Trabajo Fin de Grado
Grado en Ingeniería de Organización Industrial

APLICACIÓN DE LA FORMULACIÓN de R.k. Martin AL PROBLEMA DE STEINER EN GRAFOS

Autor: Sergio Flores Medina
Tutor: Jose Manuel García Sánchez

Dpto. Organización Industrial y Gestión de Empresas I
Universidad de Sevilla

Sevilla, 2019

Proyecto Fin de Carrera
Grado en Ingenieria de Organización Industrial

APLICACIÓN DE LA FORMULACIÓN de R.k. Martin AL PROBLEMA DE STEINER EN GRAFOS

Autor:
Sergio Flores Medina

Tutor:
Jose Manuel García Sánchez
Profesor titular

Dpto. de Organización Industrial y Gestion de Empresas I
Universidad de Sevilla
Sevilla, 2019

Proyecto Fin de Carrera: APLICACIÓN DE LA FORMULACIÓN de R.k. Martin AL PROBLEMA DE STEINER EN GRAFOS

Autor: Sergio Flores Medina

Tutor: Jose 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

A mi familia

A mis amigos

A mis maestros

Resumen

En este proyecto de fin de grado evaluaremos una formulación matemática para problemas de selección de árboles en grafos.

Concretamente, se van a estudiar dos problemas, el problema de MST (Minimum Spanning Tree) y el problema de Steiner. La formulación que se va a evaluar es la de R.k. Martin. Esta formulación conecta a los nodos del problema sin permitir ciclos. Existen otras formulaciones que no permiten ciclos, pero no se ha encontrado ningún análisis experimental de la eficiencia de dicha formulación.

La eficiencia se va a medir con una batería de problemas existente en la web y se van a evaluar los tiempos de resolución de dicha estrategia de formulación aplicada sobre esta batería de problemas.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN Y OBJETIVO DEL TRABAJO

En los últimos años, la demanda en comunicaciones de datos ha crecido exponencialmente por lo que se han necesitado nuevas y mejores alternativas o metodologías para dar solución a este tipo de problema. Debido a esto, el uso y la importancia de los grafos ha crecido exponencialmente también.

Actualmente, el uso de grafos para la resolución de problemas es algo muy común debido a su versatilidad y efectividad. Como explicaremos más adelante, una red de grafos es un conjunto de nodos o vértices unidos por aristas o arcos.

El objetivo a la hora de resolver cualquier problema de grafos es obtener una red (conjunto de aristas/arcos y nodos/vértices) que satisfaga mejor la función objetivo que se proponga en cada caso. En esta red, no suele ser necesario coger todas las aristas y todos los nodos, por lo que el problema será decidir qué nodos y qué aristas formarán parte de esta red.

La estrategia que usaremos para resolver este problema de grafos es la estrategia propuesta por R.k. Martin de la cual no hemos encontrado ninguna experimentación. Esta estrategia se presentó aplicada al problema MST y nosotros la extenderemos para presentarla también al problema de Steiner. El problema MST lo resolveremos como caso particular de tipo p donde todos los nodos son terminales.

En este trabajo, las funciones objetivo serán las proporcionadas por esos dos problemas, el MST (Minimun Spanning Tree), que consistirá en unir todos los nodos a mínimo costo, y por el problema de Steiner en la que uniremos a mínimo costo los nodos denominados como terminales.

Esta estrategia propuesta por R.k. Martin consigue, además, que no se forme ningún ciclo en su interior, algo que suele crear ineficiencias y es mejor evitar.

Por último, se hará un análisis experimental de los resultados comparándolos con otras formulaciones para ver así su eficiencia.

Para realizar este trabajo, trabajaremos con esta formulación resolviendo una batería de problemas ya predefinida. Esto lo haremos creando un código en C++ para automatizar el proceso de construcción del problema y posteriormente resolveremos el problema en LINGO.

En resumen, el objetivo del trabajo será resolver distintos problemas de grafos del tipo MST y steiner de forma óptima utilizando la formulación de R.k. Martin la cual evita ciclos y analizar los resultados y el tiempo de operación.

2 INTRODUCCIÓN A LOS GRAFOS

2.1 Definición de grafo

Dentro de la rama de las ciencias, podemos definir un grafo como un conjunto de vertices o nodos que están unidos mediante aristas o arcos.

La palabra grafo, viene de la palabra griega “grafos”, que significa dibujo. El objetivo de usar esta notación, es expresar las relaciones que tienen ciertas unidades de una manera gráfica y sencilla. Tan polivalente es el grafo, que prácticamente cualquier problema puede ser representado usando esta notación.

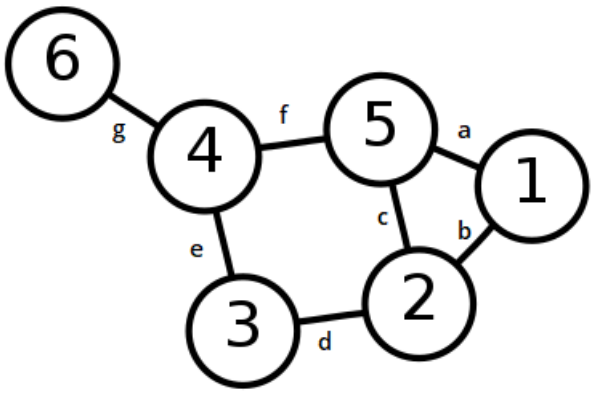


Figura 1 Grafo

Fuente: Elaboración propia

En este ejemplo podemos observar de forma mas clara los componentes de un grafo. Todo grafo, podrá ser expresado como un par ordenado G = (V,E) donde ‘V’ representa el número de nodos que normalmente suele ser finito y ‘E’ el número de aristas Como podemos apreciar, este grafo tiene 6 nodos los cuales están unidos entre sí. Estas uniones pueden ser de dos tipos (aristas o arcos), lo que cambiaría el grafo completamente. Esto diferencia a los grafos en dirigidos y no dirigidos, de los cuales veremos sus particularidades a continuación.

2.2 Tipos de grafos

En este apartado vamos a estudiar las distintas tipologías de grafos que existen o al menos las más importantes.

-Grafo simple: Este tipo de grafo tiene lugar cuando como mucho solo hay una arista de unión entre dos nodos.

Figura 2 Grafo simple

Fuente: Elaboración propia

-Multigrafo: También es conocido como grafo general. El grafo simple es una subclase de este grafo y este grafo ocurre cuando se permiten 2 aristas entre dos nodos. Estas aristas pueden llamarse en este caso lazo o múltiple.

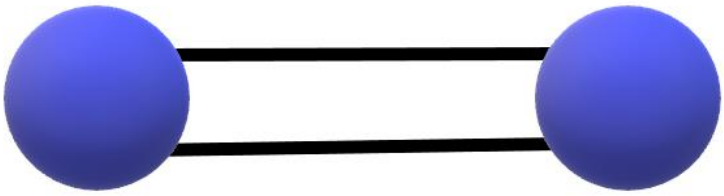


Figura 3 Multigrafo

Fuente: Elaboración propia

-Hipergrafo: Grafo en el cual las aristas tienen más de dos extremos, es decir, inciden sobre 2 o más vértices.

-Grafo infinito: Grafo de cardinalidad[1] infinita

-Grafo aleatorio: Grafo en el cual las aristas tienen unas probabilidades determinadas

-Grafo plano: Grafo en el cual los nodos y aristas pueden representarse en dos dimensiones sin que haya ninguna intersección entre ellos.

[1] Cardinalidad: Número de elementos de un conjunto

2.3 Caracterización de grafos

Podemos dividir a los grafos en dos grandes grupos. El primer grupo estará formado por los grafos no dirigidos. Para estos grafos, todos los pares posibles son no ordenados. Esto quiere decir por ejemplo, que si el grafo anterior fuera no dirigido cmpliría la siguiente propiedad {3,2}={2,3}. Ver en: Figura 1 Grafo

A este tipo de uniones entre grafos no dirigidos se le llamará aristas.

Por otro lado, tenemos los grafos no dirigidos los cuales son un poco más complejos.

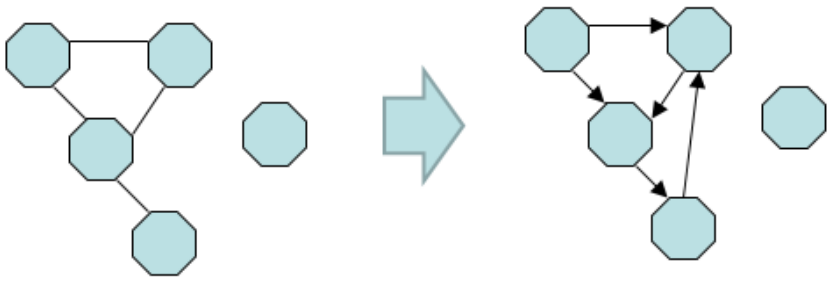


Figura 4 Grafo dirigido

Fuente: Apuntes de Jose Manuel García

En la "Figura 4 Grafo dirigido" podemos observar el cambio respecto al grafo dirigido. Este tipo de grafos tiene sus uniones direccionadas y por eso mismo se llamará dirigido. En este caso, remitiéndonos al ejemplo anterior, podremos afirmar que no tiene por qué cumplirse que {3,2}={2,3} aunque podría suceder perfectamente. A partir de ahora, cuando hablemos de las uniones de estos grafos no dirigidos, hablaremos sobre arcos.

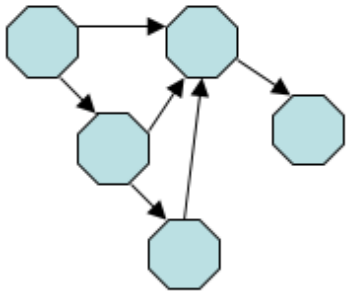


Figura 5 Grafo dirigido conexo

Fuente: Apuntes de Jose Manuel García

No en todos los grafos es necesario que todos los nodos estén conectados mediante arcos o aristas. Por lo tanto, diferenciaremos el grafo en el cual todos los nodos están unidos llamándolo "conexo".

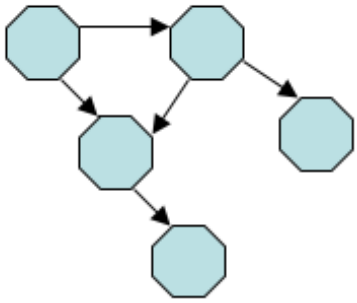


Figura 6 Grafo dirigido conexo acíclico

Fuente: Apuntes de Jose Manuel García

Por otro lado diferenciaremos tipos de grafos dependiendo de si sus cíclicos son acíclicos o no. En este caso, en la Figura 6 Grafo dirigido conexo acíclico podemos apreciar como es acíclico debido a que en ningún momento podríamos entrar en un bucle infinito. En el caso en el que se pudiera no sería acíclico. Este concepto lo explicaremos mejor en el apartado "Propiedades de un grafo".

Por ultimo, explicaremos la estructura de árbol que puede existir dentro de los grafos.

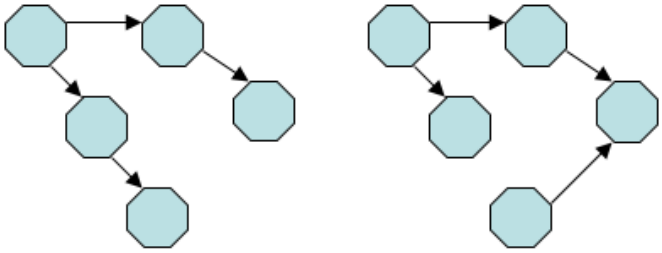
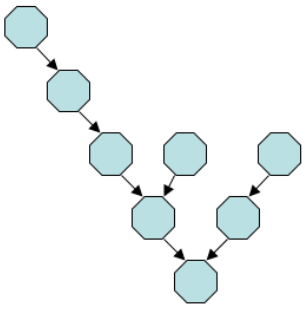


Figura 7 Árbol GDA sin caminos cerrados

Fuente: Apuntes de Jose Manuel García

La estructura de árbol será una estructura muy importante para analizar en este TFG. Esta estructura evita que se pueda cualquier ningún ciclo dentro de su propia estructura. Por lo tanto la unión puede que adopte forma de árbol, algo que observaremos mejor en la siguiente imagen. Esta estructura puede evitar ineficiencias a la hora de resolver problemas por lo que suele ser preferible.

- *Intree*: Árbol donde cada trabajo tiene a lo sumo un sucesor



- *Outtree*: Árbol donde cada trabajo tiene a lo sumo un predecesor

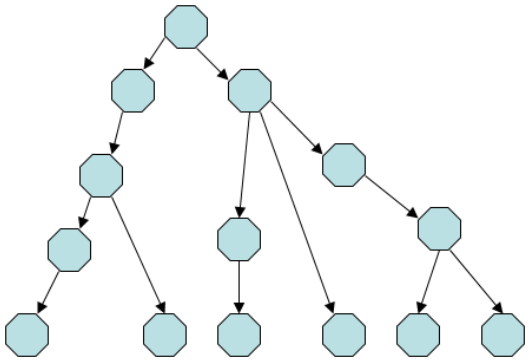


Figura 8 Intree y Outtree

Fuente: Apuntes de Jose Manuel García

Como podemos observar en la Figura 8 Intree y Outtree podemos apreciar mejor esa forma de árbol de la que se ha hablado anteriormente. Dentro de la estructura de árbol que puede tomar un grafo podemos diferenciar dos tipos. Los "Intree" en los cuales cada nodo tiene a lo sumo un sucesor pudiendo así tener varios predecesores o la estructura "Outtree" en la cual cada trabajo tiene a lo sumo un precedesor y análogamente puede tener más de un sucesor.

Si mezclamos ambas estructuras podemos llegar a tener un camino cerrado y tener ese riesgo de tener ineficiencias que es mejor evitar. Por lo tanto, habrá que tener cuidado si mezclamos ambas estructuras.

2.4 Propiedades de un grafo

En este apartado se verán las distintas propiedades que se cumplen cuando usamos la estructura de grafos.

Adyacencia: Se dirá que dos aristas o arcos son adyacentes si hay un mismo vértice o nodo que une a ambas.

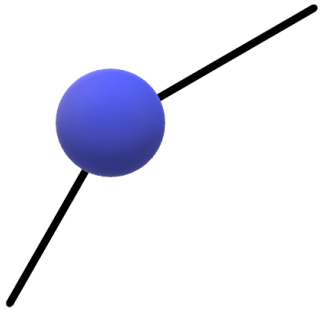


Figura 9 Adyacencia entre dos aristas

Fuente: Elaboración propia

Por ejemplo, como podemos ver en la "FIGURA 9", la arista A y B son adyacentes debido a que hay un nodo que las une a ambas.

-Ponderación: Para resolver los problemas de grafos es necesario definir los valores o pesos de los distintos actores que participan en dicho problema. Para ello se usarán las aristas, adjudicándoles su valor correspondiente para posteriormente resolver el problema. Por ejemplo en una de las muchas aplicaciones de los grafos, la red de carreteras, cada arista tiene el coste que supone ir por ese determinado camino. Las aplicaciones las estudiaremos más a fondo más adelante.

-Incidencia: Esta propiedad se cumple cuando dos nodos están unidos mediante una arista. Básicamente, una arista solo puede incidir a dos vértices o nodos.

-Etiquetado: Es una función que hace que podamos distinguir al nodo o a la arista que nosotros elijamos haciéndolos distintos al resto.

-Ciclos: Se dirá que existe un ciclo cuando avanzando por una arista a través de un nodo inicial, se puede volver al mismo nodo inicial sin haber cruzado por una misma arista dos veces. La mayoría de los grafos contienen ciclos y suelen ser un verdadero problema cuando intentamos resolver un problema de este tipo ya que crean ineficiencias. Dentro de los tipos de ciclos podemos distinguir dos tipos. El primer tipo son los acíclicos que ocurre cuando existe una union entre tres nodos pero la dirección de los arcos evita que partiendo de una nodo inicial se vuelva a pasar por ese mismo nodo. Este tipo de grafo lo podemos observar más claramente en la Figura 6 Grafo dirigido conexo acíclico. El segundo tipo de ciclos se observa cuando sí podemos volver al nodo inicial partiendo de una arista y sin repetir ninguna de ellas por el camino como hemos comentado antes. Este ciclo es aún peor que el anterior debido a que además de ineficiencias puede crear bucles infinitos. Existe también el llamado ciclo Hamiltoniano que ocurre cuando podemos recorrer el grafo pasando por los vértices solo una vez excepto el vértice de inicio y fin que es el mismo.

Cuando en cierto grafo cogemos una ruta determinada siendo esta una sucesión de vértices y de aristas la llamaremos camino de longitud n siendo n el conjunto de vértices que estén en el camino. Si todas sus aristas son distintas diremos que es simple y diremos que es cerrado si el vértice final= vértice inicial. En este caso podremos hablar igualmente de ciclo. Una trayectoria es un camino simple en el que todos sus vértices (salvo el final y el inicial en el caso de que sea un circuito cerrado) son distintos. En el caso en que la trayectoria sea cerrada como bien hemos dicho, la llamaremos circuito.

2.5 Estructuras matriciales

Para hablar sobre las estructuras matriciales usaremos como referencia la Figura 1 Grafo.

Primero definiremos los conjuntos. Para ellos definiremos V que es el conjunto de nodos V={1,2,3,4,5,6} y E que análogamente es el conjunto de aristas E={1,2},{2,3}, {1,5}, {2,5}, {4,5}, {3,4}, {4,6}

Dentro de las estructuras matriciales podemos destacar la matriz de adyacencia en la cual si solo hay una arista entre vértice y vértice tendrá valor 1 y tendrá valor 0 si la separación es mayor.

Tabla 1 Matriz de adyacencia

Nodos	1	2	3	4	5	6
1	0	1	0	0	1	0
2	1	0	1	0	1	0
3	0	1	0	1	0	0
4	0	0	1	0	1	1
5	1	1	0	1	0	0
6	0	0	0	1	0	0

Fuente: Elaboración propia

Por otro lado existe la matriz de incidencia en la cual se tomará el valor 1 si el nodo está conectado a la arista y 0 si no. Por lo tanto en un grafo simple como es nuestro caso, cada arista tendrá dos 1 siempre.

Tabla 2 Matriz de incidencia

Nodos	a	b	c	d	e	f	g
1	1	1	0	0	0	0	0
2	0	1	1	1	0	0	0
3	0	0	0	1	1	0	0
4	0	0	0	0	1	1	1
5	1	0	1	0	0	1	0
6	0	0	0	0	0	0	1

Fuente: Elaboración propia

2.6 Historia de los grafos

Actualmente, la teoría de grafos es una de las ramas más importantes de las matemáticas modernas. Esta forma de analizar los problemas tuvo sus principios en el Siglo XVIII cuando una de las mentes más brillantes que ha existido, la del matemático Suizo Leonhard Euler, creó un artículo en 1736 titulado “Solutio problematis ad geometrian situs pertinentis” que en español sería “La solución de un problema referente a la geometría de posición”. En este artículo se soluciona el famoso problema de los puentes de Königsberg.

Königsberg es actualmente Kaliningrado, Rusia y antes era una ciudad de Prusia del Siglo XVIII. El problema que queremos resolver tiene como protagonista un río, el río Pregel, el cual contenía a dos islas (C Y D) a través del mismo y comunicaba las dos partes de la ciudad (A y B) con las mismas. Todo esto anteriormente descrito se puede ver más claramente en la figura propuesta a continuación.

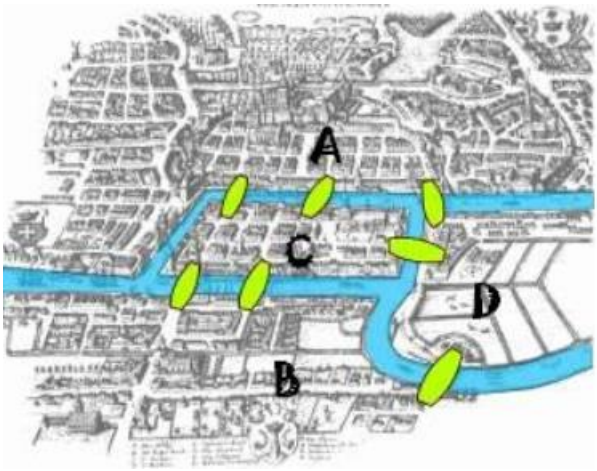


Figura 10 Puentes de Königsberg

Fuente: https://es.wikipedia.org/wiki/Problema_de_los_puentes_de_K%C3%B6nigsberg

Este río tenia 7 puentes y el problema consistía en cruzarlos todos y volver al punto de partida sin repetir ningún puente. Básicamente lo que queremos lograr es un camino simple cerrado pasando por todas las aristas lo que actualmente también se denomina como circuito euleriano.

Para la resolución de este problema, Euler transformó las distintas partes de la ciudad (A, B, C y D) en nodos y a los puentes en aristas costruyendo uno de los primeros grafos:

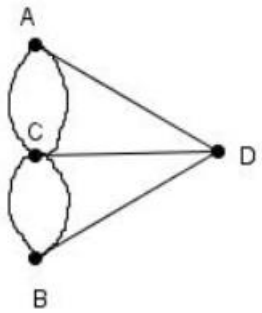


Figura 11 Grafo simplificado de la ciudad de Königsberg

Fuente: https://www.gaussianos.com/los-puentes-de-konigsberg-el-comienzo-de-la-teoria-de-grafos/

Una vez llegados a este punto, uno puede intentar resolver ese problema todas las veces que quiera que no lo resolverá. Esto es porque para poder recorrer un sistema de este tipo, los vértices intermedios deben tener un número par de aristas. Esto es para que tengan una vía de entrada y una vía de salida. El ejemplo se puede ver más claro si tenemos dos islas y queremos ir y volver de una a otra. Obviamente necesitaremos un puente para ir y otro para volver ya que con uno solo nos quedaríamos atrapados en la segunda isla. Por lo que extrapolando este ejemplo podemos concluir que este problema no es resoluble.

Anteriormente hemos explicado como el problema no sería resoluble gracias a que no hemos podido resolver el problema y extrapolando el miniproblema de las dos islas y el puente intermediario. Ahora vamos a explicar donde está la genialidad de Euler.

Euler explica que, si el punto de llegada y de salida es el mismo, obligatoriamente vamos a necesitar un número par de aristas. Esto se conoce como ciclo euleriano.

Si por el contrario el de llegada y salida fueran distintos, debe tener un número impar de aristas. A esto se le conocería como camino euleriano.

La genialidad de Euler de la que hablábamos se encuentra en que esto se puede aplicar a cualquier problema de este tipo. Por lo tanto, lo único que habrá que hacer es ver el nodo de inicio y el nodo final y contando las aristas que hay sabremos si es o no resoluble.

2.7 Aplicaciones de los grafos

Si bien es cierto que los grafos tienen infinidad de aplicaciones, aquí intantaremos concentrarnos brevemente en las más importantes. Una de las que más destaca es la aplicación de los grafos a la red de carreteras donde cada arista representa una carretera o vía y cada nodo una intersección.

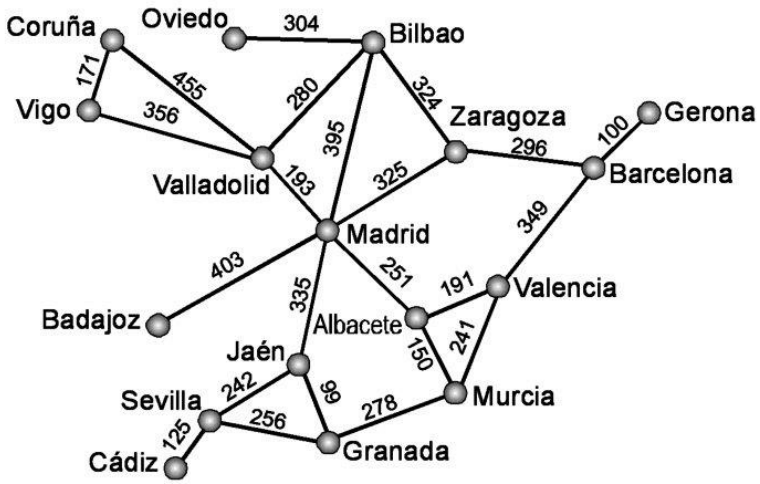


Figura 12 Red de carreteras

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

En un intento de simplificación de España, podemos ver esa aplicación directa a la red de carreteras donde cada nodo representa una ciudad y cada arista la union entre ellas con su peso incluído que en este caso es la distancia. Esto se lleva hasta la carretera más simple y gracias a los grafos y a la contribución de Floyd[2], funciona google Maps y es tan fácil encontrar la ruta más rápida entre dos puntos.

Otras aplicaciones importantes se encuentran en la biología y el habitat donde cada nodo representa un habitat y cada arista la ruta de migración que pueden hacer determinadas especies. Además también son usados los grafos en problemas de control de la producción, problemas de genética, red de informática y muchos más entre otros. Todo esto demuestra la versatilidad que tiene esta herramienta.

[2] Prominente científico estadounidense que destacó sobre todo en la rama de informática.

3 FORMULACIONES

3.1 Descripción del problema

Actualmente, los grafos son una metodología, una forma de resolver problemas, la cual su uso ha aumentado considerablemente estos últimos años, sobre todo debido a las tecnologías de la comunicación las cuales han necesitado de una innovación/alternativa a la hora de resolver los problemas.

En este Proyecto, trabajaremos con la formulación propuesta por R.k. Martin la cual también trabaja en grafos y consigue resolver problemas en los que trataremos de unir una red de nodos a través de una elección de las aristas pertenecientes a la misma red evitando cualquier tipo de ciclo dentro de la misma. Como hemos explicado antes, es preferible evitar los ciclos para que no existan ineficiencias. Esta formulación la explicaremos a continuación, pero antes explicaremos también los dos tipos de problemas con los que trabajaremos, MST y Steiner.

Todas las soluciones de este trabajo se llevarán a cabo usando las herramientas de C++ para poder crear modelos matemáticos de forma automática y resolverlos así en LINGO, ya que por el tiempo que tardaríamos en solucionarlo, resultaría imposible resolverlos a mano. Aún así, explicaremos ejemplos a mano de la formulación MST y de la formulación Steiner para poder comprenderlos mejor. Los resultados los analizaremos teniendo en cuenta los óptimos que se hayan sacado de otras formulaciones y del tiempo de ejecución.

3.2 Formulación MST

La formulación MST (Minimum Spanning Tree), será una de las dos formulaciones que usaremos para resolver los problemas propuestos. Esta forma de resolver problemas, trata de que en la solución final unamos todos los nodos de nuestro problema logrando un árbol conexo que es una forma que como hemos visto antes no contiene ningún ciclo. Además, tal y como hemos comentado antes, también conseguiremos que nuestra solución final tenga el mínimo coste.

En caso de empate en el uso de esta metodología (obtengamos dos soluciones que lleguen al óptimo) podremos adoptar como solución final cualquiera de esos árboles que contengan ese mínimo coste. Por ejemplo, en el caso en el cual todas las aristas tengan el mismo peso, podremos encontrar una gran variedad de soluciones óptimas de las cuales solo cogeremos una.

A continuación, pondremos un ejemplo gráfico a continuación para que se vea más fácilmente el resultado al usar la formulación MST.

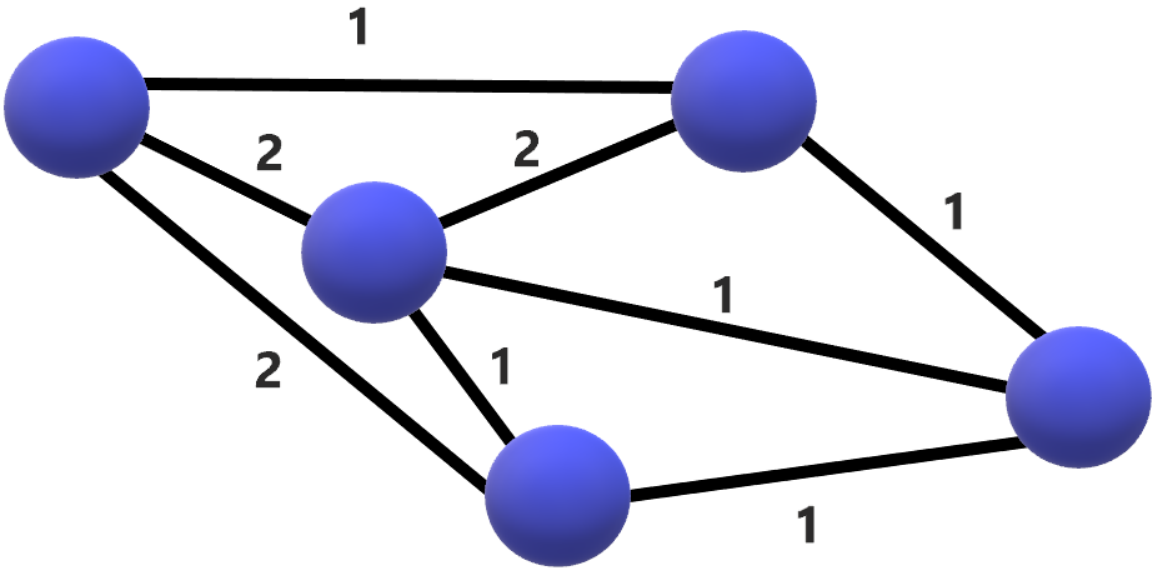


Figura 13 Red de nodos terminales

Fuente: Elaboración propia

En el grafo propuesto, presentamos una red de cinco nodos donde todos son terminales y por lo tanto habrá que unirlos a todos, ya que estamos ante un problema MST. Como también se intenta que sea a mínimo coste, intentaremos coger las aristas de coste 1 si es posible ya que son las que minimizan más este.

Por lo tanto, sabiendo esto, una posible solución del problema sería la siguiente:

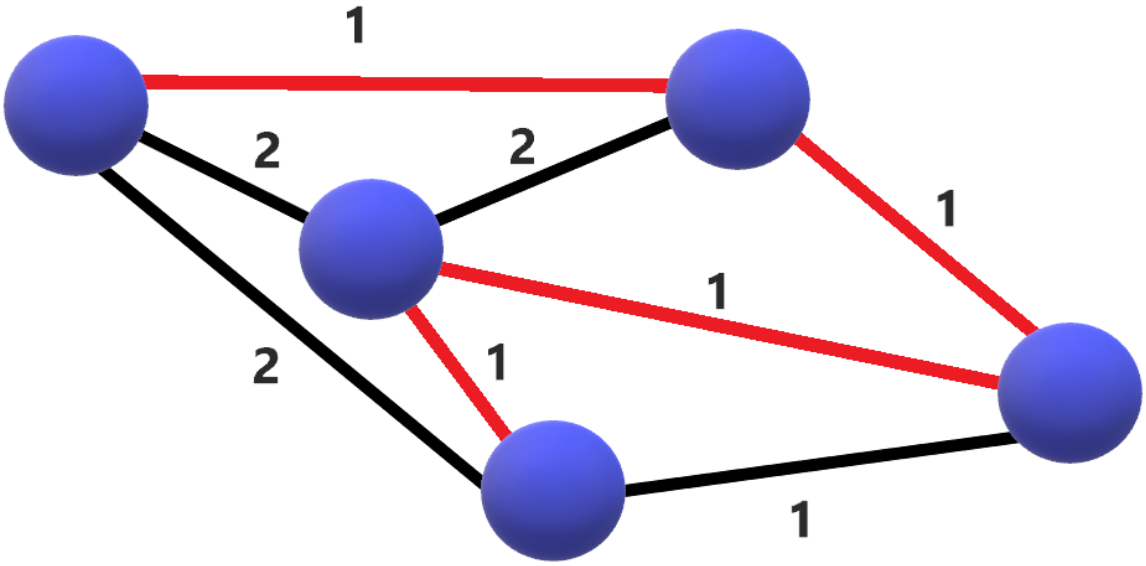


Figura 14 Solución del problema MST

Fuente: Elaboración propia

Se puede apreciar como realmente hemos obtenido el óptimo del problema. También como comentamos antes, podemos ver como no es el único óptimo posible en este problema ya que podríamos haber cogido perfectamente la otra arista de coste 1.

En resumen, esta será la forma en que solucionemos los problemas de tipo MST, uniendo todos los nodos y a coste mínimo.

3.3 Problema de Steiner en grafos

Jakob Steiner fue un matemático suizo del Siglo XIX y uno de los más importantes geómetras de la historia. Esta representación de problemas adoptó este nombre en honor a él. La idea de esta representación es encontrar la ruta mínima entre dos nodos. Esta ruta mínima puede ser análogamente la ruta de más bajo coste, la ruta más rápida etcétera.

Para el problema es necesario tener un grafo no dirigido G=(V,E) para el cual definiremos un conjunto N de nodos terminales N∈V y otro conjunto de nodos Steiner S∈V los cuales sirven de conexión entre los nodos terminales.

La idea de este tipo de problema es conectar todos los nodos terminales a mínimo costo, pudiendo adoptar en la solución final tantos nodos Steiner como sea necesario para alcazar el óptimo. Generalmente, ya que cada arista supone un coste, cuantos menos nodos Steiner haya en nuestra solución final, menor será el coste final, pero no tiene por qué ser siempre así.

A continuación veremos la representación de un problema donde los nodos rojos son los nodos terminales que hay que unir y los azules los steiner que podremos unir o no dependiendo de si nos hacen falta.

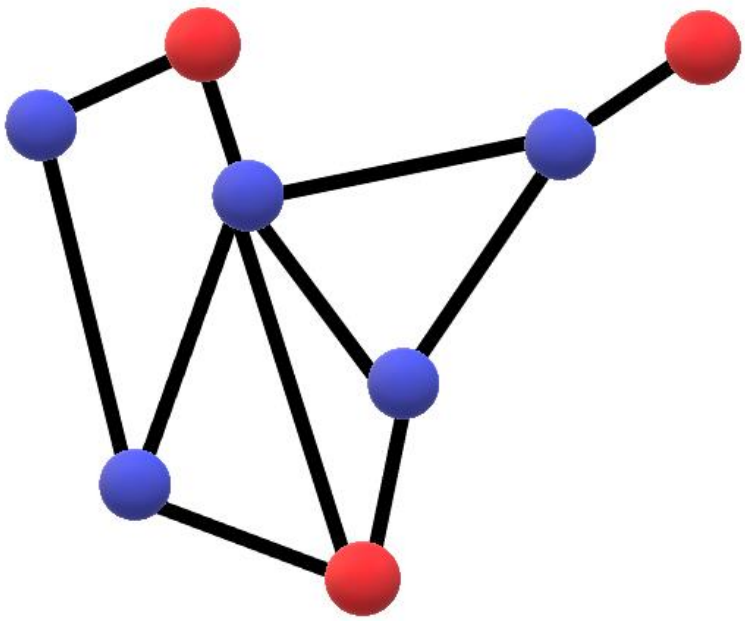


Figura 15 Grafo problema de Steiner

Fuente: Elaboración propia

Como podemos observar, tenemos distintas formas de unir todos los nodos terminales. Hay que decir que tal como hemos explicado anteriormente, la solución óptima probablemente adoptará una estructura de árbol por lo que será algo a tener en cuenta. A continuación, mostraré una solución válida de las muchas que hay que se ha elegido para este ejemplo.

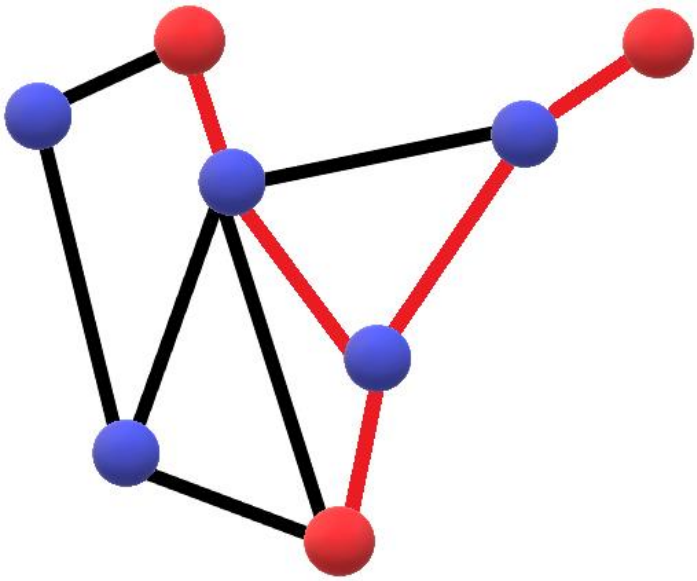


Figura 16 Solución grafo problema de Steiner

Fuente: Elaboración propia

Como Podemos observar, tenemos una solución factible, que no quiere decir que sea la óptima, puesto que hemos unido todos los nodos terminales y además hemos adoptado una estructura de árbol. Esto será algo a tener en cuenta en este TFG.

3.4 Explicación del modelo matemático de R.k. Martin

Este modelo fue propuesto por R.K. Martin y tiene como finalidad resolver problemas de grafos evitando cualquier tipo de ciclo, ya que como hemos comentado antes los ciclos hacen que el modelo sea ineficiente. Por lo que la solución de este modelo será un árbol. Tal como está puesto el modelo, resolvería el problema uniendo todos los nodos con una estructura de árbol, es decir, como si todos los nodos fueran terminales. Añadiendo unas restricciones, podemos hacer fácilmente que el problema pase a ser Steiner y que ya solo tenga que unir un número limitado de nodos terminales.

Por lo tanto, primero explicaremos el modelo abstracto y posteriormente comprobaremos si es válido o no, y en el caso de que lo sea veremos su eficiencia.

$$\min_{w,z} \sum_{e \in E} c_e w_e \tag{1.1a}$$

s.t

$$\sum_{a \in \delta^-(j)} z_a^u \leq 1, \qquad u, j \in N : i \neq u \tag{1.8a}$$

$$\sum_{a \in \delta^-(u)} z_a^u \leq 0, \qquad u \in N \tag{1.8b}$$

$$z_{ij}^u + z_{ji}^u = w_e, \qquad e = \{i, j\} \in E \tag{1.8c}$$

$$\sum_{e \in E} w_e = n - 1 \tag{1.1c}$$

$$w_e \in \{0, 1\}, \qquad e \in E \tag{1.1d}$$

$$z_a^u \geq 0, \qquad u \in N, a \in A \tag{1.8d}$$

Figura 17 Modelo matemático de R.K. Martin

Fuente: Models and methods for Traffic Engineering problems with single-path routing. Tesis de Martim Joyce-Moniz.

3.4.1 Explicación de las variables y atributos del problema

Antes de explicar el modelo, me gustaría aclarar varios conceptos. Primeramente, explicaré que quiere expresar cada variable/atributo.

- Ce: Es un valor predefinido de la cantidad de flujo que pasa por la arista e, es decir, su coste.
- n: Es el número de nodos que habrá en nuestra solución final.
- We: Es una variable binaria que nos indica que la arista ij ha sido cogida para el modelo (We=1) o no (We = 0)
- $Z_{i\,j}^u$: Es una variable binaria también que en este caso indica cuando su valor es 1, que el vértice k está en el lado del vértice j.
- Z_a^u : Es una variable binaria análoga a la anterior que hace referencia igualmente a los arcos.
- N es el conjunto nodos, E es el conjunto aristas y A es el conjunto arcos.

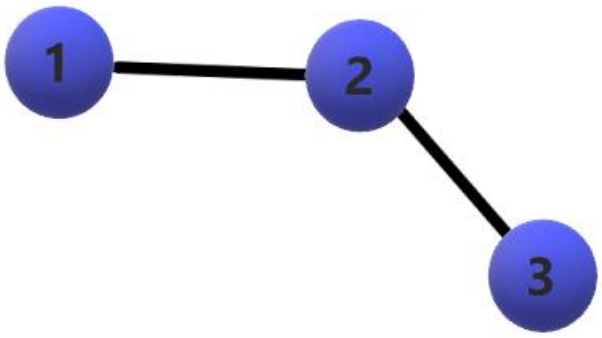


Figura 18 Explicación de la variable Z

Fuente: Elaboración propia

Por ejemplo, en este caso, podemos apreciar como las aristas W_{e1}=1 y W_{e2}=1 (e1 es la arista 1-2 y e2 es la arista 2-3) y existe un pequeño árbol por lo que podría ser una solución válida. En este caso veremos que valores toman las variables $Z^3_{1\,2}$ y $Z^3_{2\,1}$.

$Z^3_{2\,1}$:0 // El vértice 3 debe estar unido al subíndice que colocaremos más a la derecha. Como 3 no está unido por el vértice 1, su valor es 0.

$Z^3_{1\,2}$::1// El vértice 3 si está unido en este caso por el lado del vértice 2 por lo que este si coge el valor 1.

3.4.2 Explicación de las restricciones

Ahora procederé a explicar las restricciones del modelo matemático:

- La función objetivo tratará de minimizar el flujo, el cual se calcula como un sumatorio de todas las aristas We por el flujo de cada una (Ce) el cual está predefinido. Estas aristas que entraran en el sumatorio son obviamente las pertenecientes a la red E que es con la que estamos operando. Por lo tanto, en la solución final se intentarán coger aristas que tengan poco flujo.

- La restricción 1.1c nos indica que los nodos o aristas que adoptemos en nuestra solución deberán ser un número inferior a las aristas que entren en la misma. Esto es debido a que, en un árbol, para unir todos los nodos, necesitamos solo y exclusivamente una arista menos que los nodos totales.

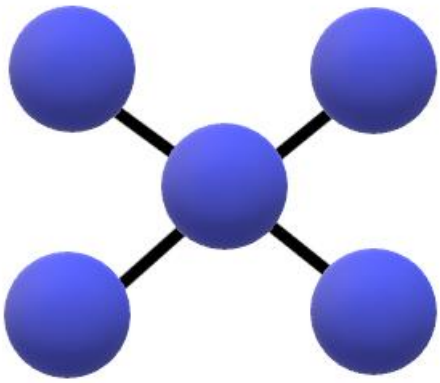


Figura 19 Explicación gráfica de la restricción 1.1c

Fuente: Elaboración propia

Como podemos apreciar, realmente en un grafo de 5 nodos solo necesitamos 4 aristas para poder unirlos a todos.

- La restricción 1.8c expresa que para toda arista We cogida en nuestra solución, debe tener una de las dos siguientes variables (Z_{ij}^{u} o Z_{ji}^{u}) el valor 1 obligatoriamente para todos los nodos pertenecientes a la red N que son los que queremos incluir en nuestra solución. Esto es debido a que si tuviesen valor 0 las dos variables Z es porque hay un nodo u que no está conectado y debería estarlo y de la misma manera, si las dos toman el valor 1 es porque hay un error ya que no es necesario coger dos mismos arcos a la vez. Por lo tanto, para incluir todos los nodos pertenecientes a N, decimos que uno de esos dos valores debe ser 1.

- La restricción 1.8a nos indica que a un solo nodo 'u' solo se puede llegar como mucho mediante un arco cuando fijamos otro nodo cualquiera. En un ejemplo generalizador, en el caso (Z_a^u) fijaremos el nodo u y uno de los dos extremos de a (en este caso el inicial) y diremos que ese sumatorio de arcos deberá ser menor que 1. También, los nodos que fijaremos no pueden ser iguales.

- La restricción 1.8b nos expresa que cuando fijamos un nodo 'u' en un arco cualquiera Z_a^u , si fijamos el nodo inicial del arco i como el mismo que 'u', este valor será <=0, ya que si fuese mayor existiría algún ciclo porque se podría llegar al nodo en cuestión por dos caminos distintos.

Como la restricción 1.8a es un poco más difícil de comprender, la explicare también gráficamente a continuación.

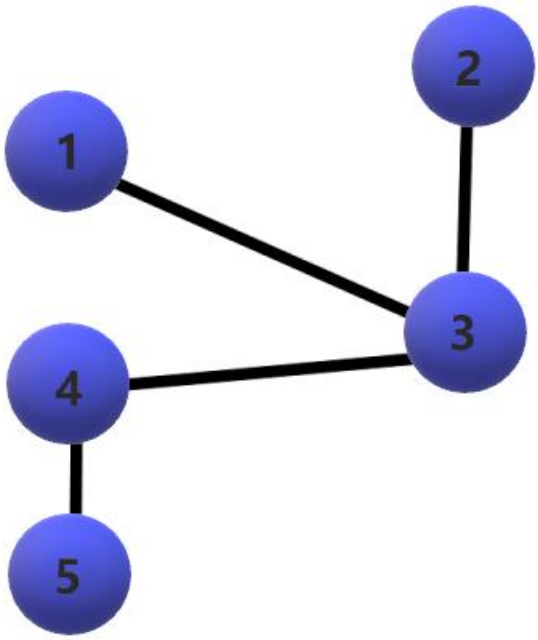


Figura 20 Explicación de la restricción 1.8a.

Fuente: Elaboración propia

Mirando este caso en el cual he adoptado una posible solución valida para un problema de 5 nodos, intentaremos explicar el sentido de la restricción 1.8a.

Por ejemplo, en este caso, fijaremos el nodo 'u' de Z_a^u como el nodo 3. Por otro lado, fijaremos el nodo 5 como el nodo inicial. El sumatorio de los arcos $Z_{5\,x}^3$ donde x representa los números 1,2 y 4 deberá ser menor o igual que 1. En efecto, el valor del sumatorio sería 1 ya que el único que se activa es el arco $Z_{5\,4}^3$.

Por otro lado, vamos a comprobar que pasaría si cogemos como nodo inicial el nodo 2 fijando como nodo 'u' el nodo 3 de nuevo. En este caso, el sumatorio de los arcos $Z_{2\,x}^3$ donde x es 1,4 y 5 es 0. Por lo tanto, el modelo no necesariamente tiene que dar 1 en todo sumatorio de la restricción 1.8a, ya que como acabamos de ver, ha dado 0 en este caso y tenemos una estructura en forma de árbol conexo correcta. Por ello pondremos que deberá ser <=1.

3.5 Otras formulaciones

En este apartado explicaremos distintas formulaciones que existen para resolver el mismo problema. Posteriormente, compararemos los resultados obtenidos de estos modelos con la formulación de R.k. Martin.

3.5.1 Formulación de Miller, Tucker y Zennit

Para explicar este modelo, primero explicaremos primero las variables del mismo:

- Llamaremos 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".

Posteriormente, la variable de decisión:

- αj: Es la única variable de decisión que tendremos en este modelo, tendrá un valor binario, valdrá uno cuando se seleccione ese arco como parte de la solución y cero cuando no.

Por último, explicaremos las restricciones del mismo:

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

Esta primera restricción sirve para asegurar que cada nodo que no sea raíz tiene nodo padre, asegurando que es el nodo destino de algún otro nodo

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

La segunda restricción obliga a todos los nodos raíz ser padre de al menos un nodo.

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

Con esta tercera restricción, imponemos que el nivel del nodo padre debe ser siempre mayor que el nodo hijo para así evitar bucles.

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

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.

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

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. Esto es debido a que, si ponemos todos los nodos en fila, no podríamos superar nunca ese nivel.

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

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, siendo αj nuestra variable de decisión.

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

3.5.2 Formulación de Derroche Laporte

Esta formulación solo tiene un pequeño cambio respecto a la formulación anterior y es la tercera restricción, que es la referida a los niveles y a los ciclos. Esta se sustituye por estas otras dos restricciones:

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

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

Básicamente imponen lo mismo, que el nivel del nodo padre debe ser mayor al nivel del nodo hijo.

Estas dos formulaciones anteriores las compararemos con la formulación de R.k Martin para ver así la eficiencia de ambas.

3.5.3 Estrategia de flujo multiple

Aquí vamos a describir la estrategia de flujo multiple, que es otra forma distinta de resolver este tipo de problemas.

Primeramente vamos a ver las variables de este problema, que son las siguientes:

- N representa el conjunto de nodos.
- A es el conjunto de arcos.
- Ti es una variable binaria que vale 1 si el nodo es terminal.
- R es una variable que diferencia al nodo raíz.
- Cij es el coste de cada arco.

A continuación, veremos las variables de decision.

- αij sería 1 si el arco en cuestión se selecciona y vale 0 si no.

Por ultimo explicaremos las restricciones:

1) $\sum Xik = \sum Xji - 1\ jk\ ,\forall i / i \neq r, Ti = 1$
2) $\sum Xki = \sum Xij\ jk\ ,\forall i / Ti = 0$
3) $Xij \leq CS * \alpha ij\ ,\forall (i,j) \in A$
4) $Xji \leq CS * \alpha ij\ ,\forall (j,i) \in A$
5) $\sum Xrj\ j = CS$
6) $(\sum Ti) - 1 = CS\ N\ i{=}0\ ,\forall i / i \in N$
7) $\alpha ij \in [0,1]\ ,\forall (i,j) \in A$

La primera restricción explica que para cualquier nodo terminal, el flujo que sale debe ser el que entra menos una unidad, es decir, cada nodo terminal se quedará con una unidad de flujo.

La segunda indica que en los nodos steiner el flujo que entra es el mismo que el que sale.

La tercera y la cuarta restringen el flujo que puede pasar por un determinado arco a una cota superior.

Esto es para que no pase más flujo por cualquier arco del que ha enviado el nodo raíz.

La quinta fija la cota superior como el flujo que manda el nodo raíz.

La sexta define la cota superior como el número de nodos menos uno.

La séptima define a la variable alfa como binaria.

Por ultimo la función objetivo es la siguiente:

$$\sum \boldsymbol{Cij} * \boldsymbol{\alpha ij}\ (\boldsymbol{i},\boldsymbol{j}) \in \boldsymbol{A}$$

Lo que hace es minimizar el coste total de transporter una unidad de flujo por el camino obtenido como solución.

3.5.4 Estrategia de flujo simple

La variables del problema y variables de flujo serían las mismas que en la estrategia que el flujo multiple. Las diferencias se pueden apreciar en las restricciones que serían las siguientes:

1) $\sum (xij\ k - xji\ k\ j{:}(i,j) \in A\ ) = 0\ ,\forall j \neq \{k,r\},\forall k \neq r/Tk = 1$

2) $\sum xrj\ k\ j{:}(r,j) \in A = 1\ ,\forall j = r/j \neq k,\forall k \neq r/Tk = 1$

3) $xij\ k \leq \alpha ij\ ,\forall (i,j),\forall k \neq r/Tk = 1$

4) $xji\ k \leq \alpha ij\ ,\forall (i,j),\forall k \neq r/Tk = 1$

5) $xij\ k \in \{0,1\}\ ,\forall (i,k)/Tk = 1,\forall j \in N$

6) $\alpha ij \in [0,1]\ ,\forall (i,j) \in A$

La primera restricción vuelve a indicar lo mismo que en el caso múltiple, el flujo total que llega desde todos los nodos conectados con I pasando por k debe ser el mismo que sale desde I hacia el resto de los nodos conectados, pasando siempre por k. Lo que llega es igual a lo que sale.

La segunda refleja que cada nodo raíz debe mandar una unidad de flujo a cada nodo terminal.

La tercera y la cuarta expresan que el flujo que pasa por cada arco debe ser menor o igual a uno.

La quinta nos indica que el flujo que pasa por cada arco debe estar entre cero y uno.

La sexta fija el alfa como una variable binaria.

Estas dos formulaciones de flujo solo las nombraremos pero no trabajaremos con ella en la parte de análisis de resultados, ya que sus resultados se comprobaron en el pasado y no fueron muy buenos.

4 LINGO

4.1 Introducción a LINGO

Lingo (LINear Generalize Optimizer) es una herramienta simple que nos ayuda a resolver problemas lineales y no lineales y analizar la solución. Además, nos ayuda a encontrar el óptimo del problema encontrando la ganancia más alta o el costo más bajo entre otros.

Una de las muchas ventajas de LINGO además de su simplicidad, es su semejanza con el lenguaje matemático. Aprender LINGO no es difícil ya que el parecido del lenguaje hace que sea muy fácil hacer conversiones del modelo matemático LINGO y viceversa.

4.2 Lenguaje de LINGO

4.2.1 Formato básico

En este punto vamos a ver la forma en la que expresaremos las distintas partes del modelo matemático en LINGO. Para empezar veremos el formato básico con el que trabajaremos que consta de 5 fases que describiremos a continuación:

1.-ENCABEZADO: Todos los ejecutables en LINGO tendrán este mismo comienzo. Debajo del encabezado pondremos los sets y data que explicaremos más adelante

Formato: MODEL:

2.- FUNCIÓN OBJETIVO: Será la función que querremos optimizar.

Formato: (MAX/MIN) = x1+2*x2+3*x3;

3.-RESTRICCIONES: En este apartado escribiremos tantas restricciones como veamos necesarias

Formato: x1+x2<=1;

4.-RESTRICCIONES DE TIPO DE VARIABLES: Aquí declararemos las variables y diremos de que tipo son.

Por defecto, todas las variables son continuas por lo que si queremos convertirlas en enteras binarias o libres deberemos realizar una notación especial.

Variables enteras: @GIN(x1);

Variables binarias: @BIN(x2);

Variables libres: @FREE(x3);

Variables comprendidas: @BND(i,X4,s); restringe la variable a valores comprendidos entre i y s.

5.- FIN: Todo modelo al igual que empieza con "MODEL:" debe acabar con "END"

Formato: END

4.2.2 SETS y DATA

Como hemos comentado antes, debajo de MODEL: hay que definir los SETS y DATA de nuestro problema.

Básicamente en SETS definiremos los nombres y el número de elementos de los conjuntos, además de los atributos y de las variables que formarán parte de cada conjunto.

Por otro lado en DATA definiremos los valores de esos atributos. Además de todo esto, será muy importante terminar la sección de SETS con un ENDSETS y la sección DATA con un ENDDATA.

Ahora vamos a definir los SETS y el DATA de un modelo muy simple en el que tenemos varios sacos de piezas y nuestro atributo es el número de piezas de cada saco.

SETS:

PIEZAS/1..5/: p;

ENDSETS

DATA

P=4 5 3 8 5:

ENDDATA

Como vemos es bastante simple de usar. Además podemos hacer conjuntos de una forma análoga, lo que hace que aumente la versatilidad de esta herramienta.

4.2.3 Especificaciones

Para escribir las especificaciones hay dos funciones que hay que tener en cuenta ya que serán usadas en prácticamente todos los modelos. Estas son @FOR y @SUM y se explicarán a continuación.

$\forall i$: Este operador que expresa "para todo i" usado en la mayor parte de los modelos matemáticos tiene su analogía en LINGO como @FOR. Si por ejemplo en el apartado anterior, PIEZAS tomara el índice i, la restricción empezaría de esta forma: "@FOR(PIEZAS(i):[3]".

$\sum pi$: Este operador es también muy usado y expresa sumatorio. En el caso de LINGO tiene su analogía en @SUM. Para expresar por ejemplo un sumatorio de todas las piezas i lo haríamos de la siguiente forma: "@SUM(piezas(i):". Como podemos observar son muy fáciles de usar y se usan de una forma bastante parecida.

[3] Hay que tener en cuenta que el paréntesis que se ha abierto antes de la palabra PIEZAS hay que cerrarlo en algún momento.

4.2.4 Formatos de signo para expresiones condicionales

A la hora de escribir restricciones en los modelos matemáticos, estas suelen tener condiciones. En el caso de LINGO estas condiciones tienen una forma un tanto especial de escribirse. En LINGO las restricciones son las que van precedidas dentro de una restriccion por la barra vertical "|".

El formato de signo que utiliza para estas expresiones condiciones es:

#EQ# → =

#NE# → ≠

#GE# → ≥

#GT# → >

#LT# → <

#LE# → ≤

#AND# → y

#OR# → o

Por ultimo hay que tener en cuenta que @FOR puede contener multiples expresiones separadas por ";" mientras que @SUM solo admite una sola expresión.

4.3 Interfaz de usuario

En este apartado veremos la interfaz de usuario del programa LINGO para comprender mejor lo comentado anteriormente. Para ello, resolveremos el problema de la mochila por su sencillez.

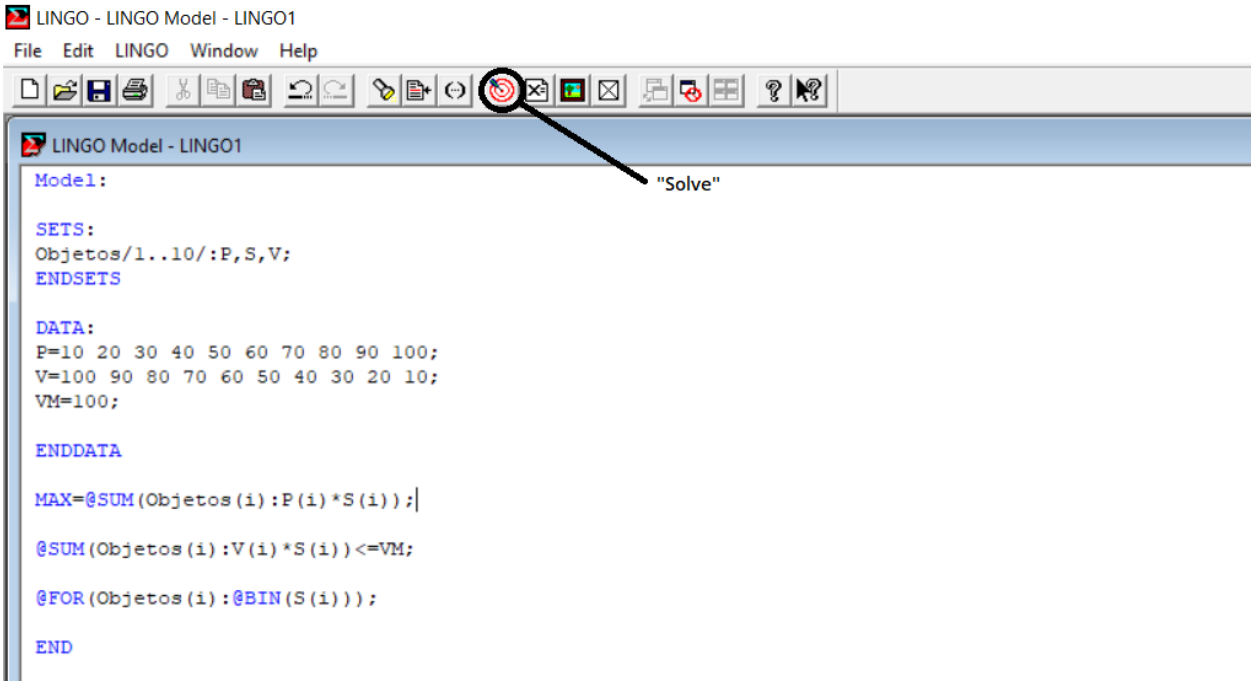


Figura 21 Problema de la mochila en LINGO

Fuente: Elaboración propia (LINGO)

Como podemos observar, nuestro modelo cumple con todo lo mencionado anteriormente. Todo modelo comienza con un "MODEL:" y seguidamente describe los SETS donde defininimos el nombre (Objetos) y el número de elementos del conjunto (10). Además, se definen los atributos (P Peso y V Volumen) y las variables (en este caso S Selección es la única variable que se define). Después de terminar los SETS se definen los valores de los atributos en DATA. Además, hemos querido tener en cuenta un valor extra que es el volumen máximo de la mochila (VM).

Una vez terminado de definir el DATA, comenzamos a definir las restricciones empezando por la función objetivo y posteriormente el resto de ellas, terminando por definir el tipo de variables si fuera necesario. En nuestro caso, es necesario definir la variable S como binaria ya que representará si se elige un objeto o no. Por último, solo quedaría resolver el modelo pulsando el botón que está marcado.

Solution Report - LINGO1

```
Global optimal solution found.
Objective value:                              340.0000
Extended solver steps:                             0
Total solver iterations:                           0


                     Variable           Value        Reduced Cost
                           VM        100.0000            0.000000
                         P( 1)       10.00000            0.000000
                         P( 2)       20.00000            0.000000
                         P( 3)       30.00000            0.000000
                         P( 4)       40.00000            0.000000
                         P( 5)       50.00000            0.000000
                         P( 6)       60.00000            0.000000
                         P( 7)       70.00000            0.000000
                         P( 8)       80.00000            0.000000
                         P( 9)       90.00000            0.000000
                        P( 10)       100.0000            0.000000
                         S( 1)       0.000000           -10.00000
                         S( 2)       0.000000           -20.00000
                         S( 3)       0.000000           -30.00000
                         S( 4)       0.000000           -40.00000
                         S( 5)       0.000000           -50.00000
                         S( 6)       0.000000           -60.00000
                         S( 7)       1.000000           -70.00000
                         S( 8)       1.000000           -80.00000
                         S( 9)       1.000000           -90.00000
                        S( 10)       1.000000           -100.0000
                         V( 1)       100.0000            0.000000
                         V( 2)       90.00000            0.000000
                         V( 3)       80.00000            0.000000
                         V( 4)       70.00000            0.000000
                         V( 5)       60.00000            0.000000
                         V( 6)       50.00000            0.000000
                         V( 7)       40.00000            0.000000
                         V( 8)       30.00000            0.000000
                         V( 9)       20.00000            0.000000
                        V( 10)       10.00000            0.000000

                        Row    Slack or Surplus      Dual Price
                          1        340.0000            1.000000
                          2        0.000000            0.000000
```

Figura 22 Solución del problema de la mochila

Fuente: Elaboración propia (LINGO)

Una vez le damos al botón de "Solve" nos saltará esta ventana emergente en la cual LINGO te mostrará la solución encontrada. Se puede observar como las variables S(7), S(8), S(9) y S(10) toman el valor de 1 y es lógico puesto que son los objetos con más peso y menos volumen. Esto hace que la función objetivo alcance el valor de 340 tal como pone en la parte superior de la solución.

Por último, me gustaría comentar la pestaña de display model, que se accede con la siguiente ruta: LINGO → Generate → Display model, o bien pulsando Ctrl+G. Esto mostrará la siguiente pantalla:

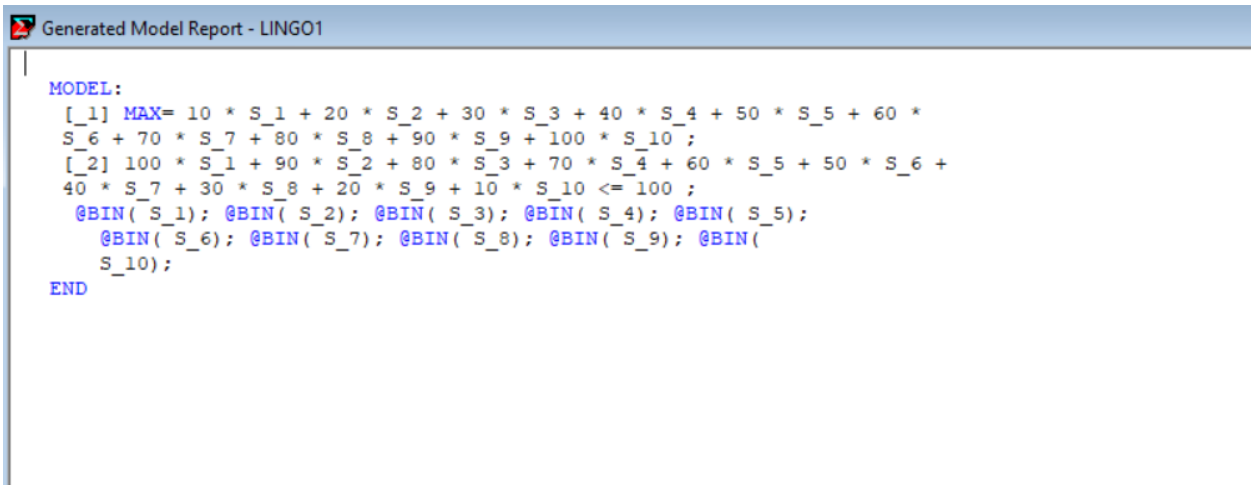


Figura 23 Display model mochila

Fuente: Elaboración propia (LINGO)

Con esto podemos ver qué hace exactamente LINGO y podremos detectar los errores más fácilmente en el caso en el que nuestro programa no funcione. En este caso, podemos ver que cuentas hace para sacar la función objetivo y para cumplir las restricciones. Además, te saca por pantalla todas las variables que participan en tu modelo y el tipo que son.

4.4 Otras consideraciones

Hay detalles que hay que tener en cuenta que no se han nombrado anteriormente y son los siguientes:

1.- Se ignorarán las líneas en blanco

2.- Para comentar una frase y que no se tenga en cuenta a la hora de compilar es necesario predecirla del símbolo "!" y al final poner ";"

3.- No se distingue entre mayúsculas y minúsculas

4.- La longitud máxima de una línea es 512 caracteres

4.5 Resolución del modelo en LINGO

Con lo que ya sabemos sobre LINGO y con el modelo matemático que disponemos, nuestro próximo objetivo será implantarlo en LINGO para ver si funciona. Para resolver este problema, hemos cogido los siguientes datos:

Lingo Model - LINGO resuelto MST4

```
Model:

SETS:
Nodos/1..5/:N;
Aristas/1..10/:X,N1,N2,Coste;
Arcos/1..20/:NO,ND;
Triple(Arcos,Nodos) :Y;
ENDSETS

DATA:
Coste= 1 2 3 4 5 6 7 8 9 10;
N1= 1 1 1 1 2 2 2 3 3 4;
N2= 2 3 4 5 3 4 5 4 5 5;
NO= 1 1 1 1 2 2 2 3 3 4 2 3 4 5 3 4 5 4 5 5;
ND= 2 3 4 5 3 4 5 4 5 5 1 1 1 1 2 2 2 3 3 4;
NUM = 5;
ENDDATA
```

Figura 24 Datos ejemplo LINGO

Fuente: Elaboración propia (LINGO)

Para entender mejor este pequeño ejemplo, he dibujado el grafo el cual quedaría de la siguiente forma:

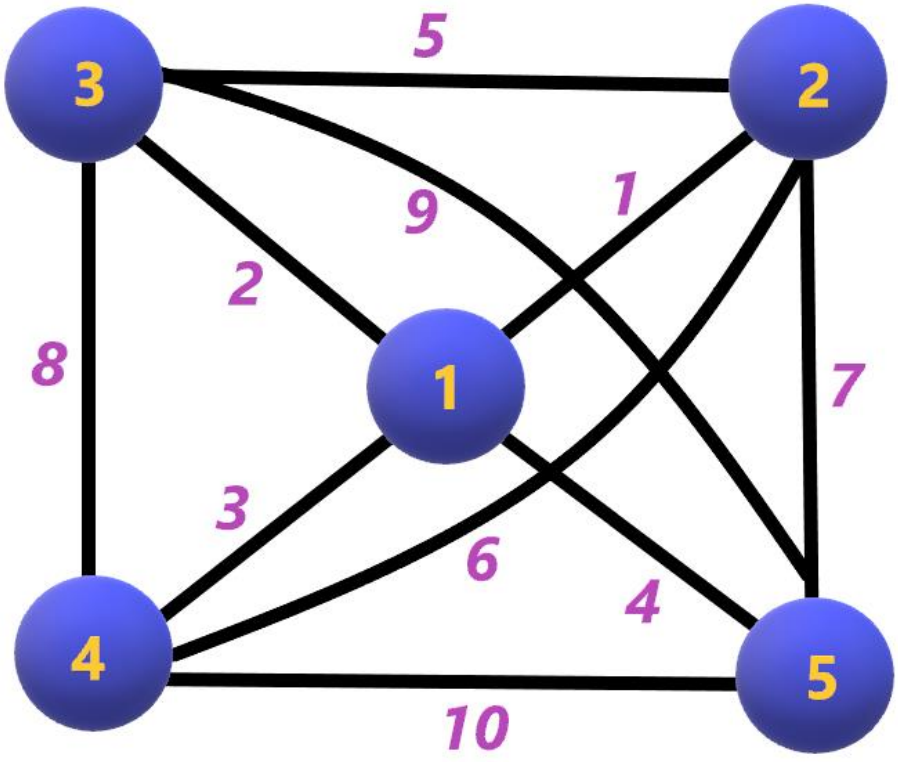


Figura 25 Grafo de los datos del ejemplo

Fuente: Elaboración propia

Como podemos apreciar, tenemos 5 nodos y 10 posibles aristas de las cuales solo podremos coger 4 de ellas. Para este modelo, he escrito el número del nodo en color amarillo en el interior del mismo, y el coste de la arista en un tono violeta al lado de la misma arista. El coste óptimo sería el que saldría de coger las 4 con menos coste, pero no siempre es posible. En este ejemplo, veremos que solución nos sale cuando resolvemos el problema.

Lo que haremos a continuación, será transcribir el modelo matemático a lenguaje LINGO y posteriormente veremos si verdaderamente este modelo funciona. Para ello, solucionaremos el modelo a mano ya que su tamaño es reducido y veremos si las soluciones son las mismas

Una vez escrito el modelo en lenguaje LINGO, quedaría de la siguiente forma:

```
MIN=@SUM(Aristas(z):Coste(z)*X(z));
@SUM(Aristas(z):X(z))=NUM-1;
@FOR(Aristas(z):@FOR(ARCOS(i)|NO(i)#EQ#N1(z) #AND# ND(i)#EQ#N2(z): @FOR(ARCOS(j)|NO(j)#EQ#N2(z)#AND#ND(j)#EQ#N1(z): @FOR(Nodos(k): X(z)=Y(i,k)+ Y(j,k) ))));
@FOR(Nodos(k):@FOR(Nodos(i)|i#NE#k: @SUM(Arcos(j)|NO(j)#EQ#i: y(j,k)) <= 1));
@FOR(Nodos(k): @SUM(Arcos(i)|NO(i)#EQ#k : y(i,k)) <= 0);
@FOR(Aristas(j):@BIn(x(j)));
```

Figura 26 Modelo de R.k. Martin en LINGO

Fuente: Elaboración propia (LINGO)

Este modelo de esa manera compila, así que solo habrá que comprobar si puede también encontrar el óptimo. Para que se pueda ver mejor, escribiré el modelo directamente en el Word también. El modelo que he plasmado en una captura antes es el siguiente:

```
MIN=@SUM(Aristas(z):Coste(z)*X(z));

@SUM(Aristas(z):X(z))=NUM-1;

@FOR(Aristas(z):@FOR(ARCOS(i)|NO(i)#EQ#N1(z) #AND# ND(i)#EQ#N2(z):
@FOR(ARCOS(j)|NO(j)#EQ#N2(z)#AND#ND(j)#EQ#N1(z): @FOR(Nodos(k): X(z)=Y(i,k)+
Y(j,k) ))));

@FOR(Nodos(k):@FOR(Nodos(i)|i#NE#k: @SUM(Arcos(j)|NO(j)#EQ#i: y(j,k)) <= 1));

@FOR(Nodos(k): @SUM(Arcos(i)|NO(i)#EQ#k : y(i,k)) <= 0);

@FOR(Aristas(j):@BIn(x(j)));

END
```

Una vez compilamos el modelo y vemos los resultados vemos que sale lo siguiente:

```
Global optimal solution found.
Objective value:                              10.00000
Extended solver steps:                               0
Total solver iterations:                            13

                     Variable           Value        Reduced Cost
                          NUM        5.000000            0.000000
                         N( 1)       0.000000            0.000000
                         N( 2)       0.000000            0.000000
                         N( 3)       0.000000            0.000000
                         N( 4)       0.000000            0.000000
                         N( 5)       0.000000            0.000000
                         X( 1)       1.000000            1.000000
                         X( 2)       1.000000            2.000000
                         X( 3)       1.000000            3.000000
                         X( 4)       1.000000            4.000000
                         X( 5)       0.000000            5.000000
                         X( 6)       0.000000            6.000000
                         X( 7)       0.000000            7.000000
                         X( 8)       0.000000            8.000000
                         X( 9)       0.000000            9.000000
                        X( 10)       0.000000            10.00000
```

Figura 27 Solución del modelo.

Fuente: Elaboración propia (LINGO)

Como se puede apreciar, este modelo ha encontrado una solución de valor 10 en la cual une el nodo 1 con el resto de nodos por lo que a priori es válida ya que forma un árbol conexo. Se realizan 13 iteraciones, lo que podría ser mejorable para un modelo tan pequeño. Aún así, ha encontrado la solución al instante.

A continuación, veremos si gráficamente podemos encontrar la misma solución que se ha encontrado con estos dos modelos. Para ello cogeremos el árbol que menor función objetivo tenga quedando la siguiente solución:

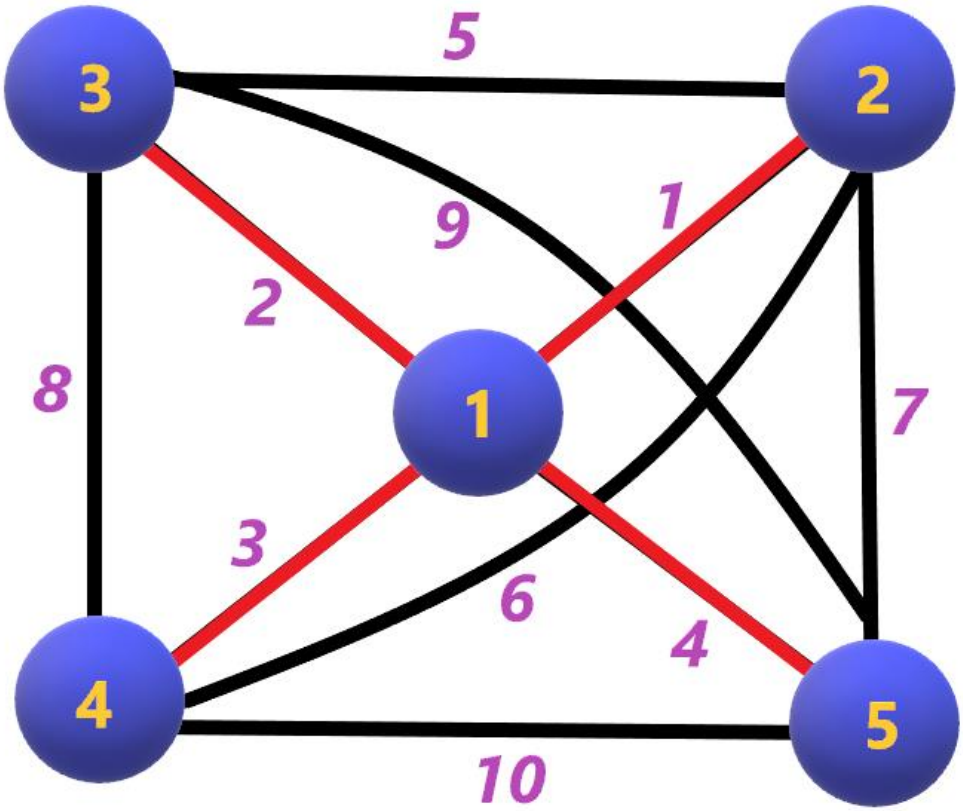


Figura 28 Solución gráfica del modelo

Fuente: Elaboración propia

En efecto, gráficamente podemos observar como en el óptimo se encuentran las 4 aristas que unen al nodo 1 con el resto de nodos y que la suma de sus costes hace un total de 10. No se puede encontrar una solución mejor puesto que son los costes más bajos y además el grafo final está formando un árbol, por lo que podemos afirmar que nuestros modelos han encontrado la solución óptima correctamente.

Por lo tanto, daremos válido el modelo de momento. Este mismo modelo se usará con grafos mucho más grandes y comprobaremos si sigue encontrando la mejor solución.

4.6 Conversión con C a LINGO

Los datos de los problemas que resolveremos proceden de archivos de texto(.txt) que tienen la siguiente estructura:

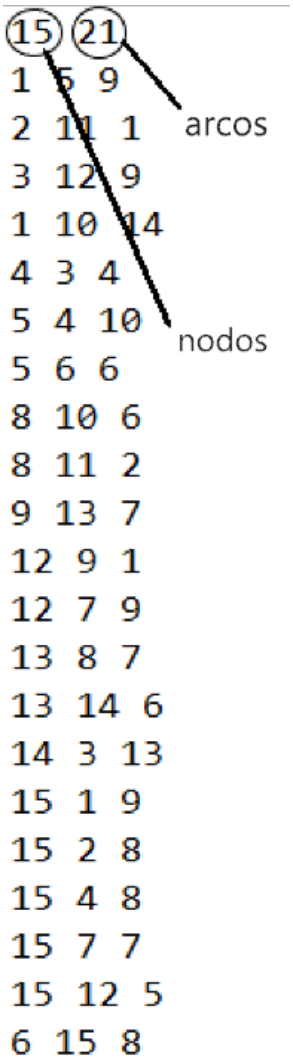


Figura 29 Estructura de los problemas en .txt

Fuente: Elaboración propia

Todos los problemas que resolvamos nos vendrán dados con esa estructura. El primer número que aparece son los nodos que hay. Por lo tanto en las dos primeras columnas veremos todos los números desde el 1 hasta el número de nodos que haya, ya que tiene que haber mínimo una unión disponible por cada uno. El Segundo número, son los arcos disponibles en el problema en cuestión. Tiene relación también con la cantidad de filas que habrá.

La primera columna de los números por debajo del número de nodos y de aristas expresa el nodo de inicio del arco, y la segunda columna el nodo final del arco. Por último, la tercera columna expresa el coste de ese arco en cuestión.

Con esa estructura, trataremos de resolver los problemas que se nos planteen. En principio, este trabajo constará de 38 problemas distintos con la misma estructura.

Para ello, nuestro primer paso será resolverlo teniendo en cuenta que todos los nodos son terminales. Es decir, trataremos de unir todos los nodos al mínimo coste. Para ello, hemos creado un código en C con el que automatizaremos el proceso de leer datos del archivo de texto y convertirlo en formato LINGO. Este código está adjuntado como anexo 1 al final del mismo trabajo. Aquí adjuntaremos a continuación un diagrama de flujo en el que se puede visualizar qué hace el código exactamente.

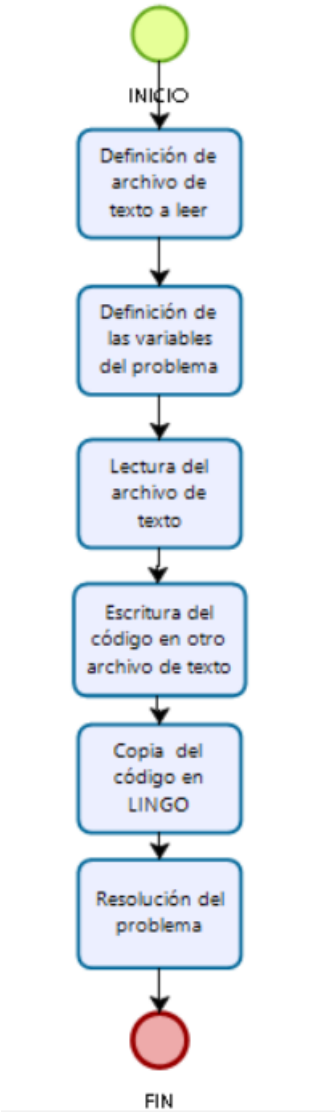


Figura 30 Diagrama de flujo del código en C

Fuente: Elaboración propia (BIZAGI)

Posteriormente, ese código nos crearía un archivo LINGO con la siguiente estructura, listo para ser compilado:

```
MODEL:
SETS:
Nodos/1.. 45/;
Aristas/1.. 87/:X,N1,N2,Coste;
Arcos/1.. 174/:NO,ND;
Triple(Arcos,Nodos) :Y;
ENDSETS
DATA:
Coste= 8 8 8 8 10 3 10 4 5 6 9 7 9 4 1 6 6 3 7 3 4 2 5 8 7 6 2 6 4 7 1 10 1 6
7 10 6 9 6 3 4 10 5 5 7 6 10 1 5 10 4 4 3 2 1 3 9 5 8 7 6 3 2 10 1 7 1 7 2 5
1 6 8 2 6 7 4 9 9 5 10 5 1 6 1 1 6 ;
N1= 1 1 1 2 2 2 2 3 3 3 4 4 4 4 5 5 5 6 7 7 7 8 8 9 10 11 11 12 13 14 15 16
18 20 20 20 20 20 22 22 23 23 24 24 25 26 26 26 26 26 26 27 28 29 29 30 30 30
30 30 31 31 32 34 34 35 35 36 36 36 37 37 37 38 39 39 39 39 40 40 40 43 44 44
44 45 45 ;
N2= 10 25 29 4 9 28 32 22 27 43 6 8 25 32 12 21 25 12 4 6 12 5 16 37 17 3 35
33 24 31 13 14 22 2 11 24 36 42 32 37 3 4 5 44 40 1 10 20 30 33 45 42 19 34
39 5 9 14 19 36 21 39 15 9 33 12 31 9 13 17 28 30 41 27 7 18 23 36 7 17 33 15
11 15 38 4 41 ;
NO= 1 1 1 2 2 2 2 3 3 3 4 4 4 4 5 5 5 6 7 7 7 8 8 9 10 11 11 12 13 14 15 16
18 20 20 20 20 20 22 22 23 23 24 24 25 26 26 26 26 26 26 27 28 29 29 30 30 30
30 30 31 31 32 34 34 35 35 36 36 36 37 37 37 38 39 39 39 39 40 40 40 43 44 44
44 45 45 10 25 29 4 9 28 32 22 27 43 6 8 25 32 12 21 25 12 4 6 12 5 16 37 17
3 35 33 24 31 13 14 22 2 11 24 36 42 32 37 3 4 5 44 40 1 10 20 30 33 45 42 19
34 39 5 9 14 19 36 21 39 15 9 33 12 31 9 13 17 28 30 41 27 7 18 23 36 7 17 33
15 11 15 38 4 41 ;
ND= 10 25 29 4 9 28 32 22 27 43 6 8 25 32 12 21 25 12 4 6 12 5 16 37 17 3 35
33 24 31 13 14 22 2 11 24 36 42 32 37 3 4 5 44 40 1 10 20 30 33 45 42 19 34
39 5 9 14 19 36 21 39 15 9 33 12 31 9 13 17 28 30 41 27 7 18 23 36 7 17 33 15
11 15 38 4 41 1 1 1 2 2 2 2 3 3 3 4 4 4 4 5 5 5 6 7 7 7 8 8 9 10 11 11 12 13
14 15 16 18 20 20 20 20 20 22 22 23 23 24 24 25 26 26 26 26 26 26 27 28 29 29
30 30 30 30 30 31 31 32 34 34 35 35 36 36 36 37 37 37 38 39 39 39 39 40 40 40
43 44 44 44 45 45 ;
NUM = 45 ;
ENDDATA
MIN=@SUM(Aristas(z):Coste(z)*X(z));
@SUM(Aristas(z):X(z))=NUM-1;
@FOR(Aristas(z):@FOR(ARCOS(i)|NO(i)#EQ#N1(z) #AND# ND(i)#EQ#N2(z):
@FOR(ARCOS(j)|NO(j)#EQ#N2(z)#AND#ND(j)#EQ#N1(z): @FOR(Nodos(k): X(z)=Y(i,k)+
Y(j,k) ))));
@FOR(Nodos(k):@FOR(Nodos(i)|i#NE#k: @SUM(Arcos(j)|NO(j)#EQ#i: y(j,k)) <= 1));
@FOR(Nodos(k): @SUM(Arcos(i)|NO(i)#EQ#k : y(i,k))  <= 0);
@FOR(Aristas(j):@BIn(x(j)));

END
```

4.7 Conversión con C a LINGO (Steiner)

Para este apartado, el archivo .txt que se cogerá será el mismo que el de antes. Sin embargo, se leerá de forma distinta. En el archivo .txt anterior no se leía el final del archivo que es lo referido a Steiner. Para ponernos en contexto, pondré un ejemplo a continuación:

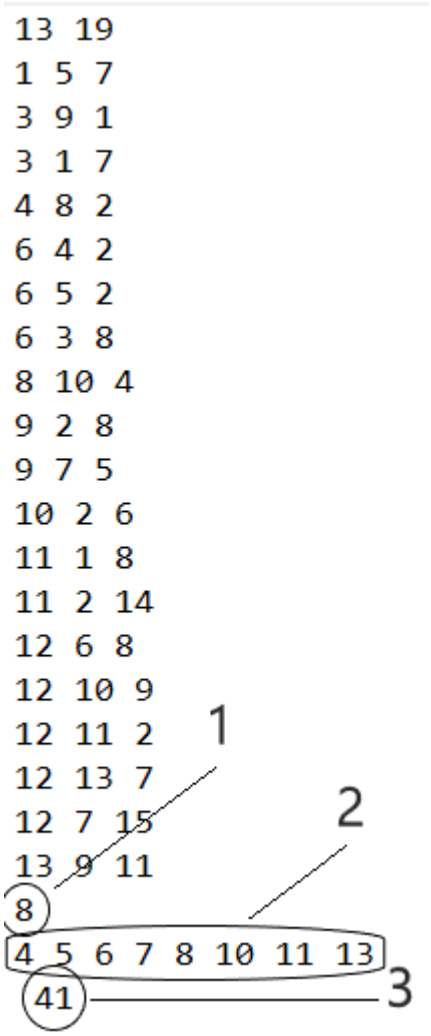


Figura 31 Archivo .txt para problemas Steiner

Fuente: Elaboración propia

Como podemos percibir, el archivo tiene exactamente la misma estructura inicial. Como se explicó antes, el primer número hace referencia al número de nodos, el segundo al número de aristas y luego de 3 en 3 se hace referencia al nodo inicial, nodo final y coste de la arista respectivamente.

La diferencia que existe es que ahora trataremos con más datos que estarán al final del mismo .txt de los cuales antes no se dijo nada. Estos datos los hemos marcado con los números 1, 2 y 3 en la "Figura 31 Archivo .txt para problemas Steiner", los cuales explicaremos a continuación.

1.- El primer dato hace referencia al número de nodos terminales que habrá. Estos serán los nodos que haya que coger obligatoriamente.

2.- Te dice qué nodos son los terminales.

3.- Es un número que hay que sumar a la solución final del modelo.

Con estos nuevos datos, nos dispondremos a resolver el modelo. Para ello crearemos un programa en C que nos automatice el proceso el cual al igual que el anterior, adjuntare al final del trabajo.

Una vez hecho el programa nos saldrán modelos con la siguiente estructura:

```
MODEL:
SETS:
Nodos/1.. 22/:TR,alfa;
Aristas/1..  33/:X,N1,N2,Coste;
Arcos/1.. 66/:NO,ND;
Triple(Arcos,Nodos) :Y;
ENDSETS
DATA:
Coste= 4 1 7 5 7 8 9 10 2 9 7 2 1 9 13 3 9 8 17 15 15 5 7 16 5 10 10 6 4 3 6
6 7;
N1= 1 2 3 5 7 8 9 10 10 11 12 13 13 14 14 15 15 15 3 10 16 17 17 13 18 18 18
19 19 20 20 14 6;
N2= 21 4 22 1 16 12 22 18 19 2 5 8 22 17 10 6 7 14 12 12 9 11 2 4 3 20 9 13
15 4 11 21 1;
NO= 1 2 3 5 7 8 9 10 10 11 12 13 13 14 14 15 15 15 3 10 16 17 17 13 18 18 18
19 19 20 20 14 6 21 4 22 1 16 12 22 18 19 2 5 8 22 17 10 6 7 14 12 12 9 11 2
4 3 20 9 13 15 4 11 21 1;
ND= 21 4 22 1 16 12 22 18 19 2 5 8 22 17 10 6 7 14 12 12 9 11 2 4 3 20 9 13
15 4 11 21 1 1 2 3 5 7 8 9 10 10 11 12 13 13 14 14 15 15 15 3 10 16 17 17 13
18 18 18 19 19 20 20 14 6;
TR= 0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1 0 1 0;
M= 37;
ENDDATA
MIN=@SUM(Aristas(z):Coste(z)*X(z))+M;
@FOR(Nodos(k)|TR(k)#EQ#1: Alfa(k)=1);
@SUM(Nodos(k):alfa(k))=@SUM(aristas(z):X(z))+1;
@FOR(Aristas(z):@FOR(ARCOS(i)|NO(i)#EQ#N1(z) #AND# ND(i)#EQ#N2(z):
@FOR(ARCOS(j)|NO(j)#EQ#N2(z)#AND#ND(j)#EQ#N1(z): @FOR(Nodos(k): X(z)=Y(i,k)+
Y(j,k) ))));
@FOR(Nodos(k):@FOR(Nodos(i)|i#NE#k: @SUM(Arcos(j)|NO(j)#EQ#i: y(j,k)) <= 1));
@FOR(Nodos(k):@SUM(Arcos(i)|NO(i)#EQ#k : y(i,k))  <= 0);
@FOR(Nodos(k)|TR(k)#EQ#0: @FOR(Aristas(z)|k#EQ#N1(z):X(z)<=alfa(k)));
@FOR(Nodos(k)|TR(k)#EQ#0: @FOR(Aristas(z)|k#EQ#N2(z):X(z)<=alfa(k)));
@FOR(Aristas(j):@BIn(x(j)));
@FOR(Nodos(k):@BIN(alfa(k)));

END
```

Se puede apreciar que la estructura ha cambiado notablemente, y es que se han añadido más datos y restricciones los cuales explicaremos a continuación.

Se crea el atributo del nodo terminal como TR, que es un nodo el cual tiene que pertenecer al modelo obligatoriamente. Por ello mismo valdrá 1 si es terminal y 0 si no.

También se crea la variable alfa que es 1 si el nodo pertenece a la red de nodos de la solución y 0 si no.

Por ultimo, antes de pasar a las restricciones, también se añade el valor M que es un valor que hay que sumar a la solución que nos de el problema para llegar a la verdadera solución.

Ya dentro de las restricciones los cambios que hay son los siguientes:

1.- Se añade el valor M sumando en la función objetivo.

2.- Como ahora no participan en la solución todos los nodos, el sumatorio de los alfa será igual al número de aristas+1. Recordamos que para unir todos los nodos necesitamos una arista menos que el total de nodos.

3.- En las dos siguientes restricciones que volveré a copiar a continuación se consigue que si cogemos una arista, automáticamente tenemos que coger si o sí los dos nodos de los extremos de la misma.

```
@FOR(Nodos(k)|TR(k)#EQ#0: @FOR(Aristas(z)|k#EQ#N1(z):X(z)<=alfa(k)));
@FOR(Nodos(k)|TR(k)#EQ#0: @FOR(Aristas(z)|k#EQ#N2(z):X(z)<=alfa(k)));
```

Esto se consigue diciendo que alfa tiene que ser mayor o igual, debido a que si la arista vale 1, entonces alfa obligatoriamente debe tomar el valor de 1 también.

4.- Por ultimo, se especifica que alfa tiene que ser una variable binaria también.

El diagrama de flujo que explica el proceso es similar al diagrama de flujo expuesto anteriormente. Como bien hemos comentado anteriormente, solo varía en unas restricciones que hay que añadir al modelo.

5 EXPERIMENTACIÓN

5.1 Batería de problemas

Antes de ver y analizar los resultados, pondremos una tabla en la que estarán recogidos el número de nodos y el número de aristas de cada problema. La tabla es la siguiente:

Tabla 3 Tabla informativa de nodos y aristas de los problemas

	NODOS	ARISTAS
Steinb1	13	19
Steinb2	15	21
Steinb3	20	25
Steinb4	40	80
Steinb5	39	80
Steinb6	45	87
Steinb7	22	33
Steinb8	26	38
Steinb9	27	35
Steinb10	55	121
Steinb11	63	129
Steinb12	63	125
Steinb13	36	56
Steinb14	42	65
Steinb15	48	69
Steinb16	77	166
Steinb17	74	153
Steinb18	82	166

steinc1	143	260
steinc2	128	234
steinc3	178	295
steinc4	193	314
steinc5	223	341
steinc6	366	837
steinc7	383	866
steinc8	387	867
steinc9	418	903
steinc10	427	891

Fuente: Elaboración propia

5.2 Resultados de los problemas con soluciones continuas y MST

Primeramente, resolveremos los mismos problemas propuestos anteriormente de forma continua. Esta forma de resolver problemas suele tardar mucho menos tiempo que la forma entera. Además, es probable que los resultados sean menores en el caso de minimizar.

Para crear los archivos LINGO de la resolución de problemas de forma continua tan solo hay que hacer un pequeño cambio respecto al modelo anterior y es quitar la última restricción, que es la siguiente:

```
@FOR(Aristas(j):@BIn(x(j)));
```

Esta restricción hace que todas las variables X(j) que son las aristas sean binarias y al quitarlas le damos la libertad que necesitan para ser continuas. Esto también es equivalente a decir que toda arista, es menor o igual a 1 $X(j) \leq 1 \; \forall j$.

Sabiendo este pequeño cambio que hay que hacer, la tabla de resultados de la resolución continua sería la siguiente:

Tabla 4 Resultados de la resolución del problema MST de forma continua

	F.O.	TIEMPO
Steinb1	53	0:00:00
Steinb2	77	0:00:00
Steinb3	66	0:00:00
Steinb4	131	0:00:00
Steinb5	116	0:00:00
Steinb6	145	0:00:00
Steinb7	108	0:00:00
Steinb8	110	0:00:00
Steinb9	79	0:00:00
Steinb10	184	0:00:00
Steinb11	188	0:00:00
Steinb12	194	0:00:02
Steinb13	178	0:00:01
Steinb14	178	0:00:01
Steinb15	232	0:00:01
Steinb16	248	0:00:04
Steinb17	205	0:00:03
Steinb18	252	0:00:05
steinc1	666	0:00:10
steinc2	683	0:00:07
steinc3	827	0:00:15
steinc4	930	0:00:18
steinc5	1044	0:00:29
steinc6	1156	0:06:19
steinc7	1247	0:08:20

steinc8	1182	0:07:47
steinc9	1293	0:09:55
steinc10	1363	0:10:34

Fuente: Elaboración propia

En este caso, solo hemos tomado datos en el resultado del tiempo y de la función objetivo obtenida. Como se puede apreciar, el tiempo es mucho menor comparado con las soluciones de la forma entera. Esto es debido a la forma de solucionar el problema.

Cuando el problema se resuelve de forma entera; lo resuelve con un Branch & Bound, mientras que este lo resuelve con uno de los siguientes tres métodos: Primal, dual o barrier, los cuales son más rápidos en completar una iteración. Mientras estaba solucionando los problemas y probando, me di cuenta de que el método barrier era el más rápido de los 3 por lo que es el método que se eligió para resolver los problemas de esta forma continua.

Conseguimos resolver todos los problemas como mucho en 10 minutos, llegando al óptimo en todos, incluso en los últimos (en rojo) los cuales ni pudimos solucionar en la forma entera que veremos a continuación ya que requerían demasiado tiempo.

Viendo estos resultados que a priori parecen ser bastante buenos, se decidió intentar resolver los problemas descartados anteriormente, del “steinc11” hasta el “steinc20” pero dio error. El problema se hace demasiado grande como para poder resolverlo por LINGO.

A continuación, propondremos una gráfica en la que compararemos los óptimos obtenidos en continua con los óptimos enteros de los cuales ya disponemos de ellos.

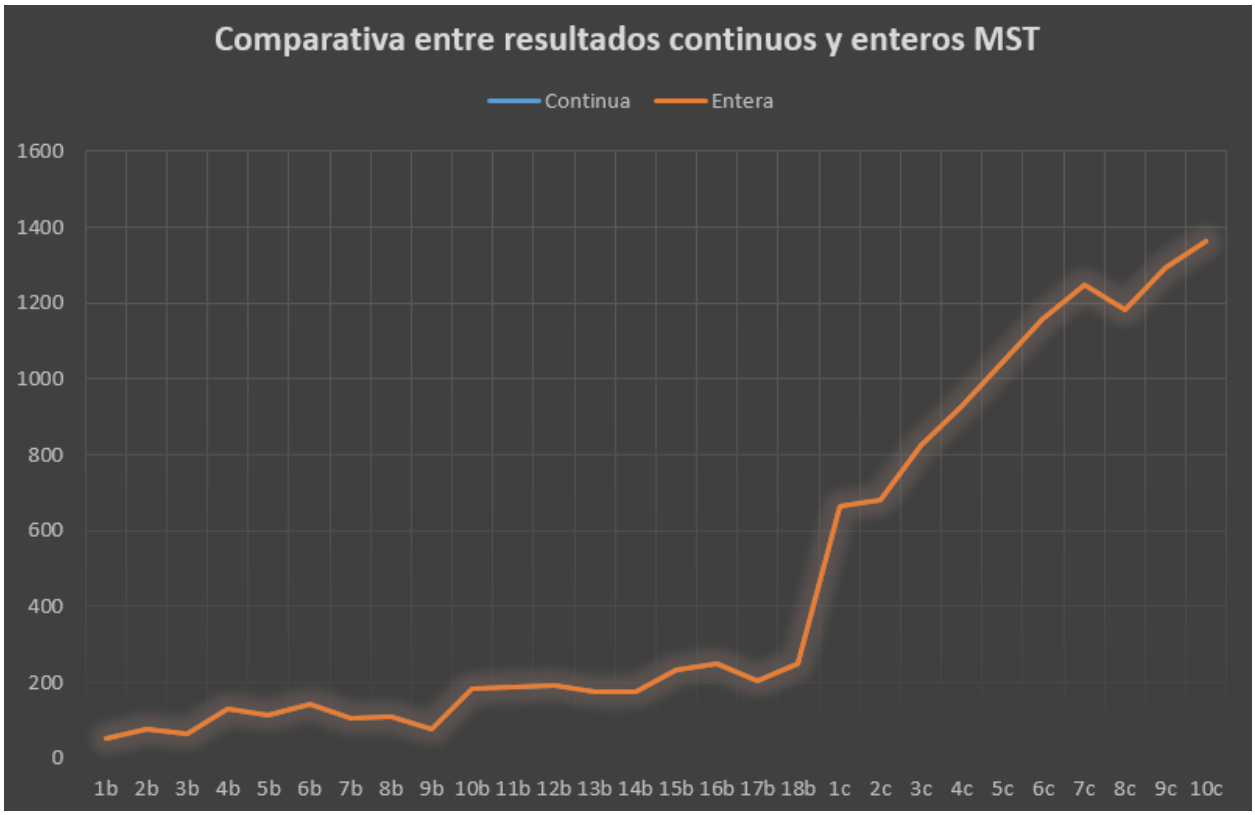


Figura 32 Comparativa entre resultados continuos y enteros MST

Fuente: Elaboración propia

En la gráfica anterior, podemos apreciar como la aproximación por la forma continua no podía ser mejor. En la forma MST, llega al óptimo en todo momento lo que es muy bueno. Es por ello que no se ve la línea azul, porque la solapa completamente.

5.3 Resultados de los problemas con soluciones entera y MST

Una vez resuelto el modelo LINGO del problema en cuestión sacaremos y adjuntaremos los datos para posteriormente analizarlos. En este caso el código es exactamente el mismo que el adjuntado anteriormente.

Este proceso se llevaría a cabo con los 38 distintos modelos que tenemos de los cuales ya tenemos sus óptimos. Una vez resueltos los 38 primeros ejercicios y recopilado los resultados, obtenemos la siguiente tabla de información, en la que comprobaremos si realmente la formulación alcanza el óptimo o no.

Tabla 5 Resultados de la resolución del problema con todos los nodos terminales

	TIEMPO	F.O.	Nº restricciones	Nº variables	Iteraciones	Memoria usada
Steinb1	0:00:00	53	418	513	310	114
Steinb2	0:00:00	77	542	651	219	142
Steinb3	0:00:00	66	902	1025	678	218
Steinb4	0:00:04	131	4802	6480	9908	1230
Steinb5	0:00:03	116	4643	6320	9047	1198
Steinb6	0:00:03	145	5942	7917	8496	1501
Steinb7	0:00:00	108	1212	1485	884	302
Steinb8	0:00:00	110	1666	2014	1263	406
Steinb9	0:00:00	79	1676	1925	1577	393
Steinb10	0:00:15	184	9682	13431	24226	2490
Steinb11	0:00:26	188	12098	16383	32866	3049
Steinb12	0:00:11	194	11846	15875	21038	2964
Steinb13	0:00:00	178	3314	4088	3319	801
Steinb14	0:00:01	178	4496	5525	5090	1076
Steinb15	0:00:02	232	5618	6693	6578	1310
Steinb16	0:01:29	248	18713	25730	61564	4747
Steinb17	0:01:20	205	16800	22797	53096	4222
Steinb18	0:02:28	252	20338	27390	80205	5078
steinc1	0:20:01	666	57631	74620	220756	13885
steinc2	0:11:44	683	46338	60138	177644	11196
steinc3	0:45:21	827	84196	105315	312419	19770
steinc4	0:52:34	930	97853	121518	360838	22858
steinc5	2:09:13	1044	125774	152427	453244	28867

Fuente: Elaboración propia

En efecto, la formulación consigue alcanzar el óptimo en cada problema.

Para crear esta tabla, hemos agrupado los distintos problemas en filas y los datos en columnas. Hemos tenido en cuenta el tiempo de realización del ejercicio, que será mejor cuanto menor tiempo tarde, la función objetivo que trataremos de minimizar, el número de restricciones, el número de variables, las iteraciones que también es deseable que sea un número de tamaño reducido y la memoria usada.

A la hora de ver los resultados, podemos ver como estos problemas están ordenados por complejidad, siendo el más fácil de resolver el primero y el más complejo el último. También se puede apreciar como en el problema 5 comienza ya a crecer exponencialmente a tiempos superiores a 2 horas, algo que veremos reflejado posteriormente.

Seguidamente, se intentó resolver más problemas de los cuales su resultado fue el siguiente:

Tabla 6 Problemas no resueltos modelo entero

	TIEMPO	F.O.	Nº restricciones	Nº variables	Iteraciones	Memoria usada
steinc6	16:14:54	1729	440300	613521	504848	110968
steinc7	1:00:52	1771	478369	664222	202846	120242
steinc8	1:03:10	1669	485300	671925	193292	121735
steinc9	1:28:22	1997	552180	755811	235940	137364
steinc10	1:08:38	1986	562788	761805	217763	138908
steinc11	0:05:27	0	1249498	2002995	0	RAN OUT
steinc12	0:05:57	0	1279438	2062935	0	RAN OUT
steinc13	0:05:49	0	1256954	2019922	0	RAN OUT
steinc14	0:05:49	0	1231035	1966032	0	RAN OUT
steinc15	0:04:54	0	1157502	1816815	0	RAN OUT
steinc16	0:00:17	0	1136003	3545137	0	RAN OUT
steinc17	0:00:17	0	1093003	3490705	0	RAN OUT
steinc18	0:00:16	0	1150503	3523969	0	RAN OUT
steinc19	0:02:33	0	1811031	3378817	0	RAN OUT
steinc20	0:00:00	0	0	100000	0	RAN OUT

Fuente: Elaboración propia

El problema steinc6 vuelve a aumentar la complejidad del mismo añadiendo más restricciones y variables y se puede ver como ni en 16 horas se consigue resolver. A partir de este hecho, se opta por dejar los problemas resolviéndose como mucho 1 hora.

Por otro lado, a partir del steinc11, en el cual vuelve a aumentar la complejidad, ni siquiera LINGO es capaz de intentar resolverlo ya que el mismo programa falla debido a lo grande que es dicho problema. Esto era algo que suponíamos ya que en la forma continua nos dio fallos también. En el último caso (steinc20), directamente da error al segundo 0. Por esto, solo nos centraremos a la hora de analizar los resultados en los problemas que se han podido resolver o que no se han resuelto por falta de tiempo (sin haber dado error) y quitaremos de nuestro análisis los problemas a partir del steinc11, ya que ni hemos podido operar con ellos.

Por último, cabe destacar que, aunque los tiempos sean muy altos, se consigue llegar al mismo óptimo en los dos modelos. Por lo tanto, se puede ver que la forma continua, que es la "lower bound" encuentra soluciones bastante buenas.

5.4 Resultados de los problemas Steiner (continua)

Aquí evaluaremos la forma continua de la formulación en el problema steiner. Recordamos que la forma continua lo que busca es la lower bound, que es la cota inferior y se suele usar como punto de partida.

Para conseguir transformar el problema en un problema steiner, es necesario quitar las siguientes dos restricciones:

```
@FOR(Aristas(j):@BIn(x(j)));
@FOR(Nodos(k):@BIN(alfa(k)));
```

Estas dos restricciones hacen que las variables alfa y x solo puedan adaptar valores binarios por lo que si las quitamos, ampliaremos el arco de soluciones además de que aumentará la velocidad de búsqueda. Una vez resuelto todos los problemas, agrupamos los datos en una tabla similar al modelo continuo anterior donde solo detallábamos el tiempo y el valor de la función objetivo.

Tabla 7 Resultados modelo Steiner continuo

	F.O.	TIEMPO
Steinb1	79,5	0:00:00
Steinb2	83	0:00:00
Steinb3	138	0:00:00
Steinb4	53,7	0:00:00
Steinb5	50,5	0:00:00
Steinb6	119,024	0:00:01
Steinb7	111	0:00:00
Steinb8	98,375	0:00:00

Steinb9	220	0:00:00
Steinb10	80,25	0:00:01
Steinb11	81,7778	0:00:03
Steinb12	167,679	0:00:06
Steinb13	159,4	0:00:00
Steinb14	230	0:00:00
Steinb15	309,5	0:00:01
Steinb16	108,568	0:00:11
Steinb17	122,929	0:00:08
Steinb18	200	0:00:28
steinc1	76,4667	0:01:01
steinc2	115,833	0:01:05
steinc3	727,241	0:08:19
steinc4	-	1:04:14
steinc5	-	1:31:42

Fuente: Elaboración propia

Los problemas en general se resuelven bastante rápido como era de esperar, aunque a partir del "steinc4" estos se complican y tardan más de 1 hora en resolverse.

Haciendo una gráfica comparativa viendo los resultados obtenidos con la formulación continua con los óptimos que recordamos que disponemos de ellos desde el principio, obtendríamos lo siguiente:

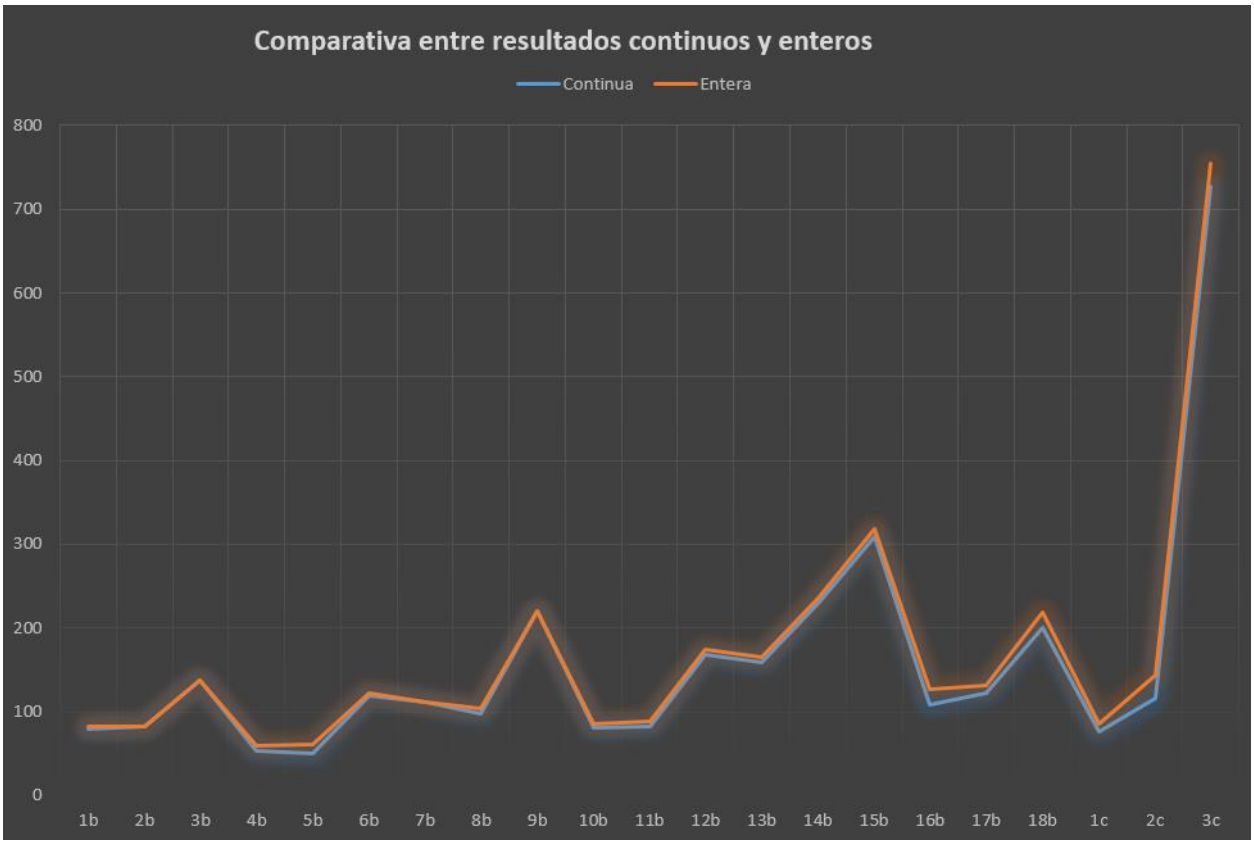


Figura 33 Gráfica comparativa entre resultados continuos y enteros en Steiner

Fuente: Elaboración propia

Como podemos apreciar, la formulación continua a pesar de no alcanzar el óptimo en la mayoría de problemas, se acerca muchísimo. Aquí se puede ver mejor como la "lower bound" que alcanza esta formulación es realmente buena.

5.5 Resultados de los problemas Steiner

Igual que en la tabla que analizaba el MST, volvemos a tomar nota de las variables, restricciones, iteraciones y memoria usada además de la función objetivo y del tiempo que tarda.

Tabla 8 Resultados modelo Steiner entero

	TIEMPO	F.O.	Nº restricciones	Nº variables	Iteraciones	Memoria usada
Steinb1	0:00:01	82	436	518	526	119
Steinb2	0:00:00	83	557	655	397	147
Steinb3	0:00:00	138	917	1030	888	225
Steinb4	0:00:45	59	4934	6511	89770	1251

Steinb5	0:00:26	61	4763	6347	28610	1218
Steinb6	0:00:12	122	6033	7937	13826	1513
Steinb7	0:00:00	111	1248	1496	565	310
Steinb8	0:00:00	104	1703	2025	1101	416
Steinb9	0:00:00	220	1690	1929	933	401
Steinb10	0:02:26	86	9868	13473	468689	2517
Steinb11	0:03:05	88	12287	16427	873907	3076
Steinb12	0:02:54	174	11958	15902	322291	2984
Steinb13	0:00:13	165	3392	4110	13476	817
Steinb14	0:00:44	235	4570	5546	61451	1092
Steinb15	0:00:23	318	5653	6703	48072	1323
Steinb16	1:04:02	172	18982	25790	7446644	4784
Steinb17	0:10:02	131	17026	22848	862188	4255
Steinb18	0:08:53	218	20499	27427	507083	5106
steinc1	0:11:28	85	58137	74758	3105105	13956
steinc2	1:00:54	153	46775	60256	11271224	11258
steinc3	3:26:33	813	84567	105418	2847760	19834
steinc4	1:01:10	1104	98196	121609	1722308	22922
steinc5	1:16:00	-	125924	152470	1004186	28919
steinc6	7:48:12	-	441947	613882	1606965	111206
steinc7	1:00:05	-	480057	664595	127241	120488
steinc8	3:48:46	-	486719	672233	837572	121956
steinc9	1:09:39	-	553501	756105	424823	137582
steinc10	1:05:55	-	563622	761990	371762	139080

Fuente: Elaboración propia

Podemos ver que el modelo es correcto ya que ecuentra el óptimo. Sin embargo sigue siendo lento. Era de esperar pues este tendría características parecidas al modelo entero.

En rojo están los problemas de los cuales no hemos podido hayar el óptimo por falta de tiempo ya que hemos puesto 1 hora como límite. En los primeros problemas sin solucionar al menos encontro una solución del problema. En los últimos se puede ver que aunque hayan pasado casi 8 horas, no consigue encontrar nada.

En este caso se intenta resolver hasta el steinc10 sin suerte, ya que el único que consigue resolver en menos de una hora es el primero.

También se puede ver que aunque las cotas inferiores no sean todas las mismas que la solución como en el caso anterior de MST, son bastante parecidas y es algo bueno también.

5.6 Comparación de resultados de la forma continua MST con la forma continua de Miller Tucker y Zemlin y con la forma de Desroche y Laporte

Evaluando los problemas hasta el problema 10c, establaceremos una comparación entre las soluciones en continua de mi modelo con el modelo de Miller Tucker y Zemlin y también con el modelo de Derroche Laporte.

Tal y como comentamos antes, en la formulación continua veremos cómo de bueno es el modelo fijándonos en la "lower bound" o cota inferior. Lo normal es que la cota inferior sea una solución mejor que la óptima en el modelo entero, por lo tanto cuanto más cerca se encuentre de ese óptimo, mejor será.

Tabla 9 Comparación de la cota inferior de los modelos presentados MST

	ÓPTIMO	R.K. MARTIN	TIEMPO R.k. Martin	M. T. Z.	D.L.
Steinb1	53	53	0:00:00	41,333	53
Steinb2	77	77	0:00:00	60,429	77
Steinb3	66	66	0:00:00	63,000	66
Steinb4	131	131	0:00:00	109,667	128
Steinb5	116	116	0:00:00	93,947	115
Steinb6	145	145	0:00:00	118,682	144
Steinb7	108	108	0:00:00	85,048	106
Steinb8	110	110	0:00:00	88,040	105
Steinb9	79	79	0:00:00	65,769	79
Steinb10	184	184	0:00:00	156,481	184

Steinb11	188	188	0:00:00	154,581	184
Steinb12	194	194	0:00:02	156,839	194
Steinb13	178	178	0:00:01	137,543	177
Steinb14	178	178	0:00:01	140,268	178
Steinb15	232	232	0:00:01	194,149	231
Steinb16	248	248	0:00:04	209,447	248
Steinb17	205	205	0:00:03	177,548	205
Steinb18	252	252	0:00:05	214,321	249
steinc1	666	666	0:00:10	541,127	682
steinc2	683	683	0:00:07	578,787	682
steinc3	827	827	0:00:15	625,492	821
steinc4	930	930	0:00:18	759,083	928
steinc5	1044	1044	0:00:29	859,865	1035
steinc6	1156	1156	0:06:19	973,597	1153
steinc7	1247	1247	0:08:20	1071,460	1247
steinc8	1182	1182	0:07:47	983,648	1180
steinc9	1293	1293	0:09:55	1057,670	1293
steinc10	1363	1363	0:10:34	1139,620	1363

Fuente: Elaboración propia

En esta tabla podemos ver la comparación entre los resultados de los modelos nombrados anteriormente. Por un lado, podemos apreciar como los tiempos del modelo de R.k. Martin comienzan a crecer exponencialmente mientras los otros, aunque no estén en las tablas, son menores de cinco segundos todos.

Por otro lado, los resultados del modelo de R.k. Martin en la forma continua MST son iguales que el óptimo, por lo que es algo muy bueno. El modelo de Derroche Laporte también tiene una muy buena cota inferior pero no siempre llega al óptimo.

5.7 Comparación de resultados de la forma continua steiner con la forma continua de Miller, Tucker y Zemlin y con la forma de Desrocher y Laporte

En este apartado haremos una comparación similar al apartado anterior pero con las soluciones continuas en la forma Steiner. Por lo tanto la tabla de comparaciones quedaría de la siguiente forma:

Tabla 10 Comparación de la cota inferior de los modelos presentados Steiner

	ÓPTIMO	R.K. MARTIN	TIEMPO R.k. Martin	M. T. Z.	D.L.
Steinb1	82	79,5	0:00:00	70,083	77,75
Steinb2	83	83	0:00:00	76,714	82,5
Steinb3	138	138	0:00:00	133,105	138
Steinb4	59	53,7	0:00:00	41,410	50,5
Steinb5	61	50,5	0:00:00	43,395	52,167
Steinb6	122	119,024	0:00:01	93,409	107,333
Steinb7	111	111	0:00:00	99,286	105
Steinb8	104	98,375	0:00:00	81,640	92,75
Steinb9	220	220	0:00:00	203,692	217,5
Steinb10	86	80,25	0:00:01	64,148	70,333
Steinb11	88	81,7778	0:00:03	67,290	79
Steinb12	174	167,679	0:00:06	137,516	159,333
Steinb13	165	159,4	0:00:00	127,786	145,75
Steinb14	235	230	0:00:00	186,951	211,917
Steinb15	318	309,5	0:00:01	283,021	313,5
Steinb16	127	108,568	0:00:11	95,263	109,833
Steinb17	131	122,929	0:00:08	111,123	119,333
Steinb18	218	200	0:00:28	179,833	204,25
steinc1	85	76,4667	0:01:01	48,049	52

steinc2	144	115,833	0:01:05	69,189	81,5
steinc3	754	727,241	0:08:19	559,006	678,833

Fuente: Elaboración propia

En esta tabla solo hemos llegado hasta el problema steinc3, ya que no pudimos resolver los siguientes problemas.

Viendo de nuevo las soluciones, vuelve a ocurrir algo parecido. Con respecto al tiempo, este vuelve a crecer exponencialmente mientras en los otros modelos sigue siendo inferior a 5 segundos.

Por otro lado, aunque en este caso las soluciones del modelo Steiner en continua no coincidan con las óptimas en Steiner entero, presenta cotas inferiores mejores que las de los otros dos modelos.

6 CONCLUSIONES

Con respecto al modelo, podemos asegurar que el modelo es eficaz ya que alcanza el óptimo en los problemas que resuelve. Sin embargo, no es nada eficiente. Tarda demasiado, hasta más de dieciséis horas en resolver modelos en los que el número de aristas y de nodos es superior a 300 cada uno y hace que se creen demasiadas variables y restricciones. Eso ha hecho que, a la hora de ver los resultados, el trabajo se hiciera lento, ya que la mayoría de los problemas tardaban más de una hora en resolverse.

Sin embargo, hemos comprobado que la formulación de R.k. Martin no se hizo para optimizar el tiempo sino para optimizar la cota inferior del problema MST. Como hemos podido comprobar, al resolver el problema MST que es de tipo P de forma continua, alcanzábamos el mismo óptimo que si lo resolvíamos de la forma entera en cualquier problema. Esto es lo que más destacamos de su formulación.

Esto mismo no pasó cuando extrapolamos la formulación al problema de Steiner que es NP. En este problema la cota inferior no era el mismo óptimo que te daba cuando resolvías el problema de forma entera. Sin embargo, si es verdad que el acercamiento al óptimo fue mayor que los otros dos modelos.

En general, este trabajo creo que se ha adecuado a mis conocimientos adquiridos en la carrera. Es cierto que para la programación en C y en LINGO tuve que aprender más, pero tuve la capacidad de hacerlo sin problema. También ha sido un trabajo desde mi punto de vista entretenido. Siempre me había gustado trabajar con modelos matemáticos y por eso mismo se me ha hecho más ameno.

7 BIBLIOGRAFÍA

1. Estudio Experimental de Formulaciones de Flujo para Resolver el Problema de Steiner en Grafo
   Trabajo Fin de Grado
   Universidad de Sevilla
   Autor: Antonio Garcia Elías
   Tutor: José Manuel García Sánchez
   Año: 2017
2. Problemas de métodos de optimización
   Apuntes de la asignatura de métodos de optimización
   Autor: Jose Manuel García Sánchez
   Año: Sin Especificar
3. C/C ++: CURSO DE PROGRAMACION 2015 (MANUAL IMPRESCINDIBLE)
   Libro de programación en C
   Autor: Miguel Ángel Acera García
   Año: 2015
4. Ficheros de texto en C
   Apuntes de apoyo para la realización del programa en C
   URL: http://www.chuidiang.org/clinux/ficheros/fichero-texto.php
5. Models and methods for Traffic Engineering problems with single-path routing
   Tesis doctoral
   Université Libre de Bruselles
   Autor: Martim Joyce-Moniz
   Año:2016
6. Problemas MST y Steiner en txt
   Ficheros txt
   Autor: Jose Manuel García Sánchez
   Año: Sin especificar
7. Diseño de redes de comunicaciones confiables. El problema de Steiner generalizado.
   Centro de cálculo – Instituto de computación, facultad de Ingeniería, Universidad de la República de Uruguay.
   Autor: Sergio Nesmachnow
   Año:2002
8. OR-Library
   Distributing Test Problems by Electronic Mail J. E. Beasley: The Journal of the Operational Research Society

8 ANEXOS

En estos anexos ilustraremos los códigos en C que hemos usado para automatizar el proceso de conversión de C a LINGO

Anexo 1: Código en C para la automatización del problema MST

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

int main()
{
     FILE *leer;  //Declara el la variable tipo fichero f
    leer= fopen("steinb7mmm.txt","r"); // almacena f.txt en la variable f
    if(leer==NULL)//si no se puede abrir, print no se puede
    {
        printf("no se puede abrir el fichero\n\n");
    }
    else// se se abre correctamente print bien
    {
        printf("Fichero abierto correctamente\n\n");
    }


    int z;
    int i;
    int nodos;
    int arcos;
    int cont=0;
    int arcos2;

    fscanf(leer,"%d",&nodos);
    fscanf(leer,"%d",&arcos);
    arcos2=arcos*2;
    VECTOR_INT nodi= DIM_VECTOR_INT(arcos);
    VECTOR_INT nodf= DIM_VECTOR_INT(arcos);
    VECTOR_INT peso= DIM_VECTOR_INT(arcos);

    for (i=0;i<arcos;i++)
    {
        fscanf(leer,"%d",&z);
        nodi[i]=z;
        fscanf(leer,"%d",&z);
        nodf[i]=z;
        fscanf(leer,"%d",&z);
        peso[i]=z;
    }
    print_vector(nodi,arcos);
```

```
    print_vector(nodi,arcos);
    print_vector(nodf,arcos);
    print_vector(peso,arcos);

    char hh1[] = "MODEL:";
    char hh2[] = "\nSETS:";
    char hh3[] = "\nNodos/1..";
    char hh3a[] = "/;\nAristas/1..";
    char hh3b[] = "/:X,N1,N2,Coste;\nArcos/1..";
    char hh3c[] = "/:NO,ND;\nTriple(Arcos,Nodos) :Y;\nENDSETS\nDATA:\nCoste=";
    char hh4[] = ";\nN1=";
    char hh5[] = ";\nN2=";
    char hh6[] = ";\nNO=";
    char hh7[] = ";\nND=";
    char hh8[] = ";\nNUM =";
    char hh9[] = ";\nENDDATA\nMIN=@SUM(Aristas(z):Coste(z)*X(z));\n@SUM(Aristas(z):X(z))=NUM-1;\n(

   FILE * fichero;
fichero = fopen( "problema1.txt" , "w" );


if(fichero== NULL)
{
    printf("\nERROR");
}
else
    {
    printf("\nBIEN LEIDO");
    }
fwrite(hh1 , 1 , sizeof(hh1) , fichero);
fwrite(hh2 , 1 , sizeof(hh2) , fichero);
fwrite(hh3 , 1 , sizeof(hh3) , fichero);
fprintf(fichero,"%d",nodos);
fwrite(hh3a , 1 , sizeof(hh3a) , fichero);
fprintf(fichero,"%d",arcos);
fwrite(hh3b , 1 , sizeof(hh3b) , fichero);
fprintf(fichero,"%d",arcos2);
fwrite(hh3c , 1 , sizeof(hh3c) , fichero);
for (i = 0; i < arcos; i++)
{
    fprintf (fichero, "%d ", peso[i]);
}
```

```
fwrite(hh4 , 1 , sizeof(hh4) , fichero);

for (i = 0; i < arcos; i++)
{
    fprintf (fichero, "%d ", nodi[i]);
}

fwrite(hh5 , 1 , sizeof(hh5) , fichero);

for (i = 0; i < arcos; i++)
{
    fprintf (fichero, "%d ", nodf[i]);
}

fwrite(hh6 , 1 , sizeof(hh6) , fichero);

for (i = 0; i < arcos; i++)
{
    fprintf (fichero, "%d ", nodi[i]);
}
for (i = 0; i < arcos; i++)
{
    fprintf (fichero, "%d ", nodf[i]);
}


fwrite(hh7 , 1 , sizeof(hh7) , fichero);

for (i = 0; i < arcos; i++)
{
    fprintf (fichero, "%d ", nodf[i]);
}
for (i = 0; i < arcos; i++)
{
    fprintf (fichero, "%d ", nodi[i]);
}

fwrite(hh8 , 1 , sizeof(hh8) , fichero);

fprintf (fichero, "%d ", nodos);

fwrite(hh9 , 1 , sizeof(hh9) , fichero);
```

```
    fclose(fichero);


        return 0;
    }
```

Anexo 2: Código en C para la automatización del problema Steiner

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

int main()
{
     FILE *leer;  //Declara el la variable tipo fichero f
    leer= fopen("steinb5.txt","r"); // almacena f.txt en
    if(leer==NULL)//si no se puede abrir, print no se pue
    {
        printf("no se puede abrir el fichero\n\n");
    }
    else// se se abre correctamente print bien
    {
        printf("Fichero abierto correctamente\n\n");
    }


    int z;
    int i;
    int nodos;
    int arcos;
    int cont=0;
    int arcos2;
    int SUMA;
    int n;
    int p;

    fscanf(leer,"%d",&nodos);
    fscanf(leer,"%d",&arcos);
    arcos2=arcos*2;
    VECTOR_INT nodi= DIM_VECTOR_INT(arcos);
    VECTOR_INT nodf= DIM_VECTOR_INT(arcos);
    VECTOR_INT peso= DIM_VECTOR_INT(arcos);

    for (i=0;i<arcos;i++)
    {
        fscanf(leer,"%d",&z);
        nodi[i]=z;
        fscanf(leer,"%d",&z);
        nodf[i]=z;
        fscanf(leer,"%d",&z);
```

```
        peso[i]=z;
    }
    print_vector(nodi,arcos);
    print_vector(nodf,arcos);
    print_vector(peso,arcos);

    fscanf(leer,"%d",&n);

    VECTOR_INT steiner= DIM_VECTOR_INT(n);
    VECTOR_INT valor=DIM_VECTOR_INT(nodos);
    for(i=0;i<nodos;i++)
    {
        valor[i]=0;
    }

    for(i=0;i<n;i++)
    {
        fscanf(leer,"%d",&p);
        valor[p-1]=1;
    }
    fscanf(leer,"%d",&SUMA);

    char hh1[] = "MODEL:";
    char hh2[] = "\nSETS:";
    char hh3[] = "\nNodos/1..";
    char hh3a[] = "/:TR,alfa;\nAristas/1.. ";
    char hh3b[] = "/:X,N1,N2,Coste;\nArcos/1..";
    char hh3c[] = "/:NO,ND;\nTriple(Arcos,Nodos) :Y;\nENDSETS\nDATA:\nCoste=";
    char hh4[] = ";\nN1=";
    char hh5[] = ";\nN2=";
    char hh6[] = ";\nNO=";
    char hh7[] = ";\nND=";
    char hh8[] = ";\nTR=";
    char hh8b[]= ";\nM=";
    char hh9[] = ";\nENDDATA\nMIN=@SUM(Aristas(z):Coste(z)*X(z))+M;\n@FOR(Nodos(k)|TR(k)#EQ#1: Alfa(k)=1);\
   FILE * fichero;
fichero = fopen( "problema1.txt" , "w" );


if(fichero== NULL)
{
    printf("\nERROR");
}
```

```
else
    {
    printf("\nBIEN LEIDO");
    }
fwrite(hh1 , 1 , sizeof(hh1) , fichero);
fwrite(hh2 , 1 , sizeof(hh2) , fichero);
fwrite(hh3 , 1 , sizeof(hh3) , fichero);
fprintf(fichero,"%d",nodos);
fwrite(hh3a , 1 , sizeof(hh3a) , fichero);
fprintf(fichero,"%d",arcos);
fwrite(hh3b , 1 , sizeof(hh3b) , fichero);
fprintf(fichero,"%d",arcos2);
fwrite(hh3c , 1 , sizeof(hh3c) , fichero);
for (i = 0; i < arcos; i++)
{
    fprintf (fichero, "%d ", peso[i]);
}

fwrite(hh4 , 1 , sizeof(hh4) , fichero);

for (i = 0; i < arcos; i++)
{
    fprintf (fichero, "%d ", nodi[i]);
}

fwrite(hh5 , 1 , sizeof(hh5) , fichero);

for (i = 0; i < arcos; i++)
{
    fprintf (fichero, "%d ", nodf[i]);
}

fwrite(hh6 , 1 , sizeof(hh6) , fichero);

for (i = 0; i < arcos; i++)
{
    fprintf (fichero, "%d ", nodi[i]);
}
for (i = 0; i < arcos; i++)
{
    fprintf (fichero, "%d ", nodf[i]);
}
```

```
fwrite(hh7 , 1 , sizeof(hh7) , fichero);

for (i = 0; i < arcos; i++)
{
    fprintf (fichero, "%d ", nodf[i]);
}
for (i = 0; i < arcos; i++)
{
    fprintf (fichero, "%d ", nodi[i]);
}

fwrite(hh8 , 1 , sizeof(hh8) , fichero);

for (i = 0; i < nodos; i++)
{
    fprintf (fichero, "%d ", valor[i]);
}
fwrite(hh8b , 1 , sizeof(hh8b) , fichero);
fprintf(fichero,"%d",SUMA);

fwrite(hh9 , 1 , sizeof(hh9) , fichero);

fclose(fichero);


    return 0;
}
```