INSTITUTO TECNOLÓGICO DE CD. MADERO

División de Estudios de Posgrado e Investigación

TESIS

Estrategias de búsqueda local para el problema de SUMCUT

PARA OBTENER EL GRADO DE:

Maestro en Ciencias en Ciencias de la Computación

Presenta:

I.S.T.I. Daniel Enrique Zamarrón Escobar

Directores de Tesis:

Dr. Héctor Joaquín Fraire Huacuja

M.C. Jesús David Terán Villanueva

Cd. Madero, Tamaulipas Junio 2013

"2013, Año de la Lealtad Institucional y Centenario del Ejército Mexicano"

Cd. Madero, Tamps; a 16 de Mayo de 2013.

OFICIO No.: U5.142/13
AREA: DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN
ASUNTO: AUTORIZACIÓN DE IMPRESIÓN DE TESIS

C. ING. DANIEL ENRIQUE ZAMARRÓN ESCOBAR
PRESENTE

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 DE SUMCUT"

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.

ATENTAMENTE
"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 'NPCO ' jar

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 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.

(*Ing. Daniel Enrique Zamarrón Escobar*)

Resumen

En este trabajo de investigación se aborda el problema de diseñar estrategias eficientes de búsqueda local para el problema del SUMCUT. El SUMCUT es un problema NP-duro que consiste en minimizar la suma de los cortes de un grafo conexo; dicho problema tiene aplicaciones en la genética, en la arqueología y en la reducción de matrices dispersas. La contribución más importante de este trabajo consiste en cinco nuevas estrategias de búsqueda local, la cuales incorporan mecanismos eficientes para la predicción del costo de una inserción consecutiva los cuales reducen la complejidad de las búsquedas locales de $O(n^3)$ a $O(n^2)$. Para validar la eficiencia de las estrategias propuestas se desarrolló una solución metaheurística del problema SUMCUT, basada en una búsqueda local iterada con procesamiento celular. Se realizaron pruebas experimentales con instancias estándar, los cuales mostraron que para las instancias más grandes que las búsquedas locales propuestas, reducen el tiempo de ejecución del algoritmo al menos un 90 % alcanzando el mismo nivel de calidad.

Resultados parciales de este trabajo se incorporaron en el artículo "Estrategias de búsqueda local para el problema de SUMCUT", el cual fue publicado en el *VI Encuentro de Investigadores de Posgrado en el área de Ciencias Computacionales* realizado en el Instituto Tecnológico de Ciudad Madero los días del 10 al 14 de diciembre del 2012.

Índice general

Índice de Tablas

Índice de Figuras

Índice de Algoritmos

Capítulo 1

Introducción

En este trabajo de investigación se aborda el problema de la suma de cortes (SUMCUT). El problema del SUMCUT es un problema de permutaciones en el cual se busca un ordenamiento tal que produzca la menor suma de todos los cortes de un grafo. Este problema posee varias aplicaciones, por ejemplo en la arqueología [17] donde los hallazgos encontrados en una Pirámide o Centro Ceremonial por citar a algunos; son organizados en un orden cronológico determinado a esto se le llama seriación. También se utiliza para minimizar el espacio de una matriz dispersa [22].

Otra aplicación está enfocada hacia la genética [16], donde hoy en día los avances en medicina y ciencia son gracias al estudio del ADN (ácido desoxirribonucleico) del ser humano. El ADN está compuesto por un grupo de nucleótidos que pueden ser adenina, timina, citosina o guanina. Con el fin de reproducir y estudiar un determinado tramo del ADN, se extrae una muestra de un espécimen y se inserta dicha muestra en otros organismos vivientes que actuarán como huéspedes, con la finalidad de que dicho huésped preserve y reproduzca la muestra de ADN del espécimen como si fueran propias del huésped. A este proceso se le llama clonación y a los fragmentos preservados de la muestra se les llama *clones*.

En el proceso de clonación, la información relativa a la posición de los clones en la muestra original se pierde y mucha de la información de la muestra original de

los huéspedes está repetida. En este contexto el problema consiste en determinar la posición de la muestra original de ADN y de los clones basándose únicamente en información redundante.

Afortunadamente los clones poseen información parcial de su contenido, a esto se le llama *huella digital*. Cuando dos clones poseen huellas digitales muy parecidas, podemos decir que hay una intersección entre ellas. A partir de todos los clones obtenidos, se busca un arreglo de clones con la mínima cantidad de traslapes que reconstruya la muestra original de ADN.

1.1. Definición formal del problema

La definición formal de SUMCUT utilizada en este trabajo es la que se describe en el artículo de Petit [27]. Sea $G = (V, E)$ un grafo conexo no dirigido con $n = |V|$ y $m = |E|$. Un ordenamiento lineal o permutación de los vértices G es una función $\varphi : V \to 1, 2, \ldots, n$, la cual asocia cada vértice del grafo con una única etiqueta. El conjunto de todos los ordenamientos lineales (soluciones) definidos sobre G se representa por $\Phi(G)$. El conjunto $L(i, \varphi, G) = \{u \in V \mid \varphi(u) \leq i\}$ contiene todos los vértices que quedan a la izquierda del punto de corte $i = 1, 2, \ldots n$. El conjunto $R(i, \varphi, G) = \{u \in V \mid \varphi(u) > i\}$ contiene todos los vértices que quedan a la derecha del punto de corte i. El conjunto $\delta(i, \varphi, G) = \{u \in L(i, \varphi, G) \mid \exists v \in R(i, \varphi, G) : (u, v) \in E(G)\}|$ contiene todos los vértices del conjunto izquierdo $L(i, \varphi, G)$ que tienen al menos un vértice adyacente en el conjunto derecho $R(i, \varphi, G)$. El corte del vértice que se ubica en la posición i en el ordenamiento φ se define como $|\delta(i, \varphi, G)|$.

El valor $SUMCUT(\varphi, G)$, es igual a la suma de los cortes de todos los vértices del ordenamiento φ:

$$SUMCUT(\varphi, G) = \sum_{i=1}^{n} |\delta(i, \varphi, G)| \tag{1.1}$$

El problema de minimización de SUMCUT consiste en minimizar el valor de $SUMCUT(\varphi, G)$ sobre todos los ordenamientos lineales $\varphi \in \phi(G)$:

$$SUMCUT(G) = min_{\varphi \in \phi(G)} SUMCUT(\varphi, G). \tag{1.2}$$

El problema del SUMCUT en [18] es equivalente al problema de minimización Profile, donde el resultado de dicho problema es la solución inversa del problema del SUMCUT. Los autores mencionan que el problema Profile así como el SUMCUT son NP-Completos [7].

1.2. Calculo de la función objetivo del SUMCUT

Para calcular el valor objetivo de un ordenamiento lineal de los vértices del grafo $G = (V, E)$ se representa con un arreglo lineal $\pi = [\pi_1, \pi_2, \ldots, \pi_n]$ y se utiliza la matriz de adyacencia del grafo $(a_{i\,j})$. Con estas estructuras de datos, el Cutwidth (CW) para un vértice (π_k), del ordenamiento dado, se calcula de la siguiente manera:

$$CW(\pi_k) = \sum_{i=1}^{k} \sum_{j=k+1}^{n} a_{\pi_i \pi_j} \tag{1.3}$$

donde:

$$a_{\pi_i \pi_j} = \begin{cases} 1, & \text{si la arista } (\pi_i, \pi_j) \in E \\ 0, & \text{en caso contrario} \end{cases}$$

El SUMCUT de la permutación π se calcula:

$$SC(\pi) = \sum_{k=1}^{n} CW(\pi_{\mathrm{k}}) \tag{1.4}$$

Así, el problema de SUMCUT se define como:

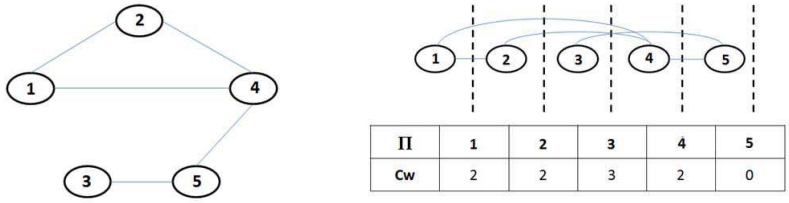
$$SC(G) = SC(\pi_{min}) \tag{1.5}$$

donde:

$$\pi_{min} = \text{argmin}_{\pi \in P}\{SC(\pi)\} \tag{1.6}$$

A continuación se presenta un ejemplo para calcular el SUMCUT.

Dado un grafo G como el mostrado la Figura 1.1(a) y una permutación $\pi = (1, 2, 3, 4, 5, 6)$, el Cutwidth de un vértice π_k se puede calcular como el número de aristas que salen del mismo vértice o de uno anterior hacia la derecha, esto se puede destacar si se dibuja una línea imaginaria entre el vértice π_k y π_{k+1}. La Figura 1.1(b) muestra los valores de Cutwidth para la permutación π sobre el grafo G.



Π	1	2	3	4	5
Cw	2	2	3	2	0

a) Grafo no dirigido b) Ancho de corte del grafo no dirigido

Figura 1.1: Ejemplo del ancho de corte

El Cutwidth del vértice 3, es igual a 3 debido a que el número de aristas que surgen del nodo hacia la derecha es 1 y el número de aristas que pasan por este son 2, como se muestra a continuación.

$$\begin{aligned} CW(3) &= \{\text{Número de aristas que pasan por el vértice}\} \\ &\quad +\{\text{Número de aristas que surgen de él hacia la derecha}\} \\ &= (\pi_1, \pi_4), (\pi_2, \pi_4) + (\pi_3, \pi_5) \\ &= 2 + 1 \\ &= 3 \end{aligned}$$

Los Cutwidths restantes del grafo G se calculan de la misma manera y sus resultados se muestran a continuación:

$$\begin{aligned} CW(1) &= 2 \\ CW(2) &= 2 \\ CW(4) &= 2 \\ CW(5) &= 0 \end{aligned}$$

Una vez determinados todos los Cutwidths del grafo G para la permutación π, el valor del SUMCUT para dicha permutación es igual a la suma de todos los Cutwidths en los diferentes puntos de corte. Para el ejemplo anterior el valor del SUMCUT se calcula de la siguiente manera:

$$\begin{aligned} SC(\pi) &= \sum_{i=1}^{5} CW(i) \\ &= 2+2+3+2+0 \\ &= 9 \end{aligned}$$

1.3. Objetivos del proyecto

1.3.1. Objetivo general

Diseñar e implementar diversas estrategias de búsqueda local para el problema de SUMCUT.

1.3.2. Objetivos específicos

- Diseñar e implementar el cálculo de la función objetivo del SUMCUT.
- Diseñar e implementar la función de inserción con actualización de índices.
- Diseñar e implementar una nueva función de predicción de costos de la inserción consecutiva.

- Diseñar e implementar las diversas estrategias de búsqueda local.
- Evaluar las diversas estrategias de búsqueda local producidas.

1.3.3. Alcances y limitantes

- Solamente se considerarán algoritmos secuenciales.
- Se utilizará lenguaje C++.
- Las instancias a emplear son Small, Grid y Harwell-Boeing.
- El problema de ordenamiento lineal que se abordará es el SUMCUT.

1.4. Organización de la tesis

El siguiente documento de tesis está estructurado de la siguiente manera:

- **Capítulo 2: Marco teórico.** En este capítulo se presentan los fundamentos teóricos que sustentan la investigación de este proyecto.
- **Capítulo 3: Estado del arte.** Esta sección muestra los antecedentes del problema del SUMCUT, así como el estado del arte de las búsquedas locales.
- **Capítulo 4: Enfoque de solución.** En este capítulo se proponen algoritmos y métodos para solucionar el problema del SUMCUT.
- **Capítulo 5: Resultados experimentales.** Esta sección muestra los resultados obtenidos por las estrategias propuestas en el capítulo anterior, la metodología propuesta y la descripción de los experimentos realizados.
- **Capítulo 6: Conclusiones.** Este apartado contiene lo más relevante de la investigación realizada en este trabajo.
- **Apéndice A: Mejores conocidos.** Este apéndice contiene todos los mejores resultados obtenidos en este trabajo.

Capítulo 2

Marco teórico

En este capítulo se hablará sobre conceptos empleados en este trabajo de investigación.

2.1. Problemas de decisión y optimización

Un problema de decisión es el que produce las respuestas: "Sí" o "No". Un problema de decisión es una pareja $\Pi = (D, Y)$ donde D es el conjunto de todas las instancias del problema y Y es el subconjunto de instancias Sí ($Y \subset D$) [10]. Un problema de optimización es un problema en donde se trata de maximizar o minimizar algun atributo numérico [10].

2.2. Complejidad computacional

La teoría de la complejidad computacional es una rama de la teoría de la computación, que estudia la cantidad de recursos necesarios para resolver los problemas de decisión. Los problemas se clasifican en las siguientes clases: P, NP, NP-completo y NP-duro [10].

- Clase P:

  Conjunto de todos los problemas de decisión que se pueden resolver con un algoritmo determinista en un tiempo polinomial. A estos problemas se les considera tratables [10].

- Clase NP:

  Conjunto de todos los problemas de decisión que se pueden resolver con un algoritmo no determinista en tiempo polinomial [10]. Resolver significa que se puede generar una solución candidata del problema con un algoritmo no determinista en tiempo polinomial y que esta solución puede ser verificada en tiempo polinomial con un algoritmo determinista. Como todo algoritmo determinista se puede considerar como un caso particular de un algoritmo no determinista, entonces $P \subseteq NP$.

- Clase NP-completo:

  Un problema de decisión se dice que es NP-completo si y solo si, pertenece a la clase NP y todo problema de NP se puede transformar a este problema [10]. Para probar que un problema de decisión π es NP-completo, se debe probar que $\pi \in NP$ y que existe $\pi^* \in NPC$ tal que π^* se transforma polinomialmente a π.

- Clase NP-duro:

  Un problema de optimización es NP-duro si su versión de decisión es NP-completo [10].

2.3. Métodos de optimización

2.3.1. Métodos exactos: ramificación y acotamiento

El algoritmo de ramificación y acotamiento (Branch and Bound en inglés) consiste en representar el problema mediante una estructura tipo árbol. Cada rama del árbol corresponde a una solución del problema, para generar las diversas soluciones el árbol se recorre en profundidad. Fue introducido por Land y Doig en 1960 [21].

Una ventaja del algoritmo de ramificación y acotamiento, es que en cada nodo del árbol se puede predecir si es viable o no continuar explorando dicho nodo.

En caso que no se pueda explorar se puede emplear un mecanismo de poda, el cual consiste en no explorar esa rama restante del nodo actual y regresar al nodo anterior para visitar las ramas no exploradas.

Una de las estrategias para acotar una rama es mediante el uso de la cota superior y cota inferior. La cota superior empieza con un valor muy grande o chico dependiendo si es un problema de maximización o minimización, y se actualiza cuando una solución generada es mejor que la cota anterior. La cota inferior es un valor que se empieza a formar mediante el recorrido del árbol. Si antes de tomar todos los nodos, el valor de la cota inferior supera al valor de la cota superior, se marca esa rama y ya no se sigue explorando.

2.3.2. Métodos aproximados

Este tipo de metodología de solución proporciona soluciones con una cercanía al óptimo determinada. En casos prácticos, la cercanía a valores óptimos es usualmente pequeña [4].

Se dice que un algoritmo para un problema dado tiene un radio aproximado de $\rho(n)$, si para alguna instancia de tamaño n, el costo C de la solución obtenida por el algoritmo está dentro de un factor de $\rho(n)$ del costo C^* de la solución óptima [4]:

$$\text{máx}(C/C^*, C^*/C) \leq \rho(n)$$

Se denomina algoritmo $\rho(n)$-aproximado a un algoritmo que logra un radio aproximado de $\rho(n)$. Las definiciones de radio aproximado de $\rho(n)$ aplican para problemas de maximización y de minimización. Para un problema de maximización, $0 < C \leq C^*$, y el radio C^*/C da el factor por el que el costo de una solución óptima es más grande que el costo de una solución aproximada. Similarmente, para un problema de minimización, $0 < C^* \leq C$, y el radio C/C^* da el factor por el que el costo de la solución aproximada es mayor que el costo de la solución óptima. Dado que se supone que todas las soluciones tienen un costo positivo,

estos radios están siempre bien definidos [4].

2.3.3. Métodos heurísticos y metaheurísticos

Los algoritmos heurísticos utilizan conocimiento del problema para encontrar soluciones aproximadas, de buena calidad, en un tiempo razonable [8][2]. Estos algoritmos se usan para resolver instancias grandes para las cuales los algoritmos exactos toman demasiado tiempo.

Una desventaja de los algoritmos heurísticos es que no cuentan con métodos para escapar de los óptimos locales. Los métodos heurísticos se clasifican en: métodos constructivos y métodos de búsqueda local [8]. Los primeros construyen una solución aplicando una estrategia determinada. Comúnmente la estrategia utilizada es voraz, de descomposición o de reducción. Los métodos de búsqueda local parten de una solución dada y tratan de mejorarla sustituyéndola por una solución cercana cuyo valor objetivo mejore al de la solución actual [8]. Para una solución dada, se tiene que definir de manera precisa cuales son las soluciones candidatas cercanas a dicha solución. Éstas soluciones constituyen una vecindad de la solución dada.

Una metaheurística puede ser vista como un marco de trabajo para un algoritmo general, el cual puede ser aplicado a diferentes problemas de optimización, realizando ligeros cambios en el mismo para adaptarse a otros problemas [23][2][8][25].

A continuación se mencionan algunas de las características principales de las metaheurísticas [8]:

- Permite que los algoritmos heurísticos escapen de óptimos locales.
- Son procesos que guían el proceso de búsqueda.
- Su objetivo es explorar eficientemente un espacio de búsqueda para encontrar soluciones cercanas al óptimo de manera iterativa.

- Son aproximados y usualmente son algoritmos no deterministas.
- No son para un problema específico.
- Pueden hacer uso de conocimiento específico del problema en forma de heurísticas que son controladas por estrategias de mayor nivel.

Búsqueda local iterada

La búsqueda local iterada (ILS por sus siglas en inglés) es una metaheurística trayectorial simple, fácil de implementar y altamente efectiva [11][23][8]. La esencia principal de esta metaheurística es la búsqueda local, por lo que el desempeño depende mucho de dicha búsqueda.

Algoritmo 1 Algoritmo de búsqueda local iterada

1: $s_0 = GenerarSolucionInicial$
2: $s^* = BusquedaLocal(s_0)$
3: **repeat**
4: $\quad s^{'} = Perturbacion(s^*)$
5: $\quad s^{*'} = BusquedaLocal(s^{'})$
6: $\quad s^* = CriterioDeAceptacion(s^*, s^{*'}, historial)$
7: **until** $CondicionDeParo$

Primero se genera una solución inicial, la cual puede ser generada a través de algún método de construcción o de manera aleatoria. Una vez generada se aplica la búsqueda local a la solución inicial para mejorar la calidad de la misma. Después se realiza un ciclo que lo constituyen una perturbación, una búsqueda local y un criterio de aceptación. La perturbación nos ayuda a mover la solución a otro lugar dentro del espacio de soluciones. Si se perturba mucho una solución podría diversificar demasiado la solución, por el contrario si la perturbación no es significativa corre el riesgo que la búsqueda local se estanque en un óptimo local fácilmente. Al finalizar de realizar la perturbación se aplica la misma búsqueda local empleada para mejorar nuestra solución inicial. El criterio de aceptación permite decidir si tomar la solución antes de empezar el ciclo o tomar la solución después del ciclo. El criterio más común es elegir la solución que tiene mejor calidad. El ciclo se repite según algún criterio de paro, este puede ser un número fijo de iteraciones, o un número de iteraciones sin mejora, entre otros.

2.4. Búsqueda local

Estos algoritmos inician a partir de una solución inicial e iterativamente tratan de reemplazar la solución actual por una mejor solución en un vecindario de la solución inicial, apropiadamente definido [2].

Michalewicz en su trabajo propone que una búsqueda local está compuesta por los siguientes puntos [25]:

1. Seleccionar una solución del espacio de búsqueda y evaluarla, definir la anterior solución como la solución actual.

2. Generar una nueva solución a partir de la solución actual y evaluarla.

3. Si la nueva solución es mejor que la solución actual, entonces se define como solución actual, en caso contrario se descarta.

4. Repetir el paso 2 y 3 hasta no poder generar una solución en el espacio de soluciones que mejore la solución actual.

2.5. Diseño de las vecindades

2.5.1. Vecindad de inserción

Sea una permutación π, π' puede ser obtenida por un movimiento de inserción que se define como:

$$InsertMove(\pi, i, j) = \begin{cases} (\pi_1, ..., \pi_{i-1}, \pi_{i+1}, ..., \pi_{j-1}, \pi_j, \pi_i, \pi_{j+1}, ..., \pi_n) \text{ para } i < j \\ (\pi_1, ..., \pi_{j-1}, \pi_i, \pi_j, \pi_{j+1}, ..., \pi_{i-1}, \pi_{i+1}, ..., \pi_n) \text{ para } i > j \end{cases}$$

Un movimiento de inserción consiste en eliminar el elemento en la posición i y luego insertarlo en la posición j. Los elementos entre i y j son desplazados, para la derecha si $i > j$ o para la izquierda si $i < j$ [20].

2.5.2. Vecindad de inserción consecutiva

Un movimiento de inserción consecutiva se define como un $InsertMove(\pi, i, i+1)$ o $InsertMove(\pi, i, i-1)$ [20].

El costo del movimiento de inserción $InsertMove(\pi, i, j)$ como lo menciona [20] se define como:

$$Cost(InsertMove(\pi, i, j)) = SC(\pi') - SC(\pi)$$

El vecindario de inserción $N(\pi)$ es el conjunto de todas las permutaciones obtenidas por un solo movimiento de inserción sobre π. El espacio de búsqueda de este vecindario es de tamaño $(n-1)^2$ [3] [20].

2.6. Algoritmos de procesamiento celular

Un algoritmo de procesamiento celular simula la ejecución paralela de varias *celdas de procesamiento* (CP). Cada CP realiza una búsqueda heurística en una determinada región del espacio de soluciones. Las CP-s pueden utilizar las misma o diferente estrategia de búsqueda y se ejecutan siempre que continúen mejorando la solución local. El Algoritmo 2 muestra los principales elementos de un algoritmo de procesamiento celular [32].

Algoritmo 2 Enfoque de procesamiento celular

```
1: InicializarCP(TotalCeldas)
2: repeat
3:   repeat
4:     for i = 1 → TotalCeldas do
5:       if !EstaEstancada(CP_i) then
6:         EjecutarCP(CPi)
7:       end if
8:     end for
9:   until CeldasEstancadas()
10:  ComunicacionDeCP()
11:  ReestructurarCP()
12: until CondicionDeParo
```

$InicializarCP$ inicializa las CP desde 1 hasta el $TotalCeldas$. Las CP pueden tener una o múltiples soluciones dependiendo de la metaheurística. Las soluciones

generadas en cada CP pueden ser aleatorias o creadas por algún método de construcción.

Ahora para cada una de las CP se ejecutan sus respectivas heurísticas mientras dicha CP no esté estancada. Se considera que una CP está estancada cuando ejecuta un número determinado de iteraciones sin producir una mejora local.

Una vez estancadas todas las CP, se ejecuta el proceso de $InteraccionDeCP()$. El propósito de este procedimiento es utilizar la información generada por cada una de las celdas para producir soluciones de mejor calidad. Para el problema del SUMCUT, la mejor solución generada por cada una de las celdas se cruza con la mejor solución de las otras celdas. En el proceso se consideran todas las combinaciones posibles entre las celdas disponibles. Si el hijo resultante de una cruza es mejor que el peor de los padres utilizados, se reemplaza dicho padre.

$ReestructurarCP()$ establece que todas las CP están no estancadas y busca la o las mejores soluciones que se hayan encontrado en todas las CP. Todo el procesamiento celular se repetirá hasta que encuentre una condición de paro: ya sea no encontrar mejoras en las celdas en un tiempo determinado u otro criterio.

Capítulo 3

Estado del arte

En esta sección se hablará sobre los antecedentes del problema de SUMCUT y el estado del arte de las búsquedas locales.

3.1. Antecedentes del problema del SUMCUT

3.1.1. Métodos heurísticos

Crespelle [5] propone realizar un grafo de intersección H utilizando aristas del grafo G, para reducir el tiempo computacional de matrices dispersas, para el problema del intervalo mínimo. Esto se demuestra a base de teoremas y demostraciones matemáticas. Utiliza una estructura de árbol PQ (PQ-Tree) para marcar los elementos de una permutación, donde las ramas de Q sólo tienen dos elementos intercambiables entre sí, y las ramas de P son los elementos no contenidos en las ramas Q que pueden permutar. A partir de la estructura se dividen los elementos de la permutación en dos conjuntos, izquierda y derecha, donde para cada uno identifica si el nodo es degenerado o primario para obtener un buen ordenamiento.

Cuthill y McKee [6] proponen el algoritmo que lleva su nombre (Algoritmo Cuthill-McKee) para encontrar el $Profile$ mínimo de un grafo, esto lo realiza reordenando la matriz para alcanzar dicho valor. A partir de este algoritmo se le hizo una mejora por George [12], la cual invierte la solución obtenida por el algoritmo

Cuthill-McKee, llamando su algoritmo RCM (Algoritmo Reverse Cuthill-McKee). Liu y Sherman [34] realizaron un análisis entre ambos algoritmos, donde demuestran que ambos son equivalentes y que el algoritmo RCM es al menos tan bueno como el original.

Gibbs en su artículo [13], analiza el algoritmo RCM propuesto por George [12] mencionando las debilidades del mismo. A partir de estas debilidades propone el algoritmo Gibbs-King, el cual puede ser empleado para el problema de minimización de $Bandwidth$ o para el problema $Profile$. El algoritmo produce valores de la misma calidad que el algoritmo RCM pero utilizando menos tiempo para resolverlo.

Kenndall [17] trabaja en el contexto de la aplicación de matrices y grafos para la serialización de objetos distintos (como fósiles, jarrones, joyería, entre otros). El término serializar es conocido en arqueología y consiste en ordenar cronológicamente diversos artefactos de múltiples eras.

Heggernes [15] presenta un algoritmo de complejidad $O(k^{2k}n^3m)$, donde dado un grafo $G = (V, E)$ se puede obtener un grafo de intervalo agregando cuando mucho k aristas, su aportación está enfocada al mapeo de una cadena de ADN y la minimización del valor $Profile$ de una matriz dispersa. Se aplican tres reglas con una complejidad $O(n^3m)$, y al final se emplea un algoritmo voraz que intentará identificar los vértices que posean el valor mínimo de separación entre los demás vértices.

3.1.2. Soluciones metaheurísticas

La tesis de Lewis [22] desarrolla la metaheurística de Simulated Annealing (Recocido Simulado) mediante movimientos de intercambios, para el problema $Profile$. Antes de empezar el Recocido Simulado la matriz es reordenada con el algoritmo Cuthill-McKee o Gibbs-King, con el que se marcan los vértices de la

matriz. El movimiento de intercambio consiste en tomar dos posiciones, i y j e intercambiar las filas i y las columnas j de la matriz, recalculando todo el costo del Profile, enfatizando que si la distancia entre i y j es muy grande puede tomar bastante tiempo computacional en realizar los movimientos de intercambio.

Como se muestra en el Algoritmo 3, éste inicia generando una solución inicial y estableciendo una temperatura inicial. Hasta que se realicen un número definido de iteraciones, se va a realizar un movimiento aleatorio de intercambio a la solución inicial y se calcula la diferencia de ambas soluciones. Si la diferencia es menor a 0 o si el valor estadístico de Boltzmann es mayor a un valor aleatorio entre 0 y 1, entonces la nueva solución reemplaza a la solución inicial. Después de realizar las iteraciones se baja la temperatura inicial un porcentaje establecido hasta que llegue a un límite de temperatura. Este algoritmo se considera actualmente la mejor solución del estado del arte para el SUMCUT [31].

Algoritmo 3 Recocido simulado por Lewis

```
1: solInicial = generarSolucionInicial()
2: tempInicial = tempInicial()
3: repeat
4:   repeat
5:     solMov = LSFirstAleatoria(solInicial)
6:     δE = E(solMov) − E(solInicial)
7:     if δE < 0 o estadisticoBoltzmann > random(0, 1) then
8:       solInicial = solMov
9:     end if
10:    bajarTemperaturaProgramada(tempInicial)
11:  until noIteraciones < 0
12: until tempInicial < limiteTemp
```

Sanchez-Oro [30] propone la metodología de GRASP para resolver el problema, evaluando dos tipos de construcciones GRASP y una técnica de búsqueda local empleando dos tipos de vecindarios, inserción e intercambio. Una de las construcciones iniciales es GRASP basada en un criterio de aceptación por umbral, la otra construcción consiste en tomar una muestra aleatoria de elementos que aún no se han seleccionado, se evalúan los elementos seleccionados con una función voraz y se toma de ellos el que produzca mejor valor de costo de dicha función (Sampled Greedy). Se crean dos combinaciones de construcciones GRASP

con los dos tipos de vecindario para tener 4 configuraciones distintas, donde las construcciones GRASP que utilizaron una muestra de los elementos dieron mejores resultados sin importar el tipo de vecindario utilizado para la búsqueda local. Sanchez-Oro realizó una actualización de su artículo [31] añadiendo dos propuestas de Path Relinking, Static Path Relinking (SPR) y Dynamic Path Relinking (DPR). Con estas nuevas propuestas se obtienen soluciones con tiempo de ejecución y desviación porcentual media menor, pero los mejores resultados siguen estando en la tesis de Lewis [31].

Como se muestra en el Algoritmo 4, éste iterativamente construye una solución y la mejora aplicando una búsqueda local para mejorar su calidad. Este proceso se repetirá un número definido de iteraciones y regresará la mejor solución encontrada.

Algoritmo 4 GRASP propuesto por Sanchez-Oro

```
1: mejorSolucion = ∅
2: for i = 1 → nIter do
3:    solucion = construir()
4:    mejorar(solucion)
5:    if SC(mejorSolucion) < SC(solucion) then
6:       mejorSolucion = solucion)
7:    end if
8: end for
```

A continuación serán detallados los métodos constructivos del GRASP empleados en la línea 3 del algoritmo.

La primera construcción inicia con la creación de nodos no etiquetados, posteriormente se etiquetan todos los nodos según su grado y se selecciona el nodo que posea el menor grado. A partir de este nodo se comienza a construir una lista de candidatos restringidos (RCL) con los nodos no seleccionados. La RCL se crea considerando una función voraz. Después se construye RCL con todos los nodos faltantes dentro del rango de la función voraz, de allí se toma un elemento aleatorio y así sucesivamente hasta que todos los nodos hayan sido seleccionados.

La segunda construcción inicia creando una lista de candidatos restringidos

(RCL) de forma aleatoria con el mismo criterio voraz que la primera construcción. Tras ello se toma el nodo con el mayor valor que la función voraz encuentre. La construcción termina hasta que todos los nodos sean tomados.

Adicional al GRASP, Sanchez-Oro propone dos algoritmos de Path Relinking: Static Path Relinking (SPR) y Dynamic Path Relinking (DPR). El algoritmo SPR inicia creando un conjunto de referencia ($RefSet$), éste se crea con el procedimiento GRASP que se ejecuta un número establecido de iteraciones generando múltiples soluciones. A partir de estas soluciones se toma un subconjunto de las mismas, la mitad de este subconjunto se selecciona según su calidad con base en el valor de la función objetivo y la otra mitad se selecciona con base en una función de distancia. Esta función de distancia es un valor absoluto entre la distancia del valor objetivo de la solución a y de la solución b. Una vez generado el $RefSet$, el algoritmo SPR genera caminos para cada par de soluciones hasta que se realicen todas las combinaciones del $RefSet$.

Algoritmo 5 Static Path-Relinking propuesto por Sanchez-Oro

```
1: refSet = ∅
2: for i = 1 → tamanoRefSet do
3:    sol = construir()
4:    mejorar(sol)
5:    anadirSiCalidad(sol, refSet)
6:    anadirSiDiversifica(sol, refSet)
7:    for all solA ∈ refSet do
8:       for all solB ∈ refSet do
9:          if solA ≠ solB then
10:            mejorCombinacion = combinar(solA, solB)
11:            actualizarMejor(mejorCombinacion)
12:         end if
13:      end for
14:   end for
15: end for
```

El método DPR empieza generando un $RefSet$ con soluciones construidas por GRASP, después se genera una solución inicial también con una construcción GRASP. Después se toma una solución aleatoria del $RefSet$ que será la guía. A partir de la solución inicial se realizan una serie de movimientos basados en el

vecindario de intercambio y se selecciona el mejor movimiento posible, también se guarda la solución que posea mayor calidad. Este proceso termina hasta que la solución inicial llegue a la solución guía. Ahora a partir de la mejor solución encontrada entre la solución inicial y la guía, se efectúa un proceso para ver si reemplaza o no a una solución del $RefSet$ que sea peor que ésta.

Algoritmo 6 Dynamic Path-Relinking propuesto por Sanchez-Oro

```
1: refSet = ∅
2: for i = 1 → tamanoRefSet do
3:    sol = construir()
4:    mejorar(sol)
5:    anadirSiDiferente(sol, refSet)
6: end for
7: for i = 1 → tamanoRefSet do
8:    sol = construir()
9:    mejorar(sol)
10:   refSetSol = seleccionSolAleatoria(refSet)
11:   mejorCombinacion = combinar(sol, refSetSol)
12:   actualizarMejor(mejorCombinacion)
13:   criterioAceptacionRefSet(mejorCombinacion, refSet)
14: end for
```

En la Tabla 3.1 se muestran las principales características de los trabajos de Lewis [22] y Sanchez-Oro [31].

Tabla 3.1: Principales Características

Autor	Problema	Metaheurística	Estructura de Vecindad	Mejores Resultados
Lewis 1993	Profile	SA	Swap	X
Sanchez-Oro 2012	SC	GRASP y Path Relinking	Insertion y Swap	

3.2. Estado del arte de búsquedas locales

En esta sección se analizarán las estrategias de búsqueda local aplicadas al problema del SUMCUT así como a los problemas de ordenamiento lineal.

Lewis [22] aplica una búsqueda $first$ para su Recocido Simulado (Simualted Annealing). Inicia tomando la permutación de la solución inicial, eligiendo aleatoriamente al elemento i y j y realizando un movimiento de intercambio. Esta búsqueda local continúa hasta que encuentre un movimiento que mejore a la solución inicial. La búsqueda local de Lewis se presenta en el Algoritmo 7

Algoritmo 7 Búsqueda local first de Lewis

```
1: costoInicial = solInicial
2: Mejora = false
3: repeat
4:    elem1 = random(1, totalElem);
5:    elem2 = random(1, totalElem);
6:    if elem1 ≠ elem2 then
7:       nvaSol = movimientoIntercambio(solInicial, elem1, elem2)
8:       costoNvaSol = costo(nvaSol)
9:       if costoNvaSol < costoInicial then
10:         solInicial = movimientoIntercambio(solInicial, elem1, elem2)
11:      end if
12:   end if
13: until (Mejora = true)
14: return (π, costo)
```

Laguna [20] en su artículo describe los vecindarios de inserción e inserción consecutiva, después de definir los tipos de vecindarios introduce las estrategias de paro: $first$ y *best*. $First$ explora una lista de sectores dada la permutación en busca del primer movimiento que mejore la solución actual; mientras que *best* realiza el movimiento que resulte el mejor valor objetivo del vecindario. Una vez introducidos estos conceptos, Laguna realiza combinaciones entre las dos estrategias y los dos vecindarios. Reporta mejores resultados con la estrategia $first$ con el vecindario de inserción.

Piñana [28] presenta en sus búsquedas locales el término de elemento crítico, donde un elemento crítico es aquel elemento que posee alguna característica que lo identifica como un elemento que puede producir una gran mejora en la solución. Piñana define como elemento crítico a un elemento de una lista de candidatos cuyos elementos tienen como valor objetivo al valor objetivo del grafo y los vectores críticos cercanos no determinados en el valor objetivo del grafo. Ella define

el vecindario de inserción y dos estrategias de paro: $first$ y $best$. Sus resultados muestran que la estrategia de selección $first$ produce mejores resultados a su problema.

Pantrigo [26] diseñó sus búsquedas locales basadas en un vecindario de inserción, con una estrategia de paro $first$ y, como Piñana, también emplea el uso de elementos críticos. Pantrigo define como elemento crítico, a los vértices cuyo valor sea igual o cercano al Cutwidth del grafo. Dado que en su problema, realizar un movimiento de inserción a veces produce valores nulos, Pantrigo también propone el predecir el costo antes de realizar el movimiento de inserción, actualizando la lista de candidatos y la realización del movimiento hasta que todos los vértices sean explorados. Sus resultados demuestran que la predicción de costos ahorra tiempo computacional.

Terán [33] propone, para el problema de LOPCC, una búsqueda local compuesta y una función de predicción de costos. Esta búsqueda está conformada por tres búsquedas locales basadas en un vecindario de inserciones consecutivas, con selección de elementos críticos. Él considera tres criterios para generar la lista de elementos críticos (LEC). En dos de sus búsquedas locales utiliza el criterio de elementos críticos considerando el valor objetivo de cada elemento, para la búsqueda local restante emplea una selección de elementos aleatorios. La primera búsqueda local se basa en la selección de elementos críticos según el valor objetivo. Ésta utiliza una estrategia de paro best para insertar los elementos en sus mejores posiciones posibles del vecindario y los elimina de la LEC. La búsqueda se realizará a partir de un porcentaje definido del tamaño de la instancia. La segunda búsqueda local intenta mejorar las soluciones de la primera búsqueda local, seleccionando elementos de la LEC de manera aleatoria con base en una distribución exponencial que da preferencia a elementos críticos. Después de haber seleccionado un elemento, éste se insertará en la mejor posición del vecindario y se elimina de la LEC, la búsqueda se realiza un número de iteraciones sin mejora. La tercera búsqueda elige el elemento más crítico de la LEC e intenta insertarlo

en la mejor posición del vecindario y luego se elimina de la LEC. Este proceso continúa mientras se haya encontrado alguna mejora en el proceso de búsqueda.

Sanchez-Oro [31] realiza una búsqueda local basada en vecindarios de inserción e intercambio, donde los vértices se recorren al azar. En concreto, el procedimiento de búsqueda local selecciona de forma aleatoria un vértice v, tal que el vértice u es $v+1$. Los vértices restantes se analizan en orden sistemático desde el primer vértice hasta el vértice n. El procedimiento intenta realizar intercambio o inserción, realiza el movimiento según el tipo de vecindario deseado. Si el movimiento reduce el valor objetivo, la solución original se actualiza. Esta búsqueda se realiza hasta que no existan movimientos que generen mejoras al valor objetivo. El algoritmo de esta búsqueda local se presenta en el Algoritmo 8, en la línea 6 es donde se elige si usar la vecindad de inserción o de intercambio.

Algoritmo 8 Búsqueda local LSf de Sanchez-Oro

```
1:  mejora = true
2:  while mejora do
3:    mejora = false
4:    for all v ∈ V do
5:      for all u ∈ V do
6:        φ' = intercambio()
7:        if SC(φ', G) < SC(φ, G) then
8:          mejora = true
9:          φ' = φ
10:       end if
11:     end for
12:   end for
13: end while
```

Después del diagnóstico del estado del arte se observa que aunque sean distintos problemas y metaheurísticas aplicadas a los mismos, las estrategias de búsqueda local que emplean son muy parecidas entre ellas, entonces se puede concluir que sea conveniente probar algunas ideas empleadas en estos problemas, así como proponer nuevas estrategias para nuestro problema de investigación.

A continuación se presenta una tabla con los datos generales de la bibliografía revisada.

Autor	**Vecindario**	**Búsqueda Local**	**Criterio de Selección**	**Predicción de Costos**
Lewis 1993	Intercambio	First	Mejora en el valor de la función objetivo	No
Laguna et al. 1998	Inserción	First y Best	Mejora en el valor de la función objetivo	No
Piñana et al. 2001	Inserción	First y Best	Lista de movimientos candidatos, selección de críticos	No
Pantrigo et al. 2010	Inserción	First	Selección de críticos y mejora en el valor de la función objetivo	Sí
Terán et al. 2012	Inserción	Compuesta	Seleccion de críticos y aleatorio	Sí
Sanchez-Oro 2012	Intercambio e Inserción	LS_f	Mejora en el valor de la función objetivo	No

Tabla 3.2: Estado del arte de las búsquedas locales

Capítulo 4

Enfoque de solución

4.1. Análisis de la función objetivo del SUMCUT

En esta sección se describe un método eficiente para calcular la función objetivo del problema SUMCUT.

Para una permutación dada π el valor objetivo de π está dado por:

$$SC(\pi) = CW(\pi_1) + \ldots + CW(\pi_{\mathrm{k}}) + \ldots + CW(\pi_{\mathrm{n}})$$

Este método tiene una complejidad de $O(n^2)$.

Por otra parte si se consideran los arreglos $C = \{c_{\pi_k} | k = 1, \ldots, n\}$ y $R = \{c_{\pi_k} | k = 1, \ldots, n\}$ donde:

$$c_{\pi_k} = \sum_{s=1}^{k-1} a_{\pi_s \pi_k} \quad r_{\pi_k} = \sum_{w=k+1}^{n} a_{\pi_k \pi_w},$$

entonces

$$CW(\pi_k) = \begin{cases} 0, \text{ para } k = 0 \\ CW(\pi_{k-1}) + R(\pi_k) - C(\pi_k), \text{ para } k = 1, 2, \ldots, n. \end{cases}$$

Por lo tanto el SUMCUT de una permutación π está dado por:

$$SC(\pi) = \sum_{k=1}^{n} CW(\pi_{k-1}) + R(\pi_k) - C(\pi_k) \tag{4.1}$$

Como se observa el costo del cálculo del valor objetivo de una permutación utilizando este método es $O(n^2)$.

Veamos un ejemplo de este nuevo cálculo del SUMCUT

Dadas una permutación $\pi = (1, 2, 3, 4, 5)$ y la matriz de adyacencia a según π, se calcularán los valores de C y R para cada elemento de π

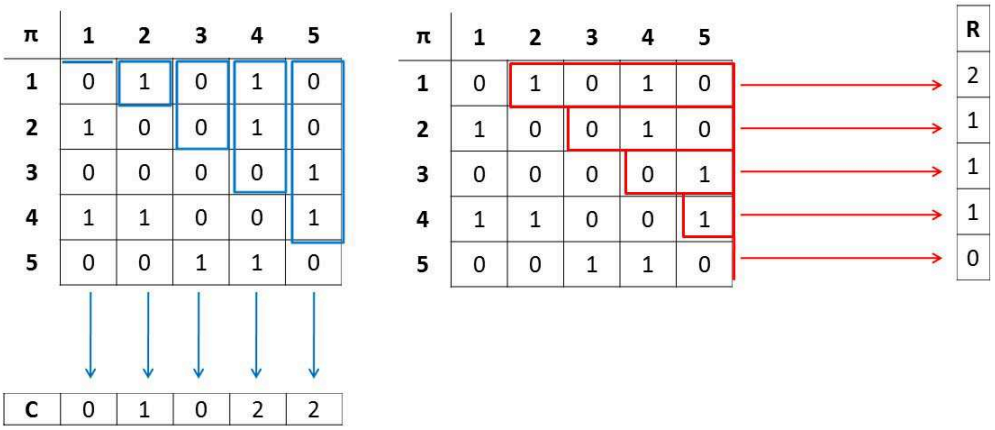


Figura 4.1: Cálculo de C y R

Por ejemplo, para un caso particular para cada una de las estructuras, por ejemplo $C(\pi_4)$ y $R(\pi_1)$

$$\begin{aligned}
C(\pi_k) &= \sum_{c=1}^{k-1} a_{\pi_c \pi_k} & R(\pi_k) &= \sum_{r=k+1}^{n} a_{\pi_k \pi_r} \\
C(\pi_4) &= \sum_{c=1}^{3} a_{\pi_c \pi_4} & R(\pi_1) &= \sum_{r=2}^{5} a_{\pi_1 \pi_r} \\
C(\pi_4) &= a_{\pi_1 \pi_4} + a_{\pi_2 \pi_4} + a_{\pi_3 \pi_4} & R(\pi_1) &= a_{\pi_1 \pi_2} + a_{\pi_1 \pi_3} + a_{\pi_1 \pi_4} + a_{\pi_1 \pi_5} \\
C(\pi_4) &= 1 + 1 + 0 & R(\pi_1) &= 1 + 0 + 1 + 0 \\
C(\pi_4) &= 2 & R(\pi_1) &= 2
\end{aligned}$$

Entonces el $SC(\pi)$ es:

$$
\begin{aligned}
CW(\pi_1) &= CW(\pi_0) + R(\pi_1) - C(\pi_1) = 0 + 2 - 0 = 2 \\
CW(\pi_2) &= CW(\pi_1) + R(\pi_2) - C(\pi_2) = 2 + 1 - 1 = 2 \\
CW(\pi_3) &= CW(\pi_2) + R(\pi_3) - C(\pi_3) = 2 + 1 - 0 = 3 \\
CW(\pi_4) &= CW(\pi_3) + R(\pi_4) - C(\pi_4) = 3 + 1 - 2 = 2 \\
CW(\pi_5) &= CW(\pi_4) + R(\pi_5) - C(\pi_5) = 2 + 0 - 2 = 0 \\
SC(\pi) &= CW(\pi_1) + CW(\pi_2) + CW(\pi_3) + CW(\pi_4) + CW(\pi_5) \\
&= 2 + 2 + 3 + 2 + 0 \\
&= 9
\end{aligned}
$$

Cambio del valor objetivo al aplicar una inserción consecutiva

En esta sección se presenta un método para predecir el cambio en el valor objetivo de una permutación cuando se le aplica un movimiento de inserción consecutiva. Aplicando este método, se puede calcular el valor objetivo de las soluciones vecinas de una permutación, sin realizar los movimientos de inserción requeridos para generarlas. Esta información es utilizada para determinar cuál es el movimiento que conviene realizar.

Sean π y $\pi^{'}$ permutaciones, tales que $\pi^{'} = InsertMove(\pi, i, i+1)$. El cambio en el valor objetivo del movimiento de inserción está dado por:

$$
\begin{aligned}
Cost(InsertMove(\pi, i, i+1)) &= SC(\pi^{'}) - SC(\pi) \\
&= (CW(\pi^{'}_{\mathrm{i}}) - CW(\pi_{\mathrm{i}})) + (CW(\pi^{'}_{(\mathrm{i}+1)}) - CW(\pi_{(\mathrm{i}+1)}))
\end{aligned}
$$

donde

$$CW(\pi_{\mathrm{i}}^{'}) \;=\; CW(\pi_{\mathrm{i}}) + R(\pi_{(\mathrm{i}+1)}) - C(\pi_{(\mathrm{i}+1)})$$

y

si $a[\pi_{\mathrm{i}}^{'}][\pi_{(\mathrm{i}+1)}^{'}] = 1$ entonces

$$CW(\pi_{(\mathrm{i}+1)}^{'}) \;=\; CW(\pi_{(\mathrm{i}+1)}) - (R(\pi_{\mathrm{i}}) - 1) + (C(\pi_{\mathrm{i}}) + 1).$$

En caso contrario

$$CW(\pi_{(\mathrm{i}+1)}^{'}) \;=\; CW(\pi_{(\mathrm{i}+1)}) - C(\pi_{\mathrm{i}}) + R(\pi_{\mathrm{i}}).$$

La complejidad de realizar un movimiento de inserción consecutiva con este método está asociada a tres operaciones. Al combinar el nuevo método del cálculo del SUMCUT con el método para predecir el costo de un movimiento de inserción consecutiva producen una complejidad de $O(n^2)$. Mientras que en el caso tradicional se realiza todo el recalculo de la función objetivo n veces produciendo una complejidad de $O(n^3)$.

Veamos un ejemplo del cambio de costo del movimiento de inserción consecutiva:

Sea $\pi = (a, b, c, d, e)$, $SC(\pi) = 9$ y $\pi^{'} = InsertMove(\pi, d, e)$, el cambio de costo del movimiento es el siguiente:

$$\begin{aligned} CW(\pi_{\mathrm{d}}^{'}) &= CW(\pi_{\mathrm{d}}^{'}) + R(\pi_{\mathrm{e}}^{'}) - C(\pi_{\mathrm{e}}^{'}) \\ &= 2 + 0 - 2 \\ &= 0 \end{aligned}$$

Figura 4.2: Ejemplo del costo de inserción consecutiva

Realizando la actualización de los elementos de C y R:

$$\begin{aligned} C(\pi_{\mathrm{d}}^{'}) &+= 1 \\ R(\pi_{\mathrm{e}}^{'}) &+= 1 \\ C(\pi_{\mathrm{e}}^{'}) &-= 1 \\ R(\pi_{\mathrm{d}}^{'}) &-= 1 \end{aligned}$$

Calculando $CW(\pi_{\mathrm{e}}^{'})$:

$$\begin{aligned} CW(\pi_{\mathrm{e}}^{'}) &= CW(\pi_{\mathrm{e}}^{'}) - R(\pi_{\mathrm{d}}^{'}) + C(\pi_{\mathrm{d}}^{'}) \\ &= 0 - 0 + 3 \\ &= 3 \end{aligned}$$

Finalmente se obtiene el $SC(\pi)$ basado en las diferencias de $CW(d)$ y $CW(e)$

$$
\begin{aligned}
\Delta CW(d) &= CW(\pi_{\mathrm{d}}^{'}) - CW(\pi_{\mathrm{d}}) \\
\Delta CW(d) &= 0 - 2 \\
\Delta CW(d) &= -2 \\
\Delta CW(e) &= CW(\pi_{\mathrm{e}}^{'}) - CW(\pi_{\mathrm{e}}) \\
\Delta CW(e) &= 3 - 0 \\
\Delta CW(e) &= 3 \\
SC(\pi^{'}) &+= \Delta CW(d) + \Delta CW(e) \\
SC(\pi^{'}) &+= -2 + 3 \\
SC(\pi^{'}) &+= 1
\end{aligned}
$$

Como el $SC(\pi) = 10$, entonces $SC(\pi^{'}) = 11$, por lo tanto el $Cost(InsertMove(\pi, d, e)) = 1$. Como es un valor positivo significa que no mejoró la solución.

4.1.1. Diseño de las búsquedas locales

Las búsquedas locales sirven para intentar mejorar una solución, lo cual consiste en tomar la solución actual y buscar una mejor solución dentro de su vecindad [8]. Las estrategias de búsqueda local básicas son: First, Best y Óptima.

- **First:** Evalúa el vecindario en busca del primer movimiento de inserción que produzca una mejora sobre el valor objetivo de la solución actual.

- **Best:** Una modificación de First, evalúa todo el vecindario de inserción buscando la mejor posición que mejore el valor objetivo.

- **Óptima:** Una modificación de Best, evalúa toda la vecindad de inserción hasta encontrar el mejor movimiento que mejore el valor objetivo, si existiera una mejora, el proceso se vuelve a repetir. En el Algoritmo 9 se muestra

la estructura de esta búsqueda.

En la Figura 4.3 se muestra la permutación $\pi = (1, 2, 3, 4, 5)$ con un valor objetivo de $SC(\pi) = 15$ y el elemento que se pretende posicionar $i = 3$. Se realiza una exploración de las diferentes posiciones de la permutación actual recorriendo primero las posiciones a la izquierda de la posición i, y luego las posiciones a su derecha. Para cada posición visitada, se determina el valor objetivo de la permutación que se genera al aplicar las inserciones consecutivas correspondientes. La exploración se detiene en la posición que produce una mejora en el valor objetivo de la permutación actual. Como se observa en la figura, la primera mejora que se encuentra requiere cambiar el elemento de la posición $i = 3$ a la posición $j = 1$, dando como resultado la permutación $(3, 1, 2, 4, 5)$ que tiene un valor objetivo $SC = 13$.

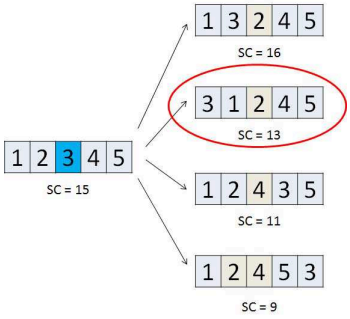


Figura 4.3: Búsqueda First

En la Figura 4.4 se muestra la permutación $\pi = (1, 2, 3, 4, 5)$ con un valor objetivo de $SC(\pi) = 15$ y el elemento que se pretende posicionar $i = 3$. Se realiza una exploración de las diferentes posiciones de la permutación actual recorriendo primero las posiciones a la izquierda de la posición i, y luego las posiciones a su derecha. Para cada posición visitada, se determina el valor objetivo de la permutación que se genera al aplicar las inserciones consecutivas correspondientes. La exploración se detiene una vez que se evalúan todas las posiciones posibles y se elige la posición que genere la mayor mejora en el valor objetivo de la permutación

actual. Como se observa en la figura, la mayor mejora se logra al cambiar el elemento de la posición $i = 3$ a la posición $j = 5$, dando como resultado la permutación $(1, 2, 4, 5, 3)$ que tiene un valor objetivo $SC = 9$.

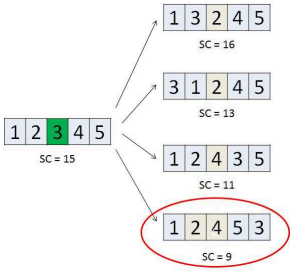


Figura 4.4: Búsqueda Best

En la Figura 4.5, se muestra la permutación $\pi = (1, 2, 3, 4, 5)$ con un valor objetivo de $SC(\pi) = 15$ y el elemento que se pretende posicionar $i = 3$. Se realiza una exploración de las diferentes posiciones de la permutación actual recorriendo primero las posiciones a la izquierda de la posición i, y luego las posiciones a su derecha. Para cada posición visitada, se determina el valor objetivo de la permutación que se genera al aplicar las inserciones consecutivas correspondientes. La exploración se detiene una vez que se evalúan todas las posiciones posibles y se elige la posición que genere la mayor mejora en el valor objetivo de la permutación actual. En este punto, se actualiza la permutación actual realizando el reposicionamiento del elemento considerado $i = 3$ y se aplica de nueva cuenta el procedimiento anterior a la nueva permutación. Este proceso termina cuando no se encuentra alguna posición que genere una mejora a la permutación actual. Como se observa en la figura, la mayor mejora se logra al cambiar el elemento de la posición $i = 3$ a la posición $j = 5$, dando como resultado la permutación $(1, 2, 4, 5, 3)$ que tiene un valor objetivo $SC = 9$.

En el Algoritmo 9 se describe la estructura de una búsqueda local genérica. El proceso recibe una permutación π y regresa la permutación mejorada y su valor objetivo. El proceso consiste en determinar para cada elemento de π la posición en la que se produce una mejora una vez insertada y el costo de dicha inserción. Para cada mejora encontrada se realiza la inserción correspondiente y se actualiza la permutación actual y su valor objetivo. El proceso termina una vez que se han

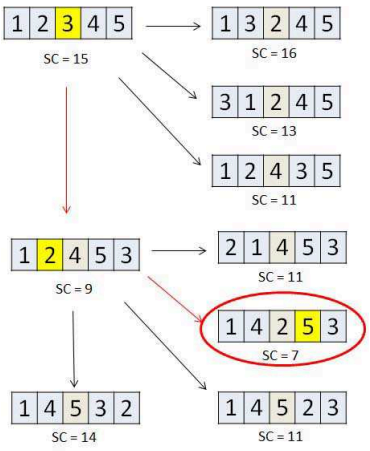


Figura 4.5: Búsqueda Óptima

analizado todos los elementos de la permutación actual.

Algoritmo 9 Algoritmo de búsqueda local

Input: π
Output: $bestPositionToInsert$

1: $VO = SC(\pi)$
2: **repeat**
3: $Mejora = false$
4: **for** $i = 1 \rightarrow n$ **do**
5: $j^* = argmin_{j=1,\ldots,n \text{ and } j \neq i} Cost(InsertMove(\pi, i, j))$
6: $cost_{min} = Cost(InsertMove(\pi, i, j^*))$
7: **if** $cost_{min} < 0$ **then**
8: $\pi = InsertMove(\pi, i, j^*)$
9: $VO = VO - cost_{min}$
10: $Mejora = true$
11: **end if**
12: **end for**
13: **until** $(Mejora = true)$
14: **return** (π, VO)

4.1.2. Métodos de selección de elementos

En una búsqueda local un conjunto de posiciones de la permutación actual son seleccionados para tratar de reposicionarlos y producir una mejora en el valor objetivo de la permutación. La búsqueda descrita en la sección anterior selecciona todas las posiciones de la permutación actual, sin embargo, esta selección no

necesariamente debe ser exhaustiva. En este trabajo se consideran los siguientes métodos de selección de elementos:

- **Selección óptima:**

  Se seleccionan todas los elementos de la permutación π y para cada uno de ellos, se inserta en la mejor posición posible.

- **Selección crítica total:**

  En [28] se define un elemento crítico como aquel que posee un valor de Cutwidth elevado. El método de selección crítica total consiste en ordenar de mayor a menor la permutación, de acuerdo al valor del Cutwidth de cada una de sus posiciones, y se eligen las posiciones que se ubican en la primera mitad de este ordenamiento. Cada elemento seleccionado se trata de reposicionar en la mejor posición posible.

- **Selección crítica aleatoria:**

  Este método consiste en seleccionar un 50 % de las posiciones de π de manera aleatoria. Los elementos seleccionados se ordenan de mayor a menor de acuerdo al valor de su Cutwidth. Se selecciona el 50 % de las posiciones que se ubican en la primera mitad del ordenamiento anterior. Para cada elemento se trata de reposicionar en la mejor posición posible.

A continuación se presenta un ejemplo del método de selección crítica total:

Sea π una permutación donde $\pi = (1, 2, 3, 4)$ y $CW = (2, 3, 2, 0)$ sean los cortes para cada elementos de π y $SC = 7$, se va a realizar la búsqueda local basada en la selección de elementos Crítica Total.

Primero se ordena π de mayor a menor según CW donde π quedará reordenado en $(2, 1, 3, 4)$, cabe resaltar que π no ha sido afectado. De esta reordenación se toma la primera mitad de los elementos de mayor valor crítico, en este caso serían los elementos 2 y 1 de π.

Ahora el elemento 2 se insertará en la mejor posición de todas las posibles, este movimiento resultará en $\pi = (2, 1, 3, 4)$ debido a que reduce el valor objetivo a $SC = 6$. Continuaremos con el elemento 1 intentando encontrar la mejor posición para insertar dicho elemento, pero ninguno de los movimientos reduce a SC.

Al final de intentar insertar todos los elementos críticos en sus mejores posiciones, se vuelve a reordenar π según su CW y elegir la mitad de los elementos críticos. En este caso ya no existe ningún movimiento que mejore dichos elementos y al no existir una mejora, la búsqueda local termina entregando $\pi = (2, 1, 3, 4)$.

En la Figura 4.6 se puede apreciar el ejemplo anterior descrito.

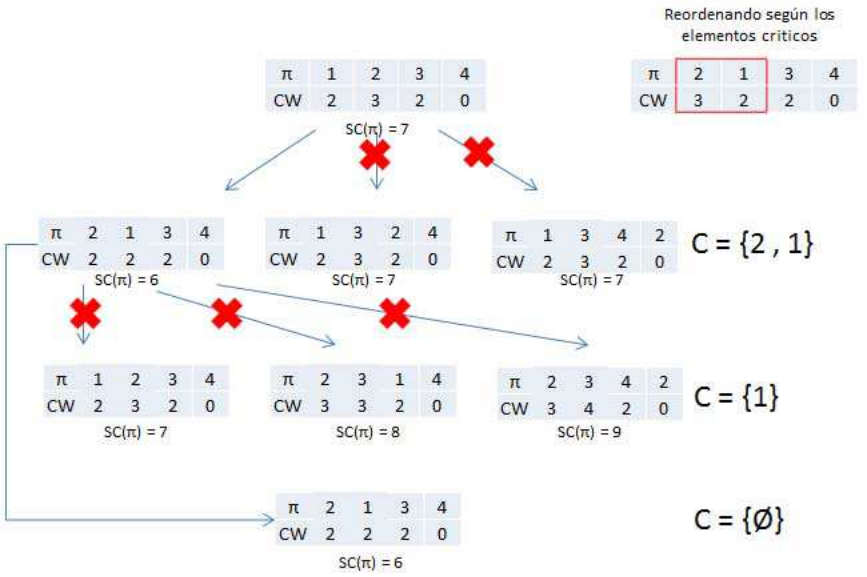


Figura 4.6: Ejemplo del método de selección crítica total

4.2. Búsqueda local iterada propuesta

Este algoritmo construye una solución inicial, la cual es mejorada utilizando una búsqueda local. De aquí inicia un ciclo hasta alcanzar una condición de paro. Dentro del ciclo se perturba y se mejora la solución actual. El ciclo continua hasta que se alcance un número determinado sin mejora local. La Figura 10 muestra la estructura general del algoritmo y las funciones que utiliza.

El algoritmo utilizado fue diseñado de la siguiente manera:

Algoritmo 10 Búsqueda Local Iterada

$S_0 = GenerarSolucionInicial()$
$S^* = BusquedaLocal()$
repeat
  $S' = PerturbarSolucion(S^*)$
  $S^{*'} = BusquedaLocal(S')$
until $CondicionDeParo()$

- $GenerarSolucionInicial()$:

  Esta función construye la permutación inicial de manera aleatoria.

- $BusquedaLocal()$:

  En esta función se aplican las estrategias de búsqueda local first, best u óptima descritas en la sección 3.2.

- $PerturbarSolucion()$:

  El porcentaje de perturbación es de 15 % del tamaño de la instancia.

- $CondicionDeParo()$:

  La condición de paro del algoritmo es de 10 iteraciones sin mejora y que el tiempo de ejecución no pase de 1000 segundos.

4.3. Búsqueda local iterada celular

Para evaluar experimentalmente la factibilidad de las búsquedas locales propuestas, se implementó una solución metaheurística basada en búsqueda local iterada celular. El enfoque celular se basó en la estructura propuesta por Terán [32].

El algoritmo celular ejecuta en pseudoparalelo múltiples procesos de búsqueda, los cuales se mueven a través del espacio de soluciones. Cada celda de procesamiento es una búsqueda local iterada que inicia a partir de una solución aleatoria mejorada con una búsqueda local. Una vez que las celdas entran en ejecución realizan un ciclo de perturbación aleatoria y mejora local. Este ciclo continúa hasta que se observe que en un número determinado de iteraciones ya no se producen

mejoras al mejor valor localmente encontrado. Una vez alcanzada esta condición se considera que la celda está estancada por lo que ya no es necesario ejecutarla como consecuencia de que ya no produce mejoras locales. Las celdas de procedimiento se ejecutan una después de otra (pseudoparalelo) en un ciclo global hasta que en un cierto número de iteraciones globales no se observe una mejora global. Dentro del ciclo global una vez que todas las celdas estén estancadas, se realiza el proceso de comunicación entre dichas celdas, con el propósito de encontrar una mejora global. El proceso de comunicación consiste en cruzar todas las mejores soluciones locales de las celdas entre ellas, si una de las cruzas mejora a uno de los padres, este se remplaza por el hijo. Una vez realizadas todas las cruzas, se actualiza el mejor valor global y se reinician todas las celdas para continuar su procesamiento. Este algoritmo se describe en el Algoritmo 11

Algoritmo 11 Algoritmo de Cellular ILS

Input: $numeroCeldas$

for $CeldaActual = 1 \rightarrow numeroCeldas$ **do**
  $GeneraSolucionAleatoria(CeldaActual)$
  $BusquedaLocal(CeldaActual)$
end for
repeat
  for $CeldaActual = 1 \rightarrow numeroCeldas$ **do**
    if $CeldaEstancada == false$ **then**
      for $i = 1 \rightarrow iterPorCelda$ **do**
        $PertubarSolucionDeCelda(CeldaActual)$
        $BusquedaLocal(CeldaActual)$
      end for
      if $CeldaActual < OptimoCeldaActual$ **then**
        $OptimoCeldaActual = CeldaActual$
      else
        $CeldaEstancada = true$
      end if
    end if
  end for
  $ComunicacionDeCeldas()$
until $CondicionDeParo()$
return $(\pi, costo)$

Capítulo 5

Resultados experimentales

En este capítulo se presentan los resultados de los diversos experimentos realizados para este trabajo de tesis.

5.1. Hardware y software

Todos los experimentos fueron realizados el siguiente este entorno de prueba:

- **Procesador:** Intel Xeon 3.06 GHz.
- **Memoria RAM:** 4 GB.
- **Sistema operativo:** Windows XP Service Pack 3.
- **Entorno de desarrollo:** Microsoft Visual Studio 2008.
- **Lenguaje de programción:** Visual C++.

Las instancias empleadas para verificar el desempeño de nuestro algoritmo se dividen en 3 grupos: Small, Grid y Harwell-Boeing. Estas instancias pueden ser

descargadas del sitio del Proyecto Optsicom (http://www.optsicom.es/cutwidth/). A continuación se dará un breve antecedente de las instancias.

- **Small:** Fueron introducidas por Martí [24] en el 2008. Este conjunto de 84 grafos fue usado para el problema de minimización del Bandwidth. El número de vértices va desde 16 a 24, con un número de aristas desde 18 hasta 49.

- **Grid:** Introducidas por Rolim [29] en 1995. Conjunto de 81 matrices, donde cada una de ellas fue creada por el resultado de dos productos cartesianos. El número de vértices es de 9 hasta 729.

- **Harwell-Boeing:** Provenientes de la Colección de Matrices Dispersas de Harwell-Boeing, 88 matrices diseñadas para diversas pruebas de áreas científicas o de ingeniería. El número de vértices es de 30 hasta 700, con aristas desde 46 hasta 41686.

5.2. Experimentos

A continuación se describen los experimentos realizados.

5.2.1. Experimento 1

La finalidad de este experimento es comprobar de manera práctica que la nueva forma de calcular el SUMCUT y la forma de predecir el cambio del costo del movimiento de inserción consecutiva producen resultados estadísticamente similares empleando menos tiempo para calcularlos.

En la tabla 5.1 se encuentran los resultados de la experimentación realizada. En la primera columna se menciona la búsqueda local empleada, en la segunda columna se especifica el método de cálculo de la función objetivo y de búsqueda en

el vecindario, denotando con *Original* la metodología original y con *Propuesta* la metodología propuesta. En la tercera columna se encuentra el promedio del valor objetivo obtenido, en la cuarta columna el tiempo promedio utilizado en segundos, y finalmente en la quinta columna se encuentra el porcentaje de reducción de tiempo entre ambas metodologías.

En esta tabla se puede observar que para el conjunto de instancias Small, Grid y Harwell-Boeing, para todos los conjuntos de instancias usando ambas metodologías se obtiene la misma calidad en sus soluciones; sin embargo, se puede notar una gran diferencia en el tiempo de ejecución de las mismas. Para el conjunto Small con la búsqueda local First utilizando la metodología *Propuesta* se emplea el 52 % del tiempo promedio que la metodología *Original*, 2 % en la búsqueda local Best y 3 % en la búsqueda local Óptima. En el caso del conjunto Grid para sus tres búsquedas locales existe un ahorro del 99 % en el tiempo entre metodologías. Finalmente para las Harwell-Boeing su búsqueda local First tiene un 98 % de ahorro en el tiempo, y en la búsqueda Best y Óptima un 99 %. Se puede observar en los resultados que entre más grande sea la instancia, el ahorro de tiempo se vuelve más notorio.

5.2.2. Experimento 2

Debido a que la información sobre los mejores resultados para los conjuntos de instancias anteriormente mencionados no eran muy claros, se propone establecer los mejores valores conocidos para tomar como futuras referencias para los demás experimentos de este trabajo.

El algoritmo para generar estos resultados está definido de la siguiente manera: Inicia generando una permutación aleatoria y aplicándole una búsqueda local óptima, este proceso se repite para cada instancia hasta emplear 18000 segundos (300 minutos) arrojando el mejor valor posible encontrado por instancia.

En la Tabla 5.2 se muestran los resultados obtenidos de dicha experimenta-

Búsqueda Local	Forma de Cálculo	Búsqueda Local (Promedio)	Tiempo (Segundos)	Porcentaje de Ahorro
Small				
First	*Original*	138.17	0.46	48.12 %
	Propuesta	138.17	0.234	
Best	*Original*	68.18	13.76	98.33 %
	Propuesta	68.18	3.23	
Óptima	*Original*	61.81	64.51	97.93 %
	Propuesta	61.81	2.09	
Grid				
First	*Original*	49078.52	16468.61	99.11 %
	Propuesta	49078.52	147.154	
Best	*Original*	474.19	2798.48	99.96 %
	Propuesta	474.19	1.094	
Óptima	*Original*	411.88	44257.70	99.98 %
	Propuesta	411.88	7.155	
Harwell-Boeing				
First	*Original*	237662.09	26145.45	98.76 %
	Propuesta	237662.09	326.667	
Best	*Original*	9486.82	337462.62	99.99 %
	Propuesta	9486.82	3.549	
Óptima	*Original*	7782.35	336385.06	99.99 %
	Propuesta	7782.35	15.063	

Tabla 5.1: Resultados de la experimentación 1

ción. En la primera columna se encuentra el conjunto de instancias utilizado, en la segunda columna se especifica el valor objetivo promedio de dicho conjunto de instancias, la tercera columna presenta el tiempo promedio empleado en segundos y finalmente la cuarta columna muestra el porcentaje de error según los mejores valores conocidos.

Conjunto	VO	Tiempo	% Error
Small	60.58	18000	0 %
Grid	3421.21	18000	1.6 %
Harwell-Boeing	81302.05	18000	100.62 %

Tabla 5.2: Resultados de la experimentación 2

5.2.3. Experimento 3

En este experimento se busca encontrar una configuración apropiada para una Búsqueda Local Iterada Celular (ILS Celular), donde la mejor configuración es la que resulte en una menor cantidad de desviación de error dada una cantidad determinada de celdas iniciales que provean de mejores resultados para cada conjunto de instancias. El ILS Celular inicia de la siguiente manera: Primero se define un número de celdas a utilizar, para cada celda se genera una solución aleatoria y se le aplica una búsqueda local. Después de inicializar las celdas, cada celda no estancada realizará una perturbación aleatoria y aplicará una búsqueda local un número determinado de veces, posteriormente se verifica si mejora el óptimo de esa celda, si dicho óptimo mejora la celda es actualizada, en caso contrario se marca como estancada. Una vez que todas las celdas estén estancadas se realiza una comunicación que consiste en cruzar todas las soluciones de las celdas entre ellas, si una de las cruzas entre dos soluciones de celda es mejor que la peor solución de un padre, se reemplaza por el resultado de dicha cruza. Finalmente, está la condición de paro la cual analiza las soluciones de cada celda, si ninguna celda mejora el óptimo global se marca como una iteración sin mejora, al cumplir tres iteraciones sin mejora el algoritmo termina. Para estos experimentos emplearemos una búsqueda local Óptima como la búsqueda local del algoritmo y el número de celdas a probar serán 1, 3, 5 y 10.

En la Tabla 5.3 se encuentran los resultados de esta experimentación, la cual está construida de la siguiente manera: en la primera columna se encuentra el número de celdas empleadas en esa experimentación, en la segunda columna se encuentra el valor objetivo promedio obtenido, en la tercera columna se encuentra el tiempo promedio en segundos, y finalmente en la cuarta columna está el porcentaje de error de esa configuración basado en los mejores conocidos.

Para el conjunto de instancias Small, la mejor configuración de celdas resulta en ser 1, debido a que es la que obtiene el mejor valor objetivo promedio, tiempo promedio y bajo porcentaje de error respecto a los mejores conocidos. Otra buena configuración para este conjunto es la 5 con tiempos muy cercanos así como calidad similar a la configuración de 1 celda.

Para las instancias Grid se puede observar que, entre más celdas se agreguen, la calidad de las soluciones mejora, pero también el tiempo empleado empieza a incrementarse. La configuración que da mejores resultados a costa del consumo de tiempo es la de 10 celdas, pero la configuración que da un balance aceptable entre tiempo y calidad en las soluciones es la de 5 celdas.

Finalmente para el conjunto Harwell-Boeing encontramos en los resultados que la configuración que da mejor calidad de soluciones es la de 5 celdas, el mejor tiempo la de 1 celda y el mejor porcentaje de error la de 3 celdas. Las configuraciones de 3 y 10 Celdas producen resultados similares así como su porcentaje de error pero con una diferencia en el tiempo empleado para obtener dichos resultados. Al igual que en el conjunto Grid, la configuración de 5 celdas provee un buen balance entre calidad y tiempo en sus resultados.

N° de Celdas	Valor Objetivo (Promedio)	Tiempo Promedio (Segundos)	Porcentaje de Error
Small			
1	60.71428571	0.14834524	0.216814464
3	60.9047619	0.16091667	0.554493724
5	60.78571429	0.14991667	0.352857124
10	60.79761905	0.16222619	0.363508059
Grid			
1	3571.444444	75.90254321	5.801645954
3	3448.074074	102.8804074	2.3705829
5	3424.135802	112.9363704	1.975679171
10	3405.209877	122.5385926	1.528773051
Harwell-Boeing			
1	82930	101.6926322	69.54066998
3	79860.13793	119.6966897	52.29948114
5	78848.58621	135.8731839	54.77383762
10	79371.88506	158.2618851	53.5631744

Tabla 5.3: Resultados de la experimentación 3

Después del análisis de estos resultados, se encontró que la mejor configuración que funcionó para este problema con el algoritmo ILS Celular es la de 5 celdas, debido al balance entre la calidad de las soluciones, tiempo promedio requerido y un porcentaje de error razonable.

5.2.4. Experimento 4

La finalidad de este experimento es la de evaluar nuevas búsquedas locales para este problema utilizando el ILS Celular. Este experimento se realizó con el conjunto de instancias Harwell-Boeing y se eligió utilizar 5 celdas para el algoritmo. Las búsquedas locales a evaluar en este experimento: *Optima*, *CriticaAleatoria* y *CriticaTotal*.

En la Tabla 5.4 se muestran los resultados de la experimentación realizada. Esta tabla está construida de la siguiente manera: en la primera columna se menciona la búsqueda local empleada, en la segunda columna se encuentra el valor objetivo promedio del conjunto de instancias, la tercera muestra el tiempo promedio empleado en segundos, y finalmente la cuarta columna indica el porcentaje de error promedio para este conjunto de instancias.

En esta misma tabla, se puede apreciar que la búsqueda local que produce menores costos es la *Optima*, contra la *CriticaAleatoria* y la *CriticaTotal* con un porcentaje de diferencia de 14 % y 4 % respectivamente. Con respecto al tiempo promedio empleado, *Optima* y *CriticaAleatoria* consumen una cantidad de tiempo similar, siendo la búsqueda local *CriticaAleatoria* la que consume menor tiempo, ahorrando un 4 % y 35 % para las búsquedas *Optima* y *CriticaTotal* respectivamente. Se generó una lista con las mejores soluciones producidas por el algoritmo con las diversas búsquedas locales para calcular el porcentaje de error de cada una de las experimentaciones. Estos porcentajes de error muestran que la búsqueda local *Optima* obtiene menor margen de error con respecto a la búsqueda local *CriticaTotal* y *CriticaAleatoria*.

Búsqueda Local	Valor Obj	Tiempo	% Error
Crítica Aleatoria	89874.19	127.46	142.11 %
Crítica Total	82025.26	173.33	77.08 %
Óptima	78761.67	133.11	52.37 %

Tabla 5.4: Resultados de la experimentación 4

En la Tabla 5.5, encontramos el ranking producido por la prueba de Friedman [19] realizada con el software de la Universidad de Granada [9]. La prueba de

Friedman nos indica el desempeño de los algoritmos: a mayor valor en el ranking, mejor desempeño posee el algoritmo.

Para una confiabilidad del 95 %, se establece un α de 0.05, y para que la prueba sea estadísticamente significativa, debe ocurrir que el valor $p - value$ calculado por la prueba de Friedman sea igual o menor que α. Dado que el valor de $p - value$ resulta ser $6.197E^{-}11$, dicho valor es menor al valor de α; por tanto, se puede establecer que existe una diferencia significativa entre los tres algoritmos presentados.

Búsqueda Local	Ranking
Crítica Total	2.10344828
Crítica Aleatoria	1.09195402
Óptima	2.8045977

Tabla 5.5: Resultados de la prueba de Friedman

Capítulo 6

Conclusiones y trabajos futuros

En este proyecto de tesis se abordó el problema de SUMCUT, el cual es un problema de optimización NP-duro con importantes aplicaciones en diferentes áreas del conocimiento. El propósito de este trabajo fue desarrollar nuevas estrategias de búsqueda local, las cuales permitieran el desarrollo de soluciones metaheurísticas del problema.

La contribución más importante de este trabajo es la reducción de la complejidad de las búsquedas locales de $O(n^3)$ a $O(n^2)$, como consecuencia de la incorporación de un mecanismo de predicción del costo de una inserción consecutiva. Este mecanismo incorpora cinco diferentes búsquedas locales las cuales fueron evaluadas en un algoritmo de búsqueda local iterada con procesamiento celular. Para las instancias más grandes el benchmark estándar, los resultados experimentales muestran que estas estrategias logran reducir en al menos un 90 % del tiempo de ejecución del algoritmo sin reducir la calidad de las soluciones (Ver capítulo 4 y sección 5.1.1).

Resultados parciales de este trabajo se incorporaron en el artículo "Estrategias de búsqueda local para el problema de SUMCUT", el cual fue publicado en el *VI Encuentro de Investigadores de Posgrado en el área de Ciencias Computacionales* realizado en el Instituto Tecnológico de Ciudad Madero los días del 10 al 14 de diciembre del 2012.

Algunos trabajos futuros con los que se puede continuar esta investigación es el desarrollo de soluciones metaheurísticas que incorporen las búsquedas locales en este trabajo.

Apéndice A

Mejores Conocidos

En este anexo se presenta la tabla de los mejores conocidos para los conjuntos de instancia Small, Grid y Harwell-Boeing. La Tablas A.1, A.2 y A.3 estan construidas de la siguiente manera: En la primera columna se encuentra el nombre de la instancia, en la segunda y tercera columna se muestran el valor de los vértices y aristas respectivamente para dicha instancia, la cuarta columna indica el mejor valor de SC encontrado, y finalmente la última columna indica si dicho valor de SC es el óptimo. Si en dichas tablas, el nombre de una instancia se encuentra en negrita, significa que el valor del SC para esa instancia en particular es el valor óptimo.

La instancia *Grid3x3* de la Tabla A.3 fue calculada con un algoritmo exacto basado en el enumerativo de Bertacco [1]. El conjunto de instancias Small fue resuelto con Gurobi [14].

Tabla A.1: Mejores conocidos para el SUMCUT para el conjunto Harwell-Boeing

Nombre de la Instancia	No. de Vértices	No. de Aristas	Mejor Valor de SC
494_bus.mtx.rnd	494	586	12150
662_bus.mtx.rnd	662	906	22806
685_bus.mtx.rnd	685	1282	11510
Continua en la siguiente pagina			

Tabla A.1 – continúa de la página anterior

Nombre de la Instancia	No. de Vértices	No. de Aristas	Mejor Valor de SC
arc130.mtx.rnd	130	715	15120
ash292.mtx.rnd	292	958	6439
ash85.mtx.rnd	85	219	956
bcspwr01.mtx.rnd	39	46	106
bcspwr02.mtx.rnd	49	59	162
bcspwr03.mtx.rnd	118	179	732
bcspwr04.mtx.rnd	274	669	4894
bcspwr05.mtx.rnd	443	590	8266
bcsstk01.mtx.rnd	48	176	1132
bcsstk02.mtx.rnd	66	2145	47905
bcsstk04.mtx.rnd	132	1758	29812
bcsstk05.mtx.rnd	153	1135	11059
bcsstk06.mtx.rnd	420	3720	61855
bcsstk20.mtx.rnd	467	1295	7038
bcsstk22.mtx.rnd	110	254	958
bcsstm07.mtx.rnd	420	3416	55852
can_144.mtx.rnd	144	576	3224
can_161.mtx.rnd	161	608	6696
can_292.mtx.rnd	292	1124	17695
can_445.mtx.rnd	445	1682	35706
curtis54.mtx.rnd	54	124	454
dwt_209.mtx.rnd	209	767	6425
dwt_221.mtx.rnd	221	704	3925
dwt_234.mtx.rnd	117	162	966
dwt_245.mtx.rnd	245	608	4339
dwt_310.mtx.rnd	310	1069	6454
Continua en la siguiente pagina			

Tabla A.1 – continúa de la página anterior

Nombre de la Instancia	No. de Vértices	No. de Aristas	Mejor Valor de SC
dwt_361.mtx.rnd	361	1296	12072
dwt_419.mtx.rnd	419	1572	15641
dwt_503.mtx.rnd	503	2762	41759
dwt_592.mtx.rnd	592	2256	25051
fs_183_1.mtx.rnd	183	701	17202
fs_541_1.mtx.rnd	541	2466	97940
fs_680_1.mtx.rnd	680	1464	16559
gent113.mtx.rnd	104	549	6233
gre_216a.mtx.rnd	216	660	7868
gre_115.mtx.rnd	115	267	2582
gre_185.mtx.rnd	185	650	6072
gre_343.mtx.rnd	343	1092	17275
gre_512.mtx.rnd	512	1680	34056
hor_131.mtx.rnd	434	2138	36500
ibm32.mtx.rnd	32	90	485
impcol_a.mtx.rnd	206	557	6322
impcol_b.mtx.rnd	59	281	2076
impcol_c.mtx.rnd	137	352	4057
impcol_d.mtx.rnd	425	1267	18780
impcol_e.mtx.rnd	225	1187	15597
lns_131.mtx.rnd	123	275	2330
lns_511.mtx.rnd	503	1425	26047
lund_a.mtx.rnd	147	1151	11326
lund_b.mtx.rnd	147	1147	11192
mbeacxc.mtx.rnd	487	41686	4380504
mcca.mtx.rnd	168	1662	23018
Continua en la siguiente pagina			

Tabla A.1 – continúa de la página anterior

Nombre de la Instancia	No. de Vértices	No. de Aristas	Mejor Valor de SC
nnc261.mtx.rnd	261	794	9170
nnc666.mtx.rnd	666	2148	37839
nos1.mtx.rnd	158	312	624
nos2.mtx.rnd	638	1272	2544
nos4.mtx.rnd	100	247	1031
nos5.mtx.rnd	468	2352	55776
nos6.mtx.rnd	675	1290	11328
plat362.mtx.rnd	362	2712	36570
plskz362.mtx.rnd	362	880	7074
pores_1.mtx.rnd	30	103	383
pores_3.mtx.rnd	456	1769	383
saylr1.mtx.rnd	238	445	3127
saylr3.mtx.rnd	681	1373	22704
sherman4.mtx.rnd	546	1341	15039
shl_200.mtx.rnd	663	1720	134931
shl_400.mtx.rnd	663	1709	138264
shl_0.mtx.rnd	663	1682	129333
steam1.mtx.rnd	240	1761	28508
steam2.mtx.rnd	600	6580	161692
steam3.mtx.rnd	80	424	1416
str_200.mtx.rnd	363	3049	104797
str_600.mtx.rnd	363	3244	118242
str_0.mtx.rnd	363	2446	73086
west0132.mtx.rnd	132	404	4932
west0156.mtx.rnd	156	371	5619
west0167.mtx.rnd	167	489	6517
Continua en la siguiente pagina			

Tabla A.1 – continúa de la página anterior

Nombre de la Instancia	No. de Vértices	No. de Aristas	Mejor Valor de SC
west0381.mtx.rnd	381	2150	125189
west0479.mtx.rnd	479	1889	77486
west0497.mtx.rnd	497	1715	47415
west0655.mtx.rnd	655	2841	166071
will199.mtx.rnd	199	660	17850
will57.mtx.rnd	57	127	335

Tabla A.2: Mejores conocidos para el SUMCUT para el conjunto Small

Nombre de la Instancia	No. de Vértices	No. de Aristas	Mejor Valor de SC
p17_16_24	16	24	70
p18_16_21	16	21	48
p19_16_19	16	19	39
p20_16_18	16	18	36
p21_17_20	17	20	44
p22_17_19	17	19	37
p23_17_23	17	23	51
p24_17_29	17	29	82
p25_17_20	17	20	40
p26_17_19	17	19	40
p27_17_19	17	19	39
p28_17_18	17	18	32
p29_17_18	17	18	29
p30_17_19	17	19	39
p31_18_21	18	21	39
p32_18_20	18	20	46
Continua en la siguiente pagina			

Tabla A.2 – continúa de la página anterior

Nombre de la Instancia	No. de Vértices	No. de Aristas	Mejor Valor de SC
p33_18_21	18	21	47
p34_18_21	18	21	39
p35_18_19	18	19	35
p36_18_20	18	20	44
p37_18_20	18	20	48
p38_18_19	18	19	32
p39_18_19	18	19	36
p40_18_32	18	32	93
p41_19_20	19	20	36
p42_19_24	19	24	57
p43_19_22	19	22	42
p44_19_25	19	25	70
p45_19_25	19	25	56
p46_19_20	19	20	37
p47_19_21	19	21	39
p48_19_21	19	21	41
p49_19_22	19	22	45
p50_19_25	19	25	51
p51_20_28	20	28	73
p52_20_27	20	27	70
p53_20_22	20	22	47
p54_20_28	20	28	68
p55_20_24	20	24	51
p56_20_23	20	23	54
p57_20_24	20	24	56
p58_20_21	20	21	41
Continua en la siguiente pagina			

Tabla A.2 – continúa de la página anterior

Nombre de la Instancia	No. de Vértices	No. de Aristas	Mejor Valor de SC
p59_20_23	20	23	50
p60_20_22	20	22	47
p61_21_22	21	22	44
p62_21_30	21	30	91
p63_21_42	21	42	148
p64_21_22	21	22	45
p65_21_24	21	24	51
p66_21_28	21	28	77
p67_21_22	21	22	40
p68_21_27	21	27	68
p69_21_23	21	23	53
p70_21_25	21	25	57
p71_22_29	22	29	69
p72_22_49	22	49	195
p73_22_29	22	29	65
p74_22_30	22	30	67
p75_22_25	22	25	55
p76_22_30	22	30	73
p77_22_37	22	37	122
p78_22_31	22	31	79
p79_22_29	22	29	67
p80_22_30	22	30	74
p81_23_46	23	46	179
p82_23_24	23	24	56
p83_23_24	23	24	48
p84_23_26	23	26	52
Continua en la siguiente pagina			

Tabla A.2 – continúa de la página anterior

Nombre de la Instancia	No. de Vértices	No. de Aristas	Mejor Valor de SC
p85_23_26	23	26	50
p86_23_24	23	24	43
p87_23_30	23	30	83
p88_23_26	23	26	58
p89_23_27	23	27	71
p90_23_35	23	35	100
p91_24_33	24	33	96
p92_24_26	24	26	58
p93_24_27	24	27	55
p94_24_31	24	31	84
p95_24_27	24	27	59
p96_24_27	24	27	53
p97_24_26	24	26	57
p98_24_29	24	29	71
p99_24_27	24	27	62
p100_24_34	24	34	98

Tabla A.3: Mejores conocidos para el SUMCUT para el conjunto Grid

Nombre de la Instancia	No. de Vértices	No. de Aristas	Mejor Valor de SC
Grid3x3	9	12	24
Grid3x6	18	27	57
Grid3x9	27	42	90
Grid3x12	36	57	123
Grid3x15	45	72	156
Grid3x18	54	87	189
Continua en la siguiente pagina			

Tabla A.3 – continúa de la página anterior

Nombre de la Instancia	No. de Vértices	No. de Aristas	Mejor Valor de SC
Grid3x21	63	102	222
Grid3x24	72	117	255
Grid3x27	81	132	288
Grid6x3	18	27	57
Grid6x6	36	60	200
Grid6x9	54	93	323
Grid6x12	72	126	446
Grid6x15	90	159	569
Grid6x18	108	192	692
Grid6x21	126	225	815
Grid6x24	144	258	938
Grid6x27	162	291	1061
Grid9x3	27	42	2270
Grid9x6	54	93	90
Grid9x9	81	144	323
Grid9x12	108	195	668
Grid9x15	135	246	935
Grid9x18	162	297	1202
Grid9x21	189	348	1469
Grid9x24	216	399	1736
Grid9x27	243	450	2003
Grid12x3	36	57	123
Grid12x6	72	126	446
Grid12x9	108	195	935
Grid12x12	144	264	1575
Grid12x15	180	333	2037
Continua en la siguiente pagina			

Tabla A.3 – continúa de la página anterior

Nombre de la Instancia	No. de Vértices	No. de Aristas	Mejor Valor de SC
Grid12x18	216	402	2506
Grid12x21	252	471	2964
Grid12x24	288	540	3441
Grid12x27	324	609	3903
Grid15x3	45	72	156
Grid15x6	90	159	569
Grid15x9	135	246	1202
Grid15x12	180	333	2048
Grid15x15	225	420	3048
Grid15x18	270	507	3810
Grid15x21	315	594	4482
Grid15x24	360	681	5212
Grid15x27	405	768	5980
Grid18x3	54	87	189
Grid18x6	108	192	692
Grid18x9	162	297	1470
Grid18x12	216	402	2535
Grid18x15	270	507	3875
Grid18x18	324	612	5399
Grid18x21	378	717	6445
Grid18x24	432	822	7320
Grid18x27	486	927	8418
Grid21x3	63	102	222
Grid21x6	126	225	815
Grid21x9	189	348	1742
Grid21x12	252	471	2984
Continua en la siguiente pagina			

Tabla A.3 – continúa de la página anterior

Nombre de la Instancia	No. de Vértices	No. de Aristas	Mejor Valor de SC
Grid21x15	315	594	4539
Grid21x18	378	717	6444
Grid21x21	441	840	8469
Grid21x24	504	963	9964
Grid21x27	567	1086	11093
Grid24x3	72	117	255
Grid24x6	144	258	938
Grid24x9	216	399	2015
Grid24x12	288	540	3475
Grid24x15	360	681	5299
Grid24x18	432	822	7478
Grid24x21	504	963	9972
Grid24x24	576	1104	12629
Grid24x27	648	1245	14941
Grid27x3	81	132	288
Grid27x6	162	291	1061
Grid27x9	243	450	2276
Grid27x12	324	609	3964
Grid27x15	405	768	6039
Grid27x18	486	927	8512
Grid27x21	567	1086	11279
Grid27x24	648	1245	14483
Grid27x27	729	1404	18085

Bibliografía

[1] Bertacco, L., Brunetta, L., Fischetti, M.: The linear ordering problem with cumulative costs (2004)

[2] Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Computing Surveys **Vol. 35**(No. 3), pp. 268–308 (2003)

[3] Congram, R.: Polynomially searchable exponiential neighbourhoods for sequencing problems in combinatorial optimization. Faculty of mathematical studies, University of Southampton (2000)

[4] Cormen, H., Stein, C., Rivest, R., Leiserson, C.: Introduction to Algorithms, 2nd edn. McGraw-Hill Higher Education (2001)

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

[6] Cuthill, E., McKee, J.: Reducing the bandwidth of sparse symmetric matrices. Proceedings of the 1969 24th national conference **ACM'69**, 157–172 (1969)

[7] Díaz, J., Maria, S., Jordi, P.: A Survey of Graph Layout Problems (2002)

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

[9] García, S., Herrera, F.: An Extension on "Statistical Comparisons of Classifiers over Multiple Data Sets"for all Pairwise Comparisons.

Journal of Machine Learning Research **9**, 2677–2694 (2008). URL http://www.jmlr.org/papers/volume9/garcia08a/garcia08a.pdf

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

[11] Gendreau, M., Potvin, J.: Handbook of Metaheuristics. International Series in Operations Research and Management Science. Springer (2010). URL http://books.google.com.mx/books?id=xMTS5dyDhwMC

[12] George, J., Liu, J.: Computer solution of large sparse positive definite systems. Prentice-Hall series in computational mathematics (1981)

[13] Gibbs, N.E., Poole, W.G., Stockemeyer, P.K.: An algorithm for reducing the bandwidth and profile of a sparse matrix. SIAM Journal of Numerical Analysis **13(2)**, 236–250 (1976)

[14] Gurobi Optimization, I.: Gurobi optimizer reference manual (2012). URL http://www.gurobi.com

[15] Heggernes, P., Paulz, C., Telley, J., Villangery, Y.: Interval completion with few edges (2007)

[16] Karp, R.: Mapping the genome: some combinatorial problems arising in molecular biology. Proceedings of the twenty-

fth annual ACM symposium on Theory of computing (STOC 93), 278–285 (1993)

[17] Kendall, D.: Incidence matrices, interval graphs and seriation in archeology. Pacific J. Math **28, Number 3 (1969)**(3), 565–570 (1969)

[18] Kuo, D., Chang, G.J.: The profile minimization problem in trees. SIAM Journal on computing pp. 23:71–81 (1994)

[19] Kvam., P.H., Vidakovic, B.: Nonparametric Statistics With Applications to Science and Engineering. Wiley Series in Probability and Statistics. Wiley (2007)

[20] Laguna, M., Martí, R., Campos, V.: Intensification and diversification with elite tabu search solutions for the linear ordering problem (1998)

[21] Land, A., Doig, A.: An automatic method of solving discrete programming problems. Econometrica **28**(3), 497–520 (1960)

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

[23] Luke, S.: Essentials of Metaheuristics. Lulu (2009). Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/

[24] Martí, R., Campos, V., Piñana, E.: Branch and bound for the matrix bandwidth minimization. European Journal of Operational Research **13**, 513–528 (2008)

[25] Michalewicz, Z., Fogel, D.: How to solve it: Modern heuristics (1999)

[26] Pantrigo, J., Martí, R., Duarte, A., Pardo, E.: Scatter search for the cutwidth minimization problem (2010)

[27] Petit, J.: Addenda to the survey of layout problems. Bulletin of the EATCS **105**, 177–201 (2011)

[28] Piñana, E., Plana, I., V.Campos, Martí, R.: Grasp and path relinking for the matrix bandwidth minimization (2001)

[29] Rolim, J., Sýkora, O., Vrt'o., I.: Cutwidth of the de bruijn graph. RAIRO Informatique Théorique et Applications **29**(6), 509–514 (1995)

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

[31] Sanchez-Oro, J., Duarte, A.: Grasp con path relinking para el problema del sumcut. Congreso Maeb 2012 (2012)

[32] Terán, D., Fraire, H., Carpio, J., Pazos, R., Puga, H., Martínez, J.: Experimental study of a new algorithm-design-framework based on a cellular computing. Recent Advances on Hybrid Intellight Systems; Studies in Computational Intelligence **451/2013**, 517–532 (2013)

[33] Terán-Villanueva, J., Pazos, R., Martínez, J., Lopez-Loces, M., Zamarrón, D., Santiago, A.: Hybrid grasp with composite local search and path-relinking for the linear ordering problem with cumulative costs. International Journal of Combinatorial Optimization Problems and Informatics **3**(1) (2011)

[34] W., L., Sherman, A.: Comparative analysis of the cuthill-mckee and reverse cuthill-mckee ordering algorithms for sparse matrices. Tech. Rep. 28, University of Waterloo, Yale University