Proyecto Fin de Grado

Ingeniería de Organización Industrial

Análisis experimental de las estrategias antibluces de Miller-Tucker-Zemlin y Desrochers-Laporte para problemas de selección de árboles en grafos

Autor: Isabel Delgado Gallo

Tutor: Jose Manuel García Sánchez

Dpto. Organización Industrial
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Proyecto Fin de Grado

Ingeniería de Organización Industrial

Análisis experimental de las estrategias antibluces de Miller-Tucker-Zemlin y Desrochers-Laporte para problemas de selección de árboles en grafos

Autor:
Isabel Delgado Gallo

Tutor:
José Manuel García Sánchez
Profesor titular

Dpto. de Organización Industrial
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Proyecto Fin de Grado: Análisis experimental de las estrategias antibluces de Miller-Tucker-Zemlin y Desrochers-Laporte para problemas de selección de árboles en grafos

Autor: Isabel Delgado Gallo

Tutor: José Manuel García Sánchez

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

Resumen

Sabemos que, a la hora de modelar un problema para poder resolverlo, podemos modelarlo de diferentes formas llegando con todas ellas a la solución óptima.

En este trabajo lo que se hará será analizar dos estrategias de formulación para dos tipos de problemas asociados a la selección de nodos en grafos.

Concretamente, se trata del problema Minimum Spanning Tree y del problema de Steiner. En el primero se busca la selección del árbol de expansión a coste mínimo, donde deben seleccionarse todos los nodos del grafo, mientras que en el problema de Steiner solo es necesario seleccionar los nodos identificados como terminales.

La resolución de estos dos problemas se realiza con dos estrategias de formulación matemática.

Para el análisis de las dos estrategias se va a utilizar una batería de problemas existente en Internet. Se trata de la OR library, que se encuentra en la página web del departamento de investigación operativa del Imperial College de Londres.

A partir de los datos proporcionados por dicha batería, y a través de la creación de un programa en C, obtenemos unos archivos legibles para el programa LINGO.

De esta forma lo que se pretende es estudiar a través de los tiempos de resolución del programa que tipo de formulación sería más eficiente para cada tipo de problema según distintos niveles de dificultad.

Esto será muy útil para cuando los problemas sean de gran tamaño ya que conseguiremos llegar a la solución óptima empleando menos tiempo.

Índice

ÍNDICE DE FIGURAS

Figura 34. Gráfico de columnas apiladas para comparar las diferencias entre el resultado continuo y el resultado entero para el problema MST.

Figura 35. Gráfico de columnas apiladas para comparar las diferencias entre el resultado continuo y el resultado entero para el problema Steiner.

1 INTRODUCCIÓN - OBJETIVO DEL PROYECTO

En este trabajo se presenta el análisis experimental de dos estrategias de formulación para evitar ciclos en la obtención de árboles de un grafo. En concreto, las estrategias son las propuestas por Miller, Tucker y Zemlin (MTZ) y Desrochers y Laporte (DL) para el problema del viajante de comercio.

Los problemas de optimización sobre los que se van a comparar las estrategias anteriores serán el problema Minimum Spanning Tree (MST) y Steiner.

En el problema MST se busca la selección del árbol de expansión a coste mínimo, donde deben seleccionarse todos los nodos del grafo, mientras que en el problema de Steiner solo es necesario seleccionar los nodos identificados como terminales.

El análisis se hará sobre la batería OR- Library desarrollada en la página web del departamento de investigación operativa del Imperal College de Londres.

Los modelos matemáticos de ambos se han generado con el uso de un programa en C que lee los datos e implementa el modelo en un formato compatible para las librerías de optimización LINGO.

La resolución de los modelos se ha realizado con la librería LINGO 10.0.

El análisis computacional incluye el análisis de los valores continuos, el número de iteraciones, el tiempo empleado, la memoria usada, y la solución obtenida.

Los problemas analizados están agrupados según la dificultad de su resolución en clase “b”, “c” y “d” siendo estos últimos los más difíciles de resolver. En las dos primeras formulaciones para el primer tipo de problema hemos analizado solamente los problemas de clase “b” y “c” dado que no contábamos con las soluciones óptimas y no íbamos a poder llegar a resolver los de tipo “d” en menos de una hora. En cambio, para la tercera y cuarta formulación hemos analizado también los de tipo “d” ya que en este caso si contábamos con la solución óptima y aunque el programa no terminase de resolver el problema en una hora sí que podemos evaluar el error de la solución obtenida.

Ya que en las dos primeras formulaciones lo que se busca es encontrar la solución óptima teniendo en cuenta que todos los nodos del problema deben pertenecer a la solución, y en cambio en las dos últimas formulaciones solo tienen que pertenecer a la solución los nodos terminales, a través de las dos primeras formulaciones vamos a llegar a un resultado y a través de las dos últimas vamos a llegar a otro.

De esta forma podremos estudiar los problemas para así poder llegar a la conclusión de como modelar nuestras restricciones según nuestro tipo y tamaño de problema para poder resolverlos de manera más eficiente ya que los tiempos de resolución variarán según la formulación empleada.

En nuestro caso los problemas que vamos a estudiar van a ser problemas de grafos.

Empezaremos por un problema que tiene 13 nodos y 19 aristas, y terminaremos con uno que tiene 1000 nodos y 6887 aristas.

La siguiente imagen es la representación gráfica del primer problema de nuestra batería.

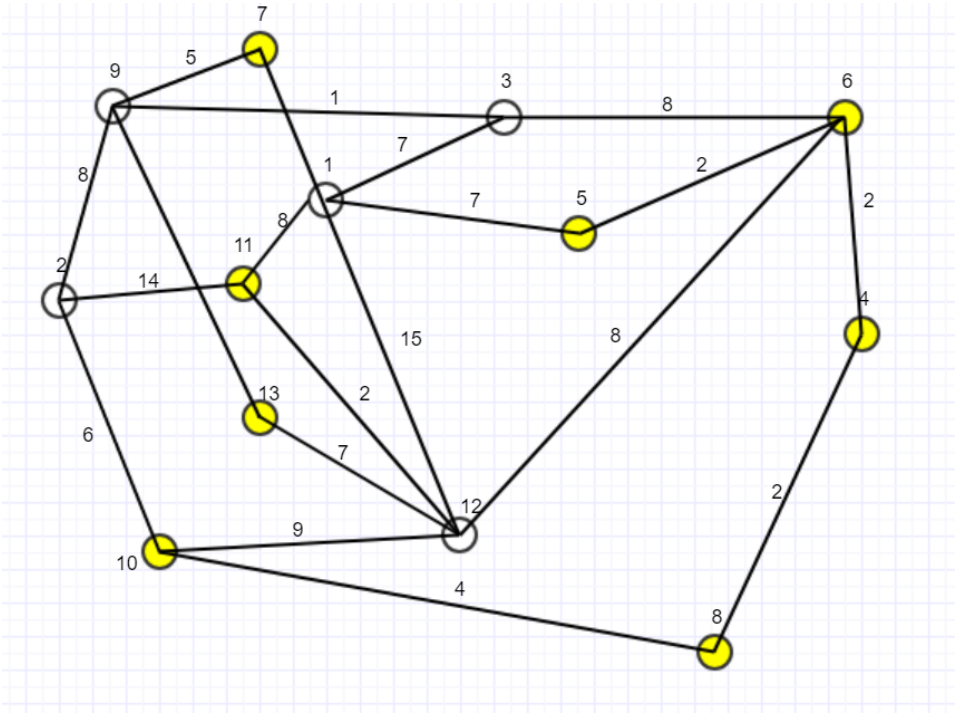

Figura 1. Fuente: Elaboración propia.

Este estudio se ha llevado a cabo de la siguiente forma:

1) En primer lugar hemos realizado las formulaciones de los problemas.

2) Posteriormente y ya que los problemas se han dado en un determinado formato, ha sido necesaria la creación de dos programas en C de tal forma que pasándole los archivos de texto donde estaba la información de los diferentes problemas, cambiase el formato de estos a lenguaje que el software para programación lineal LINGO fuera capaz de leer. Esto lo hemos realizado acorde a las dos formulaciones que habíamos realizado a papel.

3) Una vez hecho esto hemos resuelto los problemas con LINGO y hemos apuntando los diferentes tiempos de resolución de cada uno de los programas según cada formulación.

Debido al gran tamaño de alguno de los problemas, cuando intentábamos resolver un problema a través de lingo y pasaba una hora el programa funcionando y aún no se había llegado al óptimo, hemos interrumpido el programa y apuntado a que resultado había llegado.

Ya que por otro lado contábamos con la solución óptima a la que tendrían que haber llegado los problemas que no llegaron al óptimo en una hora, se han comparado los resultados obtenidos con el resultado al que debería haber llegado, de esta forma hemos podido calcular el porcentaje de error con respecto al óptimo.

Una vez hechos estos tres pasos, lo único que ha habido que hacer es estudiar los resultados obtenidos y a partir de ahí sacar conclusiones.

A continuación, se procede a mostrar un diagrama de flujo que representa de forma esquemática el procedimiento seguido en el trabajo.

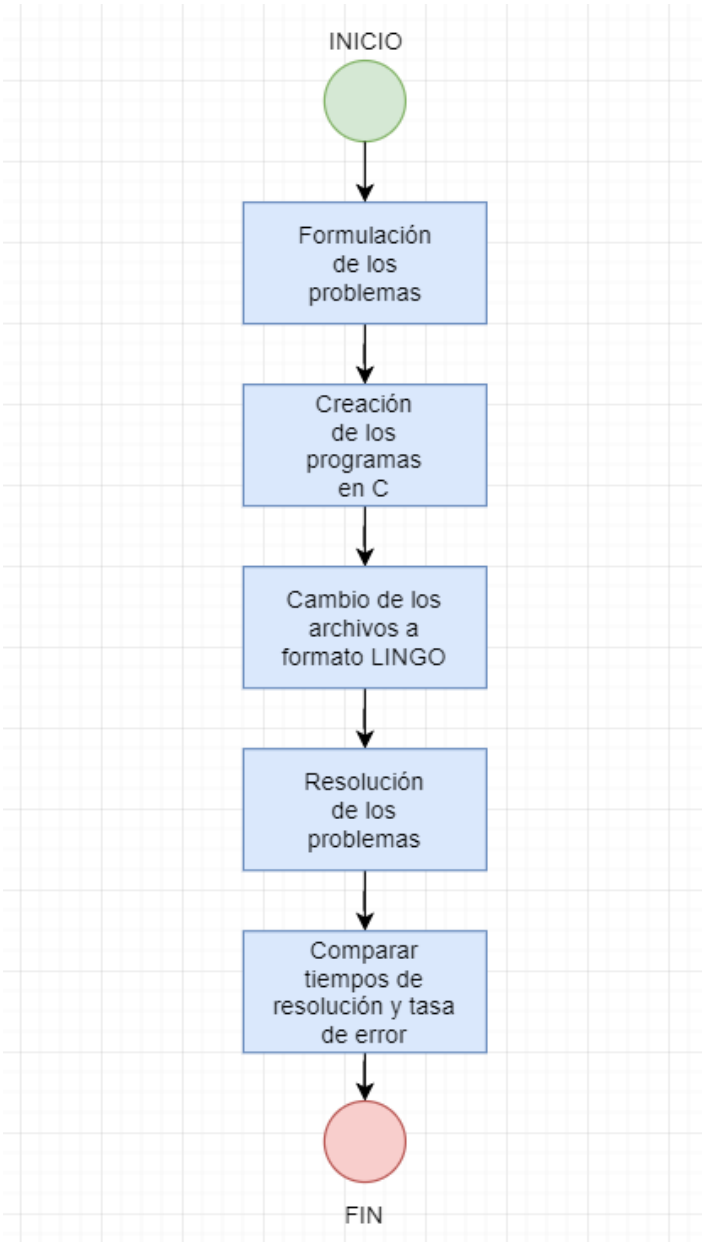

Figura 2. Fuente: Elaboración propia.

2 DESCRIPCIÓN DEL PROBLEMA

Ya que este trabajo se centra en la resolución de problemas de grafos. Se va a explicar brevemente algunos de los conceptos más importantes.

2.1 Grafos

La palabra grafos viene de la palabra griega graphein que puede traducirse como "grabar o escribir".

Para las ciencias de la computación y las matemáticas, un grafo es una representación gráfica de un problema a través de la unión por medio de líneas de diferentes puntos. A estos puntos los llamaremos nodos, y a las líneas aristas.

El primer documento escrito que trata sobre que son los grafos, fue realizado por Leonhard Euler en el año 1736. El cual fue un matemático, físico y filósofo suizo muy conocido por el número de Euler que aparece en muchas fórmulas de cálculo y física.

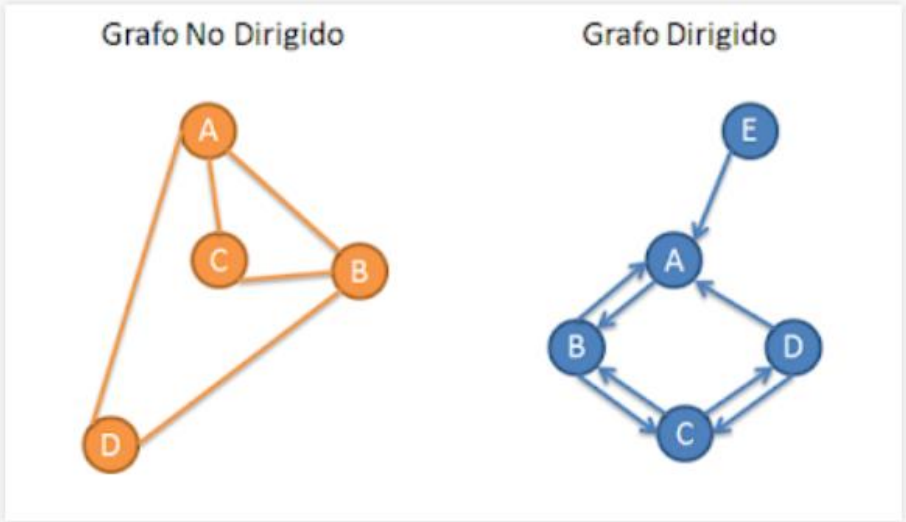

Figura 3. Fuente: http://estructurasdedatoslk.blogspot.com/2015/12/java-implementacion-de-grafos.html

Hay muchos tipos de grafos, los cuales se pueden clasificar principalmente en los siguientes:

- Grafo simple: Aquel que conecta dos nodos a través de una arista.
- Multígrafo: Aquel que conecta dos nodos a través de más de una arista
- Grafo orientado o grafo dirigido: Son aquellos grafos en los que sus aristas tienen una orientación la cual viene indicada con una flecha, esto lo que nos indica es que en algunos casos el flujo solo va a poder ir en una dirección y no en ambas como en un grafo simple.
- Grafo completo: Es aquel en el que todos sus nodos están conectados entre ellos por una arista.

Dentro de los grafos hay muchas variantes ya que podemos añadirles además a las aristas un peso o una probabilidad.

En la actualidad los grafos se pueden utilizar para la resolución de diferentes tipos de problemas.

Una de sus aplicaciones más comunes hoy en día es en el transporte. El cual hoy en día es uno de los sectores que más aporta a la generación de riqueza, de hecho, desde hace años está por encima del 5% del valor añadido bruto nacional de España. El transporte por carretera es el más utilizado, muy por encima del transporte ferroviario marítimo y aéreo.

El transporte trata una situación en la cual se envía un bien desde uno o varios puntos de origen hasta uno o varios puntos de destino, a la vez que se satisfacen una serie de restricciones.

A través del estudio de las diferentes rutas podríamos llegar a saber cuál es el camino más corto entre dos puntos en una red de carreteras o también como podríamos organizar la distribución de mercancías entre los puntos de almacenaje y los puntos de distribución.

Debido a que la mayoría de las empresas suelen tener muy en cuenta todos los factores que puedan influir en un aumento o disminución de los costes, realizar de la forma más eficiente posible las rutas puede llegar a suponer una gran disminución de los costes relacionados con el carburante, tiempo y mano de obra.

Es muy importante que, a la hora de intentar hacer las diferentes rutas de la forma más eficiente posible, analicemos todos los factores que influyen en nuestra solución, tales como el tráfico o la zona geográfica.

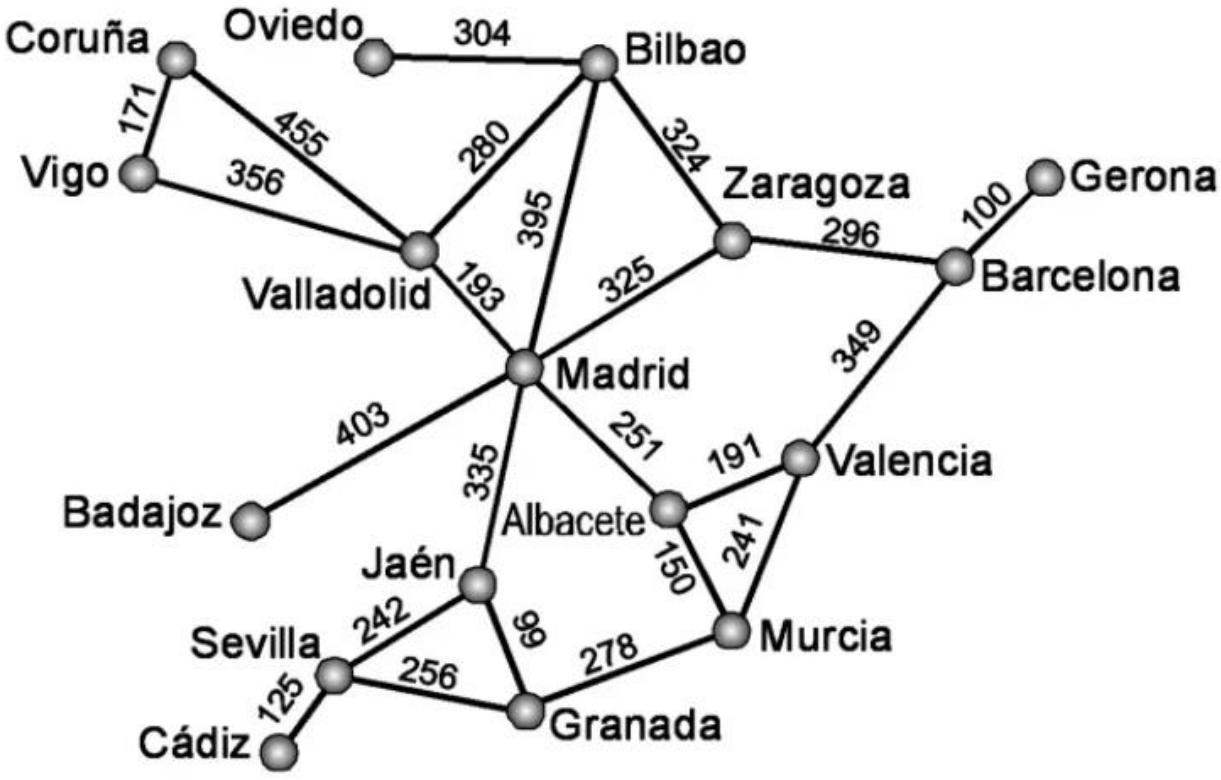

Figura 4. Fuente: https://rootear.com/desarrollo/grafos

También es muy importante tener en cuenta que hay veces en las que todos los nodos del problema no están conectados entre sí por una arista. Por ejemplo, en el caso que quisiéramos ir a todos las ciudades representadas en la imagen anterior, tendríamos que tener en cuenta que entre ciudades como por ejemplo Sevilla y Coruña no habría una carretera directa, por lo que sería necesario el pasar primero por otra ciudad.

El problema del viajante es un caso particular de los problemas de transporte, y es uno de los más famosos de la matemática computacional.

Este tipo de problema trata de determinar el recorrido de tal forma que, empezando por una ciudad determinada, se pase por todas las demás una sola vez para volver finalmente a la primera, de manera que se minimice la distancia total recorrida. Este problema fue formulado por primera vez en 1930 y es uno de los problemas de optimización más estudiado debido a su gran aplicación.

En este tipo de problemas tomaríamos los nodos como las diferentes ciudades o puntos en el mapa a los que tenemos que llegar o desde donde podemos salir, y las aristas serían las diferentes combinaciones que hay para movernos entre ellos, que en este caso serían las carreteras.

Aunque puede parecer que llegar a la solución no es una tarea muy difícil ya que sólo habría que probar cuál de las posibles combinaciones de rutas sería la óptima, cuando el número de ciudades es elevado, las posibles combinaciones aumentan de manera exponencial.

Por ejemplo, si quisiéramos resolver el problema del viajante y tuviéramos 20 ciudades, ¡las posibles combinaciones de estas ciudades serían 20!. Si suponemos que nuestro ordenador es capaz de evaluar una solución por cada microsegundo, para 20 ciudades tardaría aproximadamente 77.146 años en resolver el problema.

Debido a que las posibles combinaciones aumentan de manera exponencial este problema puede encuadrarse dentro de los problemas llamados NP-completos. Que son considerados los más difíciles de resolver dentro de los problemas NP.

En nuestro caso, el problema MST pertenece a la categoría de problemas P, mientras que el problema de Steiner pertenece a la categoría de los NP-Completos.

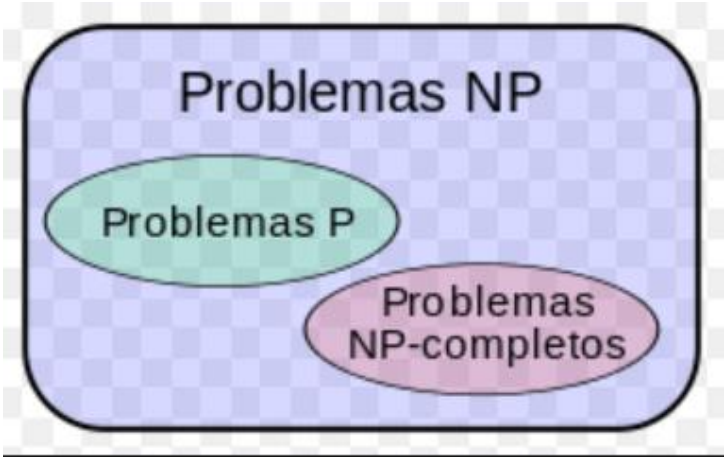

Figura 5. Fuente: https://es.wikipedia.org/wiki/Clases_de_complejidad_P_y_NP

La red de internet se podría considerar también como un grafo dirigido en el que cada página web queda representada por un Nodo y en el caso en el que a través de una de las páginas web se pueda acceder a otra habrá una arista "enlace" que conectará estas dos páginas web.

También podríamos considerar la red de tal forma que todos los ordenadores que hay conectados a internet fueran los nodos y en el momento en el que dos ordenadores estuvieran conectados entre sí tendrían una arista que los uniera. En este caso deberíamos también considerar que no siempre están los mismos ordenadores conectados por lo que sería un grafo en el que sus conexiones irían apareciendo y desapareciendo.

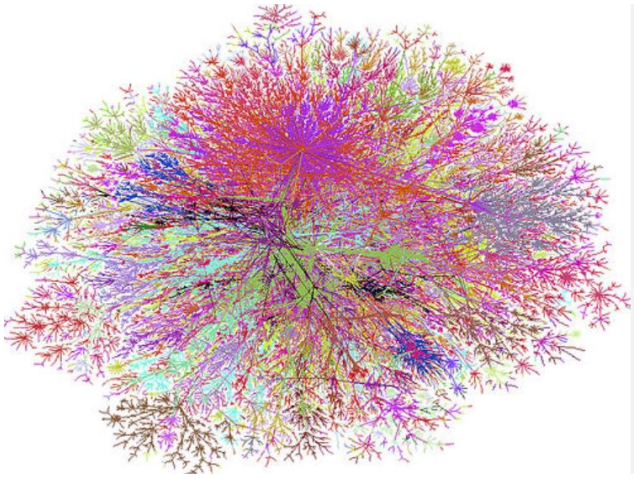

Figura 6. Fuente: http://informatica.blogs.uoc.edu/2012/11/12/la-belleza-de-las-redes-herramientas-de-visualizacion-de-grafos/

2.2 Árbol de expansión mínima

El árbol de expansión consiste en que dado un grafo tal que G(N,A) dónde "N" es el número de nodos y "A" el número de aristas queremos conectar todos sus nodos a través de (N-1) aristas al menor coste posible.

Al crear el árbol es muy importante que no existan ciclos, además debe existir una ruta entre cada par de vértices. Un grafo puede tener muchos árboles de expansión.

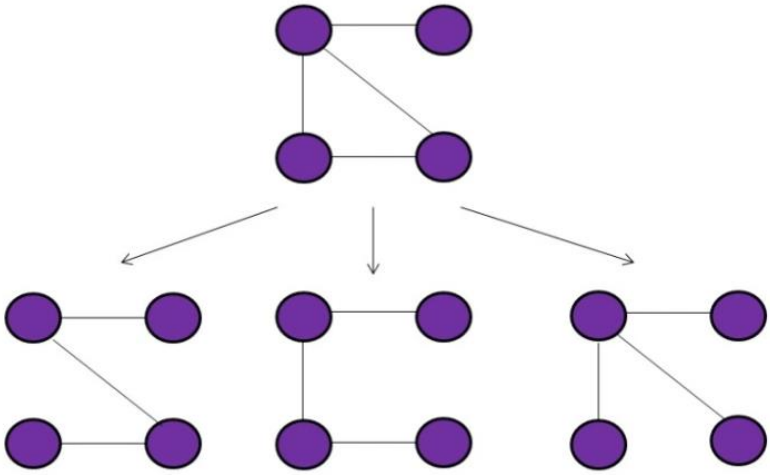

Figura 7. Fuente: https://jariasf.wordpress.com/2012/04/19/arbol-de-expansion-minima-algoritmo-de-kruskal/

En la figura anterior podemos ver que en los tres casos se cumplen las condiciones de unir todos los nodos a través de (N-1) aristas que en este caso N es 4 y el número de aristas que debe haber por consiguiente es 3.

El árbol de expansión mínima consiste en la creación del árbol de la misma forma, pero teniendo en cuenta también otra variable, el peso de las aristas. Y lo que queremos conseguir es que la suma de los pesos de todas las aristas incluidas en la solución sea lo más pequeña posible.

Usando el ejemplo anterior, le añadiremos el peso a cada una de las aristas y quedaría tal que así:

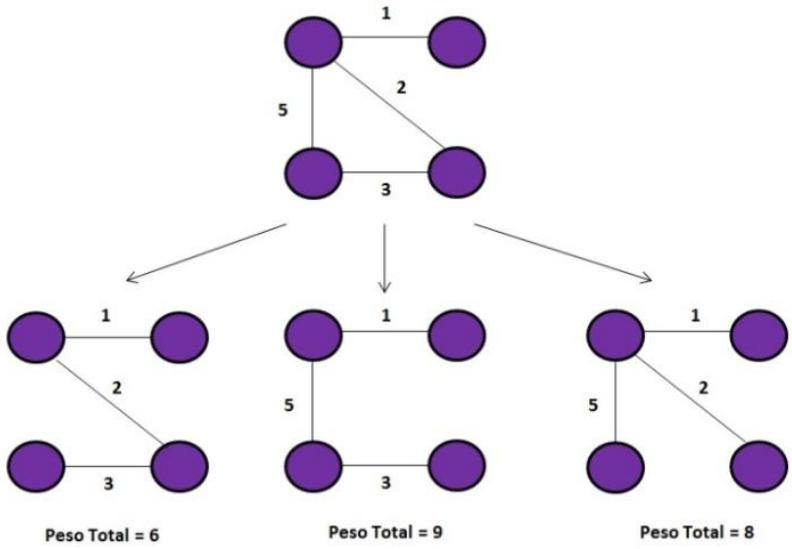

Figura 8. Fuente: https://jariasf.wordpress.com/2012/04/19/arbol-de-expansion-minima-algoritmo-de-kruskal/

Aunque como dijimos anteriormente las tres soluciones posibles son árboles de expansión, sólo se consideraría como árbol de expansión mínima el primero de todos, que es el que tiene el peso total de 6 que es el menor.

Este tipo de problemas pueden ser resueltos según varios algoritmos, los dos algoritmos más conocidos para la resolución de este tipo de problemas son:

-Algoritmo de Kruskal: Primero ordenamos las aristas del grafo según su peso de forma creciente. Posteriormente los examinamos empezando por aquella de menor costo y si al incorporar esta arista al grafo no se forman ciclos, la añadiremos a nuestra solución. La construcción de la solución terminará cuando ya tengamos las n-1 aristas seleccionadas o cuando ya se hayan examinados los todos los arcos de la red.

-Algoritmo de Prim's: Consiste en construir un árbol partiendo desde uno de los nodos de forma arbitraria, este nodo lo unimos con los vecinos más cercanos previa examinación de las aristas que les llegan para saber cuál es la que posee menor coste. Hacemos esto hasta revisar todas las aristas de la red.

Las dos primeras formulaciones consisten en resolver los problemas según árbol de expansión mínima.

2.2.1 FORMULACIÓN DEL PROBLEMA MST CON LA ESTRATEGIA MTZ

Elementos participantes del Problema: Pueden ser de naturaleza diversa, desde personas, lugares, herramientas… Los diferentes elementos pueden tener ciertas características a las cuales llamaremos variables asociadas. Estas variables participan en las acciones que se producen en el sistema y también tienen que cumplir las restricciones de este.

- Llamamos N al conjunto de nodos del grafo.
- Llamamos A al conjunto de aristas del grafo.
- Cj: es el valor del coste asignado a la arista "j".
- Xi: es el valor del nivel del nodo "i".
- Ri: es una variable binaria tal que el primer nodo que vayamos a poner en nuestro árbol tenga valor 1 y los otros tengan valor 0. Esto es para diferenciar los nodos, ya el primer nodo que pongamos no va a tener nodo padre, por lo que va a tener que cumplir otro tipo de condiciones que los nodos que tienen padre.
- Origenj: es el número del nodo origen de la arista "j".
- Destinoj: es el número del nodo destino de la arista "j".

Variables de decisión: Son acciones que se producen en el sistema para las cuales es necesario determinar su valor. Se encuentran asociadas a los elementos participantes del problema. Las variables son acciones simples, pero es posible poseer variables de decisión cuyos valores estén condicionados al valor de otras variables de decisión. Los valores que pueden asignarse a estas variables pueden ser un valor binario (0 o 1), un valor entero o un valor continuo.

- αj: En nuestra formulación esta será la única variable de decisión que tendremos, tendrá un valor binario, valdrá uno cuando se seleccione ese arco como parte de la solución y cero cuando no.

Restricciones: Son normas definidas en el sistema que deben cumplirse.

- $\sum_{\forall j/Destino j=i} \alpha j = 1 \qquad \forall i/Ri=0$

La primera restricción sirve para asegurarnos de que todos los nodos que no sean raíz tienen nodo padre, poniendo la condición de que deben ser el destino de alguna de las aristas que forman el árbol de la solución.

- $\sum_{\forall j/Origen j=i} \alpha j \geq 1 \qquad \forall i/Ri=1$

La segunda restricción es para que aquellos nodos que sean nodo raíz tienen que ser padre de al menos un nodo.

- $-x_{Origenj} + x_{Destinoj} + (N-1) * \alpha j \leq N - 2 \quad , \forall j / R_{Origenj} = 0, R_{Destinoj} = 0$

La tercera restricción hace referencia a los niveles. Con esta restricción impedimos la formación de bucles al imponer que el nivel del nodo padre siempre debe ser mayor que el nivel del nodo que estemos analizando.

- $x_{Origenj} \geq 1 \qquad , \forall j / R_{Origenj} = 0$

La cuarta restricción en cambio nos dice que un nodo que no sea raíz tiene que tener un nivel mayor o igual a uno. Ya que el único nodo que tendrá nivel cero será el nodo raíz.

- $x_{Origenj} \leq N - 1 \qquad , \forall j / R_{Origenj} = 0$

La quinta restricción nos dice que el nivel de un nodo que no sea padre tiene que ser menor que el número de nodos menos uno.

- $\alpha j \in [0,1] \qquad , \forall j \in A$

La sexta restricción sirve para imponer que nuestra variable de decisión tiene que tomar un valor binario.

Función objetivo: La función objetivo lo que busca es minimizar el coste de las aristas que pertenecen a la solución.

- $\text{Min } \Sigma \text{ Cj} * \alpha j \qquad , \forall j \in A$

2.2.2 FORMULACIÓN DEL PROBLEMA MST CON LA ESTRATEGIA DL

Para la segunda formulación tanto los elementos participantes del problema como las variables de decisión y la mayoría de las restricciones son iguales a la primera formulación.

La única variante que tiene con respecto a la primera formulación es que la tercera restricción (la cual hacía referencia a los niveles) se sustituye por otras dos restricciones.

- $-x_{Origenj} + x_{Destinoj} + (N-1) * \alpha j + (N-3) * \alpha_{j+(\frac{A}{2})} \leq N-2$
  $\forall j / j \leq (\frac{A}{2}), R_{Origenj} = 0, R_{Destinoj} = 0$
- $-x_{Origenj} + x_{Destinoj} + (N-1) * \alpha j + (N-3) * \alpha_{j-(\frac{A}{2})} \leq N-2$
  $\forall j / j > \left(\frac{A}{2}\right) + 1, R_{Origenj} = 0, R_{Destinoj} = 0$

Estas dos nuevas restricciones imponen que el nivel del nodo padre siempre debe ser mayor que el nivel del nodo que estemos analizando.

Tanto $\alpha_{j+(\frac{A}{2})}$ como $\alpha_{j-(\frac{A}{2})}$ se refieren a arista contraria.

Lo de arista contraria quiere decir que, si estamos comprobando la arista que va desde el nodo uno al nodo dos, su arista contraria será aquella que va desde el nodo dos al nodo uno. Con estas dos restricciones impedimos la formación de bucles.

2.3 Problema de Steiner

El problema de Steiner consiste en encontrar el árbol de expansión a coste mínimo teniendo en cuenta que, sólo es necesario seleccionar los nodos identificados como terminales. Fue propuesto por el alemán Jacob Steiner a principios del siglo XIX.

Debido a que es un problema genérico, engloba un gran abanico de situaciones y problemas que son modelables mediante este.

Es uno de los problemas más clásicos y estudiados dentro del campo de la optimización combinatoria, y diversos algoritmos han sido propuestos para su resolución. Esto es debido a que cuando las instancias del problema crecen en tamaño y complejidad, los algoritmos exactos tradicionalmente empleados dejan de ser aplicables a la práctica y surgen como alternativa, técnicas heurísticas de resolución que, aunque muchas de las veces no nos dan el resultado óptimo, nos da un resultado aproximado de bastante calidad en un tiempo razonable.

La mayoría de las versiones del problema de Steiner son NP-completo.

En el problema del árbol de Steiner, a diferencia del problema de árbol de expansión mínima, los nodos se dividen en terminales y no terminales. Los nodos terminales deben incluirse en la solución mientras que los nodos no terminales (también llamados nodos Steiner), pueden pertenecer a la solución o no. La única finalidad de incluir los nodos Steiner en la solución es para reducir el coste o maximizar el beneficio.

En la siguiente figura podemos observar un ejemplo de un problema Steiner, donde los nodos terminales vienen representados por el color rojo y los nodos Steiner por el color azul.

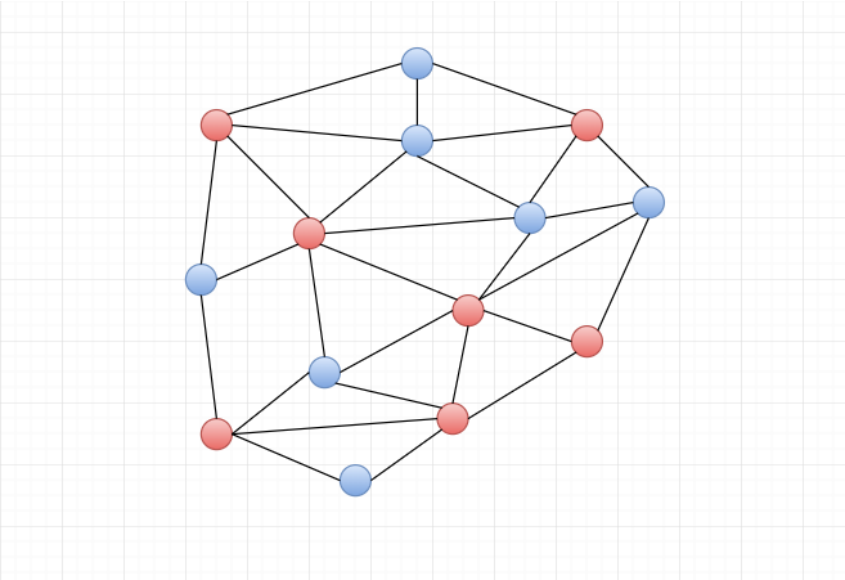

Figura 9. Fuente: Elaboración propia.

La solución de este problema es un subgrafo en el que se deben incluir todos los nodos terminales, pero no necesariamente los Steiner por lo que estos últimos solo pertenecerán a la solución si al incluirlos, ésta mejora. Una posible solución de este problema sería la siguiente.

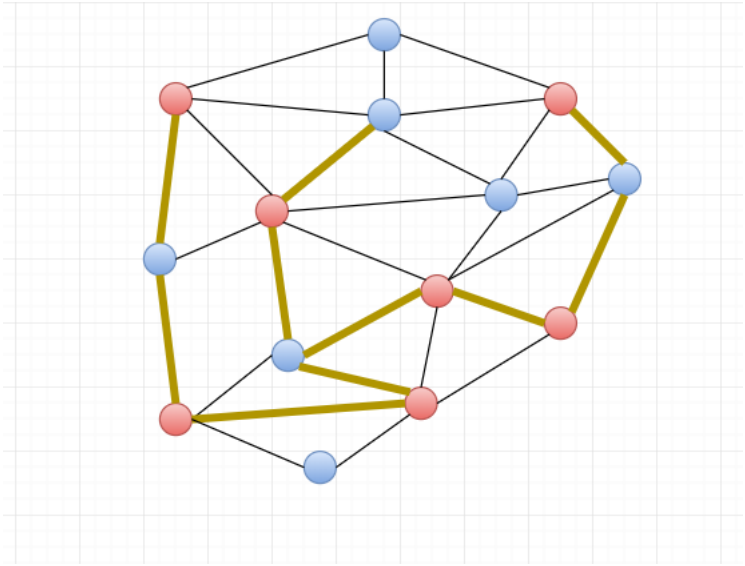

Figura 10. Fuente: Elaboración propia.

El camino naranja indica la solución del problema, donde podemos observar que en este caso se han añadido como parte de la solución, además de todos los nodos terminales, cuatro nodos Steiner.

Si resulta que el problema no contiene nodos Steiner, entonces la solución coincidirá con el árbol de expansión mínima.

Si contiene exactamente (n-2) nodos Steiner se dice que es completo.

Las dos últimas formulaciones consisten en resolver los problemas según el problema de Steiner, por lo que para estas dos formulaciones si tendremos en cuenta los nodos terminales y el valor último que hay que sumarle a la función objetivo.

2.3.1 FORMULACIÓN DEL PROBLEMA STEINER CON LA ESTRATEGIA MTZ

Elementos participantes del Problema: Para la tercera formulación hay que añadir dos elementos más aparte de los que ya identificamos para las dos primeras formulaciones.

- Ti: Es una variable binaria que vale uno cuando el nodo "i" es terminal y 0 cuando no lo es.
- M: Valor que hay que sumarle a la función objetivo

Variables de decisión: Además de la alfa ya mencionada anteriormente, para la tercera formulación identificaremos otra variable de decisión.

- βi: Tendrá un valor binario, valdrá uno cuando el nodo i se seleccione como parte de la solución y cero cuando no.

Restricciones:

- $\sum_{\forall j/Destino_j=i} \alpha_j = 1 \qquad \forall i/Ri=0, Ti=1$

La primera restricción sirve para asegurarnos de que todos los nodos que no sean raíz y que además sean terminales, tienen nodo padre, poniendo la condición de que deben ser el destino de alguna de las aristas que forman el árbol de la solución.

- $\sum_{\forall j/Destino_j=i} \alpha_j = \beta_i \qquad \forall i/Ri=0, Ti=0$

La segunda restricción impone que si hay alguna arista que pertenezca a la solución cuyo destino es un nodo no terminal, ese nodo debe aparecer en la solución. A la vez que impone que, si un nodo pertenece a la solución, debe haber alguna arista que le llegue. Esta restricción se debe a que una arista debe conectar siempre dos nodos, no puede haber una arista que conecte un nodo con nada. Además de que un nodo que no sea terminal y aparezca en la solución siempre tiene que tener un padre.

- $\alpha_j \leq \beta_i \qquad \forall i/Ri=0, Ti=0, \forall j/Origen_j=i$

La tercera restricción nos dice que si hay una arista que pertenezca a la solución cuyo origen es un nodo no terminal, ese nodo debe aparecer en la solución. Esta restricción se debe a que una arista debe conectar siempre dos nodos.

- $\sum_{\forall j/Origen=i} \alpha_j \geq 1 \qquad \forall i/Ri=1$

La cuarta restricción impone que el nodo raíz tiene que ser el origen de al menos una de las aristas pertenecientes a la solución.

- $x_{Origenj} + x_{Destinoj} + (N-1) * \alpha j \leq N - 2 \quad , \forall j / R_{Origenj} = 0, R_{Destinoj} = 0$

La quinta restricción hace referencia a los niveles. Con esta restricción impedimos la formación de bucles al imponer que el nivel del nodo padre siempre debe ser mayor que el nivel del nodo que estemos analizando.

- $x_{Origenj} \geq 1 \quad , \forall j / R_{Origenj} = 0$

La sexta restricción en cambio nos dice que un nodo que no sea raíz tiene que tener un nivel mayor o igual a uno. Ya que el único nodo que tendrá nivel cero será el nodo raíz.

- $x_{Origenj} \leq N - 1 \quad , \forall j / R_{Origenj} = 0$

La séptima restricción nos dice que el nivel de un nodo que no sea padre tiene que ser menor que el número de nodos menos uno.

- $\alpha j \in [0,1] \quad , \forall j \in A$

La octava restricción sirve para imponer que nuestra variable de decisión alfa tiene que tomar un valor binario.

- $\beta i \in [0,1] \quad , \forall i \in N$

La novena restricción sirve para imponer que nuestra variable de decisión beta tiene que tomar un valor binario.

Función objetivo: La función objetivo lo que busca es minimizar el coste de las aristas que pertenecen a la solución.

- $\text{Min } \Sigma \, Cj * \alpha j + M \quad , \forall j \in A$

2.3.2 FORMULACIÓN DEL PROBLEMA STEINER CON LA ESTRATEGIA DL

Para la cuarta formulación tanto los elementos participantes del problema como las variables de decisión y la mayoría de las restricciones son iguales a la tercera formulación.

La única variante que tiene con respecto a la tercera formulación es que la quinta restricción (la cual hacía referencia a los niveles) se sustituye por otras dos restricciones.

- $-x_{Origenj} + x_{Destinoj} + (N-1) * \alpha j + (N-3) * \alpha_{j+(\frac{A}{2})} \leq N-2$
  $\forall j / j \leq (\frac{A}{2}), R_{Origenj} = 0, R_{Destinoj} = 0$
- $-x_{Origenj} + x_{Destinoj} + (N-1) * \alpha j + (N-3) * \alpha_{j-(\frac{A}{2})} \leq N-2$
  $\forall j / j > \left(\frac{A}{2}\right) + 1, R_{Origenj} = 0, R_{Destinoj} = 0$

Estas dos nuevas restricciones imponen que el nivel del nodo padre siempre debe ser mayor que el nivel del nodo que estemos analizando.

Tanto $\alpha_{j+(\frac{A}{2})}$ como $\alpha_{j-(\frac{A}{2})}$ se refieren a arista contraria.

Con estas dos restricciones impedimos la formación de bucles.

Si nos fijamos, nos damos cuenta de que la cuarta formulación varía con respecto a la tercera formulación lo mismo que la segunda formulación a la primera.

3 IMPLEMENTACIÓN

3.1 Cambio de formato a un archivo txt a través de C

A través de un compilador de lenguaje de programación C, se ha realizado un programa que permite la lectura de ficheros de datos y la creación de otro archivo de texto.

La finalidad de este programa es cambiar el formato de los ficheros que se han dado y pasarlo a un formato que el programa LINGO (el cual ha sido el que se ha utilizado para la resolución de la batería de problemas) pueda leer.

Esto se ha realizado con el fin de no tener que cambiar el formato uno a uno de los archivos de texto, ya que es una tarea muy repetitiva y que nos llevaría mucho tiempo debido a que algunos de los problemas analizados tenían una gran cantidad de datos.

El formato inicial de los archivos de texto era el siguiente:

4 5 → El primer número hace referencia al número de nodos y el segundo número al número de arcos.

1 2 4

1 3 1

1 4 2

2 3 5

3 4 3→ Estas filas hacen referencia a los arcos. El primer número es el nodo origen, el segundo número es el nodo destino y el tercer número es el coste de ese arco.

2 → Número de nodos terminales

1 3 → Nodos terminales

5→ Coste final que habría que sumar a la función objetivo

3.1.1 DESCRIPCIÓN DEL CÓDIGO

En primer lugar, definimos las variables y los vectores que vamos a utilizar.

Además, definimos un contador llamado “i” que inicializaremos en 0 para su posterior uso en bucles for.

Abrimos el fichero en el cual se encuentran nuestros datos en modo lectura, facilitándole al programa su ruta.

Creamos otro fichero de texto con el nombre y ruta que le indiquemos. Estos dos últimos pasos se han realizado con la utilización de la función fopen y otorgándoles a cada archivo el permiso de lectura y escritura según correspondiera.

Es muy importante que comprobemos que se ha abierto bien el archivo de lectura y que se ha creado correctamente el fichero de escritura.

En el código están definidas cuatro funciones.

La primera es para que me devuelva tres vectores, uno para los nodos origen, otro para los nodos destino y otro para los costes. Esta función la hemos realizado recorriendo el archivo de lectura y almacenando los datos a través de la función fscanf. Cada vez que esta función lee una fila y almacena cada uno de los números en su correspondiente vector, incrementamos el valor del contador. De esta forma en el vector nodo origen el primer dato será el correspondiente al nodo origen de la primera arista, el segundo dato será el correspondiente al nodo origen de la segunda arista y así sucesivamente. Por otro lado, en el vector para los nodos destino, el primer dato será el nodo destino de la primera arista, el segundo dato será el nodo destino de la segunda arista. Y por último en el vector para los costes ocurre igual que con los dos vectores anteriores.

La segunda función es para reservar memoria para vectores dinámicos ya que no sabemos cuántas aristas vamos a tener por lo que no sabemos la longitud de los vectores anteriores. Haciendo esto no es necesario reservar memoria que después no vayamos a utilizar, ya que reservamos exactamente la que necesitamos, de esta forma conseguimos un programa más eficiente y más rápido.

La tercera y cuarta función es para leer y almacenar el número de nodos y el número de aristas a través de la función fscanf.

Una vez definidas todas las funciones las llamamos en nuestro programa principal.

Es importante comprobar también que la reserva de memoria para los vectores dinámicos se ha realizado correctamente.

A través de la función fprintf y la utilización de bucles for, con el contador “i” escribimos los vectores y los datos almacenados en el fichero destino con el formato que el programa LINGO es capaz de leer.

Por ejemplo, en la sección DATA, en Origen, en este caso lo que tenemos que ir imprimiendo uno a uno son los componentes de vector1, en Destino, los componentes de vector 2 y por último en c los componentes de vector3.

Para Raíz lo único que ha sido necesario añadirle al programa ha sido un bucle for con la condición de que para la primera vez que se recorriese el for me escribiese un 1 y todas las demás un 0. Y que se hiciese tantas veces como número de nodos tuviera mi problema.

Por último, es muy importante que cerremos los dos ficheros, tanto el de lectura como el de escritura, ya que si no dará fallos cuando queramos ejecutar el programa.

Una vez hecho este código, en el caso en el que queramos cambiar la formulación de una de las restricciones lo único que tenemos que hacer es cambiar la última parte del programa en la que indicamos que es lo que queremos que se nos escriba en el fichero de escritura a través de la función fprintf.

A continuación, se muestra cómo quedaría el archivo de texto creado.

```
MODEL:
SETS:
Nodo /1..4/: x, Raiz;
Aristas /1..10/: alfa, c, Origen, Destino;
ENDSETS
DATA:
  Origen = 1, 1, 1, 2, 3, 2, 3, 4, 3, 4;
  Destino = 2, 3, 4, 3, 4, 1, 1, 1, 2, 3;
  c = 4, 1, 2, 5, 3, 4, 1, 2, 5, 3;
  Raiz = 1, 0, 0, 0;
  N= 4;
  ENDDATA
 Min = @SUM(Aristas(j):c(j)*alfa(j));
 @FOR(Nodo(i)| Raiz(i)#EQ#0: @SUM(Aristas(j)| Destino(j)#EQ#i: alfa(j))=1);
 @FOR(Nodo(i)| Raiz(i)#EQ#1: @SUM(Aristas(j)| Origen(j)#EQ#i: alfa(j))>=1);
 @FOR(Aristas(j)|Raiz(Origen(j))#EQ#0 #AND# Raiz(Destino(j))#EQ#0:-x(Origen(j))+ x(Destino(j)) + ((N-1)*alfa(j)) <= (N-2));
 @FOR(Aristas(j)|Raiz(Origen(j))#EQ#0: x(Origen(j))<=N-1);
 @FOR(Aristas(j)|Raiz(Origen(j))#EQ#0: x(Origen(j))>=1);
 @FOR(Aristas(j):@BIN(alfa(j)));
END
```

Figura 11. Fuente: Elaboración propia.

3.1.2 AMPLIACIÓN DEL CÓDIGO

Para el problema de Steiner ha sido necesario realizar una pequeña ampliación del código. Ya que ha sido necesario añadir tres funciones más.

La primera de ellas es para almacenar el número de nodos terminales a través de la función fscanf.

La segunda función sirve para almacenar en un vector aquellos nodos que son terminales. De tal forma que, si un problema tiene unos nodos terminales que sean 3, 4 y 5, se creará un vector que tenga tres componentes, donde su primer componente será 3, su segundo componente 4 y su tercer componente 5.

Este vector es necesario ya que, en los nodos, además de las características anteriores, también tendremos la característica T, que será 1 cuando el nodo sea terminal y 0 cuando no lo sea.

La tercera función es para almacenar a través de fscanf el valor que habrá que sumarle posteriormente a la función objetivo.

Además de estas tres funciones también ha sido necesario establecer una condición para que cuando imprimiese los valores de "Raíz", imprimiese 1 no la primera vez que recorriese el for sino la primera vez que lo hiciese y coincidiese a la vez que el valor de "T" fuese también uno. Esto es así en este caso ya que no podemos poner como nodo "Raíz" un nodo que no es necesario que pertenezca a la solución.

3.2 Lingo

Lingo es un software de optimización matemática que nos sirve como herramienta para resolver y analizar problemas lineales y no lineales a través de su formulación.

Una de las grandes ventajas de LINGO es que permite expresar el problema de una forma muy similar a la anotación matemática normal. Aun así, es necesario que escribamos el problema en su propio lenguaje.

3.2.1 FORMATO BÁSICO DE LINGO

El formato básico se puede dividir en 5 partes:

1) Encabezado: Aquí lo único que tendríamos que poner sería MODEL:

2) Función Objetivo: Pondríamos Max o Min según si queremos maximizar o minimizar y se escribiría tal que: Max/Min= 2*x1+3*x2; Es importante tanto el igual después del Max o el Min como también el punto y coma al final.

3) Restricciones: Las escribiríamos tal que: 2*x1-3*x2>= 3; Es importante no olvidar aquí tampoco el punto y coma al final de cada una de las restricciones.

4) Restricciones de tipo variables: Por defecto LINGO considera todas las variables como continuas y mayores o iguales que cero. Si alguna de nuestras variables fuera distinta habría que especificarlo tal que:

@GIN(x1); Si queremos definirla como variable entera.

@FREE(x1); Si queremos definirla como variable libre.

@BIN(x1); Si queremos definirla como variable binaria

En este caso también es necesario poner el punto y coma al final de cada una de las restricciones de tipo variable.

5) Fin: El fin del modelo se indicaría simplemente con un END.

Lo anterior descrito se vería en el programa LINGO de la siguiente forma:

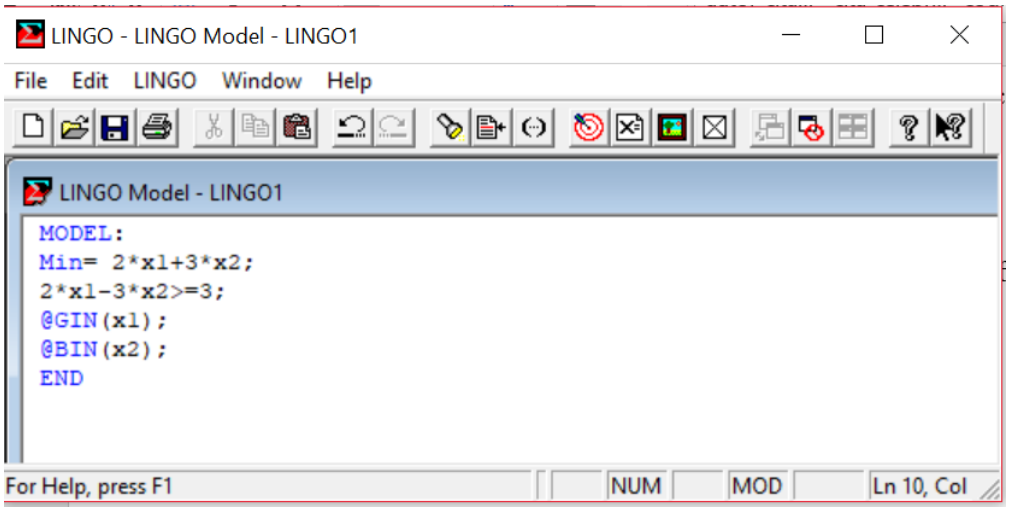

Figura 12. Fuente: Elaboración propia.

Si quisiéramos resolver este problema lo único que habría que hacer sería darle al botón de la diana roja y blanca que se puede ver en la imagen anterior.

Una vez le demos a ese botón nos aparecerá un cuadro tal que:

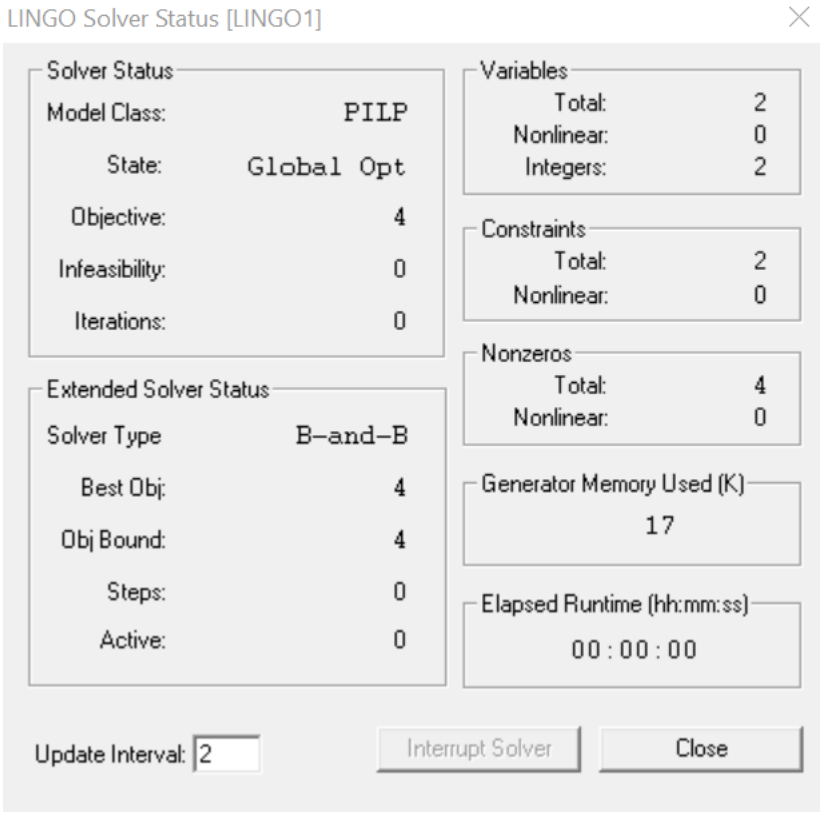

Figura 13. Fuente: Elaboración propia.

Esto es lo que debe aparecer en el caso de no haya habido ningún error.

Model Class nos indica el tipo de modelo: ILP quiere decir programación lineal entera.

State: Muestra la situación actual de la solución, que puede ser Feasible (factible, aunque aún no la haya encontrado), Infeasible (no factible), Global opt (Óptimo global) o Unbounded (No acotada).

Objective: Es el valor de la función objetivo.

Iterations: Número necesario de iteraciones para llegar a la solución.

Infeasibility: Cantidad por la cual no se verifican las restricciones.

Variables: Muestra el número total de variables del modelo.

Constraints: Es el número total de restricciones.

Nonzeros: Muestra el número de coeficientes no nulos.

Generator Memory Used (K): Cantidad de memoria usada.

Elapsed Runtime: Nos indica el tiempo empleado en la resolución del modelo.

En este caso la solución sería la siguiente:

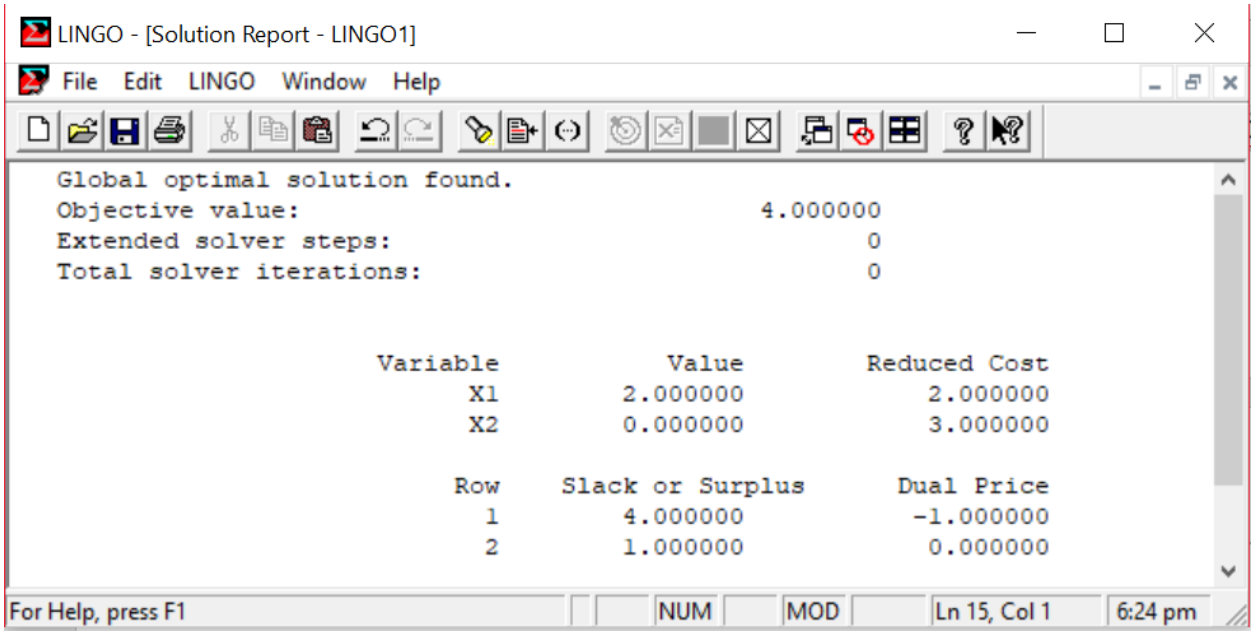

Figura 14. Fuente: Elaboración propia.

Donde podemos ver que el valor óptimo de nuestra función objetivo sería 4 y que esto ocurriría cuando la variable x1 toma el valor 2 y la variable x2 toma el valor 0.

3.2.2 USO DE CONJUNTOS

Si queremos expresar un modelo de forma simplificada, necesitamos el uso de conjuntos.

Los conjuntos tienen una equivalencia directa con los actores del problema.

Hay dos tipos de conjuntos:

-Conjuntos primitivos: Recogen los elementos conjunto del problema y sus atributos propios o compartidos con elementos que no estén en un conjunto.

-Conjuntos derivados: Recogen las características compartidas entre elementos conjunto. Están formados, por tanto, por los elementos primitivos ya definidos.

Las características sólo se definen una única vez.

Los elementos que no forman conjunto no es necesario definirlos. Sus características se definen directamente en la sección de valores (DATA).

Por otro lado, LINGO posee dos secciones para la definición de los conjuntos:

-Sección SET: Se definen los nombres de los diferentes elementos y el número de cada uno de ellos.

-Sección DATA: Se definen los valores de las diferentes características de los elementos.

Para que se entienda de forma más sencilla se ha puesto un ejemplo:

Suponiendo que una empresa dispone de 10 almacenes donde están las unidades de su producto.

Cada almacén posee Ai unidades de producto.

También tenemos un conjunto de 12 clientes, a los cuales la empresa tiene que abastecer desde los almacenes.

Cada cliente demanda Dj productos.

Por último, tenemos el coste de enviar un producto desde el almacén i al cliente j, definido tal que cij.

Teniendo estos datos nuestra tabla de actores sería la siguiente:

Actores	Tipo	Características
Almacenes	Conjunto i=1..10 Unitario	- Existencias (Ai ; Valor entero; Compartida) - Coste (cij; Continuo; Compartida)
Clientes	Conjunto j=1..12 Unitario	- Demanda (Dj; Valor entero; Compartida) - cij
Producto	Medible discreto	- Ai; Dj; cij

Figura 15. Fuente: Apuntes de la asignatura Métodos cuantitativos de gestión.

Y nuestra definición de conjuntos sería:

```
SETS:
      Almacenes /1..10/: Existencias;
      Clientes /1..12/: Demanda;
      Almacences_Clientes(Almacenes, Clientes) : Coste;
ENDSETS
DATA:
 Existencias = 40, 50, 30, 45, 100, 34, 15, 12, 18, 50;
 Demanda = 20, 25, 25, 30, 10, 10 ,16, 23, 25, 35, 30, 13;
 Coste = 1, 4, 3, 2, 5, 5, 7, 9, 1, 2 , 3, 3,  1, 4, 3, 2, 5, 5, 5, 5, 1, 2 , 3, 3,
            4, 4, 3, 2, 5, 5, 7, 9, 1, 2 , 3, 2, 3, 1, 4, 3, 2, 5, 4, 7, 9, 1, 2 , 3, 2,
            5, 4, 3, 5, 5, 5, 7, 7, 1, 2 , 3, 3, 1, 4, 3, 2, 3, 5, 7, 9, 1, 2 , 1, 1,
            7, 4, 3, 6, 5, 5, 7, 2, 1, 2 , 7, 1,  1, 4, 3, 2, 5, 2, 7, 6, 1, 2 , 2, 3,
            3, 4, 5, 7, 5, 5, 7, 3, 1, 2 , 5, 3,  1, 4, 3, 2, 5, 2, 7, 6, 1, 2 , 2, 4,
            9, 4, 3, 2, 5, 5, 7, 1, 1, 2 , 3, 3,  1, 4, 3, 2, 1, 5, 7, 7, 1, 2 , 3, 3;
ENDDATA
```

Figura 16. Fuente: Apuntes de la asignatura Métodos cuantitativos de gestión.

Los diferentes valores que pone en el conjunto DATA en este caso son inventados, es simplemente un ejemplo para que se vea como tendría que quedar la formulación en LINGO de este problema.

3.2.3 SINTAXIS

A la hora de realizar las especificaciones de los problemas en LINGO es muy importante el conocimiento de dos funciones:

- @FOR: Es usado para generar restricciones sobre los miembros del conjunto. Su significado es "para todo". De esta forma si por ejemplo tenemos que aplicar una restricción a todos los nodos del grafo sería algo tal que: @FOR(Nodo(i): "aquí escribiríamos la restricción"); Si en cambio la restricción fuera para todas las aristas sería: @FOR(Aristas(j): "aquí escribiríamos la restricción");

- @SUM: Calcula la suma de una expresión sobre los miembros de un conjunto, es decir que si por ejemplo nuestra función objetivo fuera minimizar la suma de todos los pesos de las aristas por alfa (siendo Alfa igual a 1 si la arista pertenece a la solución), sería: Min=@SUM (Aristas(j): c(j)*alfa(j));

Por otro lado, tenemos los calificadores condicionales cuyo uso es opcional. Sirven para limitar los elementos del conjunto sobre los que se aplican las diferentes restricciones. En la siguiente imagen se pueden observar todos los calificadores que hay:

#EQ#	=	*#And#*	y
#NE#	≠	*#or#*	o
#GE#	≥		
#GT#	>		
#LT#	<		

Figura 17. Fuente: Apuntes de la asignatura métodos cuantitativos de gestión

Si quisiéramos poner una condición que se cumpliese por ejemplo solamente para los nodos que la variable Raíz fuera igual a 0 se pondría:

@FOR(Nodo(i)|Raíz #EQ#0: "Aquí pondríamos la restricción");

3.3 Formulación de los problemas

A continuación, se va a mostrar cómo quedaría el problema steinb1 (el cuál es el primero de nuestra batería de problemas) en formato LINGO según las cuatro formulaciones explicadas anteriormente.

```
13 19
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
8
4 5 6 7 8 10 11 13
 41
```

Figura 18. Fuente: OR-Library.

Como vamos a poder observar en las distintas formulaciones de este primer problema, el número de aristas que va a aparecer va a ser justo el doble y esto es porque las aristas son de doble sentido, es decir, aquí, por ejemplo, la primera arista va del nodo "1" al nodo "5" pero además también va del nodo "5" al nodo "1" y para que el programa lo entendiese era necesario ponerlo como dos aristas distintas aunque tendrán el mismo coste y no se podrán seleccionar las dos a la vez.

FORMULACIÓN DEL PROBLEMA MST CON LA ESTRATEGIA MTZ

```
MODEL:
SETS:
Nodo /1..13/: x, Raiz;
Aristas /1..38/: alfa, c, Origen, Destino;
ENDSETS
DATA:
  Origen = 1, 3, 3, 4, 6, 6, 6, 8, 9, 9, 10, 11, 11, 12, 12, 12, 12, 12, 13, 5, 9, 1, 8, 4, 5, 3, 10, 2, 7, 2, 1, 2, 6, 10, 11, 13, 7, 9;
  Destino = 5, 9, 1, 8, 4, 5, 3, 10, 2, 7, 2, 1, 2, 6, 10, 11, 13, 7, 9, 1, 3, 3, 4, 6, 6, 6, 8, 9, 9, 10, 11, 11, 12, 12, 12, 12, 12, 13;
  c = 7, 1, 7, 2, 2, 2, 8, 4, 8, 5, 6, 8, 14, 8, 9, 2, 7, 15, 11, 7, 1, 7, 2, 2, 2, 8, 4, 8, 5, 6, 8, 14, 8, 9, 2, 7, 15, 11;
  Raiz = 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0;
  N= 13;
  ENDDATA
 Min = @SUM(Aristas(j):c(j)*alfa(j));
 @FOR(Nodo(i)| Raiz(i)#EQ#0: @SUM(Aristas(j)| Destino(j)#EQ#i: alfa(j))=1);
@FOR(Nodo(i)| Raiz(i)#EQ#1: @SUM(Aristas(j)| Origen(j)#EQ#i: alfa(j))>=1);
 @FOR(Aristas(j)|Raiz(Origen(j))#EQ#0 #AND# Raiz(Destino(j))#EQ#0:-x(Origen(j))+ x(Destino(j)) + ((N-1)*alfa(j)) <= (N-2));
 @FOR(Aristas(j)|Raiz(Origen(j))#EQ#0: x(Origen(j))<=N-1);
 @FOR(Aristas(j)|Raiz(Origen(j))#EQ#0: x(Origen(j))>=1);
 @FOR(Aristas(j):@BIN(alfa(j)));
END
```

FORMULACIÓN DEL PROBLEMA MST CON LA ESTRATEGIA DL

```
MODEL:
SETS:
Nodo /1..13/: x, Raiz;
Aristas /1..38/: alfa, c, Origen, Destino;
ENDSETS
DATA:
  Origen = 1, 3, 3, 4, 6, 6, 6, 8, 9, 9, 10, 11, 11, 12, 12, 12, 12, 12, 13, 5, 9, 1, 8, 4, 5, 3, 10, 2, 7, 2, 1, 2, 6, 10, 11, 13, 7, 9;
  Destino = 5, 9, 1, 8, 4, 5, 3, 10, 2, 7, 2, 1, 2, 6, 10, 11, 13, 7, 9, 1, 3, 3, 4, 6, 6, 6, 8, 9, 9, 10, 11, 11, 12, 12, 12, 12, 12, 13;
  c = 7, 1, 7, 2, 2, 2, 8, 4, 8, 5, 6, 8, 14, 8, 9, 2, 7, 15, 11, 7, 1, 7, 2, 2, 2, 8, 4, 8, 5, 6, 8, 14, 8, 9, 2, 7, 15, 11;
  Raiz = 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0;
  N= 13;
  ENDDATA
 Min = @SUM(Aristas(j):c(j)*alfa(j));
 @FOR(Nodo(i)| Raiz(i)#EQ#0: @SUM(Aristas(j)| Destino(j)#EQ#i: alfa(j))=1);
 @FOR(Nodo(i)| Raiz(i)#EQ#1: @SUM(Aristas(j)| Origen(j)#EQ#i: alfa(j))>=1);
 @FOR(Aristas(j)| j#LE#19 #AND# Raiz(Origen(j))#EQ#0 #AND# Raiz(Destino(j))#EQ#0 :-x(Origen(j))+x(Destino(j))+ (N-1)*alfa(j)+ (N-3)*alfa(j+19) <= N-2);
 @FOR(Aristas(j)| j#GE#20 #AND# Raiz(Origen(j))#EQ#0 #AND# Raiz(Destino(j))#EQ#0 :-x(Origen(j))+x(Destino(j))+ (N-1)*alfa(j)+ (N-3)*alfa(j-19) <= N-2);
 @FOR(Aristas(j)|Raiz(Destino(j))#EQ#0: x(Destino(j))<=N-1);
 @FOR(Aristas(j)|Raiz(Origen(j))#EQ#0: x(Origen(j))<=N-1);
 @FOR(Aristas(j)|Raiz(Destino(j))#EQ#0: x(Destino(j))>=1);
 @FOR(Aristas(j)|Raiz(Origen(j))#EQ#0: x(Origen(j))>=1);
 @FOR(Aristas(j):@BIN(alfa(j)));
END
```

FORMULACIÓN DEL PROBLEMA STEINER CON LA ESTRATEGIA MTZ

```
MODEL:
SETS:
Nodo /1..13/: x, Raiz,T,Beta;
Aristas /1..38/: alfa, c, Origen, Destino;
ENDSETS
DATA:
  Origen = 1, 3, 3, 4, 6, 6, 6, 8, 9, 9, 10, 11, 11, 12, 12, 12, 12, 12, 13, 5, 9, 1, 8, 4, 5, 3, 10, 2, 7, 2, 1, 2, 6, 10, 11, 13, 7, 9;
  Destino = 5, 9, 1, 8, 4, 5, 3, 10, 2, 7, 2, 1, 2, 6, 10, 11, 13, 7, 9, 1, 3, 3, 4, 6, 6, 6, 8, 9, 9, 10, 11, 11, 12, 12, 12, 12, 12, 13;
  c = 7, 1, 7, 2, 2, 2, 8, 4, 8, 5, 6, 8, 14, 8, 9, 2, 7, 15, 11, 7, 1, 7, 2, 2, 2, 8, 4, 8, 5, 6, 8, 14, 8, 9, 2, 7, 15, 11;
  T = 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1;
  Raiz = 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0;
  N= 13;
  ENDDATA
 Min = @SUM(Aristas(j):c(j)*alfa(j))+41;
 @FOR(Nodo(i)| Raiz(i)#EQ#0 #AND# T(i)#EQ#1: @SUM(Aristas(j)| Destino(j)#EQ#i: alfa(j))=1);
 @FOR(Nodo(i)| Raiz(i)#EQ#0 #AND# T(i)#EQ#0: @SUM(Aristas(j)| Destino(j)#EQ#i: alfa(j))=Beta(i));
 @FOR(Nodo(i)| Raiz(i)#EQ#0 #AND# T(i)#EQ#0: @FOR(Aristas(j)| Origen(j)#EQ#i: alfa(j)<=Beta(i)));
@FOR(Nodo(i)| Raiz(i)#EQ#1: @SUM(Aristas(j)| Origen(j)#EQ#i: alfa(j))>=1);
 @FOR(Aristas(j)|Raiz(Origen(j))#EQ#0 #AND# Raiz(Destino(j))#EQ#0:-x(Origen(j))+ x(Destino(j)) + ((N-1)*alfa(j)) <= (N-2));
 @FOR(Aristas(j)|Raiz(Origen(j))#EQ#0: x(Origen(j))<=N-1);
 @FOR(Aristas(j)|Raiz(Origen(j))#EQ#0: x(Origen(j))>=1);
 @FOR(Aristas(j):@BIN(alfa(j)));
 @FOR(Nodo(i):@BIN(Beta(i)));
```

FORMULACIÓN DEL PROBLEMA STEINER CON LA ESTRATEGIA DL

```
MODEL:
SETS:
Nodo /1..13/: x, Raiz, T, Beta;
Aristas /1..38/: alfa, c, Origen, Destino;
ENDSETS
DATA:
  Origen = 1, 3, 3, 4, 6, 6, 6, 8, 9, 9, 10, 11, 11, 12, 12, 12, 12, 13, 5, 9, 1, 8, 4, 5, 3, 10, 2, 7, 2, 1, 2, 6, 10, 11, 13, 7, 9;
  Destino = 5, 9, 1, 8, 4, 5, 3, 10, 2, 7, 2, 1, 2, 6, 10, 11, 13, 7, 9, 1, 3, 3, 4, 6, 6, 6, 8, 9, 9, 10, 11, 11, 12, 12, 12, 12, 12, 13;
  c = 7, 1, 7, 2, 2, 2, 8, 4, 8, 5, 6, 8, 14, 8, 9, 2, 7, 15, 11, 7, 1, 7, 2, 2, 2, 8, 4, 8, 5, 6, 8, 14, 8, 9, 2, 7, 15, 11;
  T = 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1;
  Raiz = 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0;
  N= 13;
  ENDDATA
 Min = @SUM(Aristas(j):c(j)*alfa(j)) + 41 ;
 @FOR(Nodo(i)| Raiz(i)#EQ#0 #AND# T(i)#EQ#1: @SUM(Aristas(j)| Destino(j)#EQ#i: alfa(j))=1);
 @FOR(Nodo(i)| Raiz(i)#EQ#0 #AND# T(i)#EQ#0: @SUM(Aristas(j)| Destino(j)#EQ#i: alfa(j))=Beta(i));
 @FOR(Nodo(i)| Raiz(i)#EQ#0 #AND# T(i)#EQ#0: @FOR(Aristas(j)| Origen(j)#EQ#i: alfa(j)<=Beta(i)));
@FOR(Nodo(i)| Raiz(i)#EQ#1: @SUM(Aristas(j)| Origen(j)#EQ#i: alfa(j))>=1);
 @FOR(Aristas(j)| j#LE#19 #AND# Raiz(Origen(j))#EQ#0 #AND# Raiz(Destino(j))#EQ#0 :-x(Origen(j))+x(Destino(j))+ (N-1)*alfa(j)+ (N-3)*alfa(j+19) <= N-2);
 @FOR(Aristas(j)| j#GE#20 #AND# Raiz(Origen(j))#EQ#0 #AND# Raiz(Destino(j))#EQ#0 :-x(Origen(j))+x(Destino(j))+ (N-1)*alfa(j)+ (N-3)*alfa(j-19) <= N-2);
 @FOR(Aristas(j)|Raiz(Destino(j))#EQ#0: x(Destino(j))<=N-1);
 @FOR(Aristas(j)|Raiz(Origen(j))#EQ#0: x(Origen(j))<=N-1);
 @FOR(Aristas(j)|Raiz(Destino(j))#EQ#0: x(Destino(j))>=1);
 @FOR(Aristas(j)|Raiz(Origen(j))#EQ#0: x(Origen(j))>=1);
 @FOR(Aristas(j):@BIN(alfa(j)));
 @FOR(Nodo(i):@BIN(Beta(i)));
END
```

4 EXPERIMENTACIÓN

A través de la experimentación lo que pretendemos hacer es una comparación.

Cada experimentación se basa en la resolución de los diferentes problemas según las diferentes formulaciones.

Los problemas que vamos a resolver están clasificados según su complejidad en diferentes clases, tipo “b”, tipo “c” y tipo “d” siendo estos últimos los más complejos, hecho que repercute directamente en el tiempo de resolución que por consiguiente será mayor para estos problemas que para los otros.

En la tabla que hay a continuación podemos observar que problemas son los que he analizado, pudiendo saber su número de nodos, su número de aristas y el número de nodos terminales.

Problema	Número Nodos	Número Arcos	Número Nodos Termi.
steinb1	13	19	8
steinb2	15	21	11
steinb3	20	25	15
steinb4	40	80	9
steinb5	39	80	12
steinb6	45	87	25
steinb7	22	33	11
steinb8	26	38	15
steinb9	27	35	23
steinb10	55	121	13
steinb11	63	129	19
steinb12	63	125	36
steinb13	36	56	14
steinb14	42	65	21
steinb15	48	69	38
steinb16	77	166	17
steinb17	74	153	23
steinb18	82	166	45
steinc1	143	260	5
steinc2	128	234	10
steinc3	178	295	75
steinc4	193	314	102
steinc5	223	341	180
steinc6	366	837	5
steinc7	383	866	10
steinc8	387	867	79
steinc9	418	903	124
steinc10	427	891	242
steinc11	499	2005	5
steinc12	499	2065	10
steinc13	498	2026	83

steinc14	499	1968	125
steinc15	500	1815	250
steinc16	500	3417	5
steinc17	500	3463	10
steinc18	500	3496	83
steinc19	500	3352	125
steinc20	500	12500	250
steind1	272	504	5
steind2	283	519	10
steind3	350	585	148
steind4	359	590	207
steind5	470	708	377
steind6	759	1730	5
steind7	749	1721	10
steind8	802	1778	166
steind9	802	1768	246
steind10	836	1781	485
steind11	993	4442	5
steind12	1000	4437	10
steind13	998	4354	167
steind14	998	4309	250
steind15	996	3921	498
steind16	1000	8048	5
steind17	1000	8061	10
steind18	1000	7755	167
steind19	1000	7552	250
steind20	1000	6887	500

Figura 19. Fuente: Elaboración propia

Para poder observar a priori la diferencia entre los diferentes problemas, se han realizado seis gráficos.

El primero es un gráfico en el que se puede observar la evolución del número de nodos de los problemas tipo “b” según vamos avanzando en el número del problema.

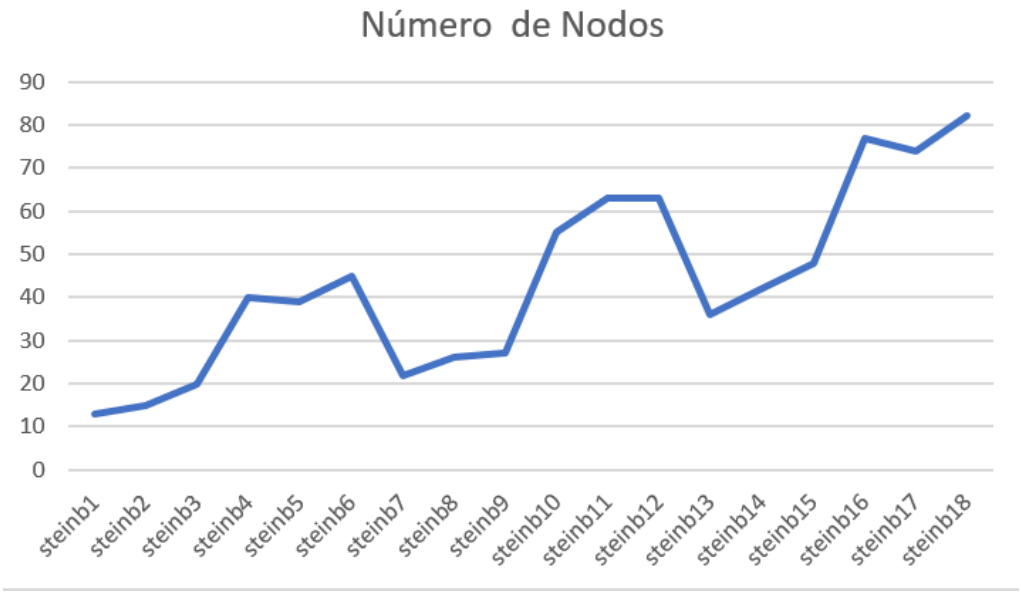

Figura 20. Fuente: Elaboración propia

Aquí podemos observar que el número de nodos de los problemas crece hasta llegar al problema número 6, una vez llegado ahí decrece en el problema número 7 para volver a empezar a crecer hasta el problema número 13 donde vuelve a decrecer para seguir creciendo hasta el problema número 17 que decrece un poco para volver a crecer posteriormente en el número 18.

Partiendo de estos datos, podemos hacernos una idea de que probablemente el último problema tendrá un tiempo de resolución mayor que el primero de todos, ya que, al aumentar el número de nodos, aumenta la complejidad del problema.

En el siguiente gráfico podemos ver la evolución del número de nodos de los problemas tipo “c”

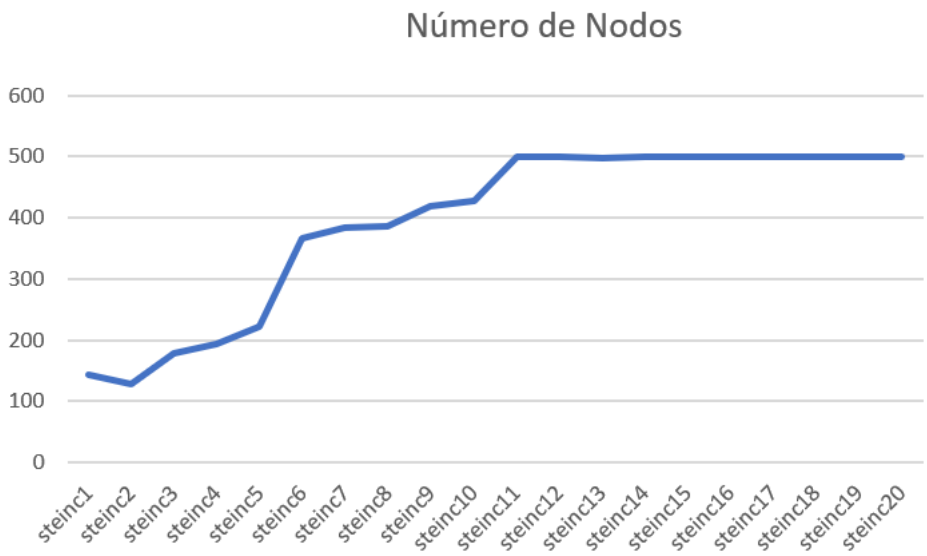

Figura 21. Fuente: Elaboración propia

Aquí podemos observar que el número de nodos de los problemas crece hasta llegar al problema número 11, una vez llegado ahí se mantiene en un número constante de 500 nodos en cada problema.

En el siguiente gráfico podemos ver la evolución del número de nodos para los problemas de tipo “d”.

Figura 22. Fuente: Elaboración propia

Aquí podemos observar que la evolución del número de nodos de los problemas de tipo “d” es muy parecida a la evolución seguida por los problemas de tipo “c”. Ya que el número de nodos crece hasta llegar al problema número 11 a partir del cual el número se mantiene en 1000 nodos.

En el siguiente gráfico podemos ver la evolución del número de arcos de los problemas tipo “b”.

Figura 23. Fuente: Elaboración propia

Podemos apreciar que en el caso de los problemas de tipo “b” la evolución que llevan a cabo el número de arcos es idéntica a la evolución del número de nodos. Por lo que suponemos que aquellos que tienen mayor número de nodos y de arcos serán aquellos en los que se tarde más en llegar a su solución.

En el siguiente gráfico podemos observar la evolución de los arcos de los problemas tipo “c”.

Figura 24. Fuente: Elaboración propia

En este caso el número de arcos va creciendo progresivamente hasta llegar al último problema donde da un gran salto, ya que pasa de tener cerca de 4000 arcos a tener 12500. Debido a que el número de nodos de los últimos problemas también era mayor que el número de nodos de los primeros problemas, suponemos que los tiempos de resolución de los últimos serán mayores que los tiempos de resolución de los primeros.

En el siguiente gráfico podemos observar la evolución del número de arcos de los problemas tipo “d”

Figura 25. Fuente: Elaboración propia

Como podemos observar en el gráfico el número de arcos va creciendo progresivamente, aunque da algunos saltos, por ejemplo, del problema número 5 al 6 prácticamente dobla el número de arcos, después al pasar del problema 10 al 11 pasa de tener algo menos de 2000 arcos a tener más de 4000. Y al pasar del problema 15 al 16 también da un gran salto. A partir del problema numero 17 parece que decrece un poco el número de arcos de los problemas.

En el siguiente gráfico podemos observar la evolución del número de nodos terminales de los problemas de tipo “b”.

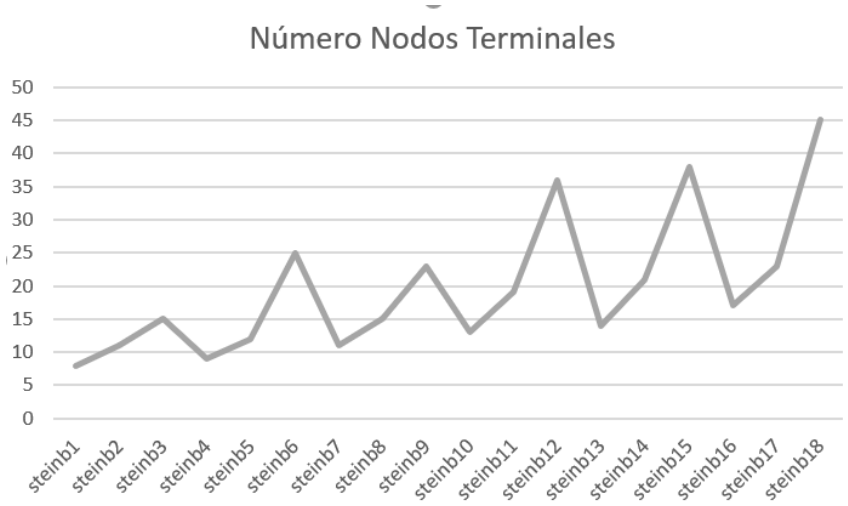

Figura 26. Fuente: Elaboración propia

Podemos observar que el número de nodos terminales va dando picos de subida y bajada constantemente. Y que los problemas 6, 12, 15 y 18 son aquellos que tienen un mayor número de nodos terminales. A partir de estos datos y teniendo en cuenta el número de nodos y el número de aristas de cada uno de los problemas, podríamos decir finalmente que dentro de los problemas del grupo “b” los más complejos dado su número de nodos, arcos y nodos terminales serán los problemas 6, 12 y 18. En este caso del problema número 15 no podríamos aún sacar conclusiones debido a que aunque es uno de los que tiene mayor número de nodos terminales no posee un gran número de nodos ni de arcos en comparación con los demás.

En el siguiente gráfico podemos observar la evolución del número de nodos terminales de los problemas de tipo “c”.

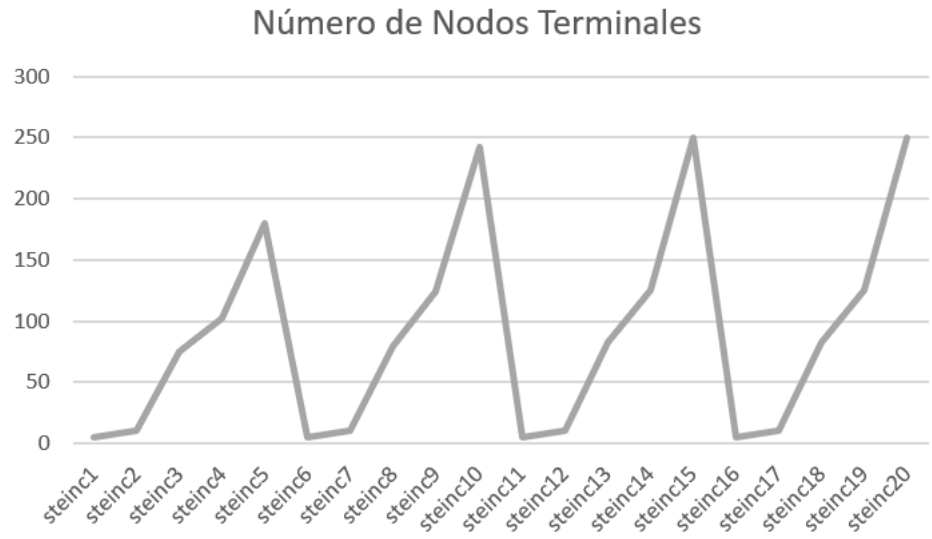

Figura 27. Fuente: Elaboración propia

Podemos apreciar que el número de nodos terminales va dando picos al igual que en el caso de los problemas de tipo “b”. Aunque en este caso los picos son más pronunciados. Fijándonos exclusivamente en este gráfico podríamos decir que los problemas más complejos serían el número 5, 10, 15 y 20, pero teniendo en cuenta también el número de nodos y el número de arcos sería complicado sacar conclusiones ya que aquellos con mayor número de nodos terminales no coinciden con aquellos con mayor número de nodos y de arcos.

En el siguiente gráfico podemos observar la evolución del número de nodos terminales de los problemas de tipo “d”.

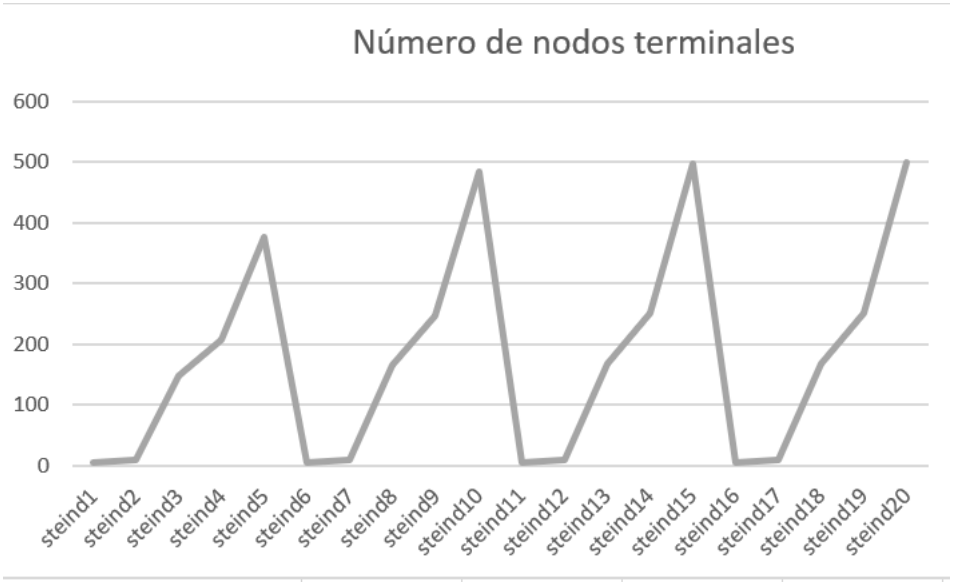

Figura 28. Fuente: Elaboración propia

Podemos observar que el número de nodos terminales va dando picos al igual que en el caso de los problemas de tipo “b” y “c”. Además, si nos fijamos, nos damos cuenta de que siguen una evolución prácticamente idéntica a los problemas de tipo “c” con la diferencia de que aquí el número de nodos es prácticamente el doble.

Lo único que podríamos decir entonces sería que el problema 20 sí que necesitará un mayor tiempo de resolución ya que no hay ningún otro problema que tenga ni más nodos que él, ni más aristas, ni más nodos terminales.

4.1 Resultados computacionales

Como ya se explicó anteriormente, debido al gran tamaño de alguno de los problemas, cuando pasada una hora el programa LINGO no había sido capaz de llegar al óptimo lo hemos interrumpido y hemos anotado el resultado obtenido.

Debido a que no contábamos con los óptimos de las dos primeras formulaciones, para estas solamente hemos analizado los problemas de tipo "b" y "c" ya que los de tipo "d" tardaban casi todos más de una hora en resolverse.

Es necesario aclarar que los óptimos según la tercera y cuarta formulación de los problemas a los que no se ha llegado la solución en una hora, son datos obtenidos en experimentos anteriores, es decir que no han sido calculados en este trabajo.

En las tablas siguientes se presentan para cada grupo de problemas, el número de iteraciones realizadas, el tiempo de resolución, el número de variables, la solución obtenida, el óptimo, el resultado continuo, y la diferencia entre el resultado continuo y el óptimo. (Para saber el resultado continuo de los problemas lo único que ha habido que hacer es resolver de nuevo todos los problemas, pero esta vez quitando la condición de que tanto alfa como beta tenían que ser binarios)

Además de lo anterior también hemos definido el parámetro "Error" como un porcentaje, de tal forma que, aunque no hayamos llegado al óptimo podemos saber cuánto de alejada esta nuestra solución.

$$ERROR = \frac{Solución\ obtenida - Óptimo}{Óptimo}$$

Por otro lado, la finalidad de calcular los resultados del problema continuo es para saber si un problema se comporta bien ante una determinada estrategia de resolución.

Esto es debido a que la solución del problema continuo es el primer valor con el que empezamos a probar como solución en el problema entero.

Por lo que teniendo en cuenta esto, si tenemos dos problemas, para los dos el óptimo es 28, y el valor de la solución del primer problema en continuo es 26'5, mientras que la solución en continuo del segundo problema es 10'32, ya que este último va a empezar a probar desde un número mucho más alejado del óptimo que el primer problema, en consecuencia, deberá un tiempo de resolución mayor siempre y cuando utilicemos la misma estrategia de resolución para los dos.

Para calcular la solución de los problemas en continuo, lo único que ha habido que hacer ha sido quitar la condición de que tanto alfa como beta tenían que ser variables binarias.

Y esas restricciones las hemos sustituido por:

$$\alpha \leq 1$$

$$\beta \leq 1$$

Resultados obtenidos para el problema MST con la formulación MTZ

Problema	Iteraciones	Tiempo	Vbles	Sol. Obt.	Óptimo	Error	Result. Cont.	Difer.
steinb1	139	0:00:00	51	53	53	0,00%	41,333	11,667
steinb2	230	0:00:00	57	77	77	0,00%	60,429	16,571
steinb3	58	0:00:00	70	66	66	0,00%	63,000	3,000
steinb4	19258	0:00:02	200	131	131	0,00%	109,667	21,333
steinb5	514	0:00:00	199	116	116	0,00%	93,947	22,053
steinb6	3827	0:00:01	219	145	145	0,00%	118,682	26,318
steinb7	403	0:00:00	88	108	108	0,00%	85,048	22,952
steinb8	11731	0:00:01	102	110	110	0,00%	88,040	21,960
steinb9	227	0:00:00	97	79	79	0,00%	65,769	13,231
steinb10	2174	0:00:01	297	184	184	0,00%	156,481	27,519
steinb11	466650	0:01:33	321	188	188	0,00%	154,581	33,419
steinb12	26592	0:00:03	313	194	194	0,00%	156,839	37,161
steinb13	2663	0:00:00	148	178	178	0,00%	137,543	40,457
steinb14	381	0:00:00	172	178	178	0,00%	140,268	37,732
steinb15	1692	0:00:00	186	232	232	0,00%	194,149	37,851
steinb16	3347	0:00:00	409	248	248	0,00%	209,447	38,553
steinb17	1422	0:00:00	380	205	205	0,00%	177,548	27,452
steinb18	2E+06	0:03:14	414	252	252	0,00%	214,321	37,679
steinc1	15024	0:00:10	663	666	666	0,00%	541,127	124,873
steinc2	120476	0:00:18	596	683	683	0,00%	578,787	104,213
steinc3	214848	0:00:26	768	827	827	0,00%	625,492	201,508
steinc4	225501	0:00:33	821	930	930	0,00%	759,083	170,917
steinc5	2*10^7	0:06:24	905	1044	1044	0,00%	859,865	184,135
steinc6	115014	0:00:23	2040	1156	1156	0,00%	973,597	182,403
steinc7	2539	0:00:07	2115	1247	1247	0,00%	1071,460	175,540
steinc8	10^7	0:54:50	2121	1182	1182	0,00%	983,648	198,352
steinc9	33805	0:00:12	2224	1293	1293	0,00%	1057,670	235,330
steinc10	22494	0:00:06	2209	1363	1363	0,00%	1139,620	223,380
steinc11	272879	0:02:43	4509	862	862	0,00%	784,161	77,839
steinc12	113008	0:00:26	4629	888	888	0,00%	800,209	87,791
steinc13	10^7	1:00:00	4550	872		*	792,201	*
steinc14	875665	0:04:23	4435	855	855	0,00%	785,137	69,863
steinc15	337870	0:01:35	4130	882	882	0,00%	800,176	81,824
steinc16	10^6	0:16:57	7534	503	503	0,00%	499,000	4,000
steinc17	177689	0:04:06	7426	499	499	0,00%	499,000	0,000
steinc18	37807	0:00:21	7492	503	503	0,00%	503,000	0,000
steinc19	422533	0:12:57	7204	504	504	0,00%	504,000	0,000
steinc20	569030	0:34:17	25500	509	509	0,00%	508,004	0,996

Figura 29. Fuente: Elaboración propia

Resultados obtenidos para el problema MST con la formulación DL

Problema	Iteraciones	Tiempo	Vbles	Sol. Obt.	Ópt.	Error	Res. Cont.	Difer.
steinb1	32	0:00:00	51	53	53	0,00%	53	0
steinb2	45	0:00:00	57	77	77	0,00%	77	0
steinb3	81	0:00:00	70	66	66	0,00%	66	0
steinb4	4915	0:00:01	200	131	131	0,00%	128	3
steinb5	433	0:00:00	199	116	116	0,00%	115	1
steinb6	2421	0:00:00	219	145	145	0,00%	144	1
steinb7	171	0:00:00	88	108	108	0,00%	106	2
steinb8	9269	0:00:01	102	110	110	0,00%	105	5
steinb9	74	0:00:00	97	79	79	0,00%	79	0
steinb10	398	0:00:01	297	184	184	0,00%	184	0
steinb11	10^6	0:01:44	321	188	188	0,00%	184	4
steinb12	641	0:00:01	313	194	194	0,00%	194	0
steinb13	1660	0:00:00	148	178	178	0,00%	177	1
steinb14	136	0:00:00	172	178	178	0,00%	178	0
steinb15	272	0:00:00	186	232	232	0,00%	231	1
steinb16	563	0:00:01	409	248	248	0,00%	248	0
steinb17	711	0:00:00	380	205	205	0,00%	205	0
steinb18	784829	0:01:10	414	252	252	0,00%	249	3
steinc1	8408	0:00:02	663	666	666	0,00%	682	16
steinc2	154766	0:00:16	596	683	683	0,00%	682	1
steinc3	17048	0:00:04	768	827	827	0,00%	821	6
steinc4	116728	0:00:59	821	930	930	0,00%	928	2
steinc5	4*10^7	1:00:00	905	1044	1044	0,00%	1035	9
steinc6	7329	0:00:03	2040	1156	1156	0,00%	1153	3
steinc7	3153	0:00:02	2115	1247	1247	0,00%	1247	0
steinc8	7*10^6	0:40:57	2121	1182	1182	0,00%	1180	2
steinc9	3521	0:00:01	2224	1293	1293	0,00%	1293	0
steinc10	4297	0:00:02	2209	1363	1363	0,00%	1363	0
steinc11	7503	0:00:04	4509	862	862	0,00%	862	0
steinc12	73800	0:00:14	4629	888	888	0,00%	887	1
steinc13	2*10^7	1:00:00	45550	872		*	870	*
steinc14	3*10^6	0:12:01	4435	855	855	0,00%	853	2
steinc15	180540	0:00:46	4130	882	882	0,00%	881	1
steinc16	144234	0:03:41	7534	503	503	0,00%	503	0
steinc17	158993	0:10:44	7426	499	499	0,00%	499	0
steinc18	3*10^6	1:00:00	7492	511	503	1,59%	503	0
steinc19	642698	0:12:23	7204	504	504	0,00%	504	0
steinc20	254546	0:07:09	25500	509	509	0,00%	509	0

Figura 30. Fuente: Elaboración propia

Resultados obtenidos para el problema Steiner con la formulación MTZ

Problema	Iteraciones	Tiempo	Vbles	Sol. Obt.	Ópt.	Error	Res. Cont.	Difer.
steinb1	85	0:00:00	64	82	82	0,00%	70,083	11,917
steinb2	142	0:00:00	72	83	83	0,00%	76,714	6,286
steinb3	79	0:00:00	90	138	138	0,00%	133,105	4,895
steinb4	2079	0:00:01	240	59	59	0,00%	41,410	17,590
steinb5	682	0:00:00	238	61	61	0,00%	43,395	17,605
steinb6	12196	0:00:02	264	122	122	0,00%	93,409	28,591
steinb7	212	0:00:00	110	111	111	0,00%	99,286	11,714
steinb8	10008	0:00:01	128	104	104	0,00%	81,640	22,360
steinb9	585	0:00:00	124	220	220	0,00%	203,692	16,308
steinb10	2087	0:00:00	352	86	86	0,00%	64,148	21,852
steinb11	4*10^5	0:00:37	384	88	88	0,00%	67,290	20,710
steinb12	22685	0:00:03	376	174	174	0,00%	137,516	36,484
steinb13	1306	0:00:01	184	165	165	0,00%	127,786	37,214
steinb14	5322	0:00:01	214	235	235	0,00%	186,951	48,049
steinb15	1205	0:00:00	234	318	318	0,00%	283,021	34,979
steinb16	9471	0:00:02	486	127	127	0,00%	95,263	31,737
steinb17	10^5	0:00:15	454	131	131	0,00%	111,123	19,877
steinb18	6413	0:00:02	496	218	218	0,00%	179,833	38,167
steinc1	2*10^5	0:00:24	806	85	85	0,00%	48,049	36,951
steinc2	7*10^5	0:01:42	724	144	144	0,00%	69,189	74,811
steinc3	7*10^5	0:01:36	946	754	754	0,00%	559,006	194,994
steinc4	2*10^6	0:05:43	1014	1079	1079	0,00%	902,859	176,141
steinc5	6*10^6	0:14:19	1128	1579	1579	0,00%	1379,840	199,160
steinc6	7*10^5	0:03:10	2406	55	55	0,00%	29,003	25,997
steinc7	6*10^5	0:02:45	2498	102	102	0,00%	58,042	43,958
steinc8	2*10^7	1:00:00	2508	519	509	1,96%	338,335	170,665
steinc9	5*10^6	1:00:00	2642	714	707	0,99%	519,305	187,695
steinc10	2*10^6	0:09:10	2636	1093	1093	0,00%	890,425	202,575
steinc11	10^6	0:07:34	5008	32	32	0,00%	16,008	15,992
steinc12	5*10^6	0:45:30	5128	46	46	0,00%	26,013	19,987
steinc13	9*10^6	1:00:00	5048	266	258	3,10%	190,095	67,905
steinc14	2*10^6	0:15:23	4934	323	323	0,00%	266,084	56,916
steinc15	9*10^6	1:00:00	4630	556	556	0,00%	479,130	76,870
steinc16	3*10^5	0:00:42	8034	11	11	0,00%	8,000	3,000
steinc17	9*10^5	0:03:28	7926	18	18	0,00%	15,000	3,000
steinc18	2*10^6	1:00:00	7992	113	113	0,00%	107,504	5,496
steinc19	3*10^5	0:02:10	7704	146	146	0,00%	145,004	0,996
steinc20	38599	0:53:04	26000	267	267	0,00%	266,000	1,000
steind1	3*10^5	0:00:50	1552	106	106	0,00%	55,097	50,903
steind2	7*10^5	0:02:04	1604	220	220	0,00%	122,145	97,855
steind3	3*10^5	0:00:40	1870	1565	1565	0,00%	1267,750	297,250
steind4	2*10^5	0:00:24	1898	1935	1935	0,00%	1661,710	273,290
steind5	27343	0:00:42	2356	3250	3250	0,00%	2874,830	375,170
steind6	5*10^6	1:00:00	4978	77	67	14,93%	15,024	51,976

steind7	5*10^6	0:52:01	4940	103	103	0,00%	59,019	43,981
steind8	7*10^6	1:00:00	5160	1098	1072	2,43%	765,197	306,803
steind9	8*10^6	1:00:00	5140	1497	1448	3,38%	1093,310	354,690
steind10	8*10^6	1:00:00	5234	2110	2110	0,00%	1709,420	400,580
steind11	10^7	1:00:00	8870	42	29	44,83%	13,002	15,998
steind12	2*10^6	1:00:00	10674	45	42	7,14%	21,009	20,991
steind13	5*10^6	1:00:00	10704	540	500	8,00%	388,069	111,931
steind14	6*10^6	1:00:00	10614	709	667	6,30%	533,094	133,906
steind15	8*10^6	1:00:00	9834	1147	1116	2,78%	974,113	141,887
steind16	3*10^6	1:00:00	18096	14	13	7,69%	9,000	4,000
steind17	4*10^5	1:00:00	18122	24	23	4,35%	17,002	5,998
steind18	7*10^5	1:00:00	17510	256	223	14,80%	216,006	6,994
steind19	4*10^6	1:00:00	17104	330	310	6,45%	302,004	7,996
steind20	4*10^5	1:00:00	1577	544	539	0,93%	535,008	3,992

Figura 31. Fuente: Elaboración propia

Resultados obtenidos para el problema Steiner con la formulación DL

Problema	Iteraciones	Tiempo	Vbles	Soluc. Obt.	Ópt.	Error	Res. Cont.	Difer.
steinb1	36	0:00:00	64	82	82	0,00%	77,75	4,25
steinb2	54	0:00:00	72	83	83	0,00%	82,5	0,5
steinb3	88	0:00:00	90	138	138	0,00%	138	0
steinb4	184	0:00:00	240	59	59	0,00%	50,5	8,5
steinb5	169	0:00:00	238	61	61	0,00%	52,167	8,833
steinb6	26294	0:00:02	264	122	122	0,00%	107,333	14,67
steinb7	60	0:00:00	110	111	111	0,00%	105	6
steinb8	9156	0:00:00	128	104	104	0,00%	92,75	11,25
steinb9	78	0:00:00	124	220	220	0,00%	217,5	2,5
steinb10	381	0:00:00	352	86	86	0,00%	70,333	15,67
steinb11	29714	0:00:03	384	88	88	0,00%	79	9
steinb12	737	0:00:00	376	174	174	0,00%	159,333	14,67
steinb13	207	0:00:00	184	165	165	0,00%	145,75	19,25
steinb14	4246	0:00:01	214	235	235	0,00%	211,917	23,08
steinb15	257	0:00:00	234	318	318	0,00%	313,5	4,5
steinb16	907	0:00:00	486	127	127	0,00%	109,833	17,17
steinb17	48332	0:00:04	454	131	131	0,00%	119,333	11,67
steinb18	449	0:00:00	496	218	218	0,00%	204,25	13,75
steinc1	12319	0:00:04	806	85	85	0,00%	52	33
steinc2	4993	0:00:02	724	144	144	0,00%	81,5	62,5
steinc3	4*10^5	0:00:46	946	754	754	0,00%	678,833	75,17
steinc4	3*10^5	0:00:22	1014	1079	1079	0,00%	1020,78	58,22
steinc5	3*10^6	0:05:57	1128	1579	1579	0,00%	1536,06	42,94
steinc6	28343	0:00:06	2406	55	55	0,00%	30,25	24,75
steinc7	1737	0:00:01	2498	102	102	0,00%	66	36
steinc8	5*10^6	0:11:52	2508	509	509	0,00%	428,167	80,83
steinc9	3*10^5	0:00:31	2642	707	707	0,00%	604,111	102,9
steinc10	4*10^5	0:00:43	2636	1093	1093	0,00%	1032,28	60,72
steinc11	14928	0:00:10	5008	32	32	0,00%	18,75	13,25
steinc12	93199	0:00:16	5128	46	46	0,00%	29,167	16,83
steinc13	3*10^6	0:07:37	5048	258	258	0,00%	219,583	38,42
steinc14	4398	0:00:06	4934	323	323	0,00%	298,715	24,29
steinc15	2*10^7	1:00:00	4630	557	556	0,18%	529,244	26,76
steinc16	13998	0:00:15	8034	11	11	0,00%	8,333	2,667
steinc17	15832	0:01:23	7926	18	18	0,00%	15,333	2,667
steinc18	18379	0:00:31	7992	113	113	0,00%	109,292	3,708
steinc19	7715	0:00:13	7704	146	146	0,00%	146	0
steinc20	9471	0:01:28	26000	267	267	0,00%	267	0
steind1	9764	0:00:02	1552	106	106	0,00%	69,667	36,33
steind2	3224	0:00:01	1604	220	220	0,00%	146,25	73,75
steind3	3599	0:00:02	1870	1565	1565	0,00%	1442,85	122,2
steind4	5222	0:00:01	1898	1935	1935	0,00%	1850,99	84,01
steind5	4910	0:00:02	2356	3250	3250	0,00%	3199,22	50,78

steind6	2*10^6	0:04:18	4978	67	67	0,00%	24	43
steind7	2730	0:00:04	4940	103	103	0,00%	66,25	36,75
steind8	6*10^5	0:02:05	5160	1072	1072	0,00%	880,708	191,3
steind9	2*10^6	0:06:49	5140	1448	1448	0,00%	1264,96	183
steind10	2*10^5	0:00:33	5234	2110	2110	0,00%	1984,32	125,7
steind11	36076	0:00:34	10870	29	29	0,00%	14,25	14,75
steind12	19638	0:00:28	10874	42	42	0,00%	25,542	16,46
steind13	3*10^5	0:01:18	10704	500	500	0,00%	435,722	64,28
steind14	8*10^6	1:00:00	10614	667	667	0,00%	604,153	62,85
steind15	2*10^5	0:00:58	9834	1116	1116	0,00%	1070,5	45,5
steind16	4*10^5	0:01:49	18096	13	13	0,00%	9	4
steind17	10^6	0:18:14	18122	23	23	0,00%	18,167	4,833
steind18	3*10^6	1:00:00	17510	223	223	0,00%	219,333	3,667
steind19	2*10^5	0:55:08	17104	310	310	0,00%	306,444	3,556
steind20	17258	0:00:37	15774	539	539	0,00%	539	0

Figura 32. Fuente: Elaboración propia

4.2 Conclusiones

Una vez hemos realizado el estudio de nuestra batería de problemas vamos a proceder a analizar las soluciones y sacar una serie de conclusiones derivadas de los resultados obtenidos.

Ya que hemos analizado dos tipos de problemas, el MST y el Steiner, los evaluaremos por separado.

4.2.1 ANÁLISIS PROBLEMAS MST

Por un lado, si nos fijamos en los resultados computacionales relacionados con el problema MST, podemos observar que tanto para la formulación MTZ como para la DL, los tiempos de resolución van aumentando progresivamente a medida que vamos avanzando en los problemas.

En la primera etapa correspondiente a los problemas de clase "b" la diferencia en el tiempo de resolución entre uno y otro es casi inexistente debido a que el programa es capaz de resolverlos en muy pocos segundos al tratarse de problemas de menor complejidad debido a su tamaño.

En la segunda etapa, correspondiente a los problemas de clase "c" podemos notar algo más de diferencia que con respecto a los problemas de tipo "b" ya que aquí observamos que en el caso de la formulación MTZ solo ha habido un problema, el steinc13, al cual no se ha llegado a la solución óptima después de una hora de resolución, mientras que en el caso de la formulación DL, esto ha ocurrido además de con el steinc13, con el steinc5 y steinc18.

Debido a que para los problemas MST no contábamos con la solución óptima y que el problema steinc13 no ha sido resuelto por ninguna de las dos estrategias no podemos evaluar el error en este caso.

Aunque sí que podemos evaluar el error para los problemas steinc5 y steinc18, donde podemos apreciar que, aunque el programa aún no había parado, el steinc5 ya había llegado al óptimo mientras que, el steinc18 no había llegado y su solución tiene un error del 1,59%. Para entenderlo mejor, a continuación, se presenta un gráfico que representa el valor del error de los distintos problemas.

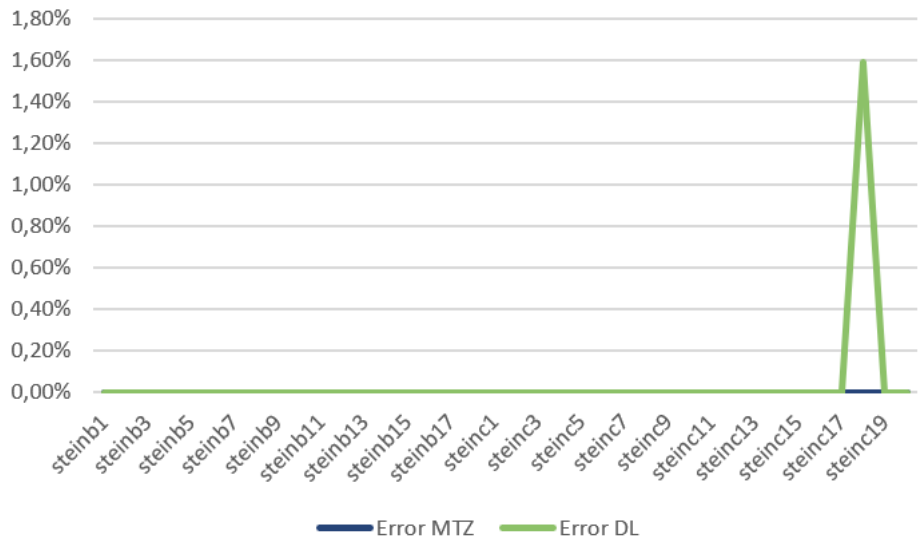

Figura 33. Fuente: Elaboración propia

Por otro lado, además de lo mencionado anteriormente de que los tiempos de resolución van aumentando progresivamente, no hemos podido detectar ningún patrón entre el número de nodos y el número de arcos y los tiempos de resolución.

A continuación, se muestra un resumen de la media de iteraciones por problema y la media de tiempo de los problemas en los que se ha llegado a la solución. Primero para la formulación MTZ y luego para la formulación DL.

Media iteraciones	Media Tiempo
1197502,703	0:03:57

Media iteraciones	Media tiempo
377023,9429	0:02:38

En este caso y aunque no tenemos demasiada información podríamos concluir que a pesar de que la media de tiempo y la media de iteraciones sea menor para la formulación DL, en general, para el problema MST sería mejor utilizar la formulación MTZ ya que se ha llegado al óptimo en todos los problemas a excepción de uno, mientras que con la formulación DL ha habido tres problemas a los que no se ha llegado al óptimo.

4.2.2 ANÁLISIS PROBLEMAS STEINER

Para el caso de los problemas Steiner, podemos observar lo mismo que para el problema MST. El tiempo de resolución de los problemas va aumentando a medida que vamos avanzando en ellos.

En la primera etapa, la cual corresponde a los problemas de clase de tipo “b”, prácticamente no hay diferencia entre los tiempos de resolución según una formulación y otra, esto se debe a que al ser problemas que no son muy complejos de resolver, el programa los resuelve inmediatamente.

Para la segunda etapa, correspondiente a los problemas de clase de tipo “c”, podemos notar más diferencia en los tiempos de resolución, de hecho, para el caso de la formulación MTZ no se ha conseguido llegar al óptimo en los problemas: steinc8, steinc9, steinc13, cuyo mayor error es para el steinc13 con un 3,10%. Por otro lado, en esta misma formulación, para el steinc15 y steinc18 ya se había llegado a la solución, pero el programa aún no había terminado de comprobar todas las posibles soluciones. Mientras, para la formulación DL para el único al que no se ha llegado al óptimo en una hora ha sido para el steinc15 con un error del 0,18%.

Para la tercera etapa, correspondiente a los problemas de tipo “d” es donde notamos la mayor diferencia en los tiempos de resolución. Para la estrategia de formulación MTZ no se ha conseguido llegar al óptimo ni en el problema steind6 ni en todos los problemas entre el steind8 y el steind20, con excepción del problema steind10, que se había llegado al óptimo pero el programa aún no había terminado de comprobar todas las posibles soluciones. Dentro de estos problemas a los que no se ha llegado al óptimo, el steind11 es el que se ha quedado más alejado con un error del 44,85%. Para la estrategia de formulación DL, en cambio, se ha conseguido llegar al óptimo en todos los problemas, aunque para el steind14 y steind18 el programa aún no había comprobado todas las soluciones posibles por lo que aún no había parado. Para entenderlo mejor, a continuación, se presenta un gráfico que representa el valor del error de los distintos problemas.

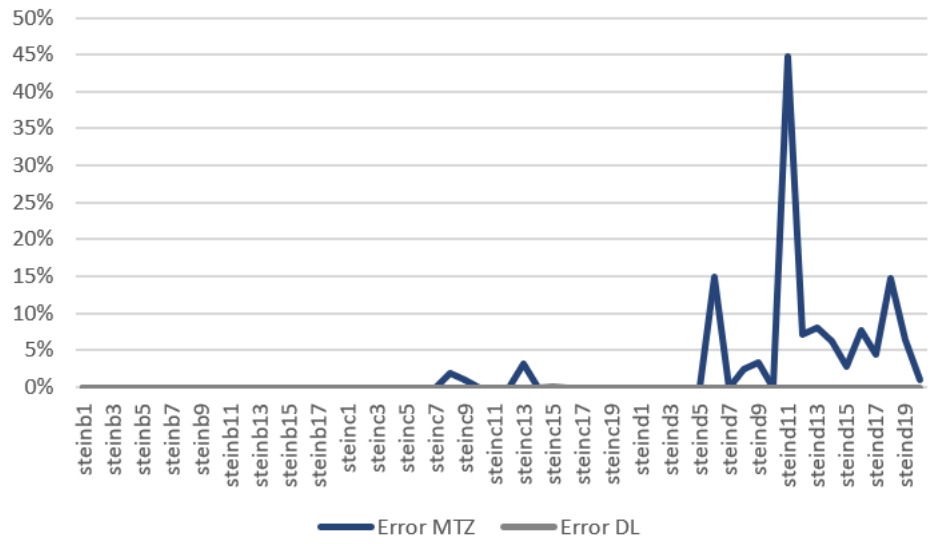

Figura 34. Fuente: Elaboración propia

Por otro lado, además de lo mencionado anteriormente de que los tiempos de resolución van aumentando progresivamente, en este caso tampoco hemos podido detectar ningún patrón entre el número de nodos y el número de arcos y los tiempos de resolución.

A continuación, se muestra un resumen de la media de iteraciones por problema y la media de tiempo de los problemas en los que se ha llegado a la solución. Primero para la formulación MTZ y luego para la formulación DL.

Media iteraciones	Media tiempo
772220,5128	0:05:45

Media iteraciones	Media tiempo
357132,7818	0:02:17

Por lo que con los datos con los que contamos, en este caso, podríamos decir que la estrategia más eficiente para este tipo de problemas sería la estrategia DL, ya que en este caso se puede observar que es bastante superior que la estrategia MTZ, además de haber conseguido llegar a la solución óptima en un mayor número de problemas.

4.2.3 ANÁLISIS DE LOS RESULTADOS CONTINUOS

Como ya expliqué anteriormente, se supone que en los problemas en los que su resultado continuo se encuentre cerca de su resultado entero se tardará menos en acercarse al óptimo que en aquellos en los que su resultado continuo se encuentre muy alejado.

A continuación, se muestra un gráfico de columnas apiladas para comparar las diferencias entre el resultado continuo y el resultado entero para el problema MST.

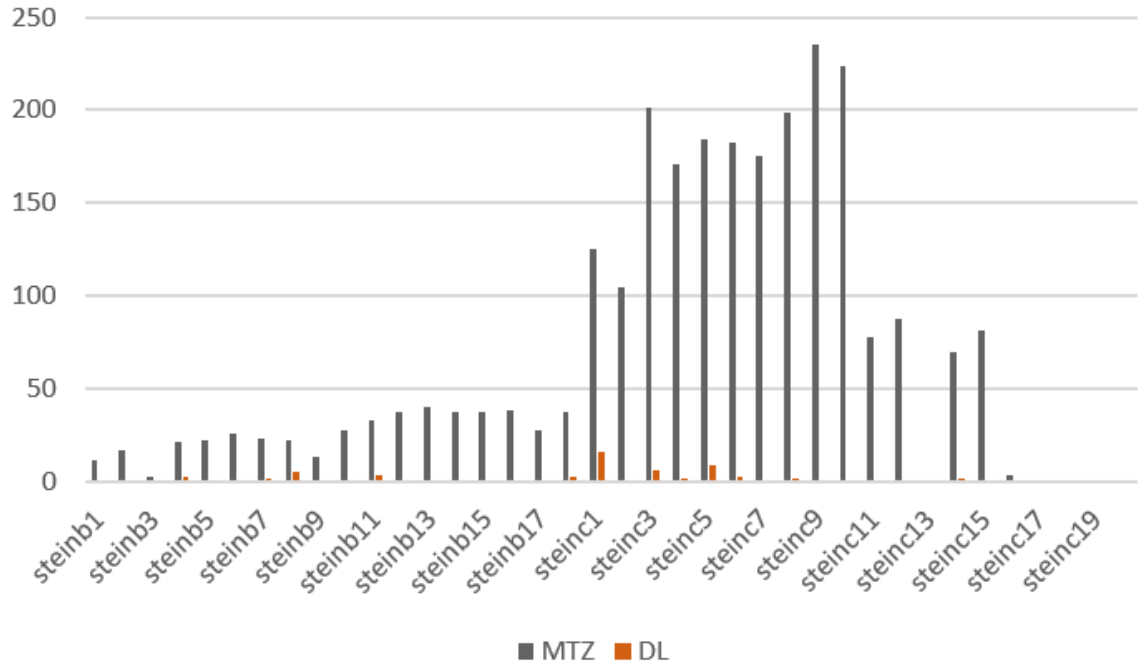

Figura 35. Fuente: Elaboración propia

Observando las diferencias que hay entre los resultados continuos y enteros de cada uno de los problemas, en teoría la formulación DL debería ser mucho más eficiente que la formulación MTZ. Hecho que no se cumple ya que para el caso de la formulación DL se ha llegado al óptimo en un número menor de problemas, por lo que podríamos concluir que la formulación MTZ es ligeramente mejor que la formulación DL.

A continuación, se muestra un gráfico de columnas apiladas para comparar las diferencias entre el resultado continuo y el resultado entero para el problema Steiner.

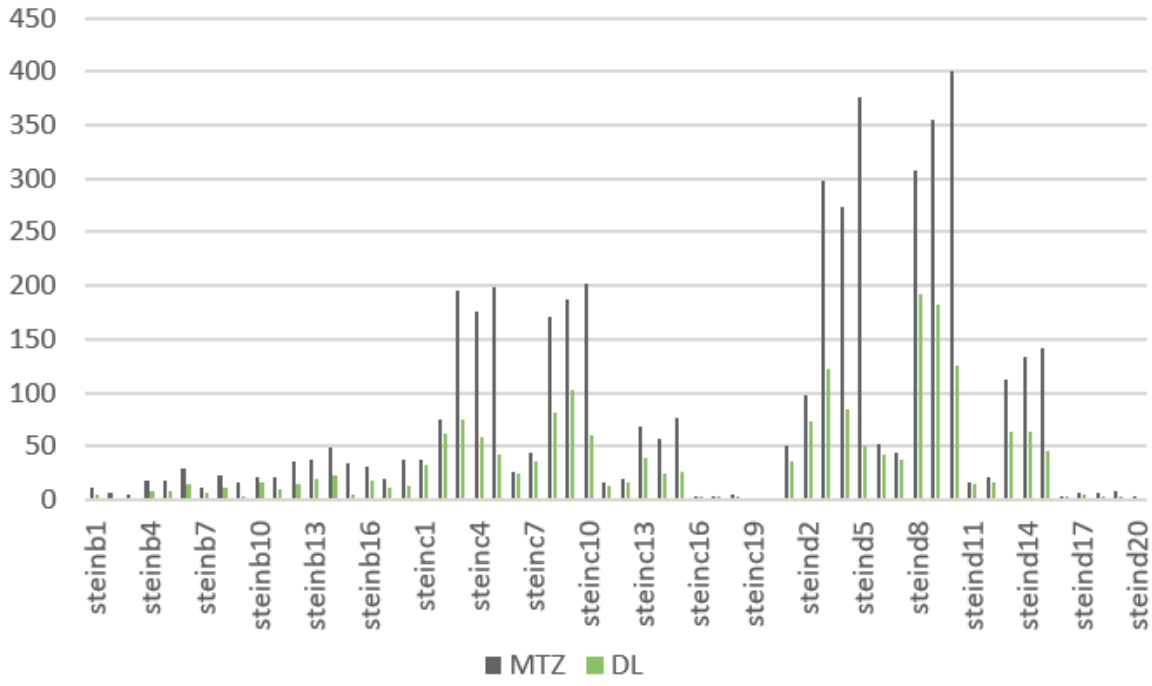

Figura 36. Fuente: Elaboración propia

Observando las diferencias entre los resultados continuos y enteros, en teoría la formulación DL debe ser mejor que la formulación MTZ. A diferencia del problema MST, podríamos concluir que en este caso si se cumple, ya que, observando los tiempos de resolución y el número de problemas en los que se ha llegado al óptimo, nos damos cuenta de que la formulación MTZ es mejor que la formulación DL.

5 BIBLIOGRAFÍA

1. Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints

Autores: Martin Desrochers y Gilbert Laporte

Año: 1990

2. Manejo de ficheros de texto en C

https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C/Manejo_de_archivos

3. Grafos

https://rio.upo.es/xmlui/bitstream/handle/10433/3597/2112-6647-1-SM.pdf?sequence=1&isAllowed=y

https://es.wikipedia.org/wiki/Teor%C3%ADa_de_grafos

4. LINGO

http://www1.frm.utn.edu.ar/ioperativa/lingo_lindo.pdf

5. Apuntes asignatura métodos cuantitativos de gestión

Resolución de Problemas. Librerías de Optimización.

Autor: José Manuel García Sánchez

Año: Sin especificar

6. Árbol de expansión mínima

https://jariasf.wordpress.com/2012/04/19/arbol-de-expansion-minima-algoritmo-de-kruskal/

http://www2.udec.cl/~grafos/grafos/teoria/teoria1.htm

7. Problema del viajante

http://www.cs.us.es/~fsancho/?e=71

8. OR- Library

Distributing Test Problems by Electronic Mail J. E. Beasley: The Journal of the Operational Research Society

6 ANEXO

En el anexo voy a incluir los códigos en C anteriormente descritos. Para que sea más fácil entenderlos he realizado anotaciones en el código.

6.1 Código del problema MST con la estrategia MTZ

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

//FUNCION QUE RELLENA TRES VECTORES CON EL NODO ORIGEN, NODO DESTINO Y COSTE
void pesos(int aristas, int * vector, int * vector2, int * vector3);
//FUNCION PARA RESERVAR MEMORIA EN VECTORES DINAMICOS
int * reservaMemoria (int aristas);
//FUNCIONES PARA ALMACENAR EL NUMERO DE NODOS Y DE ARISTAS
int almacenarnodos();
int almacenararistas();

int main()
{
    //DECLARACION DE VARIABLES
    //Nodos y aristas se corresponden a los valores de almacenar
    int nodos;
    int aristas;
    int arcos;
    //Este sera el vector correspondiente a  n1
    int *vector=NULL;
    //Este sera el vector correspondiente a n2
    int *vector2=NULL;
    //Este sera el vector correspondiente a costes
    int *vector3=NULL;
    //La variable i se corresponde con un contador
    int i=0;
```

```
 //DECLARACION DE FICHEROS
 FILE *fd=NULL;
 FILE *ft=NULL;

 //ABRIMOS LOS FICHEROS, EL ORIGINAL SERA EN MODO LECTURA Y EL DESTINO EN MODO ESCRITURA
 char direccion[]="C:\\Users\\isabe\\Documents\\Universidad\\Cuarto\\TFG\\mi tfg\\FINAL\\fichero.txt";
 ft = fopen(direccion,"rt");
     char direccion2[]="C:\\Users\\isabe\\Documents\\Universidad\\Cuarto\\TFG\\mi tfg\\FINAL\\fichero2.txt";
 fd = fopen(direccion2,"w");


 //CONTROL DE ERRORES PARA VER SI LOS FICHEROS ABREN CORRECTAMENTE
 //Si el fichero origen no ha abierto, entra aqui
 if (ft==NULL)
 {
     printf("Error al tratar de leer el archivo");
     return 1;
 }
 //Si el fichero destino no ha abierto, entra aqui
 else if (fd==NULL)
 {
     printf("Error al crear el archivo");
     return 1;
 }
```

```
//Si ambos han abierto, entra aquí
else
{
    //ESCRIBIMOS EN EL FICHERO DESTINO LAS PRIMERAS LINEAS QUE SE EXIGEN
    fprintf(fd,"MODEL: \nSETS: \n");
     //CALCULAMOS EL NUMERO DE FILAS DE ARISTAS Y DE NODOS QUE HAY
    aristas=almacenararistas();
    nodos=almacenarnodos();
    //ESCRIBIMOS LAS SIGUIENTES LINEAS
    arcos=aristas*2;
    fprintf(fd, "Nodo /1..%d/: x, Raiz;\nAristas /1..%d/: alfa, c, Origen, Destino;\nENDSETS \nDATA: \n  Origen = ",nodos,arcos);
    //RESERVAMOS MEMORIA PARA LOS VECTORES
    vector=reservaMemoria(aristas);
    vector2=reservaMemoria(aristas);
    vector3=reservaMemoria(aristas);

    //COMPROBAMOS QUE SE HA REALIZADO CORRECTAMENTE LA RESERVA DE MEMORIA DE AMBOS VECTORES

    if(vector==NULL)
        printf("FALLO EN LA RESERVA DE MEMORIA\n");

    if(vector2==NULL)
        printf("FALLO EN LA RESERVA DE MEMORIA\n");

    if(vector3==NULL)
        printf("FALLO EN LA RESERVA DE MEMORIA\n");


 //RELLENAMOS LOS VECTORES.
 pesos(aristas,vector, vector2,vector3);

 //ESCRIBIMOS DICHOS VECTORES EN EL FICHERO DESTINO
 //VECTOR DE n1.
 for (i=0; i<aristas; i++)
 {
     if(i==(aristas-1))

         fprintf(fd, "%d, ",vector[i]);
     else
      fprintf(fd, "%d, ",vector[i]);
 }
 for (i=0; i<aristas; i++)
 {
     if(i==(aristas-1))

         fprintf(fd, "%d;",vector2[i]);
     else
      fprintf(fd, "%d, ",vector2[i]);
 }
 fprintf(fd, "\n  Destino = ");

 //VECTOR DE n2
 for (i=0; i<aristas; i++)
 {
     if(i==(aristas-1))

         fprintf(fd, "%d, ",vector2[i]);
     else
      fprintf(fd, "%d, ",vector2[i]);
 }
  for (i=0; i<aristas; i++)
 {
     if(i==(aristas-1))

         fprintf(fd, "%d;",vector[i]);
     else
      fprintf(fd, "%d, ",vector[i]);
 }

 fprintf(fd, "\n  c = ");
```

```
 //VECTOR DE COSTES
 for (i=0; i<aristas; i++)
 {
     if(i==(aristas-1))

         fprintf(fd, "%d, ",vector3[i]);
     else
      fprintf(fd, "%d, ",vector3[i]);
 }
 for (i=0; i<aristas; i++)
 {
     if(i==(aristas-1))

         fprintf(fd, "%d;",vector3[i]);
     else
      fprintf(fd, "%d, ",vector3[i]);
 }

//Nodo Raiz
 fprintf(fd, "\n  Raiz = ");
  for (i=0; i<nodos; i++)
{
    if(i==0)

        fprintf(fd, "%d, ",1);
    else
    {
        if(i==(nodos-1))
        {
           fprintf(fd, "%d; ",0);
        }
        else
        {
            fprintf(fd, "%d, ",0);
        }
    }

}
           fprintf(fd, "\n  N= %d;",nodos);
           fprintf(fd, "\n  ENDDATA");
           fprintf(fd, "\n Min = @SUM(Aristas(j):c(j)*alfa(j)); \n ");
           fprintf(fd, "\n @FOR(Nodo(i)| Raiz(i)#EQ#0: @SUM(Aristas(j)| Destino(j)#EQ#i: alfa(j))=1);");
           fprintf(fd, "\n @FOR(Nodo(i)| Raiz(i)#EQ#1: @SUM(Aristas(j)| Origen(j)#EQ#i: alfa(j))>=1); ");
           fprintf(fd, "\n @FOR(Aristas(j)|Raiz(Origen(j))#EQ#0 #AND# Raiz(Destino(j))#EQ#0:");
           fprintf(fd, "-x(Origen(j))+ x(Destino(j)) + ((N-1)*alfa(j)) <= (N-2));");
           fprintf(fd, "\n @FOR(Aristas(j)|Raiz(Origen(j))#EQ#0: x(Origen(j))<=N-1);");
           fprintf(fd, "\n @FOR(Aristas(j)|Raiz(Origen(j))#EQ#0: x(Origen(j))>=1);");
           fprintf(fd, "\n @FOR(Aristas(j):@BIN(alfa(j))); ");
           fprintf(fd, "\nEND");

  }
  //CERRAMOS AMBOS FICHEROS
  fclose(ft);
  fclose(fd);
  return 0;
```

```
int * reservaMemoria (int filas)
{
    return (int *) malloc(sizeof(int)*filas);
}


void pesos(int aristas, int *vector, int *vector2, int *vector3)
{
    //CADENA EN LA QUE GUARDARE LA PRIMERA FILA DEL FICHERO ORIGEN
    char cadena [30];
    //CONTADOR PARA LOS VECTORES
    int i=0;
    //VARIABLE PARA IDENTIFICAR EN EL FICHERO DESTINO EL N1
    int num1=0;
    //VARIABLE PARA IDENTIFICAR EN EL FICHERO DESTINO EL N2
    int num2=0;
    //VARIABLE PARA IDENTIFICAR EN EL FICHERO DESTINO EL COSTE
    int num3=0;


    //DECLARAMOS EL PUNTERO DEL FICHERO
    FILE * fich=NULL;

    //ABRIMOS EL FICHERO ORIGEN EN MODO LECTURA
    fich = fopen("C:\\Users\\isabe\\Documents\\Universidad\\Cuarto\\TFG\\mi tfg\\FINAL\\fichero.txt", "r");


    //SI EL FICHERO ORIGEN ABRE BIEN, ENTRA AQUI (MISMO CONTROL DE ERROR QUE ANTES)
    if (fich)
    {
            //LEO LA PRIMERA LINEA QUE NO ME VA A SERVIR
            fgets(cadena,30,fich);

            //RECORRO SACANDO N1 N2 Y COSTES TRES ENTEROS
            while(fscanf(fich,"%d %d %d", &num1,&num2,&num3)==3)
            {
                //RELLENO LOS VECTORES
                vector[i]=num1;
                vector2[i]=num2;
                vector3[i]=num3;
                //INCREMENTO LA I PARA PODER RELLENAR EN LA SIGUIENTE POSICION DE LOS VECTORES
                i++;
            }

    }
    fclose(fich);

}

int almacenarnodos()
{
    //ESTE SERA EL VALOR DE VOLUMEN M
    int n;
    int a;

    //DECLARAMOS EL PUNTERO DEL FICHERO
     FILE * fich=NULL;

     //ABRIMOS EL FICHERO ORIGEN EN MODO LECTURA
    fich = fopen("C:\\Users\\isabe\\Documents\\Universidad\\Cuarto\\TFG\\mi tfg\\FINAL\\fichero.txt", "r");

    //SI EL FICHERO ORIGEN ABRE BIEN, ENTRARA AQUI (MISMO CONTROL DE ERROR)
    if (fich)
    {
            //LEO LOS DOS ENTEROS QUE HABRÁN EN LA PRIMERA LINEA Y LO METO EN LA VARIABLE N (NODOS) Y A(aristas)
            fscanf(fich,"%d %d",&n ,&a);

    }
    fclose(fich);
    return n;
}
```

```c
int almacenararistas()
{
    //ESTE SERA EL VALOR DE VOLUMEN M
    int n;
    int a;

    //DECLARAMOS EL PUNTERO DEL FICHERO
     FILE * fich=NULL;

     //ABRIMOS EL FICHERO ORIGEN EN MODO LECTURA
    fich = fopen("C:\\Users\\isabe\\Documents\\Universidad\\Cuarto\\TFG\\mi tfg\\FINAL\\fichero.txt", "r");

    //SI EL FICHERO ORIGEN ABRE BIEN, ENTRARA AQUI (MISMO CONTROL DE ERROR)
    if (fich)
    {
            //LEO LOS DOS ENTEROS QUE HABRÁN EN LA PRIMERA LINEA Y LO METO EN LA VARIABLE N (NODOS) Y A(aristas)
            fscanf(fich,"%d %d",&n ,&a);

    }
    fclose(fich);
    return a;
}
```

6.2 Código del problema MST con la estrategia DL

La mayor parte del código es común a los cuatro problemas por lo que sólo va a aparecer aquí las partes en las que varía el código con respecto al anterior.

```
fprintf(fd, "\n  N= %d;",nodos);
fprintf(fd, "\n  ENDDATA");
fprintf(fd, "\n Min = @SUM(Aristas(j):c(j)*alfa(j));");
fprintf(fd,"\n @FOR(Nodo(i)| Raiz(i)#EQ#0: @SUM(Aristas(j)| Destino(j)#EQ#i: alfa(j))=1); ");
fprintf(fd," \n@FOR(Nodo(i)| Raiz(i)#EQ#1: @SUM(Aristas(j)| Origen(j)#EQ#i: alfa(j))>=1);")
fprintf(fd, "\n @FOR(Aristas(j)| j#LE#%d #AND# Raiz(Origen(j))#EQ#0 #AND# Raiz(Destino(j))#EQ#0 :",aristas);
fprintf(fd,"-x(Origen(j))+x(Destino(j))+ (N-1)*alfa(j)+ (N-3)*alfa(j+%d) <= N-2); ",aristas);
int n=aristas+1;
fprintf(fd, "\n @FOR(Aristas(j)| j#GE#%d #AND# Raiz(Origen(j))#EQ#0 #AND# Raiz(Destino(j))#EQ#0 :",n);
fprintf(fd,"-x(Origen(j))+x(Destino(j))+ (N-1)*alfa(j)+ (N-3)*alfa(j-%d) <= N-2);",aristas);
fprintf(fd, "\n @FOR(Aristas(j)|Raiz(Destino(j))#EQ#0: x(Destino(j))<=N-1); ");
fprintf(fd, "\n @FOR(Aristas(j)|Raiz(Origen(j))#EQ#0: x(Origen(j))<=N-1); ");
fprintf(fd, "\n @FOR(Aristas(j)|Raiz(Destino(j))#EQ#0: x(Destino(j))>=1); ");
fprintf(fd, "\n @FOR(Aristas(j)|Raiz(Origen(j))#EQ#0: x(Origen(j))>=1);");
fprintf(fd, "\n @FOR(Aristas(j):@BIN(alfa(j))); ");
fprintf(fd, "\nEND");
```

6.3 Código del problema Steiner con la estrategia MTZ

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

//FUNCION QUE RELLENA TRES VECTORES CON EL NODO ORIGEN, NODO DESTINO Y COSTE
void pesos(int aristas, int * vector, int * vector2, int * vector3);
//FUNCION PARA RESERVAR MEMORIA EN VECTORES DINAMICOS
int * reservaMemoria (int aristas);
int * reservaMemoria2 (int x);
int * reservaMemoria3 (int nodos);
//FUNCIONES PARA ALMACENAR EL NUMERO DE NODOS Y DE ARISTAS
int almacenarnodos();
int almacenararistas();
//Para sacar el numero de nodos terminales(EL NUMERO DE COMPONENTES QUE VA A TENER MI VECTOR BINARIO)
int nodosterminales(int aristas);
// funcion que rellena el vector de nodos terminales
void vectorbinario(int aristas, int x,int *vector4);
//funcion para sacar el valor que hay que sumarle a la f.o.
int sumfunob(int aristas);

int main()
{
    //DECLARACION DE VARIABLES
    //Nodos y aristas se corresponden a los valores de almacenar
    int nodos;
    int aristas;
    int arcos;
    //Este sera el vector correspondiente a  n1
    int *vector=NULL;
    //Este sera el vector correspondiente a n2
    int *vector2=NULL;
    //Este sera el vector correspondiente a costes
    int *vector3=NULL;
    //Este será el vector correspondiente a los nodos terminales
    int *vector4=NULL;
    // Este será el vector que voy a imprimir correspondiente a los nodos terminales
    int *vector5=NULL;
    //La variable i, j y k son contadores
    int i=0;
    int j=0;
    int k=0;
    //La variable x se corresponde al número de nodos terminales
    int x=0;
    //La variable y se corresponde al valor que va a haber que sumarle a la F.O.
    int y=0;

    //DECLARACION DE FICHEROS
    FILE *fd=NULL;
    FILE *ft=NULL;

    //ABRIMOS LOS FICHEROS, EL ORIGINAL SERA EN MODO LECTURA Y EL DESTINO EN MODO ESCRITURA
    char direccion[]="C:\\Users\\isabe\\Documents\\Universidad\\Cuarto\\TFG\\Codigos en C\\fichero.txt";
    ft = fopen(direccion,"rt");
        char direccion2[]="C:\\Users\\isabe\\Documents\\Universidad\\Cuarto\\TFG\\Codigos en C\\steind20.txt";
    fd = fopen(direccion2,"w");


    //CONTROL DE ERRORES PARA VER SI LOS FICHEROS ABREN CORRECTAMENTE
    //Si el fichero origen no ha abierto, entra aqui
    if (ft==NULL)
    {
        printf("Error al tratar de leer el archivo");
        return 1;
    }
    //Si el fichero destino no ha abierto, entra aqui
    else if (fd==NULL)
    {
        printf("Error al crear el archivo");
        return 1;
    }
```

```
 //Si ambos han abierto, entra aqui
 else
 {
     //ESCRIBIMOS EN EL FICHERO DESTINO LAS PRIMERAS LINEAS QUE SE EXIGEN
     fprintf(fd,"MODEL: \nSETS: \n");
      //CALCULAMOS EL NUMERO DE FILAS DE ARISTAS Y DE NODOS QUE HAY
     aristas=almacenararistas();
     nodos=almacenarnodos();
     //ESCRIBIMOS LAS SIGUIENTES LINEAS
     arcos=aristas*2;
     fprintf(fd, "Nodo /1..%d/: x, Raiz,T,Beta;\nAristas /1..%d/: alfa, c, Origen, Destino;",nodos,arcos);
     fprintf(fd,"\nENDSETS \nDATA: \n  Origen = ");
     //Calculamos el número de nodos terminales que hay
     x=nodosterminales(aristas);
     //RESERVAMOS MEMORIA PARA LOS VECTORES
     vector=reservaMemoria(aristas);
     vector2=reservaMemoria(aristas);
     vector3=reservaMemoria(aristas);
     vector4=reservaMemoria2(x);
     vector5=reservaMemoria3(nodos);

     //COMPROBAMOS QUE SE HA REALIZADO CORRECTAMENTE LA RESERVA DE MEMORIA DE AMBOS VECTORES

     if(vector==NULL)
         printf("FALLO EN LA RESERVA DE MEMORIA\n");


if(vector2==NULL)
    printf("FALLO EN LA RESERVA DE MEMORIA\n");

if(vector3==NULL)
    printf("FALLO EN LA RESERVA DE MEMORIA\n");
 if(vector4==NULL)
    printf("FALLO EN LA RESERVA DE MEMORIA\n");

if(vector5==NULL)
    printf("FALLO EN LA RESERVA DE MEMORIA\n");

 // Calculamos el valor que hay que sumarle a la F.O.
 y=sumfunob(aristas);
//RELLENAMOS LOS VECTORES.
pesos(aristas,vector, vector2,vector3);
vectorbinario(aristas,x,vector4);
//ESCRIBIMOS DICHOS VECTORES EN EL FICHERO DESTINO
```

La parte de escribir los vectores es igual a las dos anteriores formulaciones con excepción del vector binario y el nodo raíz:

```
    //VECTOR BINARIO
    for (i=1; i<nodos+1; i++)
    {
        if(i==(nodos))

            fprintf(fd, "%d;",vector5[i]);
        else
         fprintf(fd, "%d, ",vector5[i]);
    }


//Nodo Raiz
 fprintf(fd, "\n  Raiz = ");
  for (i=1; i<nodos+1 ; i++)
{

        if(i==(nodos))
        {
           fprintf(fd, "%d; ",0);
        }
        else
        {   //Para asegurarnos que el nodo raiz es terminal
            if(k==0 && vector5[i]==1)
            {
                fprintf(fd,"%d, ",1);
                k=k+1;
            }
            else
            {
                fprintf(fd, "%d, ",0);
            }

        }
    }
```

```
            fprintf(fd, "\n  N= %d;",nodos);
            fprintf(fd, "\n  ENDDATA");
            fprintf(fd, "\n Min = @SUM(Aristas(j):c(j)*alfa(j))+%d;",y);
            fprintf(fd,"\n @FOR(Nodo(i)| Raiz(i)#EQ#0 #AND# T(i)#EQ#1: @SUM(Aristas(j)| Destino(j)#EQ#i: alfa(j))=1);");
            fprintf(fd,"\n @FOR(Nodo(i)| Raiz(i)#EQ#0 #AND# T(i)#EQ#0: @SUM(Aristas(j)| Destino(j)#EQ#i: alfa(j))=Beta(i));");
            fprintf(fd,"\n @FOR(Nodo(i)| Raiz(i)#EQ#0 #AND# T(i)#EQ#0: @FOR(Aristas(j)| Origen(j)#EQ#i: alfa(j)<=Beta(i)));");
            fprintf(fd,"\n@FOR(Nodo(i)| Raiz(i)#EQ#1: @SUM(Aristas(j)| Origen(j)#EQ#i: alfa(j))>=1);");
            fprintf(fd, "\n @FOR(Aristas(j)|Raiz(Origen(j))#EQ#0 #AND# Raiz(Destino(j))#EQ#0: ",aristas);
            fprintf(fd,"-x(Origen(j))+ x(Destino(j)) + ((N-1)*alfa(j)) <= (N-2)); ");
            fprintf(fd, "\n @FOR(Aristas(j)|Raiz(Origen(j))#EQ#0: x(Origen(j))<=N-1);");
            fprintf(fd, "\n @FOR(Aristas(j)|Raiz(Origen(j))#EQ#0: x(Origen(j))>=1);");
            fprintf(fd, "\n @FOR(Aristas(j):@BIN(alfa(j))); ");
            fprintf(fd, "\n @FOR(Nodo(i):@BIN(Beta(i))); ");
            fprintf(fd, "\nEND");

    }
    //CERRAMOS AMBOS FICHEROS
    fclose(ft);
    fclose(fd);
    return 0;
}

int * reservaMemoria (int filas)
{
    return (int *) malloc(sizeof(int)*filas);
}

int * reservaMemoria2 (int x)
{
    return (int *) malloc(sizeof(int)*x);
}
int * reservaMemoria3 (int nodos)
{
    return (int *) malloc(sizeof(int)*nodos);
}
```

Aquí vendrían las funciones de pesos, almacenar aristas y almacenar nodos que son iguales a las dos formulaciones anteriores.

```
int nodosterminales(int aristas)
{
    int x,h;
    int i=0;
char cadena [30];
aristas=aristas+1;
     FILE * fich=NULL;
    fich = fopen("C:\\Users\\isabe\\Documents\\Universidad\\Cuarto\\TFG\\Codigos en C\\fichero.txt", "r");

    if (fich)
    {
            while (fgets(cadena,30,fich))
        {
           i++;
            while(fscanf(fich,"%d", &x)==1 && i==aristas)
                  {
                      h=x;
                      break;
                  }
        }


    }
    return h;
    fclose(fich);

}
```

```c
void vectorbinario(int aristas, int x,int *vector4)
{
    char cadena [30];
    int i=0;
    int numl=0;
    int j=0;

    aristas=aristas+1;
    FILE * fich=NULL;
    fich = fopen("C:\\Users\\isabe\\Documents\\Universidad\\Cuarto\\TFG\\Codigos en C\\fichero.txt", "r");
    if (fich)
    {
            fgets(cadena,30,fich);
 while (fgets(cadena,30,fich))
        {
           i++;
            while(fscanf(fich,"%d", &numl)==1 && i==aristas)
            {
                vector4[j]=numl;
                //INCREMENTO LA J PARA PODER RELLENAR EN LA SIGUIENTE POSICION DE LOS VECTORES
                j++;
            }
        }

    }
    fclose(fich);

}

//Para sacar el número que hay que sumarle a la función objetivo
int sumfunob(int aristas)
{
    int x,h;
    int i=0;
char cadena [30];
aristas=aristas+1;
     FILE * fich=NULL;
    fich = fopen("C:\\Users\\isabe\\Documents\\Universidad\\Cuarto\\TFG\\Codigos en C\\fichero.txt", "r");

    if (fich)
    {
            while (fgets(cadena,30,fich))
        {
           i++;
            while(fscanf(fich,"%d", &x)==1 && i>aristas)
                {
                    h=x;
                    break;
                }
        }


    }
    return h;
    fclose(fich);
```

6.4 Código del problema Steiner con la estrategia DL

La mayor parte de este código es común al anterior por lo que nada más que van a aparecer las partes en las que haya habido algún cambio.

```
            fprintf(fd, "\n  N= %d;",nodos);
            fprintf(fd, "\n  ENDDATA");
            fprintf(fd, "\n Min = @SUM(Aristas(j):c(j)*alfa(j)) + %d ;",y);
            fprintf(fd,"\n @FOR(Nodo(i)| Raiz(i)#EQ#0 #AND# T(i)#EQ#1: @SUM(Aristas(j)| Destino(j)#EQ#i: alfa(j))=1);");
            fprintf(fd,"\n @FOR(Nodo(i)| Raiz(i)#EQ#0 #AND# T(i)#EQ#0: @SUM(Aristas(j)| Destino(j)#EQ#i: alfa(j))=Beta(i));");
            fprintf(fd,"\n @FOR(Nodo(i)| Raiz(i)#EQ#0 #AND# T(i)#EQ#0: @FOR(Aristas(j)| Origen(j)#EQ#i: alfa(j)<=Beta(i)));");
            fprintf(fd,"\n@FOR(Nodo(i)| Raiz(i)#EQ#1: @SUM(Aristas(j)| Origen(j)#EQ#i: alfa(j))>=1);");
            fprintf(fd, "\n @FOR(Aristas(j)| j#LE#%d #AND# Raiz(Origen(j))#EQ#0 #AND# Raiz(Destino(j))#EQ#0 :",aristas);
            fprintf(fd,"-x(Origen(j))+x(Destino(j))+ (N-1)*alfa(j)+ (N-3)*alfa(j+%d) <= N-2);",aristas);
            int n=aristas+1;
            fprintf(fd, "\n @FOR(Aristas(j)| j#GE#%d #AND# Raiz(Origen(j))#EQ#0 #AND# Raiz(Destino(j))#EQ#0 : ",n);
            fprintf(fd,"-x(Origen(j))+x(Destino(j))+ (N-1)*alfa(j)+ (N-3)*alfa(j-%d) <= N-2);",aristas);
            fprintf(fd, "\n @FOR(Aristas(j)|Raiz(Destino(j))#EQ#0: x(Destino(j))<=N-1); ");
            fprintf(fd, "\n @FOR(Aristas(j)|Raiz(Origen(j))#EQ#0: x(Origen(j))<=N-1); ");
            fprintf(fd, "\n @FOR(Aristas(j)|Raiz(Destino(j))#EQ#0: x(Destino(j))>=1); ");
            fprintf(fd, "\n @FOR(Aristas(j)|Raiz(Origen(j))#EQ#0: x(Origen(j))>=1);");
            fprintf(fd, "\n @FOR(Aristas(j):@BIN(alfa(j))); ");
            fprintf(fd, "\n @FOR(Nodo(i):@BIN(Beta(i))); ");
            fprintf(fd, "\nEND");

    }
    //CERRAMOS AMBOS FICHEROS
    fclose(ft);
    fclose(fd);
    return 0;
}
```