Proyecto Fin de Grado

Ingeniería de las Tecnologías Industriales

Aplicaciones en la industria del problema de Steiner y su resolución mediante algoritmos genéticos

Autor: Virginia Martínez Lacañina

Tutor: Pablo Cortés Achedad

Dep. Organización Industrial y Gestión de Empresas II

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2018

Proyecto Fin de Grado
Ingeniería de las Tecnologías Industriales

Aplicaciones en la industria del problema de Steiner y su resolución mediante algoritmos genéticos

Autor:
Virginia Martínez Lacañina

Tutor:
Pablo Cortés Achedad
Catedrático

Departamento de Organización Industrial y Gestión de Empresas II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2018

Proyecto Fin de Carrera: Aplicaciones en la industria del problema de Steiner y su resolución mediante algoritmos genéticos

Autor: Virginia Martínez Lacañina

Tutor: Pablo Cortés Abadía

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2018

El Secretario del Tribunal

A mi familia

A mis maestros

AGRADECIMIENTOS

Los ejercicios resueltos al final de este documento han sido obtenidos del “Imperial College” de Londres del cual se está muy agradecido. Ya que sin esos problemas hubiera sido más complicado el estudio del problema de Steiner seguido en este trabajo de carrera.

Por otro lado, la labor desarrollada por todas las personas interesadas en el problema de Steiner y sus aplicaciones han hecho posibles este trabajo en el cual se han incorporado muchas de sus ideologías y teorías.

También hacer mención a todos los profesores del departamento de Organización Industrial y Gestión de Empresas II de la Escuela Técnica Superior de la Universidad de Sevilla. Con ellos, mi interés por la optimización y modelado de problemas se ha formado y gracias a ellos se ha podido llevar a cabo este proyecto.

Finalmente, especial atención a Pablo Cortés Achedad, tutor y pilar de este proyecto. Sin él hubiese sido imposible realizar un trabajo tan minucioso y cuidado. Estoy muy agradecida por su implicación y orientación.

Virginia Martínez

Sevilla, 2018

OBJETO DEL PROYECTO

En el siglo XIX se empezó a formular el problema de Steiner fruto de la curiosidad de algunos matemáticos, como Fermat, que querían obtener: puntos estratégicos en el espacio que permitiesen minimizar la distancia a otros puntos dados.

Sin embargo, la verdadera utilidad del problema surgió con la revolución industrial y las nuevas tecnologías. Para aquel entonces, el matemático Steiner estaba completamente inmerso en la resolución del problema. Así, las redes de infraestructuras, las redes eléctricas, las tuberías de aguas y, en general, todos los problemas referentes a la optimización de caminos fueron las aplicaciones que dio expansión al problema de Steiner. Años más tarde, gracias al desarrollo de las computadoras, fue posible la creación de algoritmos heurísticos que facilitasen la resolución de problemas NP-Completos. Como el algoritmo de Steiner, del que no es posible su resolución en tiempo polinomial.

Sin ir más lejos, los algoritmos genéticos surgieron como necesidad para la resolución del problema, con cada vez más aplicaciones sofisticadas. Tal es el caso de la posible inclusión de los más de 1000 transistores que llega a tener un microprocesador, la minimización de las rutas, la estructura mínima energética de las proteínas o la optimización de un porfolio de acciones… Todas ellas son ejemplo de aplicaciones que motivan el desarrollo de la metaheurística.

Es por ello, el presente documento que explica este problema tan peculiar de tanta utilidad actualmente. Comenzando por una reseña histórica, se realiza una pequeña introducción del problema. Para avanzar en el tema, en la sección 2 se abarca la teoría del problema de Steiner, así como las primeras técnicas de resolución exactas. Durante la sección 3, se informa de la diversa índole de aplicaciones del problema con casos prácticos y reales. A continuación, en la sección 4 se explica el método de resolución que da respuesta al algoritmo genético implementado en este documento. Se aborda tanto el algoritmo genético como el árbol de mínima expansión (MST). Seguidamente, en la sección 5 se muestra el programa implementado. Este, con más de 10 funciones y 400 líneas de código será detalladamente explicado función a función mostrándose sus diagramas de flujo y pseudocódigos. Finalmente, la sección 6 culmina con la resolución real de problemas de Steiner utilizando el programa implementado en MATLAB.

Para terminar, durante las conclusiones obtenidas se podrá ver cómo la complejidad de los problemas solucionados dificulta la obtención del óptimo, debido a que el problema es NP-Completo. Por otro lado, el ajuste de parámetros claves, como el número de iteraciones y la densidad de nodos terminales, podrán disminuir significativamente el tiempo de implementación y el error respecto al óptimo cometido por las técnicas heurísticas. Por ello, se pretende finalmente destacar cómo los algoritmos genéticos resultan ser técnicas muy potentes de resolución mejorando los resultados considerablemente para la mayoría de los casos. Además, el MST se interpone como un algoritmo de peso para problemas de difícil resolución por su breve implementación.

Por todo ello, se pretende hacer llegar la gran utilidad del problema. Se muestran aplicaciones de una gran importancia para no solo la optimización económica o energética, sino también, para la investigación del hombre sobre sus antepasados y morfología. A parte, la implementación real del problema de Steiner complementa al documento. Este permite dar a conocer a sus lectores la posibilidad de generar algoritmos válidos para la resolución del problema.

ÍNDICE

Índice de Figuras

Índice de Tablas

Índice de Gráficas

1 INTRODUCCIÓN

"Una cosa es continuar la historia y otra repetirla"

Jacinto Benavente (1866-1954)

Hace ya más de dos siglos desde que el matemático suizo Leonhard Euler (1707-1783) se planteó estudiar la posibilidad de pasar una sola vez por todos los puentes de la ciudad de Königsberg. Para atravesar el río Pregel, existían siete puentes dispuestos según la Figura 1 que facilitaban el acceso a los diferentes puntos de la ciudad. La disposición de los puentes resultó ser de un llamativo interés para el matemático.

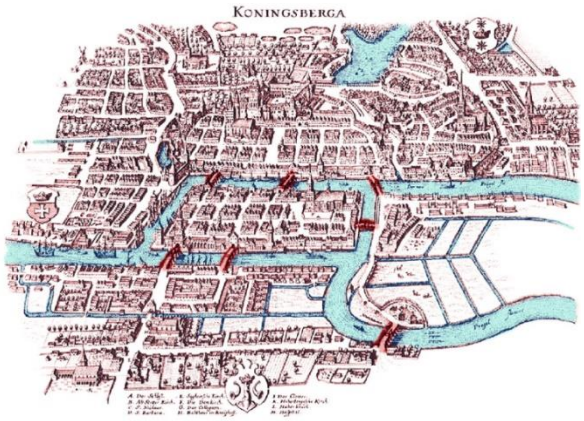


Figura 1. Visualización puentes de Königsberg

Fuente: Research gate. El uso de redes complejas en economía: alcances y perspectivas. (2017).

La resolución de este problema originó los cimientos de la teoría de grafos, la cual quedó abandonada durante los siglos XVII y XVIII. La carencia de las ciencias computacionales, imprescindibles para la resolución de problemas semejantes con mayor número de parámetros, dieron al olvido esta rama de las matemáticas. (Hans Jürgen Prömel y Angelika Steger, 2002).

Por otro lado, Fermat (1601-1665) a principios del siglo XVII formuló al final de su libro "*Treatise on Minima and Maxima*" lo siguiente:

> *Dados tres puntos en un plano, encontrar el cuarto punto de forma que la suma de las distancias de los tres puntos dados sea mínima.*

No fue hasta antes de 1640 cuando su amigo Torricelli, formando un triángulo con los tres puntos dados, dio una solución geométrica para este problema llamado el problema de Fermat. (Dietmar

Cieslik, 2005) Su solución se muestra a continuación.

Los tres círculos circunscritos en los triángulos equiláteros, construidos en los lados exteriores de las aristas que unen los tres puntos dados, interceptan en el punto de mínima distancia.

En la Figura 2 se puede ver la resolución propuesta por Torricelli, cuyo punto que minimiza las distancias de los tres puntos dados, fue nombrado punto de Torricelli:

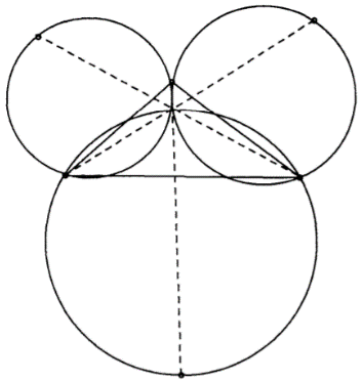


Figura 2. Solución de Torricelli al problema de Fermat

Fuente: Frank K. Hwang et. al (1992)

Posteriormente, en los años siguientes se sucedieron una serie de descubrimientos entorno al problema de Fermat. "Según Frank K. Hwang et. al (1992), en 1647 Cavalieri en su libro "*Excercitationes Geometricae*" demostró que las líneas que unen los tres puntos dados con el punto de Torricelli forman 120º. En 1750, Simpson formuló en su libro "*Doctrine and Application of Fluxions*" que las tres líneas que unen el vértice exterior de cada uno de los triángulos equiláteros, formados en cada lado exterior de las aristas que unen los tres puntos dados, pasan por el punto de Torricelli cuando se dirigen al vértice opuesto del triángulo principal. Las líneas nombradas fueron llamadas líneas de Simpson en honor a su autor, en la Figura 2 se las puede apreciar en línea discontinua. Casi un siglo más tarde, en 1834 fue Heinen quien probó que la longitud de las líneas de Simpson es igual a la suma de las distancias desde el punto de Torricelli a los tres puntos dados. Además, Heinen el mismo año de este descubrimiento junto con Bertrand en 1853 mostró que si el ángulo de alguno de los tres puntos es mayor o igual a 120º, el punto de Torricelli queda fuera del triángulo dado, situándose en el vértice del ángulo obtuso."

"Siguiendo con Frank K. Hwang et. al (1992), con las demostraciones referentes al problema de Fermat, Simpson en su libro "*Fluxions*" propuso a modo de ejercicio el problema de Fermat generalizado a un espacio de d-dimensiones con n puntos a los que calcular la mínima distancia entre ellos. Esta fue probablemente una de las publicaciones sobre el problema de Fermat que a más matemáticos conocidos atrajo, entre ellos Steiner."

Para continuar, Jarník y Kössler (1934) se plantearon la siguiente cuestión: "Encontrar la red más corta que interconecta los n puntos en un plano". En particular, se dedicaron a estudiar los casos para n = 3, 4 y 5 en un polinomio regular. Dichos autores no nombraron el trabajo de Fermat envuelto en este problema debido a que aparentemente se trataba de dos problemas diferentes.

Así, fue como poco a poco se perdió la relevancia del problema de Fermat y se pasó a nombrar problema de Steiner.

Los autores que finalmente propiciaron el olvido del problema de Fermat fueron Courant y Robbins (1941) en su famoso libro "*What is Mathematics?".* Fueron los primeros en afirmar que el problema de Fermat es la red de mínima distancia que interconecta tres puntos en el plano. Sin embargo, nunca nombraron ni a Fermat para el problema de n = 3 puntos ni a Jarník y Kössler para el problema general de n puntos. Por el contrario, ellos atribuyeron el problema al matemático Steiner nombrándolo el problema de Steiner. La popularidad del libro de Courant y Robbins fue la razón principal del cambio de nombre al problema. Además, fueron los que se encargaron de darle difusión.

De esta forma, a Steiner se le atribuyen los méritos del problema de Steiner, que aparte de sus cimientos en el problema de Fermat, fue posible gracias al legado que Leonhard Euler dejó sobre la teoría de grafos.

Con ello, el problema de Steiner considera un grupo de nodos, puntos llamados nodos terminales, ya sea en el plano o en un espacio de n-dimensiones, que están conectados entre sí a través de arcos, cada uno de ellos con una longitud diferente. Adicionalmente, pueden existir puntos intermedios entre los nodos terminales, llamados puntos Steiner. El objetivo final del problema de Steiner consiste en encontrar el camino mínimo que une todos los nodos terminales pudiendo hacer uso de los puntos Steiner (Frank K. Hwang et. al, 1992). Dicho árbol mínimo que une todos los nodos terminales se denomina árbol mínimo de Steiner, SMT. (D.Z. Dhu et. al, 2000).

Como se comentaba al principio el avance en las ciencias computacionales, al igual que las aplicaciones emergentes, ha propiciado el desarrollo expansivo del problema de Steiner en el último siglo. Aplicaciones industriales como el diseño de circuitos integrados, redes de infraestructuras y redes de telecomunicaciones, entre otros campos, han fomentado el estudio del problema que se comenzó a plantear hace ya más de dos siglos.

Debido a sus aplicaciones industriales, el problema del árbol de Steiner se ha extendido a varios espacios métricos. "Entre los más populares se encuentran el árbol de Steiner en el plano euclídeo, el árbol rectilíneo de Steiner y el árbol de Steiner en grafos, los cuales se califican cómo los clásicos problemas de Steiner, según Xiuzhen Cheng et. al (2001)".

Sin embargo, "siguiendo con Xiuzhen Cheng, Karp en 1972 mostró que el árbol de Steiner en grafos es NP-Completo. Lo que demuestra su imposibilidad de ser resuelto en tiempo polinomial. Posteriormente, Garey y Johnson en 1978 determinaron que el árbol rectilíneo de Steiner es también NP-Completo. Mientras que Garey, Graham y Johnson mostraban de igual forma que para el problema de Steiner en un plano euclídeo no es posible su resolución en tiempo polinomial denominándolo, de nuevo, NP-Completo. Lo que llevó a concluir que los problemas clásicos de Steiner improbablemente tenían soluciones óptimas eficientes. Por ello, se han puesto grandes esfuerzos, desde el conocimiento de este inconveniente, en la utilización de buenas aproximaciones para la resolución del árbol mínimo de Steiner". Se busca el compromiso entre un tiempo de ejecución moderado y la mejor aproximación de la solución óptima que puede estar lejos del óptimo por la complejidad del problema (G. Robins, 2000). Tal compromiso se ha tenido en cuenta durante las últimas décadas donde grandes progresos han sido llevados a cabo en el estudio de los árboles de Steiner. Entre ellos, la conjetura de Gilbert-Pollak.

La conjetura de Gilbert-Pollak sobre el ratio de Steiner en 1968 fue clave para dar paso a los algoritmos heurísticos en la resolución del problema para minimizar los tiempos de resolución. A la conjetura le rodea un interesante folclore que se explica a continuación.

La universidad de California tiene 9 campus que están conectados a través de la línea telefónica privada que los une, la cual se subcontrata a una empresa telefónica. Antes de 1967, el gasto de la universidad en telefonía era determinado por la distancia del árbol mínimo de expansión de los lugares que se querían comunicar. Pero ese mismo año, una compañía aérea descubrió que el árbol mínimo de Steiner (SMT) era de distancia menor que el árbol de mínima expansión. Llamado actualmente árbol de mínima expansión ("*minimum spanning tree")*, MST, se muestra diferente al SMT por no existir puntos Steiner. Luego, la compañía aérea colocó algunos servicios en los puntos Steiner para abaratar los gastos impuestos por la compañía telefónica. Dichos puntos aumentaron la utilización del teléfono y disminuyó el coste que suponía utilizarlo. Con ello, la compañía de telefonía se preguntó como debían ser remodeladas las tarifas por distancia de línea contratada con la implementación del SMT (Xiuzhen Cheng et. al, 2001).

Esta cuestión fue la que motivó el estudio del ratio de Steiner que mide la distancia del problema de Steiner en comparación con el problema de expansión mínima. De nuevo, según Xiuzhen Cheng "La conjetura de Gilbert y Pollak fue la que dio respuesta al valor del ratio de Steiner para el plano euclídeo. Se demostró que la distancia del árbol mínimo de Steiner debía ser como poco $\sqrt{3}/2$ la distancia del árbol mínimo de expansión. Con lo que se demuestra la gran relación entre el MST que es resuelto en tiempo polinomial y el SMT que es NP-Completo. De esta observación se puede obtener una aproximación del árbol mínimo de Steiner en un tiempo computacional razonable a partir del MST. Con ello, el trabajo que desarrollaron Gilbert y Pollak supuso un punto de inflexión en el estudio del árbol de Steiner. Dicho problema, que había sido estudiado con fines matemáticos y que había tenido un progreso muy lento, fue introducido en la industria moderna por el descubrimiento de su resolución en tiempo polinomial". Desde entonces, el árbol de Steiner atrae cada vez más atención y el número de investigaciones crece muy rápido.

Adicionalmente, el MST es considerado como la mejor aproximación polinomial del árbol mínimo de Steiner. Desde hace más de veinte años (1968-1990), se han descubierto muchos algoritmos de aproximación polinomial (por ejemplo, Chang en 1979, Korthonen en 1979, Kou y Makki en 1987, Smith, Lee y Liebman en 1981, Waxman e Imase en 1988, Smith y Shor en 1992…) (C. Gröpl, et. al, 2000). Sin embargo, ninguno de ellos consigue mejorar el ratio de Steiner. Esta situación se extiende a los demás espacios métricos dejando por solucionar el problema de la mejor aproximación. Este problema cuestiona la existencia de un algoritmo que aproxime en tiempo polinomial el árbol mínimo de Steiner mejor que el MST. No fue hasta 1993 cuando Zelikovsky probó que el ratio de Steiner podía ser $11/6$ para el problema de Steiner en grafos que anteriormente era 2 (A. Zelikovsky, 1993). Un año después en 1994, Berman y Ramaije también demostraron para el árbol rectilíneo de Steiner que existía una mejor aproximación de tiempo polinomial que el MST. Posteriormente, se consiguió reducir la aproximación del árbol de Steiner para todas las métricas poniendo en duda la efectividad del MST. Sin embargo, a día de hoy es el algoritmo de aproximación por excelencia debido a su sencillez. El MST permite aproximar fácilmente proporcionando soluciones de calidad.

En la literatura, existen otras muchas técnicas diseñadas para resolver el árbol mínimo de Steiner apoyándose en el MST, el cual permite obtener soluciones iniciales admisibles. Dada la urgencia de dar una solución al SMT debido a sus crecientes aplicaciones en la industria, surgieron técnicas metaheurísticas que mejoraban a las técnicas heurísticas. Entre ellas, se encuentran "*simulated annealing*" y los algoritmos genéticos. El primer método estudiado por Dowsland en 1991 proporcionaba resultados mejores que los métodos clásicos de resolución. Por otro lado, los algoritmos genéticos, que se desarrollaron algo después, fueron capaces de mejorar todos los algoritmos anteriores.

Luego, el empleo de algoritmos genéticos junto con algoritmos de resolución del problema MST surgen como enfoques muy prometedores para proporcionar soluciones que resuelvan los problemas de Steiner (Xiuzhen Cheng and Ding-Zhu Du, 2001). El diseño de redes de telecomunicación, de infraestructuras, de redes eléctricas, de tuberías, de redes biológicas o incluso el diseño de circuitos integrados son los motivos que fomentan el estudio del problema de Steiner. En este documento, se podrá estudiar en profundidad el problema de Steiner, sus aplicaciones y especialmente su resolución llevada a cabo con algoritmos genéticos.

Finalmente, aunque hay un potencial muy grande de mejora en la resolución del árbol de Steiner, se ha conseguido, en la medida de lo posible, proporcionar soluciones que mejoran considerablemente las aproximaciones inicialmente dadas. Los avances tecnológicos y las nuevas aplicaciones que todavía faltan por llegar serán las que den un giro de 360º a la resolución del árbol de Steiner. Mientras tanto, se trabaja en las técnicas encontradas y se estudian sus mejoras que permiten traer al mundo de hoy las soluciones del mañana.

2 ÁRBOL DE STEINER

"No hay rama de la matemática, por abstracta que sea, que no pueda aplicarse algún día a los fenómenos del mundo real"

Nikolai Lobachevski

2.1 El problema de Steiner

Jacob Steiner, matemático suizo de comienzos del siglo XIX, propuso un problema de importancia fundamental dentro del campo de la optimización. El problema de Steiner en redes considera un conjunto de puntos llamados nodos terminales y nodos Steiner conectados a través de arcos que poseen costes. Un árbol dentro del grafo será aquel conjunto de nodos y arcos, los cuales conectan todos los nodos terminales pudiendo conectar algunos nodos Steiner.

La resolución del problema de Steiner en grafos consiste en obtener el árbol mínimo, de forma que todos los nodos terminales estén conectados entre sí minimizando el coste total de los arcos que forman parte del árbol. Para ello, se puede hacer uso de los nodos Steiner (Frank K. Hwang et. al , 1992).

Matemáticamente, el problema de Steiner en grafos se define del siguiente modo:

Dado: Un grafo no dirigido conectado por arcos ponderados $G = (V, E)$, donde *V* es el número de nodos, *E* el número de arcos, $c(e) > 0$ denota el peso del arco e ∈ E y N_G es un conjunto de nodos terminales, tal que $N_G \subseteq V$.

Encontrar: Un subgrafo conectado $T = (V(T), E(T))$ en G, tal que $N_G \subseteq V(T)$ y $\sum_{e \in E} c(e)$ sean mínimos.

Un árbol $T = (V(T), E(T))$, que representa la solución del problema de Steiner, se llama árbol mínimo de Steiner ("*Steiner minimal tree", SMT*). Un ejemplo del árbol mínimo de Steiner se muestra en la Figura 3, donde los nodos $S(T) = V(T) \setminus N_G$ se denominan nodos Steiner o puntos Steiner de *T* (aquellos representados en negro en la figura). Se puede apreciar que los nodos Steiner en *T*, en su mayoría, presentan grado dos debido a que son nodos internos de algún camino en *T*. El grado de un nodo viene determinado por el número de nodos adyacentes directamente conectados.

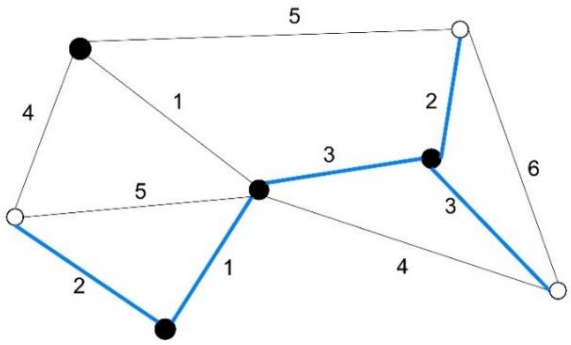


Figura 3. Ejemplo árbol de Steiner

Fuente: Creación propia

Como se ha nombrado anteriormente, el problema de Steiner en grafos es NP-Completo. (M. R. Garey et. al., 1977 y R. M. Karp, 1972). Siendo imposible para un gran número de casos en grafos, la resolución del problema en tiempo polinomial. Adicionalmente, para algunos casos especiales como grafos planares (Garey et. al, 1977) y grafos completos (donde cualquier $e \in E$ conecta dos nodos cualesquiera del grafo) (Bern, 1989) tampoco es resoluble el problema en tiempo polinomial. Por ello, algoritmos capaces de reducir el tamaño del grafo manteniendo sus propiedades, son cruciales en la reducción del tiempo de resolución del árbol de Steiner, especialmente en algoritmos exactos.

2.2 Reducción de grafos

Las técnicas de reducción de grafos juegan un papel muy importante para los algoritmos exactos y heurísticos de resolución del problema de Steiner. Especialmente, para los algoritmos exactos que llegan a tener un tiempo de resolución exponencial muy por encima de la duración media de los algoritmos heurísticos. Poder disminuir el número de grafos supone todo un reto capaz de reducir sustancialmente el tiempo computacional de los algoritmos de resolución del problema de Steiner.

La condición indispensable que cumplen los algoritmos reductores de grafos es minimizar el tamaño del grafo inicial mientras se conserva al menos un árbol mínimo de Steiner (pueden existir más de una solución óptima). De esta forma, siempre se podrá obtener la solución del grafo original a partir de un grafo con menos nodos y/o arcos.

Dichas técnicas de reducción permiten disminuir los nodos del grafo podando nodos Steiner de grado uno, a la misma vez que, eliminar los arcos $e = (u, v) \epsilon E$ de coste *c(e)* cuando exista algún camino entre *u* y *v* en el grafo *G* de coste menor que *c(e).* (Marcus Brazil y Martin Zachariasen, 2010).

Aunque en la práctica ya no sean usados, los primeros métodos de reducción de grafos fueron presentados por Duin (1993), Duin y Volgenant (1989), y Hwang, Richards y Winter (1992). Winter propuso una serie de reducciones que eran particularmente útiles para los problemas rectilíneos del problema de Steiner (por ejemplo, "*the Hanan grid graph"*). Duin (2000) y Uchoa, Aragão y Ribeiro (2002) extendieron la idea de Winter para los problemas particulares rectilíneos aplicados

al diseño de microprocesadores. Además, debido a las ventajas que proporciona poder disminuir el número de arcos y nodos, actualmente existen algoritmos de reducción que pueden disminuir grafos a una pequeña porción del original usando tiempos polinomiales de bajo grado. (Cuanto mayor grado tenga el polinomio, mayor será el esfuerzo de computación). Dichos algoritmos son atribuidos a Polzin y Vahdati Daneshmand (2001-2004).

En el algoritmo heurístico trabajado en este documento, se desprecia el tiempo ganado en la reducción del grafo, a pesar de las ventajas que ello pueda conllevar. Por el contrario, se tiene en consideración la eliminación de todos los nodos Steiner de grado uno pertenecientes al árbol mínimo de Steiner antes de proporcionar la solución. Por ello, aunque no se utilice la reducción de grafos en el algoritmo trabajado, es conveniente el conocimiento de las técnicas nombradas. Sobre todo, es relevante recalcar la importancia de eliminar aristas y/o nodos Steiner para el acondicionamiento del problema antes de aplicar una de las técnicas de resolución exactas nombradas en este apartado. Antes de ello, se definen otros árboles de Steiner considerados en espacios euclídeos o, comúnmente, bidimensionales.

2.3 Árboles de Steiner en otros espacios métricos

Aparte de los árboles de Steiner en grafos utilizados en su mayoría en tres dimensiones, existen otros árboles de Steiner de dos dimensiones. En ellos, se utilizan distancias euclídeas y distancias Manhattan de especial interés para nuevas aplicaciones tecnológicas. Son los llamados árboles euclídeos de Steiner ("*minimum euclidean Steiner tree"*) y árboles de Steiner rectilíneos ("*minimum rectilinear Steiner tree"*), los cuales serán presentados en los siguientes subapartados.

2.3.1 Minimum Euclidean Steiner Tree

El árbol mínimo euclídeo de Steiner surge durante la planificación y construcción de largas redes donde sólo se tiene un conjunto de nodos terminales en un espacio definido. No existe, por consiguiente, arcos de unión entre nodos permitiendo la libertad de añadir nodos intermedios para reducir las distancias entre los nodos terminales. Los llamados problemas de árboles euclídeos de Steiner se utilizan en aplicaciones que envuelven espacios métricos con distancia euclídea, confinados en planos de dos dimensiones.

Las aplicaciones más conocidas se reparten en campos como; redes de telecomunicaciones, distribución de redes eléctricas o tendido de tuberías de aceite o agua. En dichas estructuras la mayoría de los costes se ven involucrados en la conexión de los nodos y generalmente se pueden elegir puntos Steiner intermedios. Otras aplicaciones como el diseño de circuitos integrados o microprocesadores usan este tipo de árboles de Steiner donde se tienen que construir redes de grandes dimensiones.

Formalmente, en el árbol euclídeo de Steiner se tiene un conjunto $P = \{p_1, p_2, \dots, p_N\}$ de N puntos coplanares llamados puntos de lugar. El problema del árbol euclídeo mínimo de Steiner consiste en encontrar la mínima conexión entre los N puntos, donde el árbol puede contener otros nodos en el plano, diferentes de los puntos de lugar. Este conjunto $S = \{s_1, s_2, \dots, s_M\}$ de *M* puntos extra son los llamados puntos Steiner.

Un sencillo ejemplo es dado en la Figura 4 donde los cuatro nodos mostrados deben de ser conectados mediante un árbol de mínima distancia. Si se conectan mediante el MST se obtendrá el árbol de la izquierda con coste 10. En cambio, si se añaden puntos de Steiner extra, mostrados en el centro con color negro, la distancia del árbol es reducida más de un 10%.

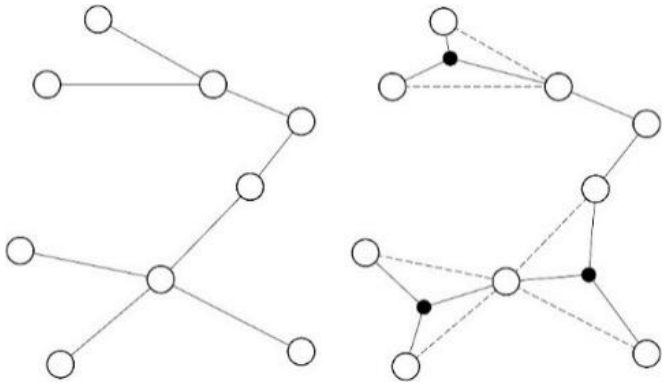


Figura 4. MST vs. SMT

Fuente: Creación propia

De forma similar, la Figura 5 muestra los mismos resultados que la figura anterior. En este caso, se hace repetidamente uso del punto de Fermat-Torricelli para posicionar puntos Steiner que minimicen aquellas distancias entre nodos que formen un triángulo.

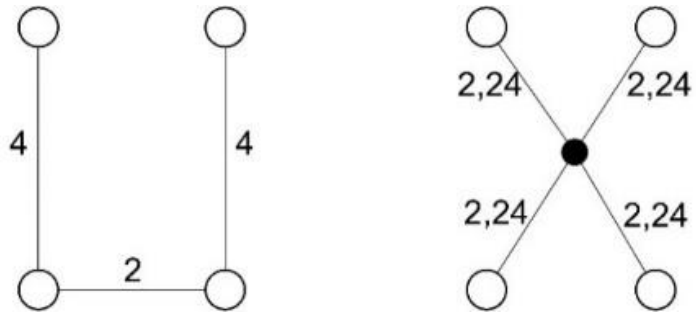


Figura 5. Otro ejemplo MST vs. SMT

Fuente: Creación propia

Encontrar el árbol mínimo de Steiner resulta más flexible que el árbol mínimo de expansión hallado en cualquiera de las dos figuras presentadas. Esto es debido, a que se puede añadir cualquier punto intermedio en una posición conveniente para minimizar la distancia del árbol. Adicionalmente, como el tiempo computacional crece a medida que existen más nodos de lugar en el plano, este problema ha sido también catalogado como NP-Completo. (Gilbert y Pollak, 1986)

2.3.2 Minimum rectilinear Steiner Tree

El problema del mínimo árbol de Steiner rectilíneo es similar al problema del mínimo árbol euclídeo de Steiner, con la restricción de que los arcos a conectar, en el conjunto de nodos, presentan distancia Manhattan, es decir, están colocados en horizontal y vertical. En la Figura 6 se ilustra tal problema donde nueve puntos de lugar se tienen que conectar mediante arcos horizontales y verticales para formar un árbol rectilíneo mínimo de Steiner. En la figura de la izquierda, se muestran los puntos a conectar, mientras que en la figura de la derecha se ilustra el problema resuelto.

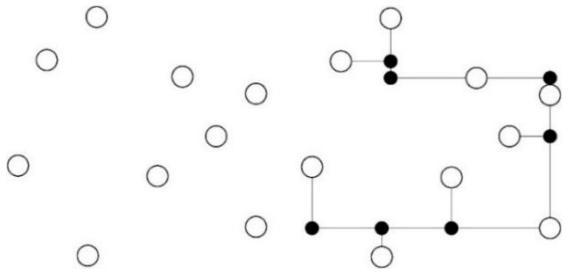


Figura 6. Solución árbol de Steiner rectilíneo

Fuente: Creación propia

Se demostró que se trata de un árbol de Steiner NP-Completo (Garey y Johnson, 1977). Aunque Hwang probó que el ratio entre el MST y el árbol mínimo rectilíneo de Steiner no puede ser mayor que 3/2, lo cual facilita la resolución por técnicas heurísticas tomando el MST como punto de partida. Gracias a esta demostración y a la rápida progresión de las tecnologías, se han unido las técnicas de resolución del árbol de Steiner rectilíneo y la nanotecnología para determinar las conexiones óptimas entre los miles de componentes de un microprocesador. Un avance muy importante que permite fabricar microchips cada vez más pequeños. Aparte, el árbol de Steiner rectilíneo presenta un diseño de red fácil de llevar a la práctica siendo el más recomendado para las conexiones de componentes en un microprocesador.

Toda la información sobre el apartado árboles de Steiner en otros espacios métricos se puede encontrar aquí (Xiuzhen Cheng and Ding-Zhu Du, 2001).

Sin más demora, se exponen a continuación explícitamente las técnicas exactas más conocidas para la resolución de árboles de Steiner en grafos.

2.4 Técnicas de resolución exactas

Las técnicas exactas fueron las primeras investigadas para la resolución del árbol de Steiner. A pesar de ser aplicables para varios espacios métricos, se explicarán específicamente para árboles de Steiner en grafos los cuales son de gran interés en el estudio realizado. Aunque los tiempos de procesamiento son no polinomiales para grafos de gran tamaño, los **métodos exactos se consideran de gran valía cuando los grafos contienen un número reducido de nodos**. De este modo, se presentan en este apartado concisamente las dos grandes opciones de resolución exacta del problema de Steiner; programación dinámica y programación entera (lineal).

2.4.1 Programación dinámica

El principio de la programación dinámica es resolver problemas recursivamente, es decir, dar solución a problemas empezando de abajo arriba guardando soluciones óptimas de subproblemas y combinando dichas soluciones dentro de soluciones óptimas de subproblemas más grandes. La programación dinámica es muy efectiva cuando se tratan casos especiales en los que, por ejemplo, el número de terminales y nodos Steiner es pequeño. Además, fue uno de los primeros métodos utilizados en la resolución no trivial del problema del árbol de Steiner en grafos de forma exacta.

El problema reside en obtener el coste total de un árbol mínimo de Steiner de un subconjunto de nodos *X* siendo $X \subseteq N_G$, y finalmente $C(N_G)$, que es el coste total del árbol del grafo *G* que contiene $|N_G|$ nodos terminales.

2.4.1.1 Algoritmo de Dreyfus-Wagner

El algoritmo de Dreyfus-Wagner (1972) es una de las implementaciones más ingeniosas de la programación dinámica. Por ser una de las primeras estudiadas y de las más conocidas se presenta a continuación su funcionamiento.

Para comenzar se proporcionan las siguientes consideraciones: $v \in V \setminus X$, $c(X \cup v)$ es el coste total de un árbol mínimo de Steiner para $X \cup v$ donde *v* no tiene restricciones de grado, $c_v(X \cup v)$ es el coste total de un árbol mínimo de Steiner para $X \cup v$ donde v tiene grado dos o más y T_v es el mínimo árbol de Steiner para el conjunto $X \cup v$, donde v tiene grado mayor o igual a dos.

Se puede descomponer T_v en dos subárboles T_v^1 y T_v^2. De manera, que T_v^1 sea el SMT para $X' \cup v$ donde $X' \subset X$, y T_v^2 el SMT para $(X \setminus X') \cup v$.

Se obtiene, por consiguiente, la expresión:

$$C_v(X \cup v) = min_{\emptyset \subset X' \subset X}\{c(X' \cup v) + c((X \setminus X') \cup v)\}$$

Que significa que el mínimo coste del conjunto de nodos del grafo más el nodo v, donde v tiene grado mayor o igual a dos, es igual; al mínimo de los costes de un subconjunto de nodos X' más el nodo *v*, más el coste del subconjunto restante de nodos $X \setminus X'$ más el nodo *v*.

Ahora bien, considerando T como el mínimo árbol de Steiner para el conjunto $X \cup v$, donde *v* no tiene restricciones de grado, se pueden presentar dos posibles situaciones. Si v tiene grado dos o más $c(X \cup v) = c_v(X \cup v)$.

En cambio, si v tiene grado uno se considera $\mathrm{p_T(v, w)}$ en T el camino único de v al primer vértice w en T, donde w puede ser un terminal o nodo Steiner de grado mayor o igual a 3 (todos los vértices interiores, si hay, de $\mathrm{p_T(v, w)}$ son vértices Steiner de grado dos).

- Si w es un nodo terminal, el coste total de T será el del camino más corto entre v y w más el coste del árbol mínimo de Steiner para X donde $\mathrm{w} \in \mathrm{X}$.
- Si w es un nodo Steiner de grado mayor o igual a 3, el coste total de T es el peso del camino más corto entre v y w más el coste del árbol mínimo para $X \cup w$, donde $w \notin X$ y w tiene grado ≥ 2.

Se obtiene la siguiente expresión que minimiza el coste del árbol de Steiner según el algoritmo de Dreyfus-Wagner.

$$C(X \cup v) = \min\{min_{w \in X}\{c_p(v,w) + c(X)\},\ min_{w \in V \setminus X}\{c_p(v,w) + c_w(X \cup w)\}\}$$

Donde se elige el mínimo entre las posibles conexiones de v con w siendo este otro nodo terminal o un nodo Steiner de grado mayor o igual a 3.

Este algoritmo se ve utilizado en pequeños conjuntos de nodos donde se puede implementar con eficiencia. Las variaciones de este algoritmo son las encargadas de resolver largos problemas de grafos eficientemente.

2.4.2 Programación entera

El uso de la programación lineal entera en problemas de optimización se remonta a Yang y Wing (1973). Aunque fueron Aneja (1980) y Wong (1984) los primeros en usar la programación lineal entera para la resolución de los problemas de Steiner en grafos.

En esta sección se presentan tres de las más conocidas formulaciones de programación entera. La primera formulación expuesta será la del árbol de expansión ("*spanning tree formulation"*), la cual no se ha utilizado demasiado en los problemas de grafos sino más bien para la resolución de árboles mínimos de Steiner y árboles de expansión mínima en hipergrafos. Seguidamente, la formulación de los cortes ("*cut formulation"*), cuya una de sus variantes es la que actualmente se utiliza y para finalizar la formulación de flujos *("multi-commodity Flow formulation"*) que contiene gran número de variables y restricciones.

2.4.2.1 Spanning tree formulation

El modelo de optimización del árbol de expansión con nodos Steiner se presenta así:

$$\min \sum_{e \in E} c(e)\, x_e$$

$$\begin{aligned} s.t.:\quad & x(E) = y(S_G) + |N_G| - 1 \\ & x(E(W)) \le y(S_G \cap W) + |N_G \cap W| - 1 \qquad W \subset V,\ W \cap N_G \neq \varnothing \\ & x_e \in \{0,1\} \qquad e \in E \\ & y_v \in \{0,1\} \qquad v \in S_G \end{aligned}$$

Los datos necesarios para llevar a cabo el modelo de optimización son:

V: conjunto de nodos Steiner y nodos terminales en *G*

E: conjunto de aristas en *G*

$c(e)$: coste asociado a la arista $e \in E$

S_G: nodos Steiner en *G*

N_G: nodos terminales en *G*

Las variables utilizadas en la formulación representan:

x_e: $\begin{cases} 1 & \text{si } e \in E \text{ forma parte del árbol de Steiner} \\ 0 & \text{en otro caso} \end{cases}$

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

x(E(W)): aristas dentro del conjunto de nodos *W* que forman parte del árbol de Steiner

y_v: $\begin{cases} 1 & \text{si } v \in S_G \text{ forma parte del árbol de Steiner} \\ 0 & \text{en otro caso} \end{cases}$

Donde la función objetivo de la formulación se encarga de sumar todos los costes de las aristas que forman parte de la solución. En la primera restricción, se indica que el número de aristas del árbol de Steiner debe ser igual al número de nodos menos uno. En la segunda restricción, se asegura que cuando se tenga un árbol de Steiner candidato a ser solución no se pueden formar lazos. Lo que es equivalente a; el número de arcos que unen un subconjunto de nodos W (donde $W \subset V,\ W \cap N_G \neq \emptyset$) pertenecientes a un árbol de Steiner tienen que ser como mucho, iguales al número de los nodos de W menos uno. (Goemans et. al, 1993)

Para la mejor comprensión del algoritmo se expone por pasos un ejemplo gráfico que recoge la idea principal del método. Se han omitido los nodos Steiner para no enturbiar la explicación.

1º paso:

Supongamos que se tiene el grafo representado en la **¡Error! No se encuentra el origen de la referencia.**. El conjunto de aristas marcadas en color será una de las primeras soluciones que se obtendría sin hacer caso a la segunda restricción de la formulación. Como se puede ver, con dicha solución la función objetivo está minimizada y se cumple que el número de aristas de la solución es el número de nodos menos uno.

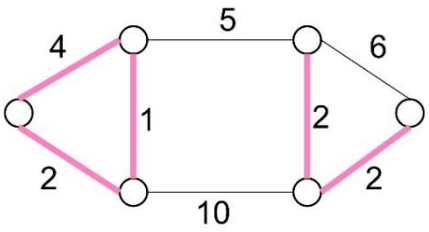


Figura 7. Paso 1 Spanning Tree Formulation

2º paso:

Sin embargo, al aplicar la segunda restricción se observa la existencia de un lazo correspondiente con el óvalo sombreado de la **¡Error! No se encuentra el origen de la referencia.**. Se deja de cumplir que el número de aristas, que forman parte de la solución, que unen el subconjunto de nodos W rodeado sea menor o igual al número de nodos en W menos 1.

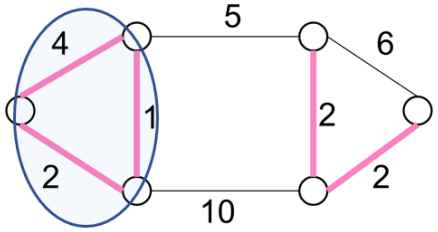


Figura 8. Paso 2 Spanning Tree Formulation

3º paso:

La arista que forma parte del lazo con mayor coste es eliminada en la Figura 9 y se busca otro arco con el coste mínimo posible que cumpla la primera restricción de la formulación. En este caso, el arco con coste cinco será el que pasa a formar parte del árbol.

Para este ejemplo, se habría terminado con el algoritmo obteniendo una función objetivo de doce unidades de coste.

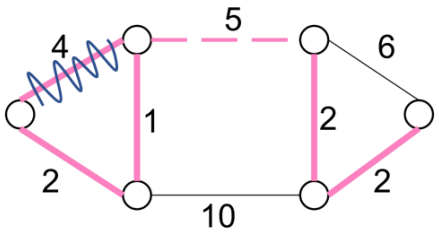


Figura 9. Paso 3 Spanning Tree Formulation

La formulación del árbol de expansión recuerda al algoritmo de Kruskal utilizado en el método heurístico en el que se basa este documento. Sin embargo, el algoritmo de Kruskal será utilizado para calcular el árbol de mínima expansión ("*mínimum spanning tree*", MST) donde no figuran nodos Steiner.

2.4.2.2 Cut formulation

La formulación de cortes se focaliza en la conectividad del grafo, lo que significa que debe existir un camino entre cualquier par de terminales del árbol de Steiner. (Aneja et. al, 1980)

El modelo de optimización para la formulación de los cortes es:

$$\min \sum_{e \in E} c(e) x_e$$

$$s.t.: \quad x(\delta(W)) \geq 1 \qquad W \subseteq V, \quad W \cup N_G \neq \emptyset$$

$$x_e \in \{0,1\} \qquad e \in E$$

Los datos introducidos en el modelo son:

V: conjunto de nodos Steiner y nodos terminales en *G*

E: conjunto de aristas en *G*

N_G: nodos terminales en *G*

$c(e)$: coste asociado a la arista $e \in E$

W: subconjunto de nodos o nodo terminal perteneciente a *V*

Donde las variables de interés son:

$x(\delta(W))$: conjunto de arcos que están unidos al subconjunto *W*

x_e: 1 si $e \in E$ forma parte del árbol de Steiner

0 en otro caso

Al igual que en la formulación anterior, la función objetivo en "*cut formulation*" se encarga de minimizar el coste total del árbol de expansión. La primera restricción asegura que al producirse un corte alrededor de un conjunto de nodos o rodeando a un nodo terminal, siempre exista un

arco que traspasa ese corte que forma parte del árbol de Steiner.

Se presenta a continuación un ejemplo que ilustra el funcionamiento del método.

En la Figura 10 se puede ver una de las primeras soluciones proporcionadas por el algoritmo sin tener en cuenta la primera restricción. Cuando se aplica la restricción del corte para el conjunto *W* formado por los tres nodos que forman un lazo, se puede observar cómo no existe unión con el resto de la solución planteada. Luego, será necesario que el arco con menor coste, en este caso aquel de coste cinco, pase a formar parte de la solución. Finalmente, se eliminará de la solución el arco de coste cuatro debido a que sin ella el coste total disminuye y se sigue cumpliendo la restricción del corte para $\forall W \subseteq V$.

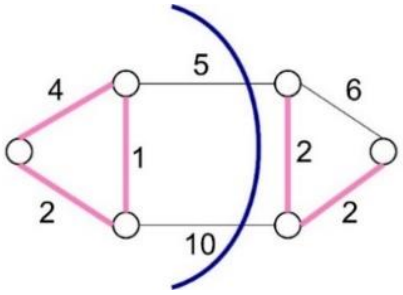


Figura 10. Final Spanning Tree Formulation

Fuente: Elaboración propia

2.4.2.3 Multi-commodity Flow Formulation

Para este método se utiliza el grafo dirigido obtenido del grafo original. Un grafo dirigido es aquel cuyos nodos están conectados mediante dos arcos (en vez de uno) a otro nodo adyacente. El coste de los arcos que conectan un par de nodos será el coste original del grafo anterior. Un ejemplo se muestra en la Figura 11.

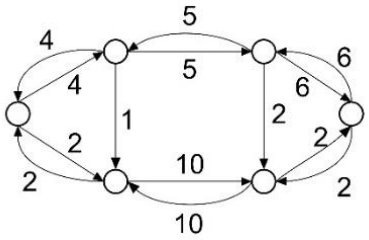


Figura 11. Ejemplo Flow Formulation

Fuente: Elaboración propia

Para la resolución del árbol de Steiner es preciso generar un flujo desde un nodo que se llamará *r* a uno final *t*.

El modelo de programación entera se expresa matemáticamente:

$$\min \sum_{a \in A} c(a) w_a$$

$$s.t.: \quad f^t(\delta^+(v)) - f^t(\delta^-(v)) = \begin{cases} 1 & v = r \\ -1 & v = t \\ 0 & v \in V \setminus \{r,t\} \end{cases} \qquad v \in V \quad t \in N_G \setminus \{r\}$$

$$f_a^t \le w_a \quad t \in N_G \setminus \{r\}, \; a \in A$$

$$f_a^t \ge 0 \quad t \in N_G \setminus \{r\}, \; a \in A$$

$$w_a \in \{0,1\} \quad a \in A$$

Los datos de interés para el modelo son:

r: nodo terminal de salida de flujo del grafo *G*, $r \in V$

t: nodo terminal de entrada de flujo del grafo *G*, $t \in V$

V: conjunto de nodos Steiner y nodos terminales en *G*

N_G: nodos terminales en *G*

$c(a)$: coste del arco *a*

Donde las variables utilizadas significan:

w_a: $\begin{cases} 1 & \text{si a forma parte del árbol de Steiner} \\ 0 & \text{en otro caso} \end{cases}$

$f^t(\delta^+(v))$: se trata del flujo de salida a un nodo *v*

$f^t(\delta^-(v))$: flujo de entrada al nodo *v*

f_a^t : flujo en el arco *a*

El modelo está organizado de la siguiente forma. La función objetivo representa la suma de costes del árbol de Steiner solución. A su vez, la primera restricción muestra el balance de flujo de cada nodo. Para continuar, las dos siguientes restricciones hacen referencia al máximo de flujo que puede llevar un arco, siendo de 1 unidad. Por último, se hace mención a w_a, que es una variable binaria.

Se muestra a continuación un ejemplo de la técnica:

El objetivo del algoritmo es ir cambiando los nodos r y t hasta que se obtenga un árbol de Steiner cuyo camino de r a t sea de coste mínimo.

Un ejemplo se muestra en la Figura 12, donde se comprueba que el balance de flujo en *r* es de 1 y el de *t* de -1, al igual que en el resto de los nodos donde pasa flujo, no se retiene nada. Para este caso en particular se obtiene un coste total de 18 unidades.

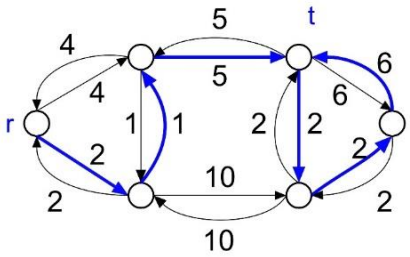


Figura 12. Ejemplo resolución Flow Formulation

Fuente: Elaboración propia

El algoritmo sigue iterando hasta que encuentra la solución mostrada en la Figura 13, donde finalmente el coste total se consigue minimizar a doce unidades.

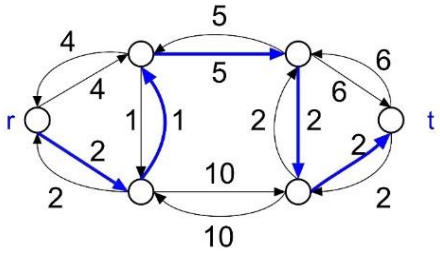


Figura 13. Continuación ejemplo resolución Flow Formulation

Fuente: Elaboración propia

Como se comentaba anteriormente, el mayor problema de esta formulación es el gran número de restricciones, lo que hace imposible la implementación de este algoritmo en grafos de gran escala. Ello también ocurre en los anteriores algoritmos de resolución exacta, donde el tiempo de resolución deja de ser polinomial cuando el número de nodos del grafo aumenta. Es por ello, la necesidad de algoritmos heurísticos que den una solución muy próxima al óptimo y en el mejor de los casos el óptimo.

Más información sobre las técnicas de resolución exactas se encuentran en la referencia a continuación. (Marcus Brazil y Martin Zachariasen, 2010)

3 APLICACIONES DEL ÁRBOL DE STEINER

"La teoría no es la raíz sino la floración de la práctica"

Ernst von Feuchtersleben

3.1 Perspectiva global

En este apartado se muestra la utilidad del árbol de Steiner reflejada en un abanico de aplicaciones de la más diversa índole. Desde el interés por la optimización de caminos a mediados del siglo XVIII que propició el nacimiento de la teoría de grafos, cantidad de aplicaciones han emergido fruto de los avances tecnológicos. A su vez, la necesidad del hombre de abaratar costes o incluso de curar enfermedades también ha jugado un rol muy importante. Aplicaciones en redes neuronales, optimización de rutas aéreas y enrutamiento de cables en circuitos integrados pertenecen a los campos de mayor crecimiento en la actualidad.

La utilidad del problema de Steiner será puesta en práctica con las aplicaciones potenciales que se detallarán en la sección 3.2. Adicionalmente, antes de comenzar con los ejemplos escogidos, se proporciona una visión global más amplia de la envergadura del problema de Steiner en la sociedad actual. De hecho, la gran mayoría de aplicaciones del árbol de Steiner se incluye en la siguiente clasificación presentada por Dietmar Cieslik (2005). Dicha clasificación consta de doce tipos de aplicaciones del árbol de Steiner, las cuales son:

- Redes de grandes regiones:

Los problemas de Steiner en este campo pretenden conectar grandes regiones mediante el camino más corte posible. Para ello, se hace uso del árbol de Steiner, ya que problemas de gran escala necesitan de un algoritmo de optimización. En muchos casos, existe ya una red establecida con cantidad de posibilidades para conectar los puntos de interés. Sin embargo, cuando dichos recorridos son curvos sin salvaguardar la distancia mínima entre dos puntos adyacentes, es decir, unidos por un solo arco, es posible remodelar la red. De esta manera, se podría obtener una red con caminos diferentes a los anteriores con menor distancia. Ubicaciones internacionales centrales o centros de distribución y planificación de tuberías de agua y gas o largas distancias de líneas telefónicas son ejemplos característicos de largas redes regionales.

- Redes regionales:

El diseño de redes interurbanas como redes de comunicación, líneas ferroviarias o carreteras son problemas resolubles con el árbol de Steiner. Mediante su resolución, es posible orientar el diseño de las redes y ajustar óptimamente los costes.

- Superficies mínimas:

Para averiguar el área mínima en la estructura de un tetraedro, se pueden utilizar películas de jabón que permiten minimizar superficies energéticas. Así, sumergiendo una estructura tetraédrica en una solución con jabón, se podrá obtener el área mínima. La misma técnica es posible usar para encontrar el SMT para puntos en el plano euclídeo. En este sentido, hay muchas similitudes entre el problema de Steiner y superficies de mínimo área, que son muy socorridas para abordar estos problemas.

- Localización de instalaciones ("*facility location*")

Los problemas pertenecientes a la teoría de la ubicación ("*location theory"*) permiten posicionar una nueva instalación en un territorio con el objetivo de minimizar costes en el servicio al cliente. De este modo, el territorio es el espacio donde se sitúa la red, los clientes están situados en los puntos y el coste es la suma de las distancias hasta la nueva instalación.

- Ingeniería mecánica

En el campo de la ingeniería mecánica los puntos de Steiner son claves en el desarrollo de sistemas en equilibrio. Si se tiene un conjunto de nodos terminales y se describe el nodo de Steiner, como punto que minimiza las distancias entre los nodos terminales, las fuerzas que se generan desde el punto de Steiner hacia todos los nodos terminales se consideran que están en equilibrio.

- Redes de macro escalas

Plantas de procesamiento químico, sistemas arteriales urbanos, cables de televisión y sistemas intraurbanos similares son aplicaciones típicas. Frecuentemente, las conexiones de estructuras tienen que ser diseñadas en un entorno de una minuciosa estructura interna. En estas situaciones, la métrica rectilínea es usada habitualmente. Si la estructura de las conexiones posibles es predeterminada, es posible formular el problema como uno de diseño de redes en grafos.

- Minas

Los caminos en una mina subterránea pueden ser árboles mínimos de Steiner en un espacio tridimensional. Durante la vida de una mina, el mineral preciado es generalmente transportado desde numerosos puntos de acceso hasta los depósitos del mineral obtenido en la superficie a través de una red de rampas empinadas y ejes verticales.

- Redes de escalas intermedias

Sistemas eléctricos, de calefacción y aire acondicionado son ejemplos de problemas de optimización de redes donde los puntos Steiner pueden reducir el coste global del árbol. Los espacios métricos rectilíneos son los más utilizados en estas aplicaciones.

- Redes de comunicaciones

Durante el último par de décadas los avances tecnológicos han detonado una explosión en el desarrollo de las redes de comunicación. Es una opinión ampliamente aceptada que la eficiencia y estabilidad de las conexiones es necesaria para tener una ventaja competitiva en la sociedad actual. Dichas características son obtenidas mediante la optimización del problema de Steiner.

- Redes de microescalas

Probablemente uno de los ejemplos más prácticos del problema de Steiner sea el diseño de circuitos eléctricos. La creación de redes de integración a gran escala (“*very large scale integration”*, VLSI) es un ejemplo de los problemas de Steiner donde la distancia de conexión total es crucial en la resolución del problema. En esta clase de aplicaciones, los espacios métricos rectilíneos vuelven a ser los más utilizados. El problema comúnmente conocido como el problema rectilíneo de Steiner fue primeramente investigado por Hanan en 1966. Hanan demostró que el árbol de Steiner en el plano formado mediante arcos ortogonales, contrario al caso de las distancias euclídeas, es un caso especial del problema de Steiner en grafos.

- Estructura de las proteínas

Uno de los problemas clave en la bioquímica actual es la predicción de la estructura tridimensional de las proteínas desde la estructura primaria donde se encuentran las uniones más básicas entre los aminoácidos hasta la estructura cuaternaria. El árbol de Steiner en grafos ayuda a entender las posiciones de estas largas cadenas moleculares. Con la intención de examinar esta potente área de aplicación y otras relacionadas, posibles vinculaciones entre la función objetivo del problema de Steiner y la función objetivo de estas aplicaciones en las ciencias bioquímicas necesitan ser examinadas.

En particular, la configuración óptima de cualquier conjunto de puntos en un espacio tridimensional es una cadena infinita de tetraedros que comparten cara, conocidos como triple hélice. Fueron Gilbert y Pollak los que declararon que justamente cuando los nodos forman un politopo regular se alcanza la configuración óptima.

- Redes evolutivas

Las secuencias moleculares son usadas para reconstruir el curso de la evolución. Desde que la evolución de un conjunto de especies se ha asumido que procede de un predecesor común, se modela la evolución de especies en forma de árbol. La cuestión clave resulta de reconstruir el árbol basado en datos pasados. Los datos se pueden recoger en forma de:

- Secuencias de ADN, que son moléculas con información contenida formada por nucleótidos.
- Proteínas, las cuales son moléculas operacionales compuestas de secuencias de aminoácidos.
- Secuencias de RNA, que es una mezcla de las dos anteriores formadas también por nucleótidos.

En la clasificación de Dietmar Cieslik no están incluidas algunas de las aplicaciones más innovadoras explicadas en la sección 3.2, entre ellas, las redes neuronales o la optimización de rutas aéreas. A pesar de ello, la clasificación nombrada hace una introducción a la versatilidad del problema de Steiner.

3.2 Ejemplos de aplicaciones

En la sección 3.2, se van a ejemplar trece problemas actuales resueltos con el problema de Steiner, los cuales se encuentran en siete campos completamente diferentes. Dichos campos son; infraestructuras, redes de telecomunicaciones, diseño de circuitos integrados, redes biológicas, marketing, control aéreo y finanzas. En la Tabla 1 **¡Error! No se encuentra el origen de la referencia.**, se propone a modo de esquema visual los campos elegidos en los que se aplicará el árbol de Steiner. Las aplicaciones escogidas se ven clasificadas en la **¡Error! No se encuentra el o rigen de la referencia.** en función del campo, subcampo, aplicación y finalmente la fuente de donde proviene la información.

Tabla 1. Aplicaciones y campos elegidos

Campo	**Subcampo**	**Aplicación**	**Fuente**
Infraestructuras		Redes ferroviarias	Tren 2020: Propuesta ferroviaria para una nueva realidad
		Redes eléctricas	Daniel O. Anaut et. al (2009)
		Redes de minas subterráneas	Markus Brazil et. Al (2015)
Redes de telecomunicaciones		Redes de fibra óptica	Jaime Prieto Zapardiel (2014)
	Red inalámbrica	Eficiencia energética en Wireless Network	Panos M. Pardalos et. al (2013)
		Enrutamiento Multicast (Multicast Routing)	Xiuzhen Cheng and Ding-Zhu Du (2001)
Diseño de circuitos integrados		VLSI (very large scale integration)	Xiuzhen Cheng and Ding-Zhu Du (2001)
Redes biológicas		Estructura molecular	Rubem P. Mondaini (2007)
		Redes neuronales	Panos M. Pardalos et. al (2013)
		Árboles filogénicos	Xiuzhen Cheng and Ding-Zhu Du (2001)

Marketing		Marketing viral	Panos M. Pardalos et. al (2013)
Control aéreo		Rutas aéreas	Narges Norouzi et. al (2016)
Finanzas		Redes financieras	Seong Eun et. al. (2015).

Finalmente, se comienza con la exposición de las aplicaciones más relevantes del problema de Steiner.

3.2.1 Infraestructuras:

La optimización de caminos en el ámbito de la ingeniería civil es la aplicación por excelencia del algoritmo de Steiner. A pesar de haber ganado terreno en diversos campos, el árbol de Steiner sigue siendo primordial en la conexión óptima de ciudades, regiones, países, continentes e incluso en el sector privado para reducir la inversión económica de proyectos y conseguir otros objetivos como el mejor rendimiento posible. En la actualidad, las redes más interesantes en la optimización de recursos son; redes de tuberías de agua o gas, redes eléctricas o carreteras, vías ferroviarias e incluso caminos en minería.

Entre las aplicaciones nombradas dentro del campo de las infraestructuras, se pasa a la explicación de tres ejemplos; redes ferroviarias, redes eléctricas y redes de minas subterráneas que sirven de referencia para poder construir las redes típicas de este sector.

3.2.1.1 Redes ferroviarias:

Se pretende poder conectar mediante transporte ferroviario de alta velocidad las principales ciudades españolas más demandadas. Para ello, se exponen en un plano las diferentes alternativas viables que conectan las ciudades de interés más algunos puntos intermedios (puntos Steiner) que convenga elegir por situación geográfica. A la misma vez, a cada camino que conecte dos ubicaciones, se le asigna un coste que puede ser función de la inversión en ese tramo y de la distancia del arco.

Una vez realizado el diseño de la red, se aplica el árbol de Steiner y se obtiene la solución de la Figura 14, la cual fue elegida por una de las compañías ferroviarias más importantes del país.

Figura 14. Mapa España con solución del SMT

Fuente: Tren 2020. Propuesta Ferroviaria para una nueva realidad

3.2.1.2 Redes eléctricas:

Las redes eléctricas encargadas de distribuir la electricidad se pueden modelar mediante un árbol de Steiner teniendo en consideración una serie de restricciones. Dentro de los aspectos a tener en cuenta en la explotación de una red de distribución de energía eléctrica se presenta:

1. La reducción de las pérdidas por efecto Joule (pérdidas técnicas) resulta ser de vital importancia para mejorar la eficiencia en la prestación del servicio.
2. Las redes de distribución eléctrica deben estar configuradas, en su mayoría, en forma radial para una fácil operación y conveniente coordinación de las protecciones.
3. Se deben satisfacer todos los requerimientos de carga.

Estos requisitos conducen a un complejo problema de optimización que, planteados en términos matemáticos, resulta ser NP-Completo. La solución óptima exacta puede ser obtenida al examinar todas las posibles combinaciones de caminos que transporten la electricidad, demandando su resolución un excesivo tiempo de operación computacional.

Por ello, se utilizan en este sector, entre otros, algoritmos genéticos aplicados al problema de Steiner para obtener una solución admisible buena en un tiempo computacional razonable.

3.2.1.3 Redes de minas subterráneas:

La minería es una industria importante mundialmente practicada. La reducción de los costes en la extracción de minerales preciados es una cuestión clave para las compañías mineras. La competitividad del mercado minero hace que cada euro marque la diferencia. Por ello, se han desarrollado métodos para el modelado y la optimización de minas a cielo abierto. Algoritmos como los de Lerchs y Grossmann (1965) revolucionaron el diseño de estas minas permitiendo a

los ingenieros y planificadores la manipulación de datos asociados con un diseño óptimo. Respecto a las minas subterráneas, no existían algoritmos semejantes aplicados bajo tierra. Sin embargo, un grupo de investigadores de la universidad de Melbourne lograron encontrar un algoritmo de resolución para las minas subterráneas. Restricciones como el ancho de túnel para que puedan pasar los dispositivos extractores y los metros de túnel son parámetros indispensables para obtener los caminos óptimos por donde extraer el mineral.

Una mina subterránea consiste en una serie de túneles interconectados, túneles que llevan hasta el yacimiento del mineral ("*ore body*" o "*stopes*", en inglés) y túneles que se utilizan para la extracción a la superficie. En las minas, el yacimiento del material preciado es identificado mediante test geológicos o por perforación de pozos de relleno. Con esta información los ingenieros mineros determinan los puntos terminales donde está localizado el yacimiento del mineral.

El material es finalmente extraído mediante uno de los posibles métodos de extracción (como "stoping", "caving" o "room and pillar") y transportado a la superficie mediante la red de acceso, principalmente compuesta por "crosscuts" y "declines" tal como se muestra en la Figura 15.

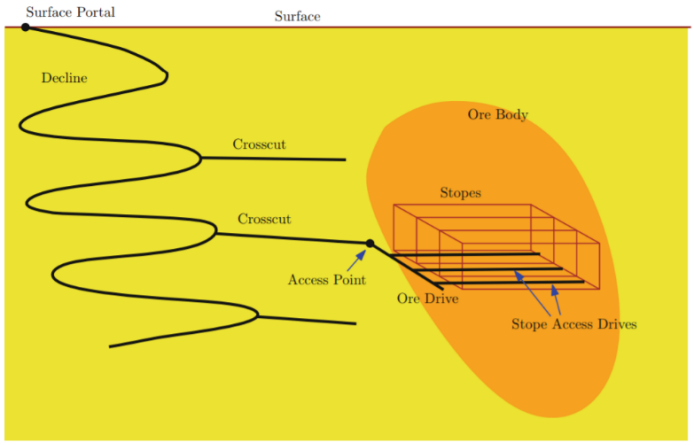


Figura 15. Esquema de mina subterránea

Fuente: Markus Brazil, Martin Zachariasen (2015)

Por tanto, se define el problema de Steiner con *e* túneles donde $e \in E$ y *T* siendo un grafo, tal que $T = (N, E)$ donde *N* representa el número de yacimientos a excavar. Finalmente, se muestran las restricciones y la función objetivo a tener en cuenta para la resolución del problema de optimización.

Los túneles deben ser elegidos cumpliendo con las siguientes restricciones:

1. No se pueden excavar regiones atravesando un yacimiento o donde previamente se ha excavado.
2. La inclinación de los túneles no debe traspasar un máximo para que las máquinas puedan rodar por los caminos.
3. Los caminos de túnel deben de ser lo más cortos posibles para minimizar los costes.

Seguidamente, la función objetivo viene definida por:

$$C(T) = \sum_{e \in E} (d + h \times t_e) \times l_e$$

Donde:

- d: el coste de túnel por metro
- h: el coste del túnel por tonelada a extraer y kilómetro a excavar
- t_e: la cantidad de mineral a transportar por el túnel *e*
- l_e: la distancia del túnel *e*

3.2.2 Redes de telecomunicaciones

En el último siglo, las telecomunicaciones han tenido un gran impacto en todos los aspectos de la vida. No cabe duda de que la transformación de la revolución industrial a la era de la información ha sido principalmente influenciada por los avances tecnológicos en el campo de las telecomunicaciones.

La minimización de costes, de energía o encontrar la conexión óptima son ejemplos de objetivos primordiales en los problemas de optimización que también se utilizan en este campo. Mediante el problema de Steiner aplicado a redes de fibra óptica y redes inalámbricas como el enrutamiento multicast, se podrá ver el interés del problema de Steiner en esta rama de la ingeniería.

3.2.2.1 Redes de fibra óptica:

La fibra óptica se utiliza para la distribución de servicios avanzados como televisión, internet de banda ancha y telefonía. Se trata de una red de gran ancho donde el haz de luz del emisor se distribuye hacia múltiples fibras siguiendo diferentes direcciones. Su complejidad depende de la extensión, la cobertura de esta y, evidentemente, del ancho de banda. Regiones donde hay que hacer llegar menos información, permiten disminuir la cantidad de fibra óptica y los costes de instalación y mantenimiento.

La fibra óptica tiene mucho que ofrecer a las telecomunicaciones y cada vez es más frecuente encontrarlas, ya que soportan grandes cantidades de datos abaratando el coste respecto de las instalaciones de cobre. Por otro lado, múltiples ventajas como; bandas de frecuencia elevadas, inmunidad frente interferencias electromagnéticas, vida media de los cables superior a los cables de conductores y bajo coste potencial (debido a la abundancia del material empleado, óxido de silicio) hacen que las crecientes necesidades de servicios de telecomunicaciones se vean satisfechas por esta tecnología.

Por ello, el problema de Steiner tiene un papel crucial para la planificación de dichas instalaciones. En la Figura 16 se muestra un ejemplo de una empresa consultora estadounidense que pretende unir veintiuna ciudades norteamericanas consiguiendo disminuir las distancias entre puntos por el árbol de expansión mínima (*mínimum spanning tree, MST*) utilizando el árbol mínimo de Steiner (*Steiner minimal tree, SMT*). En la primera imagen, tenemos el problema inicial con todos los posibles caminos entre las diversas ciudades. Seguidamente, se muestra la resolución del árbol de expansión mínima mediante el algoritmo de Kruskal. Por último, se introducen puntos

Steiner según el principio del punto de Torricelli, donde el árbol mínimo euclídeo de Steiner visto en la sección 2.3.1 permite disminuir las distancias respecto al MST.

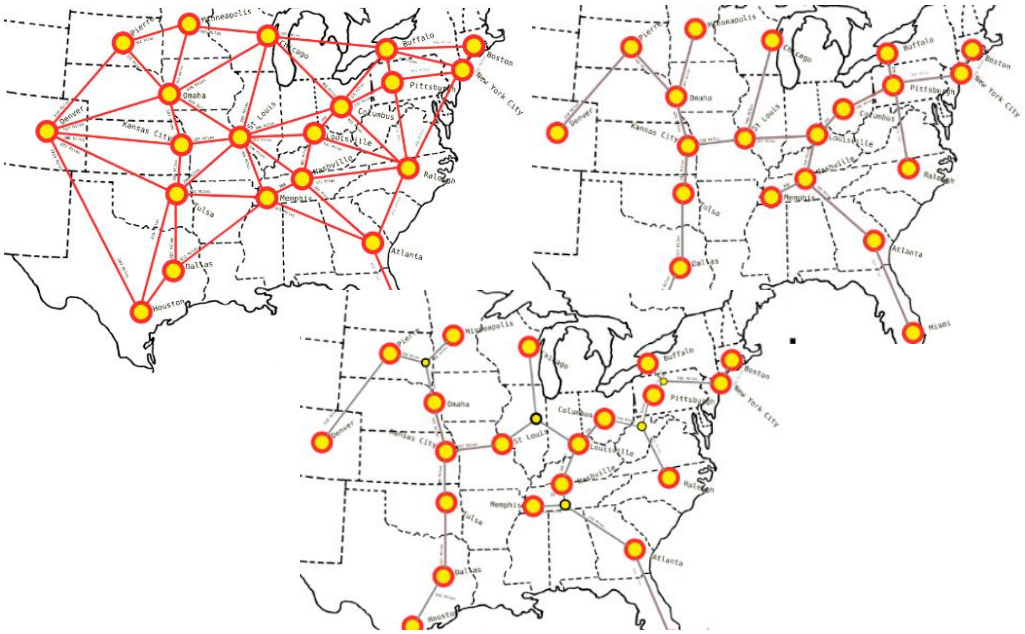


Figura 16. Procedimiento SMT para ciudades estadounidenses

Fuente: Noah Todd (2013). Kruskal Steiner Fiber Optic Cable Project. Prezi

3.2.2.2 Red inalámbrica (Wireless network):

En el campo de las telecomunicaciones, la red inalámbrica trata las conexiones de nodos que se da por medio de ondas electromagnéticas sin necesidad de una red cableada. Cada nodo es un emisor y/o receptor que suelen ser antenas, móviles o algún dispositivo destinado a emitir o recibir señales. A diferencia de las redes alámbricas conectadas físicamente, mediante la conexión por radiofrecuencia se consiguen abaratar los costes de cableado, pero hay que incluir una mayor seguridad contra posibles intrusos.

Al igual que las conexiones de telecomunicación por cable, es imprescindible encontrar el mejor camino de un emisor a un receptor a través de las conexiones establecidas entre los nodos. En las conexiones mediante señales, el objetivo es minimizar el retraso de la señal aumentando el rendimiento de la conexión. Hay que tener en cuenta que existe una frecuencia máxima de transmisión de datos, lo que resulta un impedimento para transportar grandes cantidades de información ya que la capacidad de una señal para transmitir datos aumenta con la frecuencia.

A continuación, se presentan dos buenos ejemplos del problema de Steiner aplicados a la eficiencia energética en "*Wireless Networks"* y al enrutamiento multicast.

3.2.2.2.1 Eficiencia energética en "*Wireless Networks"*:

Algunos dispositivos que forman parte de una red inalámbrica son actuadores, sensores u otros aparatos de pequeño tamaño que presentan un límite de batería que hay que optimizar.

Problemas en particular que se resuelven con una red de sensores que están interconectados son; seguridad nacional, vigilancia, atención médica y monitoreo ambiental entre otros.

Una vez que se tienen los puntos donde se colocan los dispositivos que emiten y reciben ondas, es imprescindible minimizar la distancia de la señal que emite cada nodo para reducir el consumo de energía. Así se pretende crear un camino que una a todos los nodos minimizando la distancia de la señal lo mínimo posible para gestionar óptimamente el recurso energético. Para ello, se utiliza la siguiente igualdad que relaciona la distancia entre nodos con la energía de la señal.

$$P_s \geq d(s,t)^{\alpha}$$

Donde:

- P: energía de la señal emitida por s
- d(s,t): distancia entre nodos s y t
- α: constante entre 2 y 5

Se dispone de un ejemplo en la Figura 17 donde cada nodo terminal representa un sensor. Para su implementación se definen; los puntos Steiner, que son sensores que pueden o no ser añadidos para el mejor funcionamiento de la red, y los costes energéticos para unir los sensores que se toman de cada arco. Aplicando el algoritmo desarrollado en este proyecto se obtiene la solución marcada en negrita que representa un coste total de 37 unidades energéticas.

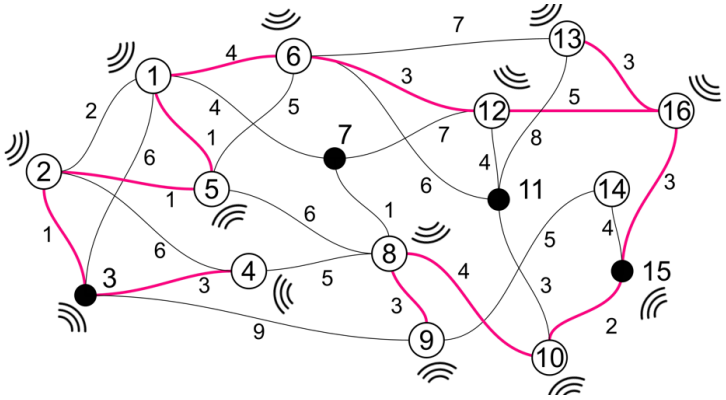


Figura 17. Ejemplo red de sensores Wi-Fi

Fuente: Creación propia

Aparte, existen otro tipo de problemas de optimización combinatoria en los que no es posible ajustar la distancia de la señal. Por eso, es necesario combinar sensores en modo activo y desactivo para optimizar una malla de excesivos sensores.

3.2.2.2.2 Enrutamiento multicast (Multicast routing):

El concepto *multicast* abarca la habilidad de conectar un subconjunto de receptores a una red. Este servicio *multicas*t es ofrecido por el "*packet switched network" (PSN)* que se encarga de suministrar copias de un paquete de información a un conjunto de destinatarios simultáneamente. Básicamente, la idea consiste en poder transmitir archivos multimedia como audios o videos mediante la conexión de servidores en una red inalámbrica de ordenadores. La

comunicación *multicast* viene siendo de gran importancia para poder llevar a cabo teleconferencias, colaboración remota o el aprendizaje a distancia como las universidades a distancia. Por consiguiente, debido a la necesidad de conectar varias fuentes a varios destinatarios o simplemente una fuente a varios receptores, el enrutamiento multicast ha ido cobrando importancia con el paso de los años.

Para dar soporte multicast es necesario cumplir algunos requisitos de calidad de servicio (*quality of services,* QoS), entre los más utilizados el *end-to-end delay*, que se trata de poner un límite al retraso de señal total añadido en los caminos desde el emisor hasta el receptor final. Mediante esta restricción se asegura minimizar el retraso de la señal aumentando el rendimiento de la conexión, tal como se explicó en el apartado *Wireless Network*.

En la Figura 18 se muestra un ejemplo donde se quieren conectar los nodos $\{v_4, v_5, v_7, v_8\}$ con la fuente v_1. Las restricciones que se presentan son minimizar el coste total y cumplir que el retraso esté por debajo de 20 unidades. Se puede observar cómo es imposible crear un único camino en el grafo desde v_1 a v_8 debido a que se debe cumplir con el requisito de limitar el retraso a 20 unidades como máximo. Luego, aunque los costes sean mayores para la solución representada en la figura, se deben coger dos caminos desde v_1. Los nodos v_6 y v_2 serían nodos Steiner en dicha solución.

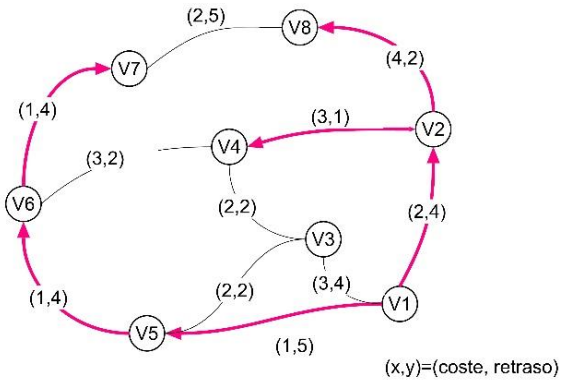


Figura 18. Ejemplo red de enrutamiento multicast

Fuente: Creación propia

Por supuesto, este tipo de problemas es resuelto mediante el árbol de Steiner. Se pueden distinguir entre dos tipos de problemas; los estáticos y los dinámicos de enrutamiento multicast (*"Multicast Routing", MR*). La característica principal de los problemas estáticos es la existencia de un solo problema de optimización. Al ser conocidos tanto la fuente de información como el número y la posición de los miembros del grupo multicast (receptores), se optimiza la red antes de que se ponga en funcionamiento. Por el contrario, la resolución dinámica deberá tener en cuenta que los receptores pueden unirse o dejar el grupo multicast durante el periodo de conexión, lo que supone una dificultad añadida al tener que estar optimizando la conexión cada *x* tiempo. Los problemas dinámicos, también conocidos como "*on-line multicast problema",* podrían resolverse implementando algoritmos de problemas estáticos cada vez que cambia el número y posición de los receptores. El mayor inconveniente de ello, son los costes prohibitivos que supone la optimización constante de la red. Además, los sistemas reales no toleran grandes cambios en los árboles multicast debido a que los paquetes de datos están constantemente

viajando por la red. Un cambio brusco de red supondría la disrupción del flujo de información ocasionando el colapso.

Algoritmos como *the "greedy algorithm" (GA)*, "*the predetermined path search algorithm" (PPS)* o "*the ARIES algorithm"* son los más adecuados para resolver el árbol de Steiner en problemas dinámicos de enrutamiento multicast, los cuales escapan del objetivo de este proyecto. Mediante esta aplicación se muestra otro motivo más que justifica el interés mostrado en las últimas décadas por encontrar un algoritmo resoluble en tiempo polinomial del problema planteado por Steiner.

3.2.3 VLSI (very large scale integration):

Se trata del proceso de creación de circuitos integrados combinando miles de módulos (transistores en su mayoría) dentro de un único chip. Su rápido desarrollo tecnológico y el crecimiento de la complejidad de los VLSI, debido a que cada vez más se necesitan microprocesadores más pequeños y potentes con mayor número de transistores, hacen que la aplicación del problema de Steiner sea crucial para unir los diferentes componentes (enrutamiento) de forma óptima.

El problema del enrutamiento es solventado mediante algoritmos de optimización como el árbol de Steiner rectilíneo (visto en la sección 2.3.2) que es utilizado para conectar el conjunto de módulos dentro del microprocesador usando la mínima distancia posible. Llegados a este punto, el diseño general del circuito ya ha sido determinado. Tanto la posición de cada componente en el chip, como los posibles caminos entre los transistores son conocidos para poder comenzar con el problema de enrutamiento. Para ello, se compaginan en el diseño del microprocesador eficientes algoritmos con un proceso de diseño altamente automatizado.

Un objetivo clave para diseñar buenos VLSI es utilizar la mínima separación posible entre los cables que se necesitan implantar. De este modo, al ser el área ocupada por los cables aproximadamente proporcional a la longitud total de cable utilizado, también se podrá minimizar el área. Luego, esto evita el retraso de señal y el consumo de energía, y favorece el rendimiento del sistema.

Tradicionalmente se ha venido considerando que los cables están dispuestos en dos direcciones ortogonales cada una de ellas en una capa diferente y que los circuitos se pueden modelar efectivamente como redes planas en una geometría rectilínea o Manhattan. En cambio, en los modernos diseños microscópicos de VLSI son otros criterios los que dominan el enrutamiento de pines, dando prioridad a otros modelos más efectivos para dichos diseños. Estos criterios son la distancia, la densidad, la inductancia, el ruido, la energía, la tridimensionalidad o la urgencia con la que tiene que pasar la información por un tramo de cable (zonas de la ruta críticas).

Los modelos alternativos actuales para el enrutamiento óptimo de componentes que reciben más atención son las mallas hexagonales y las mallas octo-cuadradas. En el modelo hexagonal, el cable puede estar orientado en tres direcciones con pendientes por ejemplo de 0 a $\pm\Pi/3$. La eficiencia de este modelo fue descubierta por primera vez en 1979. En el modelo de enrutamiento de las mallas octo-cuadradas, los caminos horizontales y verticales utilizados en el modelo del árbol mínimo de Steiner rectilíneo son complementados con caminos con pendientes de $\pm\Pi/4$.

Debido a su sencillez en comparación con otros modelos, las redes con direcciones ortogonales siguen siendo de gran relevancia en el campo de la electrónica. Lo que demuestra la utilidad del problema de Steiner. En la Figura 20, se muestra una foto real tomada desde un microscopio eléctrico de un chip con la estructura de pines diseña a partir del modelo rectilíneo de Steiner. En la Figura 19, se aprecia la conexión de los pines según una solución admisible del problema.

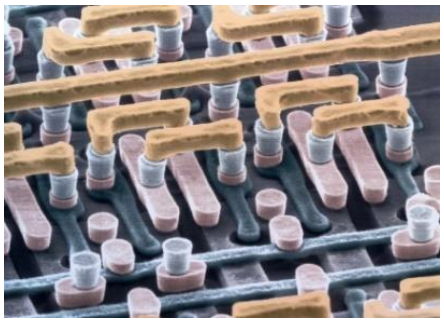


Figura 20. Foto real de un microship

Fuente: Shortest Paths and Steiner Trees in VLSI Routing, Steven Peyer (2007)

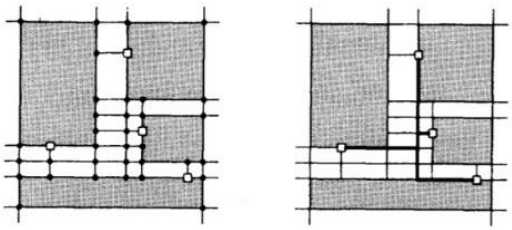


Figura 19. Solución de los pines

Fuente: Class Steiner trees and VLSI-design, Edmund Ihler (1995)

3.2.4 Redes biológicas

En el campo de la biología, se pueden determinar ciertos enigmas de la naturaleza a base del problema de Steiner. En esta sección se van a tratar temas relacionados con el cuerpo humano que presentan una gran similitud con el árbol de Steiner al igual que los árboles filogénicos que resuelven incertidumbres sobre los antecesores de las especies. Respecto al cuerpo humano, se ha descubierto recientemente que la minimización de energía potencial de la estructura molecular de las proteínas puede resolverse con el árbol de Steiner. Por otro lado, las interconexiones neuronales en el cerebro humano también siguen un patrón que se puede modelar con el problema de Steiner.

Aunque se trata de uno de los campos con mayor investigación en el futuro, hoy es posible obtener resultados asombrosos a partir de los algoritmos de optimización.

3.2.4.1 Estructura molecular

La bioquímica es un campo que cada vez más se ve integrado a la ingeniería. Se ha descubierto que la estructura molecular de las proteínas, así como de los ácidos nucleicos que forman el ADN y los agregados moleculares como el virus del mosaico del tabaco es la misma que el modelo de puntos de un árbol mínimo de Steiner. Parece ser, que la naturaleza utiliza principios matemáticos para minimizar la energía local y así formar una estructura que busca la estabilidad durante etapas de crecimiento molecular.

La condición para posicionar los átomos en la malla molecular es minimizar la energía potencial consiguiendo así una red estable con un consumo energético bajo. Este descubrimiento permite a los científicos avanzar en la comprensión de la formación de biomoléculas y su evolución.

En la Figura 21 se muestra la estructura de un aminoácido.

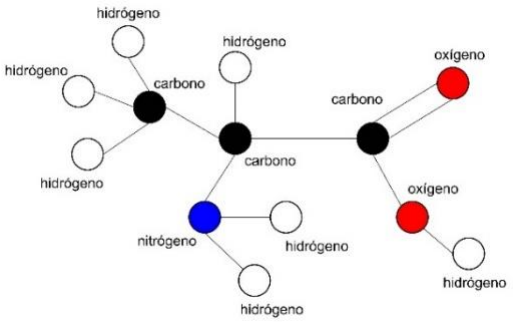


Figura 21. Estructura aminoácido

Fuente: Elaboración propia

3.2.4.2 Redes neuronales:

La teoría de grafos aplicada al problema de Steiner también se puede utilizar para el modelaje de las conexiones neuronales que ocurren en el cerebro. El entendimiento de dichas interacciones abre todo un campo de investigación a la cura de enfermedades.

El análisis de los patrones de conectividad de las funciones del cerebro es extremadamente complejo debido a la gran cantidad de neuronas y a las conexiones dinámicas entre ellas. Se estima que el cerebro humano tiene $8.3\ x\ 10^9$ neuronas y que el número de conexiones es aproximadamente $6.6\ x\ 10^{13}$. Por ello, la representación de dichas estructuras como un grafo de neuronas (posicionada cada una de ellas en un nodo), presenta un coste computacional prohibitivo.

Las técnicas utilizadas habitualmente van orientadas a la agrupación de neuronas en nodos dependiendo de su funcionalidad. De esta manera, se consigue abarcar todo el cerebro en un problema simplificado de igual valía y mejor manejo. Estudios como el sistema visual cortical de un macaco o las propiedades del cerebro humano son investigaciones llevadas a cabo mediante el árbol de mínima expansión, entre otras técnicas. Para poder unir los arcos entre nodos se estipula un valor de correlación entre funciones del cerebro que hay que superar para determinar que dos nodos están enlazados.

Aunque se trata sin duda de una de las aplicaciones más desconocidas, estudiando las conexiones neuronales es posible diagnosticar y tratar trastornos psiquiátricos (entre otras enfermedades) y se pretende conseguir una visión de los procesos de propagación de señales entre las unidades funcionales del cerebro y las neuronas.

3.2.4.3 Árboles filogénicos:

El problema del árbol de Steiner ha tenido gran relevancia en el ámbito biológico. Desde hace más de 100 años, los biólogos interesados en “sistemática” han intentado deducir los árboles evolutivos que presentan las especies de hoy. En los últimos 30 años, es cuando han sido investigados algoritmos matemáticos para construir árboles, motivados por el avance

tecnológico y por las diversas razones que fomentan su investigación. El motivo más habitual es el interés por la topología que responde preguntas de clasificación básica. Otras veces, se pretende saber cuándo se ha producido la divergencia entre dos especies o cómo de largos son los arcos del árbol.

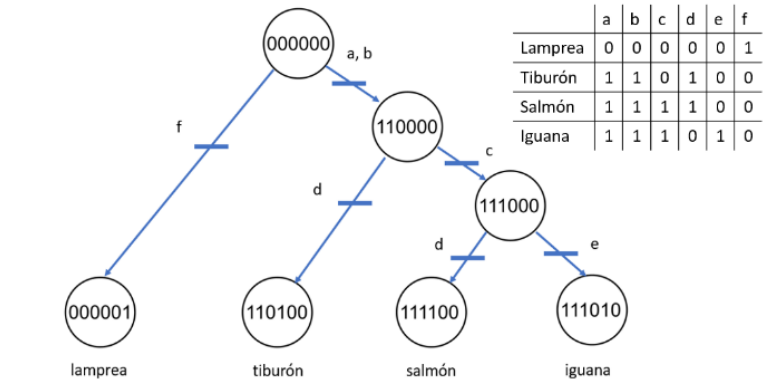
Estas cuestiones pueden ser solventadas mediante el problema del árbol de Steiner. Los datos iniciales son un conjunto de especies, más generalmente llamado taxa, que se presenta adicionalmente con información sobre las relaciones existentes entre cada taxón (cada grupo de especies). Las agrupaciones de especies se presentan en las extremidades del árbol asignándoles nodos terminales. Por otro lado, los nodos internos son referidos a la taxa antecesora y son los llamados nodos Steiner. La solución que se busca es el árbol que mejor ajusta la evolución de la taxa, siendo una variable de interés la longitud de las ramas del árbol. Por ello, el problema de Steiner para árboles filogénicos se define como:

Definiendo a cada carácter como c y al conjunto de caracteres como C, tal que $c \in C$, a cada taxón como s y a la taxa como S, tal que $s \in S$ y a cada vector de caracteres perteneciente a un taxón como v_{cs}. Si se tienen dos vectores v_{ci} y v_{cj} tal que $i, j \in S: i \neq j$, la distancia entre ellos, $dist(v_{ci}, v_{cj})$, viene definida por el número de caracteres diferentes que presentan ambos vectores.

Se pretende obtener:

$$\min \sum_{j \in S} \sum_{\substack{i \in S \\ i \neq j}} dist(v_{ci}, v_{cj})$$

En la Figura 22 se muestra la solución a un ejemplo donde se pueden ver los taxones (lamprea, tiburón, salmón e iguana) y el conjunto de caracteres que pueden poseer (aletas pélvicas, mandíbula, membrana ósea, aleta pectoral, pulmones y lengua áspera). Cuando un carácter presenta el valor de 1 significa que el taxón posee esa característica. Cuando el carácter es 0, la característica no pertenece al animal.



	a	b	c	d	e	f
Lamprea	0	0	0	0	0	1
Tiburón	1	1	0	1	0	0
Salmón	1	1	1	1	0	0
Iguana	1	1	1	0	1	0

a	b	c	d	e	f
Aletas pélvicas	Mandíbula	Membrana ósea	Aleta pectoral	Pulmones	Lengua áspera

Figura 22. Ejemplo árbol filogénico

Fuente: Elaboración propia

Para la parametrización del modelo del ejemplo se han utilizado; nodos terminales, que son cada una de las especies representadas, nodos Steiner, que son las especies antepasadas ya extinguidas de las que procede cada taxón y como coste del arco la distancia entre los vectores de caracteres, la cual hay que minimizar.

Aunque hay que decir, que la complejidad de este problema es NP-Completo (no se puede resolver en tiempo polinomial), el mayor desafío de este campo es condicionar los parámetros iniciales para poder empezar a resolver el problema. Se han descrito más de dos millones de especies de plantas y animales, pero se sabe que falta por descubrir hasta diez veces este número. Luego, se trata de un problema primordial para investigar la diversidad de organismos vivos en el planeta.

3.2.5 Marketing viral

El marketing viral consiste en transmitir una nueva idea o comportamiento al mayor número de personas en el menor tiempo posible. Las redes sociales son las aliadas perfectas para propagar un mensaje de forma exponencial. Por tanto, la idea consiste en que en un tiempo limitado un mensaje se difunda por toda la sociedad a través de personas. Las preguntas clásicas para obtener estos resultados son: a cuántas personas inicialmente se necesita transmitir la idea y cómo tienen que estar dispuestas para difundir el mensaje rápidamente.

Este problema se soluciona mediante la resolución del problema de Steiner. Los nodos serán las personas que van a propagar el mensaje y los arcos las distancias entre personas que se definen en función de la amistad, cercanía del lugar o de un interés común. Una regla importante en este tipo de grafos es la transitividad de la red, donde se comprueba que dos nodos que son vecinos de un tercero tienen gran probabilidad de estar relacionados entre sí y por tanto más probabilidad de que fluya la información entre ellos.

Supongamos que se quiere lanzar un nuevo producto anti-edad y se pretende que lo conozcan el mayor número de personas lo más rápido posible. Se dispone de clientas por ciudades representada cada clienta en un nodo. Decir que para simplificar el problema se han omitido los nodos Steiner que serían otras personas que no son clientes pero que son potenciales compradores del producto. Como se puede ver, la Figura 23 muestra que las clientas que viven en una misma ciudad tienen menor distancia entre ellas porque hay más posibilidades de que se difunda la idea. Por otro lado, algunos nodos conectan con otras ciudades ya que las clientas son amigas de otras clientas fuera de su ciudad formándose la red de contactos que se muestra en la figura. Al resolver el MST, se obtiene la solución marcada en color que indica la óptima transmisión de información. El MST muestra a quién hay que convencerle de la crema, pues esos nodos son los que mejor conectan con más personas. Por tanto, el trabajo del vendedor de cremas es hacer llegar a los cinco nodos de color la información, ya que son “los puentes” entre las distintas ciudades que conectan con un mayor número de personas.

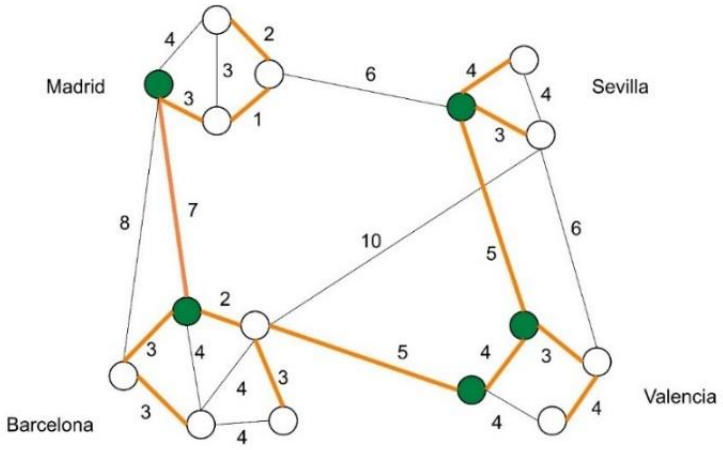


Figura 23. Red de personas utilizadas en el marketing viral

Fuente: Elaboración propia

Aunque pueda parecer sencillo a simple vista y evocar a la inutilidad del algoritmo para resolver este tipo de problemas, este método está diseñado para representar redes de gran tamaño. El ejemplo de arriba solo es una muestra para el entendimiento del problema. A pesar de ello, se puede observar cómo es posible llegar a más de 16 personas (suma total de) solamente contactando con 5, lo cual abarata en gran medida los costes de marketing y de ventas. El marketing viral, es un fenómeno que cada vez se le da más uso y se desarrollan algoritmos para la mejora de su funcionamiento.

3.2.6 Rutas aéreas

Con la implementación de las nuevas tecnologías, los modelos de control de vuelo están siendo sustituidos por modelos de optimización de rutas que permiten minimizar el tiempo de vuelo y por ende el consumo de combustible. El control de vuelo pasa a un segundo plano operativo donde la estrategia se ve guiada por la optimización de la ruta aérea.

Si se obtienen las velocidades del viento en diferentes puntos del globo terrestre, se puede determinar en qué lugares sopla el viento a favor para que una nave vuele más rápido. La velocidad y dirección del viento en una ubicación determinada, será un dato clave para poder estimar la ruta óptima entre dos puntos. Por otro lado, la distancia del lugar donde está el avión hasta el destino será también crucial para poder parametrizar el modelo de optimización.

El problema de Steiner permite modelar en un grafo el origen y destino de la ruta como puntos terminales y los diferentes lugares por donde se puede pasar en la ruta, como puntos Steiner. De este modo, se consigue una malla parametrizada de posibles lugares donde el avión puede viajar para interconectar un origen y un destino.

Los parámetros a tener en cuenta son:

- Distancia: Se trata de un parámetro dependiente de la distancia desde la arista en cuestión respecto del destino a donde se pretende viajar y de la longitud en sí del arco.
- Velocidad máxima: Es la velocidad que alcanza el avión para un determinado arco

Mediante la minimización de la distancia y la maximización de la velocidad máxima, en el modelo de Steiner se consiguen modelar problemas de optimización de rutas para trayectos de larga distancia. Para poder trabajar con una función objetivo donde a la vez se necesita minimizar y maximizar, es recomendable utilizar una función objetivo como esta:

$$\min \frac{distancia\ total}{velocidad\ máxima\ total}$$

En la Figura 24 se muestra un ejemplo del problema planteado para un destino de larga distancia como Múnich-Sídney donde es posible minimizar el tiempo de vuelo en dos horas.

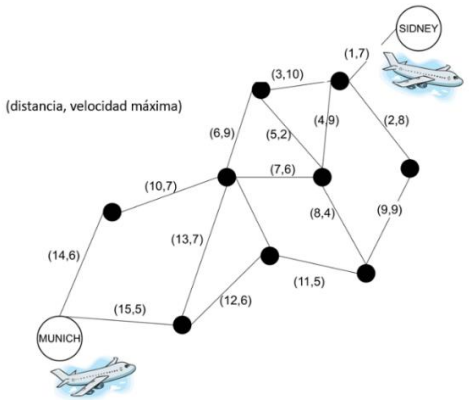


Figura 24. Ejemplo vuelo Múnich-Sídney

Fuente: Elaboración propia

Aunque la tecnología va dando pasos de gigante, en el sector aéreo es ahora cuando se está comenzando a realizar este tipo de cálculos para optimizar el consumo de combustible.

3.2.7 Redes financieras

La diversificación de una cartera de acciones es una de las estrategias más usadas por los inversores para reducir el riesgo sobre un conjunto de valores. Ello conlleva elegir invertir en empresas donde las acciones no estén correlacionadas para que el recorrido del precio de los valores a lo largo del tiempo sea distinto y poder amortiguar las bajadas del mercado.

Mediante el MST (*minimum spanning tree*) se pueden averiguar qué acciones no están relacionadas para poder organizar un porfolio diversificado. Para ello, se define una red compleja donde cada nodo representa una acción que estará unida a otra en función del coeficiente de correlación sobre los beneficios normalizados logarítmicamente del precio del valor.

En esta ecuación, se define el beneficio normalizado de la acción i en un periodo de tiempo de $\Delta t = 1$

$$r_i(t) = [lnP_i(t) - lnP_i(t-1)]/\sigma_i$$

La correlación entre dos acciones i y j sería:

$$C_{ij} = E\left[r_i(t) * r_j(t)\right] - E[r_i(t)] * E\left[r_j(t)\right]$$

Mediante la correlación se puede obtener la distancia entre dos acciones d_{ij} donde $d \in [0, 2]$ siendo 0 cuando las acciones están perfectamente correlacionadas y 2 cuando no están nada correlacionadas.

$$d_{ij} = \sqrt{2(1 - C_{ij})}$$

De esta forma, se calcula el MST minimizando la distancia entre acciones. Aquí se muestra un ejemplo en la Figura 25 donde se ha omitido el nombre de las acciones de los nodos para simplificar la imagen. Como se puede observar, el camino en azul es la solución que conecta todos los nodos terminales (no existen nodos Steiner) mediante la mínima distancia posible. Los nodos coloreados son un ejemplo de conjunto de valores que podrían seleccionarse para formar parte de una cartera. Esto es debido a que no están conectados por arcos y están lejos los unos de los otros dentro de la solución del MST.

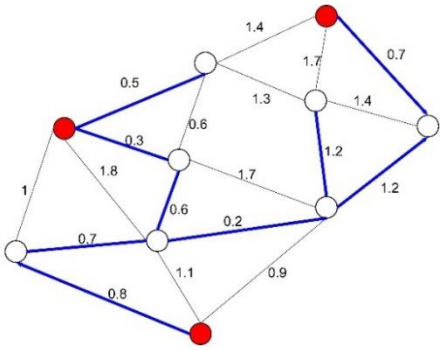


Figura 25. Ejemplo acciones posibles de ser incluidas en una cartera de inversión

Fuente: Elaboración propia

Mediante el *mínimum spanning tree* se han realizado estudios sobre como la crisis del 2008 en la bolsa de valores ha originado mayor correlación en los precios de las acciones. Algo que es evidente debido a la brusca bajada general de los precios, pero que se puede analizar viendo como los gráficos han evolucionado antes, durante y después de la crisis. En la Figura 26 se aprecian estos cambios para el mercado de valores de Corea.

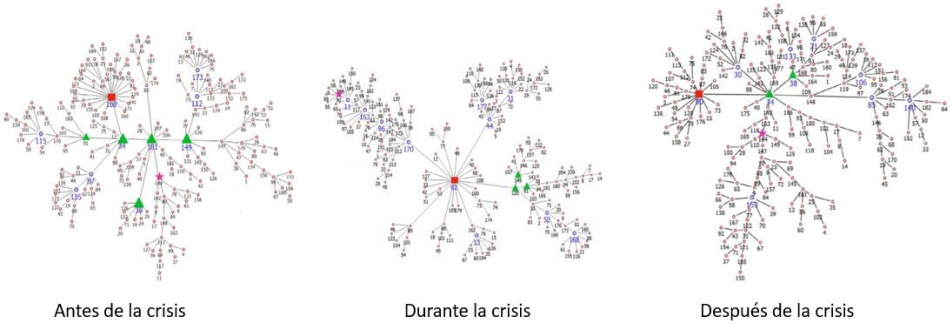


Figura 26. Acciones del mercado de valores de Corea

Fuente: Structural Changes in the Minimal Spanning Tree and the Hierarchical Network in the Korean Stock Market around the Global Financial Crisis. Seong Eun Maeng et. al 2015

Como última conclusión, se realiza un recopilatorio de las aplicaciones vistas que dará cierre a este capítulo.

3.3 Recapitulación de las aplicaciones

Al comienzo del capítulo, en la perspectiva global, se ponía en antecedentes al lector sobre la gran variedad de aplicaciones del problema de Steiner y la necesidad de la utilización de este. A su vez, durante la sección 3.2 se han puesto en práctica las aplicaciones más destacadas del problema, las cuales muestran la envergadura del árbol de Steiner en la sociedad actual.

Por respetar el orden seguido en el trabajo, las infraestructuras es el campo de aplicación más antiguo del árbol de Steiner. Es el que ha permitido el óptimo despliegue de la línea ferroviaria en nuestro país. Teniendo en cuenta; la inversión requerida en cada tramo y la distancia necesaria entre destinos. También ha fomentado la instalación de red eléctrica que requiere de unas pérdidas de Joule mínimas. A su vez, la red de minas también ha sido objeto de observación por su estrecho compromiso entre la minimización del coste y el ancho suficiente de los caminos para la extracción del mineral.

Respecto a las redes de telecomunicaciones, la red de fibra óptica ha permitido el desarrollo de la banda ancha gracias al árbol de Steiner. A lo que se suma la red inalámbrica, la cual permite la conexión "*Wireless Networks*" optimizando la energía capaz de emitir cada sensor de la red. También, el enrutamiento multicast, que permite transmitir archivos multimedia mediante la conexión de servidores en una red inalámbrica de ordenadores, ha sido objeto de observación por el problema de Steiner.

Adicionalmente, las nuevas tecnologías cada vez más arraigadas al presente han tenido un rol dentro del árbol de Steiner. La tecnología VLSI permite conectar los componentes de un circuito integrado. Para ello, se ha considerado la minimización del retraso de señal y de consumo de energía, a través de la reducción de cable. Por si fuera poco, en el cuerpo humano también se han detectado patrones que se pueden modelar con el problema de Steiner. Tanto en las biomoléculas, como en el modelado de las neuronas para descifrar sus conexiones e investigar enfermedades, se utiliza el árbol de Steiner. Por otro lado, el estudio de los árboles filogénicos puede responder preguntas sobre clasificación básica. Además, se puede determinar la divergencia entre dos especies para conocer los predecesores de los animales actuales.

Otras aplicaciones recientemente descubiertas son el marketing viral, la optimización de rutas aéreas y la diversificación de los porfolios de acciones.

Mediante el marketing viral es posible minimizar el número de personas a las que hay que hacer llegar una idea. Posteriormente, estas personas son las encargadas de difundir esa idea. Es posible obtener este efecto realizando una red de contactos en la que se les da más valor a los contactos mejor relacionados. Dichas relaciones serán las personas que en primera instancia transmitirán la idea, las cuales permiten disminuir el esfuerzo de ventas al ser minimizadas.

Para continuar, las rutas aéreas pueden ser optimizadas teniendo en cuenta la distancia del vuelo y la velocidad máxima que alcanza el avión por zonas. Parece asombroso pensar que, si se modifica suavemente la trayectoria de la nave, se pueden llegar a ahorrar hasta dos horas de combustible en trayectos largos.

Por último, la diversificación de un porfolio es posible también debido al árbol de Steiner. Si cada relación entre las acciones candidatas a pertenecer al porfolio se las pondera con una correlación, es posible elegir las menos relacionadas que aseguran reducir el riesgo del porfolio.

Finalmente, hay que decir que las aplicaciones expuestas muestran lo que actualmente se está llegando a conseguir con el árbol de Steiner. Sin contar la variedad de algoritmos que alberga la teoría de grafos, los cuales se usan para otro tanto de aplicaciones. Es por ello, la importancia de este trabajo de explicar el funcionamiento del problema de Steiner y del algoritmo en particular seguido en este documento. Durante la próxima sección se despeja dicha incógnita.

4 MÉTODOS DE RESOLUCIÓN

"La vida está llena de señales; el secreto consiste en saber interpretarlas. Ghosh lo denominaba heurística, método de resolver un problema para el que no existe ninguna fórmula"

Abraham Verghese

Como se ha nombrado con anterioridad, los métodos deterministas conocidos para encontrar árboles mínimos de Steiner (SMT) necesitan tiempo exponencial de resolución. Esto refuerza el interés y el reciente énfasis en el desarrollo de aproximaciones resolubles en tiempo polinomial. En específico, los algoritmos heurísticos son las técnicas idóneas para conseguir estos resultados en el tiempo considerado. Existen aparte numerosas razones por las que el desarrollo de la heurística se ha acelerado en los últimos años, entre ellas;

- A veces, los métodos de resolución óptimos no son conocidos. Por ello, las técnicas heurísticas proporcionan un árbol solución de mínima distancia. Para el caso de los grafos ponderados y del estudio en cuestión de este trabajo se busca el coste total mínimo de los arcos que forman parte de la solución.
- Actualmente, solo los problemas de relativamente pequeño tamaño pueden ser resueltos utilizando algoritmos exactos. De tal modo, los problemas más complejos deben ser abordados por vías de aproximación.
- Las soluciones heurísticas pueden ser utilizadas como límite superior para mejorar la eficiencia de los algoritmos deterministas.

Es por ello, que se expone y se lleva a cabo una de las numerosas técnicas de resolución metaheurísticas del problema de Steiner basada en algoritmos genéticos y con la ayuda del árbol de mínima expansión. En la sección 4.2 se trata el árbol de mínima expansión, imprescindible en la fase inicial para crear soluciones admisibles (no tienen por qué ser óptimas), y en la sección **¡Error! No se encuentra el origen de la referencia.** se abordan los algoritmos genéticos.

4.1 Minimum spanning tree (MST)

El problema del árbol de expansión mínima es uno de los más típicos y populares problemas de optimización combinatoria. Los métodos de resolución del MST han generado importantes ideas para la combinatoria moderna y juegan un papel crucial en el diseño de algoritmos computacionales. Es por ello, el interés de la comprensión del árbol de expansión mínima.

Por si fuera poco, la implementación del árbol de expansión mínima es crucial para la generación de la población inicial (primeras soluciones admisibles) del algoritmo genético para resolver el

problema de Steiner en grafos. El MST se aplica tantas veces como individuos (soluciones) se quieran considerar en la primera población. Para generar diversidad de soluciones en un mismo grafo a partir del MST, será necesario cambiar los costes de los arcos aleatoriamente por cada solución. Este hecho, será clave en el transcurso del algoritmo para proporcionar buenas soluciones.

El árbol mínimo de expansión se define como:

Dado: Un grafo no dirigido conectado por arcos ponderados $\mathrm{G} = (\mathrm{V}, \mathrm{E})$, donde *V* es el número de nodos, *E* es el número de arcos y c(e)>0 denota el peso del arco $e \in E$.

Encontrar: Un subconjunto conectado $\mathrm{T} = (\mathrm{V}, \mathrm{E}(\mathrm{T}))$ en G, tal que $\sum_{e \in E} c(e)$ sea mínimo

---

Lo que es equivalente a pensar, que el MST forma un árbol mínimo de Steiner donde todos los nodos considerados son terminales y no existen nodos Steiner. Sin embargo, el MST no es posible utilizarlo como el algoritmo de resolución para problemas de Steiner debido a que su solución para la gran mayoría de problemas de diseño de redes no es óptima. Por otro lado, se trata de una técnica muy buena para obtener árboles dentro de una red. Por eso, se utiliza para generar las primeras soluciones admisibles del algoritmo heurístico.

Entre las técnicas del MST más destacadas se encuentran los algoritmos de Boruvka, Prim y Kruskal.

4.1.1 Algoritmo de Boruvka

Se trata del primer problema de optimización de redes realizado por Otakar Boruvka (1926). Surgió a partir de la necesidad de construir la red eléctrica de la región histórica de Moravia.

Para el desarrollo del método es imprescindible empezar considerando a cada nodo del grafo como un componente distinto para luego continuar seleccionando todos los arcos de menor coste que inciden en cada uno de los nodos. A partir de ahí, se busca en cada etapa los arcos de coste mínimo que inciden en cada componente, evitando que se formen ciclos, hasta que se cumple que el número de nodos es el número de arcos solución menos uno.

4.1.2 Algoritmo de Prim

Se trata de una de las técnicas más sencillas para resolver el problema del árbol de expansión mínima. Fue desarrollado por primera vez por el matemático checo Vojtech Jarnik. Aunque no fue hasta 1957 cuando aparecía por primera vez publicado de forma independiente por el ingeniero informático estadounidense Robert C. Prim, quien le dio fama y por cuyo apellido es conocido.

La solución al problema propuesto por Prim se basa en la idea de ir conectando nodos secuencialmente hasta alcanzarlos a todos. Teniendo como dato de entrada un grafo no dirigido, el algoritmo empieza a construir el árbol a partir de un nodo seleccionado arbitrariamente. A continuación, itera seleccionando en cada etapa el arco de menor coste (uno cualquiera si hay varios) que une un nodo del árbol con otro que aún no está en él, con la restricción de no poder formar ciclos. El proceso se repite hasta añadir todos los nodos, obteniendo como resultado un árbol de expansión cuyo coste será mínimo.

4.1.3 Algoritmo de Kruskal

Su creación se debe a Joseph Kruskal (1956), compañero de Prim. Conocido desde entonces por su apellido, la idea principal del algoritmo de Kruskal se resume en conectar arcos, y no nodos, secuencialmente. De esta forma, es innecesario elegir un nodo de partida.

El método comienza eligiendo el arco de menor coste de un grafo no dirigido. Este proceso de elegir los arcos de menor coste se continúa hasta unir todos los nodos del grafo sin formar lazos. O lo que es lo mismo hasta que el número de nodos sea igual al número de arcos menos uno. Finalmente, se obtiene un árbol mínimo de expansión.

El algoritmo de Kruskal recibe especial atención por ser el algoritmo elegido para realizar el programa de resolución de árboles de Steiner en grafos. Se ha elegido esta metodología por ser muy simple de implementar computacionalmente. Se elige desde el primer paso el arco de menor coste y se hace en cada implementación un recuento de nodos raíces para evitar la formación de lazos. Para la posterior comprensión del programa implementado en Matlab sobre este método, se demuestra a continuación como se opera con este algoritmo en base a un ejemplo.

Supongamos que se tiene el siguiente grafo mostrado en la Figura 27:

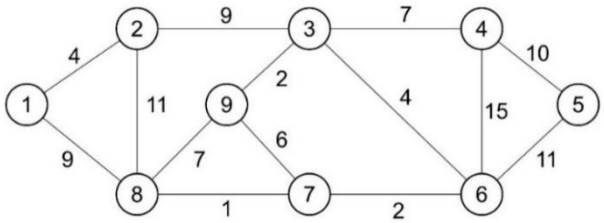


Figura 27. Ejemplo Algoritmo de Kruskal

Si ordenamos las aristas según los costes de menor a mayor obtenemos la Tabla 2:

Tabla 2. Costes asignados a las aristas del ejemplo ordenados de menor a mayor

Arcos	**8-7**	**3-9**	**6-7**	**1-2**	**3-6**	**7-9**	**3-4**	**8-9**	**1-8**	**2-3**	**4-5**	**2-8**	**5-6**	**4-6**
Coste	1	2	2	4	4	6	7	7	9	9	10	11	11	15

El primer paso del algoritmo es señalar el arco de menor coste a la vez que se le asigna el nodo 8 a la raíz del nodo 7 (podría haber sido al revés, esto se aplica para el resto de pasos) obteniendo los siguientes resultados. La Figura 28 muestra dichos cambios:

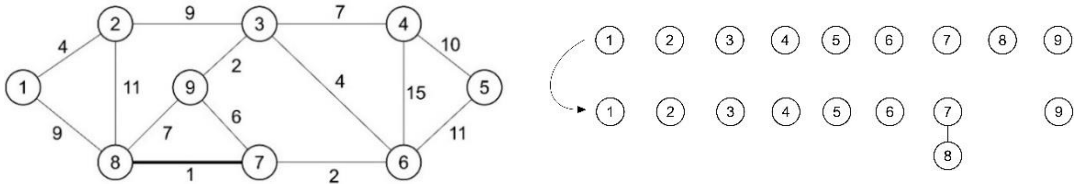


Figura 28. Primer paso del Algoritmo de Kruskal y diagrama de nodos raíces correspondiente

En el segundo paso, la Figura 29 señala el arco siguiente con menor coste (3-9) y se asigna el nodo 9 como raíz del nodo 3.

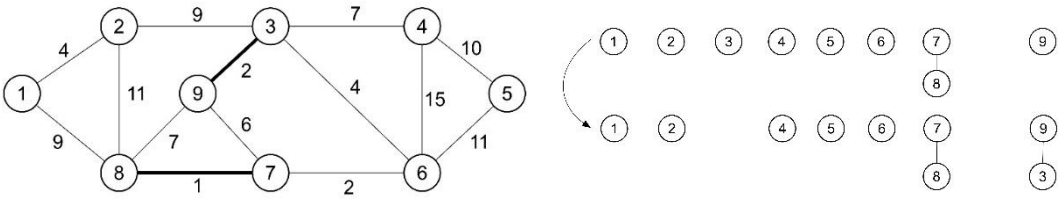


Figura 29. Segundo paso del Algoritmo de Kruskal y diagrama de nodos raíces correspondiente

En el tercer paso, la Figura 30 muestra el arco 6-7 que pasa a formar parte de la solución y se añade el nodo 6 a la raíz del nodo 7.

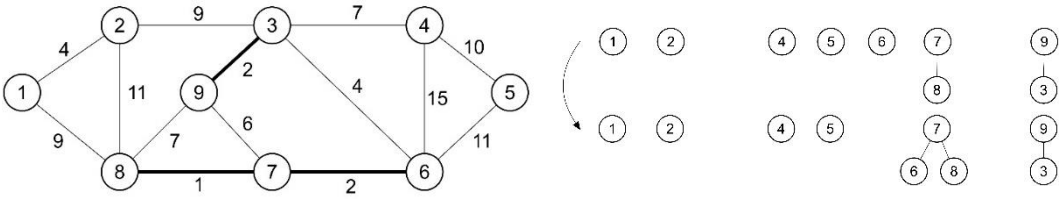


Figura 30. Tercer paso del Algoritmo de Kruskal y diagrama de nodos raíces correspondiente

En el cuarto paso, el arco 1-2 se puede también señalar y se incluye el nodo 1 a la raíz del nodo 2.

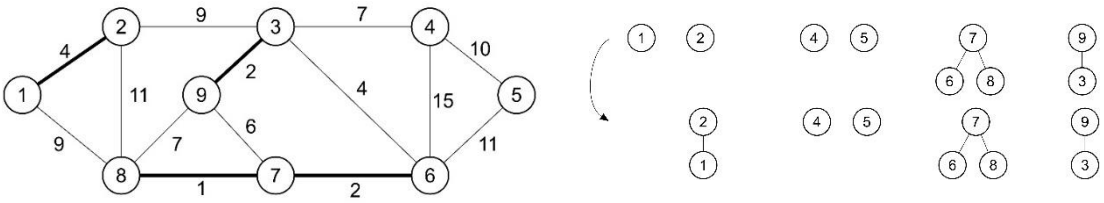


Figura 31. Cuarto paso del Algoritmo de Kruskal y diagrama de nodos raíces correspondiente

En el quinto paso, el arco 3-6 se une al conjunto solución mientras que los nodos 3 y 9 se unen a la raíz del nodo 6. Como el nodo 6 es descendiente del nodo 7, los nodos 3 y 9 podrán ser descendientes directos del nodo 7. La Figura 32 señala las nuevas medidas adoptadas.

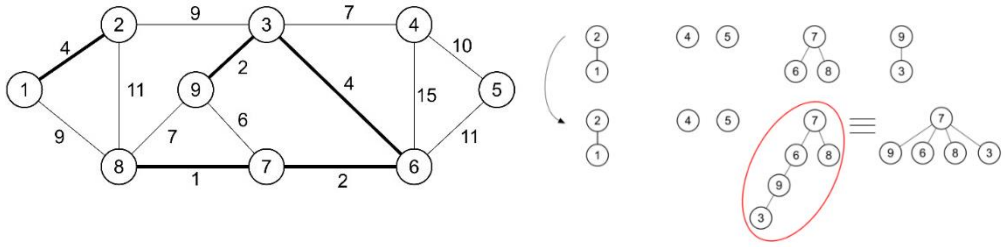


Figura 32. Quinto paso del Algoritmo de Kruskal y diagrama de nodos raíces correspondiente

En el sexto paso, el arco 7-9 no se puede incluir a la solución porque el nodo 7 es ya la raíz del nodo 9. Esto es debido a que si se une dicho arco se forma un ciclo. Luego este arco no forma parte de la solución.

En el séptimo paso, se une el arco 3-4 a la solución y en el diagrama de los nodos raíces, el nodo 4 pasa a formar descendencia del nodo raíz 7 ya que el nodo 3 es descendiente del nodo 7. La Figura 33 muestra las nuevas modificaciones en la solución.

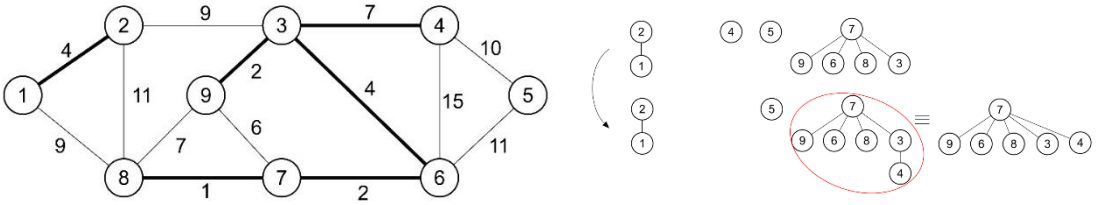


Figura 33. Séptimo paso del Algoritmo de Kruskal y diagrama de nodos raíces correspondiente

En el octavo paso, el arco 8-9 no pueden formar parte de la solución porque los nodos 8 y 9 ya comparten raíz del nodo 7. Luego, forman ciclo.

En el noveno paso, el arco 1-8 pasa a ser solución mientras que tanto el nodo 2 como el 1 en el diagrama de nodos raíces pasan a formar parte de la descendencia del nodo 7. La Figura 34 lo muestra.

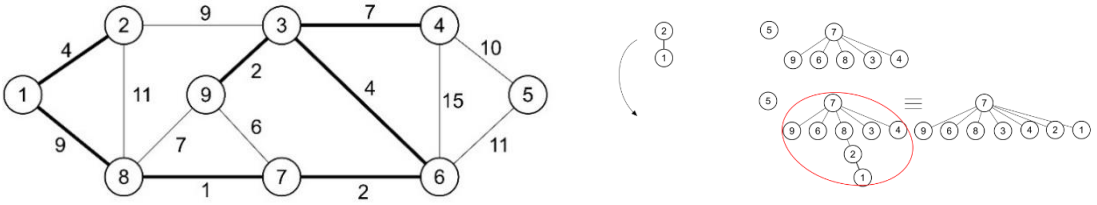


Figura 34. Noveno paso del Algoritmo de Kruskal y diagrama de nodos raíces correspondiente

En el décimo paso, no es posible agregar a la solución el arco 2-3 por formar lazo y por consiguiente ya son nodos pertenecientes a la misma raíz.

En la Figura 35 , el undécimo paso cierra el árbol mínimo de Steiner incorporando el arco 4-5 a la solución. De la misma forma, el último nodo que falta por ser descendencia del nodo 7 se incorpora a dicha raíz.

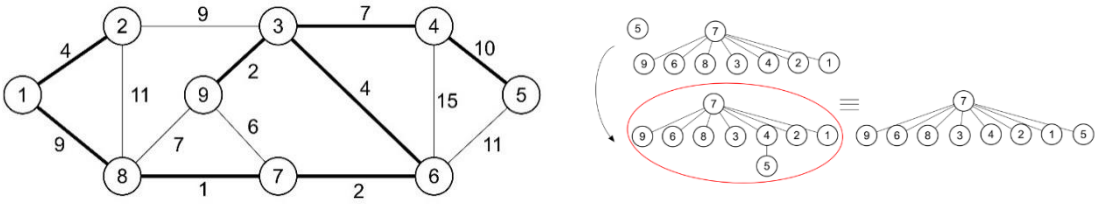


Figura 35. Undécimo paso del Algoritmo de Kruskal y diagrama de nodos raíces correspondiente

Por ende, se muestra la solución obtenida con las aristas resaltadas en negrita y el diagrama de nodos raíces solo muestra el nodo 7 como raíz de todos los nodos.

A continuación, se pasa a explicar los algoritmos genéticos que hacen uso del MST en primera instancia como se ha nombrado con anterioridad.

4.2 Algoritmos genéticos

Los algoritmos genéticos (AGs) son métodos de optimización (Goldberg, 1989) basados en la teoría de Darwin (Tutter, 1989), como la vida biológica en nuestro mundo que envuelve los mecanismos de reproducción, mutación, competición y selección natural. En la naturaleza, los individuos de una población compiten entre sí en la búsqueda de recursos tales como comida, agua y refugio. Incluso los miembros de una misma especie compiten a menudo en la búsqueda de un compañero para producir descendencia. Aquellos individuos que tienen más éxito en sobrevivir y en atraer compañeros tienen mayor probabilidad de generar un gran número de descendientes. Por el contrario, individuos poco dotados producirán un menor número de descendientes. Esto significa que los genes de los individuos mejor adaptados se propagarán en sucesivas generaciones hacia un número de individuos creciente. La combinación de buenas características provenientes de diferentes antecesores puede a veces producir descendientes con muy buenas facultades, cuya adaptación es mucho mayor que la de cualquiera de sus ascendientes. De esta manera, las especies evolucionan logrando unas características cada vez mejor adaptadas al entorno en el que viven.

El poder de los AGs proviene del hecho de que se trata de una técnica robusta y pueden tratar con éxito una gran variedad de problemas provenientes de diferentes áreas, incluyendo aquellos en los que otros métodos encuentran dificultades. Si bien no se garantiza que el AG encuentre la solución óptima del problema al final de las iteraciones, existe evidencia empírica de que se encuentran soluciones de un nivel aceptable, en un tiempo competitivo con el resto de los algoritmos de optimización combinatoria.

Los AGs están compuestos generalmente de dos procesos. El primer proceso es la generación de individuos para la producción de la siguiente población y el segundo proceso es la manipulación de los individuos seleccionados para conformar la siguiente generación mediante las técnicas de cruzamiento (reproducción) y mutación. El mecanismo de selección determina cuáles son los individuos seleccionados para la reproducción y mutación. El principio más importante de la estrategia de selección es mientras mejor sea el individuo, mayor es su probabilidad de ser padre (Razali and Geraghty, 2011).

El mecanismo de selección reduce el área de búsqueda dentro de la población descartando soluciones pobres, a su misma vez debe dar la oportunidad a todos los individuos ya sean buenos o malos de que puedan reproducirse. Es importante encontrar un equilibrio donde las soluciones pobres deben tener la oportunidad de ir a la siguiente generación, y las buenas soluciones pasen a la siguiente generación con más frecuencia que las soluciones pobres. El mecanismo de selección debe ser elegido de tal manera que la convergencia a la solución óptima global esté garantizada. Durante el desarrollo de los AGs, muchas estrategias de selección han sido propuestas y utilizadas tales como: Selección de Ruleta, Selección basada en Ranking o algunas selecciones aplicadas en estrategias evolucionarias (Geyer et al., 1999).

Para desarrollar toda la teoría del algoritmo genético se comenzará en la sección 4.2.1 con la definición del problema continuando con los parámetros a tener en cuenta. Posteriormente, en

la sección 4.2.2 se explican las diferentes partes del algoritmo genético, las cuales son: codificación binaria de las soluciones, selección, reproducción, mutación y prevención de incesto. Adicionalmente, se proporciona en la sección 4.2.3 el funcionamiento específico del AG utilizado en el proyecto. Para concluir, en la sección 4.2.4 se describen las ventajas y desventajas de los AGs.

4.2.1 Definición del algoritmo genético

Un algoritmo genético es un método iterativo de búsqueda de soluciones que imita la teoría de la evolución de Darwin para la resolución de problemas, como se puede intuir de la introducción. En el método, se parte de un conjunto de soluciones admisibles que forman la población inicial de la cual se seleccionan los individuos más capacitados (mejores soluciones). Aquellos individuos que demuestren mayor adaptación al medio se reproducen y mutan para finalmente obtener una mejor generación de soluciones.

Antes de comenzar con el algoritmo, es necesario garantizar la diversidad de soluciones en la población inicial para poder abordar el problema desde diferentes puntos del espacio. Para poder asegurar que la solución final obtenida en el algoritmo sea la óptima o la más cercana al óptimo, el problema debe abarcar soluciones iniciales lo más alejadas posibles en el espacio tal como se muestra en la Figura 36. Donde cada punto de la gráfica representa una posible solución admisible y el eje de coordenadas determina el coste asociado a cada solución.

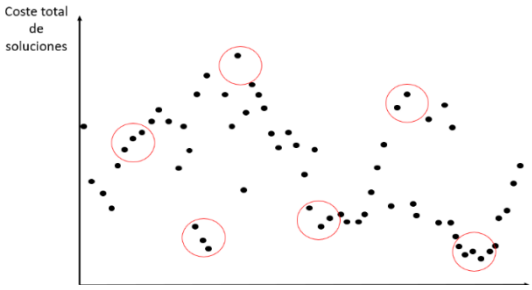


Figura 36. Espacio soluciones del problema de Steiner

Una vez que los individuos son elegidos aleatoriamente gracias al MST, se procede con el algoritmo de resolución que sigue los siguientes pasos. Para ello, será importante conocer que un gen es un arco del grafo.

- Evaluar la puntuación de cada uno de los cromosomas generados, es decir, medir el coste total de cada una de las soluciones admisibles propuestas.
- Permitir la reproducción de los cromosomas más adaptados al medio (con menor coste total).
- Con cierta probabilidad de mutación, mutar un gen de un individuo cualquiera de la población.
- Organizar la nueva población sustituyendo los individuos mutados por el individuo antes de mutar y el cromosoma obtenido por cruce por algún individuo con coste total alto.

Dichas condiciones se repiten hasta que se da un argumento de salida, ya sea un número máximo de iteraciones o cuando no se produzcan más cambios en la población (convergencia del algoritmo). En el algoritmo genético utilizado se determina el fin del algoritmo una vez que se

llega a un número máximo de iteraciones. Se escogió dicha condición, en vez de la de convergencia del algoritmo, debido a que existen problemas de Steiner donde al realizar algunas iteraciones no se avanza en disminuir el coste total de las soluciones (dando a pensar que no hay más soluciones posibles) cuando si se continua, se pueden obtener mejores resultados.

Para el estudio del algoritmo genético hay que tener en consideración una serie de parámetros que se irán cambiando en el análisis del algoritmo. El ajuste óptimo de los parámetros permite minimizar el tiempo de resolución de los problemas del árbol de Steiner, así como la mejora de la calidad de las soluciones. Dichos parámetros son:

- Tamaño de la población: Determina el número de individuos de una población. En caso de que el tamaño sea muy pequeño, la búsqueda de soluciones será escasa y poco óptima. De lo contrario, si la población es demasiado amplia, el algoritmo genético será excesivamente lento. Estudios sobre el límite máximo de individuos de una población revelan que, a partir de un tamaño determinado, no se consigue una mayor velocidad en la resolución del problema.
- Probabilidad de cruce: Indica la frecuencia con la que se producen cruces entre los cromosomas padre.
- Probabilidad de mutación: Indica la frecuencia con la que los genes de un cromosoma son mutados.

Se tomará siempre la probabilidad de cruce aproximadamente del 75-80% y la de mutación del 25-20%. Aparte, existen otros parámetros a determinar; cómo elegir los padres de entre la población para la reproducción y por qué individuo sustituir el cromosoma generado por reproducción.

4.2.2 Operaciones del algorítmo genético

En el algoritmo genético va implícito el método para resolver el problema. Hay que tener en cuenta que el algoritmo genético es independiente del problema, lo cual lo hace un método potente, al resultar útil en cualquier ámbito de acción, pero a la vez débil, pues no está especializado en ninguno.

Las soluciones codificadas en un cromosoma compiten para ver cuál constituye la mejor opción. Además, el resto de las soluciones ejercerá una presión selectiva sobre la población, de forma que sólo los que mejor se adaptan (aquellos que se acerquen a la solución) sobrevivan y dejen su material genético a las futuras generaciones, igual que en la evolución de las especies. La diversidad genética se introduce mediante mutaciones y reproducción sexual como se nombraba anteriormente. Por lo tanto, el método consiste en hallar de qué parámetros depende el problema, codificarlos en un cromosoma, y aplicar los métodos de la evolución: selección, reproducción sexual y mutaciones que generen diversidad.

4.2.2.1 Codificación binaria de los individuos

Cada uno de los arcos del problema de Steiner presentará un gen, cuyo conjunto forma el cromosoma. Aquellos arcos que formen parte de la solución serán codificados como “1” y aquellos que estén excluidos de la solución valdrán “0”. Existirán tantos genes como arcos tenga el problema.

4.2.2.2 Selección

Es necesario hacer una selección con los individuos más capacitados para que éstos sean los que se reproduzcan con mayor probabilidad de acuerdo con la teoría de Darwin. En la nombrada teoría, los cromosomas más capacitados son los que deben sobrevivir y crear una nueva descendencia más facultada. Por lo tanto, una vez evaluado cada cromosoma y obtenida su puntuación, se tiene que crear la nueva población teniendo en cuenta que los buenos rasgos de los mejores individuos se deben transmitir al futuro conjunto de soluciones. Se presentan a continuación las técnicas de selección llevadas a cabo en el algoritmo genético utilizado.

4.2.2.2.1 Selección aleatoria

Consiste en la selección al azar de cualquiera de los cromosomas de la población. Es la selección utilizada a la hora de elegir un individuo para mutar o incluso para sustituir un cromosoma por un individuo cruzado, es decir, reproducido a partir de dos padres.

4.2.2.2.2 Selección por ruleta

Consiste en obtener las probabilidades de los individuos en función de su coste. Así la probabilidad de un individuo i perteneciente a la población I vendrá determinada por:

$$P_i = \left(\frac{Coste\ total_i}{\sum_{i \in I} Coste\ total_i}\right)^{-1}$$

Donde a mejor fitness (menor coste) tenga el cromosoma, mayor probabilidad tendrá de ser elegido.

La selección por ruleta se puede utilizar para elegir a los padres para la reproducción. Si se utiliza $1/P_i$ para el cálculo de las probabilidades, se puede sustituir el hijo por un cromosoma con un coste total alto. Pues ahora, los individuos con mayor probabilidad de selección serán lo de peor fitness.

4.2.2.2.3 Selección por ranking

Consiste en dar probabilidades a los cromosomas según la distribución geométrica. Primero se ordenan los cromosomas según sus costes de menor a mayor. De tal modo que, la probabilidad del cromosoma ubicado en la posición i tendrá la probabilidad:

$$P_i = q(q-1)^i$$

Donde q representa un parámetro de análisis en el método que varía entre 0.5 y 0.6 para ver si afecta al resultado final.

Al igual que en la selección por ruleta, la selección por ranking es característica para elegir padres y si se utiliza la inversa de la fórmula se puede sustituir el hijo por un cromosoma con un coste total alto.

Tanto para la selección por ruleta como para la selección por ranking, una vez obtenidas las P_i se sigue con el siguiente procedimiento. Las probabilidades se sitúan en intervalos de forma que si son 0.5, 0.3 y 0.2 se forman intervalos 0-0.5, 0.5-0.7 y 0.7-1, respectivamente. A continuación, se elige un número generado aleatoriamente entre 0 y 1 que caerá en alguno de los tres intervalos presentados para dicho ejemplo. El cromosoma, cuya probabilidad corresponde al intervalo donde se encuentra el número aleatorio, será el seleccionado.

4.2.2.3 Reproducción

La reproducción consiste en cruzar dos progenitores para crear descendencia. Las reglas establecidas para obtener una combinación binaria de números serán:

- Si el gen de la posición i de ambos progenitores es igual, el hijo hereda ese gen.
- Si el gen de la posición i de ambos progenitores difiere, el hijo puede heredar cualquiera de los genes situados en la posición i.

Para determinar los genes del descendiente que pueden variar por no ser los genes de los padres iguales, es necesario determinar el árbol de mínima expansión para forzar admisibilidad en la solución y poder reemplazar la nueva solución por otra de la población.

4.2.2.4 Mutación

La mutación se lleva a cabo cambiando uno de los genes de un individuo que valga 1 por un 0. Se podría haber elegido cambiar un 0 por un 1, pero como después es necesario aplicar el MST para forzar la admisibilidad de la solución, se ha elegido pasar de 1 a 0 para evitar complicaciones. En las instrucciones seguidas por el MST, habrá que darle un coste muy elevado al arco que pasa a 0 para forzar dicho arco a que no forme ya parte de la solución.

4.2.2.5 Prevención de incesto

Se trata de una medida para evitar elegir padres con un parecido genético por encima de un porcentaje determinado. De esta forma, se evitaría que el programa converja en las primeras iteraciones, ya que, si la mayoría de las soluciones se parecen, no existe diversidad y se puede dar por solución un mínimo local. Poder evitar este hecho permite realizar una mayor exploración de soluciones posibles.

En la práctica, la prevención de incesto más que una ayuda resulta ser un inconveniente. Esto tiene su explicación a que hay problemas en los que rápidamente todos los individuos de la población tienen un parecido por encima de un 60%, como se ha estudiado, antes de llegar a la última iteración programada. Si se aplica la prevención de incesto, el programa no permite seguir iterando, pues no se encuentra ningún progenitor con menor parentesco.

El problema de la rápida convergencia, que se podría evitar con la prevención de incesto, es resuelto implementando 30 poblaciones, en vez de una, donde finalmente se elige el mejor resultado de entre las 30 poblaciones generadas. Así, se solventa el problema de no realizar una buena exploración de soluciones posibles a pesar de tener una población inicial diversificada. Por ello, aunque alguna de las poblaciones dé cómo mejor solución un mínimo local, existen otras 29 poblaciones diversificadas en el espacio que proporcionan otros resultados.

4.2.3 Funcionamiento del algoritmo genético

Para comprender el funcionamiento completo del algoritmo genético llevado a cabo en este estudio se procede a explicar el procedimiento seguido.

1. Lo primero de todo es codificar cada solución admisible de la población en código binario.
2. Una vez que se tiene una población diversificada se procede a realizar tantos cambios en

la población mediante reproducción y mutación como iteraciones se quieran obtener.

3. El 75-80% de las veces se optará por la reproducción donde se eligen dos padres de entre las mejores soluciones (bajo coste) y se obtiene un hijo a partir de ellos que se sustituirá en la población por una solución con peor fitness (alto coste) para así ir formando una población de soluciones cada vez más próximas al óptimo.
4. El 20-25% de las veces se produce mutación donde un gen del cromosoma cambia su valor y la nueva solución es reemplazada por la solución elegida para mutar. Con ello, se consigue explorar más lugares del espacio de soluciones.
5. Finalmente, cuando se han realizado todas las iteraciones se elige el individuo de la población con mejor fitness (menor coste).

4.2.4 Ventajas e inconvenientes

Al igual que cualquier método o técnica, el algoritmo genético también tiene ventajas e inconvenientes. Entre las ventajas:

- Los algoritmos genéticos son intrínsecamente paralelos al operar simultáneamente con varias soluciones. En vez de trabajar de forma secuencial como las técnicas tradicionales, los algoritmos genéticos son capaces de explorar el espacio de soluciones en varios lugares al mismo tiempo. Si la solución que descubren resulta subóptima basta con desecharla y seguir por otros caminos.
- Por otro lado, los problemas de optimización resueltos con algoritmos genéticos resultan menos afectados por mínimos locales (falsas soluciones) que las técnicas tradicionales. Muchos algoritmos de búsqueda pueden dar soluciones de óptimos locales ya que en la cercanía no existen mejores soluciones. Sin embargo, el algoritmo genético al explorar el espacio de soluciones de forma diversificada evita este problema.
- No necesitan conocimientos específicos sobre el problema que intentan resolver. Realizan cambios aleatoriamente en sus soluciones candidatas y luego utilizan la función de aptitud para determinar si esos cambios producen una mejora o no.
- Resulta sumamente fácil ejecutarlos en las modernas arquitecturas masivas en paralelo.
- Usan operadores probabilísticos, en vez de los típicos operadores determinísticos de otras técnicas.

El algoritmo genético también contiene desventajas citadas a continuación:

- Pueden tardar mucho en converger o no converger en absoluto, dependiendo en cierta medida de los parámetros que se utilicen; tamaño de población, número de generaciones…
- Pueden converger prematuramente debido a una serie de problemas. Si un individuo que es más apto que la mayoría de sus competidores emerge muy pronto en el curso de la ejecución, se puede reproducir tan abundantemente que merme la diversidad de la población demasiado pronto. Ello provoca que el algoritmo converja hacia el óptimo local que representa ese individuo, en lugar de rastrear el paisaje adaptativo lo bastante a fondo para encontrar el óptimo global. Esto es un problema especialmente común en las poblaciones pequeñas, donde incluso una variación aleatoria en el ritmo de reproducción puede provocar que un cromosoma se haga dominante sobre los otros.

5 IMPLEMENTACIÓN DEL ALGORITMO

"Una vez clasificado el algoritmo, ya sabes cómo empezar a romper el código"

Neal Stephenson

Para poder obtener soluciones del árbol de Steiner como la del ejemplo mostrado en la **¡Error! No se encuentra el origen de la referencia.**, es necesario seguir los pasos comentados en la sección

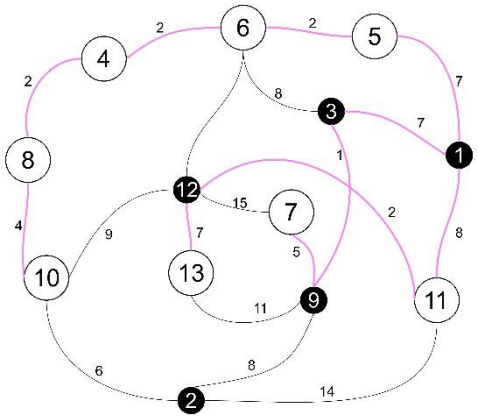


Figura 37. Ejemplo árbol de Steiner

4.2.3.

Dichos pasos se comentarán a continuación brevemente:

1. Codificación de las soluciones admisibles obtenidas por MST para la generación de la población inicial.
2. Se producen cambios en la población con una probabilidad del 75-80% para la reproducción de las especies y el 15%-20% de las veces se mutarán las soluciones.
    - Si se reproducen, se eligen dos progenitores y se reemplaza una de las soluciones de la población por el hijo.
    - Si se muta, se selecciona una solución para cambiar un gen de 1 a 0.
3. Finalmente, cuando se han realizado todas las iteraciones se elige el individuo de la población con mejor fitness.

Una vez establecidos los pasos a seguir en la resolución del problema de Steiner, se comenta cómo se ha estructurado el algoritmo de programación.

La implementación del algoritmo para resolver el problema de Steiner consta de diez funciones divididas en dos partes implementadas en el programa de programación Matlab. Se trata de un

conjunto de funciones anidadas, las cuales son dirigidas por la función final “genetico1”. La primera parte que se ejecuta se trata del árbol de mínima expansión y la segunda es la implementación del algoritmo genético haciendo uso de las soluciones generadas por el MST. En la **¡Error! No se encuentra el origen de la referencia.** se muestra un esquema general del a lgoritmo a implementar.

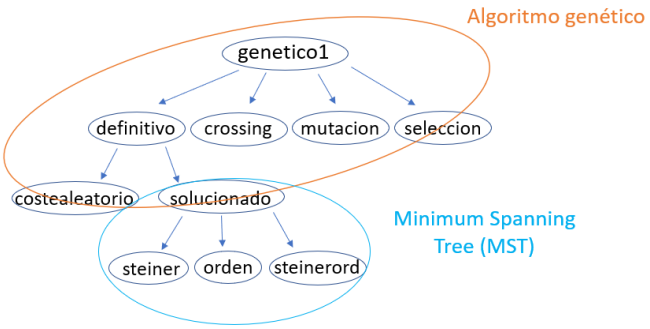
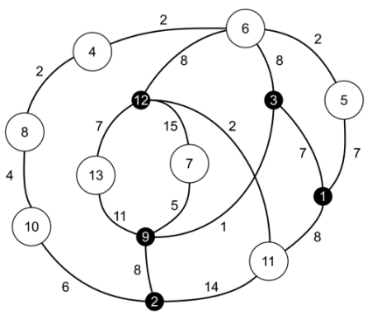


Figura 38. Esquema implementación del algoritmo

Antes de hacer una breve introducción del objetivo de cada función, es conveniente definir los parámetros de entrada al algoritmo, los cuales son:

- stein: matriz de datos que contiene los arcos con los costes correspondientes
- terminales: vector con los nodos terminales
- numvert: número de nodos terminales y Steiner que contiene el grafo
- numarist: número de arcos que unen los diferentes nodos

Se proporciona a continuación en la Figura 39 un ejemplo de cómo a partir de un grafo se construyen los parámetros a introducir en Matlab para la resolución del problema de Steiner.



stein

nodo	nodo	coste
1	5	7
3	9	1
3	1	7
4	8	2
6	4	2
6	5	2
6	3	8
8	10	4
9	2	8
9	7	5
10	2	6
11	1	8
11	2	14
12	6	8
12	10	9
12	11	2
12	13	7
12	7	15
13	9	11

terminales

4	5	6	7	8	10	11	13

numvert: 13

numarist: 19

Figura 39. Parámetros de entrada en función del grafo del que se quiere obtener la solución

Mediante los parámetros definidos, las funciones “steiner”, “orden”, y “steinerord” se encargan de definir el número de nodos Steiner, ordenar en la matriz “stein” los arcos según los costes de menor a mayor y de definir cuántos arcos desembocan en cada uno de los nodos Steiner. La

función “steinerord” se considera de especial relevancia para más adelante poder podar aquellos nodos Steiner que en la solución del MST queden como ramas.

Una vez realizada esta preparación, en la función “solucionado” se obtiene la solución del MST para los costes inicialmente dados mediante la ayuda de las funciones “steiner”, “orden” y “steinerord”. A continuación, para poder generar tantas soluciones admisibles como individuos se quiera en la población inicial, es necesario cambiar los costes aleatorios de la matriz “stein” para poder generar otras soluciones admisibles en el mismo grafo. Mediante la función “costealeatorio”, se generan los nombrados costes aleatorios para crear en la función “definitivo” dos matrices cruciales en la implementación del algoritmo. Estas son “matrizsol” con tantas soluciones (creadas a partir de los costes aleatorios) como número de individuos tenga la población y “costesol” donde se muestran los costes aleatorios utilizados para la creación de cada solución guardada en “matrizsol”. Hasta aquí se ha obtenido la población inicial para poder operar con ella mediante el algoritmo genético y obtener soluciones próximas al óptimo.

Mediante las funciones “crossing” y “mutación” se lleva a cabo el entrecruzamiento entre dos individuos y la mutación de un gen de una de las soluciones, respectivamente. Esto es posible una vez se haya definido en la función “seleccion”, cómo se quieren realizar cambios en la población de forma aleatoria o mediante ruleta o ranking.

Finalmente, la función “genetico1” enlaza las funciones explicadas para darle forma al algoritmo. Esta función coordina mediante un número de iteraciones y un número de individuos dado (aparte de los parámetros definidos anteriormente) las iteraciones que se deben de hacer para obtener una solución fiable.

Para la comprensión completa del algoritmo, a continuación, se definen detalladamente cada una de las funciones proporcionándose el objetivo, las variables de la función, el pseudocódigo y el diagrama de flujo correspondiente. Aparte, en el anexo de este documento se adjuntan las líneas de código que se han preferido omitir en este apartado para evitar contenidos redundantes.

5.1 Función ‘orden’

Se trata de la primera función del algoritmo de programación. Se verá su utilidad reflejada en los siguientes apartados.

5.1.1 Objetivo

El objetivo de esta función es reordenar la matriz inicial dada “stein”, de tal modo que los costes de los arcos (última columna de la matriz inicial en la figura 1) estén ordenados de menor a mayor. De forma que podamos convertir la matriz inicial “stein” en la matriz resultante “nuevostein”. Mediante la introducción de los parámetros “stein” y “numarist” se pretende obtener la variable deseada “nuevostein”. La **¡Error! No se encuentra el origen de la referencia.** ejemplifica el o bjetivo de la función.

Para su funcionamiento se ha definido al comienzo del código la matriz resultante, la cual primero se completa con ceros, a excepción de la última columna que se colocan los costes de las aristas de menor a mayor. Así, se evita que el software dedique un tiempo innecesario a crear en cada

implementación “celdas de memoria”. Este proceso de asignar a la memoria del programa un número (habitualmente 0) que después habrá que cambiar, será básico en la sintaxis del lenguaje de programación.

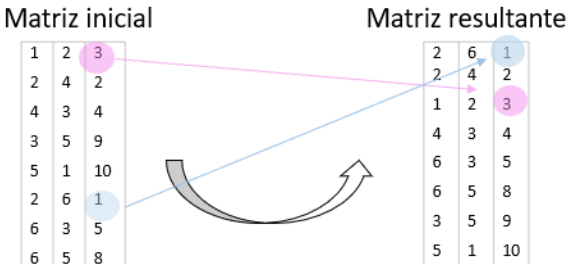


Figura 40. Esquema cambio de filas en función del coste de las aristas de menor a mayor

5.1.2 Variables de la función

En la Figura 41 se muestra un esquema de las variables de entrada, de salida e internas de la función.

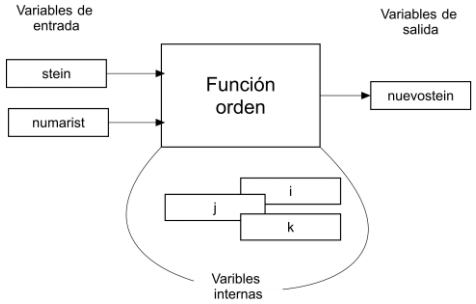


Figura 41. Variables de la función ‘orden’

5.1.3 Pseudocódigo

Se muestra el algoritmo utilizado para esta función. Los nombres de las variables empleadas significan lo siguiente:

- nuevostein: matriz objetivo de la función
- stein: matriz inicial proporcionada a la función
- numarist: número de arcos del problema

Algoritmo 1: Pseudocódigo de la función ‘orden’

Función [nuevostein]=orden(stein, numarist)

Generar matriz “nuevostein” de dimensiones “numarist” x 3 donde las dos primeras columnas son 0 y la última columna equivale a los costes de “stein” ordenados de menor a mayor.

Generar variables i, k y j con valor de 1 cada una.

Mientras i<=numarist

Mientras k<=numarist

Si stein(k,3)=nuevostein(i,3)

Copiar valor stein(k,1) en nuevostein(j,1)

Copiar valor stein(k,2) en nuevostein(j,2)

//Muy importante esta línea de código ya que en la matriz "stein" los costes pueden estar repetidos y se podría copiar el mismo coste más de una vez. Poniendo los costes ya utilizados a 0 evitamos este problema.

Implementar j

Fin Si

Implementar k

Fin Mientras

Implementar i

Fin Mientras

Devolver matriz "nuevostein" que es equivalente a "stein" pero con los costes de las aristas cambiados de menor a mayor.

Fin Función

5.1.4 Diagrama de flujo

Para comprender mejor la dinámica de la función se adjunta en la Figura 42 el correspondiente diagrama de flujo. Se pueden apreciar dos bucles anidados y un comando "*if*" dentro del bucle interior que será el encargado de ir ordenando los costes en matriz "stein".

5.2 Función 'steiner'

Siendo la segunda función que implementar del programa, se divide al igual que el resto en objetivo, varibles de la función, pseudocódigo y diagrama de flujo.

5.2.1 Objetivo

Con esta función se pretende obtener un vector que contenga los nodos Steiner (nodos "negros") que se recogerán en el vector "nsteiner". Para ello, será necesario generar un vector llamado "nodostn" de dimensión; el número de nodos del problema, tal como viene descrito en la figura 4. Con un bucle, en el vector "nodostn" se pondrán a cero las celdas que contienen los nodos terminales dejando solo los nodos Steiner. Posteriormente, mediante otro bucle se recogerán en el vector final "nsteiner" el valor de las celdas distintas de cero. Este será el procedimiento que seguir mostrado en la Figura 43.

Mediante solamente la introducción en la función de los parámetros "terminales" y "numvert" y la ayuda de las variables internas se obtendrá "nsteiner".

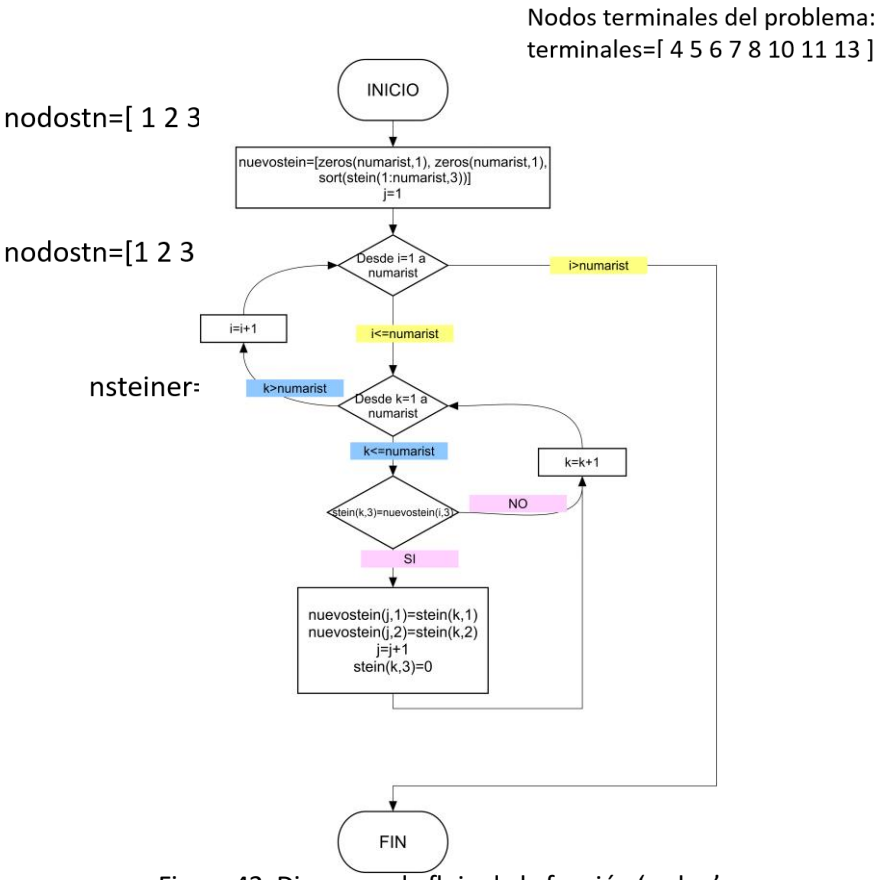
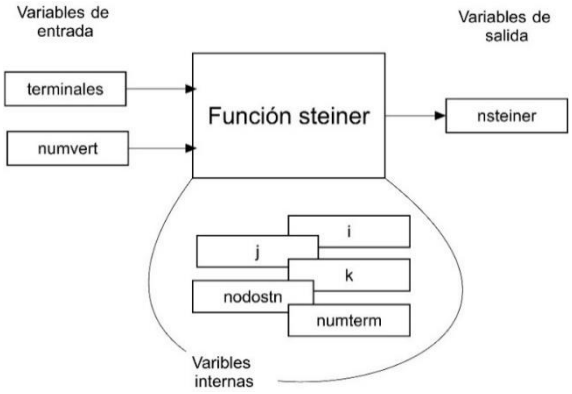


Figura 42. Diagrama de flujo de la función 'orden'

5.2.2 Variables de la función

En la Figura 44 se muestran las variables utilizadas en esta función.



5.2.3 Pseudocódigo

Se introduce para la comprensión del pseudocódigo, el significado de las siguientes variables relevantes:

- nsteiner: vector final que contiene el valor de los nodos Steiner
- terminales: valor de los nodos terminales contenidos en un vector
- numvert: variable con el número de arcos del problema
- numterm: variable con el número de nodos terminales
- nodostn: vector que contiene números del 1 al valor "numvert"

Algoritmo 2: Pseudocódigo de la función 'steiner'

Función [nsteiner]=steiner(terminales, numvert)

Generar variable "numterm" que representa el número de nodos terminales

Generar vector "nsteiner" de dimensión 1 x número de nodos Steiner, lleno de ceros

Generar vector "nodostn" de dimensión 1 x "numvert" que contiene números de 1 a "numvert"

Generar variables i y j con valor de 1 cada una

Mientras i<=numvert

Mientras j<=numterm

Si i=terminales(j)

Poner nodostn(i) a 0

Fin Si

Implementar j

Fin Mientras **//**Se han rellenado de 0 las celdas de "nodostn" donde aparecían los nodos terminales

Implementar i

Figura 44. Variables de la función 'steiner'

Fin Mientras

Poner j e i a 1

Mientras i<=numvert

Si nodostn(i) distinto a 0

Copiar valor nodostn(i) en nsteiner(j)

Sumar 1 a variable j

Fin Si

Implementar i **//**Los nodos que queden en el vector "nodostn" serán los nodos Steiner que no habrán sido cambiados a 0

Fin Mientras

Fin Función

5.2.4 Diagrama de flujo

Se presenta en la Figura 45 el diagrama de flujo de la función 'steiner'. A través de la utilización de dos bucles independientes donde uno de ellos muestra otro bucle en el interior, se pretende dar solución al vector "nsteiner". Es importante recalcar la importancia del primer comando "*if*" que permite sustituir a los nodos terminales por 0.

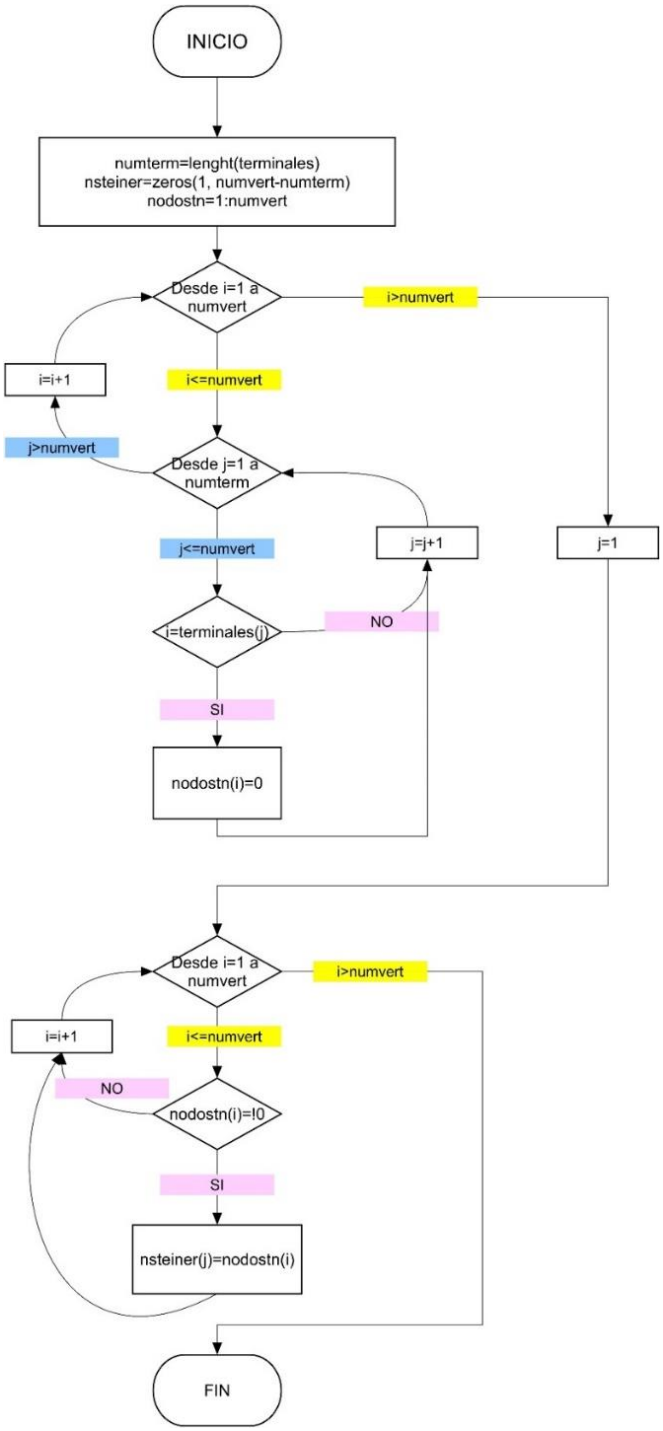


Figura 45. Diagrama de flujo de la función 'steiner'

5.3 Función ‘steinerorden’

Esta función es de vital importancia para poder podar los nodos Steiner de grado 1. A continuación se muestra detalladamente el objetivo de la función, sus variables, el pseudocódigo y el diagrama de flujo.

5.3.1 Objetivo

La función ‘steinerorden’ ordena los nodos Steiner según el número de aristas activas quedando al final del vector los nodos Steiner que no participan en la solución. Las aristas activas son aquellas que forman parte de la solución que desembocan en un nodo Steiner. De este modo, según el orden de los nodos Steiner guardados en el vector “nsteiner”, se introduce en el vector “cont” el número de aristas activas de cada nodo Steiner, tal como muestra la Figura 46. Finalmente, en el vector “nsteinerord” se guardan los nodos Steiner contenidos en el vector “nsteiner” ordenados de menor a mayor número de aristas activas tengan. Hay que tener en cuenta que aquellos nodos no participes de la solución final, como es el caso del nodo número dos de la Figura 46 adjunta, tomará el valor cero en el vector final “nsteinerord”.

Es importante considerar que para el funcionamiento de la función ‘steinerorden’ se requiere de la solución del MST, que se encuentra en la variable llamada “solorden”. Su obtención aún no se ha comentado debido a que es el objetivo de la función ‘solucionado’. Se ha preferido comentar antes esta función pues posteriormente resultará más sencillo el entendimiento de la función ‘solucionado’. Por el momento, se considera que la variable “solorden” viene dada.

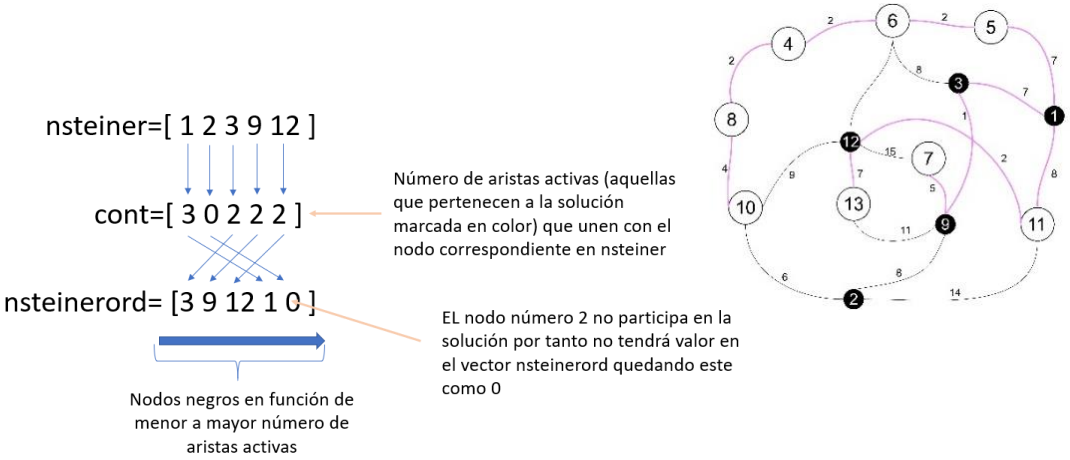


Figura 46. Esquema de cómo ordenar el número de aristas activas que puede tener un nodo Steiner

5.3.2 Variables de la función

Se muestran las variables de la función en la Figura 47.

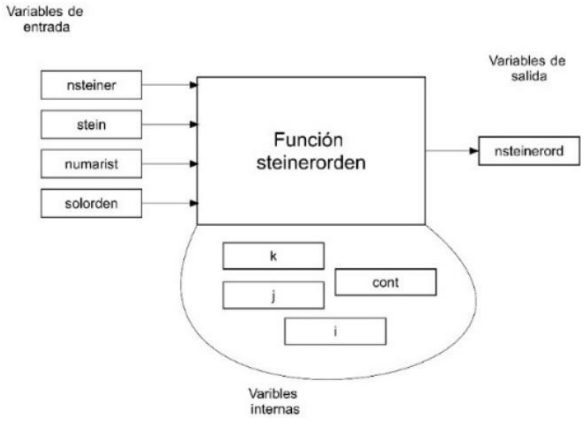


Figura 47. Variables de la función 'steinerorden'

5.3.3 Pseudocódigo

Las variables necesarias para realizar el programa son:

- nsteiner: vector que proporciona los nodos Steiner
- stein: matriz de datos que contiene los arcos con los costes correspondientes
- numarist: número de aristas del problema
- solorden: solución del problema
- nsteinerord: nodos Steiner ordenados de menos a mayor número de aristas activas
- cont: número de aristas activas que llegan a cada uno de los nodos Steiner

Algoritmo 3: Pseudocódigo de la función 'steinerorden'

Función [nsteinerord]=steinerorden(nsteiner, stein, numarist, solorden)

Generar matriz "cont" llena de ceros con las dimensiones del número de nodos Steiner

Generar matriz "nsteinerord" igual a "cont"

Mientras j<= al número de nodos Steiner

Mientras i<= numarist

Si stein(i,1)=nsteiner(j) o stein(i,2)=nsteiner(j)

Si solorden(i)=1

Entonces se le suma uno a variable "cont"

Fin Si

Fin Si

Implementar variable i

Fin Mientras

Implementar variable j

Fin Mientras

Poner j a 1

Mientras k<=máximo de los valores contenidos en matriz “cont”

Mientras i<=dimensión nsteiner

Si cont(i)=k

Copiar nsteiner(i) en nsteinerord(j)

Implementar j

Fin Si

Implementar i

Fin Mientras

Implementar k

Fin Mientras

Fin Función

5.3.4 Diagrama de flujo

En la Figura 48 se muestra el diagrama de flujo diseñado a partir de los requisitos de la función ‘steinerorden’. Se puede distinguir entre un primer gran bucle y un bucle secundario. Por otro lado, la existencia de los tres comandos “*if*” hace posible el funcionamiento de la función.

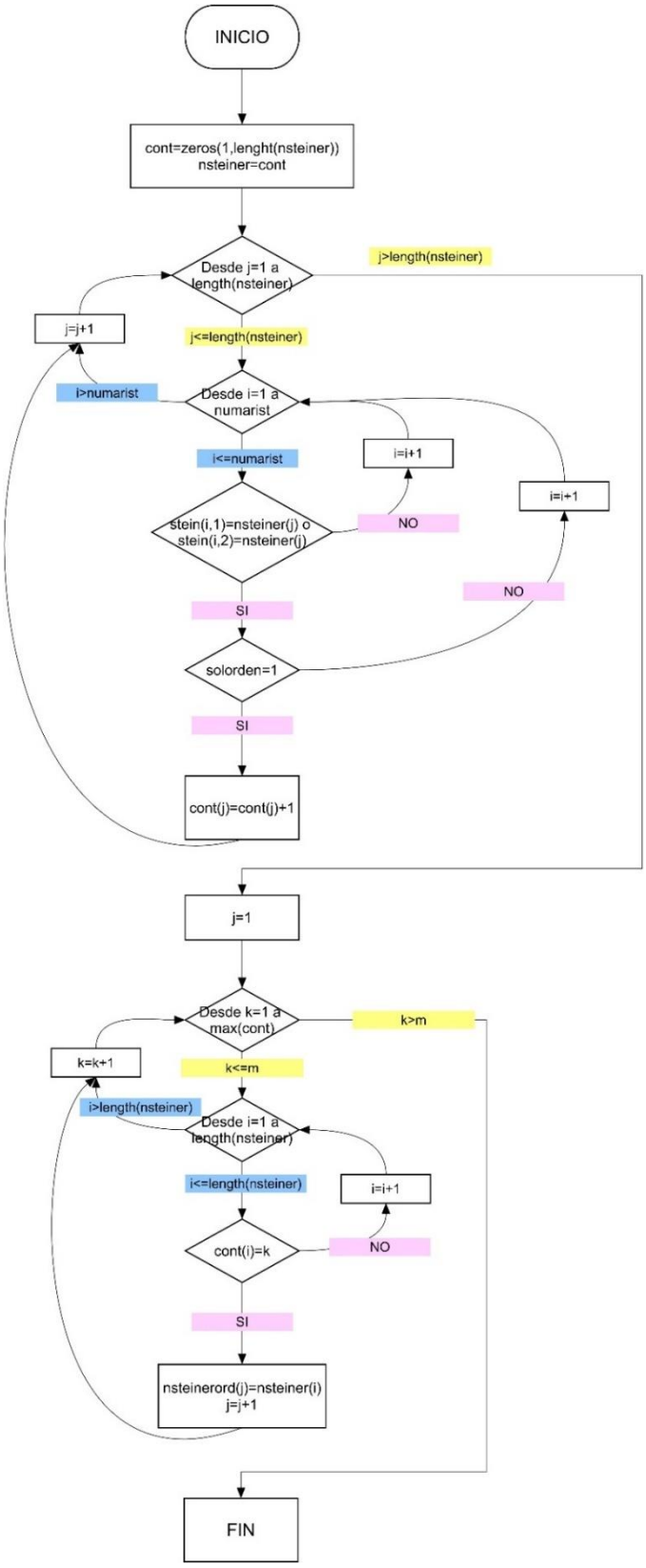


Figura 48. Diagrama de flujo de la función 'steinerorden'

5.4 Función ‘solucionado’

Mediante la función ‘solucionado’ se da cierre a la aplicación del MST en el algoritmo de programación seguido. Pues, esta función proporciona en código binario la solución del MST para obtener la población inicial. Se divide este apartado en objetivo, variables, pseudocódigo y disgrama de flujo.

5.4.1 Objetivo

Mediante la función ‘solucionado’ se pretende obtener el vector solución de dimensiones el número de arcos del grafo relleno de ceros y unos (en código binario) nombrado “solorden” como se puede ver en la Figura 49. Un 0 significará que el arco correspondiente a esa fila no forma parte de la solución y un uno, por el contrario, representa que dicho arco sí forma parte del árbol final. Como se explicó en la sección **¡Error! No se encuentra el origen de la referencia.**, para la i mplementación del MST se hace uso del algoritmo de Kruskal y se procede de la forma explicada en la sección señalada.

La obtención de la solución proporcionada por el MST es fruto de la introducción de los parámetros iniciales y de la utilización de las funciones explicadas hasta ahora.

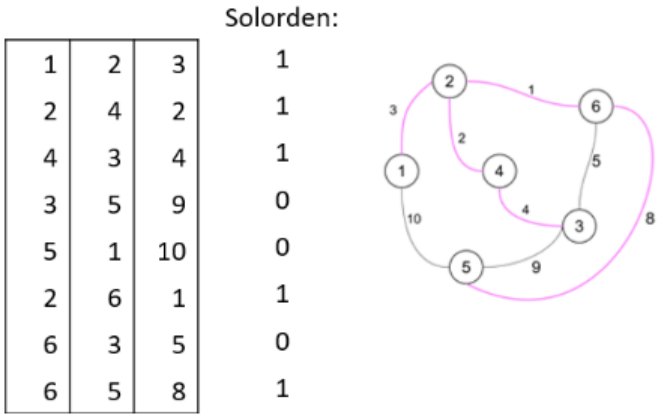


Figura 49. Esquema de cómo obtener una solución admisible en código binario

5.4.2 Variables de la función

Las variables utilizadas en la función se adjuntan en la Figura 50.

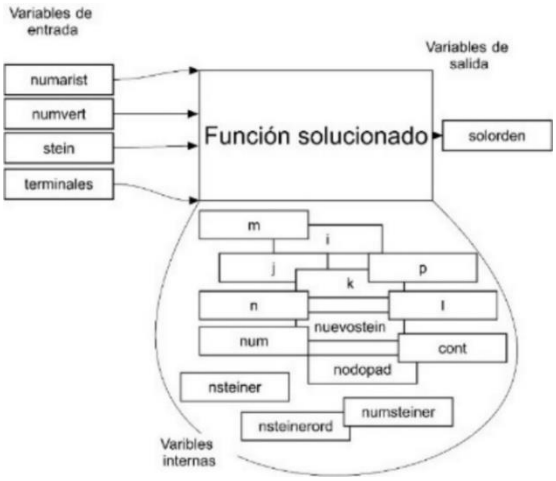


Figura 50. Variables de la función 'solucionado'

5.4.3 Pseudocódigo

Las variables importantes para el entendimiento del pseudocódigo son:

- nodopad: es una matriz interna con dos filas y de columnas el número de nodos del problema llena de números del 1 al número de nodos totales. Ello se repite tanto para la primera fila como para la segunda
- numsteiner: variable de valor el número de nodos Steiner
- cont: vector repleto de ceros con dimensión "numsteiner"

Algoritmo 4: Pseudocódigo de la función 'solucionado'

Función [solorden]=solucionado(numarist, numvert, stein, terminales)

Generar matriz "nuevostein" mediante la función 'orden' con entradas "stein" y "numarist"

Generar matriz "nodopad" igual a matriz de dimensión 2 x "numvert" rellena de número de 1 a "numvert" tanto para la primera fila como para la segunda.

Generar vector "sol" completo de números del 1 a "numarist"

Mientras i<=numarist

Mientras j=1<=numvert

Si nuevostein(i,1) es igual a nodopad(i,j)

m=j

Fin Si

Si nuevostein(i,2) es igual a nodopad(i,j)

n=j

Fin Si

Implementar j

Fin Mientras

Si nodopad(2,m) es igual a nodopad(2,n)

sol(i)=0

Sino si nodopad(1,m) es igual a nodopad(2,m)

nodopad(2,m) es igual a nodopad(2,n)

sol(i) es igual a 1

Mientras k<=numvert

Si nodopad(2,k) es igual a nodopad(1,m)

nodopad(2,k)=nodopad(2,m)

Fin Si

Implementar k

Fin Mientras

Sino si nodopad(1,m) es distinto de nodopad(2,m)

nodopad(2,nodopad(2,m)) es igual a nodopad(2,n)

p es igual a nodopad(2,m)

Mientras l<=numvert

Si nodopad(2,l) es igual a p

nodopad(2,l) es igual a nodopad(2,nodopad(2,n))

Fin Si

Implementar l

Fin Mientras

sol(i) es igual a 1;

Fin Sino si

Fin Sino si

Fin Si

Implementar i

Fin Mientras

Generar vector "nsteiner" mediante la función 'steiner' con las entradas "terminales" y "numvert"

Generar variable "numsteiner" igual al tamaño de "nsteiner"

Generar vector "steinerord" mediante la función 'steinerorden' con entradas "nsteiner", "nuevostein", "numarist" y "sol"

Generar vector de ceros “cont” de dimensión “numsteiner”

Mientras j<=numsteiner

Mientras i<=numarist

Si nuevostein(i,1) es igual a nsteinerord(j) o nuevostein(i, 2) es igual a nsteinerord(j)

Si sol(i) es igual a 1

cont(j)= cont(j)+1

num=i

Fin Si

Fin Si

Implementar i

Fin Mientras

Si cont(j) es igual a 1

sol(num) es igual a 0

Fin Si

Implementar j

Fin Mientras

Generar vector **“**solorden” lleno de ceros con las dimensiones de 1 x “numarist”

Mientras i<=numarist

Mientras j<=numarist

Si stein(i,1) =nuevostein(j,1) y stein(i,2)=nuevostein(j,2)

solorden(i)=sol(j)

Fin Si

Implementar j

Fin Mientras

Implementar i

Fin Mientras

Fin Función

5.4.4 Diagrama de flujo

El diagrama de flujo mostrado en este apartado está dividido en dos figuras debido sus grandes dimensiones. Se puede ver como un gran bucle engloba toda la Figura 51 mientras que en la Figura 52 se pueden distinguir dos bucles de menor tamaño. Al final de la función se obtiene el vector “solorden” con la solución admisible del árbol de Steiner.

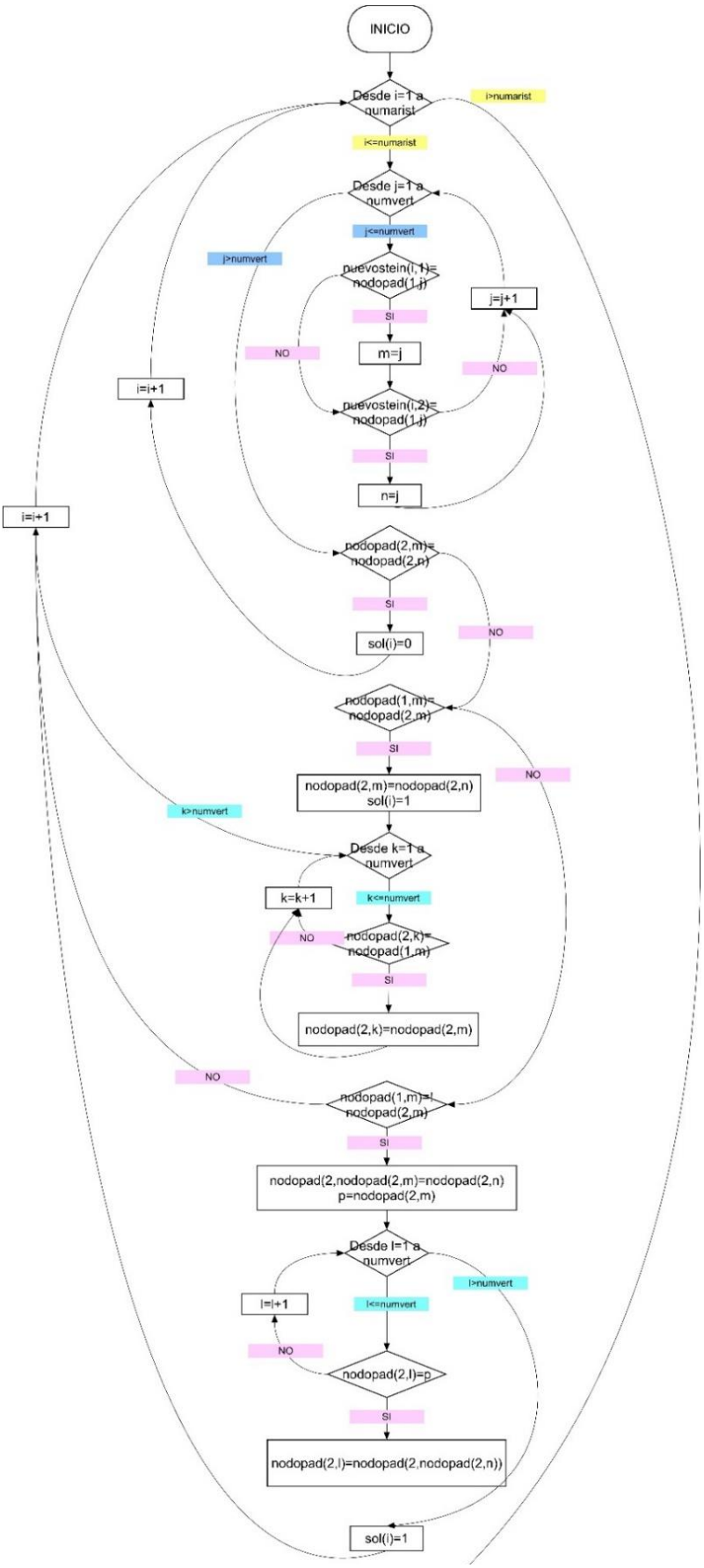


Figura 51. Diagrama de flujo de la función 'solucionado' parte I

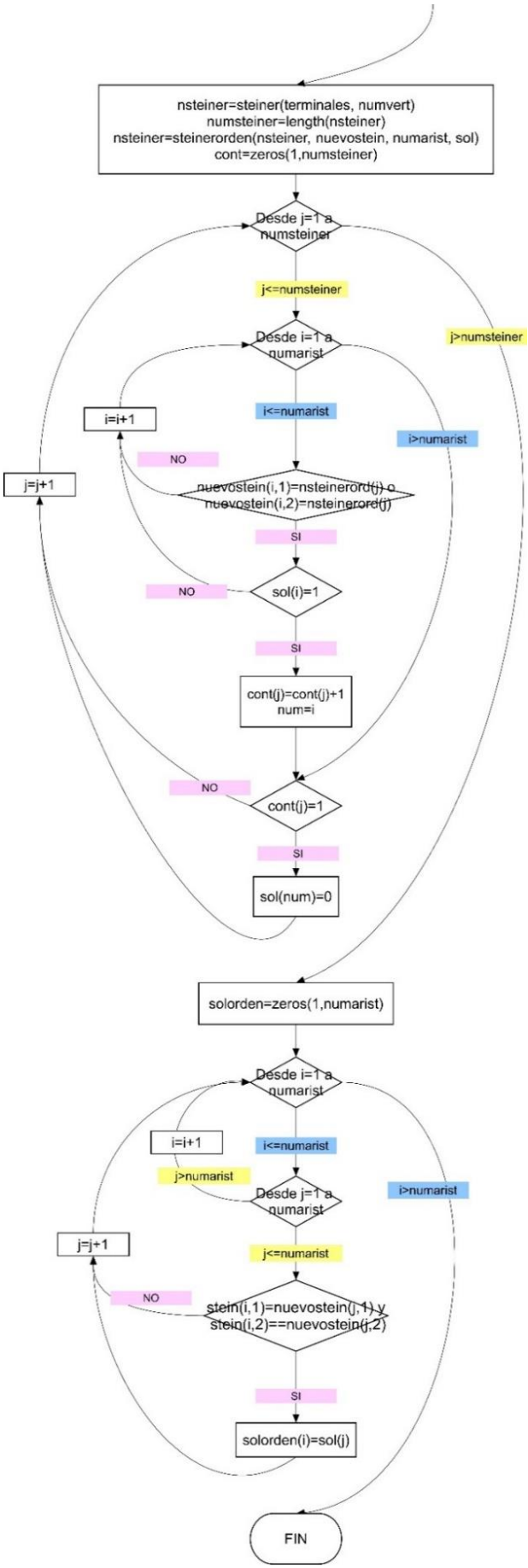


Figura 52. Diagrama de flujo de la función 'solucionado' parte II

5.5 Función ‘costealeatorio’

En esta sección se trata la función más corta de todas las presentes. Esta función solo necesita un total de 4 variables como se podrá ver. La sección se encuentra dividida en objetivo, variables de la función, pseudocódigo y diagrama de flujo.

5.5.1 Objetivo

Con la función ‘costealeatorio’ se pretende poder generar otros costes diferentes al problema inicial con la intención de crear nuevas soluciones admisibles del problema de Steiner. Para la comentada función, solo serán necesarios los parámetros “stein” y “numarist” que servirán para dar como resultado la variable “steincost”.

5.5.2 Variables de la función

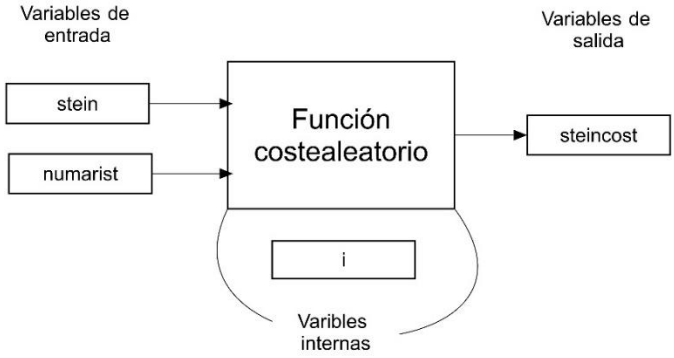


Figura 53. Variables de la función ‘costealeatorio’

5.5.3 Pseudocódigo

La variable con mayor relevancia es:

- steincost: recoge las aristas del problema inicial con los costes cambiados aleatoriamente

Algoritmo 6: Pseudocódigo de la función ‘costealeatorio’

Función [steincost]=costealeatorio(stein, numarist)

Mientras i<=numarist

stein(i,3) es igual a un número entero aleatorio entre el 1 y el 15

Implementar i

Fin Mientras

Copiar matriz stein en steincost

Fin Función

5.5.4 Diagrama de flujo

Se presentan en la Figura 54 el diagrama de flujo de la función 'costealeatorio', como se muestra se trata del diagrama más sencillo del programa. Con un pequeño bucle se cambian los costes de los arcos.

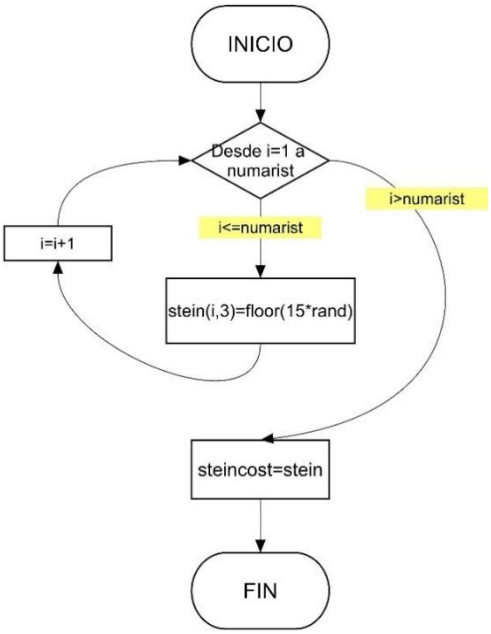


Figura 54. Diagrama de flujo de la función 'costealeatorio'

5.6 Función 'definitivo'

Esta función proporciona exclusivamente dos matrices que serán primordiales para el desarrollo del programa. En los subapartados de esta sección, se analiza en profundidad la función de estudio.

5.6.1 Objetivo

La función 'definitivo' proporciona dos matrices muy importantes en la resolución final del problema; "matrizsol" y "costesol". La variable "matrizsol" contiene N filas donde en cada una de ellas se muestra cada una de las soluciones admisibles del problema. Por otro lado, la matriz "costesol" guarda los costes de cada arista utilizados para generar las distintas soluciones. Mediante los parámetros iniciales más la inclusión del número de soluciones admisibles que se quieren formar, se obtienen las dos matrices; "matrizsol" y "costesol", que son el objeto de la función. Aparte, se hace uso de la función 'solucionado' y 'costealeatorio'.

5.6.2 Variables de la función

Se muestran las variables de entrada, salida e internas de la función 'definitivo' en la Figura 55.

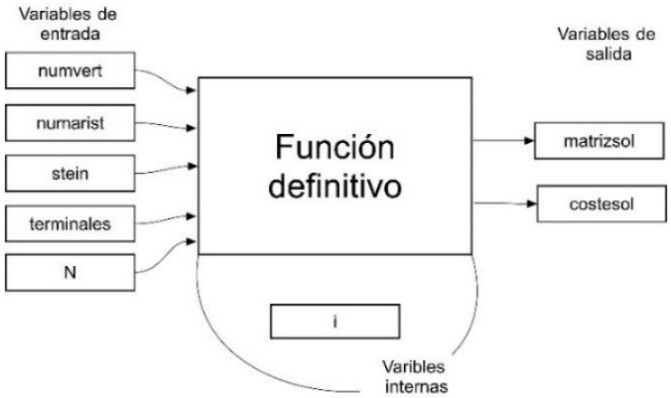


Figura 55. Variables de la función'definitivo'

5.6.3 Pseudocódigo

Las variables importantes de la función 'definitivo' para poder entender el pseudocódigo son:

- N: número de soluciones admisibles que se generarán para implementar el algoritmo genético
- matrizsol: matriz con las N soluciones admisibles
- costesol: matriz con los costes aleatorios generados de cada solución

Algoritmo 7: Pseudocódigo de la función 'definitivo'

Función [matrizsol, costesol]=definitivo(numvert, numarist, stein, terminales, N)

Generar matriz "matrizsol" llena de ceros de dimensión Nxnumarist

Generar matriz "costesol" de ceros de dimensión numaristxN

Generar vector "vectorsol" con la función solucionado y variables de entrada "numarist", "numvert", "stein" y "terminales"

matrizsol(1, :) es igual al vectorsol(:)

costesol(:, 1) es igual a stein(:, 3)

Mientras i valga entre 2 y N

steincost es igual a costealeatorio(stein, numarist)

costesol(:, i) es igual a steincost(:, 3)

Generar vector "vectorsol" con la función solucionado y variables de entrada "numarist", "numvert", "steincost" y "terminales"

matrizsol(i, :) es igual a vectorsol(:)

Implementar i

Fin Mientras

Fin Función

5.6.4 Diagrama de flujo

El diagrama de flujo de la Figura 56 esquematiza el funcionamiento del pseudocódigo. Se muestra como a partir de la definición de unos cuantos parámetros y de la utilización de un bucle, es posible obtener las dos matrices que son objeto de la función.

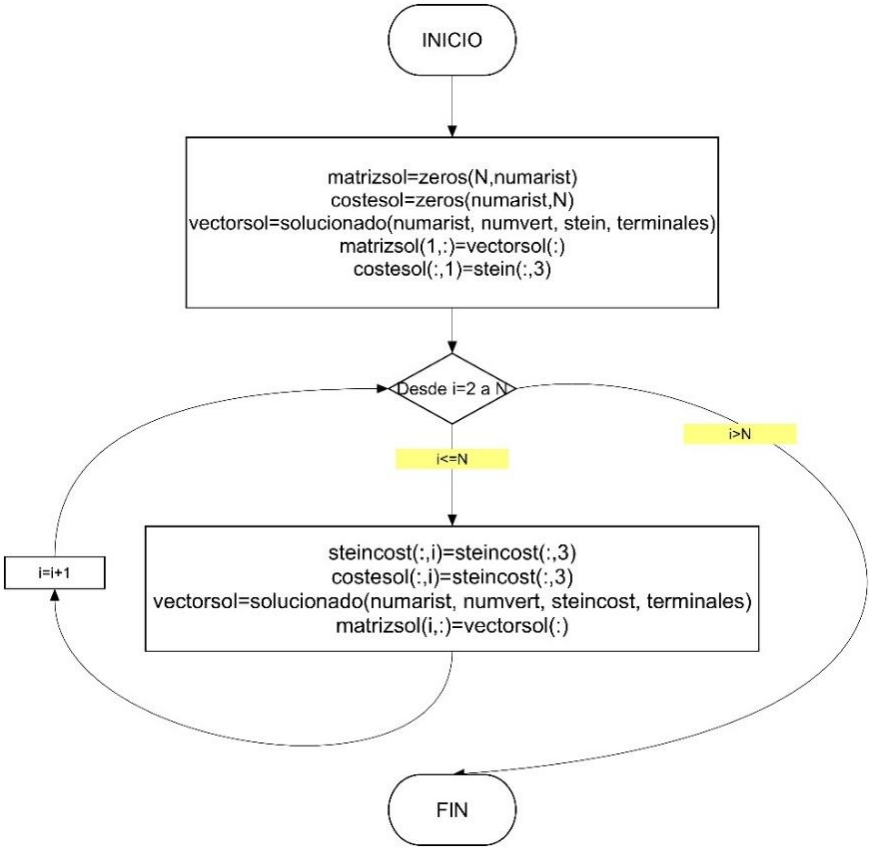


Figura 56. Diagrama de flujo de la función 'definitivo'

5.7 Función 'crossing'

Mediante la función destinada al entrecruzamiento será posible obtener un vector solución que previamente hay que forzar su admisibilidad haciendo uso del MST como se indicaba en la sección 4.2.2.3. Al igual que en el resto de funciones, se comienza la sección con el objetivo.

5.7.1 Objetivo

La función 'crossing' se encarga de obtener un vector solución "hijo" a partir del entrecruzamiento de los vectores "padre" y "madre". Cuando ambos progenitores sean 1, el hijo también valdrá 1 y cuando valgan 0, el hijo valdrá 0. En cambio, cuando los progenitores contengan valores distintos, podrá valer la celda correspondiente del vector "hijo" 1 o 0, el cual se determina aplicando el árbol de distancia mínima para asegurar admisibilidad. La Figura 57 sirve de ejemplo para el entendimiento de la función.

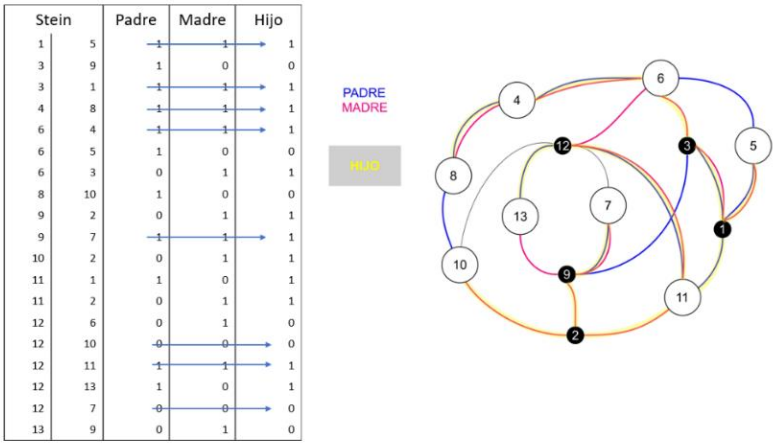


Figura 57. Obtención del vector "hijo" a partir de la reproducción

La función 'solucionado' tiene un rol crucial dentro de la función 'crossing' ya que se encargará de validar la admisibilidad de la solución hijo. Para entender el funcionamiento de la función en cuestión, se muestra a continuación como se forman los costes de la matriz "stein" proporcionada a la función 'solucionado' para dar admisibilidad a la variable "hijo".

5.7.1.1 Planteamiento

Para poder aplicar el árbol de mínima distancia, habrá que fijar unos costes; las celdas del vector "hijo" que contengan 1 tendrán mínimo coste (deberán estar dentro de la solución), las celdas que valgan 0 tendrán el coste más elevado que es 16 (ya que no forman parte de la solución) y las celdas del vector "hijo" donde se puede elegir 1 o 0, obtendrán un coste aleatorio entre 1 y 15. Así el propio algoritmo del árbol de distancia mínima determinará el valor de las celdas del vector "hijo" donde hay opción de elegir entre 1 o 0 sin formar bucles que puedan empeorar el resultado de la solución.

Por eso, a partir de definir los costes, según las instrucciones indicadas, se obtiene aplicando el MST el vector "hijo" con la garantía de que no existen caminos redundantes en la solución. Un ejemplo de ello se muestra en la Tabla 3.

Tabla 3. Cómo pasar del vector 'hijo' a la obtención de los costes para aplicar el MST

Hijo	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**0**
Coste para aplicar MST	1	13	1	1	1	5	9	16	3	1	11	6	10	8	16	1	3	16	12

5.7.2 Variables de la función

En la Figura 58 se muestran las variables utilizadas para la implementación de la función.

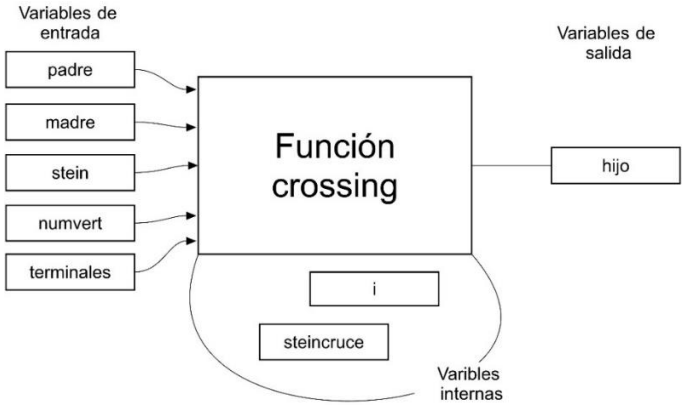


Figura 58. Variables de la función 'crossing'

5.7.3 Pseudocódigo

Las variables importantes para la comprensión del código son:

- padre y madre: vectores progenitores con soluciones diferentes
- steincruce: matriz de dimensiones 3 x número de arcos, que contiene los costes para ajustar el vector solución "hijo"

Algoritmo 8: Pseudocódigo de la función 'crossing'

Función [hijo]=crossing(padre, madre, stein, numvert, terminales)

Generar variable "numarist" que contiene las dimensiones del "padre"

Generar vector "hijo" con dimensiones "numarist" repleta de ceros

Generar matriz "steincruce" de dimensiones 3x "numarist" donde las dos primeras columnas son iguales que la matriz "stein" y la última columna está llena con el número 16

Mientras i<=numarist

Si padre(i)=madre(i) y padre(i)=0

hijo(i)=0

Sino

hijo(i)=1 y steincruce(i,3)=1

Fin Si

Si padre(i) distinto madre(i)

hijo(i)=1 y steincruce(i,3)= número aleatorio entre 1 y 15

Fin Si

Fin Mientras

hijo=solucionado(numarist, numvert, steincruce,terminales)

Fin Función

5.7.4 Diagrama de flujo

En la Figura 59 se muestra el diagrama de flujo de la función 'crossing'. Un bucle de gran tamaño será el dominante del diagrama. A su vez, en el interior de este bucle se muestran tres comandos "*if*".

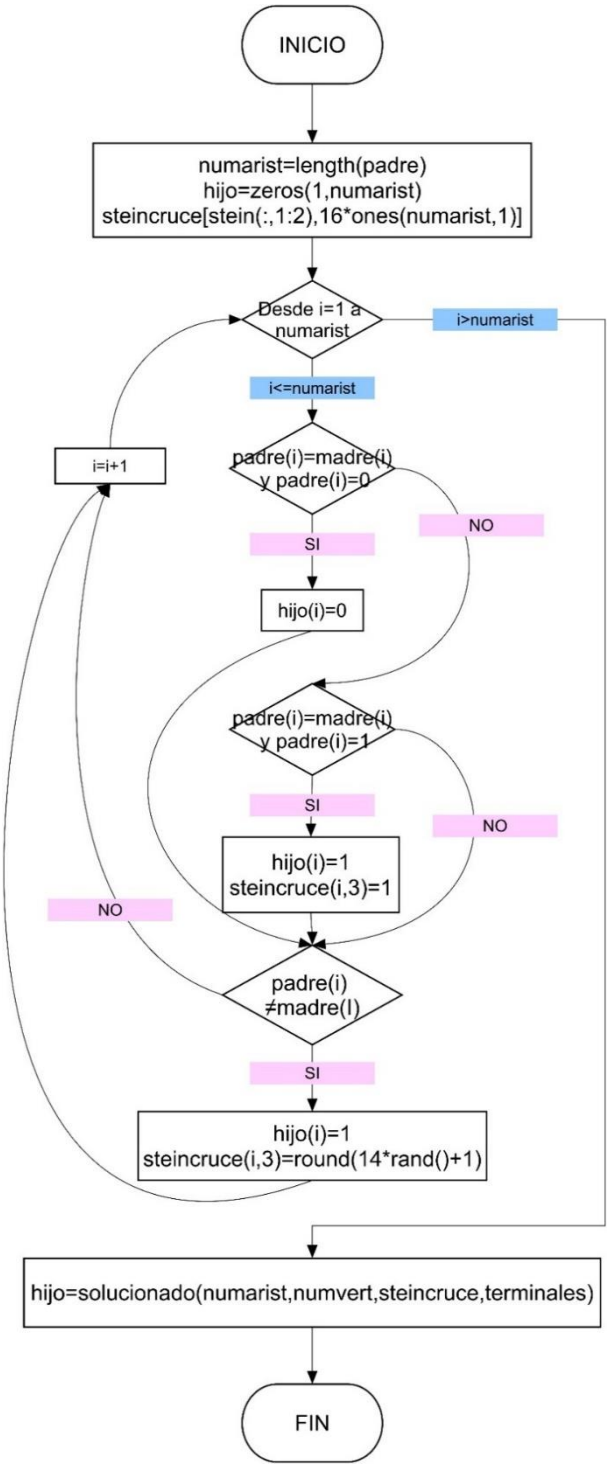


Figura 59. Diagrama de flujo de la función 'crossing'

5.8 Función ‘mutacion’

A continuación, se presenta la función que permite realizar cambios aleatorios en los genes de los individuos. El objetivo, las variables de la función, el pseudocódigo y un diagrama de flujo son los subapartados que dan paso al entendimiento de la función.

5.8.1 Objetivo

Mediante esta función se pretende mutar un gen que valga 1 a 0 de una de las soluciones de la matriz “matrizsol”. El individuo de la población se elige, o bien aleatoriamente, o mediante ranking o ruleta de reemplazo. Dando mayor probabilidad para mutar a aquellas soluciones que tengan peor “fitness” y, por tanto, un coste total de las aristas elevado. Posteriormente, para asegurar la admisibilidad de la solución será necesario aplicar el MST. Para ello, se hace inclusión de nuevo de la función ‘solucionado’.

5.8.1.1 Planteamiento

Una vez elegida la solución y el gen que mutar, se deberá utilizar el MST. Para poder aplicar el árbol de mínima distancia, habrá que fijar unos costes; las celdas de la solución a mutar que contengan 1 tendrán mínimo coste (deberán estar dentro de la solución), las celdas que valgan 0 tendrán un coste aleatorio entre 1 y 15 y la celda elegida para ser mutada tendrá el mayor coste, en este caso, infinito o simplemente 16. Esto es debido, a que el gen mutado a 0 no seguirá formando parte de la solución. Finalmente, aplicando el MST se obtiene la solución mutada que cumple admisibilidad en la solución.

5.8.2 Variables de la función

A continuación, en la Figura 60 se presentan las variables empleadas en la función ‘mutacion’.

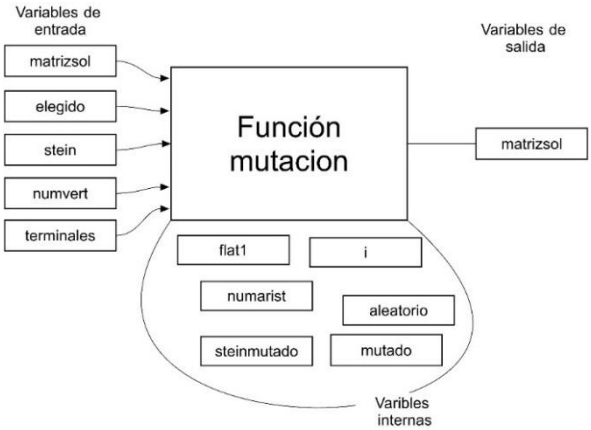


Figura 60. Variables de la función ‘mutacion’

5.8.3 Pseudocódigo

Las variables de interés son las siguientes:

- mutado: vector con la solución a mutar
- steinmutado: vector con aristas y costes preparados para poder formar la solución admisible
- flat1: bandera que puede valer 1 o 0 en función de si se ha encontrado aleatoriamente un gen dentro del individuo que valga 1 para mutar a 0
- elegido: determina el tipo de selección del individuo a mutar un gen, que puede ser aleatoriamente, ruleta o ranking

Algoritmo 8: Pseudocódigo de la función 'crossing'

Función [matrizsol]=mutacion(matrizsol, elegido, stein, numvert, nterminales)

Generar variable "numarist" mediante la dimensión de las filas de "matrizsol"

Generar variable "flat1" igual a 0

Generar vector "mutado" repleto de ceros de dimensiones "numarist"

"mutado" es igual a solución de "matrizsol" elegida para ser mutada

Mientras flat1 igual a 0

aleatorio es un número entre 1 y "numarist" elegido de forma aleatoria

Si matrizsol(elegido, aleatorio)=1

flat1=1

Fin Si

Fin Mientras

Generar matriz "steinmutado" de dimensiones "numarist" x 3 llena de ceros

Poner las dos primeras columnas de "steinmutado" iguales al vector "stein"

Mientras i<= numarist

Si mutado(1, i)= 1

steinmutado(i, 3)= 1

Sino

steinmutado(i, 3)= número aleatorio entre 1 y 15

Fin Si

Implementar i

Fin Mientras

steinmutado(aleatorio, 3)= infinito

matrizsol(elegido, :)= solucionado(numarist, numvert, steinmutado, nterminales)

Fin Función

5.8.4 Diagrama de flujo

En la Figura 61 se da paso al diagrama de flujo compuesto por 4 comandos "*if*" y un solo bucle "*for*". A pesar de la sencillez que puede desprender el objetivo de esta función, todo lo contrario se muestra en el diagrama de flujo.

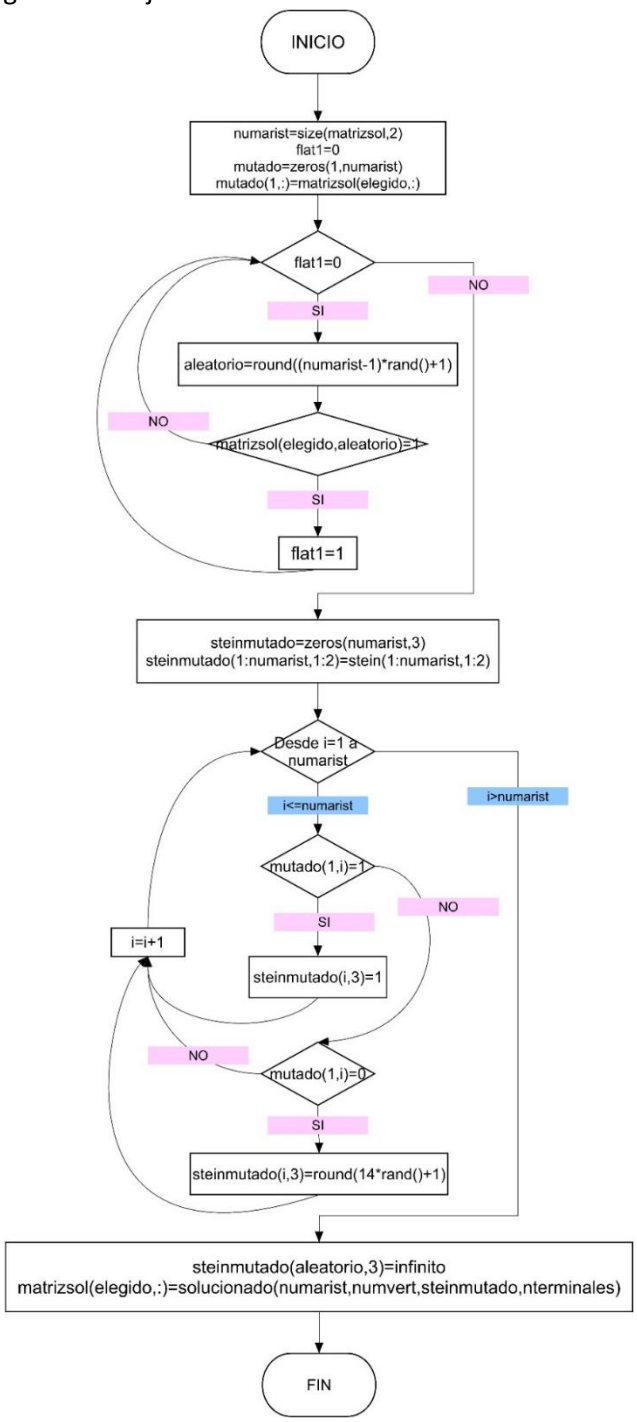


Figura 61. Diagrama de flujo de la función 'mutacion'

5.9 Función ‘seleccion’

En esta función se encuentran todas las formas posibles de selección de soluciones; ruleta, ranking y aleatorio. El apartado esta dividido en objetivo variables de la función, pseudocódigo y diagrama de flujo.

5.9.1 Objetivo

Se pretende obtener la capacidad de elegir entre las soluciones de la matriz “matrizsol” un individuo aleatoriamente o por selección/ reemplazo ranking o ruleta. Para ello, habrá primero que indicar al programa mediante la variable de entrada “bandera”, que tipo de selección se quiere realizar. Se puede elegir entre: bandera igual a 0 (aleatoriamente), bandera igual a 1 (selección por ruleta), bandera igual a 2 (selección por ranking), bandera igual a 3 (reemplazo por ruleta) o bandera igual a 4 (reemplazo por ranking). A continuación, se muestra el funcionamiento del programa y sus variables.

5.9.2 Variables de la función

Aquí se muestran las variables de la función en la Figura 62.

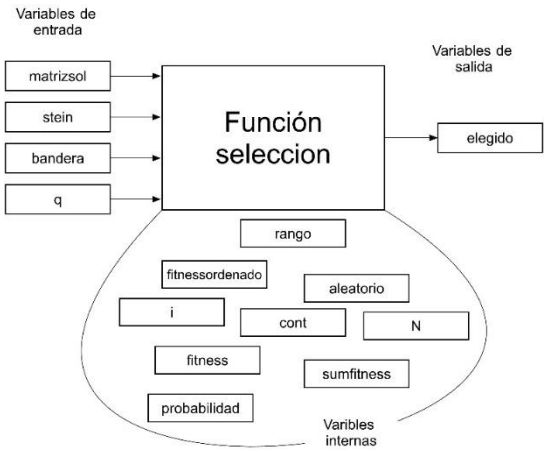


Figura 62. Variables de la función ‘seleccion’

5.9.3 Pseudocódigo

Las variables de interés son:

- elegido: número de fila de la matriz “matrizsol” que va a ser elegido para ser mutado, para ser individuo progenitor o para ser reemplazado
- bandera: indica el tipo de selección elegida para determinar la variable “elegido”
- q: probabilidad de la primera solución en la selección por ranking
- rango: matriz imprescindible para acumular las probabilidades y crear rangos de estas por individuo
- probabilidad: vector que contiene las probabilidades de cada solución elegidas por ruleta, ranking o aleatoriamente

- fitness: vector que contiene el coste total de cada solución
- sumfitness: suma los valores de "fitness" o 1/ "fitness"
- fitnessordenado: ordena la matriz "fitness" en función de los costes de menor a mayor

Algoritmo 9: Pseudocódigo de la función 'selección'

Función [elegido]=selección(matrizsol, stein, bandera, q)

Generar "N" igual al número de soluciones dentro de la matriz "matrizsol"

Generar "fitness", matriz de dos columnas la primera llena de números de 1 a "N" y la otra matrizsol(:,) x stein(:, 3)

Generar "fitnessordenado", matriz de dos columnas; la primera llena de números de 1 a "N" y la otra con los valores de la segunda columna de "fitness" ordenados de menor a mayor

Mientras i<=N

Mientras i<=N

Si fitness(j, 2)= fitnessordenado(i, 2)

Fitnessordenado(i,1)= fitness(j, 1)

Fin Si

Implementar i

Fin Mientras

Implementar i

Fin Mientras

Si bandera= 0 // aleatorio

elegido= número aleatorio entre 1 y "N"

Sino

Generar "probabilidad" igual a un vector lleno de ceros de dimensión 1 x "N"

Si bandera= 1 // ruleta de selección

Generar "sumfitness" igual a 0

Mientras i<= N

sumfitness= sumfitness+1/fitness(i, 2)

Implementar i

Fin Mientras

Mientras i<= N

probabilidad(i)= 1/fitnessordenado(i, 2)/ sumfitness

Implementar i

Fin Mientras

Fin Si

Si bandera= 2 o bandera= 4 // ranking de selección

probabilidad(1)= q

Mientras i<=N-1

probabilidad(i+1)= $q\ x\ (1-q)$^1

Implementar i

Fin Mientras

probabilidad(1)= $q + (1 - sum(probabilidad))$

Fin Si

Si bandera= 4 // ranking de reemplazo

probabilidad= ordenar probabilidades de menor a mayor

Fin Si

Si bandera= 3 // ruleta de reemplazo

sumfitness igual a 0

Mientras i<=N

sumfitness= sumfitness + fitness(i, 2)

Implementar i

Fin Mientras

Mientras i<=N

probabilidad(i)= fitnessordenado(i, 2)/sumfitness

Implementar i

Fin Mientras

Fin Si

Generar "rango" matriz de dimensiones N x 2 llena de ceros

Generar "cont" igual a 0

Mientras i<= N

cont= cont + probabilidad(i-1)

rango(i, 1)= cont

rango(i-1, 2)= rango(i, 1)

Implementar i

Fin Mientras

rango(N, 2) igual a 1

Generar “aleatorio” igual a número entre 0 y 1

Mientras i<= N

Si aleatorio>= rango(i, 1) y aleatorio< rango(i, 2)

aleatorio< rango(i, 2)

elegido= fitnessordenado(i, 1)

Fin Si

Implementar i

Fin Mientras

Fin Si

Fin Función

5.9.4 Diagrama de flujo

En las siguientes figuras se muestra el diagrama de flujo completo para la función ‘selección’. Se ve fraccionado debido a las dimensiones de este. Se pueden apreciar 7 comandos “*if*” repartidos por las dos figuras.

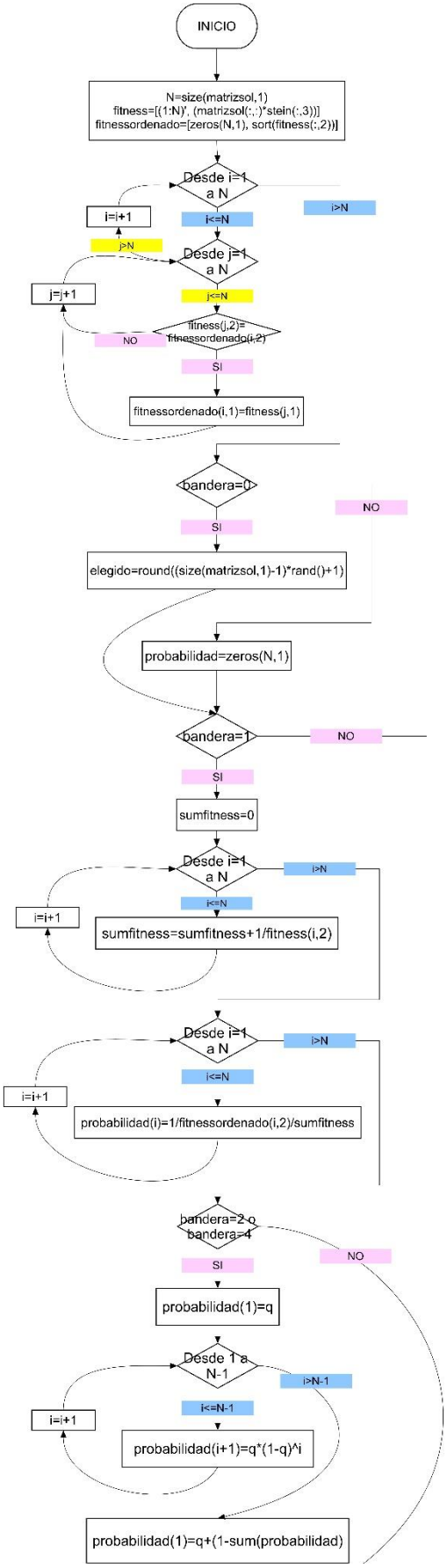


Figura 63. Diagrama de flujo de la función 'seleccion' parte I

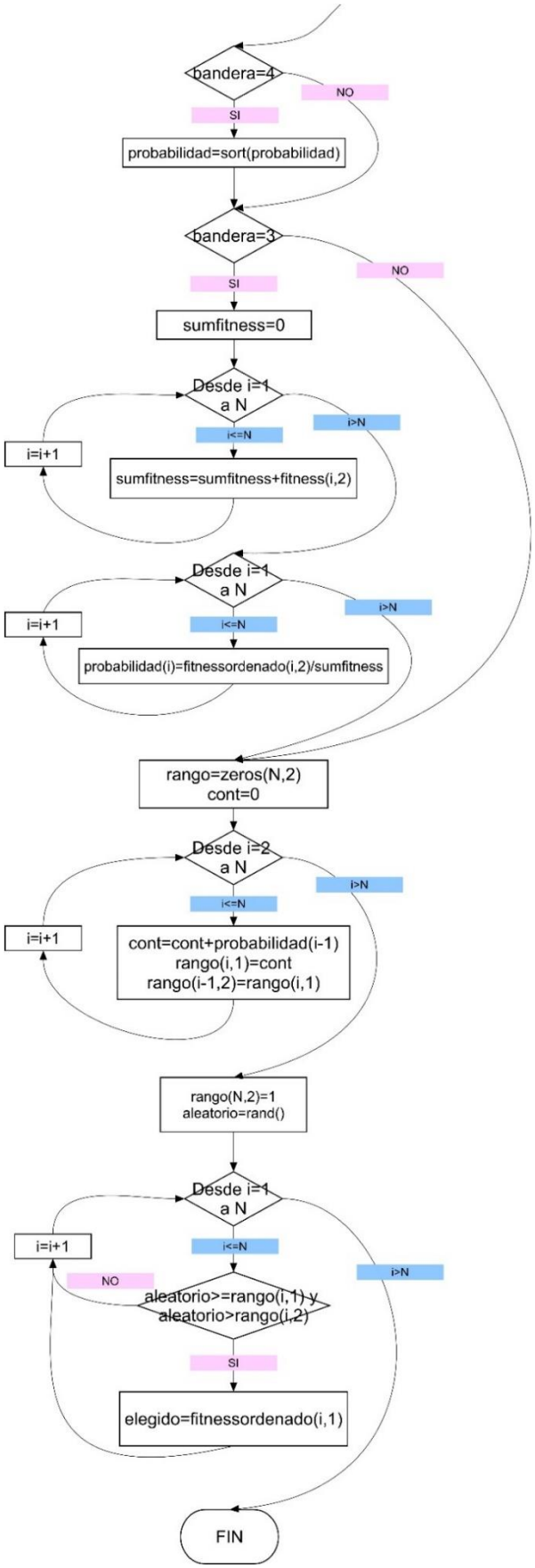


Figura 64. Diagrama de flujo de la función ‘seleccion’ parte II

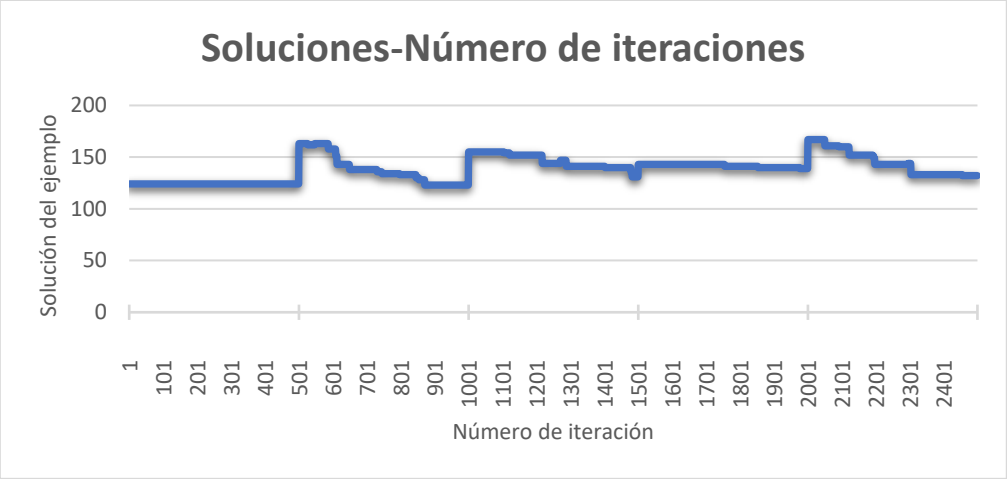
5.10 Función ‘genetico1’

La función ‘genetico1’ se encarga de dirigir el funcionamiento del programa. Es la función que da orden al resto de funciones para su cooperación y coordinación. A continuación, se muestra detalladamente el objetivo de la misma junto con sus peculiaridades.

5.10.1 Objetivo

El propósito de esta función es obtener la solución admisible más cercana al óptimo o el óptimo del árbol de Steiner a partir de; un número de iteraciones propuesto (parámetro “iteraciones”), de un número de individuos (parámetro “N”) y de los parámetros iniciales (datos de entrada). La función implementa 30 poblaciones de las cuales se extrae la mejor solución. Se ha optado por generar 30 poblaciones en vez de una para asegurar una buena exploración del espacio de soluciones. De este modo, si se tiene N=100 e iteraciones=1000, cada población de las 30 generadas tendrá 100 individuos y se implementará cada una de ellas 1000 veces, resultando en 30000 iteraciones antes de proporcionar la solución final.

Mediante las iteraciones realizadas dentro de una población, se obtienen soluciones con un coste cada vez más pequeño debido a las mutaciones y reproducciones llevadas a cabo que van extinguiendo a los individuos menos adaptados. Con ello, se define en una gráfica el coste de la mejor solución en cada una de las iteraciones obteniendo lo siguiente:



Gráfica 1. Espacio de soluciones proporcionado en las diferentes poblaciones implementadas

Se representan en la Gráfica 1 cinco poblaciones de 500 iteraciones donde cada punto de la gráfica representa la mejor solución de cada una de las iteraciones. Se pueden apreciar saltos entre población y población debido a que se estudian un conjunto completamente diferente de individuos o lo que es lo mismo, unas soluciones diferentes por cada población. El coste de la solución final según la gráfica será el que más bajo esté en el eje de ordenadas (el de más bajo coste). Para este ejemplo, el coste más bajo es de 123 unidades.

5.10.2 Variables de la función

En la Figura 65 se muestran las variables de la función 'genetico1'.

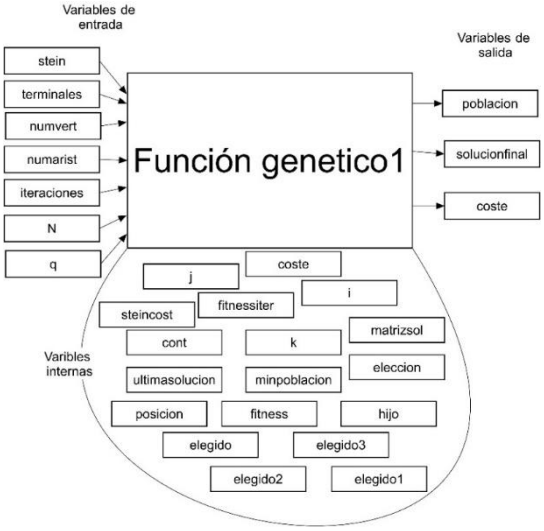


Figura 65. Variables de la función 'genetico1'

5.10.3 Pseudocódigo

Las variables de interés son las siguientes:

- población: se trata de una estructura con 3 variables llamadas "población.fitness", "población.minimo" y "población.solucion" cada una de ellas con tantos campos como poblaciones.

Para un ejemplo de un problema con 5 poblaciones y 8 iteraciones por cada población se tienen los siguientes resultados mostrados en la Tabla 4:

Tabla 4. Ejemplo particular de la estructura población de un árbol de Steiner

poblacion.fitness	poblacion.minimo	poblacion.solucion
[53 60 55 58 61 65 52 59]	52	[1 0 1 1 0 1 1 0 0 0 1 1 1]
[61 49 63 54 56 65 63 51]	49	[1 0 0 0 1 0 0 1 1 1 1 1 0]
[55 53 47 48 63 61 50 64]	47	[0 0 0 1 0 1 1 1 0 1 1 0 1]
[59 61 63 54 52 58 57 61]	52	[0 1 1 1 0 0 1 0 0 0 1 0 1]
[50 60 55 54 49 45 57 52]	45	[1 0 1 1 1 1 0 1 0 0 0 0 0]

Donde:

vector población(i).fitness: guarda los mínimos costes de cada iteración en la población i

variable población(i).minimo: contiene el mejor coste de la población i

vector población(i).solucion: expone la mejor solución de la población i

- fitnessiter: vector que se modifica en cada población que contiene el mejor fitness en cada una de las iteraciones
- elección: es un número aleatorio entre 0 y 1
- elegido, elegido1, elegido2, elegido3: números obtenidos según la función 'selección'
- hijo: solución obtenida entrecruzando dos padres
- fitness: vector que contiene los costes de cada solución de la población i que se modifica cada iteración
- ultimasolucion: mejor solución de la población i
- minpoblacion: vector que contiene el mejor coste de cada población
- steincost: se genera cada población y es la variable que permite generar individuos nuevos
- coste: guarda el coste de la mejor solución
- posicion: variable auxiliar
- solucionfinal: brinda la solución óptima o la más cercana a él

Algoritmo 10: Pseudocódigo de la función 'genetico1'

Función [solucionfinal, coste, poblacion]=genetico1(numvert, numarist, stein, terminales, N, q, iteraciones)

Generar tic

Generar matriz "matrizsol"= definido(numvert, numarist, stein, terminales, N)

Generar vector "fitnessiter" de dimensiones "iteraciones" relleno de ceros

Generar estructura "poblacion"=struct('fitness', [], 'minimo', [], 'solucion', [])

Mientras i<= "iteraciones"

elección= aleatorio de 0 a 1

Si elección>=0 y elección< 0.15

elegido=seleccion(matrizsol, stein, 0, q)

matrizsol=mutacion(matrizsol, elegido, stein, numvert, terminales)

Fin Si

Si eleccion>= 0.15

elegido1=seleccion(matrizsol, stein, 1, q)

elegido2=seleccion(matrizsol, stein, 1, q)

hijo=crossing(padre, madre, stein, numvert, terminales)

elegido3=seleccion(matrizsol, stein, 4, q)

matrizsol(elegido3, :)=hijo(:)

Fin Si

Generar matriz "fitness"= [(1:N)',(matrizsol(:, :)*stein(:, 3))]

```
fitnessiter(i) es igual al mínimo de fitness(:, 2)
Implementar i
Fin Mientras
Mientras j<=1
 Si fitness(j, 2)= fitnessiter(iteraciones)
  posicion=j
 Fin Si
Implementar j
Generar vector ultimasolucion= matrizsol(posicion, :)
Generar vector "minpoblacion" de dimensión 30
minpoblacion(1)= min(fitnessiter)
poblacion(1).fitness= fitnessiter
poblacion(1).minimo= minpoblacion(1)
poblacion(1).solucion= ultimasolucion
Mientras j<=29
 matrizsol= definitivo(numvert, numarist, stein, terminales, N)
 steincost= costealeatorio(stein, numarist)
 matrizsol(1, :)= solucionado(numarist, numvert, steincost, terminales)
  Mientras i<= iteraciones
  elección igual a número aleatorio entre 0 y 1
   Si eleccion>= 0 y eleccion< 0.15
    elegido= seleccion(matrizsol, stein, 0, q)
    matrizsol= mutacion(matrizsol, elegido, stein, numvert, terminales)
   Fin Si
   Si eleccion>= 0.15
    elegido1= seleccion(matrizsol, stein, 1, q)
    elegido2= seleccion(matrizsol, stein, 1, q)
    hijo=crossing(padre, madre, stein, numvert, terminales)
    elegido3= seleccion(matrizsol, stein, 4, q)
    matrizsol(elegido3, :)= hijo(:)
   Fin Si
```

fitness= [(1: N), (matrizsol(:, :)* stein(:, 3))]

fitnessiter(i)= min(fitness(:, 2))

Implementar i

Fin Mientras

Mientras k<= N

Si fitness(j, 2)= fitnessiter(iteraciones)

Generar variable posicion igual a k

Fin Si

Implementar k

Fin Mientras

ultimasolucion= matrizsol(posición, :)

minpoblacion(j+1)= min(fitnessiter)

población(j+1).fitness= fitnesiter

población(j+1).minimo= minpoblacion(j+1)

Implementar j

Fin Mientras

Generar coste= min(minpoblacion)

Mientras j<=30

Si minpoblacion(j, 1)= coste

posicion= j

Fin Si

Implementar j

Fin Mientras

solucionfinal= población(posición).solucion

Generar toc

Fin Función

5.10.4 Diagrama de flujo

Se muestra el diagrama de flujo de la función 'genetico1' el cual está dividido en dos figuras debido a su gran dimensión. Se puede ver el denso contenido de las órdenes para el funcionamiento completo del programa.

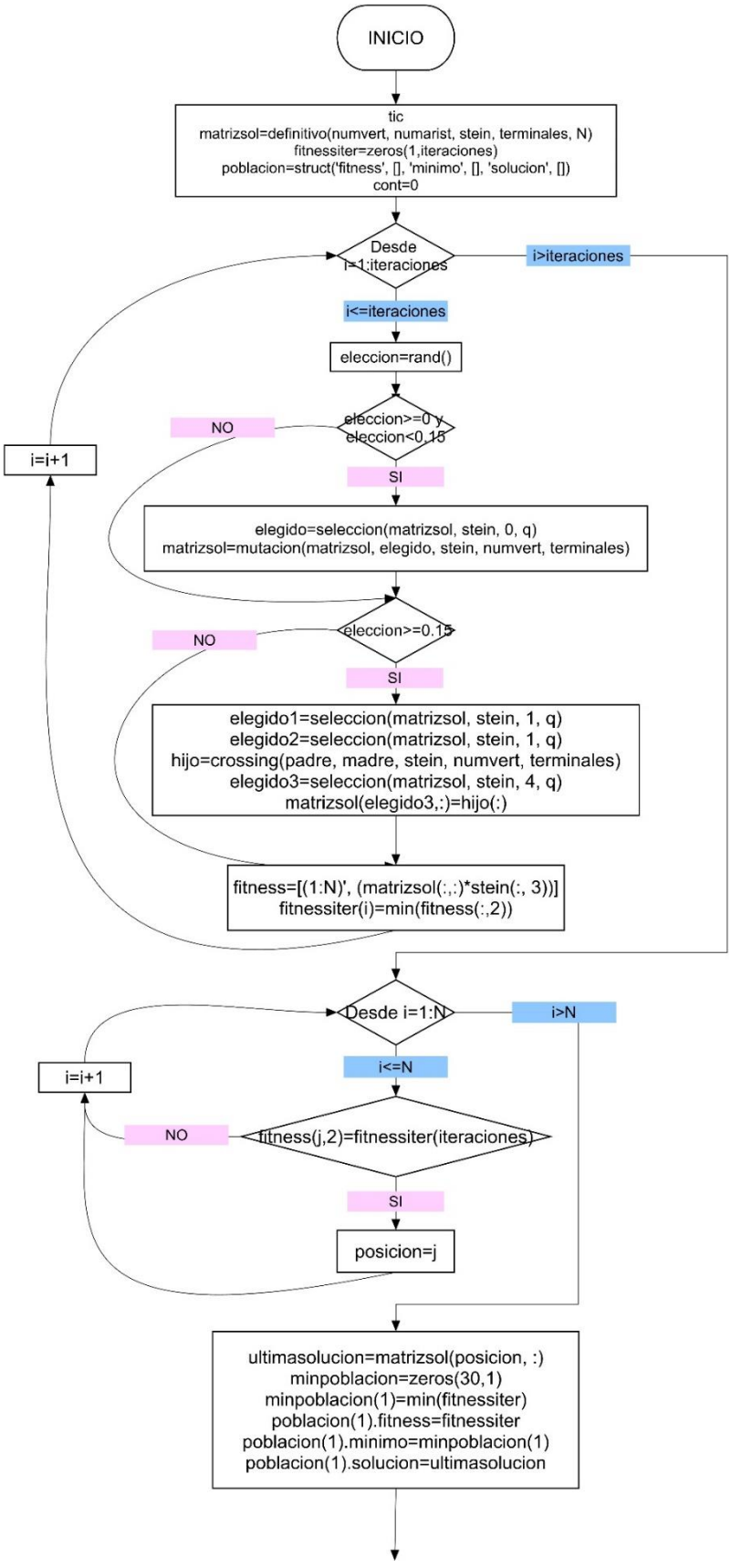


Figura 66. Diagrama de flujo de la función 'genetico1' parte I

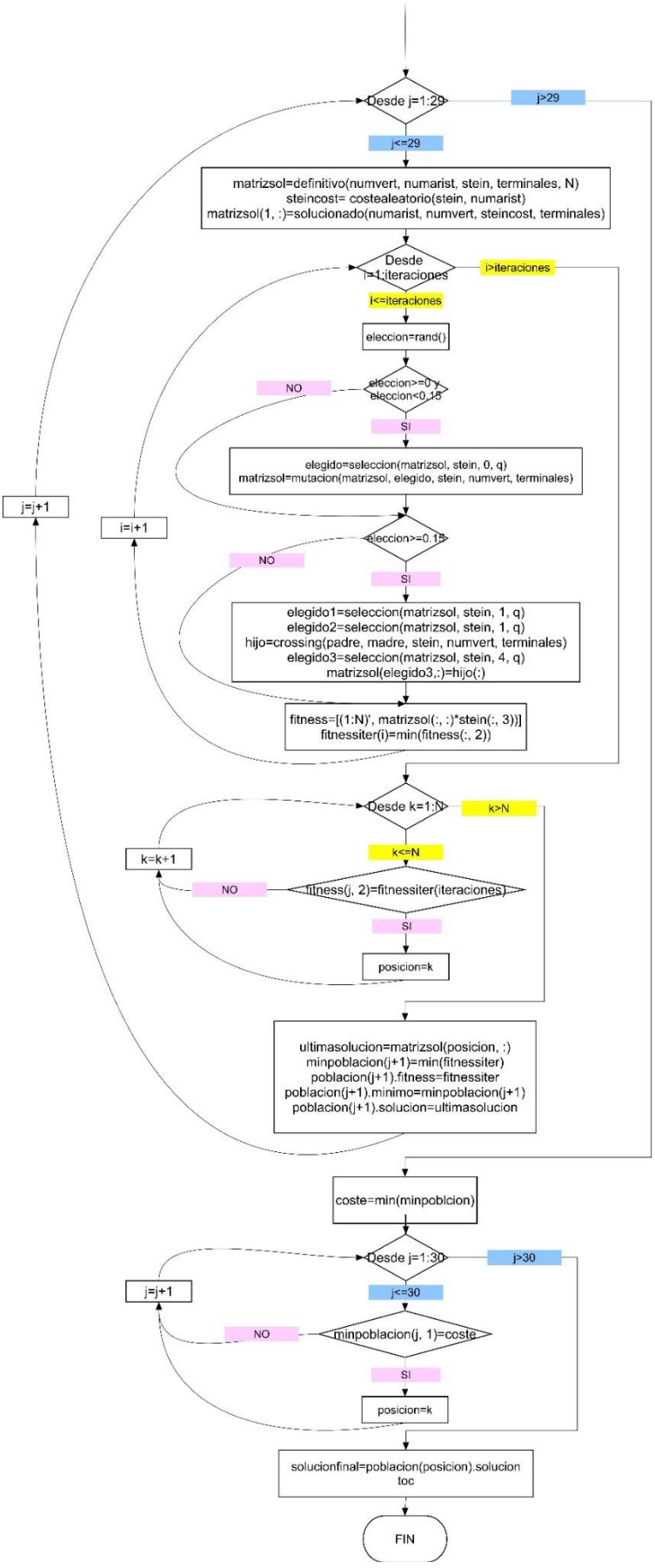


Figura 67. Diagrama de flujo de la función 'genetico1' parte II

6 RESULTADOS

"Lo bueno se hace esperar"

Dicho popular

En este último capítulo, fruto de la investigación y observación de todo lo relatado hasta el momento, se van a exponer una serie de problemas cuyas soluciones son conocidas. La intención es, por tanto, contrastar los óptimos de los problemas de Steiner con las soluciones obtenidas en el algoritmo desarrollado en este documento. Se pretende plasmar la utilidad y precisión de los algoritmos genéticos, así como dar conclusiones en base al estudio realizado en este capítulo.

En primer lugar, la sección 6.1 comienza con las características de los problemas de Steiner de estudio. Durante la sección 6.2, se abrirá paso a los resultados del programa utilizado. Así como se mostrarán las gráficas de evolución y en la sección 6.4 se explican las curvas de superficies.

6.1 Características de los problemas

Para el análisis y la comprobación de la teoría y el algoritmo heurístico desarrollado en este trabajo, se han utilizado los problemas que se encuentran en la "*OR library*" del "*Imperial College*" de Londres. Estos problemas se encuentran clasificados en "b" y "c" en función de su complejidad y dimensión de manera ascendente.

A continuación, para su posterior estudio se presentan en la Tabla 5 y Tabla 6 las características de los problemas de tipo "b" y "c", los cuales van a ser estudiados. Estas son; el número de nodos, el número de arcos, el número de nodos terminales, el óptimo del problema, así como el nombre destinado a cada problema.

Como se puede comprobar, la primera colección alberga 18 problemas, mientras que de la colección steinc se van a analizar los 10 primeros problemas de Steiner. Se centra la atención en estos problemas en particular para obtener resultados de precisión y contundentes.

Tabla 5. Problemas steinb

Problemas	Nº Nodos	Nº Arcos	Nº Terminales	ÓPTIMO
steinb1	50	63	9	41
steinb2	50	63	13	58
steinb3	50	63	25	62
steinb4	50	100	9	59
steinb5	50	100	13	56
steinb6	50	100	25	116
steinb7	75	94	13	74
steinb8	75	94	19	75
steinb9	75	94	38	75
steinb10	75	150	13	71
steinb11	75	150	19	86
steinb12	75	150	38	152
steinb13	100	125	17	128
steinb14	100	125	25	153
steinb15	100	125	50	220
steinb16	100	200	17	117
steinb17	100	200	25	101
steinb18	100	200	50	182

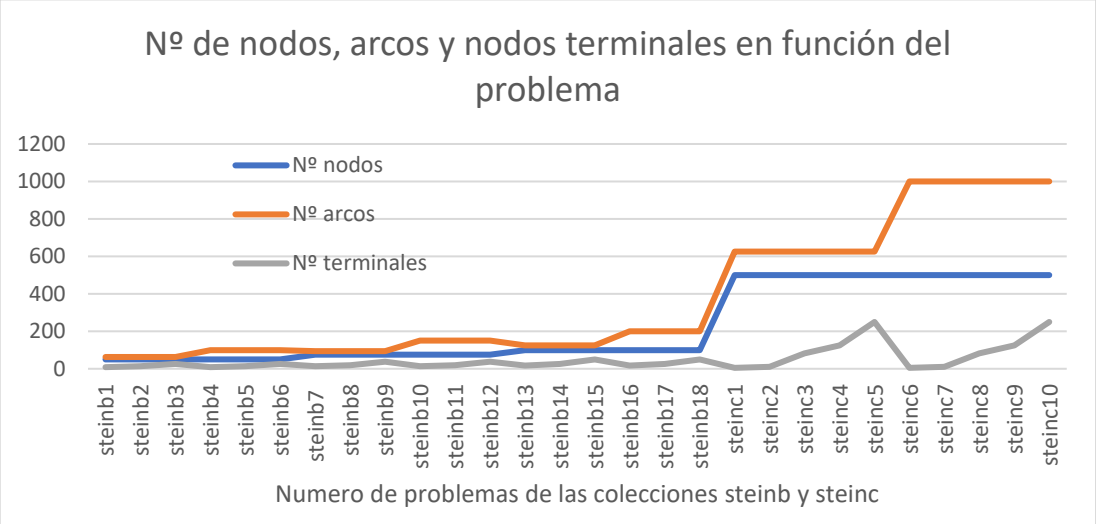
Tabla 6. Problemas steinc

Problemas	Nº Nodos	Nº Arcos	Nº Terminales	ÓPTIMO
steinc1	500	625	5	60
steinc2	500	625	10	137
steinc3	500	625	83	549
steinc4	500	625	125	731
steinc5	500	625	250	960
steinc6	500	1000	5	55
steinc7	500	1000	10	102
steinc8	500	1000	83	459
steinc9	500	1000	125	675
steinc10	500	1000	250	1009

Dichos problemas contienen tanto nodos terminales como nodos Steiner, los cuales pueden tener solamente un grado. De aquí la importancia comentada en la sección 2.2 de que el programa pueda deshacerse de ellos antes de comenzar a iterar y a generar soluciones.

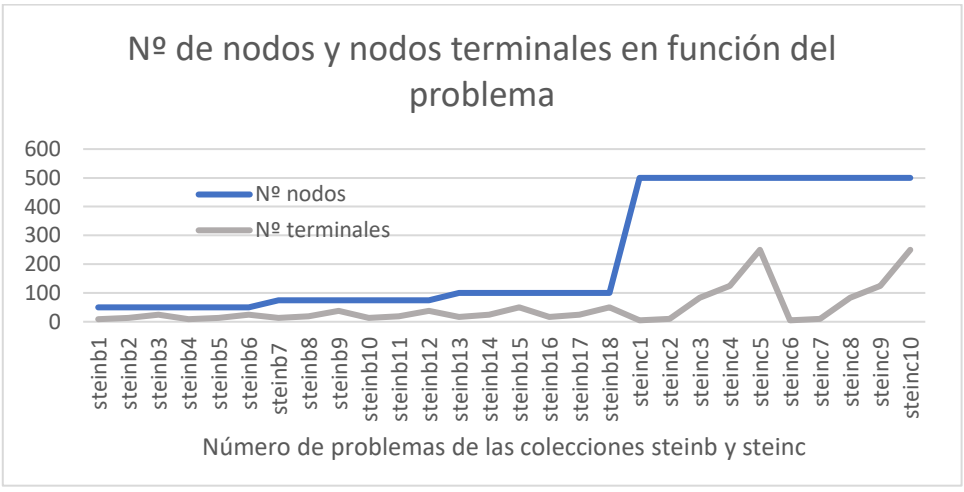
A continuación, en la Gráfica 2 se aprecian el número de nodos, arcos y nodos terminales en función del problema, primero los de tipo “b” y luego los de tipo “c”.

Se puede ver como no existe una relación lineal, pero se muestra una clara tendencia al alza. En especial, el número de arcos va creciendo de forma escalonada muy pronunciada a medida que aumenta el número del problema. Por tanto, su dificultad aumenta. El número de nodos para los problemas steinb oscila de 50 a 100. Mientras que para los problemas steinc se observa una subida en el número de nodos a 500 que se mantiene constante para toda la colección de problemas tipo c.



Gráfica 2. N.º de nodos, arcos y terminales en función del problema

En la Gráfica 3 se puede ver con más precisión la relación de nodos totales y nodos terminales. Aquí se muestra la forma de sierra de los nodos terminales y cómo a partir de los problemas de tipo "c" se pasa de 100 nodos totales a 500. Se puede intuir que la densidad de los nodos terminales (nodos totales entre nodos terminales) también tendrá forma de sierra debido a la forma de los nodos terminales y al ser prácticamente constante los nodos terminales en dos tramos del gráfico.



Gráfica 3. N.º de nodos totales y nodos terminales en relación al tipo de problema

6.2 Resultados de la ejecución

En este apartado se muestran los resultados y simulaciones obtenidos por el programa MATLAB al implementar el algoritmo heurístico desarrollado y el árbol mínimo de expansión con los problemas de Steiner de la "*OR Library*". Todos los resultados han sido realizados con un procesador Intel Core i5 de séptima generación y una memoria RAM de 8GB.

En el subapartado 6.2.1 se muestran los resultados del algoritmo genético y en la subsección 6.2.2 se visualizan los resultados del MST.

6.2.1 Resolución del algoritmo genético

Para poder obtener los resultados del algoritmo genético se ha hecho uso del MST para la generación inicial de soluciones. Es importante recordar que para evitar la prevención de incesto a su vez de para obtener soluciones variadas, en cada problema se generan 30 poblaciones de N soluciones que se iteran un número dado de veces.

En la Tabla 7 se muestra para los problemas de tipo "b", el óptimo, la solución del algoritmo genético, el tiempo de implementación en segundos y el error relativo cometido por el algoritmo genético sobre el óptimo. En el caso de la Tabla 7 se han realizado 500 iteraciones a las 30 poblaciones escogidas de 100 individuos cada uno.

Tabla 7. Resultados del algoritmo genético para los problemas de tipo b

Problema	ÓPTIMO	Sol. Algoritmo genético	Tiempo total (s)	error %
steinb1	41	41	4,072	0,000
steinb2	58	58	4,558	0,000
steinb3	62	62	4,317	0,000
steinb4	59	62	7,776	5,085
steinb5	56	57	7,655	1,786
steinb6	116	123	8,189	6,034
steinb7	74	74	5,226	0,000
steinb8	75	75	5,610	0,000
steinb9	75	75	5,314	0,000
steinb10	71	81	11,336	14,085
steinb11	86	93	12,253	8,140
steinb12	152	168	11,426	10,526
steinb13	128	128	6,554	0,000
steinb14	153	153	7,017	0,000
steinb15	220	222	7,169	0,909
steinb16	117	144	16,536	23,077
steinb17	101	111	14,861	9,901
steinb18	182	191	16,051	4,945

Se muestra como el tiempo de implementación no supera los 20 segundos y como el error más alto es del 23%. Posteriormente se estudiará el porqué de este error tan alto en comparación con el resto de las soluciones.

Tabla 8. Resultados del algoritmo genético para los problemas de tipo c

Problema	OPTIMO	genético	Tiempo total (s)	error %
steinc1	60	101	35,159905	68,3333333
steinc2	137	189	30,628644	37,9562044
steinc3	549	626	41,314931	14,0255009
steinc4	731	787	43,792027	7,66073871
steinc5	960	991	48,52707	3,22916667
steinc6	55	365	243,413428	563,636364
steinc7	102	305	274,191038	199,019608
steinc8	459	617	259,40054	34,422658
steinc9	675	828	280,163543	22,6666667
steinc10	1009	1083	256,90041	7,33399405

Para las soluciones de la Tabla 8 se muestra como el tiempo de implementación aumenta respecto a los problemas de tipo “b”. Además, se puede observar como para el steinc6 se muestra un error muy elevado posiblemente debido al reducido número de nodos terminales.

6.2.2 Resolución del árbol mínimo de expansión (MST)

En este apartado se trata la resolución de los problemas de Steiner a partir del árbol de mínima de expansión.

La Tabla 9 muestra los resultados del MST para los problemas de tipo b. Si con el algoritmo genético 8 de los 18 problemas de tipo “b” se conseguían obtener sin error, con el MST exclusivamente 3 de los 18 problemas da con el óptimo. También se puede volver a comprobar que la resolución del problema steinb16 es el que mayor error presenta con un 35% de error. Lo que supone un 10% de incremento respecto al error con el algoritmo genético. A pesar de ello, los tiempos de resolución del MST están muy por debajo de los del algoritmo genético.

Otro punto a destacar es que el error obtenido con la resolución del algoritmo genético es como mucho igual al error de la resolución por MST.

Tabla 9. Resultados del MST para los problemas de tipo b

Problemas	OPTIMO	MST	Tiempo total (s)	error %
steinb1	41	47	0,004792	14,6341463
steinb2	58	58	0,002532	0
steinb3	62	62	0,000887	0
steinb4	59	79	0,001563	33,8983051
steinb5	56	67	0,002339	19,6428571
steinb6	116	124	0,001004	6,89655172
steinb7	74	80	0,000746	8,10810811
steinb8	75	79	0,000685	5,33333333
steinb9	75	75	0,00074	0
steinb10	71	90	0,001917	26,7605634
steinb11	86	107	0,00121	24,4186047
steinb12	152	168	0,001211	10,5263158
steinb13	128	156	0,001026	21,875
steinb14	153	158	0,001259	3,26797386
steinb15	220	232	0,001165	5,45454545
steinb16	117	159	0,00186	35,8974359
steinb17	101	111	0,001672	9,9009901
steinb18	182	191	0,001545	4,94505495

En la Tabla 10 se muestran los resultados del MST para los problemas de tipo c. Como se puede ver para este tipo de problemas el error aumenta considerablemente.

Tabla 10. Resultados del MST para los problemas de tipo c

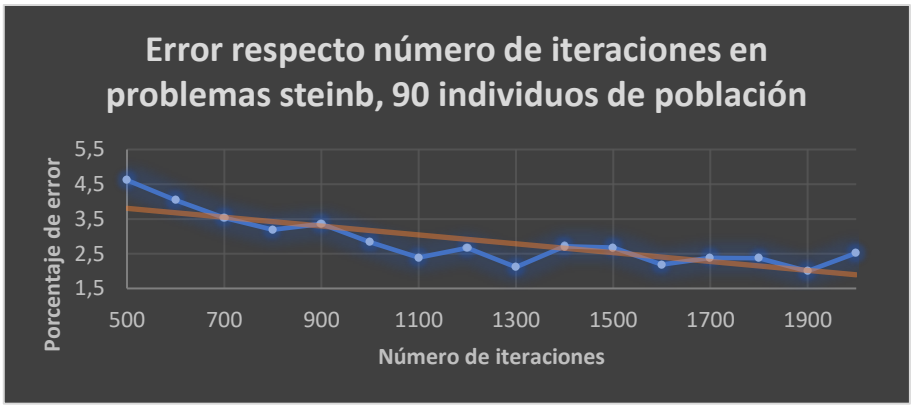
Problemas	ÓPTIMO	MST	Tiempo total (s)	error %
steinc1	60	231	0,006433	285
steinc2	137	239	0,003416	74,4525547
steinc3	549	628	0,003776	14,3897996
steinc4	731	787	0,00627	7,66073871
steinc5	960	991	0,003946	3,22916667
steinc6	55	373	0,015894	578,181818
steinc7	102	401	0,016649	293,137255
steinc8	459	617	0,016685	34,422658
steinc9	675	828	0,017076	22,6666667
steinc10	1009	1083	0,016778	7,33399405

6.3 Gráficas de evolución

En este apartado se pasa a analizar las gráficas obtenidas a partir de los resultados hallados.

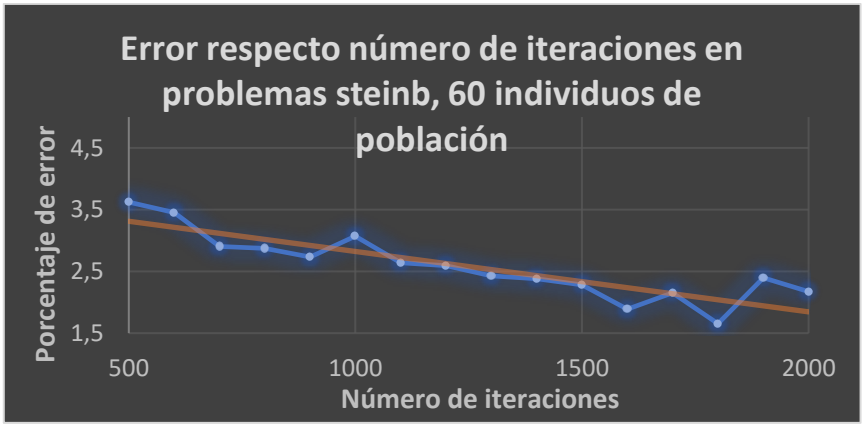
Para comenzar, en la Gráfica 4 se compara el porcentaje de error relativo del algoritmo genético respecto al número de iteraciones para los problemas de tipo "b" con una población de 90 individuos. Como se puede ver, existe una tendencia a la baja a medida que se aumenta el número de iteraciones. Por lo que, a mayor número de iteraciones, se abstrae de la gráfica que el error disminuye progresivamente. Para obtener el error se ha hecho la media de los 18 problemas para cada una de las iteraciones dadas.

La línea continua recta muestra la regresión lineal de la curva.



Gráfica 4. Porcentaje de error respecto del número de iteraciones de los problemas tipo "b" para poblaciones de 90 individuos

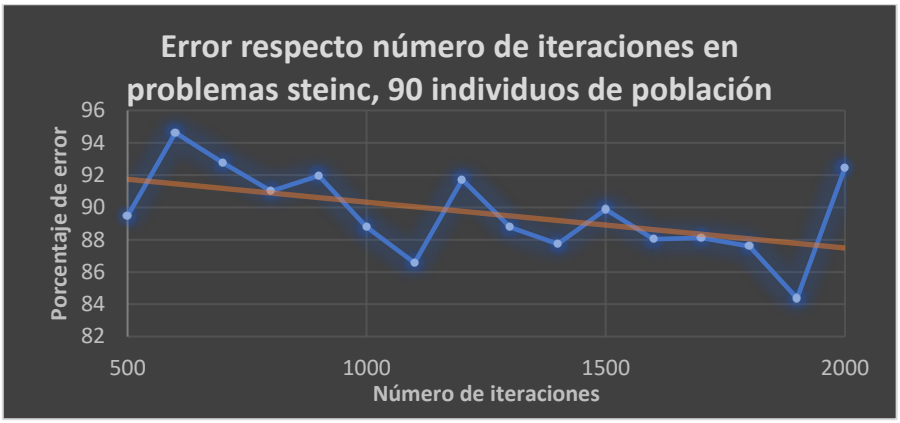
A modo de comparación, se muestra en la Gráfica 5, que para una población de 60 individuos con el resto de mismas características que la gráfica anterior, se obtienen porcentajes más bajos de error. Según la Gráfica 4 y la Gráfica 5, una reducción de la población de 90 individuos a 60 individuos beneficia al resultado del problema de Steiner de tipo "b".



Gráfica 5. Porcentaje de error respecto del número de iteraciones de los problemas tipo "b" para poblaciones de 60 individuos

En la Gráfica 6 se muestra el porcentaje de error frente el número de iteraciones para problemas tipo “c” con una población de 90 individuos. Como se muestra, aquí el error en las iteraciones más bajas llega al 95% mientras que en las iteraciones más elevadas se reduce al 86%. Como sucedía con las gráficas anteriores, se muestra también para el problema de Steiner de la colección steinc una bajada del error conforme aumenta el número de iteraciones.

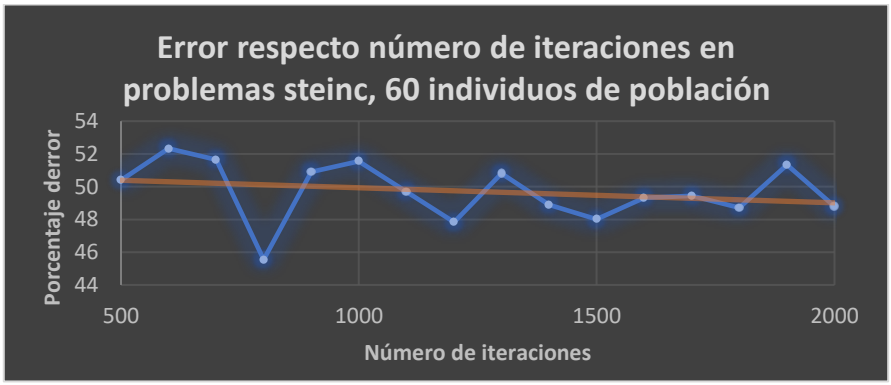
Al igual que en las gráficas análogas para el tipo “b”, se ha obtenido una media de los problemas del 1 al 10 para dar los errores en función de las iteraciones.



Gráfica 6. Porcentaje de error respecto del número de iteraciones de los problemas tipo “c” para poblaciones de 90 individuos

En la Gráfica 7 se tienen los porcentajes de error respecto del número de iteraciones de los problemas de tipo “c” con la peculiaridad de un cambio de población a 60 individuos. Parece sorprendente la brusca disminución del error al disminuir el número de la población. Mientras que antes los errores no bajaban del 85%, en la Gráfica 7 los errores no suben del 53%. Igualmente, se produce una bajada del error conforme se aumenta el número de implementaciones.

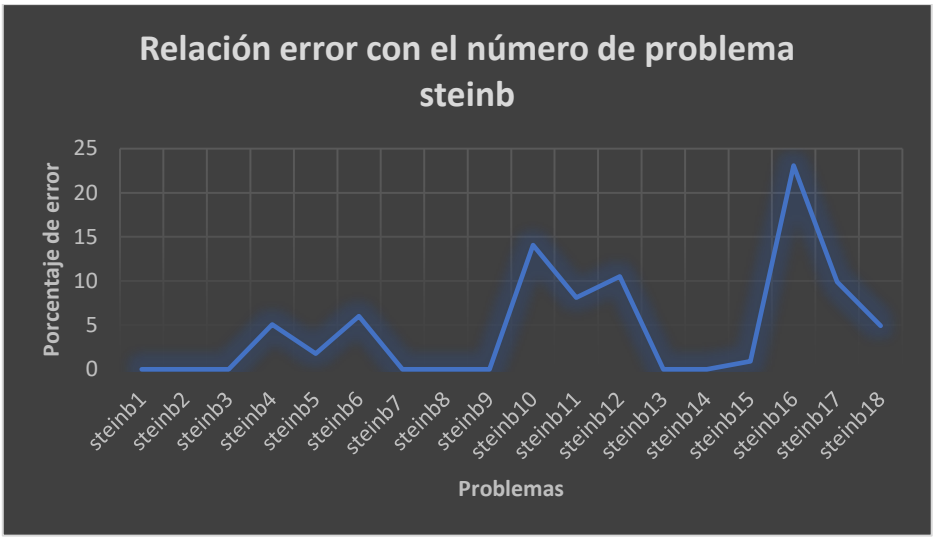
Recordar que todas las soluciones y resultados obtenidos se atienen a 30 poblaciones de N individuos y un número dado de iteraciones. Luego se obtienen resultados teniendo en cuenta gran parte del espacio de soluciones.



Gráfica 7. Porcentaje de error respecto del número de iteraciones de los problemas de tipo “c” para poblaciones de 60 individuos

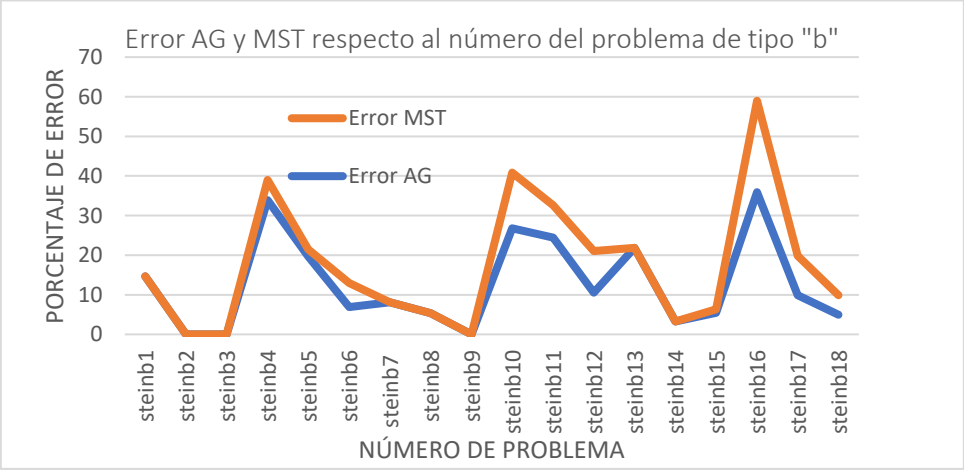
A continuación, en la Gráfica 8 se muestra el porcentaje de error respecto el número de problema de tipo “b” en vez del número de iteraciones. Para realizar estos cálculos se han empleado los datos de la Tabla 7 obtenidos con 100 individuos en cada una de las 30 poblaciones por problema y 500 iteraciones por población.

Además, se muestra como para los problemas steinb no sube el porcentaje de error a más de un 25%. Al igual que se puede ver cómo a medida que aumenta el número del problema se producen picos de errores más altos.



Gráfica 8. Porcentaje de error frente al número de problema tipo “b”

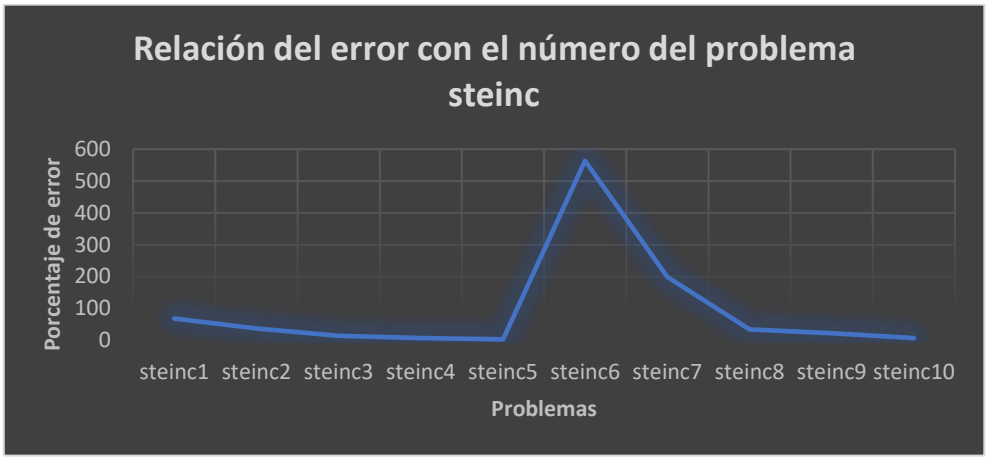
En la Gráfica 9, se observa el error provocado por el algoritmo genético en comparación con el error del MST para los problemas de tipo “b”. Se puede visualizar como el error del algoritmo genético viene acotado por el error del MST que siempre es mayor o igual.



Gráfica 9. Contraste error del MST y del AG respecto a los problemas de tipo “b”

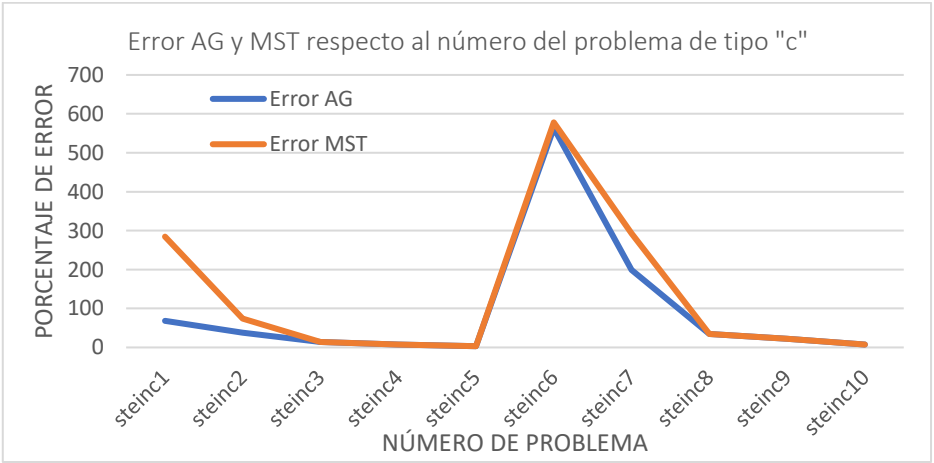
En la Gráfica 10 se determina la relación del error con el número del problema de steinc resuelto para una población de 100 individuos y 500 iteraciones tal como en la gráfica anterior.

Se puede ver como el error sube enormemente hasta alcanzar valores de 500% de error como es el caso del problema steinc6. Se podría llegar a intuir que esto es debido a las pocas iteraciones y a la complejidad del problema.



Gráfica 10. Porcentaje de error frente al número de problema tipo “c”

En la Gráfica 11, la cual compara el error cometido por el AG y por el MST respecto del número de problema de tipo “c” se aprecia gran diferencia entre los problemas steinc1 y steinc2. Luego, para los problemas de tipo “c” es posible obtener también mejores resultados con el algoritmo genético, aunque no se cumple para todos los problemas del tipo “c”.



Gráfica 11. Contraste error del MST y del AG respecto a los problemas de tipo “c”

Para que el estudio de parámetros tan importantes como son el error y el tiempo no queden poco vistos, se pasa a observar la Gráfica 12 que representa la relación del tiempo de implementación respecto al número de problema steinb con 500 iteraciones y 100 individuos.

Como se puede ver, a mayor complejidad del problema (mayor número en el nombre del problema) se obtienen mayores tiempos de implementación. Se puede apreciar una tendencia ascendente irregular debido a la diferente densidad de nodos terminales entre los problemas.

Gráfica 12. Tiempo de implementación en segundos frente el número de problema de tipo "b"

En la Gráfica 13 se muestra la misma relación. Es decir, el tiempo de implementación del problema frente el número de problema de tipo "c" con 500 iteraciones y 100 individuos.

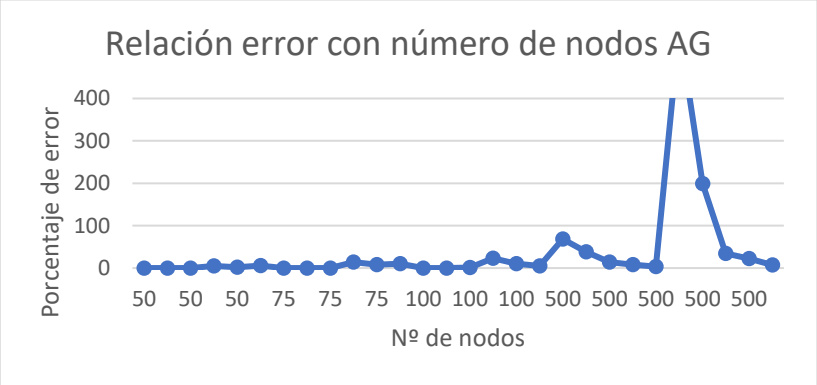
Aquí la tendencia al alza está más definida. Se podría atribuir a que la complejidad de los problemas crece a una velocidad mayor. Por otro lado, se puede observar cómo se pasa de tiempos de resolución de menos de 20 segundos para problemas de tipo "b" a tiempos de resolución de incluso más de 4 minutos para problemas de tipo "c". El cambio brusco de la curva se debe a un aumento en el número de arcos permaneciendo el número de nodos constante.

Gráfica 13. Tiempo de implementación en minutos frente el número de problema de tipo "c"

Al resultar el tiempo de resolución del MST de milésimas para cualquiera de los problemas analizados, no merece la pena realizar un gráfico que relacione el tiempo de ejecución con los problemas de Steiner analizados.

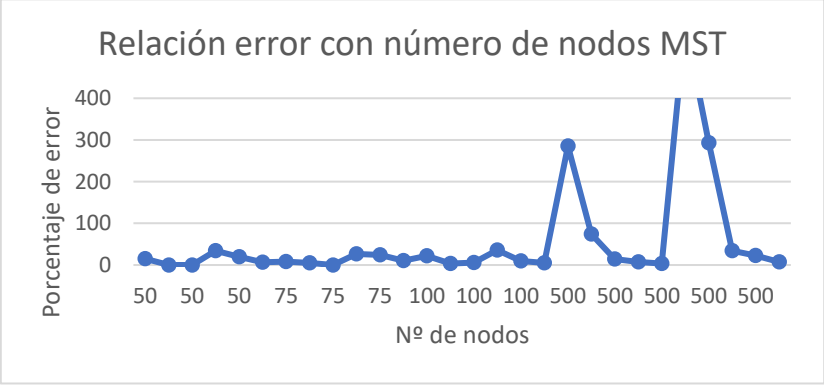
Para tener en cuenta la correlación del número de nodos, el número de arcos con el error y el tiempo de ejecución, se exponen a continuación las siguientes gráficas.

En la Gráfica 14 se muestra la relación del porcentaje de error con el número de nodos que se encuentran a lo largo de los problemas tipo "b" y "c". Como se puede ver, a medida que aumenta el número de nodos en los problemas, mayor porcentaje de error de media se obtiene por la resolución del algoritmo genético. Se muestra como el aumento no es lineal, debido a que el error no solo depende del número de nodos sino también de la densidad de los nodos terminales y del número de arcos. El pico que sobresale de la gráfica se debe al error del problema steinc6.



Gráfica 14. Porcentaje de error frente el número de nodos en la resolución de algoritmos genéticos

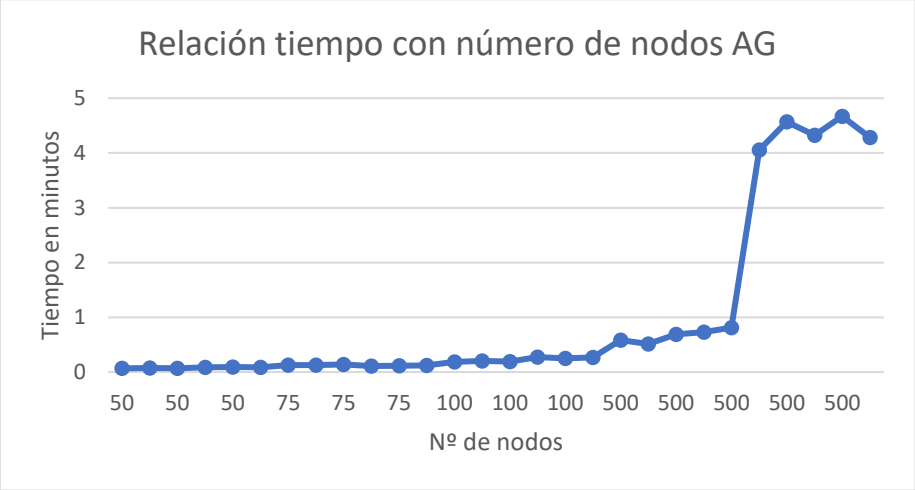
En la Gráfica 15, se aprecia el porcentaje de error en relación con el número de nodos, pero resolviendo ahora los problemas con el MST. Se puede ver el gran parentesco de la Gráfica 14 con la Gráfica 15 para los problemas con nodos mayores de 500 que pertenecen al tipo "c". Esto es así ya que como se vio en la Gráfica 11, el porcentaje de error entre las soluciones con MST y algoritmo genético es muy parecido para los problemas de tipo "c".



Gráfica 15. Porcentaje de error frente el número de nodos en la resolución del MST

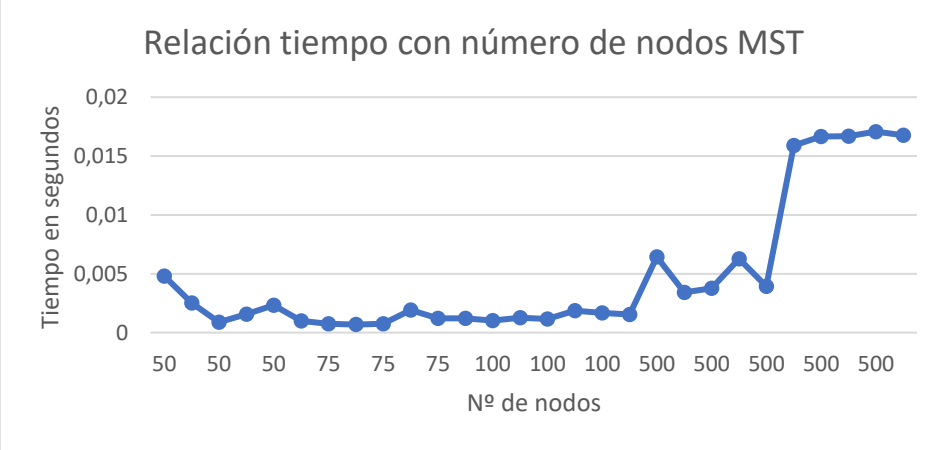
Se pasa ahora a analizar el tiempo de ejecución respecto del número de nodos.

En la Gráfica 16 se expone la relación del tiempo de ejecución respecto al número de nodos resueltos los problemas con algoritmos genéticos. Se vuelve a apreciar, como es lógico, que a mayor número de nodos mayor tiempo de ejecución. Podemos relacionar también que a mayor número de nodos mayor dificultad tiene el problema. Pues más avanzado dentro de la colección de problemas se estará. El cambio brusco de pendiente se debe a la gran diferencia entre el número de nodos y de arcos.



Gráfica 16. Relación del tiempo de ejecución respecto al número de nodos resuelto con algoritmo genético

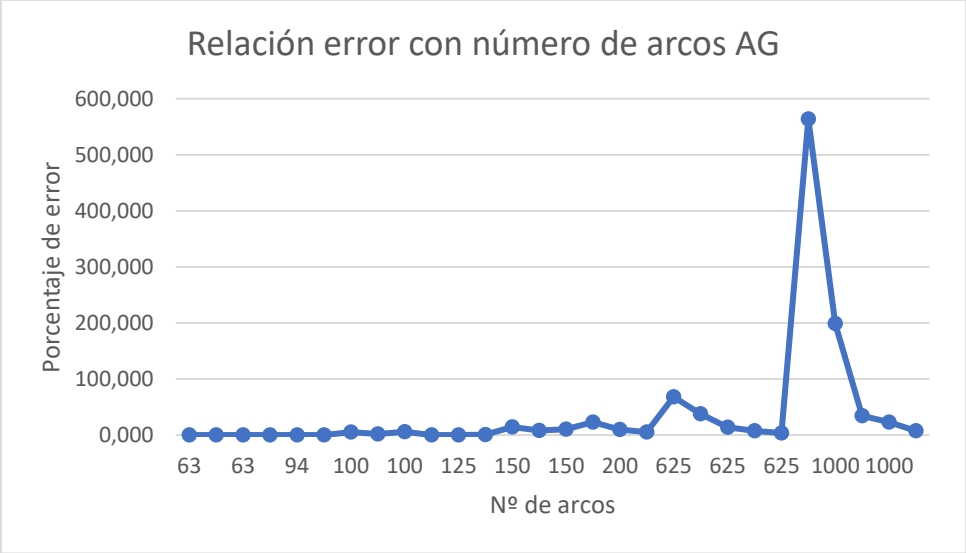
En la Gráfica 17 se visualiza el tiempo de ejecución frente al número de nodos de problemas resueltos con el MST. Se observa como el tiempo de ejecución aumenta con el número de nodos. Hay que tener en cuenta que aquí el tiempo se contabiliza en segundos, luego se muestra de nuevo el tiempo de ejecución tan reducido que precisa el algoritmo MST.



Gráfica 17. Relación del tiempo de ejecución respecto al número de nodos resuelto con MST

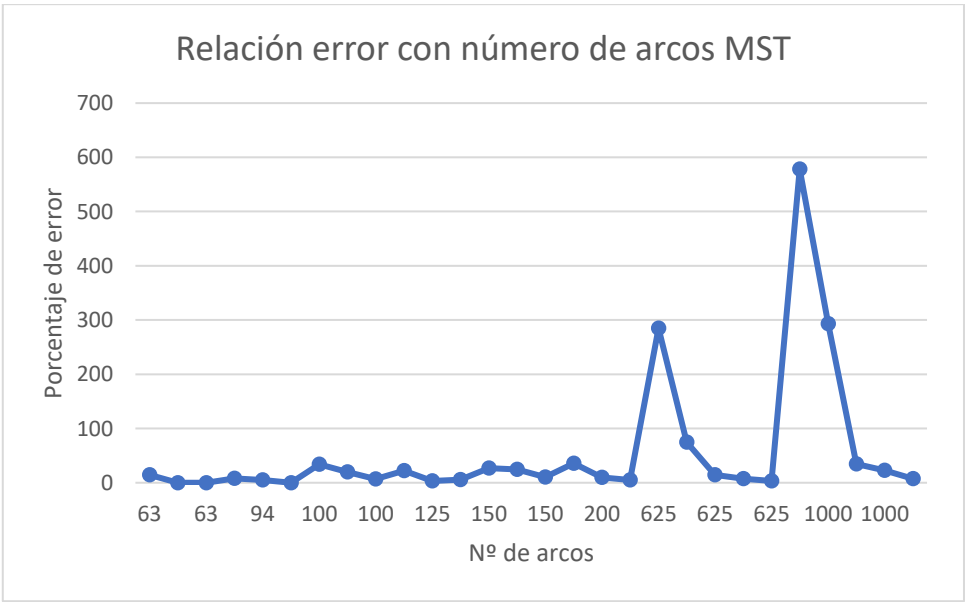
La relación del número de arcos con el porcentaje de error, así como, con el tiempo de ejecución también son fruto de observación en este estudio.

Como se puede visualizar en la Gráfica 18, el número de arcos también afecta de forma similar que el número de nodos al error.



Gráfica 18. Relación del error con el número de arcos en problemas resueltos con el AG

En la Gráfica 19 se muestra la relación del error con el número de arcos para la resolución por MST. Se aprecia como el error es mayor debido a que este algoritmo es mucho menos preciso que el algoritmo genético.

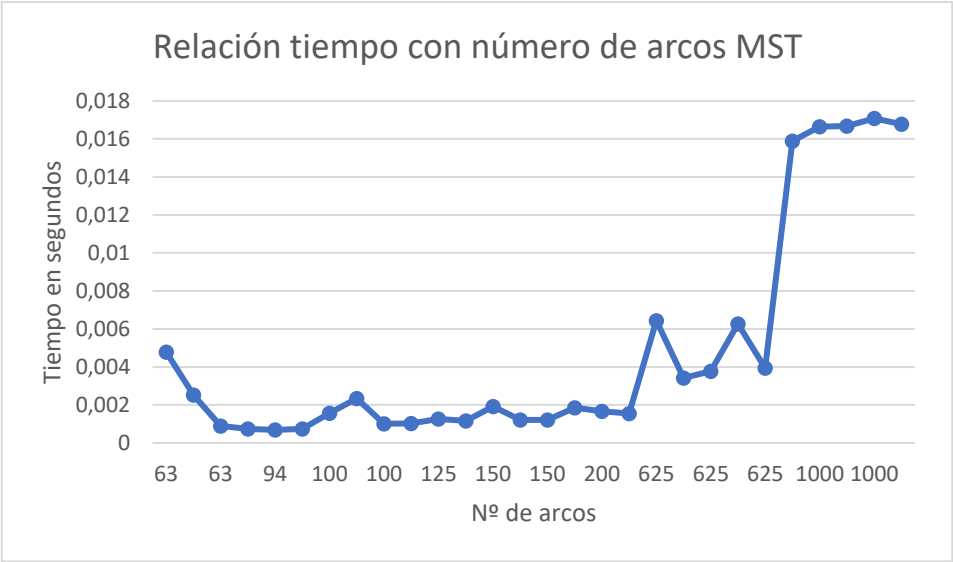


Gráfica 19. Relación del error con el número de arcos en problemas resueltos con el MST

En la Gráfica 20 se representa el tiempo de ejecución en función del número de arcos para soluciones con AG. Se muestra gran similitud con la Gráfica 16 que expone el tiempo de ejecución del AG en función del número de nodos. Se puede ver el gran salto de la pendiente cuando se pasa de 625 arcos a 1000.

Gráfica 20. Tiempo de ejecución en función del número de arcos para soluciones con AG

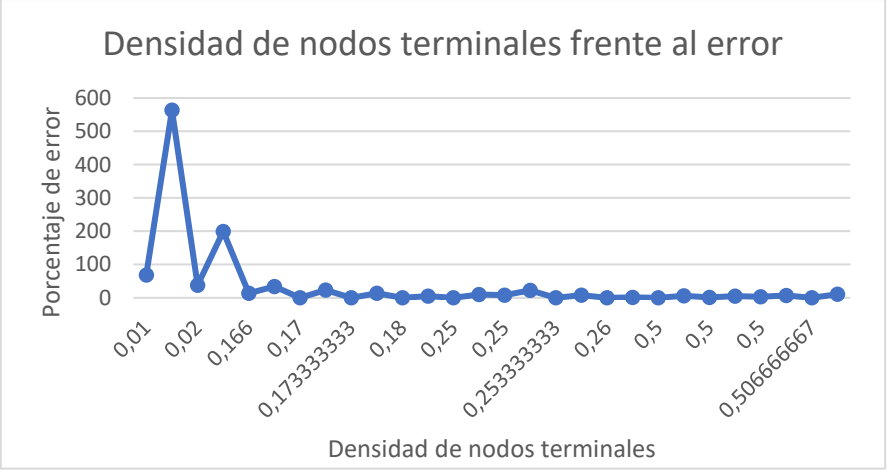
Del mismo modo, la Gráfica 21 visualiza el tiempo de ejecución frente el número de arcos para problemas resueltos por el MST. Se puede observar como el tiempo de ejecución está en segundos y la gráfica en sí tiene mucho parecido con la Gráfica 17 donde se representa el tiempo de ejecución con el número de nodos para problemas resueltos con el MST.



Gráfica 21. Tiempo de ejecución en función del número de arcos para soluciones con MST

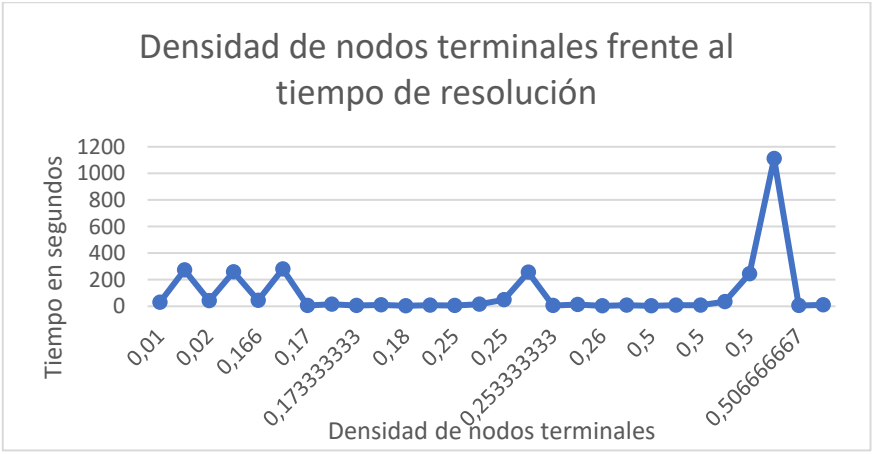
Para acabar con el análisis que relaciona el error y el tiempo de ejecución con el número de nodos y el número de arcos, se presenta a continuación la densidad de nodos terminales (nodos totales entre nodos terminales) en función del error y el tiempo de resolución.

La Gráfica 22 deja ver como a mayor densidad de nodos terminales, menor error relativo respecto al óptimo se obtiene. Esto es debido a que cuantos menos nodos Steiner existan en el problema, menos soluciones admisibles habrá. Ya que no existirán tantos posibles caminos para comunicar unos nodos con otros y esto resultará en un error en la solución más pequeño. Es decir, el espacio de soluciones se reduce y la exploración del óptimo tendrá un mayor efecto.



Gráfica 22. Densidad de los nodos terminales en función del porcentaje de error

En la Gráfica 23 se visualiza el tiempo de ejecución respecto de la densidad de nodos terminales. Aquí se puede ver como el tiempo de ejecución tiende a tener una probabilidad alta de disminuir a medida que la densidad de nodos terminales aumenta justamente por la explicación de que, a mayor número de nodos terminales, menos soluciones posibles.



Gráfica 23. Densidad de los nodos terminales en función del tiempo de resolución

6.4 Curvas de superficie

Para finalizar, en este apartado se van a analizar algunas soluciones que han despertado gran interés por sus resultados. Se van a exponer una serie de gráficas tridimensionales realizadas en base a 176 soluciones de cada uno de los problemas representados en ellas. Las soluciones cambian formándose una curva de superficie en función del número de individuos que tenga la población y en función del número de iteraciones. Particularmente, se han obtenido resultados para poblaciones entre 50 y 100 individuos y para iteraciones entre 500 y 2000.

Cada una de las curvas de superficie tiene un tiempo de ejecución de media de día y medio. Pues, realizar 176 soluciones incrementando el número de iteraciones tiene un tiempo computacional muy costoso.

Para comprender las figuras, se pasa a explicar la estructura de las mismas. En el eje Z se sitúan los costes de las soluciones, es decir, cuanta menos altura tenga la figura, mejor coste se habrá obtenido para una solución en particular. En el eje de coordenadas se muestra el rango de iteraciones que comprende desde 500 hasta 2000 iteraciones. Por otro lado, el eje de abscisas contiene el número de individuos de cada una de las 30 poblaciones con las que se halla cada solución. El eje X va de 50 individuos a 100.

Se dará comienzo con la gráfica de steinb4 pues para los problemas anteriores a este, debido a su sencillez, no se encuentran soluciones diferentes.

En la Figura 68 se puede ver como el rango de soluciones oscila entre 59 (óptimo) y 63. El mayor número de soluciones óptimas se sitúa en la zona rodeada. Se podría decir que el problema 4b presenta mayor probabilidad para obtener el óptimo cuando la población es de 80 y se ejecutan alrededor de 1500 iteraciones.

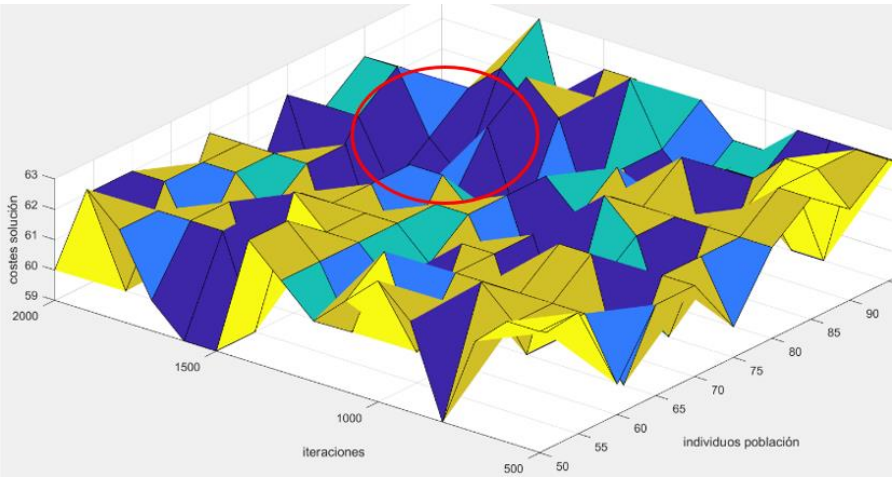


Figura 68. Curva de superficie para steinb4

En la Figura 69 se puede ver desde una vista desde arriba el plano de soluciones. Representando los cuadrados más oscuros las soluciones óptimas.

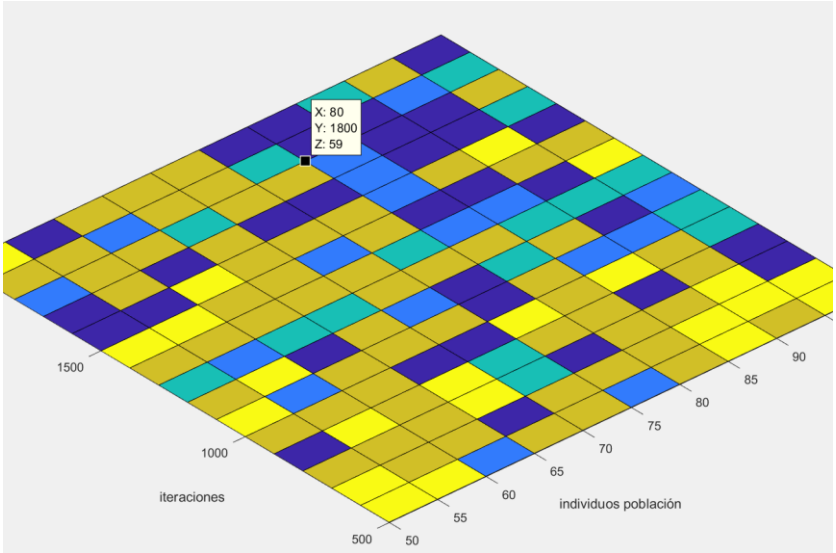


Figura 69. Curva vista desde arriba para steinb4

En la Figura 70 se ve como se muestra la curva de superficie para steinb6. Se puede ver que para este caso el rango de soluciones oscila desde 116 a 124. Por otro lado, se puede apreciar la dificultad de obtener el óptimo en este problema, pues casi para todas las soluciones del gráfico se obtiene 118 unidades de coste en vez de 116.

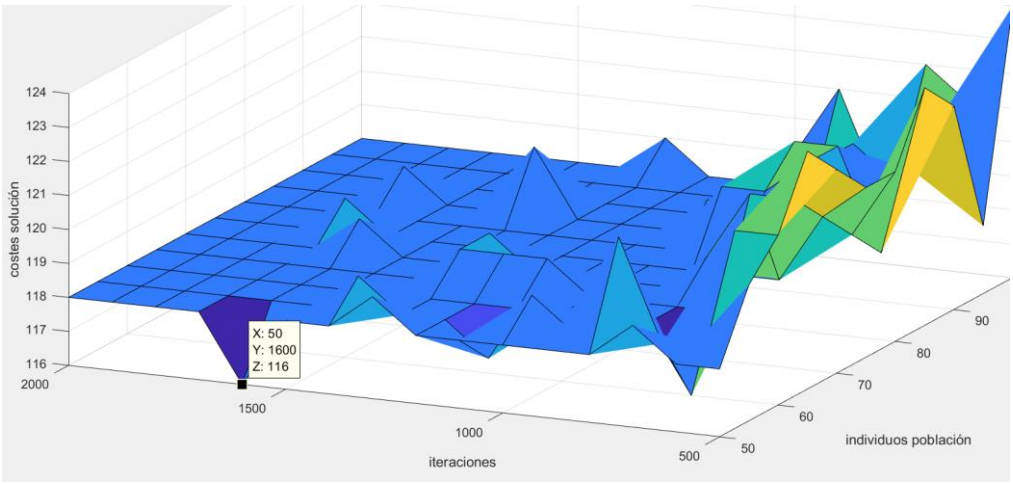


Figura 70. Curva de superficie de steinb6

En la Figura 71 se muestra la curva de superficie para steinb12. Se visualiza como para iteraciones en torno a 500 se obtienen soluciones muy por encima del óptimo. El mayor valor es de 168 y el menor de 152. Este valor es el óptimo, el cual solamente es alcanzado una vez con una población de 50 y 700 iteraciones. El rango de variación de las soluciones es de 16 unidades de coste que equivale a un error de 10.526% respecto del óptimo.

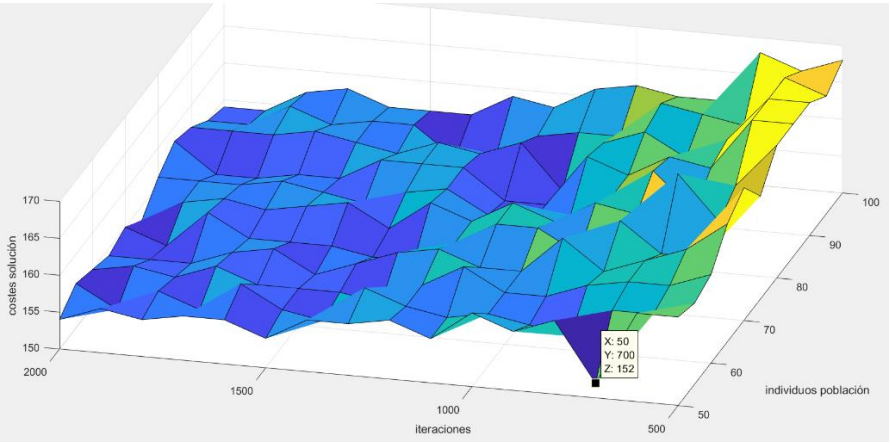


Figura 71. Curva de superficie de steinb12

Para el problema steinb16 visualizado en la Figura 72, no se encuentra la solución óptima de 117 unidades de coste, sino que el mínimo es 128 encontrándose solo en 5 de las 176 soluciones. Por otro lado, la máxima solución es de 150 unidades de coste alcanzando un error de 28.205% sobre el óptimo.

Además, justamente el punto donde más error hay respecto de la solución, es el de 500 iteraciones y 100 individuos. Este punto se empleó en la Tabla 7 para dar solución al problema steinb16, solo que en la gráfica da 148 unidades y en la tabla 144.

Por todo ello, se puede considerar el problema más complejo de la colección de tipo “b”, pues en ninguna de las 176 soluciones mostradas en la Figura 72 se da con el óptimo. Esto es debido a que steinb16 es el problema de tipo “b” con mayor diferencia entre el número de arcos y el número de nodos que contiene mayor número de nodos respecto al resto de soluciones de tipo “b”. Sumado a que, a pesar de su tamaño, la densidad de nodos terminales es muy baja, lo que permite la mayor posibilidad de soluciones posibles y la dificultad de dar con el óptimo.

Las gráficas steinb6, steinb12 y steinb16 comparten una forma muy similar.

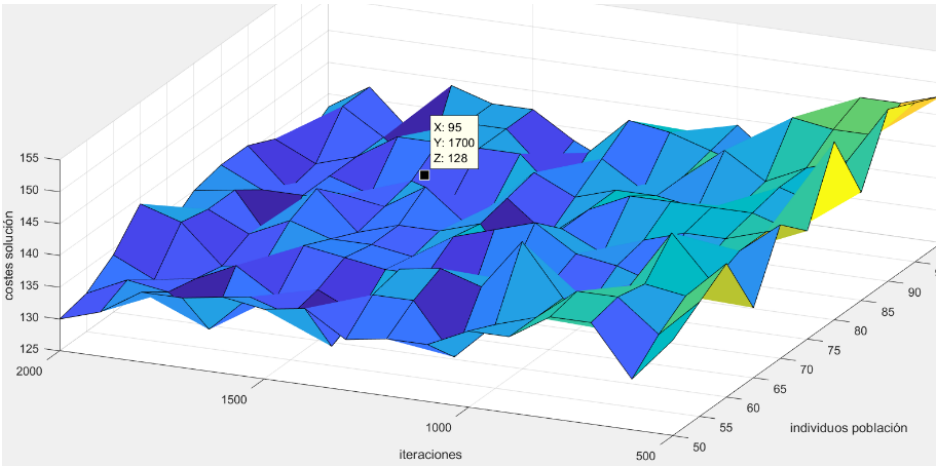


Figura 72. Curva de superficie de steinb16

A continuación, se muestran algunas gráficas de los problemas de Steiner de tipo “c”.

La gráfica de steinc1 muestra que el mínimo obtenido es de 89 (hallado solo una vez) y el máximo de 105 siendo el óptimo de 60 unidades de costes. Esto permite entender por qué el problema steinc1 presenta un error tan elevado para la implementación de 100 individuos y 500 iteraciones. Y es que para ninguna de las soluciones presentes en la Figura 73 se consigue llegar al óptimo.

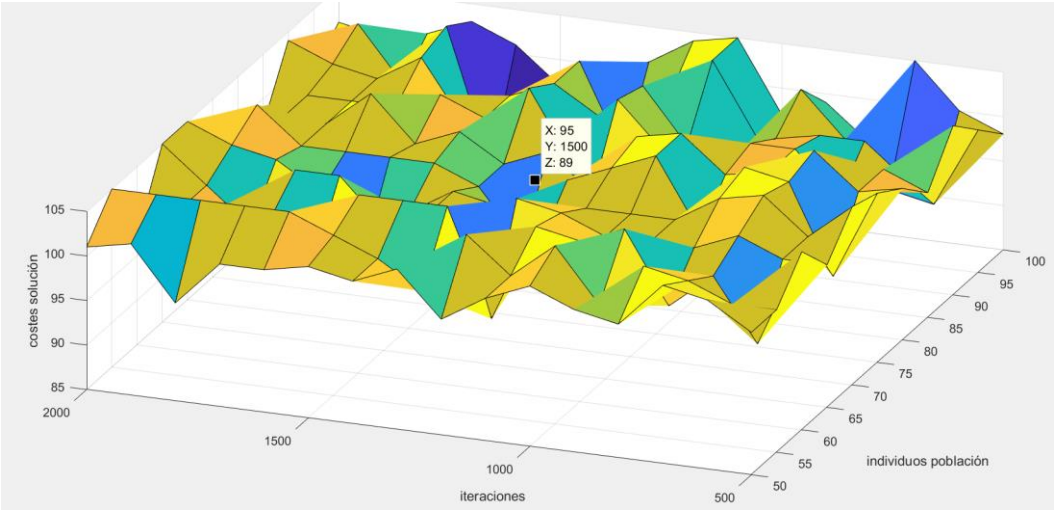


Figura 73. Curva de superficie de steinc1

Para la Figura 74 se observa el problema de steinc4 donde se puede ver cómo al aumentar el número de iteraciones, rápidamente disminuye las unidades del coste. Para esta gráfica el mínimo es de 736, que se obtiene dos veces como se muestra en la gráfica, y el máximo de 787. Debido a que el óptimo es de 731 unidades de coste, este nunca se llega a alcanzar, aunque el mínimo se aproxima mucho a él.

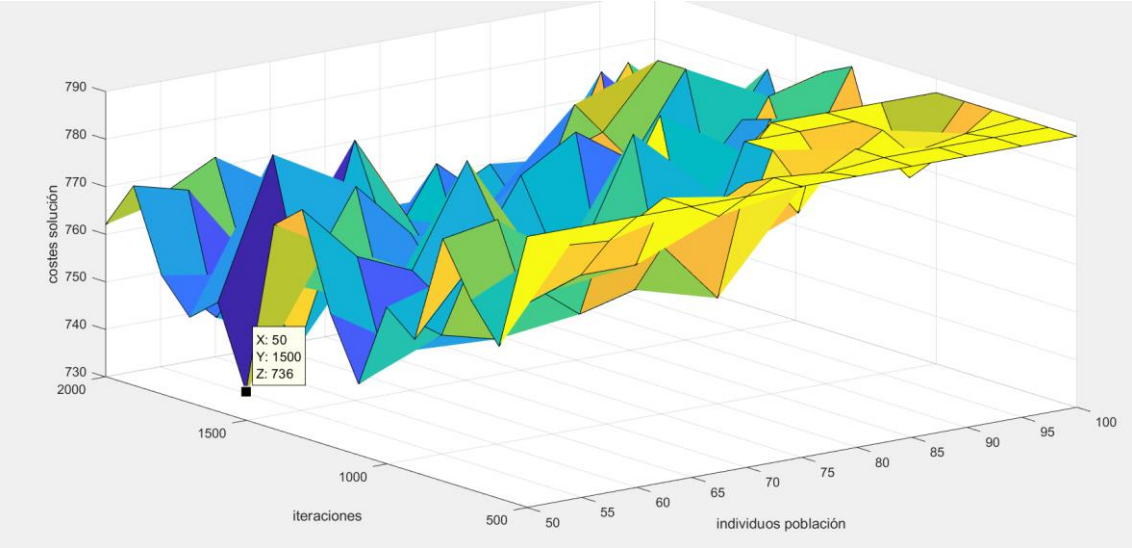


Figura 74. Curva de superficie para steinc4

En la Figura 75 se muestra con precisión cómo disminuye el número de soluciones a medida que se incrementa el número de iteraciones.

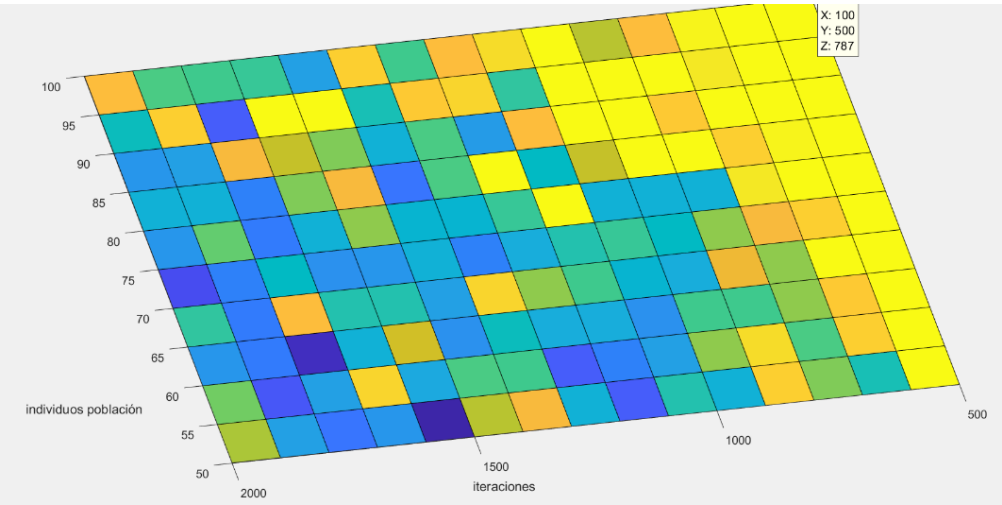


Figura 75. Curva vista desde arriba para steinc4

Para la Figura 76 se puede visualizar el gran rango de variación de las soluciones obtenidas en el problema steinc6. Siendo el máximo de 373 unidades de coste y el mínimo de 315 unidades obtenido solo una vez. Sin duda se trata de la gráfica que peor ajusta la solución pues el óptimo es de 55 unidades de coste. Es por ello, que se produce un error de mas de 500% respecto del óptimo.

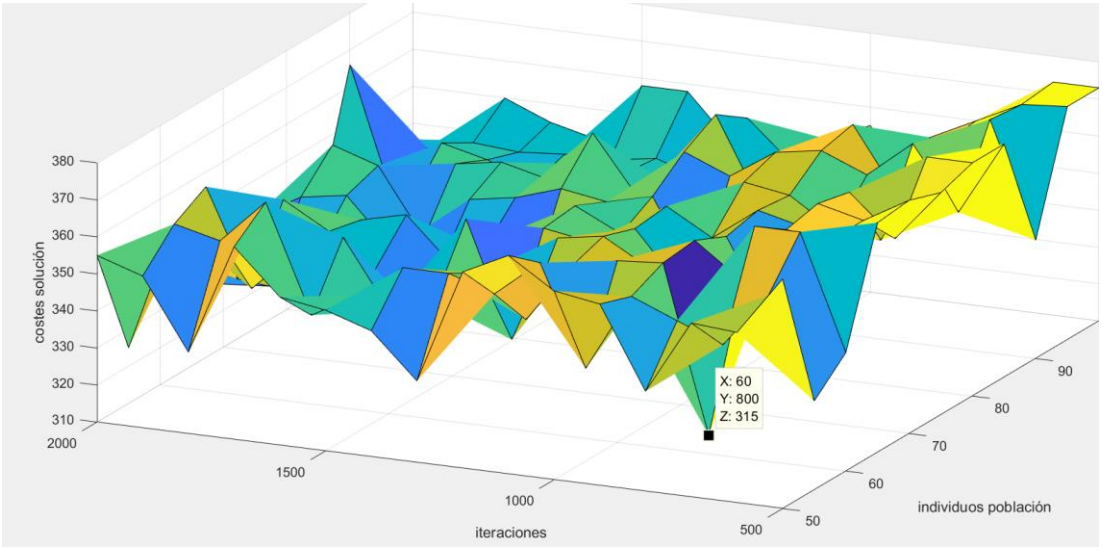


Figura 76. Curva de superficie para steinc6

Para el caso del problema steinc7, se puede ver en la Figura 77 como el máximo es de 311 y el mínimo de 254 siendo el óptimo de 102 unidades de coste. Lo que lleva a pensar que por regla general independientemente del número de implementaciones se obtendrán errores muy amplios al igual que para steinc6.

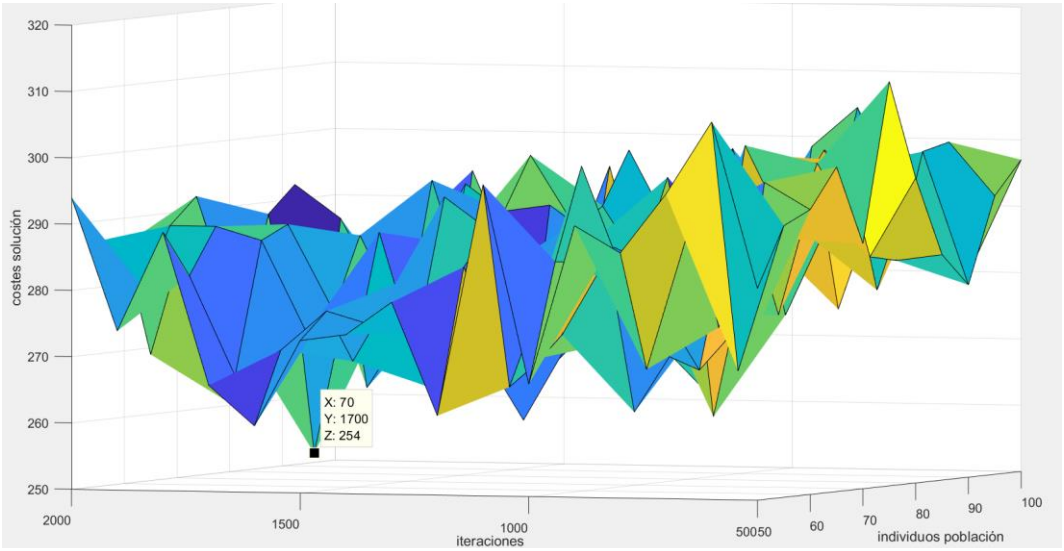


Figura 77. Curva de superficie para steinc7

En la Figura 78 se puede ver como para el problema steinc10 se consiguen máximos de 1083 y un mínimo de 1065 siendo el óptimo 1009. Luego a pesar de la diferencia del mínimo con el óptimo el error relativo se considera pequeño pues son órdenes de magnitud de miles.

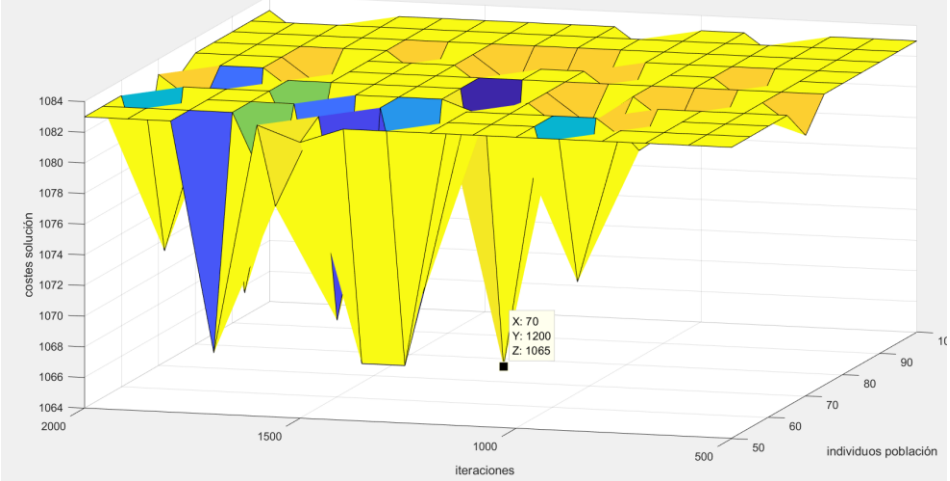


Figura 78. Curva de superficie para steinc10

7 CONCLUSIONES

Todo lo abordado en la sección 6 permitirá finalmente en el apartado 7 culminar con una conclusión precisa y clara, la cual dará final a este trabajo de grado.

El estudio realizado sobre los problemas de Steiner ha dejado ciertas conclusiones muy interesantes de comentar. Para comenzar, el hecho de que el problema de Steiner sea NP-Completo implica la imposibilidad de resolución del problema en tiempo polinomial. Esto permite comprobar que; cuanto mayor es el tamaño del problema, se obtiene mayor tiempo de resolución y el error respecto al óptimo también aumenta. Lo que significa que los problemas más avanzados en la colección tipo "c" tienen mayor tiempo de resolución y además aumenta increíblemente el error debido a que la complejidad de los problemas es mayor. Esta conclusión es aplicable a todos los problemas NP-Completos.

Respecto a la Tabla 7 y a la Tabla 8, donde se mostraban los resultados para los problemas tipo "b" y "c" respectivamente con 100 individuos y 500 iteraciones, se podía ver como el error se hace cero en los problemas donde el número de nodos y arcos es aproximadamente igual. Lo que permite determinar que cuando el número de nodos y arcos es próximo se consigue un menor error y tiempo de implementación. Esto es debido a que no existen tantas soluciones admisibles. Por otro lado, en la Tabla 7 se mostraba que los problemas steinb10 y steinb16 han tenido los mayores errores de la colección tipo "b". Esto se debe a que el número de terminales para los dos problemas es el más bajo de entre los problemas con el mismo número de nodos y de arcos. Además, en la Tabla 8 se mostraba justamente como el problema steinc6 cumple con un error muy superior al del resto de problemas de la colección debido al mismo hecho que para los problemas steinb10 y steinb16. Lo que permite concluir que cuando los nodos terminales son bajos y el número de arcos es el doble o más que el número de nodos, los errores de los problemas de Steiner aumentan considerablemente.

Otros factores que juegan un papel importante en la solución de los problemas es el número de implementaciones y de individuos (soluciones) por población. Desde la Gráfica 4 a la Gráfica 7 se demuestra como a mayor número de implementaciones menor porcentaje de error es posible obtener. Además, también se puede observar que al disminuir el número de individuos por población de 90 a 60 también disminuye el error. Lo que permite concluir la importancia del tamaño de la población pensando que una población en torno a 60 individuos cumple mejor con los resultados que una población de 90 individuos.

Respecto a la comparativa entre la resolución por MST y por algoritmos genéticos, se puede concluir que el error producido por algoritmos genéticos siempre es mayor o igual que el cometido por MST pues el algoritmo genético utiliza en primera instancia el MST para mejorar sus resultados. Esto se puede visualizar en la Gráfica 9 y Gráfica 11. Otra observación importante es que el error entre MST y algoritmo genético no difiere demasiado en los problemas de tipo "c" debido a su creciente complejidad que dificulta la resolución óptima del problema. Luego cuando los problemas alcanzan una dificultad determinada el algoritmo genético podría no mejorar en gran medida la solución del MST. Además, el tiempo de implementación que necesita la resolución por MST es de segundos y la implementación utilizando algoritmos genéticos llega a

ser de varios minutos para los casos más complicados. Es decir, el orden de magnitud de la resolución del MST es de segundos y la del algoritmo genético de minutos.

Los factores número de nodos, número de arcos y densidad de nodos terminales (nodos totales entre nodos terminales) influyen a la resolución de los problemas de Steiner. Desde la Gráfica 14 a la Gráfica 23 se observa como al aumentar el número de nodos, el número de arcos y la densidad de nodos terminales, se obtienen tiempo de resolución más elevados. En cambio, el error no aumenta progresivamente con el número de nodos y de arcos. Es en determinados números de nodos y de arcos donde se aprecian picos de error. Lo que permite concluir que el error está fuertemente ligado a la baja densidad de nodos terminales, debido a que el número de soluciones posibles crece exponencialmente.

Finalmente, durante el análisis de las curvas de superficie se puede observar la relevancia del buen ajuste de las iteraciones y del número de individuos (soluciones) de las poblaciones. Esto lleva a concluir que cuando las iteraciones están en torno a las 2000 se consiguen disminuir considerablemente los errores de cualquier tipo de problema. También la elección del número de individuos llega a influir en las soluciones de los problemas.

Con todo ello, se pretende finalmente destacar cómo los algoritmos genéticos resultan ser técnicas muy potentes de resolución mejorando los resultados considerablemente para la mayoría de los casos. Por otro lado, el MST se interpone como un algoritmo de peso para problemas de difícil resolución por su breve implementación.

8 REFERENCIAS

- (Marcus Brazil and Martin Zachariasen, 2010). Optimal Interconnection Trees in the Plane. Theory, Algorithms and Applications (pp. 302)
- (V. Jarník and O. Kössler, O minimálnich grafech obsahujících n daných bodu, Ĉas. Pêstování Mat. 63 (1934) (pp. 223-235).
- (Xiuzhen Cheng and Ding-Zhu Du, 2001). Combinatorial Optimization. Steiner Trees in Industry, Volume 11 (pp. 30)
- Zelikovsky, 1993. An 11/6-approximation algorithm for the network Steiner problem, Algorithmica 9 (1993). (pp. 463–470)
- Aneja, Y.P.: An integer linear programming approach to the Steiner problem in graphs. Networks 10, 167–178 (1980)
- Bern, M., Plassmann, P.: The Steiner problem with edge lengths 1 and 2. Inf. Process. Lett. 32, 171–176 (1989)
- C. Gröpl, S. Hougardy, T. Nierhoff, H. J. Prömel, 2000. Approximation algorithms for the Steiner tree problem in graphs, technical report, Humboldt-Universität zu Berlin, 2000.
- D.Z. Du, J.M. Smith, J.H. Rubinstein, Advances in Steiner Trees (Kluwer Academic, Dordrecht, 2000)
- Dietmar Cieslik, 2005. Shortest Connectivity. An Introduction with Applications in Phylogeny (pp. 2)
- Dreyfus, S.E., Wagner, R.A.: The Steiner problem in graphs. Networks 1, 195–207 (1972)
- Duin, C.W., Volgenant, A.: An edge elimination test for the Steiner problem in graphs. Oper. Res. Lett. 8(2), 79–83 (1989)
- Duin, C.W., Volgenant, A.: Reduction tests for the Steiner problem in graphs. Networks 19(5), 549–567 (1989)
- Duin, C.W.: Preprocessing the Steiner problem in graphs. In: Du, D.-Z., Smith, J.M., Rubinstein, J.H. (eds.) Advances in Steiner Trees, pp. 173–233. Kluwer Academic, Boston (2000)
- Duin, C.W.: Steiner's problem in graphs – approximation, reduction, variation. PhD thesis, University of Amsterdam (1993)
- E. N. Gilbert, H. O. Pollak, "Steiner minimal trees," SIAM Journal of Applied Mathemab:cs, Vol. 16, pp. 1-29, 1986
- Frank K. Hwang, Dana S. Richards, Pawel Winter, 1992. Annals of Discrete Mathematics. The Steiner Tree Problem, Volume 53 (pp. 4)
- G. Robins, A. Zelikovsky, in Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SIAM, San Francisco, 2000), pp. 770–779
- Garey, M.R., Johnson, D.S.: The rectilinear Steiner tree problem is NP-complete. SIAM J. Appl. Math. 32(4), 826–834 (1977)
- Geyer, H., Ulbig, P., and Sculz, S. (1999). Use of evolutionary algorithms for the calculation of group contribution parameters in order to predict thermodynamic properties: Part 2: Encapsulated evolution strategies. Computers and Chemical Engineering, 23(7):955-973.
- Goemans, M.X., Myung, Y.S.: A catalog of Steiner tree formulations. Networks 23, 19–28 (1993)
- Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization and Machine Learning.

Addison- Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition.
- Hans Jürgen Prömel y Angelika Steger, 2002. The Steiner Tree Problem. (1st ed.). Braunschweig (pp. 1).
- J.B. Kruskal. On the shortest spanning subtree of a graph and the travelling salesman problem. Proc. of the Am. Math. Soc., 7:48-50. 1956.
- Lerchs, H., Grossmann, I.F.: Optimum design of open-pit mines. Trans. Can. Inst. Min. Metall. Pet. 68, 17–24 (1965)
- M. R. Garey, R. L. Graham, and D. S. Johnson, "The complexity of computing Steiner minimal trees," SIAM Journal of Applied Mathematics, Vol. 34, pp. 477-495, 1977.
- Narges Norouzi, Mohsen Sadegh-Amalnick et. al. (2016). Modified Particle swarm Optimization in a time-dependent Vehice Routing Problema: Minimizing Fuel Consumption
- O. Boruvka. O jistem problema minimalnim. Acta Societ. Scient. Natur. Moravice, 3:37-58, 1926.
- Panos M. Pardalos, Ding-Zhu Du, Ronald L. Graham (2013). Handbook of Combinatorial Optimization. (2nd ed.). New York, (Volumen 2, Energy Efficiency in Wireless Network)
- Panos M. Pardalos, Ding-Zhu Du, Ronald L. Graham (2013). Handbook of Combinatorial Optimization. (2nd ed.). New York, (Volumen 1, Combinatorial Optimization Techniques for Network-Based)
- Panos M. Pardalos, Ding-Zhu Du, Ronald L. Graham (2013). Handbook of Combinatorial Optimization. (2nd ed.). New York, (Volumen 1, Combinatorial Optimization Techniques for Network-Based)
- Polzin, T., Vahdati Daneshmand, S.: Extending reduction techniques for the Steiner tree problem. In: Möhring, R., Raman, R. (eds.) Algorithms – ESA 2002, Rome. Lecture Notes in Computer Science, vol. 2461, pp. 795–807. Springer, Berlin/Heidelberg (2002)
- Polzin, T., Vahdati Daneshmand, S.: Improved algorithms for the Steiner problem in networks. Discret. Appl. Math. 112, 263–300 (2001)
- Polzin, T.: Algorithms for the Steiner problem in networks. PhD thesis, Universität des Saarlandes (2003)
- Prezi. Noah Todd (2013). Kruskal Steiner Fiber Optic Cable Project. https://prezi.com/0nlvssxrnlgt/kruskal-steiner-fiber-optic-cable-project/?webgl=0
- R. Courant and H. Robbins, 1941. R. Courant and H. Robbins, What is Mathematics? Oxford Univ. Press, New York (1941).
- R. M. Karp, "Reducibility among Combinatorial Problems," In R. E. Miller, J. W. Thatcher (Eds.), Complexity of Computer Computations, Plenum Press, New York, pp. 85-103, 1972.
- Razali, N. M. and Geraghty, J. (2011). Article: Genetic algorithm performance with different selection strategies in solving tsp. International Journal of Computer Applications Proceedings of The World Congress on Engineering 2011, 2:1134-1139.
- Redes de fibra óptica: Jaime Prieto Zapardiel (2014). Diseño de una Red de Acceso Mediante Fibra Óptica
- Redes de minas subterráneas: Markus Brazil, Martin Zachariasen (2015). Optimal Interconnection Trees in the Plane, (pp. 229-230)
- Redes eléctricas: Daniel O. Anaut, Guillermo F. di Mauro, Gustavo Meschino y Juan A. Suárez (2009). Optimización de Redes Eléctricas mediante la aplicación de algoritmos genéticos
- Redes ferroviarias: Tren 2020: Propuesta ferroviaria para una nueva realidad. Resumen ejecutivo mayo 2013.

- Research gate. El uso de redes complejas en economía: alcances y perspectivas. (2017). https://www.researchgate.net/figure/Figura-1-Los-puentes-de-Koenigsberg_fig1_320638009
- Rubem P. Mondaini (2007). Steiner Ratio of Biomolecular Structures
- Seong Eun et. al. (2015). Structural Changes in the Minimal Spanning Tree and the Hierarchical Network in the Korean Stock Market around the Global Financial Crisis
- Tuttle, R. H. (1989). Arguments on evolution. A paleontologist's perspective. American Journal of Physical Anthropology, 80(3):405-406.
- Uchoa, E., Poggi de Aragão, M., Ribeiro, C.: Preprocessing Steiner problems from VLSI layout. Networks 40, 38–50 (2002)
- Vahdati Daneshmand, S.: Algorithmic approaches to the Steiner problem in networks. PhD thesis, Universität Mannheim (2004)
- Wong, R.: A dual ascent approach for Steiner tree problems on a directed graph. Math. Program. 28, 271–287 (1984)
- Xiuzhen Cheng and Ding-Zhu Du (2001). Steiner Trees in Industry. Países Bajos (Volumen 11, Steiner Tree Based Distributed Multicast Routing in Networks)
- Xiuzhen Cheng and Ding-Zhu Du (2001). Steiner Trees in Industry. Países Bajos (Volumen 11, Steiner Trees in Uniform Orientation Metrics)
- Xiuzhen Cheng and Ding-Zhu Du (2001). Steiner Trees in Industry. Países Bajos (Volumen 11, The perfect Phylogeny Problem)
- Xiuzhen Cheng and Ding-Zhu Du, 2001. Combinatorial Optimization. Steiner Trees in Industry, Volume 11 (Foreword, pp. vii-xi and pp. 30)
- Yang, Y.Y., Wing, O.: On a multinet wiring problem. IEEE Trans. Circuit Theory 20(3), 250– 252 (1973)

9 ANEXOS

9.1 Funciones de MATLAB

9.1.1 Función orden

```
function nuevostein=orden(stein,numarist)
nuevostein=[zeros(numarist,1),zeros(numarist,1),sort(stein(1:numaris
t,3))];
j=1;
for i=1:numarist
    for k=1:numarist
    if stein(k,3)==nuevostein(i,3)
        nuevostein(j,1)=stein(k,1);
        nuevostein(j,2)=stein(k,2);
        stein(k,3)=0;
        j=j+1;
    end
    end
end
end
```

9.1.2 Función steiner

```
function [nsteiner]=steiner(terminales, numvert)
numterm=length(terminales);
nsteiner=zeros(1,numvert-numterm);
nodostn=1:numvert;
for i=1:numvert
    for j=1:numterm
    if i==terminales(j)
      nodostn(i)=0;
    end
    end
end
j=1;
for i=1:numvert
    if nodostn(i)~=0
        nsteiner(j)=nodostn(i);
        j=j+1;
    end
end
end
```

9.1.3 Función steinerorden

```
function
[nsteinerord]=steinerorden(nsteiner,stein,numarist,solorden)
```

```
cont=zeros(1,length(nsteiner));
nsteinerord=cont;
for j=1:length(nsteiner)
    for i=1:numarist
        if stein(i,1)==nsteiner(j)|| stein(i,2)==nsteiner(j)
            if solorden(i)==1
                cont(j)=cont(j)+1;
            end
        end
    end
end
j=1;
for k=1:max(cont)
    for i=1:length(nsteiner)
        if cont(i)==k
            nsteinerord(j)=nsteiner(i);
            j=j+1;
        end
    end
end
end
```

9.1.4 Función solucionado

```
function solorden=solucionado(numarist,numvert,stein,terminales)
nuevostein=orden(stein, numarist);
nodopad=[1:numvert ; 1:numvert];
sol=1:numarist;

for i=1:numarist
    for j=1:numvert
        if nuevostein(i,1)==nodopad(1,j)
            m=j;
        end
        if nuevostein(i,2)==nodopad(1,j)
            n=j;
        end
    end
      if nodopad(2,m)==nodopad(2,n)
          sol(i)=0;
      else
        if nodopad(1,m)==nodopad(2,m)
            nodopad(2,m)=nodopad(2,n);
            sol(i)=1;
          for o=1:numvert
              if nodopad(2,o)==nodopad(1,m)
                  nodopad(2,o)=nodopad(2,m);
              end
          end
        else
            if nodopad(1,m)~=nodopad(2,m)
                nodopad(2,nodopad(2,m))=nodopad(2,n);
```

```
                p=nodopad(2,m);
              for l=1:numvert
                  if nodopad(2,l)==p
                      nodopad(2,l)=nodopad(2,nodopad(2,n));
                  end
              end
                sol(i)=1;
            end
        end
      end
end

// De aquí hacia abajo implementamos la parte de quitar las aristas
activas que se unen a nodos Steiner, los cuales se pueden podar.

nsteiner=steiner(terminales, numvert);
numsteiner=length(nsteiner);
nsteinerord=steinerorden(nsteiner, nuevostein, numarist, sol);
cont=zeros(1,numsteiner);
for j=1:numsteiner
    for i=1:numarist
    if nuevostein(i,1)==nsteinerord(j)||
nuevostein(i,2)==nsteinerord(j)
         if sol(i)==1
            cont(j)=cont(j)+1;
            num=i;
         end
    end
    end
        if cont(j)==1
            sol(num)=0;
        end
end

// Creamos la solución ordenada según stein y no según costes de
menor a mayor.

solorden=zeros(1,numarist);
for i=1:numarist
    for j=1:numarist
    if stein(i,1)==nuevostein(j,1) && stein(i,2)==nuevostein(j,2)
        solorden(i)=sol(j);
    end
    end
end

end
```

9.1.5 Función costealeatorio

```
function steincost=costealeatorio(stein, numarist)
for i=1:numarist
```

```
    stein(i,3)=floor(rand*100);
end
steincost=stein;
end
```

9.1.6 Función definitivo

```
function
[matrizsol,costesol]=definitivo(numvert,numarist,stein,terminales,N)
matrizsol=zeros(N,numarist);
costesol=zeros(numarist,N);
vectorsol=solucionado(numarist,numvert,stein,terminales);
   matrizsol(1,:)=vectorsol(:);
   costesol(:,1)=stein(:,3);
for i=2:N
steincost=costealeatorio(stein,numarist);
   costesol(:,i)=steincost(:,3);
vectorsol=solucionado(numarist,numvert,steincost,terminales);
   matrizsol(i,:)=vectorsol(:);
end
end
```

9.1.7 Función crossing

```
function hijo=crossing(padre,madre,stein,numvert,terminales)
numarist=length(padre);
hijo=zeros(1,numarist);
steincruce=[stein(:,1:2),16*ones(numarist,1)];
for i=1:numarist
    if padre(i)==madre(i)&& padre(i)==0
        hijo(i)=0;

    else
        hijo(i)=1;
        steincruce(i,3)=1;
    end
    if padre(i)~=madre(i)
        hijo(i)=1;
        steincruce(i,3)=round(14*rand()+1);
    end
end

//Hasta aquí hemos creado la matriz steincruce formada por la
superposición de los padres.
Los costes más bajos(coste=1) son las aristas que hay que coger para
formar la admisibilidad.

hijo=solucionado(length(padre),numvert,steincruce,terminales);
end
```

9.1.8 Función mutacion

```
function
matrizsol=mutacion(matrizsol,elegido,stein,numvert,nterminales)
numarist=size(matrizsol,2);
flat1=0;
mutado=zeros(1,numarist);
mutado(1,:)=matrizsol(elegido,:);
while flat1==0
    aleatorio=round((numarist-1)*rand()+1);
    if matrizsol(elegido,aleatorio)==1
        flat1=1;
    end
end
steinmutado=zeros(numarist,3);
steinmutado(1:numarist,1:2)=stein(1:numarist,1:2);

//Con este bucle conseguimos pasar a una matriz steinmutado la
solución mutada pero de forma que los arcos que haya que seleccionar
valgan 1 y el resto aleatorio.
Para que generemos una solución mutada sin bucles dentro del árbol.

for i=1:numarist
    if mutado(1,i)==1
        steinmutado(i,3)=1;
    else
        steinmutado(i,3)=round(14*rand()+1);
    end
end

//Le pongo valor infinito al arco que he mutado para que no se coja
y poder generar una solución.

steinmutado(aleatorio,3)=inf;
matrizsol(elegido,:)=solucionado(numarist,numvert,steinmutado,ntermi
nales);
end
```

9.1.9 Función selección

```
function elegido=seleccion(matrizsol,stein,bandera,q)
%generamos la matriz fitness que contiene en la primera columna
números del
%1 a N y en la segunda los correspondientes fitness según el orden
de las
%soluciones de la matrizsol
N=size(matrizsol,1);
fitness=[(1:N)',(matrizsol(:,:)*stein(:,3))];
%ordenamos fitness
fitnessordenado=[zeros(N,1),sort(fitness(:,2))];
for i=1:N
    for j=1:N
        if fitness(j,2)==fitnessordenado(i,2)
```

```
            fitnessordenado(i,1)=fitness(j,1);
        end
    end
end
if bandera==0
    elegido=round((N-1)*rand()+1);
else
%como obtener las diferentes probabilidades ordenadas según la
matriz
%fitnessordenado, es decir, de menor coste (mayor fitness) a mayor
coste
%(menor fitness)
probabilidad=zeros(N,1);
%RULETA de seleccion
if bandera==1
    sumfitness=0;
    for i=1:N
        sumfitness=sumfitness+1/fitness(i,2);
    end
    for i=1:N
        probabilidad(i)=1/fitnessordenado(i,2)/sumfitness;
    end
end
%RANKING de selección
if bandera==2||bandera==4
    probabilidad(1)=q;
    for i=1:N-1
        probabilidad(i+1)=q*(1-q)^i;
    end
probabilidad(1)=q+(1-sum(probabilidad));
end
%RANKING de reemplazo
if bandera==4
    probabilidad=sort(probabilidad);
end
%RULETA de reemplazo
if bandera==3
    sumfitness=0;
    for i=1:N
    sumfitness=sumfitness+fitness(i,2);
    end
    for i=1:N
        probabilidad(i)=fitnessordenado(i,2)/sumfitness;
    end
end
   %%esto se puede hacer con ruleta ranking e hipergeométrica
    rango=zeros(N,2);
    cont=0;
    for i=2:N
        cont=cont+probabilidad(i-1);
        rango(i,1)=cont;
        rango(i-1,2)=rango(i,1);
    end
    rango(N,2)=1;
```

```
    aleatorio=rand();
    for i=1:N
        if aleatorio>=rango(i,1)&& aleatorio<rango(i,2)
            elegido=fitnessordenado(i,1);
        end
    end
end
end
```

9.1.10 Función genetico1

```
function
[solucionfinal,coste,poblacion]=genetico1(numvert,numarist,stein,ter
minales,N,q,iteraciones)
tic
matrizsol=definitivo(numvert,numarist,stein,terminales,N);
fitnessiter=zeros(1,iteraciones);
poblacion=struct('fitness',[],'minimo',[],'solucion',[]);
%cont=0;
for i=1:iteraciones
    eleccion=rand();
    if eleccion>=0&&eleccion<0.15
        elegido=seleccion(matrizsol,stein,0,q);

matrizsol=mutacion(matrizsol,elegido,stein,numvert,terminales);
    end
    if eleccion>=0.15
        elegido1=seleccion(matrizsol,stein,1,q);
        elegido2=seleccion(matrizsol,stein,1,q);

[padre,madre]=prevencion(matrizsol(elegido1,:),matrizsol(elegido2,:)
,matrizsol);
        hijo=crossing(padre,madre,stein,numvert,terminales);
        elegido3=seleccion(matrizsol,stein,4,q);
        matrizsol(elegido3,:)=hijo(:);
    end
    fitness=[(1:N)',(matrizsol(:,:)*stein(:,3))]; %fitness de cada
muestra de la poblacion1 en cada una de las iteraciones
    fitnessiter(i)=min(fitness(:,2)); %acumula el mínimo fitness de
cada iteración
   %cont=cont+1
end
for j=1:N
    if fitness(j,2)==fitnessiter(iteraciones)
        posicion=j;
    end
end
ultimasolucion=matrizsol(posicion,:);
minpoblacion=zeros(30,1);
minpoblacion(1)=min(fitnessiter);
poblacion(1).fitness=fitnessiter;
poblacion(1).minimo=minpoblacion(1);
poblacion(1).solucion=ultimasolucion;
```

```
for j=1:29
    matrizsol=definitivo(numvert,numarist,stein,terminales,N);
    steincost=costealeatorio(stein,numarist);

matrizsol(1,:)=solucionado(numarist,numvert,steincost,terminales);
    for i=1:iteraciones
    eleccion=rand();
    if eleccion>=0&&eleccion<0.15
        elegido=seleccion(matrizsol,stein,0,q);

matrizsol=mutacion(matrizsol,elegido,stein,numvert,terminales);
    end
    if eleccion>=0.15
        elegido1=seleccion(matrizsol,stein,1,q);
        elegido2=seleccion(matrizsol,stein,1,q);

[padre,madre]=prevencion(matrizsol(elegido1,:),matrizsol(elegido2,:)
,matrizsol);
        hijo=crossing(padre,madre,stein,numvert,terminales);
        elegido3=seleccion(matrizsol,stein,4,q);
        matrizsol(elegido3,:)=hijo(:);
    end
    fitness=[(1:N)',(matrizsol(:,:)*stein(:,3))]; %fitness de cada
muestra de la poblacion1 en cada una de las iteraciones
    fitnessiter(i)=min(fitness(:,2)); %acumula el mínimo fitness de
cada iteración
    end
    for k=1:N
        if fitness(j,2)==fitnessiter(iteraciones)
            posicion=k;
        end
    end
   ultimasolucion=matrizsol(posicion,:);
   minpoblacion(j+1)=min(fitnessiter);
   poblacion(j+1).fitness=fitnessiter;
   poblacion(j+1).minimo=minpoblacion(j+1);
   poblacion(j+1).solucion=ultimasolucion;
end
coste=min(minpoblacion);
for j=1:30
    if minpoblacion(j,1)==coste
        posicion=j;
    end
end
solucionfinal=poblacion(posicion).solucion;
toc
end
```