El Problema del m-Anillo Estrella

Ignacio Vidal Lana

Trabajo de fin de grado en Matemáticas
Universidad de Zaragoza

Directora: Herminia I. Calvete
Septiembre 2016

Prólogo

El Problema del *m*-Anillo Estrella (en su versión en inglés, Capacitated *m*-Ring Star Problem, o CmRSP) consiste en diseñar un conjunto de ciclos o anillos, cada uno incluyendo un depósito central, un número limitado de clientes, y algunos posibles puntos llamados puntos de transición, que pueden ser usados para ahorrar costes de conexión.

Es un problema de optimización dentro de la teoría de grafos, y tiene como objetivo encontrar el diseño óptimo de una red de telecomunicaciones, usando la topología del anillo por su capacidad de evitar la pérdida de conexión de un nodo de comunicaciones aunque haya una avería en alguna de las conexiones.

En concreto, la red consistirá de *m* anillos (conjuntos de nodos conectados, formando un camino simple cerrado) incluyendo cada uno de ellos el nodo central o depósito, un conjunto de clientes o puntos de transición (puntos por los que puede pasar la conexión abaratando los costes de la red total) y las conexiones entre ellos. Al mismo tiempo, a estos nodos se podrán conectar otros clientes de forma directa, lo que explica el nombre de «*m*-anillo estrella».

Esta memoria se divide en tres capítulos. El primero de ellos toma como referencia el artículo *The Capacitated m-Ring-Star Problem* [1], de R. Baldacci et al., donde fue planteado por primera vez el problema. En el capítulo se da una introducción al problema y se presenta su planteamiento.

El segundo capítulo toma como referencia el artículo *A heuristic procedure for the Capacitated m-Ring-Star Problem* [5], de Z. Naji-Azimi et al., y en él se estudia un algoritmo heurístico propuesto por los autores, capaz de dar en tiempos mucho más razonables resultados muy cercanos, a veces superiores, a los mejores conocidos.

En el tercer capítulo, se presenta el algoritmo basado en programación lineal entera (PLE) presentado por Z. Naji-Azimi et al. [6]. Este capítulo se divide en tres secciones. En la primera se introduce el algoritmo, en la segunda se describe de forma resumida, y en la tercera y última sección se describe de forma detallada cada uno de los procedimientos que forman el algoritmo.

Por último, en el cuarto capítulo, se presenta la codificación, realizada por quien presenta esta memoria, del algoritmo heurístico del capítulo dos, en el lenguaje de programación *C++*. El programa es descrito en detalle, usando una sección propia para sus variables y otra para sus funciones o subrutinas, y termina con un cuadro de resultados de ejemplos conocidos del *CmRSP*.

Resumen

The Capacitated *m*-Ring Star Problem (CmRSP) is an optimization problem in graph theory, consisting of designing a set of rings or cycles that contain a central node or depot and several customers or transition points, and a set of direct connections from the remaining customers to the nodes of the rings.

Due to technical limitations, each ring can contain a bounded number of customers, number known as capacity of the ring. In addition, the number of rings in the design is exactly *m*.

The problem has the objective of minimizing the total cost of the network, at the same time that satisfies the problem's limitations, i.e., to achieve the optimal feasible solution.

This problem was published for the first time by R. Baldacci et al. [1] (2007). This work takes the Baldacci's article as its reference for the first chapter, where we present an introduction of the problem, along with two integer programming formulations of the problem, a two-index formulation, and a two-commodity flow formulation.

The CmRSP is an «NP-hard» problem, and it takes large amounts of time in order to achieve its optimal feasible solution. Because of this, the second chapter of this paper is based in the article of Z. Naji-Azimi et al.[5] (2010), that provides a heuristic algorithm able to solve the CmRSP in a reasonable amount of time. In addition, it obtains most of the optimal solutions.

This algorithm presentation is divided in four sections. First one gives a general description of the algorith, while the next three sections give the explanation of the tree procedure in wich the algorithm is based on.

The Initialization procedure gives an initial feasible solution. Then the Improvement procedure tries to improve the solution by aplying iteratively three subroutines: the Swap procedure, wich tries to get a better solution swapping some pairs of nodes, the Steiner-node-removal procedure, which aims to eliminate Steiner nodes from the solution if it results in a better solution, and a Extraction-assignment procedure, which extract and reassing each customer from the solution in order to obtain an improvement.

After the Improvement procedure, the Lin-Kernighan procedure is applied to each ring in order to improve its length. This procedure has been proposed by K. Helsgaun [3] (2000). Finally, the Shaking procedure perturbs the solution in order to scape from a local optimum. The final section of the second chapter consists of a summary of the procedures, in a computer-based style.

The third chapter of this work is based on another algorithm proposed by Toth et al.[6], an Integer-Lineal Programming based algorithm. The algorithm is extensively explained, in a similar way to the second chapter, providing a first section for the introduction to the algorithm, another one for its basic description, and a third one for the explanation of every procedure of the algorithm.

The fourth and final chapter develops of a computerization of the heuristic algorithm presented in chapter two with the only exception of the Lin-Kernighan procedure, that has been eliminated of the algorithm. It consists of a C++ program, which is explained function by function. In addition, we provide a table of results for some of the classical CmRSP instances.

Índice general

Capítulo 1

Planteamiento del problema

1.1. Introducción

El Problema del m-Anillo Estrella consiste en diseñar un conjunto de m ciclos o anillos, cada uno incluyendo un depósito central, un cierto número de clientes, y algunos posibles puntos llamados puntos de transición, que pueden ser usados para ahorrar costes de conexión.

El Problema del m-Anillo Estrella es un problema de optimización dentro de la teoría de grafos. En él se usa un grafo mixto donde el depósito, los clientes, y los puntos de transición son representados con nodos, mientras las conexiones entre ellos son aristas o arcos. Los puntos de transición serán representados por nodos Steiner.

Por su situación respecto a un anillo, un nodo se considerará un nodo visitado o vértice si un anillo pasa por él, y asignado a un vértice, si el nodo tiene una conexión directa con un vértice. Notemos que un nodo no puede ser asignado y visitado al mismo tiempo. Las aristas son conexiones entre dos vértices, mientras los arcos son las conexiones directas entre nodos asignados a un vértice y dicho vértice.

El problema tiene por objetivo diseñar un conjunto de m anillos (ciclos simples) que pasen por el depósito y otros nodos, y un conjunto de arcos, que asocian a los clientes no visitados por un anillo a algún anillo. Dos anillos sólo pueden compartir el depósito, nunca otro nodo.

El número total de clientes asociados a un anillo (visitados por él o asignados a sus nodos visitados) está limitado superiormente por Q, la capacidad del anillo. El objetivo del *CmRSP* es minimizar el coste de los anillos más el coste de las conexiones directas.

La justificación de esta estructura en forma de anillos, reside en el interés de ahorrar costes en el diseño de una red de telecomunicaciones, mientras se garantiza una continuidad del servicio a los clientes. Las conexiones del anillo tienen dos sentidos, uno en sentido horario y otro anti-horario. En una estructura anillada, si hay una avería en un punto, la conexión podría volver en el sentido antihorario, mientras que en una estructura lineal todos los clientes conectados a partir de la avería quedarían desconectados. Las conexiones directas a vértices de un anillo se justifican por el ahorro que supone evitar integrar el cliente como un vértice más, frente al riesgo de perder esa conexión directa.

El límite en la capacidad del anillo tiene su origen en el límite real de las conexiones, donde un haz de fibras sólo puede dar conexión a un número limitado de clientes.

Este capítulo está basado en la descripción del problema realizada por Baldacci et al. [1]

1.2. Notaciones y definiciones

Sea un grafo mixto $G = (V, E \cup A)$ en el que V representa el conjunto de nodos, E el conjunto de aristas y A el conjunto de arcos.

En el conjunto V distinguimos el nodo 0 que representa al depósito central y el nodo $n+1$ que es una copia del nodo 0 que se introduce para facilitar la formulación del problema, el conjunto de clientes U, y el conjunto W de nodos Steiner o de transición. Denotamos $V' = U \cup W$, conjunto que tiene n nodos

entre clientes y nodos Steiner como elementos. Por tanto, $V = \{0, n+1\} \cup V'$ y $|V| = n+2$. Recordamos que todo nodo visitado por un anillo es conocido como vértice.

El conjunto A representa las posibles conexiones entre clientes y vértices, es decir: $A = \{(i,j) : i \in U, j \in C_i\}$, donde C_i representa el conjunto de vértices de V' a los que se puede conectar el cliente i. Si $i \in U$ es un vértice, supondremos que se conecta a si mismo ($i \in C_i$).

El conjunto de aristas $E = \{\{i,j\} : i \in U, j \in V, i \neq j\}$ representa las conexiones posibles que pueden utilizarse en la construcción de los anillos. Dado un subconjunto $E' \subset E$, $V(E')$ denota el conjunto de vértices incidentes con al menos una arista de E'.

Denominamos camino simple entre dos vértices i_1 e i_k a una sucesión de vértices y aristas tal que de cada uno de los vértices existe una arista hacia el vértice sucesor, que comienza en i_1 y termina en i_k. Un anillo R es un par (E', A'), donde $E' \subset E$ es un camino simple del nodo 0 al nodo $n+1$, y $A' \subseteq A$ son las conexiones entre clientes de $U \setminus V(E')$ y vértices de $V(E')$. Notemos que $V(E')$ es el conjunto de vértices visitados por R, y $U \setminus V(E')$ son los clientes no visitados por R. Decimos que un cliente i está asociado a un anillo R si es visitado por el camino simple ($i \in V(E')$), o bien está conectado a un vértice del anillo (existe un vértice j tal que $(i,j) \in A'$).

Se han de construir m anillos, que tendrán una capacidad de Q clientes cada uno. Para que el problema sea factible, $mQ \geqslant |U|$. Por último, a cada arista $e = \{i,j\} \in E$ se le asocia un «coste de conexión» no negativo $c_e = c_{ij}$, mientras a cada arco $(i,j) \in A$ se le asocia un «coste de conexión directa» no negativo d_{ij} (con $d_{ii} = 0$ para todo $i \in U$).

En la Figura 1.1 se muestra un ejemplo de una solución factible a un hipotético problema con n=19, m=3, Q=6, $|U|$=15, $|W|$=4.

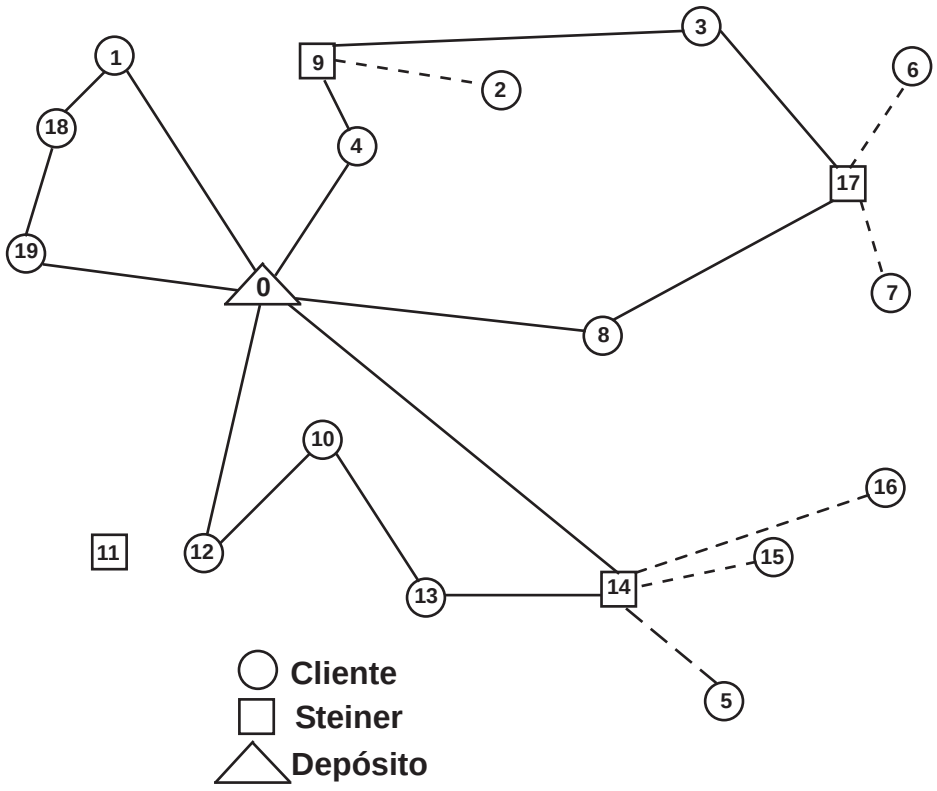

Figura 1.1: Una solución factible para un problema CmRSP

1.3. Formulación del problema

Se van a presentar dos formulaciones distintas. La primera en la que se utilizan las variables que aparecen de manera natural asociadas a los arcos y las aristas; la segunda en la que el problema se reformula como un problema de flujo en redes con dos mercancías.

1.3.1. Formulación con dos índices

Dado $S \subseteq V$, sea $\delta(S) = \{\{i,j\} \in E : i \in S, j \notin S\}$, el conjunto de aristas con sólo uno de los nodos perteneciente a S.

Si $S = \{i\}$, para simplificar la notación se escribirá $\delta(i)$ en lugar de $\delta(\{i\})$.

Además, dado $S \subseteq V$, $U(S)$ indicará el conjunto de clientes en S, es decir, $U(S) = S \cap U$.

Para cada arista $e \in E$, sea x_e una variable binaria que toma valor 1 si y sólo si e pertenece a un anillo de la solución. De ahora en adelante, si e conecta dos nodos i y j, entonces e y $\{i,j\}$ se usarán indistintamente para designar la misma arista.

Para cada arco $(i,j) \in A$, sea z_{ij} una variable binaria que vale 1 si y sólo si el cliente i está asignado al nodo j.

En adelante, para simplificar la notación, extenderemos algunas sumas a los pares $(i,j) \notin A$ asumiendo que en tal caso las sumas adicionales son iguales a 0.

Si un cliente i es visitado por un anillo, éste se asignará a si mismo, es decir, $z_{ii} = 1$.

Además, para cada $j \in W$, sea w_j una variable binaria que vale 1 si y sólo si el nodo Steiner (o de transición) j está en un anillo.

El problema puede ser formulado de la siguiente manera, que denotamos F1:

$$\min \sum_{e \in E} c_e x_e + \sum_{(i,j) \in A} d_{ij} z_{ij} \tag{1}$$

sujeto a

$$\sum_{e \in \delta(0)} x_e = m \tag{2}$$

$$\sum_{e \in \delta(n+1)} x_e = m \tag{3}$$

$$\sum_{e \in \delta(i)} x_e = 2 z_{ii} \qquad \forall i \in U \tag{4}$$

$$\sum_{e \in \delta(i)} x_e = 2 w_i \qquad \forall i \in W \tag{5}$$

$$\sum_{j \in V'} z_{ij} = 1 \quad \forall i \in U \tag{6}$$

$$\sum_{e \in \delta(S)} x_e \geq \frac{2}{Q} \sum_{i \in U} \sum_{j \in S} z_{ij} \quad \forall S \subseteq V' : S \neq \varnothing \tag{7}$$

$$x_e \in \{0,1\} \quad \forall e \in E \tag{8}$$

$$z_{ij} \in \{0,1\} \quad \forall (i,j) \in A \tag{9}$$

$$w_i \in \{0,1\} \quad \forall i \in W \tag{10}$$

La expresión (1) representa la función objetivo que trata de minimizar el coste total obtenido como la suma de los costes de conexión de los arcos de los anillos y los costes de conexión directa de los clientes que se conectan a los nodos de los anillos. Las restricciones (2) y (3) aseguran que m anillos salen del nodo 0 y entran por el nodo $n+1$. Las restricciones (4) y (5) imponen que el grado de cada nodo $i \in V'$ es 2 si pertenece a un anillo. Estas dos restricciones permitirían eliminar una de las restricciones (2) o (3) por ser redundantes, pero se mantienen para simplificar la presentación.

Las restricciones (6) establece que un cliente i o bien está en un anillo ($z_{ii} = 1$) o está asignado a un nodo de algún anillo ($z_{ij} = 1$ para algún $j \in C_i$). Las restricciones (7) son las restricciones fraccionales de capacidad del anillo. Estas restricciones, junto a la integridad de las variables x y z, imponen que, para un subconjunto S de nodos, al menos $\left\lceil \sum_{i \in U} \sum_{j \in S} z_{ij}/Q \right\rceil$ anillos son necesarios para visitar los clientes asignados a los nodos en S, donde $\lceil . \rceil$ indica el menor entero mayor que el número.

Esta formulación puede contener subrutas compuestas únicamente por nodos de transición de coste cero. Estas subrutas se pueden eliminar sin dejar de cumplir los requisitos del problema, pero como alternativa también puede añadirse la restricción:

$$\sum_{e \in \delta(S)} x_e \geq 2w_k \quad \forall S \subseteq W \quad y \quad \forall k \in S$$

1.3.2. Formulación como problema de flujo

En esta formulación se asocian dos variables de flujo y_{ij} e y_{ji}, a cada arista $\{i, j\} \in E$ que es representada con dos arcos opuestos. Ambos arcos tienen el mismo coste $c_{\{i,j\}}$. Se exige que en una solución factible del problema, el flujo total en cualquier arista de un anillo sea exactamente Q. Las variables y identifican cada anillo a través de dos caminos de flujo. En el primer camino, que va del nodo 0 al $n+1$, el flujo y_{ij} representa el número de usuarios que son servidos por el anillo tras visitar el nodo i. En el camino opuesto, que va del nodo $n+1$ al nodo 0, la variable y_{ji} representa el número potencial de clientes que pueden ser servidos en el camino de 0 a j. Así, el flujo total que sale del nodo 0 es $|U|$, mientras que el flujo que emana del nodo $n+1$ es mQ. Si un nodo en un anillo tiene un número v de unidades de flujo, absorbe v unidades de flujo que vienen del nodo 0 y v unidades de flujo que vienen del nodo $n+1$.

La formulación como problema de flujo será la siguiente, que denotamos por F2:

$$\min \frac{1}{Q} \sum_{\{i,j\} \in E} c_{ij}(y_{ij} + y_{ji}) + \sum_{(i,j) \in A} d_{ij} z_{ij}, \tag{12}$$

sujeto a

$$\sum_{j \in V'} y_{0j} = |U| \tag{13}$$

$$\sum_{j \in V'} y_{n+1,j} = mQ \tag{14}$$

$$\sum_{j \in V'} y_{j,n+1} = 0 \tag{15}$$

$$\sum_{j \in V'} y_{j0} = mQ - |U| \tag{16}$$

$$\sum_{i \in V} (y_{ji} + y_{ij}) = 2Qz_{jj} \quad \forall j \in U \tag{17}$$

$$\sum_{i \in V} (y_{ji} + y_{ij}) = 2Qw_j \quad \forall j \in W \tag{18}$$

$$\sum_{j \in V'} z_{ij} = 1 \quad \forall i \in U \tag{19}$$

$$\sum_{i \in V} (y_{ji} - y_{ij}) = 2 \sum_{i \in U} z_{ij} \quad \forall j \in V' \tag{20}$$

$$y_{ji} + y_{ij} \in \{0, Q\} \quad \forall \{i, j\} \in E \tag{21}$$

$$y_{ij} \geq \quad \forall i, j \in V \tag{22}$$

$$z_{ij} \in \{0, 1\} \quad \forall (i, j) \in A \tag{23}$$

$$w_j \in \{0, 1\} \quad \forall j \in W \tag{24}$$

La expresión (12) es la función objetivo, que depende de las variables de flujo y los costes de conexión y conexión directa. La restricción (13) impone que el flujo total que sale del nodo 0 es igual

al número total de clientes, mientras que la restricción (14) impone que del nodo $n+1$ el flujo saliente total es igual a la capacidad total de los m anillos. La restricción (15) indica que no entra flujo al nodo $n+1$. La restricción (16) impone que el flujo que entra en el nodo 0 sea igual a la capacidad residual (no usada) de los anillos. Las restricciones (17) y (18) imponen que la suma del flujo que entra y sale de todos los nodos de los anillos sea $2Q$, asegurando que Q unidades de flujo viajan por cada arista de los anillos. Las restricciones (19) aseguran que cada cliente es asignado a un nodo, y las restricciones (20) son las restricciones de conservación de flujo que tienen en cuenta que cada cliente absorbe una unidad de flujo del nodo 0 y otra del nodo $n+1$. Las restricciones (21) imponen el requisito binario de que una arista $e = \{i, j\} \in E$ o es visitada por un anillo (suma de flujos igual a Q) o no (suma de flujos igual a 0). Las restricciones (22) exigen la no negatividad de los flujos y las restricciones (23) y (24) exigen que las variables z_{ij} y w_j tomen valor 0 ó 1.

Capítulo 2

Un algoritmo heurístico para el CmRSP

Dado que el Problema del m-Anillo Estrella (*CmRSP*) es de tipo $NP-duro$, los algoritmos de ramificación y corte propuestos por Baldacci et al. [1] invierten tiempos extremadamente altos para solucionar casos con un número elevado de variables.

Para resolver este problema, Z. Naji-Azimi et al., proponen un algoritmo heurístico [5] capaz de hallar, en un tiempo razonable, la solución óptima en numerosos ejemplos conocidos, además de mejorar las soluciones conocidas en algunos ejemplos propuestos del *CmRSP*. Este algoritmo se va a presentar en este capítulo de la memoria.

2.1. Descripción del algoritmo

El algoritmo heurístico comienza con un Procedimiento de Inicialización que construye una solución factible. La parte central del algoritmo consiste en dos fases: un Procedimiento de Mejora y un Procedimiento de Perturbación que son ejecutados de forma iterativa.

El Procedimiento de Mejora tiene por objetivo mejorar la Solución Actual (aquella que el algoritmo mantiene en el inicio de la iteración) de forma local mediante determinados movimientos de la solución. Cuando este procedimiento ya no permite mejorar la solución, se ejecuta el Procedimiento de Perturbación, que trata de evitar que el algoritmo se estanque en un óptimo local. En ocasiones, tras la perturbación de la solución actual, el Procedimiento de Mejora no es capaz de mejorar la mejor solución conocida, y tampoco de recuperar la solución actual. En esta situación, se usa la idea del umbral de aceptación, que admite que se mantenga una solución peor si su coste no es mayor que un porcentaje dado P (un parámetro de entrada del algoritmo) respecto al Mejor Coste actual. Además, en cada iteración, el Procedimiento Lin-Kernighan [3] [4] es aplicado a cada anillo, buscando reducir la longitud (coste) de los mismos. El parámetro $Max\ iter$ es un parámetro de entrada que pone un límite al número de iteraciones del algoritmo.

2.2. Inicialización

Para construir una solución inicial se propone un algoritmo de agrupamiento (clustering) propuesto por M. Fischetti et al. [2]. Se seleccionan m anillos iniciales al escoger m clientes tan alejados unos de otros como sea posible y el nodo correspondiente al depósito (el nodo 0). Los clientes serán seleccionados de uno en uno, buscando en cada turno el más lejano posible a los ya elegidos. Después se unirán cada uno de ellos al depósito central. De esta manera se tendrán m anillos de un cliente cada uno. Después se asociarán, uno por uno, el resto de clientes según su posición factible óptima (aquella que, respetando las condiciones del problema, tiene el menor coste). En este proceso, no se utiliza ningún nodo Steiner.

2.3. Procedimiento de Mejora

En esta fase, con el propósito de mejorar la solución actual, se aplica el Procedimiento de Intercambio (PI) hasta que la solución no puede mejorarse más, luego se aplica el Procedimiento de Eliminación-de-nodos-Steiner (PENS) seguido del Procedimiento de Extracción-Asociación (PEA):

2.3.1. Procedimiento de Intercambio (PI)

Se selecciona un cliente aleatoriamente y se prueban todas las formas posibles de intercambiarlo con otro nodo cercano, empezando por el nodo más cercano hasta el T-ésimo (T es un parámetro de entrada del algoritmo). En cuanto se alcanza una mejora, se actualiza la solución y ya no se prueba el intercambio con ningún otro nodo. Continuamos entonces con otro cliente seleccionado aleatoriamente de la solución que aún no haya sido elegido. El procedimiento se detiene cuando se han considerado todos los clientes de la solución actual. Si uno de los nodos intercambiados tenía clientes asignados, estos se asignarán al nodo que ahora lo sustituye (si respeta el límite de capacidad del anillo). La única excepción se da en el caso de intercambiar un nodo j visitado con un cliente i asignado dentro del mismo anillo: en este caso, primero se visita i en la posición en la que estaba j (es decir, si j era el segundo vértice de un anillo, ahora lo será i), y después se extraen todos los clientes asignados a j para reasociarlos, de uno en uno y en orden aleatorio, en su posición óptima factible dentro del anillo (visitados o asignados a otro vértice). En el caso de que j fuera un nodo Steiner, dejará de tenerse en cuenta durante la reasociación, por lo que será eliminado de la solución.

2.3.2. Procedimiento de Eliminación-de-nodos-Steiner (PENS)

Se busca eliminar nodos Steiner para mejorar la Solución Actual. En cada paso del procedimiento, se extrae un nodo Steiner (elegido de forma aleatoria) junto a los clientes asignados al mismo; se reasocian los clientes considerando los T nodos más cercanos, en la posición óptima factible. Si tras esto hay una mejora de la solución, se actualiza la Solución Actual y, si es necesario, la Mejor Solución y el Mejor Coste. Si no, se devuelven tanto los clientes como el nodo Steiner a su posición original. El proceso se realiza con todos los nodos Steiner de la solución.

2.3.3. Procedimiento de Extracción-Asociación (PEA)

Se extrae cada cliente en orden aleatorio, y se intenta situar en una posición que mejore la Solución Actual. Un cliente i puede estar en tres situaciones: ser un cliente asignado a un nodo (y no ser visitado por un anillo), ser visitado por un anillo y no tener clientes asignados, o ser visitado por un anillo y tener asignado algún cliente.

En los dos primeros casos, el procedimiento consiste en extraer el cliente i de su posición actual, tras lo cual se busca una posición mejor entre los T nodos más cercanos. Sea j uno de estos nodos; j puede ser visitado por un anillo, en cuyo caso i podría ser asignado a j, o ser visitado por el anillo antes o después de j, dependiendo de cuál sea la posición que deja un menor coste total. Si j es un nodo Steiner no visitado, i es asignado a j, y j es visitado en su posición óptima entre los T nodos visitados más cercanos a j. Se selecciona el caso óptimo entre los mencionados y se actualiza la Mejor Posición.

Sea ahora el caso en que i es un cliente visitado y con clientes asignados. Se procede entonces de forma similar al Procedimiento de Eliminación-de-nodos-Steiner: se extrae i, junto a los clientes asignados al mismo, de la solución actual. Entonces se reasocian a su posición óptima factible y se actualiza la Mejor Posición.

En ambos casos se obtiene la Nueva Solución al actualizar la Solución Actual con la Mejor Posición. En caso de mejora del coste total, se actualiza la Solución Actual.

2.4. Procedimiento de Perturbación

El Procedimiento busca que el algoritmo pueda escapar de un punto óptimo local, por lo que se perturba la solución. En concreto, se extraen I nodos (con I como parámetro de entrada del algoritmo) de forma aleatoria de la solución, junto a los clientes que puedan tener asignados, y se construye una solución restringida sin estos elementos. Después, de uno en uno, se buscan todas las posiciones posibles dentro de la solución restringida donde se puedan asignar o visitar los clientes extraídos, buscando para cada uno su posición óptima factible.

2.5. Pseudo-códigos

A continuación se incluye el esquema de todos los procedimientos anteriormente presentados:

Algoritmo Heurístico propuesto por Z. Naji-Azimi et al.

Solución Actual := *Inicialización*();
Mejor Coste:= coste(Solución Actual);
Mejor Solución := Solución Actual;
$iter = 0$;

While $iter < Max\ iter$ **do**
  $iter$:=$iter + 1$;
    While Solución Actual pueda ser mejorada **do**
      Mejora(Solución Actual);
    End while
    If (coste(Solución Actual)< Mejor Coste) **Then**
      Para cada anillo, aplica $Lin - Kernighan$ para mejorar su coste
      Actualiza Solución Actual, Mejor Coste y Mejor Solución;
    Else if (coste (Solución Actual) > $P*$(Mejor Coste)) **Then**
      Solución Actual := Mejor Solución;
    End if;
    Perturbación(Solución Actual);
End While;

Procedimiento de Inicialización

Comentario: far(S) := arg max$\{\min\{c_{[v,w]} : w \in S\} : v \in U \backslash S\}$ es una función que escoge, dado un conjunto de clientes S, el cliente más lejano a los del conjunto. Usa la función *argmax*, la cual, actuando sobre otra función, da el argumento de la misma donde se alcanza el valor máximo. En este caso actúa sobre $min\{c_{[v,w]} : w \in S\}$, cuyo dominio son los clientes no pertenecientes a S: por tanto, lo que devuelve la función far es el cliente no perteneciente a S de mayor distancia al conjunto, entendiendo ésta como la distancia al elemento de S más cercano. Cabe destacar que la función *argmax*, en caso de empate, toma el punto de dominio de valor más bajo: en nuestro caso se traduce en el cliente con numeración menor.

Begin
$centro_1 := far(\{0\})$;
  For i = 2 hasta $m+1$ **do**
    $centro_i := far(\{centro_1, centro_2, ...centro_{i-1}\})$;
  End For;
  For $i \in U$ **do**
    If(cliente i no ha sido visitado o asignado) **Then**
      Asociar i a su posición factible óptima;
  End For;
End

Procedimiento de Mejora

While la solución actual pueda ser mejorada **do**
  PI(*Solución Actual*);
End While;
PENS(*Solución Actual*);
While la solución actual pueda ser mejorada **do**
  PEA(*Solución Actual*);
End While

Procedimiento de Intercambio (PI)

```
Orden aleatorio de clientes
For i=1 hasta |U| do
   For l=1 hasta T do
      j:=l-ésimo nodo más cercano al cliente i;
      If (i y j están en el mismo anillo) y (i está asignado y j es visitado, o viceversa)
         Then
         (suponiendo que i es el cliente asignado y j el nodo visitado)
         Construir la Nueva Solución como sigue:
            Visitar al cliente i en el anillo en la misma posición en que se visitaba j;
            Extraer el nodo j del anillo, junto a sus posibles clientes asignados
            Tomar cada uno de esos nodos en orden aleatorio y si es un cliente
            asignarlo o visitarlo en la posición factible óptima dentro del anillo;
      Else
         Nueva Solución := Solución Actual con i y j intercambiados;
      End If;
      If (Nueva Solución es factible) y (coste (Nueva Solución) < coste (Solución Actual)) Then
         Solución Actual := Nueva Solución;
         Actualizar si es necesario Mejor Solución y Mejor Coste;
         Break;
      End If;
   End For;
End For
```

Procedimiento de Extracción-Asociación (PEA)

Orden aleatorio de clientes
For i=1 hasta $|U|$ **do**
  If (i es un cliente asignado) o (i es un cliente visitado sin clientes asignados) **Then**
    Extraer i de su posición actual;
    For l = 1 hasta T **do**
      j := l-ésimo nodo más cercano al cliente i;
      If (j no es un cliente asignado) **Then**
        Caso 1. Considerar la asignación de i a j;
        Caso 2. Considerar la visita de i antes o después del nodo visitado j;
        Caso 3. **If** (j es un nodo Steiner no visitado) **Then**
          considerar la asignación de i a j y visitar j en su posición óptima factible entre sus T nodos visitados más cercanos;
          End If;
        Elegir la posición factible con mínimo coste y, si es necesario, actualizar Mejor Posición;
      End If;
    End For;
  Else
    Extraer el cliente i junto a sus clientes asignados, y reasociar cada uno de ellos a su posición factible
    óptima entre los T nodos visitados más cercanos;
    Actualizar si es necesario Mejor Posición basándose en la nueva posición de los clientes;
  End If;
  Nueva Solución := Solución Actual usando la Mejor Posición;
  If (coste(Nueva Solución) < coste(Solución Actual)) **Then**
    Solución Actual := Nueva Solución;
    Si es necesario actualizar Mejor Coste y Mejor Solución;
  End If;
End For

Procedimiento de Perturbación (PP)

For i = 1 hasta I **do**
  Extraer un nodo de forma aleatoria junto a sus clientes asignados de la Solución Actual;
End For;
While no hayan sido asignados o visitados todos los clientes extraídos **do**
  Considerar el siguiente cliente no asignado ni visitado y asociarlo a su posición
  óptima factible
End While

Capítulo 3

Un algoritmo heurístico combinado con programación lineal entera para el CmRSP

3.1. Introducción

Z. Naji-Azimi et al., proponen un algoritmo que combina ideas de un procedimiento heuristico con ideas de un modelo de programación lineal entera [6]. El método se basa en un procedimiento de Búsqueda en Entornos Variables (BEV) que incorpora una mejora basada en programación lineal entera cuando el procedimiento heurístico no puede mejorar.

El procedimiento propone un cambio dinámico de la estructura del entorno de búsqueda. El algoritmo propuesto por Z. Naji-Azimi et al. utiliza la idea de cambiar el tamaño de los entornos de búsqueda durante las iteraciones, hasta que se alcanza una condición de parada.

3.2. Descripción del algoritmo

El algoritmo propuesto, denotado NST, es un algoritmo de Búsqueda en Entornos Variables reforzado con un método de mejora basado en la programación lineal entera. Se comienza construyendo una solución inicial, que es mejorada usando un Procedimiento de Búsqueda Local. Dentro del algoritmo BEV se mejora la solución en un bucle que termina cuando se da una condición de parada. Dicho bucle contiene un Procedimiento de Perturbación y un Procedimiento de Búsqueda Local. El Procedimiento de Perturbación mantiene la estructura BEV: considera inicialmente un entorno específicamente diseñado y realiza cambios aleatorios a la solución actual para explorar entornos más alejados de dicha solución. El Procedimiento de Búsqueda Local considera un entorno más restringido, e intenta mejorar la calidad de la solución dada. Siguiendo el esquema BEV, en caso de mejora del coste de la solución se vuelve al tamaño del entorno inicial; de lo contrario, se incrementa el tamaño del entorno tratando de encontrar una mejor solución factible usando para ello el Procedimiento de Búsqueda Local. Se utiliza además un Umbral de Aceptación para actualizar la solución al final de cada iteración del algoritmo BEV. Se acepta una solución peor que la actual si su coste no supera un porcentaje P del coste de la mejor solución, o si el número J de iteraciones seguidas sin mejora no supera un valor máximo $JMAX$, siendo P y $JMAX$ dos parámetros de entrada del algoritmo. Durante todo el algoritmo, se asume que un nodo Steiner sin clientes asignados, será eliminado de la solución recortando el correspondiente anillo.

3.3. Procedimiento de Inicialización

Coincide con el procedimiento de inicialización presentado en el capítulo 2.

3.4. Procedimiento de Búsqueda Local

Este procedimiento consiste de cuatro partes principales: el Procedimiento de Mejora, el Procedimiento basado en Programación Lineal Entera, la resolución de un Problema de Asignación Modificado sobre los clientes asignados a los clientes de los anillos, y el Procedimiento Lin-Kernighan [4].

Se comienza llamando al Procedimiento de Mejora, y se realizan iteraciones mientras la solución pueda ser mejorada. Si el coste de la Solución ha sido reducido, se aplica el Procedimiento Basado en Programación Lineal Entera, durante un número $ILP-ITER$ de iteraciones (donde $ILP-ITER$ es un parámetro de entrada del algoritmo), tratando de mejorar la Solución Actual. Si se obtiene una mejora, el procedimiento busca una mejora aún mayor resolviendo un Problema de Asignación Modificada. En este problema, se extraen todos los clientes asignados a clientes visitados en la Solución Actual y se reasignan a su mejor anillo factible, es decir, al mejor nodo visitado por el anillo resolviendo un Problema de Asignación Modificada en el que se minimiza el coste global de las asignaciones mientras se satisface para cada anillo la restricción de capacidad. Como la capacidad de un anillo depende sólo de los clientes que incluye, la matriz de coeficientes del Problema de Asignación Modificada es totalmente unimodular, por lo que puede resolverse eficientemente con un algoritmo de flujo de mínimo coste. Entonces se aplica a cada anillo el Procedimiento Lin-Kernighan buscando un mejor orden de los nodos del anillo. Cuando tras estas operaciones haya una mejora de la solución, se actualiza la mejor solución conocida y se llama al Procedimiento de Mejora mientras la solución pueda ser mejorada. Todos estos pasos se ejecutan en un bucle hasta que la solución ya no pueda ser mejorada más.

3.4.1. Procedimiento de Mejora

Coincide con el procedimiento de mejora presentado en el capítulo 2.

3.4.2. Procedimiento basado en Programación Lineal Entera

Este es un procedimiento heurístico de mejora basado en técnicas de programación lineal entera (PLE). Dada una solución inicial factible, el método sigue una estructura de destrucción y reparación donde se destruye la solución dada (es decir, se eliminan algunos nodos de la solución), y se repara resolviendo un modelo de PLE llamado Modelo de Reasignación. El objetivo es encontrar una solución factible mejor que la dada.

Sea z la solución factible actual del CmRSP y F un subconjunto de clientes o nodos Steiner visitados en dicha solución. Se define $z(F)$ como la solución restringida obtenida al extraer de z los nodos de F. Se añade a F los clientes asignados a los nodos en F y los nodos Steiner no visitados por z. Se considera una partición de $F = F1 \cup F2$ donde $F1$ es el conjunto de clientes de F y $F2$ el conjunto de nodos Steiner de F. Sea $R = R(z,F)$ el conjunto de anillos en la solución restringida. Un punto de inserción es una posición en la solución restringida donde se puede insertar una secuencia o un nodo subconjunto o que puede usarse para asignar uno o más clientes. Definimos secuencia como un conjunto ordenado de clientes extraídos que puede ser insertado entre dos nodos visitados, y un nodo subconjunto como un cliente extraído que puede ser asignado a un nodo visitado, o como un cliente extraído, o un nodo Steiner con al menos un cliente extraído asignado. Se denota $I = I(z,F)$ un conjunto de puntos de inserción que correspondan a las aristas de la solución restringida donde puede insertarse una secuencia o un nodo subconjunto, y $J = J(z,F)$ denota un conjunto de clientes o nodos Steiner visitados en la solución restringida, a los que pueden asignarse uno o más clientes. Además, para cada anillo $r \in R$ definimos $I(r)$ y $J(r)$ como los conjuntos de sus puntos de inserción que pertenecen a I y a J respectivamente. Sea $S = S(F)$ el conjunto de todas las secuencias o nodos subconjunto factibles que pueden obtenerse combinando los nodos de F, junto a todos los clientes individuales en F que pueden asignarse a los nodos de J. Se define $q(s)$ como el número de clientes de la secuencia o nodo subconjunto $s \in S$. Para cada punto de inserción $i \in I$ sea $S_i \subseteq S$ el conjunto de todos los nodos subconjunto y secuencias que se pueden insertar en i. Para cada punto de inserción $j \in J$ sea $S'_j \subseteq S$ el conjunto de todos los clientes individuales que se pueden insertar en j. Además, para cada $i \in I$ y $w \in F$ se define $S_i(w) \subseteq S_i$ como el

conjunto de secuencias o nodos subconjunto que incluyen el nodo w y pueden ser insertados en i. Sea γ_{si} el coste de asignación o visita por insertar s en i, y d_{vj} el coste de asignación por asignar $v \in S'_j$ a j. También se define para cada anillo $r \in R$, $\tilde{q}(r)$ y $\tilde{c}(r)$ como el número de clientes y el coste del anillo en la solución restringida respectivamente.

El Modelo de Reasignación (MR) correspondiente a z y F puede definirse como:

$$\sum_{r\in R} \tilde{c}(r) + \min \sum_{i\in I} \sum_{s\in S_i} \gamma_{si} x_{si} + \sum_{j\in J} \sum_{v\in S'_j} d_{vj} y_{vj}, \tag{1}$$

sujeto a

$$\sum_{i\in I} \sum_{s\in S_i(v)} x_{si} + \sum_{j\in J} y_{vj} = 1, \qquad v \in F1, \tag{2}$$

$$\sum_{i\in I} \sum_{s\in S_i(w)} x_{si} \leq 1, \qquad w \in F2, \tag{3}$$

$$\sum_{s\in S_i} x_{si} \leq 1, \qquad I \in I, \tag{4}$$

$$\sum_{i\in I(r)} \sum_{s\in S_i(w)} q(s) x_{si} + \sum_{j\in J(r)} \sum_{v\in S'_j} y_{vj} \leq Q - \tilde{q}(r), \qquad r \in R, \tag{5}$$

$$x_{si}, y_{vj} \in \{0,1\}, \qquad i \in I, j \in J, s \in S_i, v \in F1, \tag{6}$$

donde las variables de decisión x_{si} e y_{vj} se definen como:

$$x_{si} = \begin{cases} 1 & \text{si la secuencia o nodo subconjunto } s \in S_i \text{ es insertado en } i \in I, \\ 0 & \text{en otro caso} \end{cases} \tag{7}$$

y

$$y_{vj} = \begin{cases} 1 & \text{si el cliente } v \in F1 \text{ es insertado en } j \in J, \\ 0 & \text{en otro caso} \end{cases} \tag{8}$$

La función objetivo a ser minimizada (1) da el coste total de los anillos consistente en los costes de visita y de asignación. Las restricciones (2) imponen que cada cliente extraído v sea visitado o asignado una sola vez. Las restricciones (3) implican que cada nodo Steiner sea usado a lo sumo una vez. Las restricciones (4) imponen que en cada punto de inserción $i \in I$ se puede insertar a lo sumo una secuencia o nodo subconjunto. Esta restricción no se aplica a los puntos de inserción $j \in J$. Por último las restricciones (5) imponen que cada anillo en la solución final cumpla la restricción de capacidad.

Los pasos principales del Procedimiento Basado en Programación Lineal Entera son los siguientes:

- **Fase de Selección:** Se construye F eligiendo de cada anillo de la actual solución z todos los nodos visitados en posición par, o los visitados en posición impar. Ambas opciones tienen la misma probabilidad.
- **Fase de Extracción:** Se extraen de z los nodos antes elegidos y se construye la solución restringida $z(F)$. Se añaden a F todos los clientes asignados a nodos de F y todos los nodos Steiner no visitados en z.
- **Fase de Inicialización:** Para cada $i \in I$ se inicializa S_i con la secuencia o nodo subconjunto básico, consistente en el nodo extraído de i en la Fase de Extracción, junto a la secuencia consistente en un único cliente perteneciente a F, que tenga el menor coste de visita. Para cada $j \in J$ se inicializa S'_j con el cliente perteneciente a F con el menor coste de asignación. Se inicializa la relajación lineal del Modelo de Reasignación (PL-MR) considerando los subconjuntos iniciales S_i ($i \in I$) y S'_j ($j \in J$) y se resuelve.
- **Fase de Generación de Columnas:** Para cada punto de inserción $i \in I$ (o $j \in J$) se resuelve su correspondiente problema de generación de columnas a través del Procedimiento Heurístico de Generación de Columnas (descrito más adelante) y se añaden a S_i (o a S'_j) todas las secuencias

o nodos subconjunto (o clientes pertenecientes a $F1$) tales que las variables asociadas x_{si} (o y_{vj}) tengan un coste reducido menor que un umbral dado RC_{max} (donde RC_{max} es un parámetro de entrada).

- **Fase de Reasignación:** Usando las secuencias y nodos subconjunto generados en las fases anteriores, se construye el Modelo de Reasignación correspondiente y se busca la solución óptima con un límite de tiempo $tiempo-PLE$ segundos (donde $tiempo-PLE$ es un parámetro de entrada).

Procedimiento Heurístico de Generación de Columnas

Para cada punto de inserción usamos un método heurístico para resolver su correspondiente Fase de Generación de Columnas. Concretamente, para cada $j \in J$ y $v \in F1$ consideramos la posible asignación de $\{v\}$ a j. Si el coste reducido es menor que el umbral RC_{max}, se añade v a S'_j. Para cada punto de inserción $i \in I$, por ejemplo $i = (a,b)$, se generan primero todas las secuencias s que consisten en uno o dos clientes pertenecientes a $F1$, y si el coste reducido correspondiente a la inserción de s en i es menor que RC_{max}, se añade s a S_i. A continuación, se generan todos los nodos subconjunto s obtenidos al asignar de uno a cinco clientes pertenecientes a $F1$ a un nodo perteneciente a F, y si el coste reducido es menor que RC_{max} se añade s a S_i.

Para cada punto de inserción $i \in I$ o $j \in J$, al generar las secuencias o nodos subconjunto se consideran sólo los RP nodos más cercanos al punto de inserción (donde RP es un parámetro de entrada). El coste reducido correspondiente es calculado de la siguiente forma. Sean $\vartheta_v^1, \vartheta_w^2, \vartheta_i^3, y \vartheta_r^4$ las variables duales asociadas a las restricciones (2)-(5) de la relajación lineal del modelo lineal MR donde $v \in F1, w \in F2$, $i \in I$ y $r \in R$, y sea $\tilde{\vartheta} = (\tilde{\vartheta}_v^1, \tilde{\vartheta}_w^2, \tilde{\vartheta}_i^3, \tilde{\vartheta}_r^4)$ la solución dual óptima correspondiente a PL-MR. Sea $s \in S_i$ la secuencia o nodo subconjunto a insertar en el punto de inserción $i \in I$, entonces el correspondiente coste reducido viene dado por:

$$RC_{si} := \gamma_{si} - \sum_{v \in S} \tilde{\vartheta}_v^1 - \sum_{w \in S} \tilde{\vartheta}_w^2 - \tilde{\vartheta}_i^3 - q(s)\tilde{\vartheta}_{r_i}^4$$

Para cada nodo subconjunto $s \in S'_j$ a ser insertado en $j \in J$, el correspondiente coste reducido viene dado por:

$$RC_{sj} := d_{vj} - \sum_{v \in S} \tilde{\vartheta}_v^1 - \tilde{\vartheta}_{r_j}^4$$

3.5. Procedimiento de Perturbación

Aplicado tras el Procedimiento de Búsqueda Local, el Procedimiento de Perturbación sigue la estructura de Búsqueda en Entornos Variables, y expande el entorno de búsqueda de forma dinámica. Se produce una solución que está en el entorno de tamaño K de la Solución Actual, $N_K(Solución\ Actual)$ que se define como el conjunto de soluciones que pueden ser obtenidas de la Solución Actual eliminando de la misma aleatoriamente K nodos junto a sus posibles clientes asignados, y asociando cada uno de ellos en orden aleatorio y de uno en uno a su posición óptima factible (la que genera menor coste de visita o asignación). En este procedimiento no se reinsertan los nodos Steiner que se hayan podido extraer.

Capítulo 4

Programación de una versión del algoritmo heurístico

4.1. Introducción

En este capítulo proponemos una computerización del algoritmo heurístico propuesto por Z. Naji-Azimi et al. [5]. Esta computerización es llevada a cabo con el lenguaje de programación *C++*. El programa consta de un único fichero con dieciocho funciones distintas (incluyendo la función «main»), y ocupa en su versión final un total de 1424 líneas de programa (incluyendo varias líneas en blanco para facilitar la lectura posterior). Los parámetros de entrada son: el número de nodos, el número de anillos, la capacidad del anillo, y la proporción de clientes respecto al total de nodos. Así como el número identificador y las coordenadas de cada uno de los nodos del problema, excepto el nodo depósito que es añadido a los demás por el propio programa en la función main. Si bien el algoritmo propuesto por los investigadores contenía otros parámetros de entrada (como el número de iteraciones a realizar, o el número de nodos a considerar en diversos procedimientos), dado que a intención era reproducir los experimentos originales, se han mantenido fijos dentro del programa, conservando los valores indicados en el artículo original.

La entrada de datos se produce por un fichero de texto, que contiene en la primera línea, y en este orden, el número de nodos del problema, el número máximo de anillos, la capacidad del anillo, y la proporción de clientes respecto al total de nodos (que en el programa y durante el resto de la sección se denominará «alpha»). Salvo alpha, que es un número real entre 0 y 1, los demás datos son números enteros. Las demás líneas consisten en tres números enteros, que representan el número identificador del nodo, su primera y su segunda coordenada, en este orden. Todos los datos se han obtenido de los documentos presentes en la librería TSPLIB, concretamente elegimos tres archivos de su sección TSP [7], los archivos «eil51.tsp», «eil76.tsp» y «eil101.tsp», para así coincidir con los utilizados por Z. Naji-Azimi et al. Pese a que la librería TSPLIB provee su propio formato de fichero de entrada, se modificaron para adaptarlos a la forma antes indicada por problemas con el formato de los archivos descargados.

4.2. Variables

Las principales variables del programa son los «Nodo», representación de los nodos del problema, que se construyen en el programa a través de un «struct» con las siguientes variables:

- **id:** Variable «int» que representa el número identificador del nodo.
- **c1:** Variable «int» que representa la primera coordenada del nodo.
- **c2:** Variable «int» que representa la segunda coordenada del nodo.

- **anil:** Variable «int» usada para identificar en qué anillo es visitado o está asignado el nodo, por defecto vale -1.
- **index:** Variable «int» usada para situar a un nodo visitado según su orden dentro de un anillo. El primer nodo de un anillo después del depósito se denominaría con el valor 1, el siguiente con el valor 2, y así sucesivamente. Si un nodo no es visitado por un anillo, esta variable toma valor -1.
- **asig:** Variable «bool» que indica si un nodo es asignado a otro (en caso afirmativo toma el valor $true$, de lo contrario, y por defecto, toma el valor $false$).
- **visit:** Variable «bool» que indica si un nodo es visitado por un anillo (en caso afirmativo toma el valor $true$, de lo contrario, y por defecto, toma el valor $false$).
- **asignados:** «Vector de punteros a Nodo» donde son almacenados los punteros a los nodos que tenga asignados (vacío por defecto).
- **padre:** Variable «puntero a Nodo» que apunta al nodo al que estaría asignado, en caso de que esto ocurra (valor $NULL$ por defecto).

Todos los Nodo salvo el que representa al depósito son construidos al principio de la función main, con los valores id, c1 y c2 obtenidos del fichero escogido. Tras esto, el nodo depósito se inicia con estos tres valores en 0. Las demás variables toman sus valores por defecto.

Otra variable global del programa es el array de «doubles» llamado «dista»; de dos dimensiones con tamaño 300, este array almacena el valor de la distancia entre dos Nodo aplicando la función «dist» que se describe más adelante (siendo $dista[i][j]$ el «double» con la distancia entre los nodos i y j).

Si consideramos N el número de nodos y m el número de anillos del problema, toda solución en el programa es representada por un array de N Nodo junto a un array de m vectores de punteros a Nodo que representan los anillos. En estos vectores el orden de sus componentes representa el orden en que serían visitados los nodos en el problema. Como el algoritmo exige la comparación de soluciones, en numerosas funciones se incluye la instrucción de generar una nueva solución a partir de otra conocida. Esto se realiza copiando primero todos los Nodo en un bucle «for» de tamaño N, seguido de dos bucles «for» anidados que, anillo por anillo, copian en la nueva solución los Nodo del anillo respetando su orden. Como los Nodo contienen variables puntero que aún apuntarían a los Nodo de la solución antigua, tras realizar la copia se actualizan para que apunten a los Nodo de la nueva solución.

4.3. Funciones

Se incluye a continuación un resumen de las funciones del programa.

4.3.1. actanil

Función «void» que actualiza un anillo actualizando las variables de todos sus Nodo visitados y asignados. En un único bucle «for» que recorre todos sus nodos visitados, se actualizan las variables «index», «anil», «visit» (que tomará el valor $true$) y «asig» (que tomará el valor $false$). Si el Nodo tiene nodos asignados, estos nodos serán actualizados al modificar sus variables «anil», «padre» (que ahora apuntará al Nodo al que esté asignado), «index» (que tomará el valor -1), «visit» (que tomará el valor $false$) y «asig» (que tomará el valor $true$). No devuelve nada.

4.3.2. capanillo

Función «int» que calcula el número de clientes que tiene un anillo entre nodos visitados y asignados. Tras inicializar una variable a en 0, un bucle «for» recorre cada nodo visitado, sumando a a la cantidad que tenga de nodos asignados, salvo que sea el nodo depósito en cuyo caso no ocurre nada. Si, además, el nodo visitado es un cliente, se suma 1 a a. Tras esto, se devuelve a.

4.3.3. compSol

Función «void» que compara la Solución Actual con la Mejor Solución (a introducir como parámetro), actualizando la Mejor Solución en caso de mejora.

Comienza con un «if» con condición coste(SoluciónActual)<coste(MejorSolución). Si esto ocurre, copia la Solución Actual en la Mejor Solución nodo por nodo (mediante un «for» que recorre cada nodo) y anillo por anillo (mediante otro «for» que recorre cada anillo). Tras esto, como los punteros en este momento apuntarían a nodos de la Solución Actual, son actualizados para que apunten a los nodos respectivos de la Mejor Solución (mediante un «for» para limpiar las referencias de los punteros «asignados» y «padre», y otro más que, en caso de que su semejante en Solución Actual tenga nodos asignados, introduce los punteros a los respectivos nodos en Mejor Solución).

4.3.4. coste

Función «double» que calcula el coste total de una solución. Tras inicializar una variable «double» 0,0, en un único bucle «for» que recorre cada anillo, primero se ejecuta un «do-while» que recorre todos los nodos visitados por un anillo, sumando a la variable la distancia entre el nodo visitado y su siguiente. Después, y dentro del «do-while» todavía, si este nodo tiene nodos asignados, se suman a la variable la distancia entre estos y el nodo visitado. Finalmente se devuelve el valor de la variable.

4.3.5. dist

Función «double» encargada de calcular la distancia euclídea entre dos nodos, tomando como parámetros dos Nodo. Devuelve dicha distancia. Designando las coordenadas de un nodo como Nodo $a := (a_1, a_2)$ fórmula usada es:

$$distancia(Nodoa, Nodob) = \sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2}$$

4.3.6. elesimo

Función «Nodo» que calcula, dado un parámetro «int» llamado *l* y un Nodo llamado *base*, el *l*-ésimo nodo más cercano a *base*. Un bucle «for» que recorre todos los nodos crea un array de «pair», variables formadas por dos elementos. En este caso, el primero elemento es el «double» correspondiente a la distancia entre el nodo y *base*, mientras el segundo elemento es el «id» del Nodo. Después se usa la función «sort» que ordena este array en función de sus primeros elementos, es decir la distancia a *base*, de menor a mayor. Finalmente se devuelve el Nodo cuyo «id» corresponda el segundo elemento de la posición *l* del array.

4.3.7. far

Función «void» encargada de calcular el nodo más alejado de un conjunto de nodos (los llamados «centro» del Procedimiento de Inicialización en el algoritmo original). Tras calcularlo, integra este Nodo en el conjunto dado.

Comienza con un bucle formado por dos «for» anidados, el primero recorre todos los clientes y el segundo todos los nodos «centro» que se hayan calculado hasta el momento de la ejecución de la función; en el bucle se calcula la distancia más corta de cada cliente al conjunto de nodos «centro», mediante una variable «double» que comienza tomando el valor de la distancia al depósito y que se actualiza cada vez que se halla una distancia menor a uno de los nodos de dicho conjunto. Tras este bucle, otro «for» que recorre todos los clientes saca el máximo de los valores antes hallados, de esta forma sacamos el máximo entre los mínimos de las distancias al conjunto «centro» de los clientes. El nodo que corresponda a esta distancia, será el nuevo nodo «centro». Como el conjunto se actualiza dentro de la función, no se devuelve ningún valor.

4.3.8. inifactopt

Función «void» que calcula, dada una solución completa (lista de Nodo más array de anillos), la posición óptima para cada cliente no visitado ni asignado. Función sólo utilizada al final del Procedimiento de Inicialización.

En un único bucle «for» que recorre cada cliente, si el nodo no está en ningún anillo, se llega a un segundo «for» anidado con el primero que recorre cada anillo. En este bucle, tras comprobar que el anillo admite otro cliente sin superar su límite de capacidad, se recorre cada nodo visitado por el anillo y se comparan dos casos: o bien se visita el nodo en disputa después del nodo visitado, o bien se asigna a éste. Mediante una comparación de costes se escoge la forma óptima y se aplica, usando la función «actanil» tras la aplicación para hacer efectivo el cambio. La función no devuelve nada.

4.3.9. mejora

Función «void» que aplica consecutivamente a Solución Actual el Procedimiento de Intercambio (PI), el Procedimiento de Extracción de Nodos Steiner (PENS) y el Procedimiento de Extracción-Asociación (PEA). Tras cada función compara la Solución Actual con la Mejor Solución, actualizando esta última si hay alguna mejora. Tanto PI como PEA son aplicados mientras se note una mejora en el coste.

Su estructura es como sigue. Primero se crea una nueva solución llamada MSol que toma el valor de la Solución Actual. A continuación, un «do-while» tiene como órdenes ejecutar la función «PI» con la Solución Actual y después comparar la Solución Actual resultante con la Mejor Solución. La condición «while» es la de detectar una mejora en el coste. Una vez que esto no ocurra, se ejecutan seguidamente «PENS» con la Solución Actual y se compara después con la Mejor Solución, tras lo cual viene otro «do-while» idéntico al primero, pero con orden de ejecutar «PEA» en lugar de «PI». Una vez que tras ejecutar «PEA» no se dé ninguna mejora en la solución, se termina la función sin devolver ningún valor.

4.3.10. mostrarAn

Función «void» que muestra en la consola de ejecución los anillos de la solución. Un bucle «for» recorre cada anillo con una función de salida de texto en pantalla que muestra el número del anillo y la cantidad de clientes que tiene (usando la función «capanillo» que describimos más tarde). Tras esto, y aún dentro del bucle, se recorre cada nodo visitado por el anillo, y si tuviera nodos asignados se mostrarían en pantalla indicando el nodo «padre» y todos los asignados que tiene. Es una función «void» y no devuelve ningún valor.

4.3.11. opfacan

Función «void» que calcula la posición óptima de un Nodo *base* dentro de un anillo dado. Se diferencia de «inifactopt» en que aquí se busca introducir el cliente en un anillo específico, mientras que la otra función miraba en todos los anillos al mismo tiempo. Además, mientras «inifactopt» sólo tiene uso una vez justo en la inicialización de la solución, «opfacan» es usado otras veces durante el programa.

Tras comprobar que el anillo tiene capacidad para un cliente más, se inicializan dos variables «double»: *cost* en 0 y ref en 50000, y se inicia un bucle «for» que recorre todos los nodos visitados por el anillo y compara en cada uno de ellos dos opciones, visitar el Nodo *base* después del nodo, o asignarlo al mismo. Se recoge en *cost* el caso más óptimo, y si *cost* es menor que ref, se iguala ref a *cost* y se guarda la posición del nodo y el caso elegido, de modo que se almacena siempre la posición óptima entre todas las comprobadas. Al final del proceso, ref tendrá el coste menor entre todas las opciones posibles, y se aplica dicho cambio. La función no devuelve nada.

4.3.12. PEA

Función «void» que implementa el Procedimiento de Eliminación-Asociación. Igual que «PENS», ordena a través de un array los clientes de forma aleatoria, que son recorridos por un bucle «for». Dentro de este bucle, y tras crear una Nueva Solución, se distinguen dos casos: que tenga clientes asignados o que no los tenga. En el primer caso, se repite exactamente lo mismo que ocurría con los Nodo de *extra* en la función «PENS». En caso de que no tenga asignados, si el Nodo es un nodo visitado o un cliente asignado, primero se elimina el Nodo de la Nueva Solución, y actualizamos sus variables. Después, en un bucle que recorre los T Nodo más cercanos, se distinguen otros dos casos. Si el Nodo l-ésimo es un Nodo visitado, se compararán tres posiciones posibles del Nodo eliminado respecto al Nodo l-ésimo: visitarlo antes, visitarlo después, o asignarlo. Se tomará el caso óptimo y si hay mejora en el coste se actualizará la solución. En el caso en que el Nodo eliminado sea un nodo Steiner no visitado, se plantearán dos casos. Tras asignar el Nodo eliminado al nodo Steiner, visitar este antes o después de los nodos que se encuentran entre los T más cercanos al nodo Steiner. Se toma el caso óptimo y se actualiza la solución en caso de haber mejora. La función no devuelve nada.

4.3.13. PENS

Función «void» que implementa el Procedimiento de Eliminación de Nodos Steiner. Primero se crea un vector de punteros a Nodo llamado *steiners*, donde se almacenan todos los nodos Steiner de la Solución, que son encontrados recorriendo anillo por anillo todos los nodos visitados. Después se reordenan de forma aleatoria con la función «random shuffle», tras lo cual se recorren por un «for» que en cada paso crea una Nueva Solución que iguala con la Solución Actual. Tras esto, los clientes asignados a los nodos Steiner son almacenados en el vector de punteros a Nodo *extra*, y sus variables actualizadas de forma que no están en la Solución ni asignados a ningún Nodo. Un bucle «for» recorre todos los nodos de *extra*, que cogiendo el Nodo l-ésimo más cercano al cliente, compara tres casos distintos: asignar el nodo a Nodo l-ésimo, visitarlo antes, o visitarlo después. Se escoge el caso óptimo y si se da una mejora en la solución se actualiza. La función no devuelve nada.

4.3.14. PI

Función «void» que implementa el Procedimiento de Intercambio. En ella se crea un array de números del 1 al $|U|$ que son ordenados de forma aleatoria, este array será usado para acceder de forma aleatoria a todos los clientes en un bucle «for», donde en cada iteración se inicia otro bucle «for» que recorre los T nodos más cercanos al cliente (donde T es un parámetro de entrada que por defecto en el programa vale 5), hallados mediante la función «elesimo», de forma que l toma los valores entre 1 y T. Para cada cliente y su Nodo l-ésimo, se crea la Nueva Solución, con la que trabajaremos durante el resto del bucle, que es igualada a la Solución Actual. A continuación, se comprueba que el Nodo l-ésimo no sea un nodo Steiner no visitado; si lo fuera, se pasaría al siguiente Nodo l-ésimo, si no, se comprueba si comparte anillo con el cliente. En este caso, se intercambian los nodos intercambiando todas sus variables (salvo «id», «c1» y «c2»). Si no comparten anillo, se comprueba que el intercambio sea factible comprobando que ambos anillos admitan el cambio de capacidad (cuando se intercambian dos nodos, también se intercambian sus clientes asignados). Si esto se cumple, se intercambian todas sus variables, salvo «id», «c1» y «c2». Se utiliza «actanil» en cualquiera de los casos tras el intercambio (si no comparten anillo, se aplica «actanil» en el anillo del Nodo cliente y del Nodo l-ésimo). Finalmente se comparan Nueva Solución con Solución Actual (que no ha sido modificada desde la creación de Nueva Solución) mediante «compSol». Esta función no devuelve nada.

4.3.15. PP

Función «void» que implementa el Procedimiento de Perturbación. Se toman I (parámetro que vale $\lceil alpha * |U| + 0{,}5 \rceil$) clientes aleatorios (con el mismo método que en «PENS»), se introducen en un vector de punteros a Nodo llamado *extra*, y se eliminan de la solución, así como todos los clientes que

tengan asignados (que también son incluidos en *extra*. Esto se hace mediante un bucle «for» que aparte de actualizar las variables de los clientes seleccionados, busca los clientes que tenga asignados para repetir con ellos el proceso. Tras esto, otro bucle «for» recorre todos los Nodo del ahora actualizado *extra*, con el que se anida otro «for» que recorre cada anillo, distinguiendo dos casos: si el anillo está vacío, el cliente será su primer (y de momento único) nodo visitado. Si no lo está, comparamos tres opciones, visitarlo antes o después del nodo visitado, o asignarlo al mismo. Escogemos la opción óptima y actualizamos la solución.

4.3.16. quitasig

Función «void» usada cuando un nodo asignado se libera, es decir, deja de ser asignado al nodo correspondiente. Esta función tiene como objetivo el eliminar de forma efectiva el Nodo asignado del vector «asignados» de su Nodo padre. Un bucle «for» recorre los nodos asignados del Nodo padre, cuando la «id» del Nodo coincide con la del que queremos liberar, se usa la función «erase» sobre esta posición del vector, borrando así el Nodo asignado. En el programa, tras aplicar esta función se aplica inmediatamente «actanil», de forma que se actualice toda la información del anillo y sus nodos.

4.4. Resultados

Se probaron cinco ejemplos, que se basan en los ejemplos de la librería TSPLIB con 51, 76, y 101 nodos. Los parámetros fueron idénticos a los utilizados en el artículo de Z. Naji-Azimi et al., a saber: $IterMax = 2000$, $T = 0{,}2 * |V|$, I=$0{,}5 * |U|$. Se considera que el coste de asignación y visita son idénticos: la distancia euclídea entre los nodos. Se recogen los resultados en la siguiente tabla, tras realizar 10 pruebas con cada ejemplo:

Ejemplo	|U|	Q	Tiempo medio (segundos)	Mejor Coste	Coste Medio	Peor Coste
A10-n051-m03	12	5	6.823	405.595	408.054	413.151
A25-n076-m03	37	14	34.497	547.542	552.063	574.846
A26-n076-m04	37	11	27.704	639.661	651.209	657.188
A27-n076-m05	37	9	28.883	723.457	729.336	735.914
A34-n101-m03	25	10	95.473	465.299	431.178	474.348

Podemos comprobar que los resultados de este programa están alejados de los conseguidos por los investigadores originales. Esto se puede explicar por la falta de implementación del Procedimiento de Lin-Kernighan, que reordena los nodos de cada anillo reduciendo el coste total. Respecto al aumento en los tiempos, dado que desconocemos el código fuente del programa usado por Z. Naji-Azimi et al. es difícil de analizar, pero podría influir la estructura de la solución (ya que es creada en casi todas las funciones, a veces hasta $N * T$ veces), ya que en este programa requiere cada copia o creación de la solución dos bucles (uno de N iteraciones y otro anidado tantas iteraciones como la suma del número de nodos visitados por cada anillo multiplicado por m).

Bibliografía

[1] R. BALDACCI, M. DELL'AMICO, J. SALAZAR GONZÁLEZ, *The Capacitated m-Ring-Star Problem*, Operations Research, Vol.55, No.6, pp 1147-1162, 2007.

[2] M. FISCHETTI, J. SALAZAR GONZÁLEZ, P. TOTH, *A branch-and-cut algorithm for the symmetric generalized traveling salesman problem*, Operations Research, Vol. 45, No. 3, pp 378-394, 1997.

[3] K. HELSGAUN, *An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic*, European Journal of Operational Research, Vol. 126, pp 106-130, 2000.

[4] S. LIN Y B.W. KERNIGHAN, *An effective heuristic algorithm for the traveling salesman problem*, Operations Research, Vol. 21, No. 2, pp 498-516, 1973.

[5] Z. NAJI-AZIMI, M. SALARI, P. TOTH, *A heuristic procedure for the Capacitated m-Ring-Star Problem*, Journal of Operational Research, Vol. 207, No. 3, pp 1227-1234, 2010.

[6] Z. NAJI-AZIMI, M. SALARI, P. TOTH, *An Integer Linear Programming based heuristic for the Capacitated m-Ring-Star Problem*, Journal of Operational Research, Vol. 217, No. 1, pp 17-25, 2012.

[7] RUPRECHT-KARLS-UNIVERSITÄT HEIDELBERG, *TSPLIB*, http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.