INSTITUTO TECNOLÓGICO DE CD. MADERO

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

Solución del LOPCC Utilizando Estrategias de Diversificación e Intensificación en Búsqueda Tabú

Tesis
Para Obtener el Grado de
Maestro en Ciencias en Ciencias de la Computación

Presenta:
I.S.C. Jorge Alberto Curiel Moreno

Director:
Dr. Juan Javier González Barbosa.

Codirector:
Dr. Héctor J. Fraire Huacuja

CD. MADERO, TAMPS. JUNIO 2012

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

SECRETARÍA DE
EDUCACIÓN PÚBLICA

Cd. Madero, Tamps; a 13 de Junio de 2012.

OFICIO No.: U5.115/12
AREA: DIVISIÓN DE ESTUDIOS
DE POSGRADO E INVESTIGACIÓN
ASUNTO: AUTORIZACIÓN DE IMPRESIÓN
DE TESIS

C. ING. JORGE ALBERTO CURIEL MORENO
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:

"SOLUCIÓN DEL LOPCC UTILIZANDO ESTRATEGIAS DE DIVERSIFICACIÓN E INTENSIFICACIÓN EN BÚSQUEDA TABÚ "

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 ' jar

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

Declaración de Originalidad

Declaro y prometo que éste 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 las publicaciones.

Además, en caso de infracción de los derechos de terceros derivados de éste 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 Cd. Madero y sus autoridades.

20 de Junio de 2012, Cd. Madero, Tamps.

I.S.C. Jorge Alberto Curiel Moreno

Resumen

Hoy en día los procesos de comunicación son inherentes casi a cada momento de nuestra vida, las tecnologías de comunicación han avanzado tanto que se cuenta con medios de comunicación casi en cualquier lugar. En este contexto, una tarea de suma importancia es la optimización en el Sistema Universal de Telecomunicaciones Estándar. Cuyo objetivo principal es, mediante técnicas de reducción de interferencia, garantizar beneficios tales como minimización de la energía requerida para la transmisión y una recepción adecuada para todos los usuarios, mediante la eliminación de la interferencia provocada por las mismas terminales.

Para eliminar dicha interferencia se han creado algunas técnicas denominadas de reducción de interferencia. La cancelación sucesiva de interferencia (SIC) es una de estas. La SIC detecta las terminales móviles (MTs) dado un orden predeterminado y remueve la interferencia asociada, mejorando la comunicación para los siguientes usuarios.

Un problema crucial en el diseño del sistema de cancelación sucesiva de interferencia es, por lo tanto, la elección del orden de detección, el cual se puede observar como una aplicación real del problema de Ordenamiento Lineal con Costos Acumulados (LOPCC por sus siglas en inglés).

Summary

Today communication processes are inherent to almost every moment of our life, the communication technologies are advanced enough that the media are in pretty much anywhere. In this context, a task of great importance is the optimization at the Universal Mobile Telecommunications System Standard. Whose main purpose is ensure benefits such as minimization of the energy required for transmission and a reception appropriate for all users, trough techniques for reducing the generated interference from the terminals itself.

For this purpose, ie to eliminate the interference some techniques have been created and are known as techniques of interference reduction. Successive interference cancellation (SIC) is one of these. This technique consists in detecting the MTs given a predetermined order and removes the associated interference, improving communication for the following users.

A crucial problem at the design of successive interference cancellation system is, therefore, the choice of the order of detection, which can be observed as an real application the problem of costs Planning Linear with Accumulated (LOPCC).

Contenido

Listado de figuras

Listado de Tablas

CAPÍTULO 1

1. Introducción

En este capítulo de introducción se abordan los motivos y las razones que dieron origen a este proyecto. Se ilustran los antecedentes del proyecto, la definición del problema de investigación, la justificación que respalda el proyecto, los objetivos planteados en un inicio, así como los alcances y limitaciones de la presente investigación.

1.1 Antecedentes

Hoy en día los procesos de comunicación son inherentes casi a cada momento de nuestra vida, las tecnologías de comunicación han avanzado tanto que se cuenta con medios de comunicación casi en cualquier lugar. Dentro de esta área un punto muy importante es la comunicación del tipo móvil, comúnmente llamada comunicación celular.

En las comunicaciones móviles, las terminales móviles (MTs, Mobile Terminals) se comunican simultáneamente con una base común. Para distinguir las señales originadas por las diferentes MTs, el Estandar Universal de Telecomunicaciones Moviles (UMTS, Universal Mobile Telecommunications System) adopta la técnica de acceso multiple por división de código (CDMA, Code Division Multiple Access), donde cada MT es identificada por un código especifico. En situaciones reales, las MTs interfieren entre sí, debido a las distorsiones creadas por la propagación de las ondas de radio, de ahí la necesidad de mantener dicha interferencia en un nivel aceptable. Para lograr este objetivo se han creado algunas técnicas de reducción de interferencia. Una de estas técnicas recibe el nombre de cancelación sucesiva de interferencia (SIC, Successive Interference Cancellation). La SIC detecta las señales de las MTs dado un orden predeterminado,

después de esto remueve la interferencia asociada, mejorando la comunicación para los siguientes usuarios.

Como se puede observar un problema crucial en el diseño del sistema de cancelación sucesiva de interferencia es la elección del orden de detección. Usualmente, los usuarios son ordenados decrecientemente de acuerdo a la potencia recibida, aunque un mejor desempeño se puede obtener considerando además al nivel de interferencia mutua entre los usuarios. Una segunda problemática es la elección del nivel de potencia , en la que el *i*-esimo usuario tiene que transmitir sus datos.

El problema de la selección del orden de detección en la SIC, se puede visualizar como una aplicación real de un problema de optimización conocido como el problema de Ordenamiento Lineal con Costos Acumulados (LOPCC).

1.2 Descripción del Problema de Investigación

En el contexto de la optimización en sistemas UMTS, en especifico en la técnica SIC, el problema LOPCC consiste en encontrar una configuración u orden de detección óptimo para el SIC, que garantice la minimización de la energía requerida para la transmisión y una recepción adecuada para todos los usuarios .

En este contexto podemos definir al LOPCC de la siguiente manera: sea d_i como el nivel de energía con el que el usuario *i* debe transmitir sus datos y c_{ij} como la interferencia generada de el usuario *i* para el usuario *j*, la solución optima del LOPCC asociado, minimiza la energía de transmisión (también maximiza la duración de las baterías de las MTs) mientras asegura una recepción apropiada para todos los usuarios [Duarte 2006].

1.3 Objetivos

El objetivo general de este proyecto será el evaluar técnicas de diversificación e intensificación de búsqueda tabú en el contexto del problema del problema de ordenamiento lineal con costos acumulados.

En cuanto a los objetivos específicos del proyecto estos serán los siguientes:

- Analizar la solución tabú de referencia [Duarte 2006].
- Implementar la solución de referencia de tal manera que se obtengan resultados muy similares a esta, en cuanto a calidad y esfuerzo computacional.
- Definir estrategias de mejora a partir de la solución de referencia.
- Implementar estrategias de mejora.
- Evaluación experimental de estrategias de mejora en la solución propuesta

1.4 Justificación del Proyecto

Como se puede observar en el estado del arte y pese a sus aplicaciones prácticas y complejidad, el problema de LOPCC no ha sido tan estudiado como otros problemas de optimización, como el problema de TSP el cual ha sido abordado en múltiples ocasiones por un gran número de investigadores. Además, como se ha visto el problema LOPCC es un problema NP-Duro, por lo cual abordarlo con un método convencional no resulta factible, es por eso que una solución metaheurística resulta ideal para abordarlo.

En la literatura se encontraron pocos artículos referentes a este problema y solo uno de ellos utilizó un procedimiento metaheurístico para intentar resolver el problema [Duarte

2006]. En ese trabajo, el cual es la solución de referencia de esta propuesta, se propuso una búsqueda tabú.

La búsqueda tabú propuesta en el trabajo de [Duarte 2006], además, es susceptible de mejorarse incorporando algunas de las estrategias que la búsqueda tabú proporciona y que no fueron incluidas.

1.5 Organización de la Tesis

En esta sección se describe como está organizado éste documento, así como la información que contiene cada uno de los capítulos.

En el capítulo 2, se presentan los fundamentos teóricos que sustentan el trabajo de investigación, entre otras cosas se define el problema de investigación denominado LOPCC, presentándolo como un problema complejo, mediante la teoría de NP Completez.

En capítulo 3, muestra los trabajos que han abordado hasta ahora al LOPCC, además se realiza una comparación entre estos, haciendo énfasis en sus características.

El capítulo 4, presenta la metodología de investigación que se siguió en el presente proyecto.

En el capítulo 5, aborda el diseño experimental, estableciendo un análisis comparativo de soluciones respecto al estado del arte de LOPCC.

Finalmente, la sección 6 aborda las conclusiones generadas al finalizar este proyecto.

CAPÍTULO 2

2. Marco Conceptual

En este capítulo se abordan los conceptos que base en el presente proyecto. En ellos se fundamenta la definición del LOPCC, así como su complejidad. Además se muestra un estudio acerca de los métodos de solución aplicados a problemas de optimización, con un énfasis en las técnicas metaheurísticas de optimización y en especial en la técnica denominada búsqueda tabú.

2.1 El Problema de Ordenamiento Lineal

El LOPCC es una variante del problema de Ordenamiento Lineal (LOP, Linear Ordering Problem) por lo cual es conveniente abordarlo primero. Dicho problema consiste en que dada una matriz de pesos $C=\{c_{ij}\}$, se maximice la siguiente expresión:

$$C_{LOP}(p) = \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} c_{p_i p_j}$$

donde p_i es el índice de la columna (y la fila) en la posición i en la permutación.

En el problema de ordenamiento lineal la permutación p proporciona el ordenamiento para las columnas y las filas. El LOP es equivalente al problema de encontrar, en un grafo completo con pesos, un circuito acíclico con una suma máxima de los pesos de los arcos [Reinelt 1985].

En economía, el LOP es equivalente al problema denominado problema de triangulación de tablas de entrada-salida. En este problema la economía de una región es dividida en *m* sectores y una tabla *C* de tamaño *m*x*m* es construida, donde cada entrada c_{ij} denota la cantidad de entregas del sector *i* al sector *j* en un año especifico. Entonces el problema de triangulación consiste en permutar las filas y columnas de *C* simultáneamente buscando que la suma de las entradas encima de la diagonal sea lo más grande posible [Laguna, et al. 1999].

2.2 El problema de Ordenamiento Lineal con Costos Acumulados

El problema de Ordenamiento Lineal con Costos Acumulados(LOPCC, Linear Ordering Problem with Cummulative Costs) fue introducido por Bertacco en [Bertacco, et al. 2005], como una variante del problema de Ordenamiento Lineal. Donde, dado un dígrafo con pesos no-negativos d_i y costos no-negativos c_{ij} en sus arcos, el objetivo del LOPCC es encontrar un circuito hamiltoniano $p=(p_1, p_2, \ldots, p_n)$ que minimice la siguiente expresión:

$$C_{LOPCC}(p) = \sum_{i=+1}^{n} \alpha_i$$

donde:

$$\alpha_{p_i} = d_{p_i} + \sum_{j=i+1}^{n} c_{p_i p_j} \alpha_{p_j} \qquad \text{para } i=n, n-1, \ldots, 1$$

Ejemplo LOP

Considerando una matriz de coeficientes como la siguiente:

$$C = \begin{pmatrix} 0 & 2 & 1 & 3 \\ 4 & 0 & 1 & 5 \\ 2 & 3 & 0 & 5 \\ 1 & 2 & 1 & 0 \end{pmatrix}$$

Fig. 1 Matriz de coeficientes del ejemplo de problema LOP

El valor de la función objetivo de la solución *p*=(1,2,3,4) para el LOP correspondiente es igual:

Ejemplo LOPCC

Si se toma la instancia anterior y se considera ahora como una instancia de LOPCC, con la matriz de coeficientes anterior y con pesos de los nodos iguales a *d=(3, 2, 1, 4)*. Entonces el valor de la función objetivo de la solución *p=(1,2,3,4)* se calcula de la siguiente forma:

$$C_{LOPCC}(p) = \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4$$

donde:

$$\alpha_4 = d_4 = 4$$

$$\alpha_3 = d_3 + c_{34}\alpha_4 = 1 + 5*4 = 21$$

$$\alpha_2 = d_2 + c_{23}\alpha_3 + c_{24}\alpha_4 = 2 + 1*21 + 5*4 = 43$$

$$\alpha_1 = d_1 + c_{12}\alpha_2 + c_{13}\alpha_3 + c_{14}\alpha_4 = 3 + 2*43 + 1*21 + 3*4 = 122$$

Entonces:

$$C_{LOPCC}(p) = 122 + 43 + 21 + 4 = 190$$

2.2.1 UMTS visto como LOPCC

En el contexto de los sistemas UMTS se puede definir al LOPCC si tomamos a d_i como el nivel de energía con el que el usuario *i* debe transmitir sus datos y c_{ij} como la interferencia generada de el usuario *i* para el usuario *j*, la solución optima del LOPCC asociado, minimiza la energía de transmisión (también maximiza la duración de las baterías de las MTs) mientras asegura una recepción apropiada para todos los usuarios [Duarte 2006].

2.3 Definición Formal del LOPCC

Dado un dígrafo con pesos no-negativos d_i y costos no-negativos c_{ij} en sus arcos, el objetivo del LOPCC es encontrar un circuito hamiltoniano $p=(p_1, p_2, \ldots, p_n)$ que minimice la expresión:

$$C_{LOPCC}(p) = \sum_{i=1}^{n} \alpha_i$$

donde:

$$\alpha_{p_i} = d_{p_i} + \sum_{j=i+1}^{n} c_{p_i p_j} \alpha_{p_i} \qquad \text{para} \quad i = n, n-1, \ldots, 1.$$

2.4 Métodos de Solución de Problemas de Optimización

En los problemas de optimización como el LOPCC, se pueden aplicar dos enfoques para intentar darles solución: el primero es recurrir a métodos exactos y el segundo es abordar el problema por medio de alguna heurística o método aproximado.

2.4.1 Métodos exactos

El primer enfoque para abordar cualquier tipo de problema es el uso de métodos exactos, que garanticen el llegar a la solución óptima del problema. En cuanto a los métodos exactos más utilizados podemos encontrar: la búsqueda exhaustiva, el método simplex, ramificación y acotamiento. El primer método se basa en evaluar la totalidad del universo de soluciones factibles, volviéndose un método sumamente costoso y solo aplicable en problemas de tamaño reducido. El segundo método realiza una búsqueda en los puntos adyacentes del cerco convexo del espacio de soluciones factibles del problema, es decir, evalúa solo los extremos en el espacio de soluciones factibles del problema. El tercero, realizando podas al árbol de búsqueda, observando cada una de las rutas parciales y si alguna de ellas es peor a la mejor solución encontrada no hay razón para seguir explorando ese camino. A pesar de que los métodos exactos obtienen el resultado optimo, su desempeño dista mucho de lo aceptable, lo que los vuelve inaplicables en la realidad. Es aquí donde los algoritmos heurísticos surgen como una alternativa a los métodos exactos, volviéndose una alternativa de calidad aceptable, pero con la ventaja de un bajo costo computacional.

2.4.2 Métodos heurísticos

Debido a la dificultad de resolver de forma exacta problemas del tipo combinatorio, aparecieron las heurísticas como una forma de obtener resultados de aceptables. Los métodos heurísticos son procedimientos basados en el sentido común, que ofrecen buenas soluciones a problemas difíciles, de un modo fácil y rápido [Díaz, et al. 1996]. Son varios los factores que pueden motivar a utilizar alguna heurística en la solución de problemas del tipo combinatorio, entre los cuales están:

- No existe un método exacto de resolución o este requiere el uso de recursos computacionales de una manera inaceptable.

- No se requiere una solución óptima. Si el beneficio de encontrar una solución óptima no representa una diferencia importante respecto a una solución sub-óptima.
- Cuando los datos son poco fiables.
- Existen limitaciones de recursos computacionales.
- Como pre-procesamiento en algún otro algoritmo.

Los algoritmos heurísticos se pueden diferenciar de acuerdo al modo en como buscan y construyen sus soluciones. Una posible clasificación de los métodos heurísticos puede ser la siguiente:

a) Métodos Constructivos. Realizan una construcción paulatina de la solución, agregando de acuerdo a cierto criterio sus componentes. Los algoritmos golosos son ejemplo de este tipo de algoritmos.

b) **Métodos de Descomposición.** Consisten en dividir el problema en subproblemas más pequeños, resolverlos y combinar sus soluciones para así obtener la solución al problema global.

c) **Métodos de Reducción.** Consiste en identificar una característica que presumiblemente debe tener la solución óptima, y de esta manera simplificar el problema.

d) **Manipulación del Modelo.** Consiste en modificar el modelo, disminuir sus restricciones, agregar restricciones, agrupar variables, etc.

e) **Métodos de búsqueda por entornos o vecindades.** En este tipo entran la mayoría de las metaheurísticas. Parten de una solución factible, y mediante

alteraciones a esta solución, pasan de esa solución a otras factibles en su entorno (vecindad), almacenando como optima la mejor de las soluciones visitadas.

Para poder generalizar los métodos heurísticos a una variedad de problemas, surgió el termino denominado metaheurística, término introducido por primera vez por Fred Glover [Glover, 1986].

Los algoritmos metaheurísticos son algoritmos aproximados de optimización de propósito general. Se caracterizan por que utilizan procedimientos iterativos que guían una heurística subordinada combinando de forma inteligente distintos conceptos para explorar y explotar adecuadamente el espacio de búsqueda. Una de las principales ventajas de las metaheurísticas, es que estas generalmente poseen algún mecanismo para evitar caer en mínimos locales, haciendo uso de estrategias inteligentes y aleatorias que permiten ampliar las posibilidades de elección.

Entre las principales metaheurísticas utilizadas en problemas de tipo combinatorio están:

- **Recocido Simulado (SA Simulated Annealing).** Es un algoritmo de búsqueda por entornos, basado en la termodinámica, utiliza un criterio probabilístico en la aceptación de soluciones [Kirkpatrick, et al. 1983].

- **Algoritmos Genéticos (GA Genetic Algorithm).** "Están basados en la mecánica de la selección natural y la genética" [Goldberg 1989]. Se apoyan en el principio de supervivencia del más apto, partiendo así de una población inicial de soluciones, haciéndola evolucionar hasta encontrar a un individuo lo suficientemente apto (mejor solución). Los algoritmos genéticos evitan caer en mínimos locales utilizando procesos de la evolución como cruzamiento y mutación.

- **GRASP(Greedy Randomized Adaptive Search Procedure).** Consta generalmente de dos etapas: una fase de construcción y otra de procedimiento de búsqueda local. En la primera etapa se construye una solución tentativa, que luego es mejorada en la segunda etapa por un procedimiento de búsqueda local o de intercambio.

- **Optimización basada en Colonia de Hormigas (ACO Ant Colony Optimization).** Esta técnica se basa en el comportamiento de algunas especies de hormigas al encontrar los caminos más cortos entre su comida y el hormiguero, en el cual las hormigas se comunican indirectamente a través de una sustancia química llamada feromona. Si otras hormigas encuentran esa feromona, lo más probable es que sigan la trayectoria con mayor presencia de feromona. En el algoritmo ACO cada hormiga es un agente artificial que intenta construir soluciones posibles explotando los rastros de feromona.

- **Búsqueda Tabú.** Se basa en un procedimiento guía de búsqueda local para evitar caer en óptimos locales, clasificando ciertos movimientos como "tabú" volviéndolos irrepetibles en cierto horizonte de tiempo.

En este documento se ahondará un poco más en cuanto a la técnica de Búsqueda Tabú, debido a que ésta es la técnica que se pretende utilizar en la evaluación de técnicas intentando mejorar la solución de referencia del LOPCC. Para una revisión más exhaustiva de la Búsqueda Tabú se puede recurrir al libro de Glover y Laguna [Glover, et al. 1997].

2.5 Heurísticas inherentes al LOP

Existen algunas heurísticas diseñadas para intentar resolver el LOP, como el LOPCC se puede ver como una variante del LOP, es coherente revisarlas para entender

mejor la naturaleza del problema. La primera de ellas es la que se describió en [Becker 1967], donde se propone una heurística basada en el cálculo de cocientes para realizar una clasificación de los sectores, este interpretación se obtuvo de la aplicación en economía del LOP.

Específicamente para cada sector $i=1, \ldots, m$ el valor del cociente es:

$$q_i = \frac{\sum_{k=1}^{m} e_{ik}}{\sum_{k=1}^{m} e_{ki}}$$

Entonces el sector con el más alto valor de q, es clasificado primero. Después de esto la columna y fila correspondiente son eliminadas de la matriz y el procedimiento se repite con los sectores restantes. El método de Becker es razonablemente eficiente y produce resultados buenos a pesar de su simplicidad.

Otro método es el que se planteó en [Chanas, et al. 1996] conocido como método CK por las iníciales de sus creadores. El método de CK está basado en la propiedad de simetría del LOP, esta propiedad indica que si una permutación $p=(p_1, p_2, \ldots, p_m)$ es una solución optima para el problema de maximización, entonces una solución optima para el problema de minimización es $p=(p_m, p_{m-1}, \ldots, p_2, p_1)$. En otras palabras, cuando la suma de los elementos por encima de la diagonal es maximizada, entonces la suma de los elementos que están debajo de la diagonal es minimizada. El método CK utiliza esta propiedad para escapar de óptimos locales. Específicamente cuando el método CK encuentra un optimo local p', el proceso es reiniciado pero desde la permutación p'', donde esta permutación es la inversa de p'.

2.6 Búsqueda Tabú

La Búsqueda tabú (TS, Tabú Search) es un procedimiento metaheurístico utilizado para guiar un algoritmo heurístico de búsqueda local con la finalidad de explorar soluciones ubicadas en otras regiones del espacio de búsqueda, evitando así la optimalidad local [Díaz, et al. 1996].

La búsqueda tabú ha logrado una amplia gama de éxitos en años recientes en aplicaciones prácticas de optimización, lo cual ha favorecido su estudio y crecimiento. Se han encontrado buenos resultados al aplicar tanto la búsqueda tabú estándar, así como, aplicando una hibridación con otros procedimientos heurísticos y metaheurísticos.

Se puede describir a la búsqueda tabú en tres etapas, durante las cuales se utilizan las estrategias de memoria adaptativa y exploración sensible. En la figura 1 se muestra el algoritmo básico de búsqueda tabú, donde se observan estas tres etapas.

*Nota…. Agregar la búsqueda local

1	Generación de la solución inicial
2	Hacer mientras no se cumpla condición de salida
3	Fase de intensificación
4	Fase de diversificación

Fig. 2 Algoritmo básico de la búqueda tabú

La primera etapa se refiere a un procedimiento de construcción de una solución inicial. Esta etapa puede incluir algún tipo de heurística o metaheurística con la finalidad de obtener una buena solución de partida. Una vez que se construye la solución inicial comienza el ciclo principal donde se alterna de manera inteligente entre dos procesos, uno de intensificación y otro de diversificación.

La segunda etapa es un proceso de intensificación el cual toma la solución actual, la cual en la primera iteración es la solución generada por el proceso de construcción, e intenta mejorarla. Este proceso de intensificación navega por la vecindad de la solución actual intentando encontrar el mejor miembro o solución, favoreciendo aquellas soluciones con atributos asociados a buenas soluciones.

En la tercera y última etapa se realiza un proceso de diversificación por medio del cual se intenta abarcar soluciones fuera de la vecindad de la solución actual, logrando visitar áreas del espacio de soluciones no exploradas y evitando de esta forma la optimalidad local.

La búsqueda tabú se basa en dos principios básicos: la memoria adaptativa y exploración sensible. La memoria adaptativa TS explota la historia del proceso de solución del problema haciendo referencia a cuatro dimensiones principales, consistentes en la propiedad de ser reciente, en frecuencia, en calidad, y en influencia [Melián, et al. 2006]. La exploración sensible en la búsqueda tabú, viene de la suposición que una mala elección estratégica puede producir mejor información sobre el espacio de soluciones que una buena selección hecha al azar.

Cuando se tiene una memoria sobre los sucesos con relación solución que se tiene del problema, al elegir una solución que no sea del todo satisfactoria, puede darnos pistas útiles sobre que modificaciones son pertinentes realizar para mejorar la solución, en un mediano plazo.

Los elementos básicos que conforman la búsqueda tabú tienen varias características importantes como son [Díaz, et al. 1996]:

Memoria Adaptativa

- *Selectividad (incluyendo olvido estratégico)*
- *Abstracción y descomposición (a través de memoria explícita y por atributos)*
- *Tiempo:*
    - *Recencia de eventos*
    - *Frecuencia de eventos*
    - *Diferenciación entre corto y largo plazo*
- *Calidad e impacto:*
    - *Atracción relativa de elecciones alternativas*
    - *Magnitud de cambios en relaciones de estructura o restricciones*
- *Contexto:*
    - *Interdependencia regional*
    - *Interdependencia estructural*
    - *Interdependencia secuencial*

Exploración sensible

- *Imposición estratégica de limitaciones e inducciones*
    - *(condiciones tabú y niveles de aspiración)*
- *Enfoque concentrado en buenas regiones y buenas características de las soluciones*
    - *(procesos de intensificación)*
- *Caracterización y exploración de nuevas regiones prometedoras*
    - *(procesos de diversificación)*
- *Patrones de búsqueda no monótonos*
    - *(oscilación estratégica)*
- *Integración y extensión de soluciones*
    - *(reencadenamiento de trayectorias)*

La búsqueda tabú tiene como objetivo encontrar nuevas y más efectivas formas de aprovechar los conceptos mostrados, para identificar conceptos aplicables a la búsqueda inteligente. Conforme se aprovechen estos principios, nuevas combinaciones de estrategias básicas surgen para lograr encontrar mejores soluciones [Glover, et al. 1997].

2.6.1 Uso de memoria

Las estructuras de memoria de la búsqueda tabú funcionan mediante referencia a cuatro dimensiones principales consistentes en: la propiedad de ser reciente, en frecuencia, en calidad, y en influencia. Las memorias basadas en lo reciente y en frecuencia se complementan la una a la otra para lograr el balance entre intensificación y diversificación que todo proceso de búsqueda heurística debe poseer. La dimensión de calidad hace referencia a la habilidad para diferenciar la bondad de las soluciones visitadas a lo largo del proceso de búsqueda. De esta forma, la memoria puede ser utilizada para la identificación de elementos comunes a soluciones buenas o a ciertos caminos que conducen a ellas. La flexibilidad de las estructuras de memoria permite guiar la búsqueda en un entorno multi-objetivo, dado que se determina la bondad de una dirección de búsqueda particular mediante más de una función. Finalmente, la cuarta dimensión de memoria, referida a la influencia, considera el impacto de las decisiones tomadas durante la búsqueda, no sólo en lo referente a la calidad de las soluciones, sino también en lo referente a la estructura de las mismas [Melián, et al. 2006].

El uso de memoria en la búsqueda tabú puede ser explícito como implícito. En el primer caso, se almacenan en memoria soluciones completas, mientras que en el segundo caso, se almacena información sobre determinados atributos de las soluciones que cambian al pasar de una solución a otra. Claramente se observa la principal limitante de la memoria explícita de requerir estructuras de memoria muy eficaces [Díaz, et al. 1996].

2.6.2 Memoria de Corto Plazo y Largo Plazo

Un punto clave en la búsqueda tabú es el distinguir entre la memoria de corto plazo y la de largo plazo. Cada uno de estos tipos de memoria va acompañado de estrategias propias. Sin embargo, el efecto de ambos tipos de memoria puede verse como la modificación o restricción de la vecindad de la solución actual. El efecto de la memoria

puede observarse partiendo de que la búsqueda tabú guarda una historia selectiva H de algunos de los estados encontrados en la búsqueda, y remplaza la vecindad actual N(xActual) por un entorno restringido N(H, xActual). Entonces la vecindad es determinada por la historia, determinando que soluciones se pueden alcanzar a partir de la solución actual [Glover, et al. 1997].

En las estrategias TS basadas en memoria de corto plazo N(H, xActual) es generalmente un subconjunto de N(xActual), donde el estatus tabú sirve para identificar los elementos de N(xActual) excluidos de N(H; xActual).

En las estrategias de período intermedio y largo, N(H; xActual) puede contener soluciones que no estén en N(xActual), generalmente soluciones élite seleccionadas (soluciones de alta calidad), encontradas durante el proceso de búsqueda. Estas soluciones élite se identifican típicamente como elementos de un grupo local en estrategias de intensificación de período intermedio, y como elementos de diferentes grupos en estrategias de diversificación de período largo o largo plazo [Melián, et al. 2006].

2.6.3 Reencadenamiento de Trayectorias

El reencadenamiento de trayectorias (Path Relinking - PR) fue propuesto originalmente en el contexto de la búsqueda tabú, como una estrategia de intensificación, con el objetivo principal de balancear la intensificación y la diversificación [Glover 1994]. La idea de reencadenamiento de trayectorias es buscar soluciones que compartan atributos con antiguas soluciones con la esperanza de obtener mejores soluciones [Glover, et al. 2000].

El reencadenamiento de trayectorias se basa en la generación de nuevas soluciones mediante la exploración de trayectorias que conectan soluciones de calidad elevada, iniciando una búsqueda a partir de una de estas soluciones, solución inicial x', y generando un camino a través del espacio de búsqueda que dirige a búsqueda hacia otras

soluciones denominadas soluciones guía x'' [Glover, et al. 2000]. Para generar este camino basta con seleccionar movimientos que introduzcan progresivamente atributos de la solución guía (reduciendo la distancia entre los atributos de la solución inicial y la guía).

Existen algunas variantes del método de reencadenamiento de trayectorias, una de ellas consiste en iniciar el PR desde ambos extremos simultáneamente. Produciendo dos secuencias $x' = x'(1),..., x'(r)$ y $x'' = x''(1),..., x''(s)$. Las elecciones se diseñan provocando que $x'(r) = x''(s)$ para los valores finales de *r* y *s.* En cada paso el siguiente movimiento se selecciona intentando minimizar el número de elementos restantes para alcanzar $x'(r)$ desde $x''(s)$ y viceversa. [Melián, et al. 2006]

Otra variante de reencadenamiento es aquella que utiliza un efecto tipo túnel donde frecuentemente se pueden permitir periódicamente movimientos para el reencadenamiento que normalmente no serían tomados en cuenta por crear una situación de infactibilidad. Aunque se pasa por cierto momento de infactibilidad esta variante garantiza que al final se llegue a una solución factible, pues esta termina en la solución guía x''.

Otro punto importante en el reencadenamiento de trayectorias es dependiendo de las soluciones que utilicemos como inicial y guía, se puede dar un mayor énfasis en la intensificación o diversificación. Si ambas soluciones son tomadas de un conjunto de soluciones elite obtenidas a lo largo de la ejecución del programa, el reencadenamiento tendrá un enfoque de intensificación, mientras que si se eligen de conjuntos o vecindades diferentes se puede dar un mayor empuje a la diversificación. [Melián, et al. 2006]

2.7 Complejidad de los problemas

2.7.1 Problemas de decisión y análisis de algoritmos

Un problema de decisión $\pi = (D, Y)$ es una pareja formada por un conjunto de instancias *D*, las cuales se obtienen a partir de una instancia genérica que se especifica en términos de varios componentes: conjuntos, funciones, números, etc., y un subconjunto de instancias-sí $Y \subseteq D$.

Una instancia $i \in Y$, si y sólo si, la respuesta a la cuestión del problema es sí para esa instancia. Un problema de decisión se asocia con un lenguaje formal usando algún mecanismo de codificación y un algoritmo con una máquina de Turing.

Un algoritmo se dice que resuelve un problema de decisión si y sólo si, el lenguaje aceptado por la máquina de Turing es el subconjunto de todas las cadenas asociadas con las instancias-sí del problema. En las siguientes definiciones, si no se señala otra cosa, la palabra problema se usa como sinónimo de problema de decisión.

2.7.2 La 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 denomina tratables.

2.7.3 Problemas Intratables

Son todos los problemas de decisión para los que no existe un algoritmo determinista de tiempo polinomial que los resuelva, es decir, son todos los problemas que no están en *P*.

2.7.4 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.

2.7.5 Relación entre P y NP

Como toda máquina determinista es un caso particular de una máquina no determinista, se tiene entonces que $P \subseteq NP$.

2.7.6 Transformación polinomial

Se dice que un problema de decisión $\pi_1 = (D_1, Y_1)$ se puede transformar polinomialmente en el problema de decisión $\pi_2 = (D_2, Y_2)$ si y sólo si, existe una función $f : D_1 \rightarrow D_2$ que satisface las siguientes dos condiciones:

1. f es computable con un algoritmo determinista de tiempo polinomial.
2. Para toda instancia $i \in D_1$, $i \in Y_1$ si y sólo si, $f(i) \in Y_2$.

En tal caso se dice que $\pi_1 \alpha_p \pi_2$.

2.7.7 Problemas NP-completos y NP-duros

Un problema π es *NP*-Completo si y sólo si, $\pi \in NP$ y $\forall \phi \in NP, \phi \leq \pi$. Al conjunto de todos los problemas *NP*-Completos se denomina *NPC*. Un problema de optimización se dice que es *NP*-duro, si y solo si, su versión de decisión es *NP*-Completo.

Para probar que π pertenece a *NPC*, es suficiente y necesario probar que $\pi \in NP$ y que existe $\pi^* \in NPC$ tal que $\pi^* \alpha_p \pi$. Una propiedad importante de los problemas *NP*-Completos es que si $\pi \in NPC$, entonces $\pi \in NP$, si y sólo si, $P=NP$.

2.8 Complejidad del LOPCC

En [Bertacco, et al. 2005] se realiza la prueba y se demuestra que el LOPCC es *NP*-duro mediante una reducción del problema de Circuito Hamiltoniano.

El problema del circuito hamiltoniano (HP, Hamiltonian Path) se define dado un dígrafo $G_{HP} = (V_{HP}, A_{HP})$, decidir si *G* contiene un circuito hamiltoniano.

Partiendo de una instancia $G_{HP} = (V_{HP}, A_{HP})$ del problema HP, la instancia equivalente para el LOPCC es la siguiente:

$$V := V_{HP} = \{1, ..., n\}$$
$$p_i := 1 \forall \in V$$
$$c_{ij} = \begin{cases} M & si(i, j) \in A_{HP} \\ 2M & otherwise \end{cases} \forall i, j \in V, i \neq j$$

donde *M* es un número positivo suficientemente grande. Como se puede observar la construcción se puede realizar en tiempo polinomial.

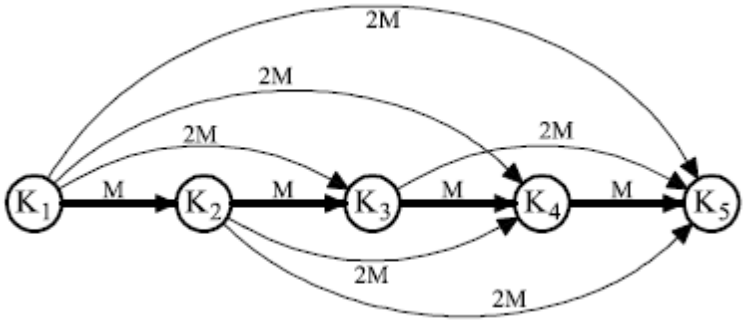

Fig. 3 Ejemplo de un buen circuito en el peor caso de la transformación.

Un ejemplo de un buen circuito de *G* solo involucraría arcos directos con costo menor o igual a *M*, mientras que los arcos transitivos tendrían costos no mayores a *2M*. Observando esto, no es difícil mostrar que, para una constante *M* lo suficientemente grande, el total de los costos acumulados asociados con el circuito es igual a $\pi(P) = M^{n-1} + O(M^{n-2})$. En cambio, si *P*, no corresponde a un circuito hamiltoniano en

G, tendría forzosamente que involucrar algún arco con costo *2M*, por lo tanto su costo acumulado $\pi(P)$ no podría ser menor que $2M^{n-1}$.

Para un valor suficientemente grande de *M*, esta transformación asegura que $\pi(P) < \pi(P')$ para un buen circuito hamiltoniano *P* y para cualquier no tan buen circuito hamiltoniano, lo que implica G_{HP} contiene un circuito hamiltoniano si y solo si alguna solución optima para el LOPCC corresponde a un buen circuito hamiltoniano.

CAPÍTULO 3

3. Estado del Arte

Como ya se ha mencionado, el LOPCC es una variante del LOP, y dado que el LOP ha sido ampliamente estudiado a través de los años, es conveniente abordar algunos de los trabajos más importantes referentes a este problema.

[Becker 1967] propuso una heurística basada en calcular cocientes para clasificar cada sector. En esta clasificación el sector con el más alto cociente se clasifica primero. La fila y columna correspondientes a este se eliminan de la matriz y se vuelve a aplicar el procedimiento a los sectores restantes.

En [Grotschel, et al. 1983] se describe un algoritmo exacto para el LOP mediante la técnica de ramificación y corte.

En 1985, Reinelt realiza un análisis de los métodos y aplicaciones del LOP más relevantes [Reinelt 1985]. Este autor también publico una biblioteca de instancias llamada LOLIB, compuesta de un conjunto de 49 instancias de tablas de entrada-salida de algunos sectores de la comunidad Europea [LOLIB 1997].

Chanas [Chanas, et al. 1996] propone una heurística basada en un mecanismo de inserción, el cual busca la mejor posición para insertar un nodo dentro de un orden parcialmente construido. Los nodos son revisados de acuerdo al orden establecido por la solución actual y una vez que se cae en un optimo local, el orden es invertido y el proceso es reiniciado.

Algunas metaheurísticas de optimización han sido aplicadas a LOP. Laguna [Laguna, et al. 1999] propuso dos procedimientos basados en la metodología de búsqueda tabú, uno limitado a estructuras de memoria de corto plazo, y otra incorporando componentes de memoria de largo plazo.

Vicente Campos [Campos, et al. 2001] aplica el algoritmo de búsqueda dispersa al LOP. Además, propone un término de corrección basado en la frecuencia con la cual un nodo aparece en una posición determinada del orden.

A diferencia del LOP, el LOPCC es relativamente poco estudiado, pues son pocos los investigadores que lo han abordado como problema de investigación. Es por eso que el estado del arte se centra en tres de los trabajos realizados acerca de este problema.

El primer trabajo acerca del LOPCC fue el de Bertacco [Bertacco, et al. 2005]. En este trabajo se demostró la NP-Completitud del LOPCC mediante una transformación desde el problema de circuito hamiltoniano. En el mismo artículo se realizó una formulación lineal entera del LOPCC con fronteras (BLOPCC), dicha formulación puede ajustarse para solucionar el LOPCC. El enfoque de solución propuesto por este trabajo fue el de solucionar este problema mediante programación lineal entera, así como, la utilización de un algoritmo enumerativo.

En [Benvenuto, et al. 2005] se realiza un posicionamiento del LOPCC en el contexto de las telecomunicaciones móviles, en este trabajo se refieren al problema como JOPCO (Joint Optimization of Power Control and Ordering). En cuanto al método de solución, este trabajo planteó una heurística similar al GRASP. Un punto a señalar de este trabajo es que las instancias reales que se utilizan son instancias pequeñas (<=16) debido a la naturaleza en su aplicación del mundo real.

Por último, la solución de referencia de este trabajo, en [Duarte 2006] se propone la única solución metaheurística encontrada. En esta solución se propone una solución mediante búsqueda tabú similar a la utilizada por [Laguna, et al. 1999] en la solución LOP. Duarte propuso una estructura de búsqueda tabú con tres etapas claramente definidas:

- La primera etapa, una heurística pseudo-aleatoria voraz para el proceso de construcción inicial. Esta heurística utiliza una lista de candidatos restringida de acuerdo al incremento de la solución objetivo, donde el nodo más atractivo es aquel con el mayor incremento.

- Una etapa de intensificación donde un determinado número de iteraciones se selecciona un nodo a mover, el cual no deberá ser tabú activo, después para seleccionar la posición donde se insertará el nodo se evalúa el impacto en la función objetivo que resulta de cambiar el nodo seleccionado a las *n-1* posiciones donde no reside actualmente. Hecho este se inserta el nodo seleccionado en la posición con la mejor diferencia en la función objetivo. Durante este proceso se acumula información acerca de la frequencia con la que un nodo ha sido movido. Esta etapa termina después de *maxIntensify* iteraciones sin mejorar la mejor solución de este proceso.

- La última etapa es un proceso de diversificación similar al de intensificación pero basado en un rankeo inverso a la frecuencia de los posibles movimientos, obtenida en el proceso de intensificación anterior. En este proceso no incorpora ninguna estrategia de memoria. Al finalizar este proceso los contadores de frequencia y las estructuras de memoria son reinicializados.

Un punto a destacar es que se propone una ecuación para minimizar el costo del cálculo de la función objetivo, la ecuación propuesta disminuye cerca de un 50% de costo computacional.

A continuación se presenta algunas tablas con los resultados obtenidos por las tres estrategias anteriores con instancias basadas en información real obtenida de redes UMTS europeas y en un grupo de instancias basadas en las instancias de la librería LOLIB. Las primeras son un grupo de 100 instancias, divididas en 4 subgrupos, mientras que las segundas son un grupo de 43 instancias. En la segunda columna se muestran los resultados obtenidos por el algoritmo enumerativo propuesto por Bertacco [Bertacco, et al. 2005], los obtenidos por el algoritmo voraz propuesto por Benveneutto [Benvenuto, et al. 2005]y en la última los resultados de la búsqueda tabú propuesta por Duarte [Duarte 2006].

Tabla 1. 25 Instancias UMTS (Tamaño 16)

	EM	**GR**	**TS_LOPCC**
Función Obj.	6.238	7.389	6.238
Desviación	0.0%	15.1%	0.0%
Núm. Óptimos Obt.	25	0	25
Tiempo CPU(segs.)	18.39	0.11	0.06

Tabla 2. 25 Instancias UMTS (Tamaño 16)

	EM	**GR**	**TS_LOPCC**
Función Obj.	14	23.052	14
Desviación	0.0%	36.4.1%	0.03%
Núm. Óptimos Obt.	25	0	24
Tiempo CPU(segs.)	7.72	0.11	0.03

Tabla 3. 25 Instancias UMTS (Tamaño 16)

	EM	**GR**	**TS_LOPCC**
Función Obj.	12.427	19.732	12.43
Desviación	0.0%	34.7	0.03

Núm. Óptimos Obt.	25	0	24
Tiempo CPU(segs.)	44.77	0.12	0.03

Tabla 4. 25 Instancias UMTS (Tamaño 16)

	EM	**GR**	**TS_LOPCC**
Función Obj.	0.0	33.014	19.05
Desviación	0.0%	40.5%	0.06%
Núm. Óptimos Obt.	25	0	24
Tiempo CPU(segs.)	22.28	0.12	0.03

Tabla 5. 43 Instancias LOLIB (Tamaño 50)

	GR	**TS_LOPCC**
Función Obj.	6.37E+12	8.26E+08
Desviación	82.71%	1.84%
Núm. Óptimos Obt.	1	42
Tiempo CPU(segs.)	1.91	1.43

Se puede observar como en los primeros cuatro grupos la primera estrategia obtuvo muy buenos resultados, obteniendo la solución óptima en cada caso, mientras que la segunda no obtuvo ningún óptimo. En cuanto a la tercer estrategia, esta obtuvo un 97% en cuanto a óptimos en un tiempo considerablemente menor con respecto a la primera.

En la quinta tabla podemos observar la ausencia de la primera estrategia, esto debido a que para instancias con tamaño mayor a 25 esta estrategia tomó un tiempo excesivo, más de 4 días para una instancia de tamaño 35. En esta misma tabla podemos observar que la tercer estrategia obtiene la mayoría de los óptimos en un tiempo mucho menor que la segunda, representando la mejor solución en instancias de tamaño mediano y grande.

CAPÍTULO 4

4. Metodología de solución

En este capítulo se describe la metodología utilizada para la solución del problema de la investigación, mediante el uso de la metaheurística búsqueda tabú como técnica base de solución.

4.1 Elección del Algoritmo Metaheurístico Base

Son diversas las razones que llevaron a tomar a la optimización por medio de búsqueda tabú como algoritmo base de esta investigación.

De acuerdo a [Birattari 2001] se propone una clasificación de algoritmos metaheurísticos basado en ciertas características. La tabla 6 muestra dichas características para los algoritmos de la sección 2.4.

Tabla 6 Características de algoritmos metaheurísticos

Característica	Recocido Simulado	Búsqueda Tabú	Algoritmos Genéticos	ACO	Grasp
Trayectoria	Sí	Sí	No	No	Sí
Enfoque Poblacional	No	No	Sí	Sí	No
Incorporación de Memoria	No	Sí	Parcialmente	Sí	Sí
Multiples Vecindades	No	No	Parcialmente	No	No
Función Objetivo Dinamica	No	Parcialmente	No	No	Sí
Inspirados en la Naturaleza	Sí	No	Sí	Sí	No

Tomando en cuenta esta información y además la estructura del LOPCC, se pudieron descartar algunos métodos. Los métodos poblacionales podrían ser factibles

debido a que cada solución del LOPCC, puede representarse con un costo mínimo, no así el cálculo de la función objetivo del problema, la cual tiene una complejidad de $O(n^2)$ lo que generaría un alto costo computacional al usar un método poblacional, al requerirse que se evaluara dicha función para múltiples individuos.

El segundo factor tomado en cuenta fue el estudio del estado del arte, el cual mostro que la búsqueda tabú implementada en [Duarte 2006] obtiene los mejores resultados superando las implementaciones de [Benvenuto, et al. 2005], encontrando la mayoría de los óptimos para las instancias pequeñas, y mejorando varias soluciones conocidas para las instancias de mayor tamaño de la literatura para LOPCC.

4.2 Implementación del Algoritmo Base

Para llevar a cabo esta tarea se realizaron las siguientes actividades:

- Se realizó un análisis de la solución de referencia e identificación de técnicas utilizadas en la solución tabú.
- De acuerdo al análisis se realizó un diseño para implementar la solución de referencia.
- Codificación a partir del diseño realizado.
- Se realizaron pruebas para garantizar que la solución implementada fuese lo más cerca posible a la solución de referencia, en cuanto a calidad de las soluciones y esfuerzo necesario.

En [Duarte 2006] se propone una solución mediante búsqueda tabú similar a la utilizada por [Laguna, et al. 1999] en la solución LOP. Duarte propuso una estructura de búsqueda tabú con tres etapas claramente definidas:

- La primera etapa, una heurística pseudo-aleatoria voraz para el proceso de construcción inicial. Esta heurística utiliza una lista de candidatos restringida de

acuerdo al incremento de la solución objetivo, donde el nodo más atractivo es aquel con el mayor incremento.

- Una etapa de intensificación donde un determinado número de iteraciones se selecciona un nodo a mover, el cual no deberá ser tabú activo, después para seleccionar la posición donde se insertará el nodo se evalúa el impacto en la función objetivo que resulta de cambiar el nodo seleccionado a las *n-1* posiciones donde no reside actualmente. Hecho este se inserta el nodo seleccionado en la posición con la mejor diferencia en la función objetivo. Durante este proceso se acumula información acerca de la frecuencia con la que un nodo ha sido movido. Esta etapa termina después de *maxIntensify* iteraciones sin mejorar la mejor solución de este proceso.
- La última etapa es un proceso de diversificación similar al de intensificación pero basado en un rankeo inverso a la frecuencia de los posibles movimientos, obtenida en el proceso de intensificación anterior. En este proceso no incorpora ninguna estrategia de memoria. Al finalizar este proceso los contadores de frecuencia y las estructuras de memoria son reinicializados.

Una vez realizado el análisis se procedió a la implementación del algoritmo base. Después de esto se realizó la experimentación para garantizar que la implementación fuese lo más cercano a lo reportado en [Duarte 2006]

En las siguientes tablas se muestran los resultados obtenidos de evaluar tanto la implementación de la solución de referencia, así como la implementación de la mejora incluyendo ambas variantes de reencadenamiento de trayectorias.

A continuación se muestran dos tablas con resultados de la experimentación con 50 instancias UMTS(tamaño 16) divididas en 3 grupos de 25 instancias cada uno. La experimentación se realizo 30 veces para cada instancia promediando tanto el valor obtenido de la función objetivo, así como el tiempo requerido para solucionar la instancia.

Tabla 7. 25 Instancias UMTS (Tamaño 16)

	[Duarte 2006]	Implementación Propia
Función objetivo	14.3	14.4
Tiempo	0.03	0.479

Tabla 8. 25 Instancias UMTS (Tamaño 16)

	[Duarte 2006]	Implementación Propia
Función objetivo	19.05	19.41
Tiempo	0.03	0.492

Después de varios intentos por igualar los resultados de la literatura, se logró conseguir la implementación original de [Duarte 2006]. Lo que fue de gran ayuda pues los esfuerzos se enfocaron totalmente en las siguientes etapas de la investigación.

4.3 Incorporación de Técnicas de Diversificación e Intensificación al Algoritmo Base

4.3.1 Reencadenamiento de Trayectorias

Se diseñaron 4 variantes de reencadenamiento de trayectorias, las cuales diferían en la manera como se seleccionaban los movimientos para generar la trayectoria hacia la solución guía.

En la primera variante la selección se realizaba de una manera secuencial, de acuerdo al orden natural de la permutación. El algoritmo del reencadenamiento de dicha variante se muestra a continuación:

1. Se selecciona aleatoriamente como solución inicial la solución obtenida del proceso de intensificación $\overrightarrow{x_{ini}}$ y como solución guía a la mejor partícula encontrada hasta el momento, $\overrightarrow{x_{guia}} = \overrightarrow{p_g}$.

2. Sea $d = \{i_1, i_2, \dots, i_r\}$ con $r \leq n$, donde *d* es el conjunto de atributos en los que las soluciones $\overrightarrow{x_{ini}}$ y $\overrightarrow{x_{guia}}$ son diferentes y *n* es el número de atributos en las soluciones.
3. Para cada diferencia *i* en *d*
    a. Se aplica el primer movimiento disponible para eliminar la diferencia en el atributo correspondiente.
    b. Se evalúa la solución obtenida mediante la aplicación de este movimiento.
    c. Se compara la solución obtenida en el paso 3.a con la solución global y si es mejor se actualiza.
4. Fin

Fig. 4 Algoritmo de reencadenamiento de trayectorias variante 1

En la segunda la decisión del movimiento a realizar se tomaba de manera aleatoria. El algoritmo del reencadenamiento para la segunda variante se muestra a continuación:

1. Se selecciona aleatoriamente como solución inicial la solución obtenida del proceso de intensificación $\overrightarrow{x_{ini}}$ y como solución guía a la mejor partícula encontrada hasta el momento, $\overrightarrow{x_{guia}} = \overrightarrow{p_g}$.
2. Sea $d = \{i_1, i_2, \dots, i_r\}$ con $r \leq n$, donde *d* es el conjunto de atributos en los que las soluciones $\overrightarrow{x_{ini}}$ y $\overrightarrow{x_{guia}}$ son diferentes y *n* es el número de atributos en las soluciones.
3. Mientras *d* no este vacío
4. Se elige aleatoriamente un movimiento
5. Se aplica el movimiento elegido en el paso anterior para eliminar la diferencia en el atributo correspondiente.
6. Se evalúa la solución obtenida mediante la aplicación de este movimiento.
7. Se compara la solución obtenida en el paso 3.a con la solución global y si es mejor se actualiza.

```
8. Fin
```

Fig. 5 Algoritmo de reecadenamiento de trayectorias variante 2

La tercera variante de reencadenamiento que se propuso fue la que se denomino reencadenamiento con reversa. En este reencadenamiento primero se evalúan las trayectorias en un sentido (inicial->guía) y después se invierte el sentido (guía->inicial), en busca de generar dos trayectorias en direcciones opuestas del espacio de búsqueda.

Finalmente, y aunque no es una estrategia de reencadenamiento en sí, también se probó el reencadenamiento tradicional pero cambiando el origen de las soluciones inicial y guía. Intentando manipular el objetivo del reencadenamiento ya fuese hacia la intensificación o hacia la diversificación.

4.3.2 Búsqueda Local Modificada

[Laguna, et al. 1999] propuso dos estrategias de búsqueda local, la estrategia *first* y la estrategia *best*. En la primera la búsqueda se realizaba seleccionando los sectores a mover recorriendo la permutación en el orden dado en busca del primer sector p_f cuyo movimiento resulte en un movimiento positivo. Mientras que la estrategia *best* seleccionaba el movimiento cuyo costo asociado sea el mejor. Claramente se puede observar que la primera variante es mucho menos costosa que la segunda, aunque la segunda garantiza obtener un óptimo local.

Tomando como base estas estrategias se intentaron algunas modificaciones con la finalidad de obtener variantes más equilibradas en cuanto al esfuerzo y la calidad se refiere.

La primera variante de búsqueda local incorporó una estrategia similar a la denominada *aspiración plus* conocida ampliamente en el ambiente tabú. Dicha estrategia establece un umbral y consiste en examinar movimientos hasta alcanzarlo, una vez

alcanzado, un número plus elementos es examinado y el mejor de todos es seleccionado [Glover, et al. 1997].

4.3.3 Reencadenamiento de trayectorias con búsqueda local

El reencadenamiento de trayectorias (Path Relinking - PR) fue propuesto originalmente en el contexto de la búsqueda tabú, como una estrategia de intensificación, con el objetivo principal de balancear la intensificación y la diversificación [Glover 1994]. La idea de reencadenamiento de trayectorias es buscar soluciones que compartan atributos con antiguas soluciones con la esperanza de obtener mejores soluciones [Glover, et al. 2000].

El reencadenamiento de trayectorias se basa en la generación de nuevas soluciones mediante la exploración de trayectorias que conectan soluciones de calidad elevada, iniciando una búsqueda a partir de una de estas soluciones, solución inicial x', y generando un camino a través del espacio de búsqueda que dirige a búsqueda hacia otras soluciones denominadas soluciones guía x'' [Glover, et al. 2000]. Para generar este camino basta con seleccionar movimientos que introduzcan progresivamente atributos de la solución guía (reduciendo la distancia entre los atributos de la solución inicial y la guía).

La variante propuesta consiste en introducir una etapa de búsqueda local dentro de la estructura del reencadenamiento de trayectorias. De manera al ir generando alguna trayectoria se realice una búsqueda local cada determinado número de pasos, de manera similar a lo realizado por [Chiarini 2004].

A continuación se muestra el algoritmo diseñado:

```
DoPathRelinking(x,g)
     1 TempSolution
     2 MejorCoste=coste(x)
     3 Mientras x <> g do
          3.1 TempSolution = SigMov(x; g)
```

```
            3.2   Si   TempSolution   es   mejor   que
        mejorCoste entonces
            3.3.1 ObtenerOptimoLocal(TempSolution)
        4 Fin
Fin.
```

Fig. 6 Algoritmo de reencadenamiento de trayectorias con búsqueda local

4.3.4 Vecindad de 2 movimientos

A diferencia de la vecindad utilizada hasta el momento en la cual cada vecino se genera aplicando un solo movimiento de inserción, se probó utilizar una vecindad donde estos se generarán aplicando dos movimientos de inserción. Dada una solution π, su vecindad de 2-intercambios $N(\pi)$ consiste de todas las soluciones obtenidas al permutar dos objetos. Este tipo de vecindad fue utilizada en el trabajo de [Chaovalitwongse, et al. 2009].

Los resultados de aplicar esta vecindad fueron desalentadores, a diferencia del problema LOP, en el LOPCC este tipo de vecindades es poco factible, pues aumenta la complejidad computacional en una unidad en el exponente, volviéndose el trabajo de evaluar una vecindad de n^3 a n^4.

4.3.5 Construcción inicial tipo GRASP

Después de realizado el análisis sobre el comportamiento de cada etapa del algoritmo de referencia [Duarte 2006], se pudo constatar que la etapa de construcción, realizada mediante una heurística pseudo-voraz, ubica al algoritmo desde un principio en vecindades sumamente atractivas, para después aplicar el algoritmo tabú señalado en anteriores ocasiones.

En esta actividad se diseñó un GRASP utilizando como heurística la función objetivo del problema. A continuación se exhibe el algoritmo diseñado:

```
Grasp()
     1 Solución=∅, LRC=∅
     2 Seleccionar aleatoriamente s de LC
     3 Solución ∪ s
     4 LC − s
     5 Mientras LC≠ ∅
          5.1 Crear LRC(LC, β)
          5.2
s=seleccionarElementoAleatoriamente(LRC)
          5.3 Solución ∪ s
     6 Fin
     7 Regresar Solución
Fin.
```

Fig. 7 Algoritmo GRASP

4.4 Evaluación del Desempeño del Algoritmo

Debido a la insuficiencia del análisis teórico en la evaluación del desempeño de algoritmos [Johnson 2002], la comunidad científica ha atraído su interés hacia análisis de tipo experimental.

Para el diseño de análisis de tipo experimental se proponen algunos principios para asegurar la obtención de conocimiento científico relevante [Barr 2001, Johnson 2002, McGeoch 1992, Moret 2003, Fraire 2005, Cruz 2004]. Entre estos principios se encuentran los siguientes:

- Los experimentos deben estar vinculados con la literatura. Es decir, deben ser capaces de compararse con los resultados de otras investigaciones.
- Se deben utilizar casos de prueba estándar.

- Se debe aportar evidencia estadística para soportar los resultados de la investigación.

CAPÍTULO 5

5. Experimentación y Resultados

5.1 Instancias Utilizadas

Para hablar de la experimentación realizada en este proyecto, es preciso detallar cuáles fueron las instancias de prueba utilizadas. Estas fueron las mismas instancias de prueba utilizadas por Duarte [Duarte 2006]. A continuación se describe dicho conjunto de instancias:

- Instancias UMTS, obtenidas del grupo de Ingeniería de la Universidad de Padova, relacionadas a optimización en redes UMTS. Este conjunto de instancias se divide en cuatro grupos de 25 instancias de tamaño *n=16*.

- El segundo grupo de instancias son las obtenidas de LOLIB [LOLIB 1997], que consisten de tablas de entrada y salida de sectores de la economía Europea. Este conjunto de instancias consta de 43 instancias de tamaño *n=50*.

- Por último, se utilizará también un grupo de 75 instancias generadas aleatoriamente, divididas en 3 conjuntos de 25 instancias cada uno.

5.2 Evaluación de Estrategias Propuestas

Como primera etapa de la evaluación de las estrategias de mejora, estas se contrastaron utilizando las instancias de tamaño 35 de tipo aleatorio.

A continuación se muestran tablas con los resultados de la experimentación. En la primera tabla (tabla 9) se exponen los resultados obtenidos con la implementación de duarte del algoritmo. En la tabla 10 se muestran los resultados de aplicar la modificación en la búsqueda local, en la tabla 11 se despliegan los resultados de utilizar como proceso de diversificación al reencadenamiento de trayectorias, mientras que en la tabla 12 se muestran los resultados de combinar las dos modificaciones anteriores.

Tabla 9 Resultados utilizando el algoritmo original

	EvalUpBest	EvalInt	EvalLoc	EvalDiv	TotEval	ImpInt	ImpDiv	ImpLoc	ImpTot	ObjVal
Prom:	84,030	130,677	95,861	41,593	172,270	111	0	20	130	0.342
%	48.78%	75.86%	55.65%	24.14%		84.73%	0.00%	15.27%		

Tabla 10 Resultados utilizando la búsqueda local modificada

	EvalUpBest	EvalInt	EvalLoc	EvalDiv	TotEval	ImpInt	ImpDiv	ImpLoc	ImpTot	ObjVal
Prom:	51,106	62,745	27,871	42,345	105,090	111	1	20	132	0.540
%	48.63%	59.71%	26.52%	40.29%		83.57%	1.00%	15.00%		

Tabla 11 Resultados aplicando reencadenamiento de trayectorias

	EvalUpBest	EvalInt	EvalLoc	EvalDiv	EvalPR	TotEval	ImpInt	ImpDiv	ImpLoc	ImpPR	ImpTot	ObjVal
Prom:	12,391	30,608	18,764	24,004	223	54,835	1	0	11	1	14	0.348
%	22.60%	55.82%	34.22%	43.77%	0.41%		10.74%	1.05%	83.64%	4.57%		

Tabla 12 Resultados aplicando reencadenamiento de trayectorias y la búsqueda local modificada

	EvalUpBest	EvalInt	EvalLoc	EvalDiv	EvalPR	TotEval	ImpInt	ImpDiv	ImpLoc	ImpPR	ImpTot	ObjVal
Prom:	15,768	31,138	17,086	28,313	265	59,716	2	1	14	1	18	0.3470
%	26.40%	52.14%	28.61%	47.41%	0.44%		12.53%	4.27%	77.06%	6.14%		

En la tabla 10 se pueden observar los resultados obtenidos por implementación de la búsqueda local modificada. Tal como en las suposiciones el costo computacional total del algoritmo se redujo drásticamente, cerca de un 30%, pero se vio afectada la calidad de soluciones obtenidas. Este resultado refuerza la idea de que la búsqueda local es un punto crítico en el desempeño del algoritmo, y que es susceptible de mejorarse.

En la tabla 11, donde se muestran los resultados obtenidos por el algoritmo en conjunción con el reencadenamiento de trayectorias se puede observar una aun mayor disminución en el costo computacional del algoritmo, reduciendo en aprox. 60% el trabajo realizado por este. Además la calidad de las soluciones obtenidas no se vio afectada considerablemente.

Por último, en la tabla 12 se despliegan los resultados referentes a aplicar la técnica de reencadenamiento y la búsqueda local modificada. En este caso se observa que se el ahorro de tiempo aun es considerable (arriba del 50%) a comparación del original, y que además se obtienen mejores resultados que de aplicar solo el reencadenamiento.

CAPÍTULO 6

6. Conclusiones

Al implementar una búsqueda local diferente a la propuesta en la solución de referencia, que dicho sea de paso no se encuentra documentada en la literatura, se pudo obtener un ahorro (30%) en cuanto a esfuerzo computacional, a costa de una perdida en la calidad de las soluciones obtenidas, con lo cual se obtuvo un aliciente para seguir experimentando en esta dirección, pues es una posible área de oportunidad en nuestro problema.

En cuanto a la inclusión del reencadenamiento de trayectorias como técnica de diversificación, se obtuvieron buenos resultados, pues en cuanto a esfuerzo computacional, este pudo disminuirse cerca de un 60%, con mucho menos pérdida de calidad que la modificación en la búsqueda local.

Al combinar las modificaciones anteriores se obtuvieron mejores resultados, pues se consiguió un ahorro parecido al de usar el reencadenamiento de trayectorias (mayor a 50%), con pequeña mejora en la calidad de las soluciones obtenidas por este.

En cambio al incorporar al reencadenamiento de trayectorias una búsqueda local, justo cuando encontrara alguna mejora, los resultados no fueron tan satisfactorios, pues aunque mejoro la calidad de las soluciones, el costo fue sumamente alto, alrededor de 70% del ahorro obtenido en el reencadenamiento se invertía en la búsqueda local.

La experimentación al reemplazar la vecindad de un movimiento por una de dos mostró que debido a la naturaleza del LOPCC no es factible este tipo de vecindades.

La eficiencia y resultados de la nuestra implementación la vuelven una buena opción para resolver problemas como el LOPCC.

Referencias

[Becker 1967] Becker, O.: 'Das Helmstädtersche Reihenfolgeproblem — die Effizienz verschiedener Näherungsverfahren', Computer uses in the Social Sciences, Berichteiner Working Conference, 1967

[Benvenuto, et al. 2005] Benvenuto, Carnevale, G., and Tomasin, S.: 'Optimum Power Control and Ordering in SIC Receivers for Uplink CDMA Systems', in Editor (Ed.)^(Eds.): 'Book Optimum Power Control and Ordering in SIC Receivers for Uplink CDMA Systems' (2005, edn.), pp.

[Bertacco, et al. 2005] Bertacco, L., Brunetta, L., and Fischetti, M.: 'The Linear Ordering Problem with Cumulative Costs', European Journal of Operational Research, 2005

[Campos, et al. 2001] Campos, V., Glover, F., Laguna, M., and Martí, R.: 'An experimental evaluation of a scatter search for the linear ordering problem', Journal of Global Optimization, 2001, 21

[Chanas, et al. 1996] Chanas, S., and Kobylanski, P.: 'A New Heuristic Algorithm Solving the Linear Ordering Problem', Computational Optimization and Applications, 1996, 6

[Chaovalitwongse, et al. 2009] Chaovalitwongse, W.A., and Pardalos, P.M.: 'Revised GRASP with Path-Relinking for the Linear Ordering Problem', 2009

[Chiarini 2004] Chiarini, B.H.: 'New Algorithm for the Triangulation of Input Output Tables and the Linear Ordering Problem', Florida University, 2004

[Díaz, et al. 1996] Díaz, A., Glover, F., Ghaziri, H.M., Gonzalez, J.L., Laguna, M., and Moscato, P.: 'Optimización Heurística y Redes Neuronales' (Paraninfo, 1996. 1996)

[Duarte 2006] Duarte, A.: 'Tabu Search for the Linear Ordering Problem with Cummulative Costs. Technical Report', 2006

[Glover 1994] Glover, F.: 'Tabú Search for Nonlinear and Parametric Optimization', Discrete Applied Mathematics, 1994

[Glover, et al. 1997] Glover, F., and Laguna, M.: 'Tabu Search' (Kluwer Academic Publishers, 1997. 1997)

[Glover, et al. 2000] Glover, F., Laguna, M., and Martí, R.: 'Fundamentals of Scatter Search and Path Relinking' (2000)

[Goldberg 1989] Goldberg, D.E.: 'Genetic Algorithm in Search, Optimization and Machine Learning' (Addison Wesley, 1989. 1989)

[Grotschel, et al. 1983] Grotschel, M., Junger, M., and Reinelt, G.: 'A Cutting Plane Algorithm for the Linear Ordering Problem', Operations Research, 1983, 32

[Kirkpatrick, et al. 1983] Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P.: 'Optimization by Simulated Annealing', Science, 1983, 220

[Laguna, et al. 1999] Laguna, M., Martí, R., and Campos, V.: 'Intensification and Diversification with Elite Tabu Search Solutions for the Linear Ordering Problem', Computers and Operations Research, 1999, 26

[LOLIB 1997] http://www.iwr.uni-heildelberg.de/groups/comopt/software/LOLIB/

[Melián, et al. 2006] Melián, B., and Glover, F.: 'Introducción a la Búsqueda Tabú', 2006

[Reinelt 1985] Reinelt, G.: 'The Linear Ordering Problem: Algorithms and Applications', Research and Exposition in Mathematics, 1985, 8

Anexo I. Código Fuente

A continuación se muestra el código fuente del algoritmo desarrollado en este proyecto:

Archivo Main.CPP

```
#include "tipos.h"
#include <math.h>
#define   GLOBAL    25

/* Primer argumento:  1 Reinelt, 2 SGB, 3 nuevo aleatorio
   Segundo:               nombre fichero
 */


int fin_inten, fin_diver, semilla, dim;
time_t t_tabu, f_tabu, tiempo, tiempoHastaMejor;
time_t t_inicio_greedy, t_fin_greedy, t_inicio_GR, t_fin_GR;

long double c_init, c_tabuLOPCC, c_GRLOPCC, Gamma;

long **lee_sgb_file(char *fichero) {
    FILE *puntfile;
    int i, j;
    long **matriz;

    if ((puntfile = fopen(fichero, "r")) == NULL)
        abortar("\nFichero de datos no encontrado");

    matriz = reserva_matriz(75);

    for (i = 1; i <= 75; i++) {
        for (j = 1; j <= 75; j++)
            fscanf(puntfile, "%ld", &matriz[i][j]);
        fscanf(puntfile, "\n");
    }
    fclose(puntfile);

    return matriz;
}

void escribe_fichero(char *nombre, time_t f_tabu, time_t t_tabu) {
    FILE *pf;

    pf = fopen("tsresult.txt", "a");
```

```
    fprintf(pf, nombre);
    fprintf(pf, "\t");
    fprintf(pf, "TS %Lf %4.2f", c_tabuLOPCC, ((float) f_tabu - t_tabu) /
CLOCKS_PER_SEC);
    fprintf(pf, "\n");
    fclose(pf);
}

float comprueba_tiempo(int paso, char *nombre, time_t t_tabu) {
    static int i = 1;
    float diftime;
    time_t tiempo;

    tiempo = clock();
    diftime = (tiempo - t_tabu) / CLOCKS_PER_SEC;

    if (diftime > paso * i) {
        i++;
        escribe_fichero(nombre, tiempo, t_tabu);
    }
    return (diftime);

}

int main(int argc, char **argv) {
    long double **matrix, **matrizLOPCC, **matrizScore;
    long double *vector, *vectorLOPCC, *alpha, *alpha_, *score, *alpha_l,
*mejorAlpha;
    long double dif_time, coste, mejor_coste, mejor_local;
    int aplicarPR = 1;

    long **matriz;
    long *tabu;
    long cambio_total, iter, change, tenure, iter_inten;

    int *score_int, *frec, *orden, *orden_invl, *orden_inv, *orden_l, *
mejorSol, *mejorSol_inv;
    int i, j, t, opcion, pos, pr, maxtime = 60;

    FILE *pf;

    numTotEval = 0;
    numEvalMejor = 0;
    numEvalInt = 0;
    numEvalDiv = 0;
    numEvalLoc = 0;
    numEvalPR = 0;
    numMejoras = 0;
    numMejorasInt = 0;
    numMejorasDiv = 0;
    numMejorasLoc = 0;
    numMejorasPR = 0;
```

```
    /************   Lectura / Generacion de Datos  *************/



    argv[1] = "4";
    argv[2] = "Instances/lolib/be75np";
    //argv[2] = "Instances/rnd/t1d35.1";
    argv[3] = "50";


    opcion = atoi(argv[1]);
    dim = atoi(argv[3]);

    Gamma = 0.625;
    vectorLOPCC = reserva_vector_long_double(dim);
    matrizLOPCC = lee_lopCC_file(argv[2], vectorLOPCC, dim);


    /************   Inicializaciones   ************************/
    //Soluciones y sus inversas
    orden = reserva_vector_int(dim);
    orden_inv = reserva_vector_int(dim);

    orden_l = reserva_vector_int(dim);
    orden_invl = reserva_vector_int(dim);
    alpha_l = reserva_vector_long_double(dim);

    mejorSol = reserva_vector_int(dim);
    mejorSol_inv = reserva_vector_int(dim);
    mejorAlpha = reserva_vector_long_double(dim);

    //Vectores para determinar el sector que se mueve (proporcional a su
score)
    score = reserva_vector_long_double(dim);
    score_int = reserva_vector_int(dim);

    frec = reserva_vector_int(dim);
    tabu = reserva_vector_long(dim);

    alpha = reserva_vector_long_double(dim);
    alpha_ = reserva_vector_long_double(dim);

    /**************Generar score vector**************/
    //Prepara la matriz para que se pueda aplicar el algoritmo del LOP al
LOP_CC
    //En concreto multiplica cada columna i de la matriz por el elemento
Pi del vector guardando en matriz intermedia
    matrizScore = preparaMatriz(matrizLOPCC, vectorLOPCC, dim);
    //Inicializa los escores y borra lo que le sobra, aqui realiza la
suma de la matriz anterior para generar los scores
    iniciaLOPCC(matrizScore, score, score_int, dim);
    //Eliminar matriz intermedia
    borrar_matriz_long_double(matrizScore, dim);
```

```
    free(score);
    /*********************************************/


    //Inicialización de parametros del algoritmo
    /*
        tenure = 1.7 * (long) sqrt(dim);
        fin_inten = 0.8 * dim;
        fin_diver = 1.7 * dim;
     */
    tenure = 1.7 * (long) sqrt(dim);
    fin_inten = 0.75 * dim;
    fin_diver = 0.75 * dim;


    long int a = 1471;
    //a = 0;
    a = time(0);
    srand(a);

    /************   Algoritmo  Tabu *******************************/
    t_tabu = clock();



    /*

        int o[] = {0, 11, 12, 15, 7, 8, 5, 1, 4, 14, 3, 9, 6, 2, 16, 13,
10};

        // for (i = 1; i <= dim; i++) {
        //     o[i]+=1;
        // }

        //int o[] = {0, 11, 12, 7, 8, 5, 1, 4, 14, 3, 9, 6, 2, 16, 13,
10, 15};
        printf("\nObj:    %Lf    \n",    calcula_costeLOPCC(matrizLOPCC,
vectorLOPCC, alpha, o, dim));

        for (i = 1; i <= dim; i++) {
            printf("%Lf,", alpha[i]);
        }

        return 0;
     */


    //Crea una solución inicial mediante un grasp, generando n soluciones
y seleccionando la mejor...
    c_init  =  construye_GRASP(orden,  orden_inv,  orden_l,  orden_invl,
alpha, alpha_l, alpha_, matrizLOPCC, vectorLOPCC, dim, 2, maxtime);
    //Sobreescribe la mejor solucion global
```

```
    bestlocal_a_actualLOPCC(orden, orden_inv, orden_l, orden_invl, alpha,
alpha_l, dim);

    printf("\nCosto inicial: %Lf", c_init);

    cambio_total = iter = 0;
    coste = mejor_coste = c_init;

    tiempo = clock();
    dif_time = (tiempo - t_tabu) / CLOCKS_PER_SEC;

    int iEval1 = 0, iEval2;

    /*
        //Se aplica una busqueda local despues de la etapa de
construcción
        iEval2 = numTotEval;
        mejor_local = optimo_localLOPCC_2opt(orden_l, orden_invl, dim,
mejor_local, matrizLOPCC, alpha_l, alpha_);
        bestlocal_a_actualLOPCC(orden, orden_inv, orden_l, orden_invl,
alpha, alpha_l, dim);
        if (mejor_local < coste) {
            numMejorasLoc++;
        }
        coste = mejor_local;
        numEvalLoc += numTotEval - iEval2;
     */


    //while (cambio_total < GLOBAL && dif_time < maxtime) {
    while (cambio_total < GLOBAL) {
        tiempo = clock();
        dif_time = (tiempo - t_tabu) / CLOCKS_PER_SEC;
        iter++;
        for (j = 1; j <= dim; j++) {
            frec[j] = 0;
            tabu[j] = -1000;
        }

        /*****************Intensificación*****************/
        change = iter_inten = 0;
        mejor_local = coste;

        //Inicializa el best local igual a la solución actual
        actual_a_bestlocalLOPCC(orden, orden_inv, orden_l, orden_invl,
alpha, alpha_l, dim);

        //El número de iteraciones que se le permite no mejorar la
solución es fin_inten = 0.8*dim
        while (change < fin_inten) {
            //while (change < fin_inten && tiempo < maxtime) {
            tiempo = clock();
            dif_time = (tiempo - t_tabu) / CLOCKS_PER_SEC;
```

```
            // Intensificacion basada en mejor insercion
            //Selecciona un indice en base al score vector
            j = select_indice(score_int); // indice original

            //Verifica que no sea tabú
            if (iter_inten - tabu[j] > tenure) {
                i = orden_inv[j]; //Elemento j de la permutación ocupa la
posición i

                tabu[j] = ++iter_inten;
                frec[j]++;

                // Busca la posición donde mejor encaja el elemento j que
ocupa la posición i
                coste += mejor_insertLOPCC(i, orden, matrizLOPCC, alpha,
alpha_, dim, &pos);

                //Inserta el elemento en la mejor posicion
                insertaLOPCC(i, pos, orden, orden_inv, alpha, alpha_);


                //Verifica si se mejoró el mejor local
                if (coste < mejor_local) {
                    actual_a_bestlocalLOPCC(orden,   orden_inv,   orden_l,
orden_invl, alpha, alpha_l, dim);
                    mejor_local = coste;
                    change = 0;
                    numMejorasInt++;
                } else
                    change++;
            }

            //Se le aplica una última búsqueda local a la solución
almacenada en orden_l , que es la mejor local que se ha encontrado hasta
el momento
            //Una vez optimizada la solución se vuelve a pasar orden
            if (change == fin_inten) {
                iEval2 = numTotEval;
                //mejor_local      =      optimo_localLOPCC_2opt(orden_l,
orden_invl, dim, mejor_local, matrizLOPCC, alpha_l, alpha_);
                mejor_local = optimo_localLOPCC(orden_l, orden_invl, dim,
mejor_local, matrizLOPCC, alpha_l, alpha_);
                bestlocal_a_actualLOPCC(orden,     orden_inv,     orden_l,
orden_invl, alpha, alpha_l, dim);
                if (mejor_local < coste) {
                    numMejorasLoc++;
                }
                coste = mejor_local;
                numEvalLoc += numTotEval - iEval2;
            }
        }
```

```
        numEvalInt += numTotEval - iEval1;
        iEval1 = numTotEval;


        //Actualiza el mejor global
        if (mejor_local < mejor_coste) {
            mejor_coste = mejor_local;
            //actual_a_best(orden, orden_inv, dim);
            actual_a_bestlocalLOPCC(orden,      orden_inv,      mejorSol,
mejorSol_inv, alpha, alpha_l, dim);
            cambio_total = 0;
            tiempoHastaMejor = clock();
            numEvalMejor = numTotEval;
        } else
            cambio_total++;

        /******** Diversificacion del Tabu usual ***************/
        calcula_frec(frec, dim);
        j = 1;
        //while (j < fin_diver && dif_time < maxtime) {
        while (j < fin_diver) {
            tiempo = clock();
            dif_time = (tiempo - t_tabu) / CLOCKS_PER_SEC;
            j++;
            t = select_indice(frec);
            i = orden_inv[t];
            coste  +=  mejor_insertLOPCC(i,  orden,  matrizLOPCC,  alpha,
alpha_, dim, &pos);
            insertaLOPCC(i, pos, orden, orden_inv, alpha, alpha_);
            actualiza_frec(frec, dim, t);
            if (coste < mejor_coste) {
                mejor_coste = coste;
                actual_a_bestlocalLOPCC(orden,    orden_inv,    mejorSol,
mejorSol_inv, alpha, mejorAlpha, dim);
                tiempoHastaMejor = clock();
                numEvalMejor = numTotEval;
                numMejorasDiv++;
            }
        }
        numEvalDiv += numTotEval - iEval1;
        iEval1 = numTotEval;

        /*        if (aplicarPR) {
                    aplicarPR = !aplicarPR;
         */
        //Aplicar  path  relinking  entre  la  solución  obtenida  por  la
búsqueda local y la obtenida por la etapa de diversificación
        coste  =  pathRelinking(orden,  orden_inv,  orden_l,  dim,  coste,
matrizLOPCC, vectorLOPCC, alpha);

        //Se le aplica una última búsqueda local a la solución encontrada
en PR
```

```
        mejor_local = optimo_localLOPCC(orden, orden_inv, dim, coste,
matrizLOPCC, alpha, alpha_);
        coste = mejor_local;

        if (coste < mejor_coste) {
            mejor_coste = coste;
            actual_a_bestlocalLOPCC(orden,      orden_inv,      mejorSol,
mejorSol_inv, alpha, mejorAlpha, dim);
            tiempoHastaMejor = clock();
            numEvalMejor = numTotEval;
            numMejorasPR++;
        }
        numEvalPR += numTotEval - iEval1;
        iEval1 = numTotEval;
        //Fin path relinking
        //    }
    }

    c_tabuLOPCC = mejor_coste;
    f_tabu = clock();

    //    printf("EB,EI,EL,ED,EPR,TE,MI,MD,ML,MPR,OBJ,TT");
    printf("\n%s,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%Lf,%4.2f",        argv[2],
numEvalMejor, numEvalInt, numEvalLoc, numEvalDiv, numEvalPR, numTotEval,
numMejorasInt, numMejorasDiv, numMejorasLoc, numMejorasPR, c_tabuLOPCC,
((float) f_tabu - t_tabu) / CLOCKS_PER_SEC);

    /*
        printf("\nTiempo   hasta   el   mejor:       %4.5f",    ((float)
tiempoHastaMejor - t_tabu) / CLOCKS_PER_SEC);
        printf("\nEvaluaciones hasta el mejor%d", numEvalMejor);
        printf("\nEvaluaciones intensificación: %d", numEvalInt);
        printf("\nEvaluaciones busq. local:     %d", numEvalLoc);
        printf("\nEvaluaciones diversificación: %d", numEvalDiv);
        printf("\nTotal Evaluaciones:           %d", numTotEval);
        printf("\nMejoras intensificación:           %d", numMejorasInt);
        printf("\nMejoras diversificación:           %d", numMejorasDiv);
        printf("\nMejoras busq. loc:                 %d", numMejorasLoc);

        printf("\nTS:         %Lf %4.2f", c_tabuLOPCC, ((float) f_tabu -
t_tabu) / CLOCKS_PER_SEC);
     */

    pf = fopen("tsresult.txt", "a");
    fprintf(pf,  "\n%s,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%Lf,%4.2f",  argv[2],
numEvalMejor, numEvalInt, numEvalLoc, numEvalDiv, numEvalPR, numTotEval,
numMejorasInt, numMejorasDiv, numMejorasLoc, numMejorasPR, c_tabuLOPCC,
((float) f_tabu - t_tabu) / CLOCKS_PER_SEC);
    fprintf(pf, " [");
    for (i = 0; i <= dim; i++) {
        fprintf(pf, " %d,", mejorSol[i]);
    }
    fprintf(pf, "]\n");
```

```
    /*
    fprintf(pf, argv[2]);
    fprintf(pf, "\t");
    fprintf(pf, "TS %Lf %4.2f", c_tabuLOPCC, ((float) f_tabu - t_tabu) /
CLOCKS_PER_SEC);
    fprintf(pf, "\t");
     */
    fclose(pf);

    //Liberaci□n de memoria
    free(orden);
    free(orden_inv);
    free(orden_l);
    free(orden_invl);
    free(alpha_l);
    free(mejorSol);
    free(mejorSol_inv);
    free(mejorAlpha);
    free(score_int);
    free(frec);
    free(tabu);
    free(alpha);
    free(alpha_);
    free(vectorLOPCC);
    borrar_matriz_long_double(matrizLOPCC, dim);

    return 0;
}
```

Archivo lopcc.CPP

```
#include "tipos.h"

void actual_a_bestlocalLOPCC(int *orden, int *orden_inv, int *orden_l,
int *orden_invl, long double * alpha, long double *alpha_l, int dim) {
    int s;

    for (s = 1; s <= dim; s++) {
        orden_l[s] = orden[s];
        orden_invl[s] = orden_inv[s];
        alpha_l[s] = alpha[s];
    }
}

//Sobreescribe la mejor solucion global

void bestlocal_a_actualLOPCC(int *orden, int *orden_inv, int *orden_l,
int *orden_invl, long double * alpha, long double *alpha_l, int dim) {
    int s;
```

```
    for (s = 1; s <= dim; s++) {
        orden[s] = orden_l[s];
        orden_inv[s] = orden_invl[s];
        alpha[s] = alpha_l[s];
    }
}

void actual_a_bestLOPCC(int *orden, int *orden_inv, int *bestSol, int
*bestSol_inv, long double * alpha, long double *bestAlpha, int dim) {
    int s;

    for (s = 1; s <= dim; s++) {
        bestSol[s] = orden[s];
        bestSol_inv[s] = orden_inv[s];
        bestAlpha[s] = alpha[s];
    }
}

long double *reserva_vector_long_double(int dim) {
    int i;
    long double *aux;

    aux = (long double*) calloc(dim + 2, sizeof (long double));
    if (!aux)
        abortar("Reserva vector long");

    for (i = 1; i <= dim; i++)
        aux[i] = 0;

    return aux;
}

/*bool *reserva_vector_bool(int dim){
    int i;
        bool *aux;

        aux = (bool*)calloc(dim + 2,sizeof(bool));
        if(!aux)
        abortar("Reserva vector bool");

    for (i = 1; i <= dim; i++)
        aux[i] = false;

        return aux;
} */

long double **reserva_matriz_long_double(int dim) {
    int i;
    long double **aux;

    aux = (long double**) calloc(dim + 1, sizeof (long double*));
    if (!aux)
        abortar("Reserva matriz");
```

```
    for (i = 1; i <= dim; i++)
        aux[i] = reserva_vector_long_double(dim);

    return aux;
}

void borrar_matriz_long_double(long double **matrix, int dim) {
    long i;

    for (i = 1; i <= dim; i++) // Paso 1: Borrar columnas
        free(matrix[i]);
    free(matrix);
}

void borrar_matriz_long(long **matriz, int dim) {
    long i;

    for (i = 1; i <= dim; i++) // Paso 1: Borrar columnas
        free(matriz[i]);
    free(matriz);
}

long double ** lee_lopCC_file(char *fichero, long double *vector, int
dim) {
    FILE *puntfile;
    int i, j;
    long double **matriz;

    if ((puntfile = fopen(fichero, "r")) == NULL)
        abortar("\nFichero de datos no encontrado");

    matriz = reserva_matriz_long_double(dim);

    for (i = 1; i <= dim; i++)
        for (j = 1; j <= dim; j++)
            fscanf(puntfile, "%Lf", &matriz[i][j]);

    for (i = 1; i <= dim; i++)
        fscanf(puntfile, "%Lf", &vector[i]);

    fclose(puntfile);
    return matriz;
}

long double **matriz_long_a_long_double(long ** matrizL, int dim) {
    int i, j;
    long double ** matrizLD;

    matrizLD = reserva_matriz_long_double(dim);

    //PARA CORREGIR LOS ERRORES DEL CASTING
    for (i = 1; i <= dim; i++)
```

```
        for (j = 1; j <= dim; j++)
            matrizLD[i][j] = (long double) matrizL[i][j];

    return (matrizLD);
}

//Genera la matriz y el vector necesarios para el LOPCC
//Aqui tengo que cambiarla ya que tenemos los valores del renglon de
pesos...

void setup2(long double ** matriz, long double **rif, long double
*vector, int dim, long double Gamma) {
    //Bertacco
    long double N0 = 0.50476587558415, NS = 16, M = 10, GNS = Gamma*NS,
GN0 = Gamma*N0;
    //LOLIB
    //long double Gamma=0.625*10E-2, N0 = 0.50476587558415, NS = 16, M =
10, GNS = Gamma*NS, GN0=Gamma*N0;
    //random
    //long double Gamma=0.625*10E-3, N0 = 0.50476587558415, NS = 16, M =
10, GNS = Gamma*NS, GN0=Gamma*N0;

    int i, j;
    long double mimRifValue = HUGE_VAL, maxRifValue = -HUGE_VAL,
mimVectValue = HUGE_VAL, maxVectValue = -HUGE_VAL;
    long double mimRif2Value = HUGE_VAL, maxRif2Value = -HUGE_VAL;

    for (i = 1; i <= dim; i++) {
        for (j = 1; j <= dim; j++) {
            if (matriz[i][i] != 0)
                rif[i][j] = GNS * matriz[j][i] / matriz[i][i]; //arc
weight
            else
                rif[i][j] = GNS * matriz[j][i]; //arc weight

            if (rif[i][j] < mimRifValue)
                mimRifValue = rif[i][j];
            if (rif[i][j] > maxRifValue)
                maxRifValue = rif[i][j];
        }
    }

    for (i = 1; i <= dim; i++) {
        if (sqrtl(matriz[i][i] >= 1E-9))
            vector[i] = GN0 / sqrtl(matriz[i][i]); //vertex weight
        else
            vector[i] = GN0; //vertex weight

        if (vector[i] < mimVectValue)
            mimVectValue = vector[i];
        if (vector[i] > maxVectValue)
            maxVectValue = vector[i];
    }
```

```
    maxVectValue = maxVectValue;
}

//Prepara la matriz para que se pueda aplicar el algoritmo del LOP al
LOP_CC
//En concreto multiplica cada columna i de la matriz por el elemento Pi
del vector

long double **preparaMatriz(long double **matriz, long double *vector,
int dim) {
    int i, j;
    long double ** matrizScore;

    matrizScore = reserva_matriz_long_double(dim);

    for (i = 1; i < dim; i++)
        for (j = 1; j < dim; j++) {
            matrizScore[i][j] = matriz[i][j];
            matrizScore[i][j] *= vector[j];
        }

    return (matrizScore);
}

/*Calcula el coste de la permutación orden*/
long double calcula_costeLOPCC(long double **matrizLOPCC, long double
*vectorLOPCC, long double *alpha, int *orden, long dim) {
    long double coste = 0;
    long i, j;

    for (i = 1; i <= dim; i++)
        alpha[orden[i]] = 0;


    for (i = dim; i >= 1; i--) {
        for (j = i + 1; j <= dim; j++) {
            alpha[orden[i]]    +=    matrizLOPCC[orden[i]][orden[j]]    *
alpha[orden[j]];
        }
        alpha[orden[i]] += vectorLOPCC[orden[i]];
        coste += alpha[orden[i]];
    }
    numTotEval++;
    return coste;
}

//Inicializa los escores y borra lo que le sobra, aqui realiza la suma de
la matriz anterior para generar los scores

void iniciaLOPCC(long double **matriz, long double *score, int
*score_int, int dim) {
    int i, j;
    long double maxScore = 0, minScore = 2000000000L, suma = 0;
```

```
    for (i = 1; i <= dim; i++)
        for (j = 1; j <= dim; j++) {
            if (i != j) {
                //Para cada elemento se suma su fila y su columna
conrrespondiente
                score[i] += matriz[i][j];
                score[j] += matriz[i][j];
            }
        }

    //Búsqueda del mayor y menor Score
    for (i = 1; i <= dim; i++) {
        if (score[i] > maxScore)
            maxScore = score[i];
        if (score[i] < minScore)
            minScore = score[i];
    }

    //Para poder hacer elecciones inversamente proporcional a la calidad,
hay que darle la vuelta (restar el mayor) al vector
    score[0] = 0;
    for (i = 1; i <= dim; i++) {
        score[i] = maxScore - score[i] + 1;
        if (score[i] < 0)
            score[i] = score[i];
        score[0] += score[i];
    }

    while (score[0] > 32500) {
        score[0] = 0;
        for (i = 1; i <= dim; i++) {
            score[i] = sqrtl(score[i]);
            if (score[i] < 1)
                score[i] = 1;
            score[0] += score[i];
        }
    }

    score_int[0] = 0;
    for (i = 1; i <= dim; i++) {
        score_int[i] = (int) score[i];
        score_int[0] += score_int[i];
    }
}

/*Selección aleatoria de un indice proporcionalmente a la calidad del
elemento;
es decir, es más probable seleccionar los elementos con mayor score*/
int select_indiceLOPCC(long double *score) {
    int cont = 1, f = 0, prob_sel;

    prob_sel = getrandom(0, score[0]);
```

```
    while (f == 0) {
        if (prob_sel <= score[cont])
            f = 1;
        else {
            prob_sel -= score[cont];
            cont++;
        }
    }
    return cont;
}

/*Selección aleatoria de un indice */
int select_indice_randomLOPCC(int dim) {
    return (getrandom(1, dim));
}

//Movimiento hacia la derecha con respecto a su posición original.
Evalúan el cambio en la función objetivo
//cuando el elemento j que ocupa las posición pos_j se coloca en la
posición pos_i

long double rightMovement(int pos_i, int pos_j, int *orden, long double
**matriz, long double *alpha, long double *alpha_) {
    int k, s;
    long double coste = 0;

    for (k = 1; k <= pos_i; k++)
        alpha_[orden[k]] = 0;

    //First term
    for (k = pos_j + 1; k <= pos_i; k++)
        alpha_[orden[pos_j]]    -=    matriz[orden[pos_j]][orden[k]]    *
alpha[orden[k]];
    coste += alpha_[orden[pos_j]];

    //Second term
    for (s = pos_i; s > pos_j; s--) {
        for (k = s + 1; k <= pos_i; k++) {
            alpha_[orden[s]]     +=     matriz[orden[s]][orden[k]]     *
alpha_[orden[k]];
        }
        alpha_[orden[s]]                                                +=
matriz[orden[s]][orden[pos_j]]*(alpha[orden[pos_j]]                      +
alpha_[orden[pos_j]]);
        coste += alpha_[orden[s]];
    }

    //Third term
    for (s = pos_j - 1; s >= 1; s--) {
        for (k = s + 1; k <= pos_i; k++) {
            alpha_[orden[s]]     +=     matriz[orden[s]][orden[k]]     *
alpha_[orden[k]];
        }
```

```
        coste += alpha_[orden[s]];
    }
    numTotEval++;
    return (coste);
}

//Movimiento hacia la izquerda con respecto a su posición original.
Evalúan el cambio en la función objetivo
//cuando el elemento j que ocupa las posición pos_j se coloca en la
posición pos_i

long double leftMovement(int pos_i, int pos_j, int *orden, long double
**matriz, long double *alpha, long double *alpha_) {
    int k, s;
    long double coste = 0;

    for (k = 1; k <= pos_j; k++)
        alpha_[orden[k]] = 0;

    //First term
    for (s = pos_j - 1; s >= pos_i; s--) {
        for (k = s + 1; k <= pos_j - 1; k++) {
            alpha_[orden[s]]     +=     matriz[orden[s]][orden[k]]     *
alpha_[orden[k]];
        }
        alpha_[orden[s]]     -=     matriz[orden[s]][orden[pos_j]]     *
alpha[orden[pos_j]];
        coste += alpha_[orden[s]];
    }

    //Second term
    for (k = pos_i; k < pos_j; k++)
        alpha_[orden[pos_j]]                                          +=
matriz[orden[pos_j]][orden[k]]*(alpha[orden[k]] + alpha_[orden[k]]);
    coste += alpha_[orden[pos_j]];

    //Third term
    for (s = pos_i - 1; s >= 1; s--) {
        for (k = s + 1; k <= pos_j; k++) {
            alpha_[orden[s]]     +=     matriz[orden[s]][orden[k]]     *
alpha_[orden[k]];
        }
        coste += alpha_[orden[s]];
    }
    numTotEval++;
    return (coste);
}

/* Dado el elemento de la posicion j de orden, se busca la mejor
   posicion en la que insertarlo. Se devuelve el update del coste
   y la mejor posicion en pos. Puede que la mejor posicion sea
   empeorar ya que no considera el dejarlo donde esta.
 */
```

```
long double mejor_insertLOPCC(int pos_j, int *orden, long double
**matriz, long double *alpha, long double *alpha_, int dim, int *pos) {
    long double best_c, c = 0;
    int k, best_pos;

    /* Inicializacion: se asegura que al terminar esta función tanto
best_c como best_pos tiene un valor correcto */
    if (pos_j > 2) {
        best_c = leftMovement(pos_j - 1, pos_j, orden, matriz, alpha,
alpha_);
        best_pos = pos_j - 1;
    } else {
        best_c = rightMovement(pos_j + 1, pos_j, orden, matriz, alpha,
alpha_);
        best_pos = pos_j + 1;
    }


    /* Insertarlo en una posicion anterior */
    for (k = pos_j - 1; k >= 1; k--) {
        c = leftMovement(k, pos_j, orden, matriz, alpha, alpha_);
        //MINIMIZACIÓN
        if (c < best_c) {
            best_c = c;
            best_pos = k;
        }
    }

    /* Insertarlo en una posterior */
    for (k = pos_j + 1; k <= dim; k++) {
        c = rightMovement(k, pos_j, orden, matriz, alpha, alpha_);
        // MINIMIZACIÓN
        if (c < best_c) {
            best_c = c;
            best_pos = k;
        }
    }

    //Obligatorio porque en alpha_ deben de quedar los mejores
incrementos.
    //Las otras opciones eran más costosas
    if (best_pos > pos_j)
        best_c = rightMovement(best_pos, pos_j, orden, matriz, alpha,
alpha_);
    else
        best_c = leftMovement(best_pos, pos_j, orden, matriz, alpha,
alpha_);

    *pos = best_pos;
    return best_c;
}
```

```
/*Inserta el elemento que se encuentra en la posición j en la posición i,
que es la que más le hace incrementar (o menos le hace decrementar) la
función objetivo.
Además, actualiza los correspondientes valores de alpha.
 */
void insertaLOPCC(int j, int pos, int *orden, int *orden_inv, long double
* alpha, long double *alpha_) {
    int k, a;

    if (pos > j) {
        a = orden[j]; //Elemento que se insertarán en la posición j
        for (k = j; k < pos; k++) { //Todos los elementos se desplazan
hacia la izda una posición para dejar libre pos
            orden[k] = orden[k + 1];
            orden_inv[orden[k + 1]] = k;
        }
        orden[pos] = a;
        orden_inv[a] = pos;
    } else {
        a = orden[j];
        for (k = j; k > pos; k--) { //Todos los elementos se desplazan
hacia la dcha una posición para dejar libre pos
            orden[k] = orden[k - 1];
            orden_inv[orden[k - 1]] = k;
        }
        orden[pos] = a;
        orden_inv[a] = pos;
    }


    a = max(pos, j);

    for (k = 1; k <= a; k++)
        alpha[orden[k]] += alpha_[orden[k]];

}

/*Inserta el elemento que se encuentra en la posición j en la posición i,
que es
la que más le hace incrementar (o menos le hace decrementar) la función
objetivo
Además, actualiza los correspondientes valores de alpha.
 */
void insertaLOPCC2(int j, int pos, int *orden, int *orden_inv) {
    int k, a;

    if (pos > j) {
        a = orden[j]; //Elemento que se insertarán en la posición j
        for (k = j; k < pos; k++) { //Todos los elementos se desplazan
hacia la izda una posición para dejar libre pos
            orden[k] = orden[k + 1];
            orden_inv[orden[k + 1]] = k;
        }
```

```
        orden[pos] = a;
        orden_inv[a] = pos;
    } else {
        a = orden[j];
        for (k = j; k > pos; k--) { //Todos los elementos se desplazan
hacia la dcha una posición para dejar libre pos
            orden[k] = orden[k - 1];
            orden_inv[orden[k - 1]] = k;
        }
        orden[pos] = a;
        orden_inv[a] = pos;
    }
}

long double improvingLOPCC(int *orden, int dim, int j, int *orden_inv,
long double ** matriz, long double *alpha, long double *alpha_) {
    long double valor;
    int pos;

    valor = mejor_insertLOPCC(j, orden, matriz, alpha, alpha_, dim,
&pos);
    if (valor < 0) {
        insertaLOPCC(j, pos, orden, orden_inv, alpha, alpha_);
        return valor;
    }

    return 0;
}

/* Calcula el optimo local desde el mejor_local (orden_l)*/
long double optimo_localLOPCC(int *orden_l, int *orden_invl, int dim,
long double coste, long double **matriz, long double *alpha_l, long
double *alpha_) {
    int i, k;
    long double old_coste;
    do {
        old_coste = coste;
        for (i = 1; i <= dim; i++)
            coste += improvingLOPCC(orden_l, dim, i, orden_invl, matriz,
alpha_l, alpha_);
        //MINIMIZACIÓN
        /*
                if (coste < old_coste) { //Para volverla una búsqueda
First
                    k++;
                    break;
                }
            } while (coste < old_coste && k < 5);
         */
    } while (coste < old_coste);
    return coste;
}
```

```
long double improvingLOPCC_2opt(int *orden, int dim, int i, int j, int
*orden_inv, long double ** matriz, long double *alpha, long double
*alpha_) {
    long double valor;
    int pos;

    valor = mejor_insertLOPCC(i, orden, matriz, alpha, alpha_, dim,
&pos);
    if (valor < 0) {
        insertaLOPCC(i, pos, orden, orden_inv, alpha, alpha_);
    }
    valor = mejor_insertLOPCC(j, orden, matriz, alpha, alpha_, dim,
&pos);
    if (valor < 0) {
        insertaLOPCC(j, pos, orden, orden_inv, alpha, alpha_);
        return valor;
    }
    return 0;
}

long double optimo_localLOPCC_2opt(int *orden_l, int *orden_invl, int
dim, long double coste, long double **matriz, long double *alpha_l, long
double *alpha_) {
    int i, j, k;
    long double old_coste;
    do {
        old_coste = coste;
        for (i = 1; i < dim; i++)
            for (j = i + 1; j <= dim; j++)
                coste += improvingLOPCC_2opt(orden_l, dim, i, j,
orden_invl, matriz, alpha_l, alpha_);
    } while (coste < old_coste);
    return coste;
}

long double pathRelinking(int *actual, int *actual_inv, int *mejor, int
dim, long double mejorCoste, long double **matrizLOPCC, long double
*vectorLOPCC, long double *alpha_l) {
    int bandera = 1, aux = 0, i, j;
    int *inicial, *inicial_inv, *guia;
    long double *alpha,*alpha_;
    inicial = reserva_vector_int(dim);
    inicial_inv = reserva_vector_int(dim);
    guia = reserva_vector_int(dim);
    alpha = reserva_vector_long_double(dim);
    alpha_ = reserva_vector_long_double(dim);

    long double coste = 0.0;

    for (i = 1; i <= dim; i++) {
        inicial[i] = actual[i];
        inicial_inv[i] = actual_inv[i];
        guia[i] = mejor[i];
```

```
    }

    //Recorre la permutacion inicial buscando diferencias
    for (i = 1; i <= dim; i++) {
        //Si la posición en las permutaciones inicial y guia son
diferentes
        if (inicial[i] != guia[i]) {
            bandera = 1;
            //Encuentra donde esta el elemento que debería ir en la
posicion i
            for (j = 1; j <= dim; j++) {
                if (inicial[j] == guia[i]) {
                    //Inserta el elemento j en la posicion i
                    insertaLOPCC2(j, i, inicial, inicial_inv);
                    break;
                }
            }
            coste = calcula_costeLOPCC(matrizLOPCC, vectorLOPCC, alpha,
inicial, dim);
            if (coste < mejorCoste) {
                actual_a_bestlocalLOPCC(inicial, inicial_inv, actual,
actual_inv, alpha, alpha_l, dim);
                //Aplica una busqueda local
                //coste = optimo_localLOPCC(actual, actual_inv, dim,
mejorCoste, matrizLOPCC, alpha_l, alpha_);
                mejorCoste = coste;
            }
        }
    }
    return mejorCoste;
}
```