, es necesario contar con una función que cuantifique la calidad de un movimiento

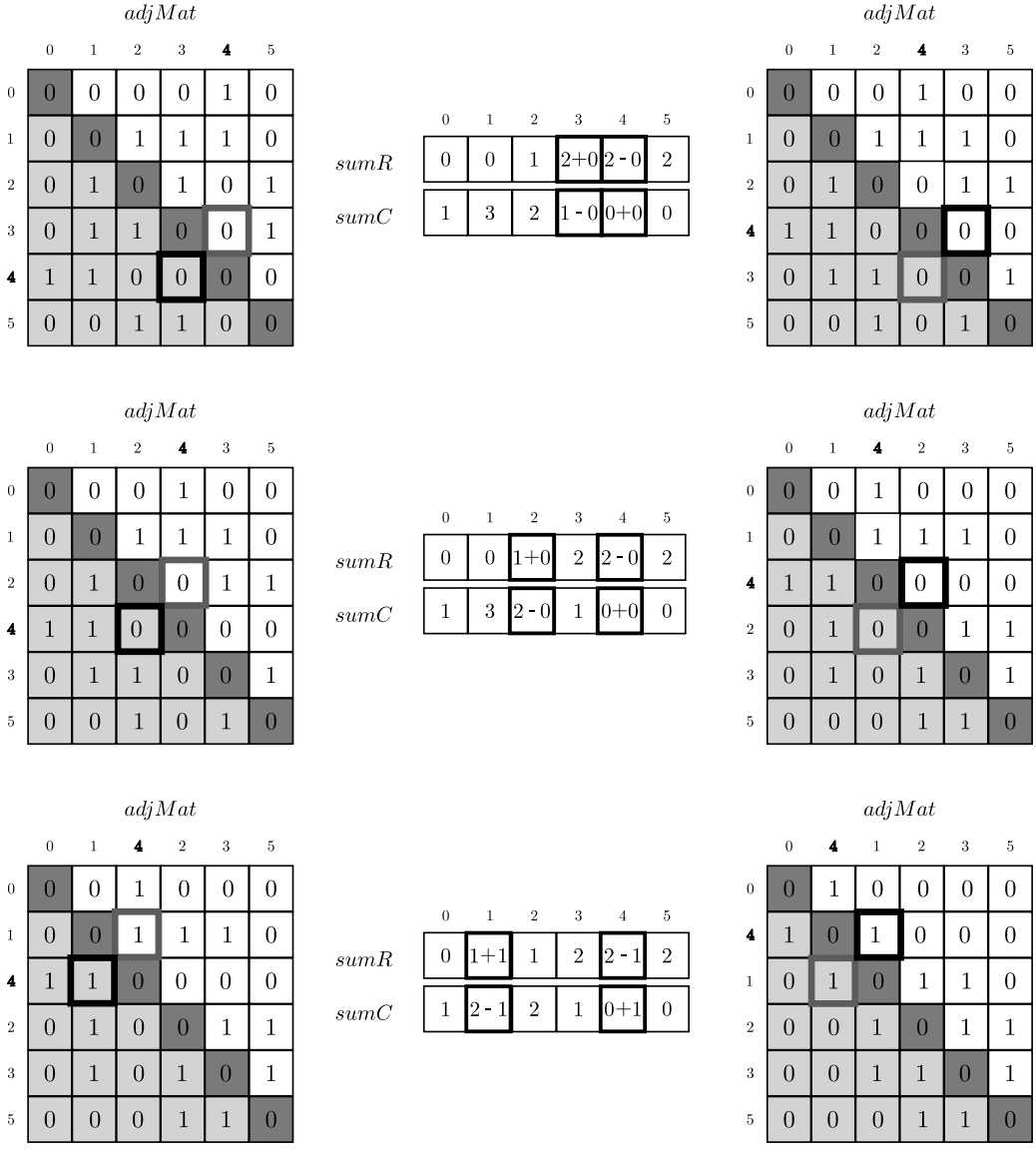


Figura 4.5: Calculo inicial del cutwidth del nodo en la posición 0 de la permutación del grafo de la Figura 3.1

Algoritmo 7 *UpdateSumR*

Require: *Instance ins*
Require: *Solution sol*
Require: *int i*

1: $currentElement = sol.permutation[i]$
2: $nextElement = sol.permutation[i+1]$
3: $sol.sumR[currentElement] -= ins.adjMat[nextElement][currentElement]$
4: $sol.sumR[nextElement] += ins.adjMat[currentElement][nextElement]$

Algoritmo 8 *UpdateSumC*

Require: *Instance ins*
Require: *Solution sol*
Require: *int i*

1: $currentElement = sol.permutation[i]$
2: $nextElement = sol.permutation[i+1]$
3: $sol.sumC[nextElement] -= ins.adjMat[nextElement][currentElement]$
4: $sol.sumC[currentElement] += ins.adjMat[currentElement][nextElement]$

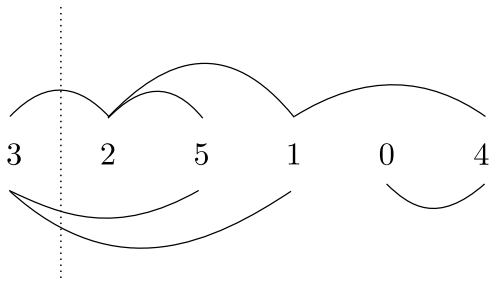
dado.

La calidad en este caso, esta dada por el cambio en $CW_\pi(G)$ y medida por la Ecuacion 4.1, la cual evalúa el costo de la inserción al realizar el movimiento $insert(\pi, j, i)$ en el arreglo lineal π.

$$ganancia(mov(\pi, j, i)) = CW_\pi(G) - CW_{\pi'}(G) \tag{4.1}$$

Ejemplificando la Ecuación 4.1 se obtendrá la ganancia de la inserción al realizar el movimiento $insert(\pi, 5, 2)$. Y de acuerdo a la Figura 4.8 se puede inferir que el CW del arreglo lineal π era de 5 antes de efectuar inserción y una vez efectuada, el CW de π' fue de 3.

$$ganancia(insert(\pi, 5, 2)) = 5 - 3$$



$adjMat$

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

Figura 4.6: Calculo inicial del cutwidth del nodo en la posición 0 de la permutación del grafo de la Figura 3.1

$$\begin{aligned} ganancia(insert(\pi, 5, 2)) &= 5 - 3 \\ &= 2 \end{aligned}$$

Por lo que al ejecutar $insert(\pi, 5, 2)$ se obtienen una ganancia de inserción de 2, que indica que el movimiento en este caso es favorable para el objetivo que estamos persiguiendo, que es el de minimizar el valor del cutwidth del grafo.

4.2. Función de Movimiento

Como se mencionó anteriormente, una de las principales fuentes de optimización de una función de búsqueda local se encuentra en la función que controla que elementos y en que posición reacomoda los elementos, esperando con esto encontrar una solución mejor a la actual, o que lleve a un cambio en el espacio de búsqueda actual.

Esto es, debido a que la función de movimiento es el núcleo de la búsqueda local, pues es la encargada de explorar todo el espacio de soluciones, intentando mejorar la solución candidata actual.

Por ello, la selección de los elementos a ser reposicionados y el criterio

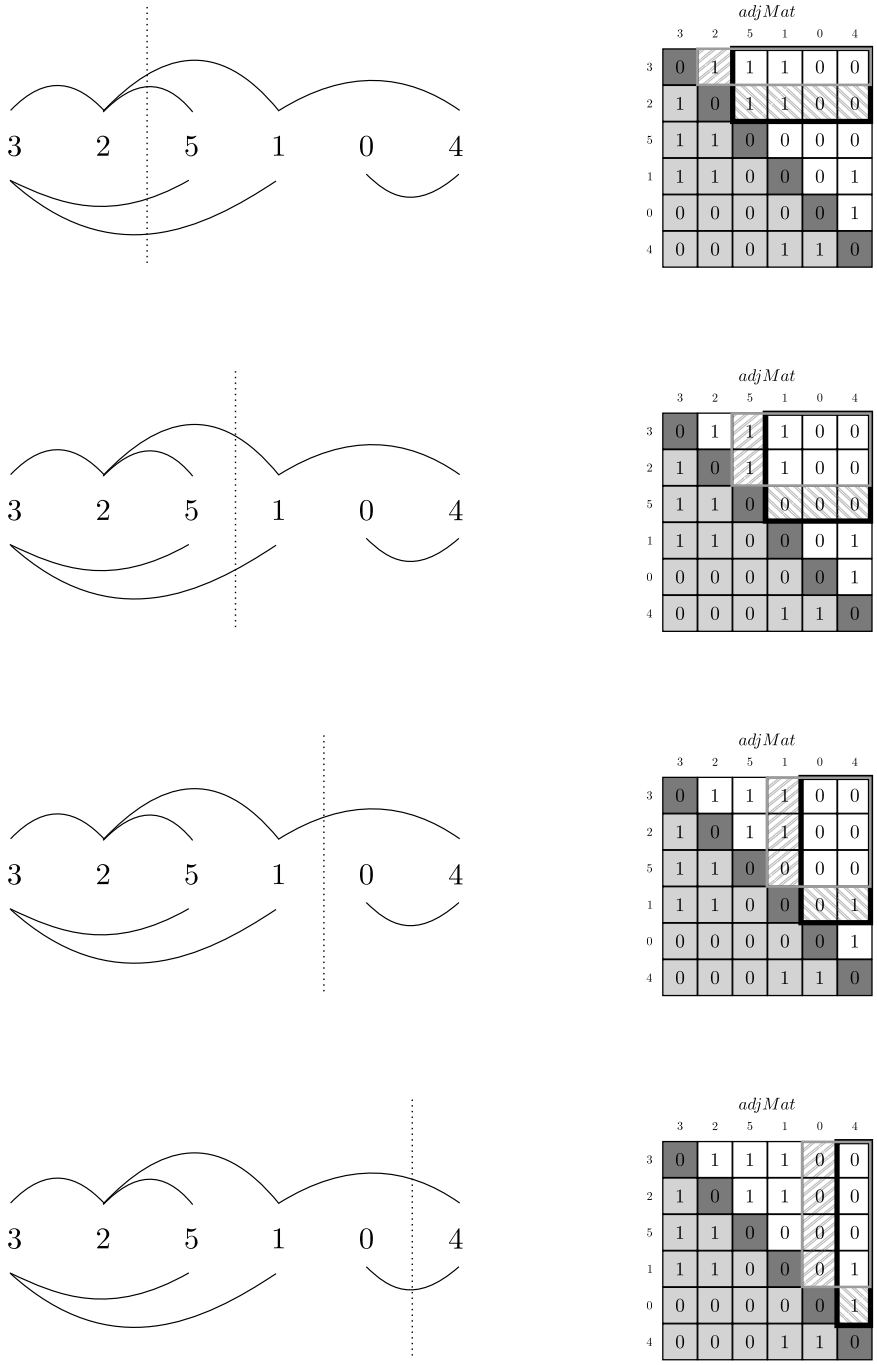


Figura 4.7: Calculo inicial del cutwidth de los nodos en las posiciones k de la permutación del grafo de la Figura 3.1

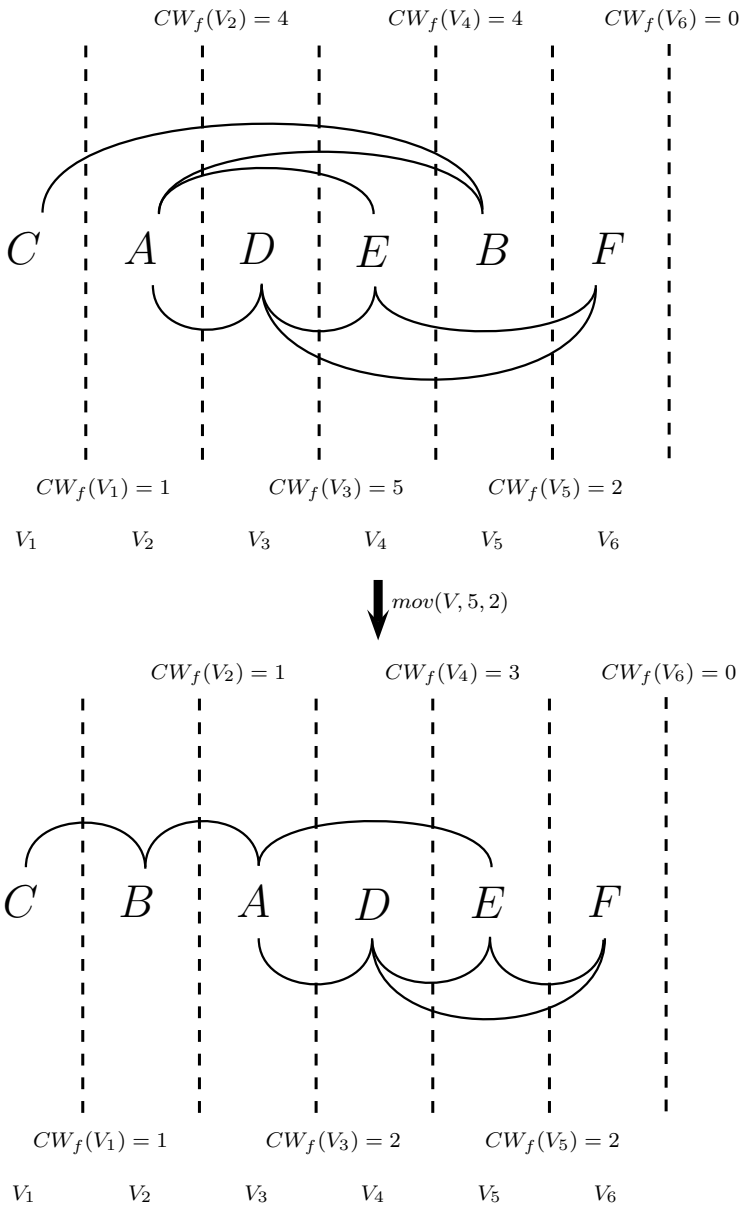


Figura 4.8: Ganancia del movimiento de inserción.

por el cual son reacomodados en una posición distinta, cobran una mayor importancia en este proyecto de investigación

4.2.1. Funciones de Selección de Elementos

La función de selección de elementos, como su propio nombre indica selecciona un subconjunto de elementos de la solución candidata los cuales explorarán dentro del vecindario buscando la posición en la cual logren mejorar el valor objetivo actual. A continuación se describirán algunas de las estrategias que fueron analizadas.

El Algoritmo 9 describe el funcionamiento de la estrategia de selección de elementos E1, la obtiene una lista de elementos críticos para realizar la exploración de la búsqueda local, basados en el valor del *cutwidth* de cada uno de los elementos de la permutación.

Esto se establece por medio del calculo de un limite inferior, en el que si el *cutwidth* del elemento que se analiza es mayor o igual a este, el elemento actual es anexado a la lista candidata de elementos críticos.

Algoritmo 9 $E1$

Require: *Solucion solucion*

1: $limInferior = solucion.cw * (1 - porcentajeCriticos)$
2: $V = \{v | v \in solucion.permutacion; v \geq limiteInferior\}$
3: **return** V

La estrategia E2, no obtiene el subconjunto de elementos de acuerdo a criterio alguno, simplemente agrupa un porcentaje de los elementos de la solución actual de manera aleatoria.

Finalmente, la estrategia E3, toma los elementos de acuerdo al orden en que se encuentran en la solución candidata actual.

4.2.2. Funciones de Reposicionamiento

La selección de elementos a reposicionar van desde la simple selección aleatoria de un conjunto de posiciones de la permutación, a la selección de elementos críticos, los cuales dependen en gran medida del problema que se desea resolver.

En el caso particular de el problema de ordenamiento lineal, se evaluaron ambas estrategias, y en el caso de los elementos críticos, se evaluaron diversas estrategias propias a las características del problema entre las cuales se encuentran las siguientes.

Dentro de las estrategias de reposicionamiento de los elementos para encontrar su mejor posición se encuentran las siguientes:

La estrategia de búsqueda BP1 (Algoritmo 10) explora el vecindario dado el elemento actual en busca de la posición de inserción que resulte en una mejora real con referencia a la solución actual y en caso de que no se encuentre alguna, el elemento es colocado en la mejor posición detectada durante la exploración, incluso si este movimiento resulta en una degradación de la calidad de la solución candidata actual. El orden de búsqueda es asimismo, sistemático de izquierda a derecha.

La estrategia de búsqueda BP2 (Algoritmo 11)intenta reubicar cada uno de los elementos de la permutación actual en la posición en la cual se mejore el valor objetivo de la solución, deteniéndose en el momento en que ya no se produce mejora alguna. El orden de búsqueda, al igual que en los casos anteriores, se realiza de manera sistemática, de izquierda a derecha.

4.3. Función de Inicializacion

Otro criterio importante para la optimización de una búsqueda local, corresponde a la implementación de funciones eficientes que produzcan soluciones iniciales de calidad. Esto es, debido a que partiendo de una región en el espacio de búsqueda que ofrezca soluciones mejores en general a las

Algoritmo 10 $BP1$

Require: $Instancia\ instancia$
Require: $Solucion\ solucion$
Require: v_j

```
1: mejor_i = −1
2: cutwidthMenor = −1
3: for i = solucion.iPermutacion[v_j] − 1 to 1 do
4:    cutwidthActual = calculoCutwidthVectorIzq()
5:    if cutwidthActual <= cutwidthMenor) then
6:       cutwidthMenor = cutwidthActual
7:       mejor_i = i
8:    end if
9: end for
10: for i = solucion.iPermutacion[v_j] − 1 to tamanioPermutacion − 1
    do
11:    cutwidthActual = calculoCutwidthVectorDer()
12:    if cutwidthActual <= cutwidthMenor) then
13:       cutwidthMenor = cutwidthActual
14:       mejor_i = i
15:    end if
16: end for
17: return mejor_i
```

que se podría obtener a partir de una solución generada de manera aleatoria es posible disminuir el tiempo que la función de búsqueda local emplea para mejorar la solución candidata. Sin embargo, la construcción de soluciones iniciales también conlleva el riesgo de quedarse estancados en óptimos locales, por lo que el desarrollar algunos mecanismos de diversificación es altamente recomendable.

El método constructivo empleado para generar una solución inicial, se puede observar en el Algoritmo 14. Este método es el empleado tanto por las metaheurísticas de [Andrade and Resende, 2007a] y [Pantrigo et al., 2010].

Este algoritmo denominado a partir de ahora C1 consiste en establecer como primer elemento de la permutación, el vector que se encuentre menos

Algoritmo 11 *BP2*

Require: *Instancia instancia*
Require: *Solucion solucion*
Require: *v_j*

```
1: mejor_i = −1
2: cutwidthMenor = cutwidthActual
3: for i = solucion.iPermutacion[v_j] − 1 to 1 do
4:    cutwidthActual = calculoCutwidthVectorIzq()
5:    if cutwidthActual < cutwidthMenor) then
6:       cutwidthMenor = cutwidthActual
7:       mejor_i = i
8:       return mejor_i
9:    end if
10: end for
11: for i = solucion.iPermutacion[v_j] − 1 to tamanioPermutacion − 1
    do
12:    cutwidthActual = calculoCutwidthVectorDer()
13:    if cutwidthActual < cutwidthMenor) then
14:       cutwidthMenor = cutwidthActual
15:       mejor_i = i
16:       return mejor_i
17:    end if
18: end for
19: return mejor_i
```

conectado, y partiendo de este inicia un ciclo en el cual se agregan los demás vectores a la solución.

Esto último se realiza generando una lista de vectores candidatos, en la cual se insertan aquellos que son adyacentes a los vectores que ya forman parte de la solución parcialmente construida.

De esta lista de candidatos, se determina el *cutwidth* de cada uno de ellos en la posición actual y se determinan el mínimo y máximo *cutwidth* obtenidos.

Posteriormente, se forma la lista de mejores candidatos, la cual explora la

lista candidata construida en el paso anterior agregada a la lista de mejores candidatos el vector evaluado si se satisface la condición que se encuentra en la linea 22 del Algoritmo 14.

En ella, *alpha* es un número aleatorio real generado entre 0 y 1, que determina el umbral de aceptación de vectores de la solución actual.

Finalmente, es elegido al azar uno de los elementos de la lista de mejores candidatos y agregada a la permutación parcialmente construida.

Algoritmo 12 C

Require: $Instancia\ instancia$
Require: $Solucion\ solucion$

1: $Solucion\ solucion$
2: V
3: C
4: MC
5: $solucion.permutacion = \emptyset$
6: $v = \arg\min_{v \in V}\{vecinos(v)\}$
7: $k = 1$
8: $solucion.permutacion[k] = v$
9: $V \setminus \{v\}$
10: **repeat**
11: $\quad k++$
12: $\quad C = \{v \in V | (w,u) \in E; w \in solucion.permutacion\}$
13: $\quad cwMin = \arg\min_{v \in C}\{cw(v)\}$
14: $\quad cwMax = \arg\max_{v \in C}\{cw(v)\}$
15: $\quad MC = \{v \in C | cw(v) \leq cwMin + \alpha(cwMax - cwMin)\}$
16: $\quad$ Seleccionar v aleatoriamente de MC
17: $\quad solucion.permutacion[k] = v$
18: $\quad V \setminus \{v\}$
19: **until** $V \neq \emptyset$
20: **return** $solucion$

Un segundo criterio de inicialización, C2, consistió en la generación de una solución candidata de forma aleatoria, esperando de esta manera diversificar la búsqueda dentro del espacio de soluciones, esperando con ello explorar

regiones que por la naturaleza misma del método constructivo anterior no pudieran ser alcanzadas por las funciones de reposicionamiento propuestas.

4.4. Experimentación

Las instancias empleadas como referencia fueron las *Grids*, de dificultad media, con un rango de 9 a 729 nodos, de las cuales su valor óptimo es conocido y las $Harwell - Boeing$, de dificultad alta, de las cuales solamente existen mejores valores conocidos.

Los dos grupos de instancias son codificadas de dos formas la primera, mediante un diccionario de claves en el que sólo se almacenan los valores distintos de cero, utilizada por $Harwell - Boeing$, y la segunda, mediante su matriz de adyacencia, en la que se encuentran codificadas las instancias del conjunto *Grids*.

Estos conjuntos de instancias pueden ser encontrados en el sitio web del Proyecto Optsicom [Martí R. and E.G., 2010].

Las diferentes búsquedas locales implementadas fueron desarrolladas en el lenguaje de programación C++, partiendo del uso de su conjunto de librerías estándar, y compilada en el IDE XCode en su versión 4.2 bajo el sistema operativo Mac OS X 10.7.2.

Las entradas que el programa necesita para operar consisten en la ruta del archivo de instancia a resolver, el valor óptimo de la instancia, o en su defecto, el mejor valor conocido, la codificación de la instancia y el nombre del archivo en el cual se guardarán los resultados.

La experimentación fue efectuada en un portátil Apple MacBook Pro, con un procesador Intel i5 a 2.3 GHz, 4 GB de RAM y sistema operativo Mac OS X 10.7.2.

De acuerdo a los componentes identificados en el Algoritmo 2, que integran a cualquier mecanismo de búsqueda local y que fueron descritos en las secciones anteriores de este capítulo, se evaluaron distintas configuraciones

dadas por las diferentes configuraciones de los componentes implementados los cuales constituyen las búsquedas locales mostradas en la Tabla 4.1.

Tabla 4.1: Métodos de Búsqueda Local Evaluados

Configuración	Inicialización	Reacomodo	Selección
LS1	C1	BP1	E1
LS2	C1	BP2	E3
LS3	C1	BP2	E2
LS4	C2	BP1	E1
LS5	C2	BP2	E2
LS6	C2	BP2	E3

Para evaluar cada una de las configuraciones, se implemento la metaheurística, Búsqueda Local Iterada, ILS, mostrada en el Algoritmo 15, el cual fue ajustado para detenerse tras diez iteraciones consecutivas en las que el método de búsqueda local no mejorara la solución actual y el porcentaje de perturbación fue establecido con un valor del veinte por ciento del tamaño de la solución.

La batería completa de experimentos, con los tres conjuntos de instancias fue realizada con un total de quince repeticiones por cada búsqueda local propuesta, para obtener una muestra que produzca datos estadísticamente significativos.

A cada una de las búsquedas locales propuestas se les fue aplicado el Test de Friedman, el cual es un test estadístico no paramétrico desarrollado por el economista Milton Friedman, similar al test ANOVA.

En el caso de que solo se evalúen dos muestras, el Test de Friedman es análogo a la Prueba de Wilcoxon, pues hace las mismas presuposiciones acerca de las características que tienen los datos a analizar, aunque en general es ligeramente menos sensible que este último [Dalgaard, 2008].

Las hipótesis evaluadas por el Test de Friedman son las que se describen

a continuación:

H_0 No existen diferencias significativas entre los conjuntos de datos analizados.

H_a Al menos dos de los conjuntos de datos analizados son distintos entre sí.

Por ello, si el p-value calculado por el Test de Friedman es menor que el nivel de significancia α establecido, la hipótesis nula H_0 se rechaza y se establece que las diferencias entre las muestras son significativas, como se establece en [Pavan and Todeschini, 2008].

El test individual en este caso, sirve para observar si el comportamiento de cada función de búsqueda es estable, es decir, si no existen diferencias significativas en los quince conjuntos resultados obtenidos en los experimentos.

Para realizar esta prueba, se hizo uso del software computacional desarrollado por [SCI2S, 2010].

A continuación, se aplicará de igual manera el test de Friedman, pero en este caso, entre cada uno de las funciones de búsqueda local que probaron ser estables en su comportamiento, tomando uno de los quince conjuntos de resultados de cada uno de los métodos evaluados, con la seguridad de que fue demostrado que los resultados obtenidos son estadísticamente consistentes entre si.

En este último caso, se tomó el ranking del test de Friedman para determinar cual es el desempeño de cada uno de ellos con respecto a los demás y también se verifico si existían diferencias significativas entre cada uno de ellos.

4.5. Resultados

En las Tablas 4.2, 4.3 y 4.4, se muestra los resultados del análisis estadístico de Friedman de los resultados obtenidos por la búsqueda local LS1 a lo largo de las quince repeticiones del experimento sobre los conjuntos de

instancias Grids, Harwell-Boeing y Small, de manera respectiva. Si el p-value calculado por el Test de Friedman es menor al nivel de significancia α dado, se rechaza la hipótesis nula de que el algoritmo probado presenta el mismo comportamiento a lo largo de todas las experimentaciones y por lo tanto se determina que su desempeño es inestable, en caso contrario es aceptada y el algoritmo presenta un comportamiento estable para es conjunto de instancias evaluado.

Tabla 4.2: Test de Friedman para LS1 - Grids, $\alpha = 0.05$

Experimento	p-value	H_0 rechazada
LS1-1	0.0005	Sí
LS1-2	0.0015	Sí
LS1-3	0.0021	Sí
LS1-4	0.0040	Sí
LS1-5	0.0041	Sí
LS1-6	0.0059	Sí
LS1-7	0.0068	Sí
LS1-8	0.0265	Sí
LS1-9	0.0965	No
LS1-10	0.1423	No
LS1-11	0.2159	No
LS1-12	0.2541	No
LS1-13	0.3672	No
LS1-14	0.4578	No

En las Tablas 4.5, 4.6 y 4.7, se muestra los resultados del análisis estadístico de Friedman de los resultados obtenidos por la búsqueda local LS2 a lo largo de las quince repeticiones del experimento sobre los conjuntos de instancias Grids, Harwell-Boeing y Small, de manera respectiva. Si el p-value calculado por el Test de Friedman es menor al nivel de significancia α dado, se rechaza la hipótesis nula de que el algoritmo probado presenta el mismo comportamiento a lo largo de todas las experimentaciones y por lo tanto se

Tabla 4.3: Test de Friedman para LS1 - HB, $\alpha = 0.05$

Experimento	p-value	H_0 rechazada
LS1-1	0.0076	Sí
LS1-2	0.0150	Sí
LS1-3	0.0168	Sí
LS1-4	0.0684	No
LS1-5	0.0823	No
LS1-6	0.0884	No
LS1-7	0.1037	No
LS1-8	0.1336	No
LS1-9	0.1358	No
LS1-10	0.1697	No
LS1-11	0.1724	No
LS1-12	0.2191	No
LS1-13	0.3011	No
LS1-14	0.3091	No

Tabla 4.4: Test de Friedman para LS1 - Small, $\alpha = 0.05$

Experimento	p-value	H_0 rechazada
LS1-1	0.0756	No
LS1-2	0.2017	No
LS1-3	0.2408	No
LS1-4	0.3515	No
LS1-5	0.4478	No
LS1-6	0.4794	No
LS1-7	0.5011	No
LS1-8	0.5066	No
LS1-9	0.5691	No
LS1-10	0.5750	No
LS1-11	0.5988	No
LS1-12	0.6169	No
LS1-13	0.6979	No
LS1-14	0.7171	No

determina que su desempeño es inestable, en caso contrario es aceptada y el algoritmo presenta un comportamiento estable para es conjunto de instancias evaluado.

Tabla 4.5: Test de Friedman para LS2 - Grids, $\alpha = 0.05$

Experimento	p-value	H_0 rechazada
LS2-1	0.0133	Sí
LS2-2	0.0201	Sí
LS2-3	0.0221	Sí
LS2-4	0.0297	Sí
LS2-5	0.0634	No
LS2-6	0.0832	No
LS2-7	0.1002	No
LS2-8	0.1156	No
LS2-9	0.1399	No
LS2-10	0.3179	No
LS2-11	0.4419	No
LS2-12	0.5303	No
LS2-13	0.6020	No
LS2-14	0.6269	No

En las Tablas 4.8, 4.9 y 4.10, se muestra los resultados del análisis estadístico de Friedman de los resultados obtenidos por la búsqueda local LS3 a lo largo de las quince repeticiones del experimento sobre los conjuntos de instancias Grids, Harwell-Boeing y Small, de manera respectiva. Si el p-value calculado por el Test de Friedman es menor al nivel de significancia α dado, se rechaza la hipótesis nula de que el algoritmo probado presenta el mismo comportamiento a lo largo de todas las experimentaciones y por lo tanto se determina que su desempeño es inestable, en caso contrario es aceptada y el algoritmo presenta un comportamiento estable para es conjunto de instancias evaluado.

En las Tablas 4.11, 4.12 y 4.13, se muestra los resultados del análisis

Tabla 4.6: Test de Friedman para LS2 - HB, $\alpha = 0.05$

Experimento	p-value	H_0 rechazada
LS2-1	0.1569	No
LS2-2	0.2128	No
LS2-3	0.2354	No
LS2-4	0.2933	No
LS2-5	0.3011	No
LS2-6	0.3920	No
LS2-7	0.5992	No
LS2-8	0.6350	No
LS2-9	0.7346	No
LS2-10	0.7927	No
LS2-11	0.8587	No
LS2-12	0.8988	No
LS2-13	0.9594	No
LS2-14	0.9797	No

Tabla 4.7: Test de Friedman para LS2 - Small, $\alpha = 0.05$

Experimento	p-value	H_0 rechazada
LS2-1	0.1328	No
LS2-2	0.3060	No
LS2-3	0.3760	No
LS2-4	0.4777	No
LS2-5	0.5067	No
LS2-6	0.5489	No
LS2-7	0.5551	No
LS2-8	0.5928	No
LS2-9	0.6381	No
LS2-10	0.6647	No
LS2-11	0.6647	No
LS2-12	0.7821	No
LS2-13	0.7891	No

Tabla 4.8: Test de Friedman para LS3 - Grids, $\alpha = 0.05$

Experimento	p-value	H_0 rechazada
LS3-1	0.0063	Sí
LS3-2	0.0109	Sí
LS3-3	0.0178	Sí
LS3-4	0.0237	Sí
LS3-5	0.0284	Sí
LS3-6	0.0297	Sí
LS3-7	0.0310	Sí
LS3-8	0.0801	No
LS3-9	0.1241	No
LS3-10	0.1399	No
LS3-11	0.2259	No
LS3-12	0.3443	No
LS3-13	0.5074	No
LS3-14	0.6082	No

Tabla 4.9: Test de Friedman para LS3 - HB, $\alpha = 0.05$

Experimento	p-value	H_0 rechazada
LS3-1	0.0140	Sí
LS3-2	0.1001	No
LS3-3	0.1130	No
LS3-4	0.1250	No
LS3-5	0.1751	No
LS3-6	0.1918	No
LS3-7	0.2525	No
LS3-8	0.2894	No
LS3-9	0.3600	No
LS3-10	0.4062	No
LS3-11	0.4158	No
LS3-12	0.5875	No
LS3-13	0.5934	No
LS3-14	0.7667	No

Tabla 4.10: Test de Friedman para LS3 - Small, $\alpha = 0.05$

Experimento	p-value	H_0 rechazada
LS3-1	0.0165	Sí
LS3-2	0.0218	Sí
LS3-3	0.0454	Sí
LS3-4	0.0799	No
LS3-5	0.0877	No
LS3-6	0.1547	No
LS3-7	0.2048	No
LS3-8	0.4902	No
LS3-9	0.5233	No
LS3-10	0.5928	No
LS3-11	0.6108	No
LS3-12	0.6725	No
LS3-13	0.7236	No
LS3-14	0.9587	No

estadístico de Friedman de los resultados obtenidos por la búsqueda local LS4 a lo largo de las quince repeticiones del experimento sobre los conjuntos de instancias Grids, Harwell-Boeing y Small, de manera respectiva. Si el p-value calculado por el Test de Friedman es menor al nivel de significancia α dado, se rechaza la hipótesis nula de que el algoritmo probado presenta el mismo comportamiento a lo largo de todas las experimentaciones y por lo tanto se determina que su desempeño es inestable, en caso contrario es aceptada y el algoritmo presenta un comportamiento estable para es conjunto de instancias evaluado.

En las Tablas 4.14, 4.15 y 4.16, se muestra los resultados del análisis estadístico de Friedman de los resultados obtenidos por la búsqueda local LS5 a lo largo de las quince repeticiones del experimento sobre los conjuntos de instancias Grids, Harwell-Boeing y Small, de manera respectiva. Si el p-value calculado por el Test de Friedman es menor al nivel de significancia α dado,

Tabla 4.11: Test de Friedman para LS4 - Grids, $\alpha = 0.05$

Experimento	p-value	H_0 rechazada
LS4-1	0.0101	Sí
LS4-2	0.0118	Sí
LS4-3	0.0162	Sí
LS4-4	0.0178	Sí
LS4-5	0.0201	Sí
LS4-6	0.0211	Sí
LS4-7	0.0332	Sí
LS4-8	0.0562	No
LS4-9	0.0610	No
LS4-10	0.1020	No
LS4-11	0.1198	No
LS4-12	0.1938	No
LS4-13	0.2159	No
LS4-14	0.2398	No

Tabla 4.12: Test de Friedman para LS4 - HB, $\alpha = 0.05$

Experimento	p-value	H_0 rechazada
LS4-1	0.0019	Sí
LS4-2	0.0100	Sí
LS4-3	0.0231	Sí
LS4-4	0.0252	Sí
LS4-5	0.0473	Sí
LS4-6	0.0933	No
LS4-7	0.2490	No
LS4-8	0.2561	No
LS4-9	0.3339	No
LS4-10	0.4256	No
LS4-11	0.4924	No
LS4-12	0.5031	No
LS4-13	0.5992	No
LS4-14	0.7603	No

Tabla 4.13: Test de Friedman para LS4 - Small, $\alpha = 0.05$

Experimento	p-value	H_0 rechazada
LS4-1	0.0785	No
LS4-2	0.1125	No
LS4-3	0.1144	No
LS4-4	0.1425	No
LS4-5	0.1869	No
LS4-6	0.2142	No
LS4-7	0.2206	No
LS4-8	0.2339	No
LS4-9	0.2771	No
LS4-10	0.3979	No
LS4-11	0.4634	No
LS4-12	0.5691	No
LS4-13	0.6725	No
LS4-14	0.7107	No

se rechaza la hipótesis nula de que el algoritmo probado presenta el mismo comportamiento a lo largo de todas las experimentaciones y por lo tanto se determina que su desempeño es inestable, en caso contrario es aceptada y el algoritmo presenta un comportamiento estable para es conjunto de instancias evaluado.

En las Tablas 4.17, 4.18 y 4.19, se muestra los resultados del análisis estadístico de Friedman de los resultados obtenidos por la búsqueda local LS6 a lo largo de las quince repeticiones del experimento sobre los conjuntos de instancias Grids, Harwell-Boeing y Small, de manera respectiva. Si el p-value calculado por el Test de Friedman es menor al nivel de significancia α dado, se rechaza la hipótesis nula de que el algoritmo probado presenta el mismo comportamiento a lo largo de todas las experimentaciones y por lo tanto se determina que su desempeño es inestable, en caso contrario es aceptada y el algoritmo presenta un comportamiento estable para es conjunto de instancias

Tabla 4.14: Test de Friedman para LS5 - Grids, $\alpha = 0.05$

Experimento	p-value	H_0 rechazada
LS5-1	0.0063	Sí
LS5-2	0.0109	Sí
LS5-3	0.0178	Sí
LS5-4	0.0237	Sí
LS5-5	0.0284	Sí
LS5-6	0.0297	Sí
LS5-7	0.0310	Sí
LS5-8	0.0801	No
LS5-9	0.1241	No
LS5-10	0.1399	No
LS5-11	0.2259	No
LS5-12	0.3443	No
LS5-13	0.5074	No
LS5-14	0.6082	No

Tabla 4.15: Test de Friedman para LS5 - HB, $\alpha = 0.05$

Experimento	p-value	H_0 rechazada
LS5-1	0.0140	Sí
LS5-2	0.1001	No
LS5-3	0.1130	No
LS5-4	0.1250	No
LS5-5	0.1751	No
LS5-6	0.1918	No
LS5-7	0.2525	No
LS5-8	0.2894	No
LS5-9	0.3600	No
LS5-10	0.4062	No
LS5-11	0.4158	No
LS5-12	0.5875	No
LS5-13	0.5934	No
LS5-14	0.7667	No

Tabla 4.16: Test de Friedman para LS5 - Small, $\alpha = 0.05$

Experimento	p-value	H_0 rechazada
LS5-1	0.0165	Sí
LS5-2	0.0218	Sí
LS5-3	0.0454	Sí
LS5-4	0.0799	No
LS5-5	0.0877	No
LS5-6	0.1547	No
LS5-7	0.2048	No
LS5-8	0.4902	No
LS5-9	0.5233	No
LS5-10	0.5928	No
LS5-11	0.6108	No
LS5-12	0.6725	No
LS5-13	0.7236	No
LS5-14	0.9587	No

evaluado.

4.6. Conclusiones

Dados los resultados obtenidos al realizar el Test de Friedman sobre cada algoritmo de búsqueda local propuesto, puede concluirse que no es suficiente depender únicamente de estos mecanismos, ya que no garantizan que la calidad de las soluciones sea consistente en cada ejecución de la función de búsqueda local.

Esto se debe a la naturaleza aleatoria presente en todos los métodos, agravándose aún más en el caso de que la solución inicial no sea generada por un método constructivo que garantice un buen punto de partida sobre el cual la función de búsqueda local pueda trabajar.

Tabla 4.17: Test de Friedman para LS6 - Grids, $\alpha = 0.05$

Experimento	p-value	H_0 rechazada
LS6-1	0.0027	Sí
LS6-2	0.0248	Sí
LS6-3	0.0700	No
LS6-4	0.0700	No
LS6-5	0.0832	No
LS6-6	0.1329	No
LS6-7	0.1472	No
LS6-8	0.1791	No
LS6-9	0.1820	No
LS6-10	0.1908	No
LS6-11	0.1969	No
LS6-12	0.2192	No
LS6-13	0.3961	No
LS6-14	0.4011	No

Tabla 4.18: Test de Friedman para LS6 - HB, $\alpha = 0.05$

Experimento	p-value	H_0 rechazada
LS6-1	0.0599	No
LS6-2	0.0869	No
LS6-3	0.0884	No
LS6-4	0.0967	No
LS6-5	0.1019	No
LS6-6	0.1403	No
LS6-7	0.1545	No
LS6-8	0.2255	No
LS6-9	0.2288	No
LS6-10	0.4609	No
LS6-11	0.6350	No
LS6-12	0.7219	No
LS6-13	0.7797	No
LS6-14	0.7927	No

Tabla 4.19: Test de Friedman para LS6 - Small, $\alpha = 0.05$

Experimento	p-value	H_0 rechazada
LS6-1	0.1497	No
LS6-2	0.2658	No
LS6-3	0.2733	No
LS6-4	0.2926	No
LS6-5	0.3515	No
LS6-6	0.3979	No
LS6-7	0.4325	No
LS6-8	0.5633	No
LS6-9	0.5988	No
LS6-10	0.5988	No
LS6-11	0.6538	No
LS6-12	0.7107	No
LS6-13	0.7693	No
LS6-14	0.8293	No

CAPÍTULO 5

PROCEDIMIENTO DE BÚSQUEDA ADAPTATIVA ALEATORIA VORAZ

En este capítulo se describirá el Procedimiento de Búsqueda Adaptativa Aleatoria Voraz que fue implementado para resolver el Problema del Ancho de Corte, el cual hace uso de las técnicas para el diseño de búsquedas locales eficientes descritas en el Capítulo 4.

5.1. Introducción

El Procedimiento de Búsqueda Adaptativa Aleatoria Voraz GRASP, por su nombre en inglés es una metaheurística multiarranque, en el que cada iteración consiste en la generación de una solución candidata de alta calidad, creada por medio de un método constructivo, la cual inmediatamente es mejorada por medio de un algoritmo de búsqueda local.

De acuerdo con [Duarte Muñoz et al., 2007] GRASP fue desarrollado originalmente por T. Feo y M. Resende como un mecanismo orientado a la re-

solución del recubrimiento de conjuntos, y posteriormente fue formalmente definida como una metaheurística de propósito general. GRASP fue inspirado originalmente a partir de la Heurística Semiconstructiva, de la cual difiere fundamentalmente en el hecho de que esta no incluye una fase de mejora de la solución construida.

En el Algoritmo 13 se observa la estructura básica del algoritmo GRASP, en el cual se aprecian claramente las funciones principales que lo componen, de las cuales cabe la pena resaltar la fase de construcción de la solución y la de mejora de la solución candidata construida.

Algoritmo 13 *GRASP*

1: x
2: **for** $i = 1$ **to** $numArranques$ **do**
3: $\quad x_c = construirSolucion()$
4: $\quad x_c = mejorarSolucion(x_c)$
5: $\quad x = actualizarSolucion(x, x_c)$
6: **end for**
7: **return** x

El método constructivo es el encargado de crear la solución candidata elemento a elemento, partiendo de una semilla, la cual puede ser seleccionada aleatoriamente o por medio de algún criterio que permita seleccionar aquel que resulte más prometedor en la construcción de una solución de alta calidad. Y una vez agregado este primer elemento a la solución, se elimina de la lista candidata de la cual nuevos elementos son seleccionados en la construcción de la solución candidata.

Posteriormente a partir de los elementos en la solución parcialmente construida, se ponderan los elementos en la lista de elementos candidatos de acuerdo a que tan buenos resultarían si fuesen insertados en la siguiente posición de la solución, almacenando los valores de aquellos que constituyan los valores máximo y mínimo entre la lista ponderada de elementos candidatos.

El siguiente paso en el método constructivo consiste en la construcción de una lista candidata reducida, la cual contiene aquellos elementos de la

lista candidata cuyo valor no supere, en el caso de que sea un problema de minimización, el umbral dado por la formula,

$$c_{min} + \alpha \times (c_{max} - c_{min})$$

En donde α que es un número real entre 0 y 1, controla el tamaño de la lista candidata, pues si a α se le asignara el valor de 0, la lista candidata reducida comprendería a la totalidad de los elementos de la lista candidata, y si el valor fuese de 1, la lista candidata reducida comprendería solamete los elementos con el menor valor encontrado al ponderar la lista candidata.

Una vez creada la lista candidata reducida, se selecciona de manera aleatoria un elemento de esta, el cual desde ahora forma parte de la solución parcialmente construida y se elimina de la lista de elementos candidatos.

Este proceso continúa mientras que la solución que se construye con este método no esté completamente construida, momento en el cual el método constructivo da paso al mecanismo de búsqueda local que mejorará la solución candidata y posteriormente a la actualización de la mejor solución global encontrada a lo largo de los diferentes arranques.

El proceso completo del GRASP concluye una vez que el criterio de parada se cumple devolviendo la mejor solución encontrada a lo largo de la ejecución de la metaheurística.

5.2. Implementación

La implementación realizada del Procedimiento de Búsqueda Adaptativa Aleatoria Voraz (GRASP), corresponde a aquella mostrada en el Algoritmo 13 estableciendo el parámetro de iteraciones globales M en cinco arranques.

El mecanismo de mejora del algoritmo $GRASP$, corresponde a aquel descrito en el Capítulo 4.

El método constructivo implementado, se describirá en detalle en la sección siguiente, así como los parámetros con los que fue configurado.

Método Constructivo

El método constructivo empleado se puede observar en el Algoritmo 14 y de manera gráfica en la Figura 5.1. Este método es el empleado tanto por las metaheurísticas de [Andrade and Resende, 2007a] y [Pantrigo et al., 2010].

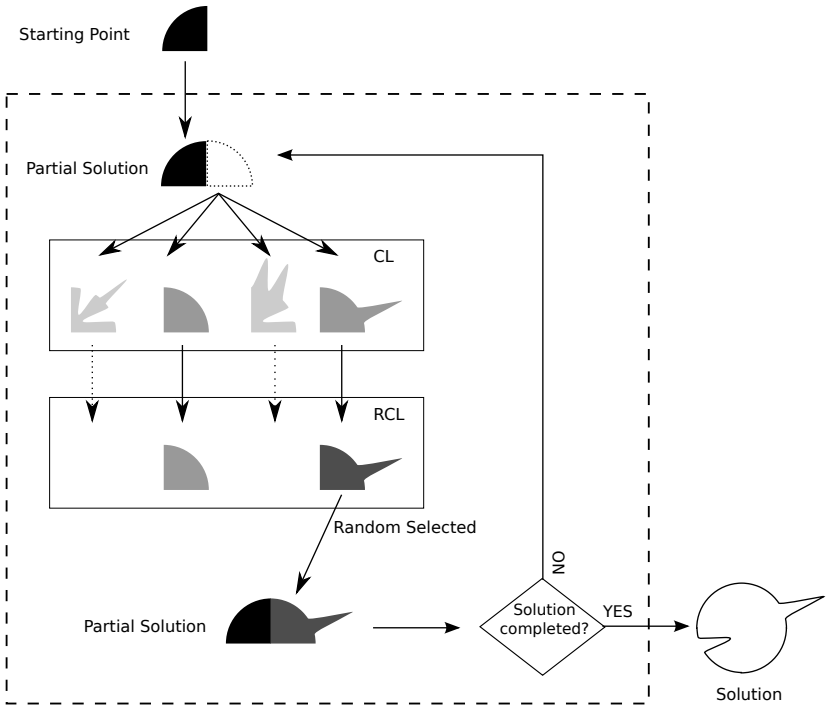


Figura 5.1: Diagrama de Método Constructivo

El algoritmo constructivo consiste en establecer como primer elemento de la permutación, el vector que se encuentre menos conectado, y partiendo de este inicia un ciclo en el cual se agregan los demás vectores a la solución.

Esto último se realiza generando una lista de vectores candidatos, en la cual se insertan aquellos que son adyacentes a los vectores que ya forman parte de la solución parcialmente construida.

De esta lista de candidatos, se determina el *cutwidth* de cada uno de ellos en la posición actual y se determinan el mínimo y máximo *cutwidth*

obtenidos.

Posteriormente, se forma la lista de mejores candidatos, la cual explora la lista candidata construida en el paso anterior agregada a la lista de mejores candidatos el vector evaluado si se satisface la condición que se encuentra en la linea 15 del Algoritmo 14.

En ella, *alpha* es un número aleatorio real generado entre 0 y 1, que determina el umbral de aceptación de vectores de la solución actual.

Finalmente, es elegido al azar uno de los elementos de la lista de mejores candidatos y agregada a la permutación parcialmente construida.

Algoritmo 14 C

Require: *Instancia instancia*
Require: *Solucion solucion*

1: *Solucion solucion*
2: V
3: C
4: MC
5: $solucion.permutacion = \emptyset$
6: $v = \arg\min_{v \in V}\{vecinos(v)\}$
7: $k = 1$
8: $solucion.permutacion[k] = v$
9: $V \setminus \{v\}$
10: **repeat**
11: $k++$
12: $C = \{v \in V | (w,u) \in E; w \in solucion.permutacion\}$
13: $cwMin = \arg\min_{v \in C}\{cw(v)\}$
14: $cwMax = \arg\max_{v \in C}\{cw(v)\}$
15: $MC = \{v \in C | cw(v) \leq cwMin + \alpha(cwMax - cwMin)\}$
16: Seleccionar v aleatoriamente de MC
17: $solucion.permutacion[k] = v$
18: $V \setminus \{v\}$
19: **until** $V \neq \emptyset$
20: **return** *solucion*

5.3. Experimentación

Las instancias empleadas como referencia fueron las *Grids*, de dificultad media, con un rango de 9 a 729 nodos, de las cuales su valor óptimo es conocido y las $Harwell - Boeing$, de dificultad alta, de las cuales solamente existen mejores valores conocidos [Martí R. and E.G., 2010].

Los dos grupos de instancias son codificadas de dos formas la primera, mediante un diccionario de claves en el que sólo se almacenan los valores distintos de cero utilizada por $Harwell - Boeing$ y la segunda mediante su matriz de adyacencia, en la que se encuentran codificadas las instancias del conjunto *Grids*.

Estos conjuntos de instancias pueden ser encontrados en el sitio web del Proyecto Optsicom [Martí R. and E.G., 2010].

Las diferentes búsquedas locales implementadas fueron desarrolladas en el lenguaje de programación C++, partiendo del uso de su conjunto de librerías estándar, y compilada en el IDE XCode en su versión 4.2 bajo el sistema operativo Mac OS X 10.7.2.

Las entradas que el programa necesita para operar consisten en la ruta del archivo de instancia a resolver, el valor óptimo de la instancia, o en su defecto, el mejor valor conocido, la codificación de la instancia y el nombre del archivo en el cual se guardarán los resultados.

La experimentación fue efectuada en un portátil Apple MacBook Pro, con un procesador Intel i5 a 2.3 GHz, 4 GB de RAM y sistema operativo Mac OS X 10.7.2.

El número de iteraciones globales establecidos para el algoritmo GRASP fue de diez arranques. Se definieron dos versiones de la metaheurística, uno que incluye la función de búsqueda local $LS2$, descríta en el Capítulo 4 y otro que no incluye ningún mecanismo de búsqueda local, esto para valorar el impacto general que tiene el algoritmo de búsqueda local en la calidad obtenida por el método GRASP.

La batería completa de experimentos con los tres conjuntos de instancias

fue realizada con un total de quince repeticiones por cada configuración de GRASP propuesto, para obtener una muestra que produzca datos estadísticamente significativos.

A la configuraciones propuesta le fue aplicado el Test de Friedman bajo las mismas hipótesis descritas en la experimentación del Capítulo 4. El Test de Friedman es un test estadístico no paramétrico desarrollado por el economista Milton Friedman, similar al test ANOVA. El test individual en este caso, sirve para observar si el comportamiento de cada función de búsqueda es estable, es decir, si no existen diferencias significativas en los quince conjuntos resultados obtenidos en los experimentos.

Para realizar esta prueba, se hizo uso del software computacional desarrollado por [SCI2S, 2010].

A continuación, se aplicó de igual manera el test de Friedman, pero en este caso, entre cada uno de las funciones de búsqueda local que probaron ser estables en su comportamiento, tomando uno de los quince conjuntos de resultados de cada uno de los métodos evaluados, con la seguridad de que fue demostrado que los resultados obtenidos son estadísticamente consistentes entre si.

En este último caso, se tomó el ranking del test de Friedman para determinar cual es el desempeño de cada uno de ellos con respecto a los demás y también se verifico si existían diferencias significativas entre cada uno de ellos.

5.4. Resultados

En el Apéndice C se pueden consultar los resultados obtenidos por el algoritmo GRASP_LS2 por instancia en una de las experimentaciones realizadas mientras que en el Apéndice D se encuentran los valores óptimos y mejores conocidos de las instancias evaluadas.

En las Tablas 5.1, 5.2 y 5.3, se muestra los resultados del análisis esta-

dístico de Friedman de los resultados obtenidos por GRASP_LS2 a lo largo de las quince repeticiones del experimento sobre los conjuntos de instancias Grids, Harwell-Boeing y Small, de manera respectiva. Si el p-value calculado por el test de Friedman es menor al nivel de significancia α dado, se rechaza la hipótesis nula de que el algoritmo probado presenta el mismo comportamiento a lo largo de todas las experimentaciones y por lo tanto se determina que su desempeño es inestable, en caso contrario es aceptada y el algoritmo presenta un comportamiento estable para es conjunto de instancias evaluado.

Tabla 5.1: Test de Friedman para GRASP_LS2 - Grids, $\alpha = 0.05$

Experimento	p-value	H_0 rechazada
GRASP_LS2-1	0.0948	No
GRASP_LS2-2	0.0966	No
GRASP_LS2-3	0.1096	No
GRASP_LS2-4	0.1156	No
GRASP_LS2-5	0.1938	No
GRASP_LS2-6	0.1969	No
GRASP_LS2-7	0.2000	No
GRASP_LS2-8	0.2159	No
GRASP_LS2-9	0.2259	No
GRASP_LS2-10	0.3179	No
GRASP_LS2-11	0.3673	No
GRASP_LS2-12	0.3961	No
GRASP_LS2-13	0.6082	No
GRASP_LS2-14	0.6269	No

Mientras tanto, en las Tablas 5.4, 5.5 y 5.6, se muestra los resultados del análisis estadístico de Friedman de los resultados obtenidos por la metaheurística GRASP_noLS a lo largo de las quince repeticiones del experimento sobre los conjuntos de instancias Grids, Harwell-Boeing y Small, de manera respectiva. Si el p-value calculado por el Test de Friedman es menor al nivel de significancia α dado, se rechaza la hipótesis nula de que el algoritmo

Tabla 5.2: Test de Friedman para GRASP_LS2 - HB, $\alpha = 0.05$

Experimento	p-value	H_0 rechazada
GRASP_LS2-1	0.0544	No
GRASP_LS2-2	0.2223	No
GRASP_LS2-3	0.2456	No
GRASP_LS2-4	0.3011	No
GRASP_LS2-5	0.3827	No
GRASP_LS2-6	0.4924	No
GRASP_LS2-7	0.5031	No
GRASP_LS2-8	0.5031	No
GRASP_LS2-9	0.5140	No
GRASP_LS2-10	0.5305	No
GRASP_LS2-11	0.6111	No
GRASP_LS2-12	0.6655	No
GRASP_LS2-13	0.6717	No
GRASP_LS2-14	0.8587	No

Tabla 5.3: Test de Friedman para GRASP_LS2 - Small, $\alpha = 0.05$

Experimento	p-value	H_0 rechazada
GRASP_LS2-1	0.3651	No
GRASP_LS2-2	0.4376	No
GRASP_LS2-3	0.4376	No
GRASP_LS2-4	0.5177	No
GRASP_LS2-5	0.5177	No
GRASP_LS2-6	0.5177	No
GRASP_LS2-7	0.6048	No
GRASP_LS2-8	0.6048	No
GRASP_LS2-9	0.6048	No
GRASP_LS2-10	0.6048	No
GRASP_LS2-11	0.6979	No
GRASP_LS2-12	0.6979	No
GRASP_LS2-13	0.8971	No
GRASP_LS2-14	1.0000	No

probado presenta el mismo comportamiento a lo largo de todas las experimentaciones y por lo tanto se determina que su desempeño es inestable, en caso contrario es aceptada y el algoritmo presenta un comportamiento estable para es conjunto de instancias evaluado.

Tabla 5.4: Test de Friedman para GRASP_noLS - Grids, $\alpha = 0.05$

Experimento	p-value	H_0 rechazada
GRASP_noLS-1	0.0023	Sí
GRASP_noLS-2	0.0072	Sí
GRASP_noLS-3	0.0076	Sí
GRASP_noLS-4	0.0080	Sí
GRASP_noLS-5	0.0094	Sí
GRASP_noLS-6	0.0154	Sí
GRASP_noLS-7	0.0166	Sí
GRASP_noLS-8	0.0226	Sí
GRASP_noLS-9	0.0378	Sí
GRASP_n0LS-10	0.0984	No
GRASP_n0LS-11	0.1262	No
GRASP_n0LS-12	0.1547	No
GRASP_n0LS-13	0.2505	No
GRASP_noLS-14	0.3961	No

En la Tabla 5.7 se muestra el ranking obtenido por el test de Friedman en el que valores mas altos, corresponden a algoritmos con un mejor desempeño al obtener las soluciones. Además, en el mismo test se calculó un p-value de $3.9153E-11$ por lo que dado un nivel de significancia $\alpha = 0.05$, se rechaza la hipótesis nula de que ambos algoritmos son equivalentes y por tanto se concluye que para el caso del conjunto de instancias Grids, las diferencias entre ambos algoritmos son significativas.

En la Tabla 5.8 se muestra el ranking obtenido por el Test de Friedman, en el que valores mas altos corresponden a algoritmos con un mejor desempeño al obtener las soluciones. Además, en el mismo test, se calculó un p-value de

Tabla 5.5: Test de Friedman para GRASP_noLS - HB, $\alpha = 0.05$

Experimento	p-value	H_0 rechazada
GRASP_noLS-1	0.0026	Sí
GRASP_noLS-2	0.0157	Sí
GRASP_noLS-3	0.0320	Sí
GRASP_noLS-4	0.0348	Sí
GRASP_noLS-5	0.0502	No
GRASP_noLS-6	0.1189	No
GRASP_noLS-7	0.1358	No
GRASP_noLS-8	0.1472	No
GRASP_noLS-9	0.1472	No
GRASP_noLS-10	0.1697	No
GRASP_noLS-11	0.3339	No
GRASP_noLS-12	0.3920	No
GRASP_noLS-13	0.4062	No
GRASP_noLS-14	0.4456	No

Tabla 5.6: Test de Friedman para GRASP_noLS - Small, $\alpha = 0.05$

Experimento	p-value	H_0 rechazada
GRASP_noLS-1	0.0376	Sí
GRASP_noLS-2	0.0454	Sí
GRASP_noLS-3	0.0454	Sí
GRASP_noLS-4	0.0830	No
GRASP_noLS-5	0.1268	No
GRASP_noLS-6	0.2206	No
GRASP_noLS-7	0.2239	No
GRASP_noLS-8	0.2305	No
GRASP_noLS-9	0.3836	No
GRASP_noLS-10	0.4740	No
GRASP_noLS-11	0.6979	No
GRASP_noLS-12	0.7236	No
GRASP_noLS-13	0.8630	No
GRASP_noLS-14	0.8630	No

Tabla 5.7: Ranking obtenido por el Test de Friedman para GRASP_LS2 vs. GRASP_noLS, conjunto Grids

Algoritmo	Ranking
GRASP_noLS	1.0062499999999985
GRASP_LS2	1.993749999999997

$4.9511 - 11$, por lo que dado un nivel de significancia $\alpha = 0.05$, se rechaza la hipótesis nula de que ambos algoritmos son equivalentes y por tanto se concluye que para el caso del conjunto de instancias Grids, las diferencias entre ambos algoritmos son significativas.

Tabla 5.8: Ranking obtenido por el Test de Friedman para GRASP_LS2 vs. GRASP_noLS, conjunto HB

Algoritmo	Ranking
GRASP_noLS	1.0057471264367808
GRASP_LS2	1.9942528735632163

En la Tabla 5.9 se muestra el ranking obtenido por el Test de Friedman, en el que valores mas altos, corresponden a algoritmos con un mejor desempeño al obtener las soluciones. Además, en el mismo test, se calculó un p-value de $4.0358 - 11$, por lo que dado un nivel de significancia $\alpha = 0.05$, se rechaza la hipótesis nula de que ambos algoritmos son equivalentes y por tanto se concluye que para el caso del conjunto de instancias Grids, las diferencias entre ambos algoritmos son significativas.

En las Tablas 5.10, 5.11 y 5.12 se presentan los resultados de las dos configuraciones de GRASP propuestas, GRASP_noLS y GRASP_LS2.

En las Tablas, *Avg.* corresponde a la media obtenida de las soluciones obtenidas en cada conjunto, *#Opt* el número de instancias de las cuales se

Tabla 5.9: Ranking obtenido por el Test de Friedman para GRASP_LS2 vs. GRASP_noLS, conjunto Small

Algoritmo	Ranking
GRASP_noLS	1.0059523809523796
GRASP_LS2	1.9940476190476162

obtuvo la solución óptima o la mejor conocida, *Dev.* el porcentaje promedio de desviación de las soluciones obtenidas con respecto al óptimo o mejor conocido de sus respectivas instancias y *Time*, el tiempo en segundos que tomo resolver todo el conjunto de instancias.

Tabla 5.10: Resultados de las metaheurísticas GRASP_LS2 y GRASP_noLS sobre el conjunto de instancias Grids

Algoritmo	Avg.	%Dev	#Opt	Time
GRASP_nLS	51.0125	71.89	1	32.04
GRASP_LS2	14.45	3.87	27	60.22

Tabla 5.11: Resultados de las metaheurísticas GRASP_LS2 y GRASP_noLS sobre el conjunto de instancias Harwell-Boeing

Algoritmo	Avg.	%Dev	#Opt	Time
GRASP_nLS	678.79	70.99	1	92.73
GRASP_LS2	317.81	3.53	46	281.38

Tabla 5.12: Resultados de las metaheurísticas GRASP_LS2 y GRASP_noLS sobre el conjunto de instancias Small

Algoritmo	Avg.	%Dev	#Opt	Time
GRASP_nLS	8.02	38.06	1	0.03
GRASP_LS2	4.91	0	84	0.2296

5.5. Conclusiones

Con las pruebas realizadas en la sección anterior se demostró que el comportamiento del algoritmo GRASP_LS2 fue estable a lo largo de las quince repeticiones del experimento en los tres conjuntos de instancias, con lo que se puede concluir que el comportamiento del algoritmo es consistente, como se puede comprobar en las Tablas 5.1, 5.2 y 5.3.

Mientras, el algoritmo GRASP_noLS aunque fue relativamente estable a lo largo de los experimentos realizados, tiene una mayor dependencia en la aleatoriedad por la naturaleza misma del método constructivo empleado, al no tener un comportamiento consistente en parte de las experimentaciones, como se puede comprobar en las Tablas 5.4, 5.5 y 5.6.

En cuanto a la calidad de las soluciones obtenidas por ambos algoritmos a lo largo de la batería de experimentos, en las Tablas 5.7, 5.8 y 5.9 se puede observar que el algoritmo GRASP_LS2 fue el que obtuvo los mejores resultados de acuerdo al ranking calculado por el test de Friedman y puede verse mucho más claramente en las Tablas 5.10, 5.11 y 5.12, en las que son mostrados los resultados de las metaheurísticas por conjuntos de instancias.

Así, se concluye que la integración de un mecanismo de búsqueda local, en este caso, la función LS2, descrita en el Capítulo 4, aporta no solo estabilidad al algoritmo, sino que adicionalmente aporta calidad que como se comprobó, no fue debido al azar.

CAPÍTULO 6

ALGORITMO DE PROCESAMIENTO CELULAR BASADO EN BÚSQUEDA LOCAL ITERADA

En este capítulo se describirán la metaheurística de Búsqueda Local Iterada con Algoritmo de Procesamiento Celular que fue implementado para resolver el Problema del Ancho de Corte, el cual hace uso de las técnicas para el diseño de búsquedas locales eficientes descritas en el Capítulo 4.

6.1. Introducción

El Algoritmo de Procesamiento Celular mostrado también en la Figura 6.1, es un marco de trabajo flexible sobre el cual es posible construir soluciones escalables y configurables, concebida desde un principio para ser ejecu-

ción en paralelo, pero que incluso en su versión secuencial, resulta altamente eficiente.

6.2. Implementación

El Algoritmo de Procesamiento Celular consta de los componentes siguientes, un *pool* de soluciones construidas y generadas aleatoriamente con una proporción de 50 %. En este caso y un conjunto de células que son metaheurísticas sencillas autocontenidas e independientes unas de otras durante proceso de mejoramiento de las soluciones que les fueron asignadas del *pool* de soluciones. Pero que una vez este es concluido, es decir cuando se llega al punto de estancamiento, son capaces de compartir información entre ellas para establecer una sinergía que les permita obtener soluciones de muy alta calidad en comparación de si se tratará simplemente de la misma metaheurística en su versión tradicional.

Las células en este caso fueron algoritmos ILS diseñados para que el punto de convergencia al estancamiento sucediera de manera rápida. Ademas, cada una de ellas tiene asignado un conjunto de parámetros que les confiere la cualidad de ser metaheurísticas de intensificación o de diversificación.

El método de compartición de información en el Algoritmo de Procesamiento Celular aquí implementado consistió en asignarles a las células de intensificación las soluciones del *pool* con peor valor objetivo y a las de diversificación, aquellas con los mejores valores objetivos, este proceso de reasignación se repite cada vez que todas y cada una de las células estancan a sus respectivas soluciones.

Finalmente un proceso de mayor jerarquía controla la mejor solución obtenida por todas las células y verifica el punto en que ninguna de las células es capaz de mejorar las soluciones que les fueron asignadas.

La metaheurística seleccionada para servir como el núcleo de las células del Algoritmo de Procesamiento Celular fue el Algoritmo ILS 15, mostrado

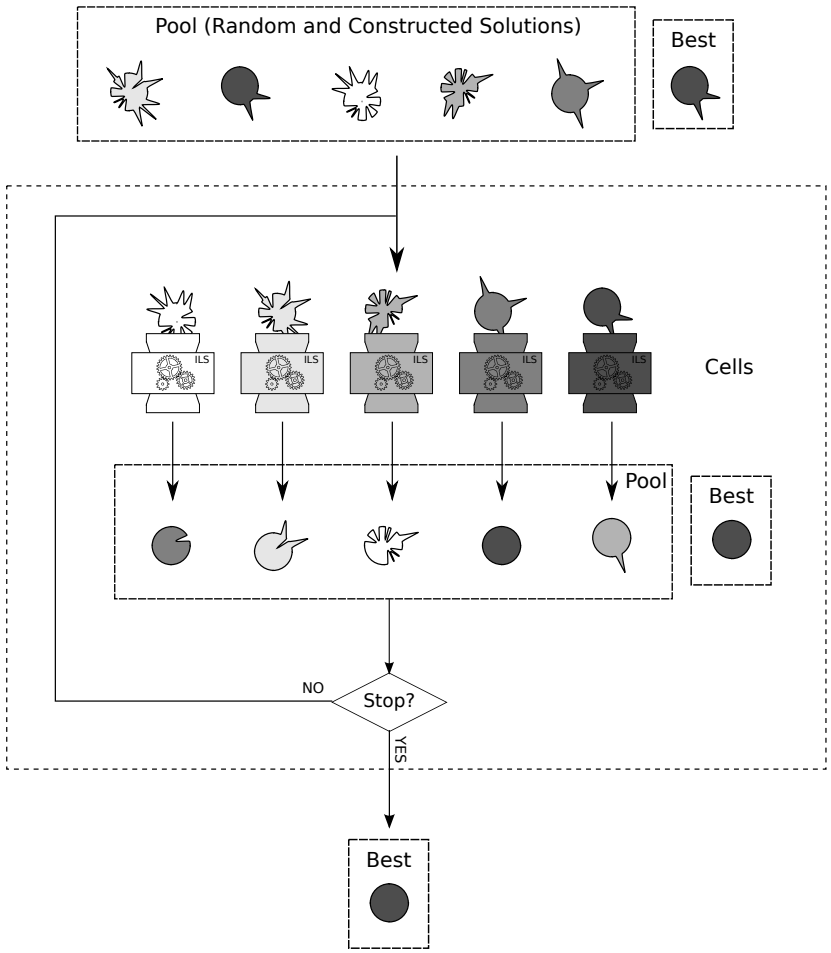


Figura 6.1: Algoritmo Celular-ILS

también en la Figura 6.2.

Esta elección fue debida a la sencillez y alta configurabilidad de su estructura, así como también, debido a que es una metaheurística altamente dependiente de mecanismos de búsqueda local, los cuales son el eje central de este proyecto de tesis.

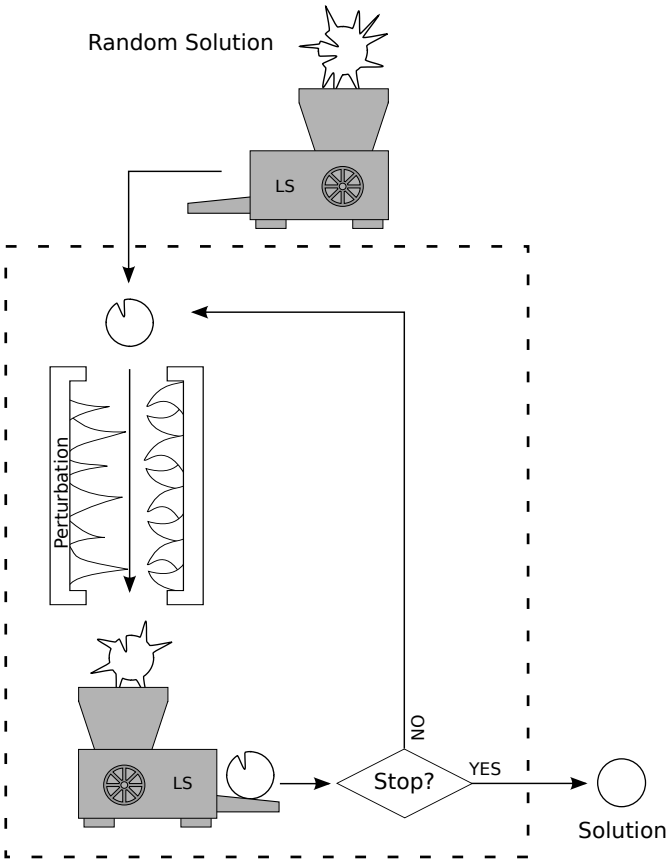


Figura 6.2: Metaheurística ILS

6.3. Experimentación

Las instancias empleadas como referencia fueron las *Grids*, de dificultad media, con un rango de 9 a 729 nodos, de las cuales su valor óptimo es conocido y las $Harwell - Boeing$, de dificultad alta, de las cuales solamente existen mejores valores conocidos.

Los dos grupos de instancias son codificadas de dos formas, la primera mediante un diccionario de claves en el que sólo se almacenan los valores distintos de cero, utilizada por $Harwell - Boeing$, y la segunda, mediante su matriz de adyacencia, en la que se encuentran codificadas las instancias del

Algoritmo 15 Metaheurística ILS

Require: $Solucion$
Require: $Instancia$

```
1: solucionAleatoria(Solucion)
2: calculoInicialCutwdith(Solucion)
3: busquedaLocal(Solucion)
4: mejorSolucion = Solucion
5: while iteracionesSinMejora < maxIteracionesSinMejora do
6:    perturbarPermutacion(Solucion, porcentajePerturbacion)
7:    busquedaLocal(Solucion)
8:    if Solucion.cw < mejorSolucion.cw then
9:       mejorSolucion = Solucion
10:      iteracionesSinMejora = 0
11:   else
12:      iteracionesSinMejora++
13:   end if
14: end while
15: optimoLocal(mejorSolucion)
16: return mejorSolucion
```

conjunto *Grids*.

Estos conjuntos de instancias pueden ser encontrados en el sitio web del Proyecto Optsicom [Martí R. and E.G., 2010].

El algoritmo de estructura celular propuesto fue desarrollado en el lenguaje de programación C++, partiendo del uso de su conjunto de librerías estándar, y compilada en el IDE XCode en su versión 4.2 bajo el sistema operativo Mac OS X 10.7.2.

Las entradas que el programa necesita para operar consisten en la ruta del archivo de instancia a resolver, el valor óptimo de la instancia, o en su defecto, el mejor valor conocido, la codificación de la instancia y el nombre del archivo en el cual se guardarán los resultados.

La experimentación fue efectuada en un portátil Apple MacBook Pro, con un procesador Intel i5 a 2.3 GHz, 4 GB de RAM y sistema operativo

Mac OS X 10.7.2.

El criterio de parada global establecido para el algoritmo de estructura celular fue de dos iteraciones consecutivas sin mejora y se crearon diez células de procesamiento ILS para resolver las instancias.

La batería completa de experimentos, con los tres conjuntos de instancias fue realizada con un total de quince repeticiones, para obtener una muestra que produzca datos estadísticamente significativos.

A la configuraciones propuesta le fue aplicado el Test de Friedman bajo las mismas hipótesis descritas en la experimentación del Capítulo 4. El Test de Friedman es un test estadístico no paramétrico desarrollado por el economista Milton Friedman, similar al test ANOVA. El test individual en este caso, sirve para observar si el comportamiento de cada función de búsqueda es estable, es decir, si no existen diferencias significativas en los quince conjuntos resultados obtenidos en los experimentos.

Para realizar esta prueba, se hizo uso del software computacional desarrollado por el Soft Computing and Intelligent Information Systems [SCI2S, 2010].

6.4. Resultados

En el Apéndice B se pueden consultar los resultados obtenidos por el algoritmo CELLS por instancia en una de las experimentaciones realizadas mientras que en el Apéndice D se encuentran los valores óptimos y mejores conocidos de las instancias evaluadas.

En las Tablas 6.1, 6.2 y 6.3, se muestra los resultados del análisis estadístico de Friedman de los resultados obtenidos por la metaheurística CELLS a lo largo de las quince repeticiones del experimento sobre los conjuntos de instancias Grids, Harwell-Boeing y Small, de manera respectiva. Si el p-value calculado por el Test de Friedman es menor al nivel de significancia α dado, se rechaza la hipótesis nula de que el algoritmo probado presenta el mismo comportamiento a lo largo de todas las experimentaciones y por lo tanto se

determina que su desempeño es inestable, en caso contrario es aceptada y el algoritmo presenta un comportamiento estable para es conjunto de instancias evaluado.

Tabla 6.1: Test de Friedman para CELLS - Grids, $\alpha = 0.05$

Experimento	p-value	H_0 rechazada
CELLS-1	0.1352	No
CELLS-2	0.1547	No
CELLS-3	0.1735	No
CELLS-4	0.1969	No
CELLS-5	0.2226	No
CELLS-6	0.3815	No
CELLS-7	0.4061	No
CELLS-8	0.4315	No
CELLS-9	0.5017	No
CELLS-10	0.5074	No
CELLS-11	0.5656	No
CELLS-12	0.6778	No
CELLS-13	0.7436	No
CELLS-14	0.8945	No

6.5. Conclusiones

Con las pruebas realizadas en la sección anterior se demostró que el comportamiento del algoritmo CELLS fue estable a lo largo de las quince repeticiones del experimento en los tres conjuntos de instancias, con lo que se puede concluir que el comportamiento del algoritmo es consistente, como se puede comprobar en las Tablas 6.1, 6.2 y 6.3.

Tabla 6.2: Test de Friedman para CELLS - HB, $\alpha = 0.05$

Experimento	p-value	H_0 rechazada
CELLS-1	0.0587	No
CELLS-2	0.1001	No
CELLS-3	0.1426	No
CELLS-4	0.1861	No
CELLS-5	0.2128	No
CELLS-6	0.2321	No
CELLS-7	0.2742	No
CELLS-8	0.2972	No
CELLS-9	0.4207	No
CELLS-10	0.4817	No
CELLS-11	0.7474	No
CELLS-12	0.7927	No
CELLS-13	0.8654	No
CELLS-14	0.9527	No

Tabla 6.3: Test de Friedman para CELLS - Small, $\alpha = 0.05$

Experimento	p-value	H_0 rechazada
CELLS-1	0.6048	No
CELLS-2	0.6048	No
CELLS-3	0.6048	No
CELLS-4	0.6048	No
CELLS-5	0.6048	No
CELLS-6	0.6979	No
CELLS-7	0.6979	No
CELLS-8	0.6979	No
CELLS-9	0.6979	No
CELLS-10	0.6979	No
CELLS-11	0.6979	No
CELLS-12	0.6979	No
CELLS-13	0.6979	No
CELLS-14	0.7958	No

CAPÍTULO 7

EXPERIMENTACIÓN Y RESULTADOS

7.1. Experimentacion GRASP_LS2 vs. CELLS

Una vez que se comprobó que el desempeño de los algoritmos $CELLS$ y $GRASP\_LS2$ es el mismo, tal como puede verificarse en las secciones de Experimentación de los Capítulos 5 y 6, se tomó una de las quince baterías de pruebas de cada uno de los algoritmos, en este caso las encontradas en los Apéndices B y C respectivamente. Ambas pruebas fueron implementadas y probadas bajo las mismas condiciones descritas con anterioridad, e incluyen los mismos grupos de instancias para evaluar el desempeño.

A los algoritmos evaluados les fue aplicado el Test de Friedman, el cual es un test estadístico no paramétrico desarrollado por el economista Milton Friedman, similar al test ANOVA. En este caso, el propósito del test se realizó para determinar si las diferencias observadas entre los algoritmos se deben a la naturaleza aleatoria de ambos o si en realidad el desempeño de uno es

superior al otro.

La medida de desempeño, en este caso, se determino de acuerdo a la ponderación dada por el test de Friedman.

Para realizar esta prueba, se hizo uso del software computacional desarrollado por [SCI2S, 2010].

7.2. Resultados GRASP_LS2 vs. CELLS

En la Tabla 7.1 se muestra el ranking obtenido por el Test de Friedman, en el que valores mas altos, corresponden a algoritmos con un mejor desempeño al obtener las soluciones. Además, en el mismo test, se calculó un p-value de $8.6835E-7$, por lo que dado un nivel de significancia $\alpha = 0.05$, se rechaza la hipótesis nula de que ambos algoritmos son equivalentes y por tanto se concluye que para el caso del conjunto de instancias Grids, las diferencias entre ambos algoritmos son significativas.

Tabla 7.1: Ranking obtenido por el Test de Friedman para GRASP_LS2 vs. CELLS, conjunto Grids

Algoritmo	Ranking
GRASP_LS2	1.2250000000000003
CELLS	1.7750000000000004

La Tabla 7.2 se muestra el ranking obtenido por el Test de Friedman, en el que valores mas altos, corresponden a algoritmos con un mejor desempeño al obtener las soluciones. Además, en el mismo test, se calculó un p-value de $1.9596E-9$, por lo que dado un nivel de significancia $\alpha = 0.05$, se rechaza la hipótesis nula de que ambos algoritmos son equivalentes y por tanto se concluye que para el caso del conjunto de instancias HB, las diferencias entre ambos algoritmos son significativas.

Tabla 7.2: Ranking obtenido por el Test de Friedman para GRASP_LS2 vs. CELLS, conjunto HB

Algoritmo	Ranking
GRASPLSHB10	1.1781609195402307
CELLILSHB4	1.8218390804597682

Finalmente, en la Tabla 7.3 se muestra el ranking obtenido por el Test de Friedman, en el que valores mas altos, corresponden a algoritmos con un mejor desempeño al obtener las soluciones. Además, en el mismo test, se calculó un p-value de 0.1266, por lo que dado un nivel de significancia $\alpha = 0.05$, se acepta la hipótesis nula de que ambos algoritmos son equivalentes y por tanto se concluye que para el caso del conjunto de instancias Small, las diferencias entre ambos algoritmos no son significativas.

Tabla 7.3: Ranking obtenido por el Test de Friedman para GRASP_LS2 vs. CELLS, conjunto Small

Algoritmo	Ranking
GRASPLSSM7	1.4166666666666659
CELLILSSM4	1.5833333333333321

7.3. Experimentacion GRASP_LS2 vs. Estado del Arte

En las secciones anteriores, se comprobó, que de las dos metaheurísticas seleccionadas implementadas, el algoritmo GRASP_LS2 resuelvo los conjuntos de instancias Grids y HB con una eficacia menor al del algoritmo CELLS,

y un desempeño similar a la del algoritmo de estructura celular, al resolver el conjunto Small.

Por ello, se resolvió comparar el desempeño de GRASP_LS2 contra aquellas implementaciones existentes del estado del arte, descritas en el artículo [Pantrigo et al., 2010], para observar si los resultados obtenidos son comparables con los de aquellas soluciones del estado del arte implementadas.

Debido a que no se disponen de los datos obtenidos por las metaheurísticas del estado del arte por instancia, se emplearon los criterios de comparación utilizados en el artículo en el que se describe el desempeño de las implementaciones del estado del arte para evaluar la calidad de solución, entre los que se encuentran el promedio de la suma de la media obtenida por las instancias de cada conjunto, el número de óptimos o mejores soluciones conocidas alcanzadas, el porcentaje de desviación con respecto al óptimo o mejor conocido y el tiempo total que tomo resolver cada conjunto de instancias.

7.4. Experimentacion CELLS vs. Estado del Arte

En las secciones anteriores, se comprobó, que de las dos metaheurísticas seleccionadas implementadas, el algoritmo CELLS obtiene los mejores resultados al resolver los conjuntos de instancias Grids y HB, y un desempeño similar al algoritmo GRASP_LS2, al resolver el conjunto Small.

Por ello, se resolvió comparar el desempeño de la metaheurística CELLS contra aquellas implementaciones existentes del estado del arte, descritas en el artículo [Pantrigo et al., 2010].

Debido a que no se disponen de los datos obtenidos por las metaheurísticas del estado del arte por instancia, se emplearon los criterios de comparación utilizados en el artículo en el que se describe el desempeño de las implementaciones del estado del arte para evaluar la calidad de solución, entre los que se encuentran el promedio de la suma de la media obtenida por las instancias

de cada conjunto, el número de óptimos o mejores soluciones conocidas alcanzadas, el porcentaje de desviación con respecto al óptimo o mejor conocido y el tiempo total que tomo resolver cada conjunto de instancias.

7.5. Resultados CELLS vs. Estado del Arte

En las Tablas 7.4, 7.5 y 7.6 se presentan los resultados de la propuesta de solución elegida, contra las metaheurísticas del estado del arte SS [Pantrigo et al., 2010], $GRASP - PR$ [Andrade and Resende, 2007b] y SA [Campos et al., 2006].

En las Tablas, $Avg.$ corresponde a la media obtenida de las soluciones obtenidas en cada conjunto, $\#Opt$ el número de instancias de las cuales se obtuvo la solución óptima o la mejor conocida, $Dev.$ el porcentaje promedio de desviación de las soluciones obtenidas con respecto al óptimo o mejor conocido de sus respectivas instancias y $Time$, el tiempo en segundos que tomo resolver todo el conjunto de instancias.

Tabla 7.4: Resultados de la metaheurística CELLS y las del estado del arte sobre el conjunto de instancias Grids

Algoritmo	Avg.	%Dev	#Opt	Time
CELLS	12.48	3.87	60	124.42
SS	13	7.76	44	210.7
GRASP-PR	38.44	201.81	2	235.16
SA	16.14	25.42	37	216.13

Tabla 7.5: Resultados de la metaheurística CELLS y las del estado del arte sobre el conjunto de instancias Harwell-Boeing

Algoritmo	Avg.	%Dev	#Opt	Time
CELLS	317.81	3.53	46	281.38
SS	315.22	1.42	59	430.56
GRASP-PR	364.83	90.97	2	557.48
SA	346.21	48.97	8	435.41

Tabla 7.6: Resultados de la metaheurística CELLS y las del estado del arte sobre el conjunto de instancias Small

Algoritmo	Avg.	%Dev	#Opt	Time
CELLS	4.91	0	84	0.2296
SS	4.91	0	84	0.07
GRASP-PR	5.2	6.54	60	0.07
SA	5.15	5.6	64	0.07

CAPÍTULO 8

CONCLUSIONES Y TRABAJOS FUTUROS

En esta sección se retomarán las conclusiones alcanzadas en los Capítulos 4, 5 y 6, así como también los resultados del análisis experimental realizado en el Capítulo 7 para determinar si la hipótesis planteada, es aceptada.

8.1. Conclusiones

Los objetivos principales de este proyecto de investigación fueron alcanzados, tal como se describe en el Capítulo 4, en el cual fueron desarrollados mecanismos de calculo de la función objetivo que permitieron el desarrollo de búsquedas locales altamente eficientes. Ademas, se realizo un análisis de la función de búsqueda local básica, buscando con ello, aislar los componentes principales que la integran, y con ello, desarrollar nuevas propuestas enfocadas a resolver el Problema del Ancho de Corte, los cuales, cuando son combinados entre ellos, derivan en distintas funciones de búsqueda local

altamente eficientes gracias a los componentes desarrollados.

Como se determinó también en el Capítulo 4, el uso de búsquedas locales como único mecanismo de solución de problemas de optimización no es suficiente, pues aunque el desempeño en algunos casos es muy bueno, no ofrece estabilidad en la calidad de las soluciones obtenidas, pues la aleatoriedad inherente a estas funciones juega un papel muy importante en la resolución de las instancias del problema.

Adicionalmente a los objetivos de este proyecto, se aprovechó el potencial de las funciones creadas en el Capítulo 4, para el desarrollo que metaheurísticas que integran mecanismos de búsqueda local, entre las que se encuentran el Procedimiento de Búsqueda Adaptativa Aleatoria Voraz, GRASP y el Algoritmo de Procesamiento Celular Basado en Búsqueda Local Iterada, CELLS, descritos en los Capítulos 5 y 6, respectivamente.

En el Capítulo 5, fue incluida la función de búsqueda local que reporto el mejor desempeño, LS2, como parte de la metaheurística GRASP, que como fue observado, dado el método constructivo que incorpora, no solamente incrementa la calidad de las soluciones producidas para las instancias del problema, sino también se observa como esta se mantiene estable a lo largo de todas las experimentaciones realizadas.

Se evaluó también la metaheurística GRASP sin la función de búsqueda local para determinar si la causante de la mejora en la calidad y estabilidad del algoritmo recaía en el método constructivo. De esta configuración se pudo concluir, una vez realizados los análisis estadísticos pertinentes, que ni la búsqueda local ni el método constructivo por si solos, son responsables de la calidad de la metaheurística, pues la búsqueda local revela su potencial cuando trabaja en sinergía con otras estrategias heurísticas, pues en la configuración GRASP sin búsqueda local, la calidad de las soluciones y la estabilidad de estas se reducían considerablemente.

Aprovechando el conocimiento obtenido, se implementó el algoritmo de estructura celular, CELLS, el cual como emplea como célula de procesamien-

to una metaheurística ILS, que depende primordialmente en las funciones de búsqueda local, las cuales partieron de un pool compuesto en parte por soluciones obtenidas del método constructivo implementado para el algoritmo GRASP. En el Capítulo 6 se concluyó finalmente que ciertamente el algoritmo conserva la estabilidad en la obtención de soluciones candidatas al resolver los diferentes conjuntos de instancias, respaldado por el análisis estadístico realizado.

Posteriormente, dados los resultados observados por el ranking generado por el test de Friedman realizado sobre las dos metaheurísticas implementadas, GRASP_LS2 y CELLS, se concluye que la metaheurística CELLS obtiene de manera consistente los mejores resultados sobre los conjuntos de instancias Grids y Harwell-Boeing, con un nivel de significancia $\alpha = 0.05$ y aunque en el conjunto Smalls, el algoritmo CELLS también tiene una mejor valoración en el ranking generado por el Test de Friedman que la metaheurística GRASP_LS2, el p-value calculado por el mismo test fue de 0.1266, por lo que la hipótesis nula en este caso no fue rechazada, asignando en este caso un desempeño estadísticamente similar.

Finalmente, dado que el algoritmo CELLS fue el que mejor desempeño obtuvo, fue el seleccionado para ser comparado con las propuestas de solución del estado del arte, de los cuales, como se puede observar en las Tablas 7.4, 7.5 y 7.6 del Capítulo 7, tiene un desempeño superior en los tres casos al de los algoritmos GRASP-PR y SA y un desempeño similar al del algoritmo SS. Sin embargo, como también se puede observar en los tiempos en los que le que tomó resolver al algoritmo CELLS cada conjunto de instancias son siempre menores a los de los algoritmos del estado del arte.

Los resultados obtenidos permiten concluir que el desarrollo de mecanismos de búsqueda local eficientes y de metaheurísticas que hagan uso de ellos, hacen posible obtener soluciones equiparables en calidad a aquellas del estado del arte y en algunos casos superiores, aceptándose con ello la hipótesis planteada al inicio de este proyecto de investigación.

Con los conocimientos obtenidos en el desarrollo del presente trabajo de investigación concernientes a la implementación de búsquedas locales y metaheurísticas eficientes, se realizó una estancia de investigación en la Universidad de Luxemburgo, con el asesoramiento del Dr. Johnatan Pecero y el Prof. Pascal Bouvry de la Universidad de Luxemburgo, Luxemburgo y del M.C. Jędrzej Musiał y el Prof. Jacek Błazewicz de la Universidad Tecnológica de Poznań, Polonia y el Prof. Hector Fraire, del Instituto Tecnológico de Ciudad Madero, México, referentes al Problema de Compras por Internet [Blazewicz et al., 2010], del cual se desarrollo un Algoritmo de Procesamiento Celular Basado en Búsqueda Local Iterada y un algoritmo exacto Branch & Bound para las versiones con y sin descuentos.

8.2. Trabajos Futuros

En lo referente al desarrollo de búsquedas locales, se recomienda el desarrollo de componentes básicos adicionales tales como nuevas funciones de inicialización, de selección de elementos críticos y de reposicionamiento, aprovechando las funciones optimizadas de calculo de la función objetivo y de exploración del vecindario implementadas.

Dado que el enfoque de este trabajo de investigación no estaba centrado en el desarrollo de metaheurísticas, se recomienda afinar los algoritmos existentes mediante el análisis y el ajuste de parámetros de los mismos.

También se sugiere, dados los tiempos logrados gracias a la implementación de búsquedas locales eficientes, incorporar heurísticas adicionales que aprovechen el tiempo ganado en mejorar las soluciones candidatas mediante mecanismos de intensificación y diversificación.

Finalmente, se recomienda explorar nuevos núcleos de procesamiento para el algoritmo de procesamiento celular, principalmente, metaheurísticas del tipo poblaciónal, tales como Scatter Search y Algoritmo Memético, o tomando ventaja de la estructura misma del algoritmo, combinaciones de diferentes

tipos de metaheurísticas tanto poblacionales como trayectoriales.

BIBLIOGRAFÍA

[Andrade and Resende, 2007a] Andrade, D. and Resende, M. (2007a). GRASP with evolutionary path-relinking. In *Proc. of Seventh Metaheuristics International Conference (MIC 2007)*. Citeseer.

[Andrade and Resende, 2007b] Andrade, D. and Resende, M. (2007b). GRASP with path-relinking for network migration scheduling. In *Proceedings of the International Network Optimization Conference*. Citeseer.

[Blazewicz et al., 2010] Blazewicz, J., Kovalyov, M., Musial, J., Urbanski, A., and Wojciechowski, A. (2010). Internet shopping optimization problem. *International Journal of Applied Mathematics and Computer Science*, 20(2):385–390.

[Campos et al., 2006] Campos, V., Piñana, E., and Martí, R. (2006). Adaptive memory programming for matrix bandwidth minimization. *Annals of Operations Research*, pages 1–17.

[Cohoon and Sahni, 1983] Cohoon, J. and Sahni, S. (1983). Heuristics for the board permutation problem. In *Proc. Int. Conf. Computer-Aided Design*, pages 81–83.

[Dalgaard, 2008] Dalgaard, P. (2008). *Introductory Statistics with R*. Statistics and Computing. Springer.

[Duarte et al., 2006] Duarte, A., Laguna, M., and Martí, R. (2006). Tabu search for the linear ordering problem with cumulative costs. *Computational Optimization and Applications*, pages 1–19.

[Duarte Muñoz et al., 2007] Duarte Muñoz, A., Pantrigo Fernández, J., and Gallego Carrillo, M. (2007). *Metaheurısticas*.

[Garey and Johnson, 1976] Garey, M. and Johnson, L. (1976). Some simplified np-complete graph problems. *Theoretical computer science*, 1(3):237–267.

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

[Hromkovic and Oliva, 2001] Hromkovic, J. and Oliva, W. (2001). *Algorithmics for hard problems*. Springer.

[Kelly and Osman, 1996] Kelly, J. P. and Osman, I. H. (1996). *Meta-Heuristics: Theory and Applications*. Kluwer Academic Publishers Norwell, MA, USA.

[Kreyszig, 2007] Kreyszig, E. (2007). *Advanced engineering mathematics*. Wiley-India.

[Martí R. and E.G., 2010] Martí R., Pantrigo J.J., D. A. and E.G., P. (2010). Optsicom project. `http://heur.uv.es/optsicom/cutwidth/`.

[Nicholson, 2007] Nicholson, T. A. J. (2007). *Optimization in industry: Optimization techniques*. Aldine.

[Pantrigo et al., 2010] Pantrigo, J., Martí, R., Duarte, A., and Pardo, E. (2010). Scatter Search for the Cutwidth Minimization Problem.

[Pavan and Todeschini, 2008] Pavan, M. and Todeschini, R. (2008). *Scientific Data Ranking Methods: Theory and Applications*. Data Handling in Science and Technology. Elsevier Science.

[Polya, 1971] Polya, G. (1971). *How to solve it: A new aspect of mathematical method*. Princeton University Press Princeton, NJ.

[Reeves, 1993] Reeves, C. R. (1993). *Modern heuristic techniques for combinatorial problems*. John Wiley & Sons, Inc. New York, NY, USA.

[Schiavinotto and Stützle, 2004] Schiavinotto, T. and Stützle, T. (2004). The linear ordering problem: Instances, search space analysis and algorithms. *Journal of Mathematical Modelling and Algorithms*, 3(4):367–402.

[SCI2S, 2010] SCI2S (2010). Soft computing and intelligent information systems. `http://sci2s.ugr.es/sicidm/#nine`.

Apéndices

APÉNDICE A

DISEÑO DE ESTRUCTURAS DE DATOS

En esta sección se mostrara el diseño de las estructuras que sirvieron como base para los algoritmos implementados con el objetivo de resolver CWP.

A.1. Estructura *Instance*

La estructura $Instance$, mostrada en la Figura A.1, tiene como proposito el almacenar la información relativa a la instancia de CW que se quiere resolver.

Entre las estructuras internas que contiene, se encuentra la matriz de adyacencia, que representa a un grafo no dirigido del cual se desea encontrar el arreglo lineal con el menor numero de cortes posibles en el vector bidimensional $adjMat$. Incluye también la variable $size$, la cual indica el número de nodos que tiene el grafo representado en la matriz de adyacencia.

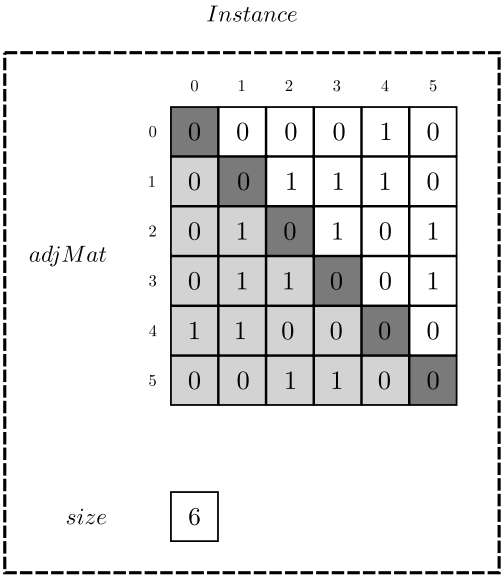


Figura A.1: Estructura *Instance*

A.2. Estructura *Solution*

La estructura *Solution*, mostrada en la Figura A.4, representa una solución candidata del arreglo lineal que se observa en la Figura A.2, la cual resuelve CWP para el grafo no dirigido almacenado en la estructura *Instance*.

Para poder representar adecuadamente este arreglo lineal, la estructura *Solution* se compone de las subestructuras siguientes:

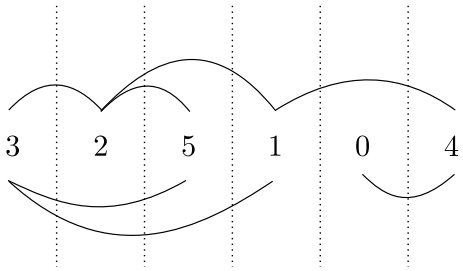


Figura A.2: Arreglo lineal

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

Figura A.3: Matriz de Adyacencia del Arreglo Lineal de la Figura A.2

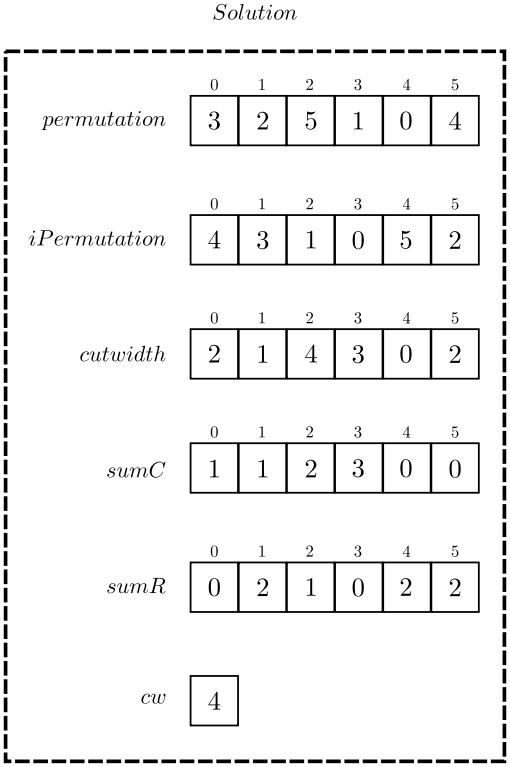


Figura A.4: Estructura *Solution*

permutation Almacena el arreglo lineal del grafo que intenta dar solución a la instancia que se quiere resolver.

iPermutation Tiene la función de actuar como índice para que el costo de acceder a un elemento en particular del arreglo lineal sea constante. Por ejemplo, en el caso de que se desee encontrar la posición del nodo con el etiquetado 3 del grafo en el arreglo lineal, solo es necesario acceder a la posición 3 del vector *iPermutacion*, el cual tiene almacenado el valor 0, que es el orden que ocupa el nodo 3 dentro del vector *permutation*.

cutwidth Guarda el cutwith que tienen los elementos del arreglo lineal de acuerdo a su etiquetado. En este caso, el nodo con la etiqueta 0 tiene un *cutwidth* igual a 2, como puede comprobarse en la Figura A.2

sumC Contiene las sumas de las columnas de cada uno de los nodos del grafo del triángulo superior de la matriz de adyacencia, dispuesta de acuerdo al orden establecido por el arreglo lineal, es decir, tal como se muestra en la Figura A.3. Sin embargo, la constante transformación de la matriz de adyacencia del grafo, resulta muy costosa computacionalmente, por lo que se cuenta con un algoritmo que realiza el cálculo sobre la matriz original de la estructura *Instance*, el cual será presentado en secciones posteriores.

sumR Contiene las sumas de los renglones de cada uno de los nodos del grafo del triángulo superior de la matriz de adyacencia, dispuesta de acuerdo al orden establecido por el arreglo lineal, tal como puede observarse en la Figura A.3.

cw Almacena el valor del *cutwidth* del grafo, dado el arreglo lineal actual.

APÉNDICE B

RESULTADOS ALGORITMO CELLS

B.1. Resultados del Algoritmo CELLS sobre el conjunto Grids

Tabla B.1: Resultados del Algoritmo CELLS sobre el conjunto de instancias Grids

Instancias	Cutwidth	Tiempo
Grid12x12	13	0.179607
Grid12x15	13	0.297291
Grid12x18	13	0.414772
Grid12x21	13	0.752375
Grid12x24	13	1.25948
Grid12x27	13	1.42221
Grid12x3	4	0.005363
Grid12x6	7	0.035292
Grid12x9	10	0.078118

Tabla B.1: Resultados del Algoritmo CELLS sobre el conjunto de instancias Grids

Instancias	Cutwidth	Tiempo
Grid15x12	15	0.298438
Grid15x15	16	0.519611
Grid15x18	16	0.900053
Grid15x21	16	1.38183
Grid15x24	16	1.91855
Grid15x27	16	3.0058
Grid15x3	4	0.009219
Grid15x6	8	0.048213
Grid15x9	12	0.172574
Grid18x12	15	0.489882
Grid18x15	21	0.782099
Grid18x18	19	1.34191
Grid18x21	19	2.70669
Grid18x24	19	4.27481
Grid18x27	19	4.46138
Grid18x3	4	0.017219
Grid18x6	8	0.154631
Grid18x9	11	0.257358
Grid21x12	13	0.634677
Grid21x15	19	1.30617
Grid21x18	22	1.80461
Grid21x21	22	4.42364
Grid21x24	22	4.49499
Grid21x27	22	8.54235
Grid21x3	4	0.020675
Grid21x6	7	0.134659

Tabla B.1: Resultados del Algoritmo CELLS sobre el conjunto de instancias Grids

Instancias	Cutwidth	Tiempo
Grid21x9	11	0.370907
Grid24x12	14	1.03695
Grid24x15	18	1.9548
Grid24x18	26	3.41876
Grid24x21	29	7.0043
Grid24x24	25	7.45589
Grid24x27	25	10.6949
Grid24x3	4	0.025905
Grid24x6	7	0.248866
Grid24x9	10	0.347819
Grid27x12	15	1.55909
Grid27x15	20	2.15707
Grid27x18	23	4.65531
Grid27x21	28	7.40889
Grid27x24	31	8.97061
Grid27x27	28	15.4936
Grid27x3	4	0.034007
Grid27x6	7	0.182938
Grid27x9	11	0.475846
Grid3x12	4	0.004766
Grid3x15	4	0.007305
Grid3x18	4	0.011141
Grid3x21	4	0.014976
Grid3x24	4	0.020472
Grid3x27	4	0.027176
Grid3x3	4	0.0005

Tabla B.1: Resultados del Algoritmo CELLS sobre el conjunto de instancias Grids

Instancias	Cutwidth	Tiempo
Grid3x6	4	0.001424
Grid3x9	4	0.002825
Grid6x12	7	0.028114
Grid6x15	7	0.059192
Grid6x18	7	0.065201
Grid6x21	7	0.143796
Grid6x24	7	0.158105
Grid6x27	7	0.221035
Grid6x3	4	0.001451
Grid6x6	7	0.006334
Grid6x9	7	0.020005
Grid9x12	10	0.096034
Grid9x15	10	0.124378
Grid9x18	10	0.224882
Grid9x21	10	0.531599
Grid9x24	10	0.410488
Grid9x27	10	0.585013
Grid9x3	4	0.003053
Grid9x6	7	0.015128

B.2. Resultados del Algoritmo CELLS sobre el conjunto HB

Tabla B.2: Resultados del Algoritmo CELLS sobre el conjunto de instancias HB

Instancias	Cutwidth	Tiempo
494_bus.mtx.rnd	21	2.86209
662_bus.mtx.rnd	33	8.76293
685_bus.mtx.rnd	34	9.42256
arc130.mtx.rnd	202	0.131267
ash292.mtx.rnd	40	1.15527
ash85.mtx.rnd	16	0.07317
bcspwr01.mtx.rnd	5	0.008654
bcspwr02.mtx.rnd	6	0.016168
bcspwr03.mtx.rnd	10	0.113499
bcspwr04.mtx.rnd	32	0.794045
bcspwr05.mtx.rnd	21	4.41975
bcsstk01.mtx.rnd	33	0.023604
bcsstk02.mtx.rnd	1089	0.021077
bcsstk04.mtx.rnd	310	0.163625
bcsstk05.mtx.rnd	115	0.186539
bcsstk06.mtx.rnd	226	4.46865
bcsstk20.mtx.rnd	21	2.92532
bcsstk22.mtx.rnd	13	0.120501
bcsstm07.mtx.rnd	234	2.59141
can__144.mtx.rnd	25	0.16545
can__161.mtx.rnd	54	0.194453
can__292.mtx.rnd	100	1.11102
can__445.mtx.rnd	126	4.62472
curtis54.mtx.rnd	13	0.017944
dwt__209.mtx.rnd	56	0.530003
dwt__221.mtx.rnd	29	0.679612

Tabla B.2: Resultados del Algoritmo CELLS sobre el conjunto de instancias HB

Instancias	Cutwidth	Tiempo
dwt__234.mtx.rnd	12	0.144161
dwt__245.mtx.rnd	37	0.610249
dwt__310.mtx.rnd	26	1.26725
dwt__361.mtx.rnd	40	2.43206
dwt__419.mtx.rnd	53	2.98141
dwt__503.mtx.rnd	146	7.05527
dwt__592.mtx.rnd	79	8.77748
fs_183_1.mtx.rnd	187	0.617183
fs_541_1.mtx.rnd	296	5.5262
fs_680_1.mtx.rnd	24	6.37801
gent113.mtx.rnd	95	0.110932
gre_216a.mtx.rnd	52	0.481353
gre__115.mtx.rnd	36	0.176575
gre__185.mtx.rnd	48	0.24576
gre__343.mtx.rnd	72	1.67895
gre__512.mtx.rnd	94	5.04235
hor__131.mtx.rnd	152	3.34617
ibm32.mtx.rnd	23	0.008055
impcol_a.mtx.rnd	48	0.432287
impcol_b.mtx.rnd	56	0.023549
impcol_c.mtx.rnd	48	0.151739
impcol_d.mtx.rnd	64	2.71804
impcol_e.mtx.rnd	174	0.577596
lns__131.mtx.rnd	31	0.126455
lns__511.mtx.rnd	75	7.19339
lund_a.mtx.rnd	113	0.21071

Tabla B.2: Resultados del Algoritmo CELLS sobre el conjunto de instancias HB

Instancias	Cutwidth	Tiempo
lund_b.mtx.rnd	111	0.188161
mbeacxc.mtx.rnd	16430	6.83594
mcca.mtx.rnd	390	0.261739
nnc261.mtx.rnd	46	0.834837
nnc666.mtx.rnd	80	8.94178
nos1.mtx.rnd	4	0.181737
nos2.mtx.rnd	4	8.93451
nos4.mtx.rnd	12	0.069689
nos5.mtx.rnd	193	4.13742
nos6.mtx.rnd	29	12.1323
plat362.mtx.rnd	155	1.86084
plskz362.mtx.rnd	31	1.93643
pores_1.mtx.rnd	17	0.004579
pores_3.mtx.rnd	29	2.47221
saylr1.mtx.rnd	18	0.667015
saylr3.mtx.rnd	48	13.3717
sherman4.mtx.rnd	48	3.8028
shl__200.mtx.rnd	378	19.4852
shl__400.mtx.rnd	392	18.6207
shl____0.mtx.rnd	368	24.0253
steam1.mtx.rnd	182	0.547568
steam2.mtx.rnd	308	5.74677
steam3.mtx.rnd	20	0.02793
str__200.mtx.rnd	571	2.76282
str__600.mtx.rnd	645	2.17723
str____0.mtx.rnd	419	2.47986

Tabla B.2: Resultados del Algoritmo CELLS sobre el conjunto de instancias HB

Instancias	Cutwidth	Tiempo
west0132.mtx.rnd	71	0.194941
west0156.mtx.rnd	56	0.262356
west0167.mtx.rnd	54	0.291063
west0381.mtx.rnd	484	3.0272
west0479.mtx.rnd	301	6.63113
west0497.mtx.rnd	199	7.57524
west0655.mtx.rnd	471	16.4471
will199.mtx.rnd	130	0.534125
will57.mtx.rnd	11	0.021841

B.3. Resultados del Algoritmo CELLS sobre el conjunto Small

Tabla B.3: Resultados del Algoritmo CELLS sobre el conjunto de instancias Small

Instancias	Cutwidth	Tiempo
p17_16_24	7	0.003178
p18_16_21	5	0.001817
p19_16_19	4	0.001501
p20_16_18	4	0.002759
p21_17_20	4	0.004228
p23_17_23	5	0.001943
p24_17_29	8	0.002208
p26_17_19	4	0.001513

Tabla B.3: Resultados del Algoritmo CELLS sobre el conjunto de instancias Small

Instancias	Cutwidth	Tiempo
p27_17_19	4	0.001422
p28_17_18	3	0.001817
p29_17_18	3	0.002045
p30_17_19	4	0.001526
p31_18_21	3	0.002194
p32_18_20	4	0.002199
p33_18_21	4	0.002091
p34_18_21	4	0.00208
p35_18_19	3	0.002056
p36_18_20	4	0.001823
p37_18_20	4	0.002125
p38_18_19	3	0.001636
p39_18_19	3	0.002108
p40_18_32	8	0.001587
p41_19_20	3	0.00218
p42_19_24	5	0.001911
p43_19_22	4	0.002476
p44_19_25	6	0.001841
p45_19_25	5	0.003679
p46_19_20	3	0.002456
p47_19_21	4	0.002762
p48_19_21	4	0.002696
p49_19_22	4	0.003485
p50_19_25	4	0.001955
p51_20_28	6	0.003125
p52_20_27	5	0.002738

Tabla B.3: Resultados del Algoritmo CELLS sobre el conjunto de instancias Small

Instancias	Cutwidth	Tiempo
p53_20_22	4	0.00208
p54_20_28	6	0.003038
p55_20_24	4	0.002919
p56_20_23	5	0.001827
p57_20_24	4	0.002378
p58_20_21	3	0.003108
p59_20_23	4	0.001901
p60_20_22	4	0.002852
p61_21_22	4	0.002736
p62_21_30	7	0.002186
p63_21_42	12	0.002313
p64_21_22	3	0.002787
p65_21_24	4	0.003658
p66_21_28	6	0.004318
p67_21_22	3	0.004073
p68_21_27	6	0.003256
p69_21_23	5	0.001985
p70_21_25	5	0.002845
p71_22_29	5	0.002281
p72_22_49	14	0.005995
p73_22_29	5	0.003556
p74_22_30	6	0.003056
p75_22_25	5	0.002085
p76_22_30	6	0.002401
p77_22_37	8	0.003509
p78_22_31	6	0.003098

Tabla B.3: Resultados del Algoritmo CELLS sobre el conjunto de instancias Small

Instancias	Cutwidth	Tiempo
p79_22_29	5	0.004013
p80_22_30	5	0.002787
p81_23_46	13	0.003545
p82_23_24	4	0.00234
p83_23_24	4	0.003373
p84_23_26	4	0.003185
p85_23_26	4	0.005636
p86_23_24	3	0.004295
p87_23_30	6	0.0024
p88_23_26	4	0.003051
p89_23_27	5	0.002801
p90_23_35	7	0.002716
p91_24_33	6	0.003868
p92_24_26	4	0.003619
p93_24_27	4	0.00414
p94_24_31	6	0.002855
p95_24_27	4	0.003643
p96_24_27	4	0.002519
p97_24_26	4	0.003753
p98_24_29	5	0.00338
p99_24_27	5	0.002761
p100_24_34	7	0.002857
p25_17_20	4	0.001365
p22_17_19	4	0.001353

APÉNDICE C

RESULTADOS ALGORITMO GRASP_LS2

C.1. Resultados del Algoritmo GRASP_LS2 sobre el conjunto Grids

Tabla C.1: Resultados del Algoritmo GRASP_LS2 sobre el conjunto de instancias Grids

Instancias	Cutwidth	Tiempo
Grid12x12	15	0.096618
Grid12x15	16	0.176039
Grid12x18	15	0.234112
Grid12x21	16	0.35086
Grid12x24	17	0.489753
Grid12x27	15	0.651181
Grid12x3	4	0.001657

Tabla C.1: Resultados del Algoritmo GRASP_LS2 sobre el conjunto de instancias Grids

Instancias	Cutwidth	Tiempo
Grid12x6	9	0.015548
Grid12x9	13	0.034013
Grid15x12	17	0.129709
Grid15x15	21	0.235306
Grid15x18	20	0.404146
Grid15x21	21	0.624445
Grid15x24	20	0.854356
Grid15x27	20	1.23456
Grid15x3	4	0.002502
Grid15x6	9	0.020562
Grid15x9	13	0.054347
Grid18x12	16	0.218399
Grid18x15	23	0.458364
Grid18x18	26	0.686895
Grid18x21	26	1.05285
Grid18x24	25	1.32154
Grid18x27	25	1.99811
Grid18x3	4	0.004241
Grid18x6	9	0.028059
Grid18x9	12	0.115641
Grid21x12	17	0.381066
Grid21x15	21	0.851442
Grid21x18	24	1.27524
Grid21x21	29	1.97391
Grid21x24	31	3.16913
Grid21x27	29	3.75619

Tabla C.1: Resultados del Algoritmo GRASP_LS2 sobre el conjunto de instancias Grids

Instancias	Cutwidth	Tiempo
Grid21x3	4	0.006071
Grid21x6	8	0.039884
Grid21x9	11	0.132489
Grid24x12	16	0.462092
Grid24x15	22	0.882046
Grid24x18	25	1.63635
Grid24x21	30	2.25791
Grid24x24	35	3.75641
Grid24x27	35	5.3309
Grid24x3	4	0.009485
Grid24x6	8	0.061506
Grid24x9	14	0.199149
Grid27x12	16	0.665595
Grid27x15	20	1.26894
Grid27x18	25	2.13381
Grid27x21	30	3.482
Grid27x24	34	5.38584
Grid27x27	38	8.24451
Grid27x3	4	0.011869
Grid27x6	8	0.078313
Grid27x9	12	0.266763
Grid3x12	4	0.00166
Grid3x15	4	0.002368
Grid3x18	4	0.004074
Grid3x21	4	0.005455
Grid3x24	4	0.007525

Tabla C.1: Resultados del Algoritmo GRASP_LS2 sobre el conjunto de instancias Grids

Instancias	Cutwidth	Tiempo
Grid3x27	4	0.011451
Grid3x3	4	0.000194
Grid3x6	4	0.00057
Grid3x9	4	0.001235
Grid6x12	7	0.010373
Grid6x15	7	0.014385
Grid6x18	7	0.027273
Grid6x21	7	0.040471
Grid6x24	7	0.0576
Grid6x27	7	0.076038
Grid6x3	4	0.000443
Grid6x6	7	0.002245
Grid6x9	7	0.006344
Grid9x12	10	0.031042
Grid9x15	12	0.05553
Grid9x18	11	0.088284
Grid9x21	12	0.125177
Grid9x24	11	0.178131
Grid9x27	12	0.262835
Grid9x3	4	0.000703
Grid9x6	7	0.005779

C.2. Resultados del Algoritmo GRASP_LS2 sobre el conjunto HB

Tabla C.2: Resultados del Algoritmo GRASP_LS2 sobre el conjunto de instancias HB

Instancias	Cutwidth	Tiempo
494_bus.mtx.rnd	32	1.82244
662_bus.mtx.rnd	39	4.17305
685_bus.mtx.rnd	45	4.4286
arc130.mtx.rnd	202	0.060048
ash292.mtx.rnd	49	0.430225
ash85.mtx.rnd	18	0.015002
bcspwr01.mtx.rnd	5	0.001672
bcspwr02.mtx.rnd	6	0.003061
bcspwr03.mtx.rnd	10	0.032792
bcspwr04.mtx.rnd	39	0.367576
bcspwr05.mtx.rnd	26	1.26833
bcsstk01.mtx.rnd	34	0.004025
bcsstk02.mtx.rnd	1089	0.012308
bcsstk04.mtx.rnd	310	0.09666
bcsstk05.mtx.rnd	115	0.096301
bcsstk06.mtx.rnd	268	1.69207
bcsstk20.mtx.rnd	30	1.3995
bcsstk22.mtx.rnd	14	0.029352
bcsstm07.mtx.rnd	261	1.659
can__144.mtx.rnd	25	0.063251
can__161.mtx.rnd	54	0.10201
can__292.mtx.rnd	115	0.522146
can__445.mtx.rnd	137	1.75525
curtis54.mtx.rnd	13	0.004571
dwt__209.mtx.rnd	61	0.196587
dwt__221.mtx.rnd	34	0.211555

Tabla C.2: Resultados del Algoritmo GRASP_LS2 sobre el conjunto de instancias HB

Instancias	Cutwidth	Tiempo
dwt__234.mtx.rnd	13	0.02902
dwt__245.mtx.rnd	55	0.258046
dwt__310.mtx.rnd	26	0.49143
dwt__361.mtx.rnd	42	0.840868
dwt__419.mtx.rnd	95	1.2928
dwt__503.mtx.rnd	143	2.35029
dwt__592.mtx.rnd	80	3.20772
fs_183_1.mtx.rnd	190	0.149254
fs_541_1.mtx.rnd	346	3.37652
fs_680_1.mtx.rnd	23	3.85023
gent113.mtx.rnd	97	0.03899
gre_216a.mtx.rnd	52	0.210147
gre__115.mtx.rnd	38	0.037549
gre__185.mtx.rnd	53	0.149252
gre__343.mtx.rnd	72	0.773895
gre__512.mtx.rnd	94	2.38469
hor__131.mtx.rnd	176	1.55497
ibm32.mtx.rnd	24	0.001693
impcol_a.mtx.rnd	48	0.171656
impcol_b.mtx.rnd	56	0.007545
impcol_c.mtx.rnd	50	0.059921
impcol_d.mtx.rnd	76	1.3188
impcol_e.mtx.rnd	178	0.269975
lns__131.mtx.rnd	38	0.040526
lns__511.mtx.rnd	94	2.06996
lund_a.mtx.rnd	113	0.095017

Tabla C.2: Resultados del Algoritmo GRASP_LS2 sobre el conjunto de instancias HB

Instancias	Cutwidth	Tiempo
lund_b.mtx.rnd	114	0.093285
mbeacxc.mtx.rnd	16441	6.25912
mcca.mtx.rnd	390	0.155873
nnc261.mtx.rnd	62	0.379991
nnc666.mtx.rnd	80	4.57627
nos1.mtx.rnd	4	0.081951
nos2.mtx.rnd	4	2.96964
nos4.mtx.rnd	12	0.025899
nos5.mtx.rnd	193	2.13178
nos6.mtx.rnd	29	4.50755
plat362.mtx.rnd	164	1.05412
plskz362.mtx.rnd	34	0.815199
pores_1.mtx.rnd	17	0.001305
pores_3.mtx.rnd	50	1.47365
saylr1.mtx.rnd	19	0.240151
saylr3.mtx.rnd	73	4.92852
sherman4.mtx.rnd	67	2.39542
shl__200.mtx.rnd	429	5.86457
shl__400.mtx.rnd	422	5.78091
shl____0.mtx.rnd	416	5.93151
steam1.mtx.rnd	209	0.341738
steam2.mtx.rnd	372	5.05142
steam3.mtx.rnd	20	0.013914
str__200.mtx.rnd	621	1.19636
str__600.mtx.rnd	656	1.18635
str____0.mtx.rnd	434	1.08975

Tabla C.2: Resultados del Algoritmo GRASP_LS2 sobre el conjunto de instancias HB

Instancias	Cutwidth	Tiempo
west0132.mtx.rnd	78	0.055804
west0156.mtx.rnd	60	0.088676
west0167.mtx.rnd	58	0.115752
west0381.mtx.rnd	491	1.33991
west0479.mtx.rnd	322	2.24363
west0497.mtx.rnd	199	2.31052
west0655.mtx.rnd	500	5.64284
will199.mtx.rnd	144	0.194304
will57.mtx.rnd	13	0.004948

C.3. Resultados del Algoritmo GRASP_LS2 sobre el conjunto Small

Tabla C.3: Resultados del Algoritmo GRASP_LS2 sobre el conjunto de instancias Small

Instancias	Cutwidth	Tiempo
p17_16_24	7	0.000429
p18_16_21	5	0.000303
p19_16_19	4	0.00028
p20_16_18	4	0.000289
p21_17_20	5	0.000262
p23_17_23	5	0.00036
p24_17_29	8	0.000429
p26_17_19	4	0.000266

Tabla C.3: Resultados del Algoritmo GRASP_LS2 sobre el conjunto de instancias Small

Instancias	Cutwidth	Tiempo
p27_17_19	4	0.000247
p28_17_18	3	0.0003
p29_17_18	3	0.000314
p30_17_19	4	0.000371
p31_18_21	3	0.000375
p32_18_20	4	0.000336
p33_18_21	5	0.000358
p34_18_21	4	0.000264
p35_18_19	3	0.000351
p36_18_20	4	0.000449
p37_18_20	5	0.000322
p38_18_19	3	0.000253
p39_18_19	3	0.000293
p40_18_32	8	0.000366
p41_19_20	3	0.000384
p42_19_24	5	0.000421
p43_19_22	5	0.000504
p44_19_25	6	0.000462
p45_19_25	5	0.000366
p46_19_20	3	0.000485
p47_19_21	4	0.00047
p48_19_21	4	0.000343
p49_19_22	4	0.000362
p50_19_25	5	0.00045
p51_20_28	6	0.000456
p52_20_27	6	0.000422

Tabla C.3: Resultados del Algoritmo GRASP_LS2 sobre el conjunto de instancias Small

Instancias	Cutwidth	Tiempo
p53_20_22	4	0.000418
p54_20_28	6	0.000422
p55_20_24	4	0.000462
p56_20_23	5	0.000412
p57_20_24	4	0.000402
p58_20_21	3	0.000468
p59_20_23	4	0.000427
p60_20_22	4	0.000436
p61_21_22	4	0.000363
p62_21_30	7	0.000513
p63_21_42	12	0.000532
p64_21_22	4	0.000368
p65_21_24	4	0.000477
p66_21_28	6	0.000507
p67_21_22	3	0.000532
p68_21_27	6	0.000502
p69_21_23	5	0.000591
p70_21_25	5	0.000483
p71_22_29	5	0.000518
p72_22_49	15	0.00075
p73_22_29	5	0.000605
p74_22_30	6	0.000434
p75_22_25	5	0.000538
p76_22_30	7	0.000508
p77_22_37	8	0.001568
p78_22_31	6	0.000552

Tabla C.3: Resultados del Algoritmo GRASP_LS2 sobre el conjunto de instancias Small

Instancias	Cutwidth	Tiempo
p79_22_29	5	0.000567
p80_22_30	5	0.000595
p81_23_46	13	0.000766
p82_23_24	5	0.000544
p83_23_24	4	0.0005
p84_23_26	5	0.00067
p85_23_26	4	0.000444
p86_23_24	3	0.000417
p87_23_30	6	0.000521
p88_23_26	4	0.000473
p89_23_27	5	0.000537
p90_23_35	7	0.000558
p91_24_33	6	0.00066
p92_24_26	5	0.00061
p93_24_27	4	0.000669
p94_24_31	6	0.000931
p95_24_27	5	0.000583
p96_24_27	4	0.000546
p97_24_26	5	0.000608
p98_24_29	6	0.000636
p99_24_27	6	0.000524
p100_24_34	7	0.00068
p25_17_20	4	0.000397
p22_17_19	4	0.000369

APÉNDICE D

VALORES ÓPTIMOS Y MEJORES VALORES CONOCIDOS DE LOS CONJUNTOS DE INSTANCIAS

D.1. Valores Óptimos del Conjunto Grids

Tabla D.1: Valores óptimos del conjunto de instancias Grids

Instancias	Cutwidth
Grid12x12	13
Grid12x15	13
Grid12x18	13
Grid12x21	13
Grid12x24	13
Grid12x27	13
Grid12x3	4

Tabla D.1: Valores óptimos del conjunto de instancias Grids

Instancias	Cutwidth
Grid12x6	7
Grid12x9	10
Grid15x12	13
Grid15x15	16
Grid15x18	16
Grid15x21	16
Grid15x24	16
Grid15x27	16
Grid15x3	4
Grid15x6	7
Grid15x9	10
Grid18x12	13
Grid18x15	16
Grid18x18	19
Grid18x21	19
Grid18x24	19
Grid18x27	19
Grid18x3	4
Grid18x6	7
Grid18x9	10
Grid21x12	13
Grid21x15	16
Grid21x18	19
Grid21x21	22
Grid21x24	22
Grid21x27	22
Grid21x3	4

Tabla D.1: Valores óptimos del conjunto de instancias Grids

Instancias	Cutwidth
Grid21x6	7
Grid21x9	10
Grid24x12	13
Grid24x15	16
Grid24x18	19
Grid24x21	22
Grid24x24	25
Grid24x27	25
Grid24x3	4
Grid24x6	7
Grid24x9	10
Grid27x12	13
Grid27x15	16
Grid27x18	19
Grid27x21	22
Grid27x24	25
Grid27x27	28
Grid27x3	4
Grid27x6	7
Grid27x9	10
Grid3x12	4
Grid3x15	4
Grid3x18	4
Grid3x21	4
Grid3x24	4
Grid3x27	4
Grid3x3	4

Tabla D.1: Valores óptimos del conjunto de instancias Grids

Instancias	Cutwidth
Grid3x6	4
Grid3x9	4
Grid6x12	7
Grid6x15	7
Grid6x18	7
Grid6x21	7
Grid6x24	7
Grid6x27	7
Grid6x3	4
Grid6x6	7
Grid6x9	7
Grid9x12	10
Grid9x15	10
Grid9x18	10
Grid9x21	10
Grid9x24	10
Grid9x27	10
Grid9x3	4
Grid9x6	7

D.2. Mejores Valores Conocidos del conjunto HB

Tabla D.2: Mejores valores conocidos del conjunto de instancias HB

Instancias	Cutwidth
494_bus.mtx.rnd	17
662_bus.mtx.rnd	27
685_bus.mtx.rnd	35
arc130.mtx.rnd	202
ash292.mtx.rnd	36
ash85.mtx.rnd	16
bcspwr01.mtx.rnd	5
bcspwr02.mtx.rnd	5
bcspwr03.mtx.rnd	10
bcspwr04.mtx.rnd	29
bcspwr05.mtx.rnd	18
bcsstk01.mtx.rnd	32
bcsstk02.mtx.rnd	1089
bcsstk04.mtx.rnd	310
bcsstk05.mtx.rnd	115
bcsstk06.mtx.rnd	227
bcsstk20.mtx.rnd	20
bcsstk22.mtx.rnd	13
bcsstm07.mtx.rnd	199
can__144.mtx.rnd	25
can__161.mtx.rnd	50
can__292.mtx.rnd	96
can__445.mtx.rnd	123
curtis54.mtx.rnd	13
dwt__209.mtx.rnd	58
dwt__221.mtx.rnd	27
dwt__234.mtx.rnd	12

Tabla D.2: Mejores valores conocidos del conjunto de instancias HB

Instancias	Cutwidth
dwt__245.mtx.rnd	27
dwt__310.mtx.rnd	26
dwt__361.mtx.rnd	39
dwt__419.mtx.rnd	55
dwt__503.mtx.rnd	138
dwt__592.mtx.rnd	70
fs_183_1.mtx.rnd	185
fs_541_1.mtx.rnd	296
fs_680_1.mtx.rnd	17
gent113.mtx.rnd	87
gre_216a.mtx.rnd	52
gre__115.mtx.rnd	36
gre__185.mtx.rnd	48
gre__343.mtx.rnd	72
gre__512.mtx.rnd	94
hor__131.mtx.rnd	145
ibm32.mtx.rnd	23
impcol_a.mtx.rnd	47
impcol_b.mtx.rnd	55
impcol_c.mtx.rnd	46
impcol_d.mtx.rnd	64
impcol_e.mtx.rnd	170
lns__131.mtx.rnd	30
lns__511.mtx.rnd	71
lund_a.mtx.rnd	113
lund_b.mtx.rnd	111
mbeacxc.mtx.rnd	16329

Tabla D.2: Mejores valores conocidos del conjunto de instancias HB

Instancias	Cutwidth
mcca.mtx.rnd	390
nnc261.mtx.rnd	46
nnc666.mtx.rnd	80
nos1.mtx.rnd	4
nos2.mtx.rnd	4
nos4.mtx.rnd	12
nos5.mtx.rnd	193
nos6.mtx.rnd	29
plat362.mtx.rnd	155
plskz362.mtx.rnd	30
pores_1.mtx.rnd	17
pores_3.mtx.rnd	29
saylr1.mtx.rnd	16
saylr3.mtx.rnd	45
sherman4.mtx.rnd	36
shl__200.mtx.rnd	387
shl__400.mtx.rnd	385
shl____0.mtx.rnd	357
steam1.mtx.rnd	182
steam2.mtx.rnd	308
steam3.mtx.rnd	20
str__200.mtx.rnd	560
str__600.mtx.rnd	606
str____0.mtx.rnd	388
west0132.mtx.rnd	71
west0156.mtx.rnd	56
west0167.mtx.rnd	54

Tabla D.2: Mejores valores conocidos del conjunto de instancias HB

Instancias	Cutwidth
west0381.mtx.rnd	480
west0479.mtx.rnd	287
west0497.mtx.rnd	181
west0655.mtx.rnd	466
will199.mtx.rnd	131
will57.mtx.rnd	11

D.3. Valores Óptimos del conjunto Small

Tabla D.3: Valores óptimos del conjunto de instancias Small

Instancias	Cutwidth
p17_16_24	7
p18_16_21	5
p19_16_19	4
p20_16_18	4
p21_17_20	4
p23_17_23	5
p24_17_29	8
p26_17_19	4
p27_17_19	4
p28_17_18	3
p29_17_18	3
p30_17_19	4
p31_18_21	3
p32_18_20	4

Tabla D.3: Valores óptimos del conjunto de instancias Small

Instancias	Cutwidth
p33_18_21	4
p34_18_21	4
p35_18_19	3
p36_18_20	4
p37_18_20	4
p38_18_19	3
p39_18_19	3
p40_18_32	8
p41_19_20	3
p42_19_24	5
p43_19_22	4
p44_19_25	6
p45_19_25	5
p46_19_20	3
p47_19_21	4
p48_19_21	4
p49_19_22	4
p50_19_25	4
p51_20_28	6
p52_20_27	5
p53_20_22	4
p54_20_28	6
p55_20_24	4
p56_20_23	5
p57_20_24	4
p58_20_21	3
p59_20_23	4

Tabla D.3: Valores óptimos del conjunto de instancias Small

Instancias	Cutwidth
p60_20_22	4
p61_21_22	4
p62_21_30	7
p63_21_42	12
p64_21_22	3
p65_21_24	4
p66_21_28	6
p67_21_22	3
p68_21_27	6
p69_21_23	5
p70_21_25	5
p71_22_29	5
p72_22_49	14
p73_22_29	5
p74_22_30	6
p75_22_25	5
p76_22_30	6
p77_22_37	8
p78_22_31	6
p79_22_29	5
p80_22_30	5
p81_23_46	13
p82_23_24	4
p83_23_24	4
p84_23_26	4
p85_23_26	4
p86_23_24	3

Tabla D.3: Valores óptimos del conjunto de instancias Small

Instancias	Cutwidth
p87_23_30	6
p88_23_26	4
p89_23_27	5
p90_23_35	7
p91_24_33	6
p92_24_26	4
p93_24_27	4
p94_24_31	6
p95_24_27	4
p96_24_27	4
p97_24_26	4
p98_24_29	5
p99_24_27	5
p100_24_34	7
p25_17_20	4
p22_17_19	4