Trabajo Fin de Máster

El problema del anillo-estrella

Autor:
José Ángel Iranzo Sanz

Directora:
Dra. Herminia I. Calvete Fernández

Máster en Modelización Matemática, Estadística y Computación

Prólogo

Resumen

Sea $G = (V, E \cup A)$ un grafo mixto, donde $V = \{0, 1, \ldots, n\}$ es el conjunto de nodos, $E = \{[i, j] : i, j \in V, i \neq j\}$ es el conjunto de ejes y $A = \{(i, j) : i, j \in V\}$ es el conjunto de arcos. Cada eje $e = [i, j] \in E$ tiene asociado un coste no negativo $c_e = c_{ij}$ y cada arco $a = (i, j) \in A$, que representa la asignación del nodo i al nodo j, tiene asociado un coste no negativo $d_a = d_{ij}$.

El problema del anillo-estrella (*Ring Star Problem, RSP*) consiste en diseñar un ciclo simple (anillo) que pasa por el nodo 0 usando los ejes de E y asignar cada nodo no visitado por el anillo al nodo más cercano del anillo usando los arcos de A. El objetivo es minimizar el coste del ciclo mas el coste de las asignaciones. Este problema tiene numerosas aplicaciones prácticas en el campo del diseño urbano de redes de telecomunicaciones.

Este trabajo se divide en cuatro capítulos. El primero sirve como introducción y justificación del problema, y plantea la relación del RSP con su aplicación práctica en el diseño de redes de telecomunicaciones.

En el segundo capítulo, se describe formalmente el problema, se detallan algunas de las formulaciones como un problema de programación lineal entera que se han propuesto en la literatura y se describen los algoritmos utilizados para resolver dichos problemas. Finalmente, se revisan algunas variantes del RSP.

En el tercer capítulo, se propone una transformación del RSP en un problema del viajante generalizado (*GTSP*) que permite resolver el RSP aplicando las técnicas y algoritmos existentes para el GTSP.

Por último, en el cuarto capítulo, se propone un algoritmo evolutivo para resolver de forma heurística el RSP y se muestran algunos de los resultados experimentales obtenidos a partir de una implementación del algoritmo en C++, cuyo código se presenta en el anexo.

Abstract

Let $G = (V, E \cup A)$ be a mixed graph, where $V = \{0, 1, \ldots, n\}$ is the node set, $E = \{[i, j] : i, j \in V, i \neq j\}$ is the edge set and $A = \{(i, j) : i, j \in V\}$ is the arc set. Each edge $e = [i, j] \in E$ is associated with a nonnegative cost $c_e = c_{ij}$ and each arc $a = (i, j) \in A$, which represents that node i is assigned to node j, is associated with a nonnegative cost $d_a = d_{ij}$.

The Ring Star Problem (RSP) is the problem of designing a simple cycle (ring) that passes through node 0 using the edges and then assigning each non-visited node to a visited one using the arcs. The objective is to minimize the sum of ring and assignment costs. The problem has many practical applications in the design of urban optical telecommunication networks.

This report is divided into four chapters. The first one introduces and justifies the problem, and discusses the use of the RSP to model real applications in the design of telecommunications networks.

The second chapter states the problem, presents some of the integer linear programming formulations proposed in the literature and describes algorithms used to solve these problems. The chapter concludes with a review of some related problems.

In the third chapter a transformation of the RSP into a generalized traveling salesman problem (*GTSP*) is proposed that allows us to solve the RSP using existing techniques and algorithms for the GTSP.

Finally, in the fourth chapter an evolutionary algorithm to solve the RSP heuristically is developed and some experimental results to prove its performance are shown. The experiment is carried out with a C++ implementation of the algorithm, which is attached at the end of this report.

Índice

1

Introducción

1.1. Descripción del problema

El problema del anillo-estrella (*The Ring Star Problem, RSP*) consiste en encontrar un ciclo simple a través de un subconjunto de nodos de un grafo, de manera que se minimiza el coste total obtenido como la suma del coste del ciclo y el coste de asignación de cada uno de los nodos que no están en el ciclo al nodo más cercano que sí lo está.

Este problema aparece a menudo en problemas reales de diseño de redes de telecomunicaciones. Una red de telecomunicaciones genérica está formada por clientes, concentradores, una unidad central, una serie de enlaces que conectan cada cliente con un concentrador y una red de conexiones que interconecta los concentradores o los conecta a la unidad central. Una forma de abordar el diseño de estas redes consiste en utilizar ciertos clientes como concentradores, conectar cada uno de los clientes restantes a uno de los utilizados como concentradores, formando así estructuras de tipo estrella, e interconectar los clientes-concentradores y la unidad central mediante una estructura de anillo. Puede verse un ejemplo de esta estructura de tipo anillo-estrella en la Figura 1.1.

La estructura de tipo anillo se utiliza para diseñar gran parte de las redes de comunicación por fibra óptica con el fin de prevenir las pérdidas de conexión causadas por un fallo puntual en algún punto de la red. El anillo se construye con un cable que contiene un haz de fibras ópticas. Se selecciona un cierto subconjunto de clientes en los que se instalarán concentradores, se interconectan los concentradores y se asigna cada cliente al concentrador más cercano. Cada cliente recibe dos hilos de fibra óptica que lo conectan con la central de intercambio de datos (unidad central) a través de los dos sentidos posibles del anillo.

Supongamos que un cliente i que está en el anillo está comunicándose con otros a través de la porción del anillo que parte de la unidad central en el sentido de las agujas

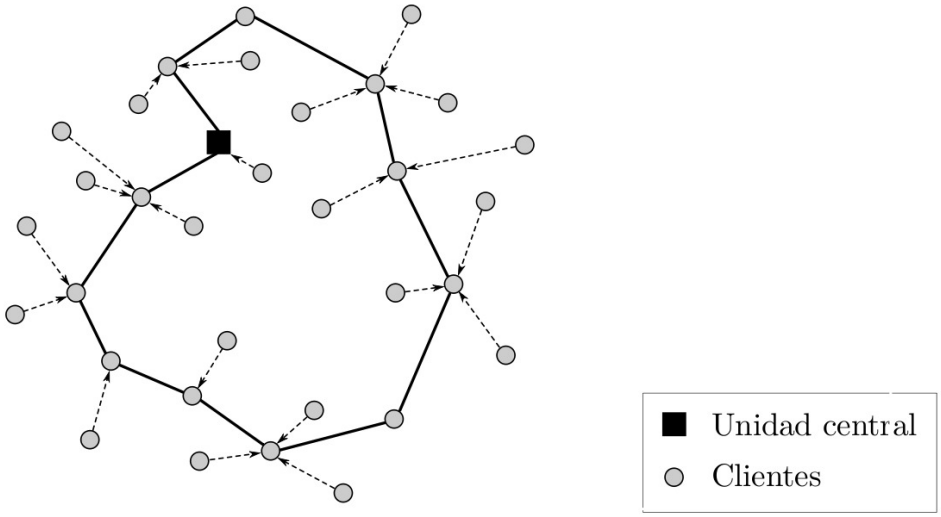


Figura 1.1: Ejemplo de estructura de tipo anillo-estrella

de reloj. Si un eje de esa porción falla, es posible redirigir la información desde/hacia el cliente i a través de la porción contraria del anillo, que no habría sido afectada por el fallo. De esta manera se garantiza un servicio de comunicación continua ante eventuales fallos o rupturas de los cables de fibra óptica que trasmiten la información.

Por otro lado, la estructura anillo-estrella, en general, provoca un menor coste. En efecto, el coste total de la red de comunicación depende de varios factores tales como el coste de los cables de fibra óptica o los dispositivos necesarios para la comunicación, pero, en general, el coste principal se debe a las excavaciones necesarias para enterrar los tubos con cables de fibra óptica bajo tierra. Por tanto, las empresas de telecomunicaciones tratan de reducir este coste tanto como sea posible. Supongamos que todos los clientes salvo uno, el cliente i, están localizados en un anillo circular y conectados entre sí mediante un cable que recorre el anillo. Para conectar el nuevo cliente i a la red, puede ampliarse el anillo para que pase por el nuevo cliente, lo que obliga a conectarlo con dos de los clientes del anillo o, si la distancia a un cliente que ya esté en el anillo es pequeña, puede considerarse la posibilidad de conectar el nuevo cliente i con el cliente más cercano del anillo mediante un eje, lo que, en general, es menos costoso ya que los costes de excavación son menores. En este caso, las comunicaciones del cliente i se realizan a través de este eje, en el que se enterraría el tubo que contiene el cable. Si se produjera un fallo puntual en este cable, el cliente i quedaría desconectado de la red, mientras que el resto de los clientes podrían continuar comunicados. Generalmente, la empresa está más dispuesta a asumir los inconvenientes generados por la incomunicación de un cliente que el costo de excavación asociado a agrandar el anillo para la inclusión de un nuevo cliente.

En ocasiones la situación real es más compleja por lo que es necesario añadir más

condiciones al problema para que se ajuste más fielmente a la realidad. Por ejemplo, en la actualidad, el cable que interconecta los concentradores contiene, en su forma estándar, un haz de 100 fibras ópticas que puede abastecer, a lo sumo, a 50 clientes. Si el número de clientes es superior a 50 surge el problema de cómo diseñar una red que llegue a todos los clientes conservando las buenas propiedades de las soluciones de tipo anillo-estrella. Una posibilidad consiste en diseñar una red con varios anillos-estrella conectados a una misma unidad central, cada uno de ellos con una cantidad máxima de clientes asociados. Además, con este modelo se consigue aún mayor fiabilidad, ya que si uno de los anillos queda totalmente inutilizado, solamente los clientes de ese anillo quedan incomunicados. Este problema, que generaliza el RSP, se denomina el problema del m-anillo-estrella con restricciones de capacidad (*The Capacitated m-Ring Star Problem, CmRSP*).

1.2. Antecedentes

Aunque algunas variantes del problema de obtención de una estructura de anillo-estrella habían sido consideradas con anterioridad, el RSP con las características que se estudian en este trabajo fue introducido por Labbé et al. (2004), que formulan el problema como un problema de programación lineal entera y desarrollan un algoritmo de resolución exacto a partir de un método de ramificación y corte basado en ciertas propiedades del poliedro definido por las restricciones. Posteriormente, Kedad-Sidhoum y Nguyen (2010) proponen una nueva formulación como un problema de programación lineal entera, que está basada en caminos simples no dirigidos, y utilizan un algoritmo similar al propuesto por Labbé et al. (2004) para su resolución. Simonetti et al. (2011) proponen una nueva formulación que transforma el problema en uno de tipo *Steiner Tree Problem* (STP). Basándose en el problema transformado, desarrollan un nuevo algoritmo de resolución del problema basado en técnicas de ramificación y corte.

Además de los algoritmos exactos, basados en general en el uso de técnicas de ramificación y corte, en la literatura se ha propuesto el uso de algoritmos metaheurísticos para resolver el problema. Pérez et al. (2003) proponen un algoritmo que combina procedimientos de búsqueda en entornos variables(*Variable Neighbourhood Search, VNS*) y de búsqueda tabú (*Tabu Search*, TS). Renaud et al. (2004) proponen un algoritmo que combina heuristicas voraces y técnicas evolutivas. Dias et al. (2006) proponen un algoritmo metaheurístico híbrido para la resolución del RSP basado en procedimientos *Greedy Randomized Adaptive Search Procedures* (GRASP) y *General Variable Neighbourhood Search* (GVNS).

En la literatura se han estudiado otros problemas directamente relacionados con el

RSP. El problema del ciclo mediano (*Median Cycle Problem*, MCP), estudiado por Pérez et al. (2003), Renaud et al. (2004) y Labbé et al. (2005), consiste en encontrar un ciclo simple con el menor coste posible sujeto a un cota superior sobre el coste total de las asignaciones de los nodos no visitados por el ciclo. En ocasiones el RSP aparece en la literatura bajo el nombre del problema del ciclo mediano, aunque en este trabajo se distinguirán ambos problemas, considerando el MCP una generalización del RSP. Lee et al. (1998) consideran el problema del anillo-estrella con nodos de Steiner, que generaliza el RSP añadiendo al problema original los llamados nodos de transición (también conocidos como nodos de Steiner o concentradores) que pueden ser utilizados para construir el anillo solución y, de esta manera, reducir costes. El uso de nodos de transición también aparece frecuentemente en la definición del CmRSP, que ha sido estudiado por Baldacci et al. (2007), Mauttone et al. (2007) y Naji-Azimi et al. (2010). El problema del anillo-estrella multidepósito, introducido por Baldacci y Dell'Amico (2010), considera el problema de diseñar una colección de anillos-estrella, cada uno de ellos partiendo de una unidad central distinta. El problema del anillo-estrella bi-objetivo, estudiado por Liefooghe et al. (2010), plantea el problema del anillo-estrella como un problema bi-objetivo, tratando de minimizar simultáneamente el coste del ciclo y el coste de las asignaciones de los clientes no visitados.

2

El problema del anillo-estrella (RSP)

2.1. Definición del problema

El RSP puede ser formulado como un problema de optimización en teoría de grafos. Sea $G = (V, E \cup A)$ un grafo mixto donde V es el conjunto de nodos, E es el conjunto de ejes y A el conjunto de arcos. Sea 0 el nodo central o nodo raíz por el que pasa necesariamente cualquier anillo solución del problema y sea $U = V \setminus \{0\}$ el conjunto de nodos que representan a los clientes. El conjunto $E = \{[i, j] : i, j \in V, i \neq j\}$ representa el conjunto de los posibles ejes del anillo. Cada uno de sus elementos $e = [i, j] \in E$ tiene asociado un coste no negativo $c_e = c_{ij}$. El conjunto A representa las posibles asignaciones, es decir, $A = \{(i, j) : i, j \in V\}$ y el arco (i, j) representa la asignación del nodo i al nodo j. Cada arco $a = (i, j) \in A$ tiene asociado un coste de asignación no negativo $d_a = d_{ij}$. El general, consideraremos $d_{ii} = 0$ para todo $i \in V$, pero podría considerarse que d_{ii} es, por ejemplo, el coste requerido por utilizar el cliente i como concentrador, y en ese caso se tendría $d_{ii} \geq 0$.

Un anillo R es un ciclo simple que visita un subconjunto de nodos que incluye el nodo central. Es decir, un anillo es un camino que comienza y acaba en el nodo 0 sin repetir ninguno de los nodos por los que pasa (salvo el 0). Para facilitar la notación denotaremos a cada anillo como $R = (V_R, E_R)$, siendo $V_R \subseteq V$ el conjunto de nodos que recorre el anillo y $E_R \subseteq E$ el conjunto de ejes por los que pasa. De esta forma, un anillo de longitud $m = |V_R|$ puede expresarse como una sucesión ordenada de nodos

$$R = [0 = r_0, r_1, r_2, \ldots, r_{m-1}, r_m]$$

que satisfacen que

$$V_R = \{0, r_1, \ldots, r_{m-1}, r_m\}$$

$$E_R = \{[0, r_1], \ldots, [r_{m-1}, r_m], [r_m, 0]\}$$

Los nodos de $V \setminus V_R$ se asignan al nodo más cercano en el anillo.

El coste del anillo R se define como la suma del coste de sus ejes:

$$c_R = \sum_{e \in E_R} c_e$$

Para cada nodo $i \in V \setminus V_R$, se define el coste de su asignación al anillo R como el menor de los costes de asignación de i a los nodos en V_R. Por tanto, el coste de asignación de los nodos del grafo al anillo R es:

$$d_R = \sum_{i \in V \setminus V_R} \min_{j \in V_R} d_{ij}$$

Teniendo en cuenta que el coste de asignación de un nodo a sí mismo es cero, podemos considerar que un nodo está asignado a sí mismo si y solo si está en el anillo. De esta forma, el coste de asignación del anillo puede reescribirse como:

$$d_R = \sum_{i \in V} \min_{j \in V_R} d_{ij}$$

En la Figura 2.1 puede verse la representación de una solución del RSP.

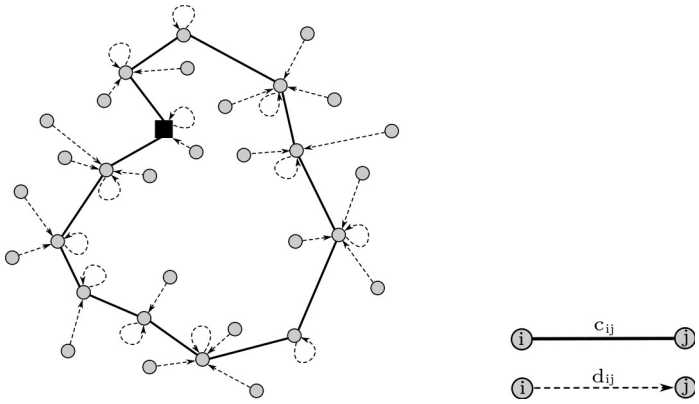


Figura 2.1: Una solución factible del RSP

Sea $\mathcal{R}$ el conjunto de anillos en G que pasan por el nodo 0. El objetivo del RSP es encontrar un anillo $R \in \mathcal{R}$ tal que:

$$c_R + d_R = \min_{S \in \mathcal{R}} \{c_S + d_S\}$$

El RSP generaliza el problema del viajante clásico (TSP), que aparece cuando $A = \emptyset$, y es, por tanto, un problema NP-duro (Karp (1972)).

A continuación se van a presentar distintas formulaciones del RSP como un problema de programación lineal entera mixta, que se han propuesto en la literatura. La primera formulación, basada en ciclos, fue propuesta por Labbé et al. (2004). Kedad-Sidhoum

y Nguyen (2010) proponen una formulación muy semejante basada en caminos simples. Finalmente, Simonetti et al. (2011) proponen una transformación del RSP al problema del árbol de Steiner (Hwang y Richards (1992)) y presentan una formulación para dicho problema transformado.

2.2. Formulación basada en ciclos (Labbé et al.)

Para cada eje $e = [i, j] \in E$, sea $x_e = x_{ij}$ la variable binaria que toma el valor 1 si el eje e aparece en el anillo y toma el valor 0 en otro caso. Para cada arco $a = (i, j) \in A$, sea $y_a = y_{ij}$ la variable binaria que toma el valor 1 si el nodo i se asigna al nodo j, que forma parte del anillo, y toma el valor 0 en otro caso. Un nodo i se considera que está asignado a sí mismo si la variable $y_{ii} = 1$, es decir, si el nodo i está en el anillo. Notemos que no es necesario tener en cuenta las variables de tipo y_{0i}, ya que para todo anillo debe cumplirse $y_{0i} = 0$ si $i \neq 0$ y $y_{00} = 1$.

Para facilitar la notación, para cada subconjunto de nodos $S \in V$ definimos:

$$E(S) = \{[i, j] \in E : i, j \in S\}$$

$$\delta(S) = \{[i, j] \in E : i \in S, j \notin S\}$$

Si S solo tiene un nodo, es decir $S = \{i\}$, para simplificar la notación escribiremos $\delta(i)$ en lugar de $\delta(\{i\})$. Para cada subconjunto de ejes $E' \subseteq E$, denotaremos

$$x(E') = \sum_{e \in E'} x_e$$

El problema RSP puede formularse como:

$$\min \sum_{e \in E} c_e x_e + \sum_{a \in A} d_a y_a \tag{2.1a}$$

sujeto a

$$x(\delta(0)) = 2 \tag{2.1b}$$

$$x(\delta(i)) = 2y_{ii} \qquad \forall i \in U \tag{2.1c}$$

$$\sum_{j \in V} y_{ij} = 1 \qquad \forall i \in U \tag{2.1d}$$

$$x(\delta(S)) \geq 2 \sum_{j \in S} y_{ij} \qquad \forall S \subset U, \forall i \in S \tag{2.1e}$$

$$x_e,\ y_a \in \{0, 1\} \qquad \forall e \in E, a \in A \tag{2.1f}$$

En esta formulación, la restricción (2.1b) asegura que el nodo 0 tiene grado 2, es decir, existen exactamente dos ejes en la solución que lo conectan con otros nodos. Análogamente, la restricción (2.1c) asegura que, en una solución factible, un nodo $i \in U$ está en

el anillo ($y_{ii} = 1$) si y sólo si tiene grado 2. La restricción (2.1d) garantiza que cada nodo $i \in U$ o bien pertenece al anillo ($y_{ii} = 1$) o bien está asignado a un nodo $j \in V$ ($y_{ii} = 0$ y $y_{ij} = 1$).

La restricción (2.1e) es una restricción de conectividad ya que asegura que un subconjunto $S \subseteq U$ está conectado a su complementario por al menos dos ejes del anillo siempre que al menos un nodo $i \in S$ esté asignado a otro $j \in S$. Además, garantiza que la solución contiene un único ciclo, es decir, sin la restricción (2.1e) el modelo admitiría soluciones como la que se ilustra en la Figura 2.2, en la que se aprecia que el conjunto S y el nodo i señalados no cumplen esta restricción.

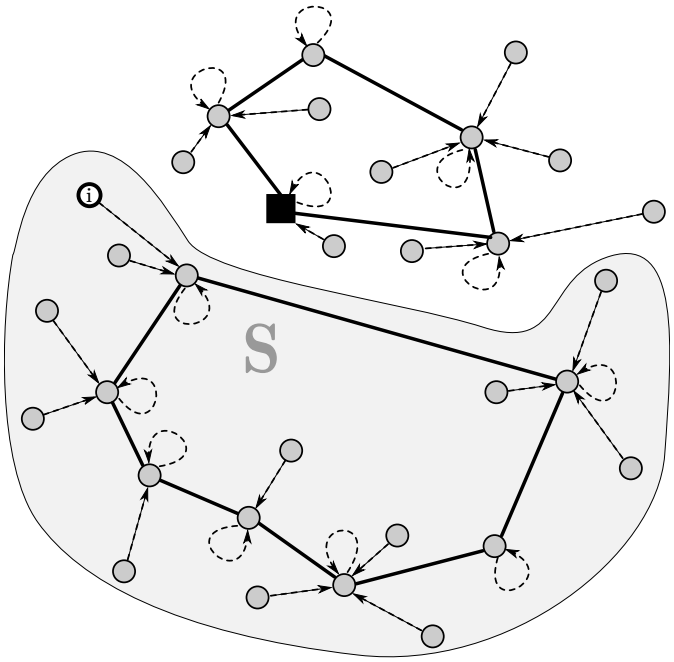


Figura 2.2: Solución no factible: no cumple la restricción (2.1e)

La combinación de las restricciones (2.1b), (2.1c) y (2.1f) garantiza que la solución contendrá al menos un anillo que pasa por el nodo 0. La combinación de las restricciones (2.1b), (2.1c), (2.1d), (2.1e) y (2.1f) asegura que cada nodo que no está en el anillo está asignado a un nodo que sí lo está.

Notemos que el modelo formulado en (2.1) impone, implícitamente, que el anillo visita al menos tres nodos, incluido el nodo 0. Si el anillo visitase menos de tres nodos uno de los que sí visitaría sería el 0 y la restricción (2.1b) no se cumpliría, ya que se tendría $x(\delta(0)) \leq 2$. El hecho de que el anillo visite al menos tres nodos aumenta la fiabilidad de la solución cuando se trata de problemas en redes de telecomunicaciones ya que, en caso de fallo de uno de los ejes, permite redirigir la información evitando perdidas de conexión. Si ésta no fuera una restricción del problema, éste podría resolverse fácilmente por enumeración ya que las soluciones que consisten en un anillo degenerado (sólo el nodo

0 o sólo el nodo 0 y otro nodo) pueden ser enumeradas fácilmente.

Este modelo puede ser extendido para incorporar restricciones de capacidad en los concentradores. Por ejemplo, si un nodo j en el anillo no puede servir a más de l_j clientes externos, bastaría añadir al modelo las restricciones:

$$\sum_{(i,j)\in A} y_{ij} \leq l_j y_{jj} \qquad \forall j \in V$$

Como se indica en Labbé et al. (2004), estas restricciones pueden complicar mucho la resolución del modelo.

Una de las características de los problemas de programación lineal entera es que la introducción de restricciones adicionales que son verificadas por las soluciones óptimas facilitan, en muchos casos, la resolución del problema ya que permiten mejorar las cotas que se obtienen para el problema lineal relajado. Veamos algunas de las desigualdades válidas para el conjunto de soluciones factibles enteras que fortalecen la relajación lineal del modelo (2.1).

Las desigualdades desarrolladas para el problema del politopo ciclo, estudiado por Bauer (1997), son también válidas para el RSP. Consideremos las siguientes desigualdades:

$$x(E(H)) + x(T) \leq \sum_{i\in H} y_{ii} + \frac{|T|-1}{2} \tag{2.2}$$

para todo $H \subseteq V$ y $T \subseteq \delta(H)$, satisfaciendo

(i) $\{i,j\} \cap \{k,l\} = \emptyset$ para $[i,j],[k,l] \in T$ si $[i,j] \neq [k,l]$.

(ii) $|T| \geq 3$ e impar.

Estas desigualdades son estándares para problemas de ciclos y se deducen fácilmente como suma de dos desigualdades. Utilizando las restricciones de tipo (2.1b) y (2.1c) para todo $i \in H$ y sabiendo que $x(T) \leq x(\delta(H))$ se tiene que

$$2x(E(H)) + x(\delta(T)) \leq 2x(E(H)) + x(\delta(H)) = \sum_{i\in H} y_{ii} \tag{2.3}$$

Por otro lado, utilizando que $x_e \leq 1$ para todo $e \in T$ se tiene que

$$x(T) \leq |T| \tag{2.4}$$

Así, sumando las desigualdades (2.3) y (2.4), dividiendo por dos y tomando la parte entera en ambos lados obtenemos la desigualdad (2.2).

Otra familia de desigualdades válidas aparece debido a que ciertas asignaciones son incompatibles. En concreto, las variables y_{ij} e y_{ik} son incompatibles cuando $j \neq k$, es decir

$$y_{ij} + y_{ik} \leq 1 \qquad \forall j, k \in V, j \neq k \tag{2.5}$$

Análogamente, se dan también la siguientes desigualdades

$$y_{ij} + y_{jk} + y_{ki} \leq 1 \qquad \forall i, j, k \in U \text{ diferentes} \tag{2.6}$$

Finalmente, combinando las restricciones (2.3), con $S = \{i, j, k\} \subseteq U$, y las restricciones (2.5) obtenemos nuevas desigualdades válidas para el RSP:

$$x(\delta(S)) \geq 2(y_{ij} + y_{jk} + y_{ki}) \qquad \forall i, j, k \in U \text{ diferentes} \tag{2.7}$$

2.2.1. Algoritmo

Para resolver el RSP, Labbé et al. (2004) proponen un algoritmo de ramificación y corte que permite encontrar un anillo, $\overline{C}$, solución óptima del RSP. Los algoritmos de ramificación y corte son usados para resolver problemas generales de programación lineal entera y combinan métodos de ramificación y acotación con métodos de planos de corte.

Los métodos de ramificación y acotación son métodos enumerativos que recorren implícitamente todas las soluciones enteras factibles (Nemhauser y Wolsey (1999)). Aunque el número de posibles soluciones aumenta de forma exponencial con respecto al numero de variables del problema, los algoritmos de ramificación y acotación introducen reglas para poder descartar conjuntos de soluciones sin evaluarlas una a una. En el proceso de ramificación se añaden restricciones al modelo que fuerzan a que una de las variables sea entera y en el proceso de acotación se permite cerrar ramas cuando se tiene la garantía de que las nuevas soluciones enteras que se introducirán al ramificar no proporcionan un valor óptimo para el problema original.

Los métodos de corte, en cambio, son métodos no enumerativos que introducen sucesivos cortes sobre la región factible hasta localizar la mejor solución entera. Un plano de corte para un problema de programación entera es una restricción que reduce la región factible del problema relajado (sin restricciones de integridad) sin eliminar soluciones factibles para el problema original.

Denotaremos por $V(\overline{C})$ el conjunto de nodos en la solución. A continuación se describen los pasos del algoritmo:

Paso 1: Inicialización.

Se inicializa el contador de iteraciones, $t = 0$, y se calcula una cota superior $\overline{z}$ del RSP óptimo:

Empezando por $V(\overline{C}) = \{0\}$, se inserta sucesivamente un nodo $i \notin V(\overline{C})$ que minimice

$$L(i, \lambda) = \lambda \; inc(\overline{C}, i) + (1 - \lambda)(-dec(\overline{C}, i))$$

donde $inc(\overline{C}, i)$ es el menor de los incrementos producido en el coste del anillo $\overline{C}$ al insertar i, y $dec(\overline{C}, i)$ es el decrecimiento producido en el coste de asignación. Esta operación se repite siempre y cuando $L(i, \lambda)$ decrezca con una nueva inserción. El cálculo se realiza para $\lambda \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$ y se selecciona la mejor solución. Los valores de λ se han elegido para generar cinco anillos, cada uno de los cuales asigna un peso diferente al coste del anillo y al coste de asignación.

Por otro lado se resuelve el subproblema lineal inicial que se formula como:

$$\begin{aligned}
\min \quad & \sum_{e \in E} c_e x_e + \sum_{a \in A} d_a y_a \\
\text{sujeto a} \quad & \\
& x(\delta(0)) = 2 \\
& x(\delta(i)) = 2y_{ii} && \forall i \in U \\
& \sum_{j \in V} y_{ij} = 1 && \forall i \in U \\
& x_{0i} \leq y_{ii} && \forall i \in U
\end{aligned}$$

Se inserta su solución en la lista $\mathcal{L}$.

Paso 2: Finalización o selección de un subproblema.

Si $\mathcal{L} = \emptyset$ el algoritmo termina. En otro caso, se selecciona el subproblema de la lista $\mathcal{L}$ con menor valor de la función objetivo y se va al Paso 3.

Paso 3: Solución del subproblema.

Se actualiza la iteración $t = t + 1$. Sea z el valor de la función objetivo del subproblema seleccionado. Si $z \geq \overline{z}$, se va al Paso 2. En otro caso, si la solución es factible para el RSP, se toma $\overline{z} = z$, se actualiza $\overline{C}$ al nuevo anillo solución e se vuelve al Paso 2. En otro caso, se continúa con el Paso 4.

Paso 4: Heurística basada en Programación Lineal.

Si la solución es fraccionaria y t es múltiplo de 5, se aplica el siguiente procedimiento heurístico:

Dada la solución fraccionaria (x^*, y^*), se ordenan las variables x^*_{ij} en orden decreciente con respecto a los valores c_{ij} y se toman, uno a uno, los ejes asociados hasta obtener el anillo C^*. Si el nodo 0 no está en C^*, se introduce en la mejor posición. Finalmente, se aplica el procedimiento 2-opt desarrollado por Lin (1965) para tratar de reducir el coste del anillo C^*. Sea z^* el valor de la función objetivo de la solución C^*. Si $z^* \leq \overline{z}$, se toma $\overline{z} = z^*$ y se actualiza $\overline{C}$ a C^*. A continuación se pasa al Paso 5.

Paso 5: Generación y separación de restricciones.

Se introducen hasta 300 restricciones de conectividad (2.1e) no satisfechas y las restricciones (2.2), (2.6) y (2.7). Si no se pueden generar restricciones se va al Paso 6 y, en otro caso, al Paso 3.

Paso 6: Ramificación.

Se crean dos subproblemas por ramificación de una variable no entera y_{ii} o x_{ij}. La primera estrategia de ramificación consiste en encontrar una variable y_{ii}. Para ello, se aplica la regla de *ramificación fuerte* con las 5 variables y_{ii} con valor fraccionario más cercano a 0.5. Si todas estas variables son enteras, se selecciona una variable x_{ij} usando el mismo criterio. Se insertan los dos subproblemas generados en $\mathcal{L}$ y se continúa al Paso 2.

2.3. Formulación basada en caminos (Kedad-Sidhoum y Nguyen)

Esta formulación es similar a la presentada por Labbé et al. (2004) y, por tanto, se utilizará la misma notación que en la Sección 2.2 para plantearla. Llamaremos s al nodo 0 y añadiremos un nodo auxiliar t copia del nodo s obteniendo así un nuevo grafo $G' = (V', E' \cup A')$ donde $V' = V \cup \{t\}$, $E' = E \cup \{[s,t]\} \cup \{[t,u] : [s,u] \in E\}$ y $A' = A$.

Así, una solución del RSP en G consiste en un camino desde s hasta t en G' que asigna cada nodo que no está en el camino a uno que si lo está (excepto t).

La formulación como un problema de programación lineal entera mixta a partir del grafo G' es similar a la anteriormente presentada basada en ciclos. La función objetivo es la misma y las restricciones que estaban relacionadas con el nodo 0, ahora están expresadas en función de s y de t. Tenemos así:

$$\min \quad \sum_{e \in E'} c_e x_e + \sum_{a \in A'} d_a y_a \tag{2.8a}$$

sujeto a

$$x(\delta(s)) = 1,\ x(\delta(t)) = 1 \tag{2.8b}$$

$$x(\delta(i)) = 2y_{ii} \qquad \forall i \in V' \setminus \{s,t\} \tag{2.8c}$$

$$\sum_{j \in V'} y_{ij} = 1 \qquad \forall i \in V' \setminus \{s,t\} \tag{2.8d}$$

$$x(\delta(S)) \geq 2 \sum_{j \in S} y_{ij} \qquad \forall S \subseteq V' \setminus \{s,t\}, \forall i \in S \tag{2.8e}$$

$$x_e,\ y_a \in \{0,1\} \qquad \forall e \in E', a \in A' \tag{2.8f}$$

En el trabajo se señala que es posible añadir desigualdades válidas al problema relajado adaptando, cuando sea posible, las que se añadieron en la Sección 2.2 para la formulación de Labbé et al. (2004). La ventaja principal de la formulación introducida por Kedad-Sidhoum y Nguyen (2010) es que permite añadir ciertas desigualdades a partir del poliedro st-camino que no se dan para la formulación basada en ciclos. En particular, se dan la siguientes desigualdades st-camino *blossom*

$$x(\delta(H) \setminus T) - x(T) + (1 - |T|)x_{st} \geq 1 - |T| \tag{2.9}$$

para todo $H \subseteq V'$ y $T \subseteq \delta(H)$ con $|T| + |H \cap \{s,t\}|$ impar.

2.3.1. Algoritmo

La idea general del algoritmo de ramificación y corte usado por Kedad-Sidhoum y Nguyen (2010) para resolver el RSP es similar a la del algoritmo desarrollado por Labbé et al. (2004) descrito en la Sección 2.2. La principal diferencia es que en el algoritmo propuesto por Kedad-Sidhoum y Nguyen (2010) se introducen las restricciones st-camino blossom (2.9). Denotaremos con $V(\overline{C})$ el conjunto de nodos en el st-camino solución $\overline{C}$. La descripción del algoritmo es la siguiente:

Paso 1: Inicialización.

Se inicializa el contador de iteraciones $t = 0$ y se calcula una cota superior $\overline{z}$ del RSP óptimo de la siguiente manera:

Empezando con $V(\overline{C}) = \{s,t\}$, se inserta sucesivamente un nodo $i \notin V(\overline{C})$ que minimice

$$L(i,\lambda) = \lambda\, inc(\overline{C}, i) + (1 - \lambda)(-dec(\overline{C}, i))$$

donde $inc(\overline{C}, i)$ es el menor de los incrementos producido en el coste del st-camino $\overline{C}$ al insertar i y $dec(\overline{C}, i)$ es el decrecimiento producido en el coste de asignación. Esta operación se repite siempre y cuando $L(i, \lambda) < 0$ con un nueva inserción.

Este proceso se realiza para $\lambda \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$ y se selecciona la mejor solución. Los valores de λ se han elegido para generar cinco soluciones, cada una con pesos diferentes para el coste del anillo y el coste de asignación.

Por otro lado se resuelve el subproblema lineal inicial que se formula como:

$$\begin{aligned}
\min \quad & \sum_{e \in E'} c_e x_e + \sum_{a \in A'} d_a y_a \\
\text{sujeto a} \quad & \\
& x(\delta(s)) = 1,\ x(\delta(t)) = 1 \\
& x(\delta(i)) = 2y_{ii} && \forall i \in V' \setminus \{s, t\} \\
& \sum_{j \in V'} y_{ij} = 1 && \forall i \in V' \setminus \{s, t\} \\
& x_e \in \{0, 1\},\ y_a \geq 0 && \forall e \in E', a \in A'
\end{aligned}$$

Se inserta su solución en la lista $\mathcal{L}$.

Paso 2: Finalización o selección de un subproblema.

Si $\mathcal{L} = \emptyset$ el algoritmo termina. En otro caso, se selecciona el subproblema de la lista $\mathcal{L}$ con menor valor de la función objetivo y se va al Paso 3.

Paso 3: Solución del subproblema.

Se actualiza la iteración $t = t + 1$. Sea z el valor de la función objetivo del subproblema seleccionado. Si $z \geq \overline{z}$ se va al Paso 2. En otro caso, si la solución es factible para el RSP, se toma $\overline{z} = z$, se actualiza $\overline{C}$ al nuevo anillo solución y se vuelve al Paso 2. En otro caso, se continúa con el Paso 4.

Paso 4: Generación y separación de restricciones.

Se introducen restricciones de conectividad (2.8e) no satisfechas y restricciones blossom (2.9). Si no es posible generar restricciones se va al Paso 5 y, en otro caso, al Paso 3.

Paso 5: Ramificación.

Se crean dos subproblemas por ramificación de una variable no entera y_{ii} o x_{ij}. La primera estrategia de ramificación consiste en encontrar una variable y_{ii}. Para ello, se aplica la regla de *ramificación fuerte* con las 5 variables y_{ii} con valor fraccionario más cercano a 0.5. Si todas estas variables son enteras, se selecciona una variable

x_{ij} usando el mismo criterio. Se insertan los dos subproblemas generados en $\mathcal{L}$ y se continúa con el Paso 2.

2.4. Formulación basada en el STP (Simonetti et al.)

Simonetti et al. (2011) proponen una transformación del problema RSP en un problema de tipo STP (*Steiner Tree Problem*) con restricciones adicionales.

El *Steiner Tree Problem* (STP), también conocido como *Minimum Steiner Arborescence Problem*, es una generalización del clásico problema del árbol de expansión mínima (*Minimum Spanning Tree Problem, MSTP*). Sea un grafo $\overline{G} = (\overline{V}, \overline{A})$ donde cada uno de sus arcos tiene asociado un cierto coste. En el MSTP se trata de encontrar el árbol de mínimo coste que contiene todos los nodos de un grafo dado. En el STP, dado $r \in \overline{V}$, denominado nodo raíz, y dado un conjunto $R \subseteq \overline{V}$ de nodos, denominados nodos terminales, el objetivo del STP es encontrar el árbol de menor coste con raíz en r que contiene los nodos de R.

2.4.1. Construcción del grafo transformado

Dado $G = (V, E \cup A)$, el grafo del RSP definido en la Sección 2.1, se construye el grafo dirigido $\overline{G} = (\overline{V}, \overline{A})$ sobre el que se plantea el STP. El grafo $\overline{G}$ se estructura en dos niveles numerados como 1 y 2.

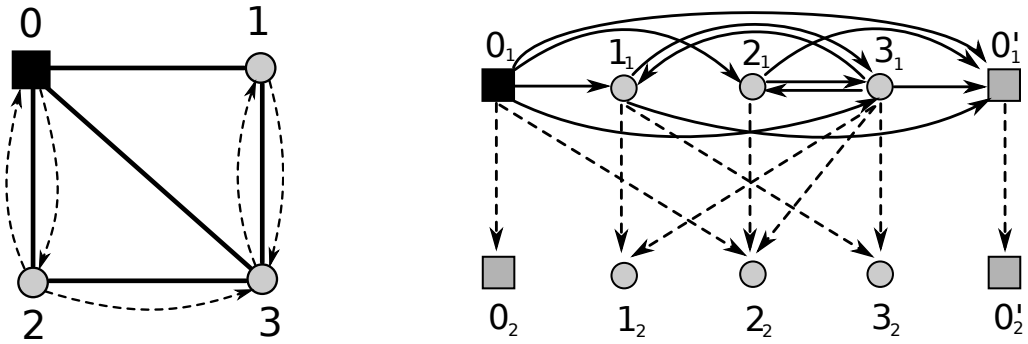
Para cada nodo $i \in V$ se crean dos nodos en $\overline{V}$, el nodo i_1 en el nivel 1 y el nodo i_2 en el nivel 2. Para el nodo 0, además de los nodos 0_1 y 0_2, se crean en el correspondiente nivel 2 nuevas copias denotadas como $0'_1$ y $0'_2$.

Para cada eje $[i, j] \in E$ con $i, j \neq 0$, se crean en $\overline{A}$ los arcos (i_1, j_1) y (j_1, i_1). Además, para cada eje del tipo $[0, i] \in E$ se incluyen en $\overline{A}$ los arcos $(0_1, i_1)$ e $(i_1, 0'_1)$. El resto de arcos de $\overline{A}$ son los de la forma (i_1, i_2) para todo $i \in V$, los (j_1, i_2) para todo $(i, j) \in A$ con $i \neq 0$ y, finalmente, los arcos $(0_1, 0'_1)$ y $(0'_1, 0'_2)$. Por tanto,

$$\begin{aligned}\overline{V} =& \{i_1 : i \in U\} \cup \{i_2 : i \in U\} \cup \{0_1, 0_2, 0'_1, 0'_2\} \\ \overline{A} =& \{(i_1, j_1), (j_1, i_1) : [i, j] \in E, i, j \in U\} \cup \{(0_1, i_1), (i_1, 0'_1) : [0, i] \in E\} \cup \{(i_1, i_2) : i \in V\} \\ & \cup \{(j_1, i_2) : (i, j) \in A, i \in U\} \cup \{(0_1, 0'_1), (0'_1, 0'_2)\}\end{aligned}$$

En la Figura 2.3 se muestra un ejemplo de un grafo y su correspondiente grafo transformado en dos niveles.

En el grafo transformado, los arcos cuyo nodo inicial y final están en el primer nivel se



(a) $G = (V, E \cup A)$ (b) Grafo transformado $\overline{G} = (\overline{V}, \overline{A})$

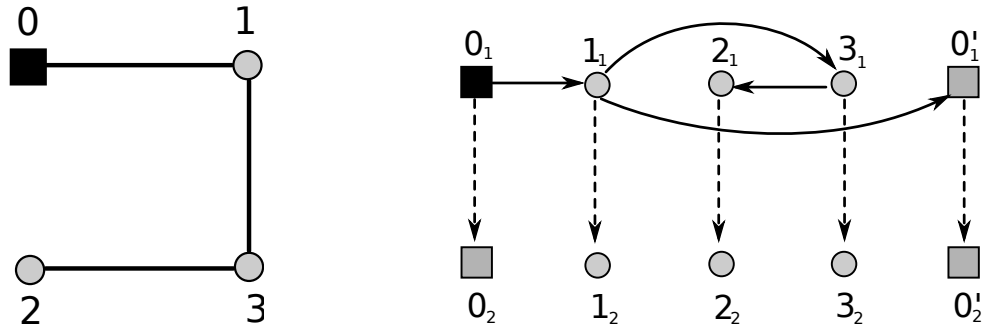
Figura 2.3: Ejemplo de transformación del RSP en STP

identifican con ejes del anillo. El resto de los arcos, es decir, los que tienen el nodo inicial en el primer nivel y el nodo final en el segundo nivel, se identifican con asignaciones de clientes que no están en el anillo a clientes que sí lo están.

Los costes que se asignan a los arcos de $\overline{A}$ se calculan de la siguiente manera. El coste de los arcos $(0_1, 0'_1)$ y (i_1, i_2) para $i_1 \in \overline{V}_1$ es cero. Dado un arco $(i, j) \in A$, el arco asociado $(j_1, i_2) \in \overline{A}$ tiene un coste de d_{ij}. Análogamente, dado un eje $[i, j] \in E$, $i, j \neq 0$, los arcos asociados (i_1, j_1) y (j_1, i_1) en $\overline{A}$ tienen un coste de c_{ij}. Finalmente, para los nodos $i \in V$ vecinos del nodo 0, el coste de los arcos $(0_1, i_1)$ e $(i_1, 0'_1)$ es c_{0i}. En general, para cada arco $a \in \overline{A}$ llamaremos $\overline{c}_a$ al coste asignado a dicho arco. Se tiene, por tanto

$$
\begin{aligned}
\overline{c}_{0_1 0'_1} &= 0 && \\
\overline{c}_{i_1 i_2} &= 0 && \forall i_1 \in \overline{V}_1 \\
\overline{c}_{j_1 i_2} &= d_{ij} && \forall (i,j) \in A \\
\overline{c}_{i_1 j_1} &= \overline{c}_{i_1 j_1} = c_{ij} && \forall [i,j] \in E, i \in U, j \in U \\
\overline{c}_{0_1 i_1} &= \overline{c}_{i_1 0'_1} = c_{0i} && \forall [0,i] \in E
\end{aligned}
$$

Para definir el problema STP asociado al RSP es necesario identificar el nodo raíz r y el conjunto de nodos terminales R. En este caso, el nodo raíz es $r = 0_1$ y R es el conjunto de nodos del nivel 2 del grafo $\overline{G}$. Además, se necesita una restricción adicional que garantice que, en cualquier solución del problema STP, para un nodo en el nivel 1 existe a los sumo un arco que parte de ese nodo y llega a otro nodo en el mismo nivel. Esta restricción adicional evita que puedan darse soluciones como la de la Figura 2.4. En la Figura 2.4b se muestra una solución factible para el problema STP. Sin embargo, en la Figura 2.4a se observa que esta solución no se identifica con ninguna solución del correspondiente RSP. Este problema STP con restricciones adicionales se denomina CSTP (*Constrained Steiner Tree Problem*).

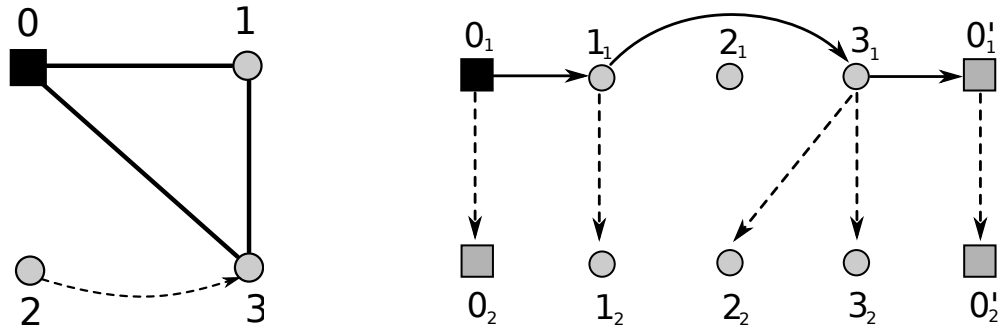


(a) Solución no factible para el RSP

(b) Solución factible para el STP

Figura 2.4: Ejemplo de solución no factible para el RSP pero sí para el STP

Notemos que, por la construcción del grafo transformado $\overline{G}$, la única forma de llegar al nodo $0'_2$ es a través del arco $(0'_1, 0'_2)$, es decir, incluyendo el nodo $0'_1$ en el árbol solución. De esta manera, la construcción del grafo $\overline{G'}$ junto con la restricción adicional garantizan que una solución factible del CSTP induce, sobre los nodos del nivel 1, un camino entre los nodos 0_1 y $0'_1$. La Figura 2.5 ilustra la correspondencia entre una solución del RSP y una del CSTP.



(a) Solución RSP

(b) Solución CTSP

Figura 2.5: Ejemplo de solución transformada

2.4.2. Formulación del CTSP asociado al RSP

Para cada arco $a = (i, j) \in \overline{A}$ definimos la variable binaria $x_a = x_{ij}$ que toma el valor 1 si el arco a pertenece al árbol solución y el valor 0 en otro caso. Con esta definición, el

CSTP sobre $\overline{G}$ se formula como el siguiente problema de programación lineal entera:

$$\min \quad \sum_{a \in \overline{A}} \overline{c}_a x_a \tag{2.10a}$$

sujeto a

$$\sum_{(i,j) \in \overline{A}, j \in \overline{V_1}} x_{ij} \leq 1 \qquad \forall i \in \overline{V_1} \tag{2.10b}$$

$$\sum_{(i,j) \in \overline{A}} x_{ij} \leq 1 \qquad \forall j \in \overline{V} \tag{2.10c}$$

$$\sum_{i \in \overline{V} \setminus S} \sum_{j \in S} x_{ij} \geq 1 \qquad \forall S \subseteq \overline{V} \setminus \{0_1\},\ S \cap \overline{V_2} \neq \emptyset \tag{2.10d}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in \overline{A},\ i,j \in \overline{V_1} \tag{2.10e}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in \overline{A},\ i \in \overline{V_1},\ j \in \overline{V_2} \tag{2.10f}$$

La función objetivo (2.10a) busca la minimización del coste del árbol solución. Las restricciones (2.10b) limitan el número de arcos que parten de un nodo en el nivel 1 a otro en el mismo nivel a uno. Las restricciones (2.10c) limitan a uno el número de arcos que llegan a un nodo en la solución. Las restricciones (2.10d), conocidas como *cortes de Steiner*, aseguran que los nodos terminales ($\overline{V_2} = R$) son alcanzados por la solución. Puede verse que, debido a los costes asignados a los arcos de $\overline{G}$, estas restricciones fuerzan la existencia de una solución óptima cuyo subgrafo inducido sobre los arcos cuyos nodos están en el nivel 1 es un camino.

Simonetti et al. (2011) hacen notar que las restricciones (2.10f) se pueden relajar a $x_{ij} \geq 0$ en esta formulación. Esto se debe a que hay un solución óptima entera del modelo siempre que se imponga que las variables de los arcos internos del nivel 1 sean enteras. Observemos también que esta formulación del CSTP admite soluciones factibles que pueden contener ciclos. Sin embargo, dado que los costes de todas las variables son no negativos y que la función objetivo es de mínimo, se observa que hay una solución óptima de tipo árbol de Steiner.

Simonetti et al. (2011) introducen una serie de restricciones adicionales con el fin de mejorar las cotas del problema lineal relajado. Señalan que por la propia construcción del modelo CSTP, y haciendo uso de las restricciones (2.10d), se obtienen las siguientes restricciones que son satisfechas por, al menos, una solución óptima:

$$\sum_{(i_1,j_1) \in \overline{A}} x_{i_1 j_1} = x_{j_1 j_2} \qquad \forall j_1 \in \overline{V_1} \setminus \{0_1\} \tag{2.11}$$

donde j_2 es la copia correspondiente al nodo j_1 en el nivel 2. Notemos que, para un par de nodos j_1 y j_2, las restricciones (2.11) fuerzan a que el arco (j_1, j_2) esté en la solución siempre que el nodo j_1 esté en la solución.

También son válidas las restricciones (2.12) ya que en un árbol de Steiner óptimo, para cada $j_1 \in \overline{V} \setminus \{0_1, 0_1'\}$, si consideramos sólo los arcos entre nodos del nivel 1, el número de arcos que llegan y que dejan j_1 debe ser el mismo:

$$\sum_{(j_1,i)\in\overline{A},i\in\overline{V_1}} x_{j_1 i} = \sum_{(i,j_1)\in\overline{A},i\in\overline{V_1}} x_{ij_1} \qquad \forall j_1 \in \overline{V_1} \setminus \{0_1, 0_1'\} \tag{2.12}$$

Combinando las restricciones (2.11) y (2.12) para todo $j_1 \in \overline{V_1} \setminus \{0_1, 0_1'\}$ se obtiene un nuevo conjunto de igualdades para el modelo (2.10) más fuertes que las restricciones (2.10b). Estas igualdades son:

$$\sum_{(j_1,i)\in\overline{A},i\in\overline{V_1}} x_{j_1 i} = x_{j_1 j_2} \qquad \forall j_1 \in \overline{V_1} \setminus \{0_1, 0_1'\} \tag{2.13}$$

La estructura y los costes del grafo $\overline{G}$ aseguran que existe una solución óptima con

$$x_{0_1 0_2} = 1 \tag{2.14}$$

y los cortes de Steiner (2.10d) para $S = \{0_2'\}$ fuerzan la igualdad

$$x_{0_1' 0_2'} = 1 \tag{2.15}$$

También son válidas para este problema las desigualdades estudiadas por Bauer (1997). Sea $H \subseteq \overline{V_1}$ un subconjunto de nodos y sean los conjuntos de arcos $\overline{A}(H) = \{(i,j) \in \overline{A} : i,j \in H\}$ y $\delta_1(H) = \{(i,j) \in \overline{A} : i \in Hj \in \overline{V_1} \setminus H\}$. Además, sea $T \subseteq \delta_1(H)$ y $\overline{T} = \{(j,i) \in \overline{A} : (i,j) \in T\}$. Entonces, se tiene la siguiente desigualdad

$$\sum_{a\in\overline{A}(H)} x_a + \sum_{a\in T\cup\overline{T}} x_a \leq \sum_{i\in H} x_{i_1 i_2} + \frac{|T|-1}{2} \tag{2.16}$$

siempre que

(i) $\{i,j\} \cap \{k,l\} = \emptyset$ para $[i,j],[k,l] \in T$ si $[i,j] \neq [k,l]$.

(ii) $|T| \geq 3$ e impar.

Esta desigualdad, adecuadamente adaptada, aparece también en los modelos de Labbé et al. (2004) y Kedad-Sidhoum y Nguyen (2010), y se obtiene de forma análoga.

Así, se puede reforzar el modelo (2.10) reemplazando las restricciones (2.10b) por las igualdades (2.13) y añadiendo las ecuaciones (2.11), (2.14) y (2.15). Este modelo se denomina RCSTM (*Reinforced Constrained Steiner Tree Model*).

2.4.3. Algoritmo

Para resolver el RSP, Simonetti et al. (2011) proponen un algoritmo de ramificación y corte sobre el problema CSTP. Así, el algoritmo se basa en los siguientes puntos:

Punto 1: Conjunto de desigualdades inicial.

Se utiliza el algoritmo heurístico de ascenso dual (*Dual Ascent Heuristic, DAH*) introducido por Wong (1984) para generar un conjunto de desigualdades de Steiner (2.10d) para la formulación RCSTM.

Punto 2: Separación de cortes.

En el algoritmo de ramificación y corte, en cada nodo del árbol de búsqueda, se resuelve la relajación lineal del RCSTM. Si en la solución óptima todos las variables son enteras, se poda el nodo correspondiente por optimalidad. En otro caso la solución es racional y las desigualdades válidas que no se satisfacen han de buscarse resolviendo un problema de separación. Se utiliza el algoritmo propuesto por Fischetti et al. (1997) para calcular las desigualdades no satisfechas de tipo (2.16) y el algoritmo propuesto por Koch y Martin (1998) para separar las desigualdades de Steiner (2.10d).

Punto 3: Cotas superiores.

Para generar una buena solución inicial para el RSP se utiliza un algoritmo heurístico GRASP desarrollado por Resende y Ribeiro (2005).

Punto 4: Estrategia de ramificación.

La estrategia de ramificación considera primero el conjunto de variables de tipo $x_{i_1 i_2}$ y, si todas ellas son enteras, las de tipo $x_{i_1 j_1}$, eligiendo en ambos casos la variable cuyo valor esté más próximo a 0.5 en la solución relajada.

2.5. Algunas variantes del RSP

2.5.1. El problema del ciclo mediano

El problema del ciclo mediano (*Median Cycle Problem, MCP*) es un problema muy similar al RSP y en ocasiones ciertos autores califican como MCP los problemas RSP. Sin embargo, puesto que habitualmente el MCP contiene restricciones adicionales a las del RSP, consideraremos el MCP como un problema distinto.

La diferencia entre el RSP y el MCP es que este último añade una cota superior $d_0 > 0$ que limita el coste total de las asignaciones. Es decir, un anillo R solución factible para el RSP es solución factible para el MCP si y sólo si $d_R \leq d_0$. Utilizando la misma notación que en la Sección 2.2, debe cumplirse la restricción:

$$\sum_{a \in A} d_a y_a \leq d_0 \tag{2.17}$$

Una formulación del MCP como problema de optimización lineal entera es, por tanto, la resultante de añadir la restricción (2.17) al modelo (2.1), es decir:

$$\min \quad \sum_{e \in E} c_e x_e + \sum_{a \in A} d_a y_a \tag{2.18a}$$

sujeto a

$$\sum_{a \in A} d_a y_a \leq d_0 \tag{2.18b}$$

$$x(\delta(0)) = 2 \tag{2.18c}$$

$$x(\delta(i)) = 2y_{ii} \qquad \forall i \in U \tag{2.18d}$$

$$\sum_{j \in V} y_{ij} = 1 \qquad \forall i \in U \tag{2.18e}$$

$$x(\delta(S)) \geq 2 \sum_{j \in S} y_{ij} \qquad \forall S \subset U, \forall i \in S \tag{2.18f}$$

$$x_e,\ y_a \in \{0, 1\} \qquad \forall e \in E, a \in A \tag{2.18g}$$

Puede observarse que el MCP generaliza el RSP, que aparece cuando d_0 es suficientemente grande ($d_0 \geq \sum_{a \in A} d_a$).

En la literatura este problema ha sido abordado de diversas maneras. Pérez et al. (2003) y Renaud et al. (2004) proponen métodos heurísticos para resolver el problema. Labbé et al. (2005) desarrollan la formulación (2.18), que modela el MCP como un problema de optimización lineal entera y proponen una forma de resolución exacta aplicando técnicas de ramificación y corte.

2.5.2. El problema del anillo-estrella bi-objetivo

Liefooghe et al. (2010) plantean modelar el problema RSP como un problema de optimización multiobjetivo.

Un problema de optimización multiobjetivo (MOP) tiene como finalidad optimizar un conjunto de $n \geq 2$ funciones objetivo $f_1, f_2, \ldots, f_n$ simultáneamente. Cada función objetivo puede ser minimizada o maximizada. En este caso supondremos sin pérdida de generalidad que todas las funciones objetivo han de ser minimizadas.

Sea X el conjunto de soluciones factibles del *espacio de decisión*. Para cada solución factible $x \in X$ se considera un *vector objetivo* $y \in \mathbb{R}^n$ definido como $y = f(x) = (f_1(x), \ldots, f_n(x))$. Sea $Y \subseteq \mathbb{R}^n$ el conjunto de vectores objetivo, es decir, $Y = \{y \in \mathbb{R}^n : y = f(x), x \in X\}$.

Dados dos elementos $y, y' \in Y$ diremos que $y \leq y'$ o que y *domina* a y' si $y_i \leq y'_i$ para todo $i = 1, \ldots, n$ e $y_i < y'_i$ para algún $i \in \{1, \ldots, n\}$.

Una solución $x \in X$ se dice que es *eficiente* si no existe ninguna otra que la domine, es decir, si no existe $x' \in X$ tal que $f(x') \leq f(x)$.

Así, un problema MOP puede formularse como:

$$\begin{aligned} \text{“min”} \quad & f(x) = (f_1(x), \ldots, f_n(x)) \\ \text{sujeto a} \quad & \\ & x \in X \end{aligned}$$

El significado de “min” está referido a encontrar el conjunto de soluciones eficientes del problema. Hay que tener en cuenta que al minimizar varias funciones objetivo de forma simultánea lo más probable es que exista una variedad de soluciones factibles y eficientes diferentes. Cada una de estas soluciones obtendrá mejores valores para unas funciones objetivo y peores para otras. En Ehrgott (2005) puede encontrarse un análisis más extenso de los problemas de optimización multiobjetivo así como de diversos métodos de resolución.

Liefooghe et al. (2010) modelan el RSP como un problema bi-objetivo, es decir, un problema multiobjetivo con sólo dos funciones objetivo. Utilizando la misma notación que en la Sección 2.2, las funciones objetivo a minimizar serían:

$$f_1(x, y) = \sum_{e \in E} c_e x_e$$
$$f_2(x, y) = \sum_{a \in A} d_a y_a$$

Es decir, por un lado ha de minimizarse el coste de los ejes del anillo y por otro el coste de las asignaciones correspondientes de los nodos no pertenecientes al anillo.

Para resolver este problema Liefooghe et al. (2010) proponen un algoritmo evolutivo elitista con búsqueda local.

2.5.3. El problema del anillo-estrella con nodos de Steiner

El problema del anillo-estrella con nodos de Steiner (*RSPS*) fue estudiado por primera vez por Lee et al. (1998). El RSPS generaliza el RSP considerando en el problema un

conjunto de nodos adicional, denominados nodos de Steiner, nodos concentradores o nodos de transición. El propósito de estos nodos es el de poder ser utilizados como parte del anillo, asignándoles los nodos clientes que no pertenecen al anillo cuando sea necesario. Los nodos de Steiner que no pertenecen al anillo no es necesario asignarlos a otros nodos que sí pertenezcan, simplemente no se utilizan. En la Figura 2.6 se ilustra una solución factible para el problema RSPS.

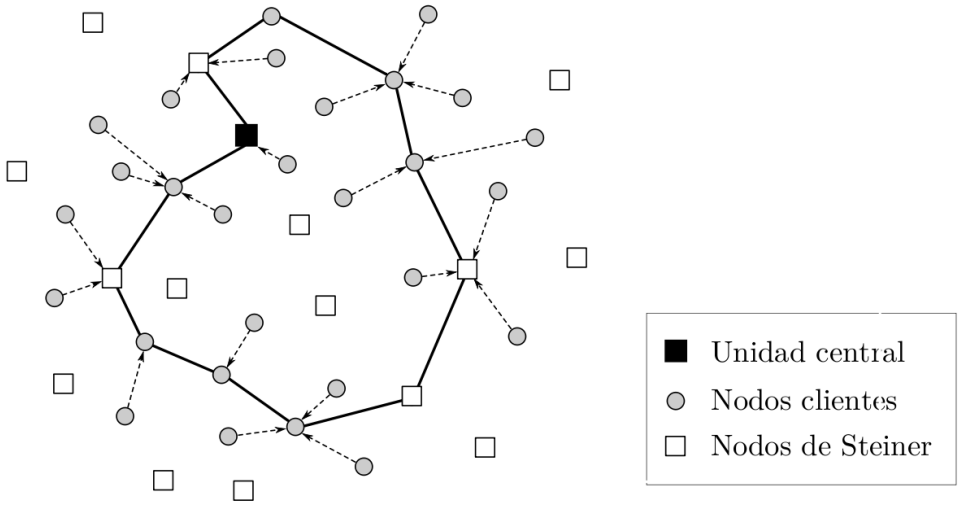


Figura 2.6: Ejemplo de solución del problema RSPS

El RSPS puede ser formulado como un problema de optimización en teoría de grafos ampliando ligeramente la notación utilizada para modelar el RSP. Sea $G = (V, E \cup A)$ un grafo mixto donde $V = \{0\} \cup U \cup W$ es el conjunto de nodos, E es el conjunto de ejes y A el conjunto de arcos. El nodo 0, igual que en el RSP, es el nodo central o nodo raíz por el que pasa necesariamente cualquier anillo solución del problema, U es el conjunto de nodos que representan a los clientes y W es el conjunto de nodos que representan a los nodos de Steiner. El conjunto de ejes E, el conjunto de arcos A y los costes de los elementos de estos conjuntos se definen de forma análoga a los definidos en la Sección 2.1.

Definimos para cada eje $e \in E$ la variable binaria x_e, que toma el valor 1 si el eje e aparece en el anillo y el valor 0 en otro caso, y para cada arco $(i, j) \in A$ la variable binaria y_{ij}, que toma el valor 1 si el nodo cliente i se asigna al nodo j y el valor 0 en otro caso. Además, para cada $j \in W$, sea w_j una variable binaria que toma el valor 1 si el nodo de Steiner j pertenece al anillo y el valor 0 en otro caso.

El problema SRSP puede formularse como:

$$\min \quad \sum_{e \in E} c_e x_e + \sum_{a \in A} d_a y_a \tag{2.19a}$$

sujeto a

$$x(\delta(0)) = 2 \tag{2.19b}$$

$$x(\delta(i)) = 2y_{ii} \qquad \forall i \in V \setminus W \tag{2.19c}$$

$$x(\delta(i)) = 2z_{ii} \qquad \forall i \in W \tag{2.19d}$$

$$\sum_{j \in V} y_{ij} = 1 \qquad \forall i \in U \tag{2.19e}$$

$$x(\delta(S)) \geq 2 \sum_{j \in S} y_{ij} \qquad \forall S \subset V \setminus \{0\}, \forall i \in S \tag{2.19f}$$

$$x_e,\ y_a,\ z_j \in \{0, 1\} \qquad \forall e \in E, a \in A, j \in W \tag{2.19g}$$

Para resolver este problema Lee et al. (1998) proponen un algoritmo exacto utilizando técnicas de ramificación y corte.

2.5.4. El problema del m-anillo-estrella

El problema del m-anillo-estrella con restricciones de capacidad (*CmRSP*) fue propuesto inicialmente por Baldacci et al. (2007) y posteriormente estudiado también por Mauttone et al. (2007) y Naji-Azimi et al. (2010).

Este problema consiste en encontrar un conjunto de m anillos con el menor coste posible de forma que todos ellos pasen por la unidad central (nodo 0) y a través de ciertos nodos clientes y/o nodos de Steiner, asignando cada nodo cliente no visitado a un nodo que sí pertenezca a uno de los anillos. El número de nodos clientes visitados y asignados a cada anillo está limitado por una cota superior Q, la capacidad del anillo, que en problemas de telecomunicaciones hace referencia al grosor del cable utilizado para construir el anillo o al número de fibras que contiene. Este problema generaliza al SRSP, que aparece cuando $m = 1$ y la capacidad de los anillos Q es igual al número de nodos clientes del problema.

En la Figura 2.7 se ilustra una solución factible para el problema CmRSP.

La formulación (2.20) del CmRSP como problema de programación lineal entera, utilizando la misma notación que para el problema RSPS descrito en la Sección 2.5.3, fue

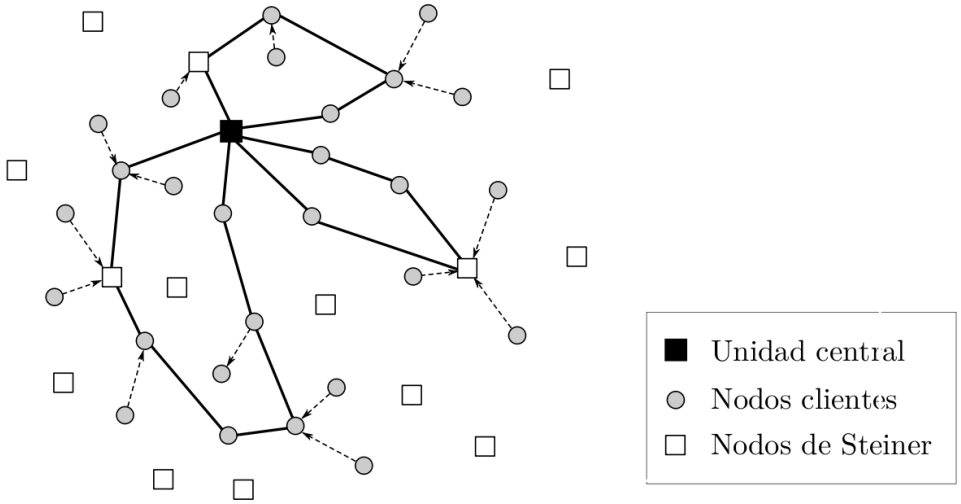


Figura 2.7: Ejemplo de solución del problema CmRSP ($m = 3$, $Q = 15$)

propuesta por Baldacci et al. (2007):

$$\min \quad \sum_{e \in E} c_e x_e + \sum_{a \in A} d_a y_a \tag{2.20a}$$

sujeto a

$$x(\delta(0)) = 2m \tag{2.20b}$$

$$x(\delta(i)) = 2y_{ii} \qquad \forall i \in U \tag{2.20c}$$

$$x(\delta(i)) = 2z_{ii} \qquad \forall i \in W \tag{2.20d}$$

$$\sum_{j \in V} y_{ij} = 1 \qquad \forall i \in U \tag{2.20e}$$

$$x(\delta(S)) \geq \frac{2}{Q} \sum_{i \in U} \sum_{j \in S} y_{ij} \qquad \forall S \subset V \setminus \{0\}, S \neq \emptyset \tag{2.20f}$$

$$x_e,\ y_a,\ z_j \in \{0, 1\} \qquad \forall e \in E, a \in A, j \in W \tag{2.20g}$$

Para resolver este problema se han propuesto diversos algoritmos, tanto exactos como heurísticos. Baldacci et al. (2007) proponen un algoritmo de resolución utilizando técnicas de ramificación y corte sobre la formulación (2.20). Mauttone et al. (2007) proponen un algoritmo metaheurístico que combina técnicas GRASP y de búsqueda tabú. Naji-Azimi et al. (2010) proponen un algoritmo heurístico basado en la aplicación iterativa de varios métodos de búsqueda local.

2.5.5. El problema del anillo-estrella multidepósito

El problema del anillo-estrella multidepósito (*MDRSP*) se define sobre una red que consta de varias unidades centrales. Desde cada una de ellas puede partir uno o varios

anillos, dependiendo de las restricciones que tenga asociada cada una de ellas. El objetivo del MDRSP es diseñar un conjunto de anillos, partiendo cada uno desde una de las unidades centrales, de forma que cada uno de los clientes esté asociado a un único anillo.

Como ocurría para el problema CmRSP, la capacidad de cada anillo está limitada por una cota superior $Q \in \mathbb{N}$. Además, para cada nodo central d existe una cota superior $m_d \in \mathbb{N}$ que limita el número de anillos que pueden partir de dicho nodo central. Se considera también que, además del conjunto de clientes, el grafo consta de ciertos nodos de transición o nodos de Steiner. Cada uno de los nodos de Steiner puede ser visitado a lo sumo por un anillo.

En la Figura 2.8 se ilustra una solución factible para el problema del anillo-estrella multidepósito.

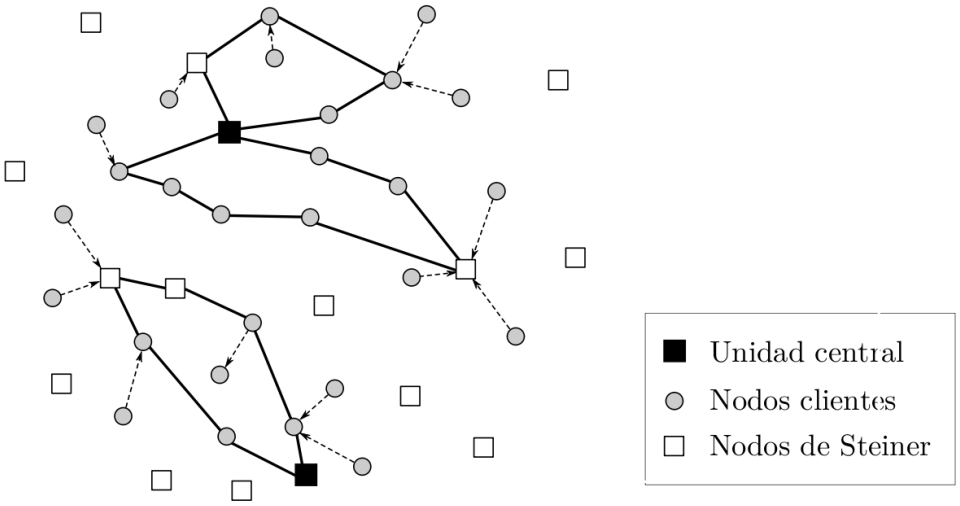


Figura 2.8: Solución factible del problema MDRSP

Este problema fue inicialmente propuesto por Baldacci y Dell'Amico (2010), quienes presentaron una formulación como problema de optimización lineal entera y propusieron diversos procedimientos heurísticos para su resolución.

Notemos que el MDRSP generaliza el problema RSP, que aparece cuando el conjunto de nodos de Steiner es vacío y existe una única unidad central, el 0, de la que puede partir un único anillo ($m_0 = 1$) cuya capacidad límite es, al menos, $Q = n$. Por tanto, el problema MDRSP es un problema NP-duro.

3

Una transformación del RSP en el problema del viajante generalizado

3.1. Introducción

En esta sección se propone una reformulación del RSP como un problema del viajante generalizado (*Generalized Traveling Salesman Problem, GTSP*) sobre una red construida apropiadamente. El GTSP se plantea sobre un grafo cuyos nodos están distribuidos en varios subconjuntos y resuelve el problema de encontrar un ciclo con el menor coste posible que pasa por todos los subconjuntos dados. Por tanto, el ciclo, además de establecer el orden en el que se visita cada subconjunto, determina también qué nodo o nodos se visita en cada uno de ellos y en qué orden. Si cada subconjunto está formado por uno de los nodos del grafo sobre el que se plantea el GTSP, el problema queda reducido al problema del viajante clásico.

El GTSP fue introducido por Srivastava et al. (1969) que presentaron la versión simétrica del problema y propusieron un algoritmo de resolución mediante programación dinámica. Simultáneamente, Henry-Labordere (1969) presentó la versión asimétrica del problema. Ambas versiones difieren en el tipo de grafo sobre el que se plantea el problema, ya sea no dirigido o dirigido respectivamente. En la tesis doctoral de Noon (1988) aparece un estudio detallado de este problema, de sus variantes y de algunos de los algoritmos de resolución existentes, tanto exactos como heurísticos. Se define, además, la forma canónica del GTSP en la que se considera que los subconjuntos en los que se divide el conjunto total de nodos son disjuntos, que no hay arcos internos en dichos subconjuntos, es decir arcos que conectan dos nodos del mismo subconjunto, y que un ciclo solución del problema visita cada subconjunto exactamente una vez. Además, si un problema no cumple alguna de estas condiciones, Noon (1988) propone las transformaciones necesarias para convertirlo a la forma canónica en tiempo polinomial.

En este trabajo nos centraremos en el GTSP asimétrico, con subconjuntos no necesariamente disjuntos, con la posible existencia de arcos internos y en el que un ciclo debe visitar cada subconjunto una única vez.

3.2. Construcción del grafo del problema GTSP transformado

Sea $G = (V, E \cup A)$ el grafo del RSP tal y como aparece descrito en la Sección 2.1. Se define el conjunto de nodos $\hat{V} = \{i_j : (i,j) \in A, i \neq 0, i \neq j\}$. Consideramos el grafo dirigido $G' = (V', A')$, donde

$$V' = V \cup \hat{V}$$

$$A' = A_1 \cup A_2 \text{ siendo } A_1 = \{(i,j),(j,i) : [i,j] \in E\} \text{ y } A_2 = \{(i_j,j),(j,i_j) : i_j \in \hat{V}\}$$

Los arcos $a \in A'$ tienen asociado el coste c'_a que se asigna de la siguiente manera. A los arcos del conjunto A_1, que provienen de los ejes de E, se les asigna el coste c_{ij}, a los arcos del conjunto A_2 que son de la forma $(j, i_j) \in A'$ se les asigna coste 0 y a los que son de la forma $(i_j, j) \in A'$ se les asigna coste d_{ij}. Es decir,

$$c'_{ij} = c_{ij},\ c'_{ji} = c_{ij} \text{ para todo eje } [i,j] \in E$$

$$c'_{i_j j} = d_{ij},\ c'_{j i_j} = 0 \text{ para todo arco } (i,j) \in A$$

De esta manera, puede pensarse que los nodos del grafo G' están estructurados en dos niveles, el nivel 0 que incluye los nodos originales, es decir el conjunto V y el nivel 1 en el que están los nuevos nodos creados $\hat{V}$. Por la construcción del grafo G', los nodos del nivel 1 no están conectados entre sí. De hecho, cada nodo i_j del nivel 1 sólo está conectado al resto de nodos mediante dos arcos, uno de entrada y uno de salida, que lo conectan al nodo j del nivel 0.

Si llamamos n al número de nodos del grafo G, m_E al número de ejes y m_A al número de arcos propios el tamaño del grafo transformado G' es de $n + m_A$ nodos y $2m_E + 2m_A$ arcos.

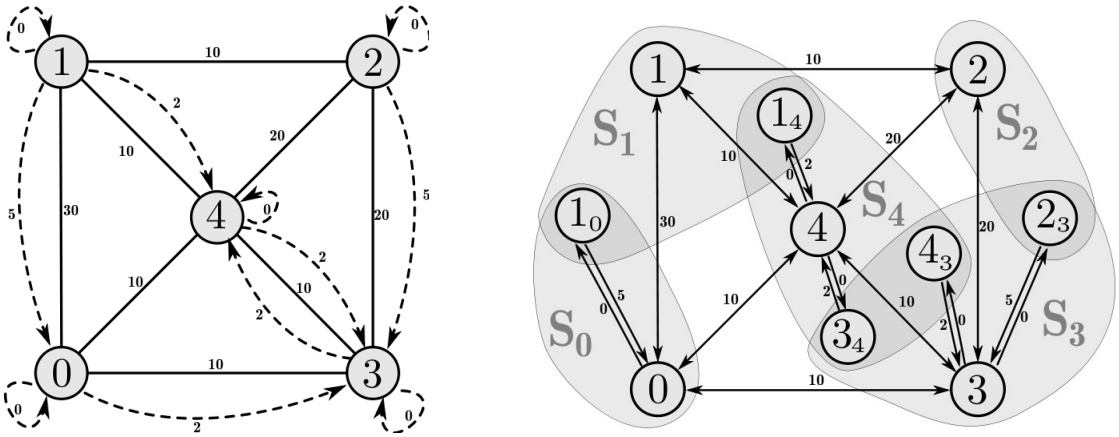
La transformación que se propone en este trabajo establece una equivalencia entre un ciclo C sobre el grafo G' y un anillo R sobre el grafo G, de manera que C pasa por el nodo i del nivel 0 si y sólo si el nodo i pertenece al anillo R, y C pasa por el nodo i_j del nivel 1 si y sólo si el nodo i no está en R y está asignado al nodo j que sí está en R. Puesto que

un anillo R del RSP para que sea factible ha de contener el nodo 0, un ciclo equivalente sobre el grafo G' deberá contener también el nodo 0, que se encuentra en el nivel 0.

En el grafo G', para cada nodo $i \in V$, definimos el subconjunto de nodos S_i como

$$S_i = \{i\} \cup \{i_j : i_j \in \hat{V}\} \cup \{j_i : j_i \in \hat{V}\}$$

El GTSP sobre G' consiste en encontrar un ciclo C, no necesariamente simple, que visita exactamente una vez cada uno de los conjuntos S_i. Notemos que los conjuntos S_i no son disjuntos y que existen arcos en A' cuyos nodos inicial y final están contenidos dentro de un mismo S_i, por lo que un ciclo que visita una única vez S_i puede contener varios nodos de S_i. En otras palabras, un conjunto S_i es visitado una única vez si el ciclo correspondiente entra, y por tanto sale, una única vez en el conjunto. En la Figura 3.1, a la izquierda se muestra un grafo G y a la derecha su correspondiente grafo transformado G'.

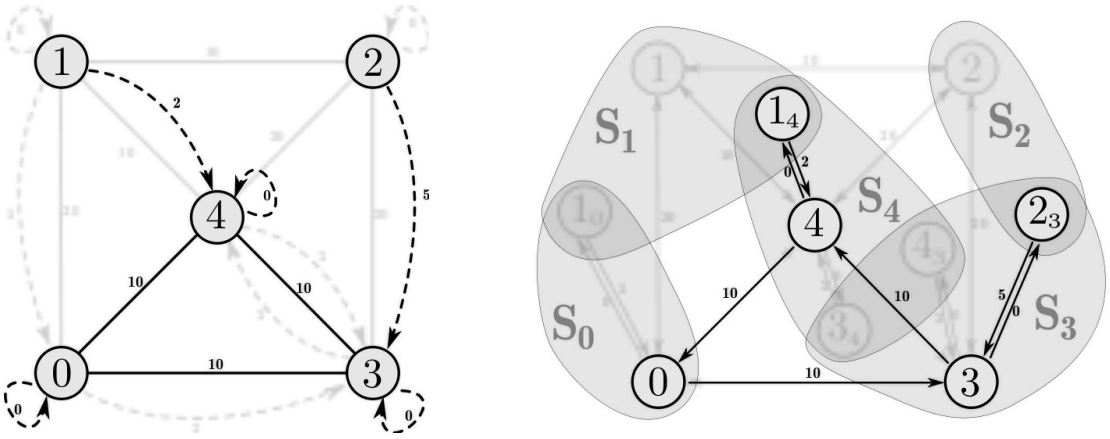


(a) Grafo para el RSP (b) Grafo para el problema transformado GTSP

Figura 3.1: Ejemplo de transformación a un problema GTSP

Por construcción, un ciclo C solución del $GTSP$ pasa necesariamente por el nodo $0 \in V'$. Además, para cada nodo $i \in V$, el ciclo C contiene o bien el nodo i o bien un único nodo del tipo $i_j \in \hat{V}$, ya que todos estos nodos pertenecen a S_i y no existen conexiones entre ellos.

Los nodos de tipo $i_j \in \hat{V}$ pertenecen exactamente a dos subconjuntos, el S_i y el S_j, y no pueden aparecer repetidos dentro de un mismo ciclo C solución factible del GTSP, ya que i_j sólo esta conectado con el nodo j que pertenece sólo a S_j. Por el contrario, los nodos de tipo $i \in V$ sí pueden aparecer repetidos dentro de un mismo ciclo C, hecho que se ilustra en la Figura 3.2 en la que aparece a la izquierda una solución factible del RSP y a la derecha la solución correspondiente del GTSP sobre el grafo transformado. En cualquier caso, ninguno de los arcos $a \in A'$ puede aparecer repetidos dentro de un mismo ciclo.



(a) Solución factible para el RSP (b) Solución factible para el GTSP

Figura 3.2: Transformación entre una solución del RSP y una del GTSP

Notemos que, de esta forma, a cada solución factible R del problema RSP se le puede asociar una solución factible C del GTSP con el mismo valor de la función objetivo sin más que considerar el ciclo formado por los arcos en el nivel 0 correspondientes a los ejes de R, tomados en uno de los dos sentidos posibles, y los arcos del nivel 0 al nivel 1 y viceversa correspondientes a las asignaciones del problema RSP. Análogamente, a cada solución factible C del GTSP se le puede asociar una solución factible R del RSP sin más que considerar el anillo formado por los ejes correspondientes a los arcos del ciclo C en el nivel 0.

Mediante esta transformación, el problema RSP se puede resolver utilizando las técnicas y algoritmos desarrollados en la literatura para el problema GTSP. Así mismo, se han desarrollado transformaciones, como la de Behzad y Modarres (2002), que permiten transformar el problema GTSP en el problema del viajante clásico (*TSP*) y dar la posibilidad, por tanto, de utilizar las técnicas desarrolladas para el TSP para resolver el RSP.

3.3. Formulación matemática del problema GTSP transformado

Utilizaremos la formulación presentada por Noon (1988) para el problema GTSP general para formular el problema transformado. Puesto que el grafo G' del problema transformado consta de nodos de tipo $i \in V$ y de tipo $i_j \in \hat{V}$, en adelante cuando se utilice la notación de un solo índice (i, j, ...) nos estaremos refiriendo a nodos de V y cuando se utilice la notación índice-subíndice (i_j, j_k, ...) nos estaremos refiriendo a nodos de $\hat{V}$.

Para cada arco $a \in A'$, sea z_a la variable binaria que toma el valor 1 si el arco a

pertenece al ciclo y el valor 0 en otro caso.

Llamaremos *subciclo* del GTSP a cualquier ciclo no degenerado contenido en la solución $\{z_a\}_{a\in A'}$ que no pasa por todos los subconjuntos de nodos S_i. Dado un subciclo T, denotaremos $\mathfrak{T}$ al conjunto de todos los posibles subciclos y $\delta(T)$ al conjunto de arcos de A' cuyo nodo inicial sea un nodo de T y cuyo nodo final no lo sea.

Teniendo en cuenta las características del grafo transformado G', el problema GTSP general puede formularse como:

$$\min \quad \sum_{a\in A'} c'_a z_a \tag{3.1a}$$

sujeto a

$$\sum_{(i,j)\in A'} z_{ij} + \sum_{(i_j,j)\in A'} z_{i_j j} = 1 \qquad \forall i \in V \tag{3.1b}$$

$$\sum_{(j,i)\in A'} z_{ji} + \sum_{(j,i_j)\in A'} z_{ji_j} = 1 \qquad \forall i \in V \tag{3.1c}$$

$$\sum_{(j,i_j)\in A'} z_{ji_j} - \sum_{(i_j,j)\in A'} z_{i_j j} = 0 \qquad \forall i_j \in \hat{V} \tag{3.1d}$$

$$\sum_{(j,i)\in A'} z_{ji} + \sum_{(j_i,i)\in A'} z_{j_i i} - \left(\sum_{(i,j)\in A'} z_{ij} + \sum_{(i,j_i)\in A'} z_{ij_i} \right) = 0 \qquad \forall i \in V \tag{3.1e}$$

$$\sum_{a\in\delta(T)} z_a \geq 1 \qquad \forall T \in \mathfrak{T} \tag{3.1f}$$

$$x_a \in \{0,1\} \qquad \forall a \in A' \tag{3.1g}$$

La restricción (3.1b) establece que para cada subconjunto S_i existe un único arco en el ciclo solución que sale del conjunto y, análogamente, la restricción (3.1c) establece que existe un único arco de entrada al conjunto S_i en el ciclo solución. Las restricciones (3.1d) y (3.1e) aseguran que para cualquier nodo, de $\hat{V}$ y V respectivamente, el número de arcos entrantes del ciclo solución es igual al número de arcos salientes. Por último, la restricción (3.1f) permite prevenir que sean factibles soluciones con ciclos disjuntos o no conectados.

4

Un algoritmo evolutivo para el RSP

Para la resolución del problema RSP se propone en este trabajo un algoritmo genético. Los algoritmos genéticos son técnicas de búsqueda estocástica inspiradas por la evolución biológica que sucede en la naturaleza. Fueron introducidos por Holland (1975) y han sido utilizados con éxito para proporcionar buenas soluciones a un gran número de problemas complejos. Cada solución del problema considerado se codifica como una cadena de símbolos que recibe el nombre de cromosoma o individuo. Cada posición de la cadena es un gen y el valor del símbolo que ocupa esa posición es un alelo. El algoritmo precisa también una función 'fitness' que evalúa la calidad de un individuo. Para comenzar el algoritmo, se genera una población de individuos y se evalúa su calidad. En las sucesivas iteraciones, que se corresponden con las generaciones, el algoritmo mantiene una población de individuos. A partir de la población actual, se generan nuevos individuos (hijos) por la aplicación de operaciones de recombinación (cruce) o mutación. Tras evaluar la calidad de los nuevos individuos, se seleccionan, mediante algún criterio, algunos de los padres y de los hijos para formar la nueva población. Los distintos algoritmos genéticos difieren en la forma de codificación de las soluciones, los operadores que aplican, la función fitness y el criterio de selección de la nueva población.

Para el problema del anillo estrella se han propuesto en la literatura algunos algoritmos de tipo genético. Renaud et al. (2004) proponen un algoritmo genético que no usa el operador mutación. Liefooghe et al. (2010) proponen un algoritmo genético adaptado al problema del anillo-estrella bi-objetivo.

A continuación se describen las características del algoritmo propuesto.

4.1. Codificación de los individuos

Sea el grafo $G = (V, E \cup A)$ y $n = |V|$. Sea una solución factible cuyo anillo asociado es R. Inicialmente, se codificó cada individuo asignándole el conjunto de nodos por los que pasa el anillo R al que representa, en el orden en el que se recorren. Puesto que todos los anillos deben contener al nodo 0 y no deben contener nodos repetidos, la codificación sería de la forma:

$$[0 = r_0, r_1, r_2, \ldots, r_m]$$

con $r_i \in V \setminus \{0\}$ para todo $1 \leq i \leq m$ y $r_i \neq r_j$ para $i \neq j$. Sin embargo, al aplicar un operador de cruce, esta codificación presenta ciertas dificultades para mantener la factibilidad de las soluciones. Por ello, se decidió finalmente utilizar la codificación propuesta por Bean (1994). Esta codificación ha sido utilizada para desarrollar diversos algoritmos evolutivos para la resolución de otros problemas de optimización en los que también es necesario establecer un orden sobre un cierto subconjunto de nodos, como son el problema del viajante, los problemas de planificación de tareas, los problemas de rutas de vehículos, etc.

La codificación de Bean (1994) se basa en un mecanismo de asignación de códigos aleatorios. Cada nodo r_i de un anillo $[0 = r_0, r_1, r_2, \ldots, r_m]$ tiene asignado un cierto código $x_i \in [0, 1]$, de forma que para cualquier par de nodos r_i y r_j del anillo, si r_i precede a r_j entonces se tiene que $x_i \leq x_j$. Por tanto, conocido el conjunto de nodos que pertenecen al anillo y el código que tiene asignado cada uno de ellos, basta con ordenar los códigos en orden creciente para conocer el orden en el que han de recorrerse los nodos del anillo. Si dos o más nodos de un anillo tienen asignado el mismo código, el orden de dichos nodos se decide arbitrariamente. Como estamos interesados en soluciones factibles, en las que el nodo 0 debe pertenecer al anillo correspondiente, asignaremos el código 0.00 al nodo 0 y para que el 0 sea el primer nodo de la lista de nodos del anillo supondremos, sin pérdida de generalidad, que el código del resto de nodos es mayor que 0.

Por ejemplo, en un grafo con $n = 8$ nodos, para codificar el anillo $[0, 3, 7, 4]$, se generan tres números aleatorios y se ordenan en orden creciente. Supongamos que estos números son 0.27, 0.35 y 0.54. Este anillo puede representarse entonces como se indica en la Figura 4.1, donde el signo '-' indica que el nodo correspondiente no está en el anillo y está asignado al nodo del anillo más cercano.

Como se verá más adelante, esta codificación de los individuos no plantea problemas de factibilidad al utilizar los operadores de cruce y mutación.

La función *fitness* que permite evaluar la calidad de las soluciones, es decir, de los

Nodo	0	1	2	3	4	5	6	7
Código	0.00	-	-	0.27	0.54	-	-	0.35

Figura 4.1: Representación mediante códigos de una solución del problema RSP

individuos de la población, asigna a cada individuo el coste del anillo al que representa, es decir, si un individuo representa el anillo $R = (V_R, E_R)$ el fitness de dicho individuo es

$$\sum_{e \in E_R} c_e + \sum_{i \in V \setminus V_R} \min_{j \in V_R} d_{ij}$$

4.2. Población inicial

El tamaño de la población, $N \in \mathbb{N}$, forma parte de los parámetros de entrada del algoritmo y permanece constante generación tras generación. Para crear un individuo de la población inicial se genera un valor s de una variable aleatoria *Uniforme* en $(0, 1)$. A continuación, se selecciona cada uno de los nodos del grafo (excepto el 0) con probabilidad s, se generan tantos códigos aleatorios como nodos han sido seleccionados y se asigna a cada nodo seleccionado el código generado. El individuo se construye entonces con el nodo 0 con el código 0.00, los nodos seleccionados con sus respectivos códigos y los nodos no seleccionados sin código.

Inicialmente se probaron también otras formas de generar la población inicial que fueron descartadas porque proporcionaban peores resultados experimentales. Entre estas formas se probó, por ejemplo, fijar la probabilidad s a priori, pero se ha observado que se obtiene entonces una población inicial sesgada que hace que el algoritmo converja inicialmente de forma más lenta. En particular, el criterio de tomar una probabilidad fija $s = 0.5$ fue utilizado por Liefooghe et al. (2010) al resolver el problema del anillo-estrella bi-objetivo mediante algoritmos evolutivos.

4.3. Operador de cruce o recombinación

En el algoritmo propuesto se utiliza un operador de tipo *cruce en un punto*. Dados dos individuos de la población, denominados padres, que están codificados como se ha indicado en el Sección 4.1, el operador de cruce en un punto se define de forma natural. En primer lugar, se selecciona aleatoriamente una posición para realizar el cruce, que divide cada uno de los dos individuos padres en dos partes. Para obtener los dos hijos resultantes, se combina la primera parte del primer padre con la segunda parte del segundo padre

(primer hijo) y la primera parte del segundo padre con la segunda parte del primer padre (segundo hijo). En la Figura 4.2 se muestran los padres y los hijos resultantes cuando la posición de cruce se encuentra entre los genes 2 y 3. Así, si los anillos padres son los anillos $[0, 4, 2, 3, 1]$ y $[0, 3, 7, 4]$, se obtienen como hijos los anillos $[0, 2, 3, 7, 4, 1]$ y $[0, 4, 3]$.

Padre 1	Nodo	0	1	2	3	4	5	6	7
[0, 4, 2, 3, 1]	Código	0.00	0.71	0.27	0.33	0.01	-	-	-
Padre 2	Nodo	0	1	2	3	4	5	6	7
[0, 3, 7, 4]	Código	0.00	-	-	0.27	0.54	-	-	0.35
Hijo 1	Nodo	0	1	2	3	4	5	6	7
[0, 2, 3, 7, 4, 1]	Código	0.00	0.71	0.27	0.27	0.54	-	-	0.35
Hijo 2	Nodo	0	1	2	3	4	5	6	7
[0, 4, 3]	Código	0.00	-	-	0.33	0.01	-	-	-

Figura 4.2: Ejemplo de cruce en un punto

Según el trabajo experimental de Renaud et al. (2004), se obtienen mejores resultados si solamente se considera uno de los dos hijos que se generan al cruzar dos padres. Partiendo de este hecho, consideraremos que al cruzar dos padres, el Padre 1 y el Padre 2, sólo se genera un Hijo formado a partir de la primera parte del Padre 1 y la segunda parte del Padre 2, habiendo determinado dichas partes mediante un mismo punto de cruce.

El punto de cruce se selecciona de forma aleatoria eligiendo un número k al azar entre 1 y $n - 2$. El Hijo resultante retiene las características de los nodos menores o iguales que k del Padre 1 y los nodos mayores que k del Padre 2. Notemos que si fuese $k = 0$ o $k = n - 1$ el Hijo resultante del cruce sería una copia del Padre 1 o del Padre 2, que no resulta de interés ya que no añade diversidad a la población.

Los padres se eligen de forma aleatoria entre los individuos de la población. Se han probado también otras opciones basadas en el valor del fitness, como la selección por torneo, pero han sido descartadas porque se obtuvieron peores resultados experimentales, ya que se producía una convergencia prematura.

El número de padres seleccionados está determinado por el valor de un parámetro $\beta \in [0, 1]$, que forma parte de los parámetros de entrada. Se generan $\lfloor N(1 - \beta) \rfloor$ hijos en cada iteración y, por tanto, es necesario seleccionar aleatoriamente $2\lfloor N(1 - \beta) \rfloor$ padres entre los individuos de la población para generarlos.

4.4. Operador de mutación

Para introducir aleatoriedad en la población se introduce el operador de mutación. La probabilidad de mutación, $\alpha \in [0, 1]$, forma parte de los parámetros de entrada del algoritmo y generalmente toma valores pequeños, del orden de $\frac{1}{n}$.

El operador de mutación selecciona, con probabilidad α, cada uno de los nodos del grafo, salvo el 0, y muta el correspondiente anillo $[0 = r_0, r_1, r_2, \ldots, r_m]$ con códigos $[0 = x_0, x_1, x_2, \ldots, x_m]$. Para cada nodo i seleccionado para mutar se procede de la siguiente manera:

- Si el nodo i pertenece al anillo se elimina de éste junto con su código y se asigna al nodo más cercano que pertenezca al anillo.

- Si el nodo i no pertenece al anillo se añade a éste en la posición que aumenta menos el coste del ciclo. Sólo se tiene en cuenta el coste del ciclo y no los costes de asignación, ya que recalcular los costes de asignación implica más tiempo de ejecución del deseable para un proceso de mutación. Si la posición que aumenta menos el coste del ciclo está entre los nodos r_j y r_{j+1} del anillo, se le asigna a i el código resultante de hacer el promedio entre los códigos de r_j y r_{j+1}, es decir,

  $$x_i = \frac{x_{r_j} + x_{r_{j+1}}}{2}$$

  Si la posición que menos aumenta el coste del ciclo está entre los nodos r_m y $r_0 = 0$, se le asigna a i el código

  $$x_i = \frac{x_{r_m} + 1}{2}$$

  que añade i al final del ciclo.

En cualquier caso, el coste del individuo mutado es distinto del individuo original, ya que cambia el conjunto de ejes del anillo y las asignaciones de los ejes que quedan fuera del anillo.

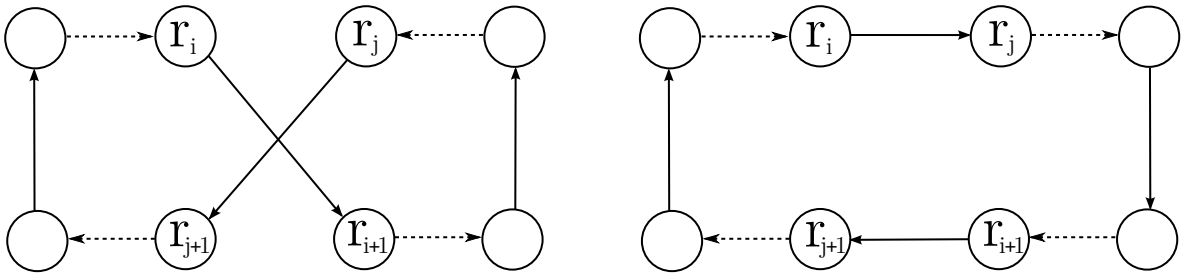
4.5. Operador de búsqueda local (2-opt)

El método 2-opt es un algoritmo de búsqueda local propuesto por primera vez por Croes (1958) para el problema del viajante. Su objetivo es mejorar la calidad de las soluciones intercambiando la posición de los nodos. Para ello, se buscan dos ejes en el anillo que se crucen y se reordenan los nodos de forma que el cruce desaparezca. Diremos

que dos ejes pertenecientes al anillo, $[r_i, r_{i+1}]$ y $[r_j, r_{j+1}]$ con $j > i + 1$, se cruzan si

$$c_{r_i r_{i+1}} + c_{r_j r_{j+1}} > c_{r_i r_j} + c_{r_{i+1} r_{j+1}}$$

Si se cruzan, basta con considerar las conexiones $[r_i, r_j]$ y $[r_{i+1}, r_{j+1}]$ e invertir el orden en el que se recorren los nodos entre r_{i+1} y r_j, ambos inclusive, para que desaparezca ese cruce. Hay que tener en cuenta, no obstante, que esta reordenación puede dar lugar a nuevos cruces. En la Figura 4.3 se muestra gráficamente la aplicación del método 2-opt. Para invertir el orden de los nodos entre r_{i+1} y r_j utilizando la codificación de códigos basta con intercambiar el valor de x_{i+1+k} por el de x_{j-k} para $0 \leq k < \frac{j-i-1}{2}$. Notemos que, tras la modificación, no es necesario recalcular las asignaciones de los nodos externos al anillo, ya que el conjunto de nodos del anillo no varía. Solamente varía el conjunto de ejes del anillo.



(a) Anillo con un cruce (b) Anillo tras aplicar el método 2-opt

Figura 4.3: Ejemplo de aplicación del método 2-opt

En el algoritmo propuesto, el operador de búsqueda local recorre todos los pares de nodos (i, j) con $i < j$ hasta encontrar algún cruce. Si no se encuentran cruces, el individuo no se modifica. En otro caso, se aplica la reordenación de nodos 2-opt sobre el cruce encontrado y se repite el proceso mientras sigan encontrándose cruces hasta un máximo de C veces, siendo $C \in \mathbb{N}$ un parámetro de entrada del algoritmo.

4.6. Selección de supervivientes

Para la selección de los individuos que pasan de una generación a la siguiente se ha elegido el criterio elitista. Es decir, se seleccionan los N mejores individuos (los de menor fitness) entre los N individuos de la generación actual correspondiente y los $\lfloor N(1 - \beta) \rfloor$ hijos generados con el operador de cruce y transformados con el operador de mutación y búsqueda local. Notemos que el parámetro $\beta \in [0, 1]$ indica el porcentaje de los mejores individuos de la generación actual que pasan directamente a la siguiente generación.

4.7. Criterio de parada

Entre los diversos criterios de parada posibles, en el algoritmo propuesto se han considerado dos. El primero de ellos limita el número de iteraciones del algoritmo a un máximo de $M \in \mathbb{N}$ iteraciones. El segundo criterio permite asegurar cierta diversidad en la población. Para ello, en cada iteración se comprueba si la diferencia entre el fitness del mejor anillo encontrado hasta el momento y el fitness medio de la población es menor que un cierto $\varepsilon \geq 0$, y en caso afirmativo el algoritmo se detiene. Notemos que el hecho de que el fitness del mejor anillo sea cercano al fitness medio de la población no implica directamente que todos los individuos de la población sean similares. Sin embargo, experimentalmente se ha comprobado que es un buen criterio de parada ya que se activa tras varias iteraciones en las que no ha habido mejora.

4.8. Descripción del algoritmo

El algoritmo evolutivo que se propone en este trabajo hace uso de varios parámetros de entrada que se resumen en la Tabla 4.1.

$N \in \mathbb{N}$	Tamaño de la población
$M \in \mathbb{N}$	Número máximo de iteraciones
$\varepsilon \in [0, \infty]$	Diferencia entre el mejor individuo y la media de la población
$\alpha \in [0, 1]$	Probabilidad de mutación en cada gen de cada individuo
$\beta \in [0, 1]$	Proporción de individuos de una generación que pasan a la siguiente
$C \in \mathbb{N}$	Número máximo de veces que se repite el proceso de búsqueda local

Tabla 4.1: Parámetros de entrada del algoritmo evolutivo

Fijados estos parámetros el algoritmo evolutivo realiza las siguientes etapas o pasos:

Paso 1: Inicialización.

Se inicializa el contador de iteraciones $t = 0$ y se genera una población inicial aleatoria con N individuos. A continuación, se aplica a cada individuo de la población inicial el método de búsqueda local y se calcula su fitness.

Paso 2: Selección de padres y recombinación.

Si $t > M$ ó la diferencia entre el menor fitness calculado y el fitness medio de la población actual es menor que ε el algoritmo termina. En otro caso, se seleccionan aleatoriamente $2\lfloor N(1-\beta) \rfloor$ padres entre los individuos de la población y se cruzan por parejas, formando $\lfloor N(1-\beta) \rfloor$ hijos.

Paso 3: Mutación.

Se aplica el operador de mutación, con probabilidad α, sobre cada uno de los hijos.

Paso 4: Búsqueda local.

Se aplica el operador de búsqueda local con parámetro C sobre cada uno de los hijos.

Paso 5: Evaluación y selección de supervivientes.

Se calcula el fitness de los hijos y se reemplaza la población actual por los N individuos, entre padres e hijos, cuyo fitness sea menor. Se actualiza el contador de iteraciones, $t = t + 1$, y se va al Paso 2.

4.9. Resultados experimentales

Se ha implementado en C++ el algoritmo que aquí se presenta siguiendo el esquema detallado en la Sección 4.8. El código completo de la implementación se adjunta en el Anexo I. Se han realizado pruebas sobre diversos tipos de grafo, generados con el mismo código, para comprobar la efectividad del algoritmo.

Los grafos utilizados para realizar las pruebas son grafos completos de tipo euclídeo con $n = 50$, $n = 100$ y $n = 200$ nodos. El conjunto de nodos se ha escogido aleatoriamente en el subconjunto $[0, 1] \times [0, 1] \subset \mathbb{R}^2$, tomando como nodo central el primer nodo generado. Sean V_{50}, V_{100} y V_{200} los conjuntos de nodos correspondientes.

Para cualquier par de nodos $i = (x_1, x_2) \in V_n$ y $j = (y_1, y_2) \in V_n$ existe tanto el arco (i, j) como el eje $[i, j]$. Los costes de dicho arcos y ejes se definen proporcionales a la distancia euclídea entre los puntos (x_1, x_2) y (y_1, y_2). En concreto, se ha tomado $d_{ij} = \sqrt{\sum_{k=1}^{2} (x_k - y_k)^2}$ y $c_{ij} = 5d_{ij}$.

El tamaño de la población que se utilizará en el algoritmo evolutivo, N, se ha elegido proporcional al número de nodos del grafo, siendo las proporciones utilizadas del 20 % y del 50 %.

La probabilidad de mutación se ha elegido como $\alpha = \frac{3}{n}$, el porcentaje de individuos que pasa directamente de una generación a la siguiente como $\beta = 1\,\%$ y el número de veces que se aplica como máximo el método 2-opt en el proceso de búsqueda local como $C = \lceil 0.01n \rceil$. Los tres parámetros se han elegido respondiendo a distintos resultados experimentales previamente realizados.

Los parámetros del criterio de parada son los siguientes. El número máximo de iteraciones se ha tomado suficientemente grande como para continuar con el algoritmo mientras haya diversidad en la población, por ello $M = 10^4$, y el valor $\varepsilon = 10^{-10}$ de forma que el algoritmo se detenga cuando todos los individuos de la población tengan prácticamente el mismo fitness.

Ejecutando treinta veces el algoritmo para cada una de las configuraciones posibles se obtiene la Tabla 4.2 de resultados, que contiene los siguientes datos:

n: Número de nodos del grafo generado.
N: Tamaño de la población del algoritmo evolutivo.
Tiempo final: Tiempo medio de cálculo hasta que el algoritmo se detiene (segundos).
Tiempo anillo: Tiempo medio de cálculo hasta encontrar el mejor anillo (segundos).
Iteración final: Iteración, en media, en la que el algoritmo se detiene.
Iteración anillo: Iteración, en media, en la que el algoritmo encuentra el mejor anillo.
Fitness: Valor medio, mínimo y máximo del mejor fitness encontrado.

n	N	Tiempo		Iteración		Fitness		
		final	anillo	final	anillo	medio	mínimo	máximo
50	10	0,16	0,11	438,53	318,40	16,14	16,08	16,30
50	25	0,48	0,33	510,23	349,70	16,11	16,08	16,20
100	20	1,78	1,45	996,73	814,23	22,20	21,95	22,55
100	50	4,78	3,70	1055,30	820,10	22,08	21,95	22,21
200	40	22,30	19,32	1963,03	1709,97	30,08	29,70	30,99
200	100	59,31	49,88	2086,13	1767,67	29,99	29,69	30,83

Tabla 4.2: Resultados computacionales

Parece observarse que al aumentar el número individuos de la población para resolver un mismo problema las cotas obtenidas son mejores, sin embargo el tiempo de cálculo también aumenta. Además, para cada problema el rango de valores comprendido entre el fitness mínimo y fitness máximo es pequeño, por lo que se puede suponer cierta estabilidad al obtener soluciones mediante este algoritmo.

La mejor solución obtenida para cada uno de los tres grafos se muestra representada gráficamente en las Figuras 4.4, 4.5 y 4.6.

También se han realizado pruebas con un criterio de parada distinto, en el que solo se tiene en cuenta el número máximo de iteraciones $M = 10^4$ y no la diferencia entre el fitness del mejor anillo y el fitness medio de la población en cada iteración. Por ejemplo, utilizando el algoritmo evolutivo con una población de 100 individuos sobre el grafo con

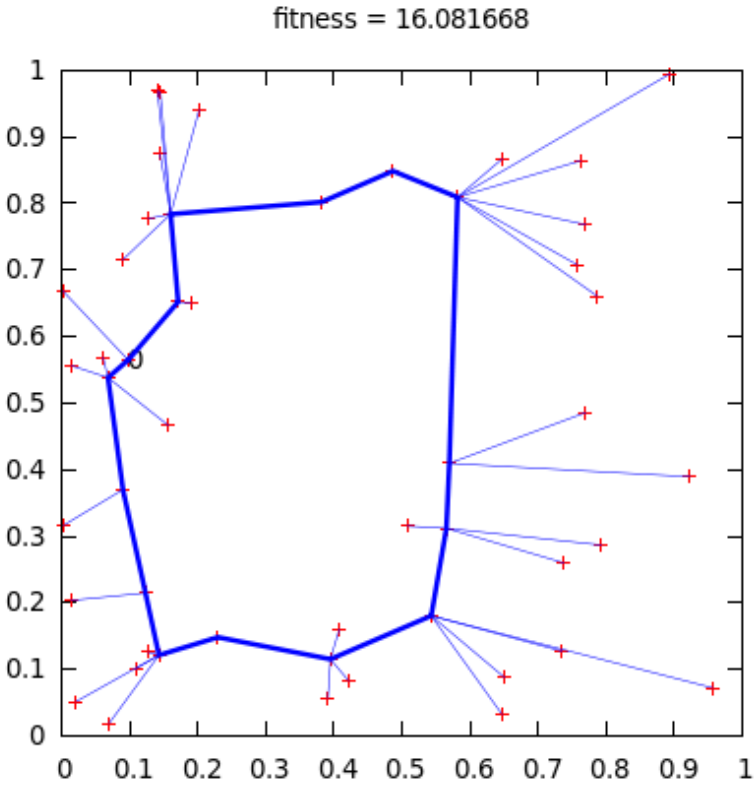


Figura 4.4: Mejor solución encontrada, grafo con 50 nodos

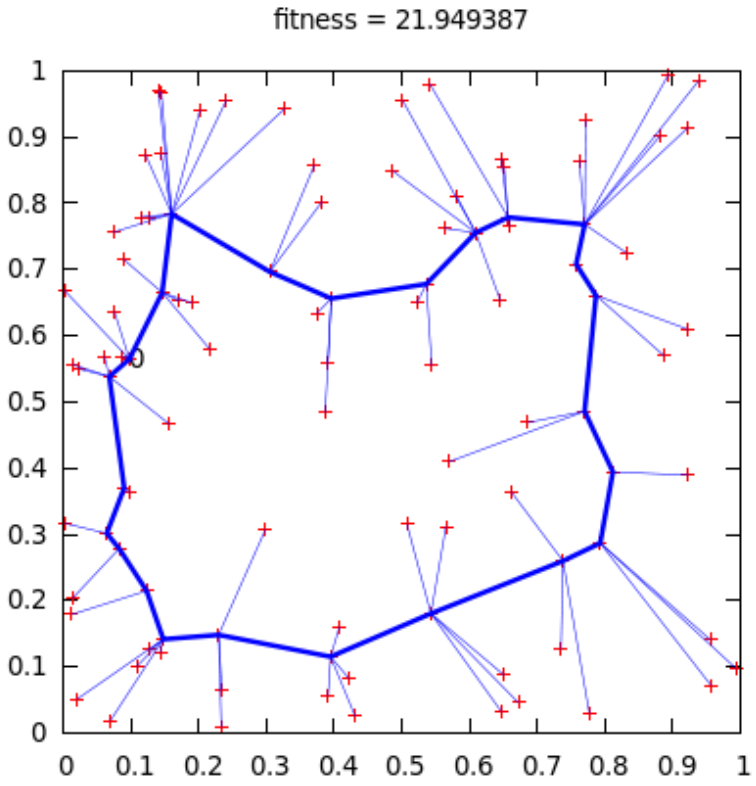


Figura 4.5: Mejor solución encontrada, grafo con 100 nodos

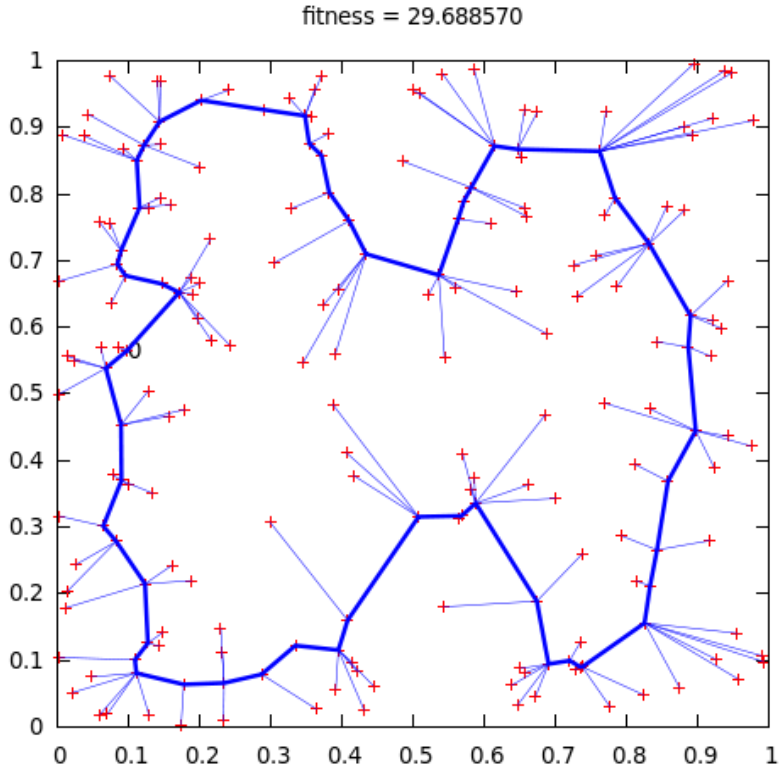


Figura 4.6: Mejor solución encontrada, grafo con 200 nodos

200 nodos se ha obtenido que, tras 10 ejecuciones, se consigue un fitness medio de 29.91, es decir, algo menor que el fitness medio obtenido mediante el criterio de parada inicial en la misma situación. El tiempo medio del algoritmo en este caso aumenta hasta los 301.09 segundos y la última mejora se realiza, en media, alrededor de la iteración 5014. En la Figura 4.7 puede verse una gráfica en el que se muestra la velocidad con la que decrece el valor medio del fitness del mejor anillo a medida que transcurren las iteraciones.

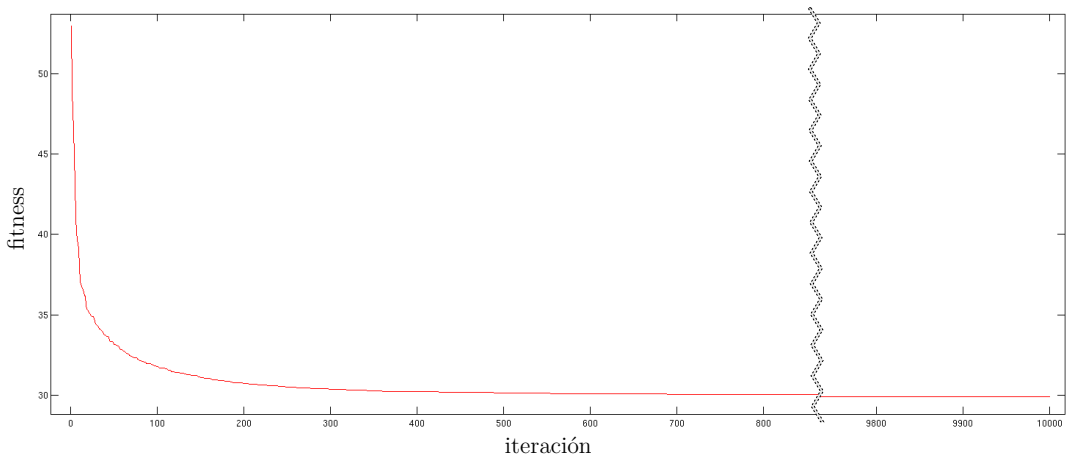


Figura 4.7: Resultados para $n = 200$ y $N = 100$ durante 10^4 iteraciones

Para dar una idea del error que se comete al resolver un problema mediante este algoritmo se han seleccionado varios problemas para los que se conoce una solución exacta,

en la mayoría de los casos, o muy próxima. Los grafos sobre los que se ha resuelto el problema RSP han sido seleccionados del conjunto de problemas TSPLIB (Reinelt (1991)) que contiene grafos con entre 50 y 200 nodos con formato 2-dimensional. Los costes de los arcos y los ejes han sido definidos para obtener soluciones que visitan aproximadamente el 100 %, el 75 %, el 50 % y el 25 % del total de nodos del grafo, de forma que para cada par de nodos $i, j \in V$ se define $c_{ij} = \lceil \omega l_{ij} \rceil$ y $d_{ij} = \lceil (10 - \omega) l_{ij} \rceil$ para $\omega \in \{3, 5, 7, 9\}$, siendo l_{ij} la distancia euclídea entre los nodos i y j.

Los parámetros de entrada del algoritmo evolutivo son $\alpha = \frac{3}{n}$, $\beta = 10^{-2}$, $C = \lceil 0.01n \rceil$, $\varepsilon = 10^{-10}$ y $M = 10^4$. El número de individuos de la población utilizada en el algoritmo es, en todos los casos, $M = 100$. Los grafos de la base de datos TSPLIB sobre los que se han realizado las pruebas son el *eil51*, *rd100* y el *kroA200*, con 51, 100 y 200 nodos respectivamente.

Ejecutando diez veces el algoritmo para cada una de las configuraciones posibles se obtiene la Tabla 4.3 de resultados, que contiene los siguientes datos:

Grafo: Nombre del grafo sobre el que se aplica el algoritmo evolutivo.
ω: Parámetro de proporcionalidad para los costes de ejes y arcos.
Óptimo: Valor óptimo encontrado por Labbé et al. (2004), salvo los casos marcados con * en los que se trata de una cota superior.
Tiempo: Tiempo medio de cálculo hasta que el algoritmo se detiene (en segundos).
Fitness: Valor medio, mínimo y máximo del mejor fitness encontrado.
%Error: Porcentaje de error relativo medio y mínimo.

El porcentaje de error relativo se calcula como $100(fit - opt)/opt$ siendo *opt* la solución dada por el algoritmo de ramificación y acotación descrito en Labbé et al. (2004) y fit la solución heurística obtenida mediante este algoritmo evolutivo. La solución dada por su algoritmo de ramificación y acotación no es la óptima en todos los casos, ya que ejecutaron el algoritmo durante un tiempo máximo de dos horas por problema. Los valores negativos que aparecen en las columnas de *%Error* corresponden a problemas en los que la solución heurística del algoritmo evolutivo es mejor que la solución encontrada por el algoritmo de ramificación y acotación en dos horas.

Notemos que, en general, en las pruebas realizadas el algoritmo tiende a dar mejores resultados a medida que aumenta el coste de los ejes con respecto al coste de los arcos. También se ha obtenido un error menor para los grafos con mayor número de nodos, mejorando en algunos casos las cotas superiores de la solución óptima que obtuvieron Labbé et al. (2004) tras dos horas de cálculo.

Grafo	ω	Óptimo	Tiempo	Fitness			% Error	
				medio	mínimo	máximo	medio	mínimo
eil51	3	1278	3,8	1358,7	1314	1404	6,31	2,82
	5	1995	3,3	2102,9	2084	2148	5,41	4,46
	7	2113	2,3	2167,3	2163	2178	2,57	2,37
	9	1244	0,8	1280	1280	1280	2,89	2,89
rd100	3	23730	10,1	25041,1	23944	26001	5,53	0,90
	5	37975	10,0	38733,9	38075	39733	2,00	0,26
	7	40915	6,9	41168,6	41042	41417	0,62	0,31
	9	31776	5,3	32601,6	32434	32789	2,60	2,07
kroA200	3	93699*	202,2	93925,2	92012	95810	0,24	-1,80
	5	138885*	61,0	143808	140844	148639	3,54	1,41
	7	158227	74,4	160230	158452	163382	1,27	0,14
	9	124678*	45,3	123581	122808	124394	-0,88	-1,50

Tabla 4.3: Resultados experimentales sobre grafos de la TSPLIB

Bibliografía

R. Baldacci y M. Dell'Amico. Heuristic algorithms for the multi-depot ring-star problem. *European Journal of Operational Research*, 203(1):270–281, 2010.

R. Baldacci, M. Dell'Amico y J.J. Salazar González. The capacitated m-ring-star problem. *Operations Research*, 55(6):1147–1162, 2007.

P. Bauer. The circuit polytope: Facets. *Mathematics of Operations Research*, 22(1): 110–145, 1997.

J.C. Bean. Genetic algorithms and random keys for sequencing and optimization. *Informs Journal on Computing*, 6(2):154–160, 1994.

A. Behzad y M. Modarres. A new efficient transformation of the generalized traveling salesman problem into traveling salesman problem. En *Proceedings of the 15th International Conference of Systems Engineering*, pp. 6–8, 2002.

G.A. Croes. A method for solving traveling-salesman problems. *Operations Research*, 6 (6):791–812, 1958.

T. Dias, G. de Sousa Filho, E. Macambira, L. dos Anjos F. Cabral y M. Fampa. An efficient heuristic for the ring star problem. En C. Álvarez y M. Serna, editores, *Experimental Algorithms*, volumen 4007 de *Lecture Notes in Computer Science*, pp. 24–35. Springer, 2006.

M. Ehrgott. *Multicriteria optimization*. Springer Verlag, Berlin, Heildeberg, 2005.

M. Fischetti, J.J. Salazar González y P. Toth. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research*, 45(3):378–394, 1997.

A.L. Henry-Labordere. The record balancing problem: A dynamic programming solution of a generalized traveling salesman problem. *RAIRO Operations Research*, B2:43–49, 1969.

J.H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975.

F.K. Hwang y D.S. Richards. Steiner tree problems. *Networks*, 22(1):55–89, 1992.

R.M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pp. 85–103, 1972.

S. Kedad-Sidhoum y V.H. Nguyen. An exact algorithm for solving the ring star problem. *Optimization*, 59(1):125–140, 2010.

T. Koch y A. Martin. *Solving Steiner tree problems in graphs to optimality*, volumen 32. 1998.

M. Labbé, G. Laporte, I. Rodríguez Martín y J.J. Salazar González. The ring star problem: Polyhedral analysis and exact algorithm. *Networks*, 43(3):177–189, 2004.

M. Labbé, G. Laporte, I. Rodríguez Martín y J.J. Salazar González. Locating median cycles in networks. *European Journal of Operational Research*, 160(2):457–470, 2005.

Y. Lee, S.Y. Chiu y J. Sánchez. A branch and cut algorithm for the Steiner ring star problem. *International Journal of Management Science*, 4(1):21–34, 1998.

A. Liefooghe, L. Jourdan y E.G. Talbi. Metaheuristics and cooperative approaches for the bi-objective ring star problem. *Computers and Operations Research*, 37(6):1033–1044, 2010.

S. Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, (44):2245–2269, 1965.

A. Mauttone, S. Nesmachnow, A. Olivera y F. Robledo. A hybrid metaheuristic algorithm to solve the capacitated m-ring star problem. En *Proceedings of the International Network Optimization Conference (INOC 2007)*. Spa, Belgium, 2007.

Z. Naji-Azimi, M. Salari y P. Toth. A heuristic procedure for the capacitated m-ring-star problem. *European Journal of Operational Research*, 207(3):1227–1234, 2010.

G.L. Nemhauser y L.A. Wolsey. *Integer and Combinatorial Optimization*. Wiley, New York, 1999.

Ch.E. Noon. *The generalized traveling salesman problem*. PhD thesis, University of Michigan, 1988.

J.A. Moreno Pérez, M.V. Marcos y I. Rodríguez Martín. Variable neighborhood tabu search and its application to the median cycle problem. *European Journal of Operational Research*, 151(2):365–378, 2003.

G. Reinelt. Tsplib–a traveling salesman problem library. *INFORMS Journal on Computing*, 3(4):376–384, 1991.

J. Renaud, F.F. Boctor y G. Laporte. Efficient heuristics for median cycle problems. *Journal of the Operational Research Society*, 55(2):179–186, 2004.

M.G.C. Resende y C.C. Ribeiro. Grasp with path-relinking: Recent advances and applications. En T. Ibaraki, K. Nonobe y M. Yagiura, editores, *Metaheuristics: Progress as Real Problem Solvers*, pp. 29–63. Springer, 2005.

L. Simonetti, Y. Frota y C.C. de Souza. The ring-star problem: A new integer programming formulation and a branch-and-cut algorithm. *Discrete Applied Mathematics*, In press, doi:10.1016/j.dam.2011.01.015, 2011.

S. Srivastava, S. Kumar, R.C. Garg y P. Sen. Generalized traveling salesman problem through n sets of nodes. *Journal of the Canadian Operational Research Society*, 7: 97–101, 1969.

R.T. Wong. A dual ascent approach for Steiner tree problems on a directed graph. *Mathematical Programming*, 28(3):271–287, 1984.

Anexo I: Código C++

Archivo: /main.cpp Página 1 de 2

```cpp
#include <iostream>
#include <stack>
#include <stdlib.h>
#include <Anillo.h>

using namespace std;

/*En este archivo se muestra el conjunto de ordenes escritas en C++ para la
resolución del problema RSP sobre grafos aleatorios generados en el conjunto
[0,1]x[0,1]. El coste de ejes y arcos es proporcional a la distancia euclídea
entre los distintos nodos*/

int main(int argc, char *argv[]){

    double d1, d2, mediaPob;
    int k, r1, r2;
    double itMax;

    int n;//Número de nodos de la red
    int tamPob;//Tamaño de la población
    double beta;//Porcentaje de la población que pasa directamente a la siguiente generación
    double pMutacion;//Probabilidad de mutación
    double factor;//Factor que multiplica el coste de los ejes: c_ij = factor * d_ij
    double eps;//Diferencia minima entre el fitness del mejor anillo y la media de la poblacion
    GNUplot plotter;// Sirve para dibujar el grafo (es necesario tener instalado GnuPlot)

    //Lectura de datos o inicialización por defecto
    n= argc>1? atoi(argv[1]) : 100;
    tamPob= argc>2? atoi(argv[2]) : 50;
    beta= argc>3 ? atof(argv[3]) : 0.01;
    pMutacion= argc>4? atof(argv[4]) : 1./double(n);
    itMax=argc>5? atoi(argv[5]) : 1e4;
    factor=argc>6? atoi(argv[6]) : 5;
    eps=argc>7? atoi(argv[7]) : 1e-4;

    vector<pair<double, double> >puntos(n);//Coordenadas de los nodos de la red
    vector<vector<double> > coste(n);//Distancias euclideas entre los puntos
    stack<Anillo> hijos;//Hijos
    multiset<Anillo> poblacion;//Población
    vector< multiset<Anillo>::const_iterator> posPob(tamPob);

    //Generación de puntos aleatorios
    set_random(2);//Semilla fija para la creación del grafo
    for(int i=0; i<n;i++){
        puntos[i]=make_pair(rand01(),rand01());
        coste[i].resize(n);
    }

    //Cálculo de costes
    for(int i=0; i<n; i++){
        for(int j=0; j<n; j++){
            d1=puntos[i].first-puntos[j].first;
            d2=puntos[i].second-puntos[j].second;
            coste[i][j]=coste[j][i]=sqrt(d1*d1+d2*d2);
        }
    }

    //Cambio de semilla (para resolver un mismo problema con distintas poblaciones)
    set_random(time(0));

    //Llamada a la función de inicialización de los anillos
    Anillo::setCostes(coste,factor);

    //Inicio del Algoritmo Evolutivo
    clock_t tiempo=clock();
    Anillo mejorAnillo;

    //Generación de la población inicial aleatoria
    for(int i=0; i<tamPob; i++)
        poblacion.insert(Anillo());

    //Recombinaciones + Mutación + Búsqueda local
```

Archivo: /main.cpp Página 2 de 2

```
    for(int j=0; j<itMax; j++){
        //Actualización del mejor anillo
        if(poblacion.begin()->getFitness()< mejorAnillo.getFitness())
            mejorAnillo = *(poblacion.begin());

        //Actualización de la posición de cada anillo de la población y cálculo de la media
        k=0; mediaPob=0;
        for(multiset<Anillo>::const_iterator it= poblacion.begin(); it!=poblacion.end(); it++){
            posPob[k++]=it;
            mediaPob+=it->getFitness();
        }
        mediaPob/=tamPob;

        //Criterio de parada
        if(mediaPob-mejorAnillo.getFitness() < eps) break;

        //Selección aleatoria de los padres y cruce
        for(int i=0; i<tamPob*(1-beta); i++){
            r1=random2(0,tamPob-1);
            r2=random2(0,tamPob-1);
            hijos.push(Anillo(*(posPob[r1]),*(posPob[r2]), pMutacion));
        }

        //Adición de los nuevos hijos a la población y selección de los mejores
        while(!hijos.empty()){
            poblacion.insert(hijos.top());
            hijos.pop();
            poblacion.erase(--poblacion.end());
        }
    }
    //Fin del algoritmo evolutivo

    cout<<"\nCoste mejor anillo:"<< mejorAnillo.getFitness();
    cout<<", tiempo de calculo : "<<(clock()-tiempo)/double(CLOCKS_PER_SEC)<<endl;
    mejorAnillo.dibujar(puntos, plotter);

}
```

Archivo: /Anillo.h Página 1 de 1

```
#ifndef HEADER_ANILLO
#define HEADER_ANILLO
#include <MisFunciones.h>
#include <map>
#include <set>
#include <vector>
#include <gnuplot.h>

class Anillo{
private:
    static int n;//Numero de nodos de la red
    static std::vector<std::vector<double> > coste;//Matriz de costes
    static int contador;//Contador de anillos
    static double factor;//Factor que multiplica el coste de los ejes usados en el ciclo
    static int maxCruces;//Nº máximo de cruces 2-opt que se quitan.

    int id; //nº que identifica el anillo creado
    std::multimap<double, int> valores; //Nodos del anillo ordenados por su código
    std::set<int> nodos; //Nodos del anillo ordenados por el nº de cada nodo
    std::map<int, int> estrellas;//Nodos no asignados y el correspondiente nodo al que se asignan
    double fitness; //Fitness del anillo

    bool quitarCruce();// Deshace el primer cruce que encuentre. Devuelve false sii no hay ninguno.
    void insertarNodo(const double & valor, const int & nodo);

public:
    // IMPORTANTE: Antes de declarar cualquier elemento de la clase anillo hay que usar esta función
    static void setCostes(const std::vector<std::vector<double> > & cost, const double & f);

    Anillo();//Genera un anillo aleatorio sobre una red de n nodos
    Anillo(const Anillo & a1, const Anillo & a2, const double & pMutacion);
    //Genera un anillo por cruce en un punto y le aplica el operador de mutación y el de búsqueda
local.

    const double & getFitness() const;
    bool estaNodo(const int & nodo) const;
    friend std::ostream& operator << (std::ostream& ostr, const Anillo & anillo);
    bool operator < (const Anillo & anillo) const;
    void operator = (const Anillo & anillo);
    void dibujar(const std::vector<std::pair<double,double> > &  puntos, GNUplot & plotter) const;

};


#endif
```

Archivo: /Anillo.cpp Página 1 de 4

```
#include "Anillo.h"

vector<vector<double> > Anillo::coste;
int Anillo::n;
int Anillo::contador;
double Anillo::factor;
int Anillo::maxCruces;



void Anillo::setCostes(const vector<vector<double> > & cost, const double & f){
    coste=cost;
    n=coste.size();
    factor= f;
    contador=0;
    maxCruces = ceil(n*0.01);
}



Anillo::Anillo(){
    double minimo, aux;
    int nodoMasCercano=0;
    fitness=0;
    id=contador++;

    insertarNodo(0.,0);
    double r=rand01();
    for(int i=1;i<n;i++)
        if(rand01()<r){
            insertarNodo(rand01(),i);
        }else
            estrellas[i]=0;

    //Optimización - Quitar cruces
    for(int i=0; i<maxCruces && quitarCruce(); i++);

    //Cálculo del coste del ciclo
    for(multimap<double, int>::const_iterator it= valores.begin(); it!= --valores.end();){
        fitness+=coste[it->second][(it++)->second];}
    fitness+=coste[(--valores.end())->second][0];
    fitness*=factor;

    //Cálculo de las asignaciones
    for(map<int, int>::iterator it= estrellas.begin();it!=estrellas.end(); it++){
        minimo=1e20;
        for(set<int>::const_iterator it2= nodos.begin(); it2!=nodos.end(); ++it2){
            aux=coste[it->first][*it2];
            if(aux<minimo){
                minimo=aux;
                nodoMasCercano=*it2;
            }
        }
        fitness+=minimo;
        it->second=nodoMasCercano;
    }
}



Anillo::Anillo(const Anillo & a1, const Anillo & a2, const double & pMutacion){
    double minimo, aux;
    int nodoMasCercano=0, r=random2(1, n-2);
    id=contador++;

    fitness=0;
    for(multimap<double, int>::const_iterator it= a1.valores.begin(); it!= a1.valores.end() ;it++){
        if(it->second <= r){
            insertarNodo(it->first, it->second);
        }
    }
    for(multimap<double, int>::const_iterator it= a2.valores.begin(); it!= a2.valores.end() ;it++){
```

Archivo: /Anillo.cpp Página 2 de 4

```
        if(it->second > r){
            insertarNodo(it->first, it->second);
        }
    }

    //Mutación - //Se selecionan 4*pMutacion*n nodos al azar y se mutan con probabilidad 1/4.
    int nNodosElegidos = ceil(4*pMutacion*n);
    for(int nn=0; nn<nNodosElegidos; nn++){
        if(rand01()< 0.25){
            r=random2(1, n-1);
            if(estaNodo(r)){// Si está en el anillo se quita.
                nodos.erase(r);
                for(multimap<double, int>::iterator it=valores.begin(); it!=valores.end(); it++)
                    if(it->second==r){
                        valores.erase(it);
                        break;
                    }
            }else{//Si no está se pone en la mejor posición para el ciclo.
                int i, k=0;
                double posI, posK=0, pos;
                multimap<double, int>::const_iterator it= valores.begin();
                minimo=1e20;
                while(it!= --valores.end()){
                    posI=it->first; i=it->second;
                    posK=(++it)->first; k=it->second;
                    aux=coste[i][r]+coste[r][k]-coste[i][k];
                    if(aux<minimo){
                        minimo=aux;
                        pos=(posI+posK)/2.;
                    }
                }
                if(coste[k][r]+coste[r][0]-coste[k][0] < minimo){
                    pos=(1.+posK)/2.;
                }
                insertarNodo(pos,r);
            }
        }
    }

    //Búsqueda local - Quitar cruces
    for(int i=0; i<maxCruces && quitarCruce(); i++);

    //Cálculo del coste del ciclo
    for(multimap<double, int>::const_iterator it= valores.begin(); it!= --valores.end();){
        fitness+=coste[it->second][(it++)->second];
    }
    fitness+=coste[(--valores.end())->second][0];
    fitness*=factor;

    //Cálculo de las asignaciones
    for(int i=1; i<n; i++){
        if(!estaNodo(i)){
            minimo=1e20;
            for(set<int>::const_iterator it= nodos.begin(); it!=nodos.end(); ++it){
                aux=coste[i][*it];
                if(aux<minimo){
                    minimo=aux;
                    nodoMasCercano=*it;
                }
            }
            fitness+=minimo;
            estrellas[i]=nodoMasCercano;
        }
    }
}



bool Anillo ::quitarCruce(){
    int r, s, rAnt, sPos, tamCiclo=valores.size();
    double t=0; vector< multimap<double, int>::iterator> pos(tamCiclo);
    for(multimap<double, int>::iterator it= valores.begin(); it!= valores.end(); it++) pos[t++]=it;
```

Archivo: /Anillo.cpp Página 3 de 4

```
    for(int i=1; i<tamCiclo; i++){
        for(int j=i+1; j<tamCiclo; j++){
            rAnt= pos[i-1]->second;
            r=pos[i]->second;
            s=pos[j]->second;
            sPos= j==tamCiclo-1 ? 0 : pos[j+1]->second;
            if(coste[rAnt][s]+coste[r][sPos] < coste[rAnt][r]+coste[s][sPos]){
                while(i<j){ swap(pos[i++]->second, pos[j--]->second);}
                return true;;
            }
        }
    }
    return false;
}



const double & Anillo::getFitness() const{
    return fitness;
}



void Anillo::insertarNodo(const double & valor, const int & nodo){
    nodos.insert(nodo);
    valores.insert(make_pair(valor,nodo));
}



bool Anillo::estaNodo(const int & nodo)const{
    return nodos.find(nodo)!=nodos.end();
}



ostream& operator << (ostream& ostr, const Anillo & anillo){
    ostr<<"Id = %"<<anillo.id<<"%";
    int k=0;
    multimap<double, int>::const_iterator it;
    for(it= anillo.valores.begin(); it!= anillo.valores.end();it++){
         ostr<<it->second<<" ";
         if(k++>anillo.n) break;
    }
    return ostr;
}



bool Anillo :: operator < (const Anillo & anillo) const{
    return fitness<anillo.fitness;
}



void Anillo :: operator = (const Anillo & anillo){
    valores = anillo.valores;
    nodos = anillo.nodos;
    estrellas = anillo.estrellas;
    fitness= anillo.fitness;
}



void Anillo::dibujar(const vector<pair<double,double > > & puntos, GNUplot & plotter) const{
    ofstream fPuntos("puntos.txt"), fAnillo("puntosAnillo.txt"), fEstrella("puntosEstrella.txt");
    string graf;

    //Guardando puntos en fichero
    for(vector<pair<double,double > >::const_iterator it = puntos.begin(); it!=puntos.end(); it++){
        fPuntos<<it->first<<"\t"<<it->second<<endl;
    }
```

Archivo: /Anillo.cpp Página 4 de 4

```
    //Guardando puntos del anillo en otro fichero
    for(multimap<double, int>::const_iterator it = valores.begin(); it!=valores.end(); it++){
        fAnillo<<puntos[it->second].first<<"\t"<<puntos[it->second].second<<endl;
    }
    fAnillo<<puntos[0].first<<"\t"<<puntos[0].second<<endl;

    //Guardando puntos de las estrellas en otro fichero
    for(map<int, int>::const_iterator it = estrellas.begin(); it!=estrellas.end(); it++){
        fEstrella<<puntos[it->first].first<<"\t"<<puntos[it->first].second<<endl;
        fEstrella<<puntos[it->second].first<<"\t"<<puntos[it->second].second<<endl<<endl;
    }

    fPuntos.close();
    fAnillo.close();
    fEstrella.close();
    graf+="set title \"fitness = "+doubleToStr(fitness)+"\"\n";
    graf+="set xrange [0:1]\n";
    graf+="set yrange [0:1]\n";
    graf+="set label '0' at "+doubleToStr(puntos[0].first)+","+doubleToStr(puntos[0].second)+"\n";
    graf+="unset key\n";
    graf+="plot 'puntos.txt' lt rgb \"red\",";
    graf+="'puntosAnillo.txt' with lines lt rgb \"blue\" lw "+doubleToStr(0.5*factor)+",";
    graf+="'puntosEstrella.txt' with lines lt rgb \"green\" lw 0.5\n";

    //system("gnuplot -persists graf.plt");
    plotter(graf);
}
```

Archivo: /gnuplot.h Página 1 de 1

```
#ifndef GNUPLOT_H_
#define GNUPLOT_H_
#include <fstream>

using namespace std;

class GNUplot{
 public:
    GNUplot() throw(string);
    ~GNUplot();
    void operator ()(const string& command);

protected:
    FILE *gnuplotpipe;
};
#endif
```

Archivo: /gnuplot.cpp Página 1 de 1

```cpp
#include "gnuplot.h"

using namespace std;


GNUplot::GNUplot()throw(string){
    gnuplotpipe=popen("gnuplot -persist","w");
    if(!gnuplotpipe){
    throw("¡No se encontro GNUPLOT!");
    }
}


GNUplot::~GNUplot(){
    fprintf(gnuplotpipe,"exit\n");
    pclose(gnuplotpipe);
}


void GNUplot::operator()(const string& command){
    fprintf(gnuplotpipe,"%s\n",command.c_str());
    fflush(gnuplotpipe);
}
```

Archivo: /MisFunciones.h Página 1 de 1

```
#ifndef HEADER_MISFUNCIONES
#define HEADER_MISFUNCIONES
#include <fstream>
#include <math.h>

const std::string doubleToStr(const double & k);
void set_random(long);
long random2(long, long);
double rand01();

#endif
```

Archivo: /MisFunciones.cpp Página 1 de 1

```cpp
#include<MisFunciones.h>


const std::string doubleToStr(const double & k){
    char resultado[15];
    sprintf( resultado, "%f", k );
    return resultado;
}



//GENERADOR DE NÚMEROS ALEATORIOS sacado de NETGEN

/*** This is a portable random number generator whose origins are
 *** unknown.  As far as can be told, this is public domain software.*/

/*** portable random number generator */

/*** Note that every variable used here must have at least 31 bits
 *** of precision, exclusive of sign.  Long integers should be enough.
 *** The generator is the congruential:  i = 7**5 * i mod (2^31-1).
 ***/

#define MULTIPLIER 16807
#define MODULUS 2147483647

static long saved_seed;


/*** set_random - initialize constants and seed */
void set_random(long seed){
  saved_seed = seed;
  random2(0,1e3);//Sirve para desechar el primer random que depende mucho de la semilla
}



/*** random - generate a random integer in the interval [a,b] (b >= a >= 0) */
long random2(long a, long b){
  register long hi, lo;

  hi = MULTIPLIER * (saved_seed >> 16);
  lo = MULTIPLIER * (saved_seed & 0xffff);
  hi += (lo>>16);
  lo &= 0xffff;
  lo += (hi>>15);
  hi &= 0x7fff;
  lo -= MODULUS;
  if ((saved_seed = (hi<<16) + lo) < 0)
    saved_seed += MODULUS;

  if (b <= a)
    return b;
  return a + saved_seed % (b - a + 1);
}


/*** rand01 - genera un número aleatorio en (0,1) */
double rand01(){
    return double(random2(1,1e9))/double(1e9+1);
}
```