Trabajo Fin de Grado

Grado en Ingeniería de Organización Industrial

Estudio Experimental de Formulaciones de Flujo para Resolver el Problema de Steiner en Grafos

Autor: Antonio García Elías

Tutor: José Manuel García Sánchez

Dep. Organización Industrial y Gestión de Empresas I

Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Grado
Grado en Ingeniería de Organización Industrial

Estudio Experimental de Formulaciones de Flujo para Resolver el Problema de Steiner en Grafos

Autor:

Antonio García Elías

Tutor:

José Manuel García Sánchez

Profesor titular

Dep. de Organización Industrial y Gestión de Empresas I

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2017

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Sevilla, 2017

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

OBJETIVO DEL TRABAJO

En la actualidad existen una serie de métodos o estrategias los cuales tienen como fin la obtención de la solución óptima de un grafo de Steiner.

Esta solución no es más que un camino marcado en el grafo. Si una cantidad de flujo sigue dicho camino, los costes totales de transporte por dicha ruta serán los menores posibles.

Con el objetivo de resolver de forma óptima un grafo de Steiner existen diferentes estrategias como pueden ser las de niveles, o las estrategias de flujo. Son estas últimas las que se estudiarán en este trabajo.

Respecto a estrategias de flujo, existen dos formulaciones básicas, por un lado un envío de flujo individualmente a cada nodo terminal, y por otro lado, un envío colectivo. Etiquetaremos estas estrategias como estrategia de flujo simple y estrategia de flujo múltiple.

No tenemos constancia de ningún estudio previo comparando ambas estrategias para resolver el problema de Steiner.

El objetivo es analizar experimentalmente, usando una librería de optimización, ambas estrategias, utilizando un conjunto de problemas de Steiner (steinbXX y steincXX) planteados en la OR library [7] mantenida por J.Beasley, en la página web del departamento de investigación operativa, del Imperial College de Londres.

La finalidad es obtener datos concluyentes que sirvan de ayuda a la hora de escoger una de las dos estrategias, dependiendo del tipo de problema al que nos estemos enfrentando.

INTRODUCCIÓN

Una red de comunicaciones (grafo) se compone de un conjunto de nodos emisores/receptores de información conectados por enlaces que permiten la transmisión de la información.

Uno de los principales problemas de construcción de grafos plantea el diseño de una topología de interconexión de sus nodos de modo que la red verifique ciertas características de confiabilidad. La confiabilidad de una red es una medida que evalúa la probabilidad de éxito en la comunicación entre pares de nodos.

La gran demanda de comunicaciones de datos en los últimos quince años ha propulsado el diseño y desarrollo de la infraestructura de redes. El continuo crecimiento en el tamaño de los problemas ha conducido a los investigadores a proponer alternativas a los enfoques exactos tradicionales para la resolución de problemas complejos. En este contexto, varias heurísticas se han aplicado al problema de diseño en redes de comunicaciones confiables. Entre ellas, las técnicas de programación evolutiva, y los algoritmos genéticos en particular, se han manifestado como métodos flexibles y robustos para la solución de los complejos problemas de optimización subyacentes al diseño de redes de comunicaciones.

Considerando una red de comunicaciones con ciertos nodos distinguidos, denominados terminales, el Problema de Steiner Generalizado (GSP) refiere al diseño de una subred de mínimo costo, que verifique ciertos requerimientos prefijados de conexión entre pares de nodos terminales.

La minimización del costo total de una red de comunicaciones y la maximización de su confiabilidad son objetivos contrapuestos. Por ello, un modelo que minimice el costo total de la red satisfaciendo el requisito de conexión entre todo par de nodos terminales sin agregar redundancia de caminos constituye una solución muy sensible a las fallas en los nodos o en los enlaces que afectarían la operatividad de la red.

En este trabajo se analizará el problema de Steiner generalizado, explicando ciertas aplicaciones que éste pueda tener en la vida real y posteriormente se concretará con las definiciones de las diferentes estrategias de flujo, donde se plasmarán sendos modelos matemáticos especificando el objetivo por el cual se han definido sus correspondientes variables, restricciones y funciones objetivo.

Acto seguido se realizará una breve introducción a la herramienta Lingo, la cual usaremos para automatizar el proceso de resolución de los problemas. Se detallará el proceso que se ha seguido hasta llegar al código de C con el que hacemos la conversión de los enunciados a los modelos en leguaje Lingo.

Por último se detallarán los experimentos realizados y se expondrán y compararán los resultados obtenidos con las distintas estrategias, para finalizar con las conclusiones pertinentes.

PROBLEMA DE STEINER GENERALIZADO

Fue propuesto por el matemático alemán Jacob Steiner a principios del siglo XIX y consiste en encontrar la ruta mínima que interconecte varios puntos de una red modelada como un grafo con costes asociados y no dirigido. Es un problema genérico, que engloba un amplio espectro de situaciones en áreas como telecomunicaciones, transporte y abastecimiento y que pueden ser expresadas mediante modelos de redes de transporte o comunicación.

El problema consiste en obtener una serie de caminos con un mínimo costo, representado por un árbol de expansión, para interconectar dos nodos o destinos dentro de una red de comunicación, representada por un grafo no dirigido, tomando en cuenta el costo de transición de los nodos intermedios en cada uno de los caminos. En general, la solución óptima no es trivial de calcular, debido a que este problema ha sido probado como un NP-Completo (Esbensen, 1994).

DEFINICIÓN

Dado un grafo pesado no dirigido G=(N,A), con vértices y aristas, se tiene un subconjunto N de nodos denominados terminales, los cuales se definen como T Є N. Dado dos nodos terminales del conjunto, se tiene que un camino interconecta dos nodos terminales directamente o a través de otros S nodos denominados nodos Steiner S Є (T-N), los cuales sirven de interconexión entre estos dos nodos terminales dados.

Dicho grafo no tiene por qué tener una representación en dos dimensiones, puede tener arcos cruzados, lo que implica que posee tres dimensiones espaciales.

La representación de dicho problema en dos dimensiones es un caso particular del problema de Steiner generalizado, el cual tiene representación gráfica en el plano.

Se puede observar un grafo constituido por nodos Terminales (verde) y nodos Steiner (rojo) en la Fig.1. En dicha representación hay arcos cruzados lo cual indica que estamos en un caso común.

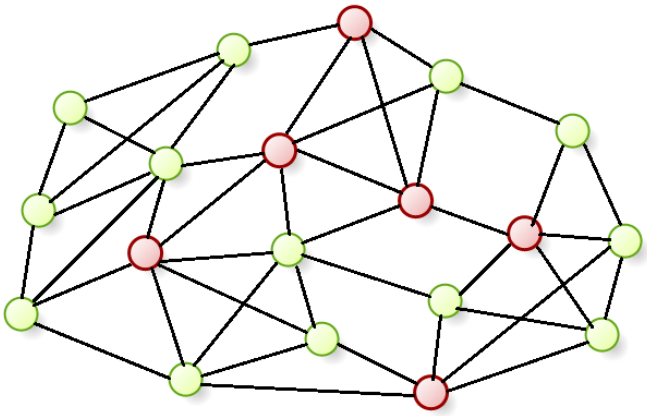


Fig.1 Grafo Steiner

Fuente: Elaboración propia

La solución de dicha red es un subgrafo, en el que todos los nodos terminales (verdes) están conectados y los nodos Steiner (rojos) pueden, o no, estar en dicha red, de manera que sirvan para conectar nodos terminales. Una posible solución a dicha red en la Fig.2.

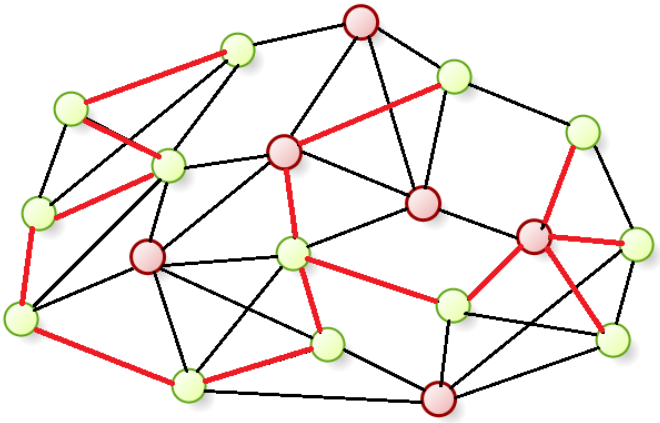


Fig.2 Solución Grafo Steiner

Fuente: Elaboración propia

El camino marcado en rojo indica la solución del problema. Esta solución engloba a un conjunto de nodos en el cual se han de encontrar todos los nodos terminales. También se pueden incorporar nodos Steiner a la solución si el modelo lo estima oportuno, como en este caso.

Un grafo puede tener varias soluciones admisibles, como se pueden ver a continuación:

Fig.3 Soluciones Admisibles

Fuente: Elaboración propia

La solución óptima de dicha red siempre forma un árbol, es decir, tiene N-1 arcos conectados, siendo N el número de nodos terminales más el número de nodos Steiner que forman parte de la solución, donde los arcos suman el coste mínimo de la red.

APLICACIONES DEL PROBLEMA DE STEINER

Un caso particular de nuestro problema es el STP Steiner Tree Problem.

La diferencia entre STP y el problema generalizado es que en este caso podemos hacer una representación del grafo en dos dimensiones ya que no hay cortes entre arcos del grafo.

Dicho problema tiene aplicaciones en el diseño de circuitos eléctricos y redes de telecomunicaciones.

El campo de las telecomunicaciones es particularmente rico en problemas que han desembocado en el tradicional problema de Steiner. Entre ellos se encuentran el problema del diseño de red jerárquica (Dynamic Predicate Stasching Copy Problem) que constituye una aplicación para la entrega de información y distribución en los sistemas de computadores.

La mayoría de las versiones del problema de Steiner son NP-completo. De hecho uno de estos pertenece a la lista de 21 problemas NP-completos de Karp. Algunos casos concretos pueden ser resueltos por tiempo polinómico, sin embargo, en la práctica se usa la heurística debido a la dificultad de encontrar la solución óptima.

En teoría de la complejidad computacional, la clase de complejidad NP-completo es el subconjunto de los problemas de decisión en NP tal que todo problema en NP se puede reducir en cada uno de los problemas de NP-completo.

Se puede decir que los problemas de NP-completo son los problemas más difíciles de NP y muy probablemente no formen parte de la clase de complejidad P.

La razón es que de tenerse una solución polinómica para un problema NP-completo, todos los problemas de NP tendrían también una solución en tiempo polinómico.

Si se demostrase que un problema NP-completo, llamémoslo A, no se pudiese resolver en tiempo polinómico, el resto de los problemas NP-completos tampoco se podrían resolver en tiempo polinómico.

Esto se debe a que si uno de los problemas NP-completos distintos de A, digamos X, se pudiese resolver en tiempo polinómico, entonces A se podría resolver en tiempo polinómico, por definición de NP-completo.

Ahora, pueden existir problemas en NP y que no sean NP-completos para los cuales exista solución polinómica, aun no existiendo solución para A.

APLICACIÓN EN LAS REDES DE TELECOMUNICACIONES

Una red de ordenadores (también llamada red de comunicaciones de datos o red informática) es un conjunto de equipos informáticos y software conectados entre sí por medio de dispositivos físicos que envían y reciben impulsos eléctricos, ondas electromagnéticas o cualquier otro medio para el transporte de datos, con la finalidad de compartir información, recursos y ofrecer servicios.

En este caso los equipos informáticos (PCs) serían los nodos del grafo y el canal de transmisión de los datos entre ordenadores serían los arcos.

Se podría concretar un poco más el problema si consideramos como nodos terminales a los servidores que dan cobertura a la red, de manera que la información tenga que pasar por ellos necesariamente, y nodos Steiner a los computadores personales.

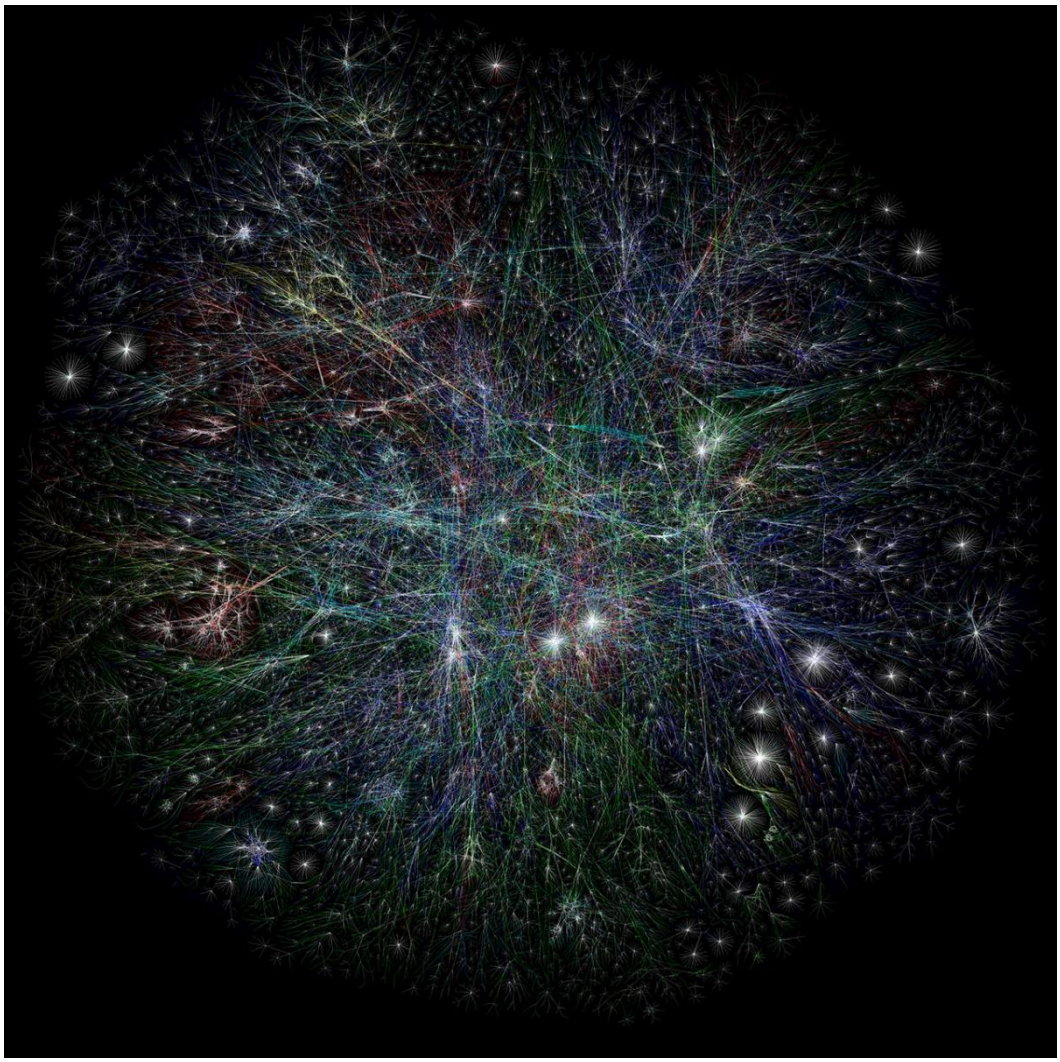


Fig.4 Mapa de Internet

Fuente: https://blog.kaspersky.es/amazing-internet-maps/7185/

Cada línea dibujada entre dos nodos representa el enlace entre dos direcciones IP. La longitud de las líneas es proporcional al tiempo de espera entre los nodos. La imagen representa 30% de las redes tipo C accesibles al programa de colección de datos de 2005.

APLICACIÓN EN LA RED DE CARRETERAS

Podría darse el caso de un camión de una compañía de transporte que tiene que entregar varios paquetes en diferentes ciudades, las cuales conformarán el conjunto de nodos terminales, y el resto de ciudades que formarán los nodos Steiner.

Las carreteras que conectan estas ciudades serían los arcos y el coste de cada uno se podría ajustar relacionando los kilómetros de longitud de la vía con el consumo del vehículo, sumando un coste adicional si hay peaje, o dándole más peso a las autovías por donde la velocidad límite es mayor y, por consiguiente, debemos tardar menos.

Con todos estos datos conseguiríamos conformar un grafo de Steiner y podríamos optimizar la ruta de transporte en la medida de lo posible.

Fig.5 Mapa de Carreteras Península

Fuente: Google Maps

En la imagen anterior vemos el mapa de autovías y autopistas de la Península, que deja entrever un grafo con una serie de carreteras (arcos) que unen varias ciudades (nodos).

Usando como origen cualquier ciudad de las que están reflejadas en el mapa (nodo raíz) y analizando datos para asignar costes a cada autovía, se puede modelar un problema de optimización de Steiner.

El PROBLEMA DE STEINER DE FLUJO

En la actualidad se dispone de dos estrategias de flujo que resuelven el problema de Steiner, la estrategia de flujo simple y la estrategia de flujo múltiple.

Ambas tienen el mismo fin, conseguir un ruta que minimice los costes de transporte de flujo, pero cada método plantea el problema de forma diferente.

La estrategia de flujo múltiple se presenta de la forma más sencilla posible con restricciones modeladas mediante el uso de la lógica y el sentido común. En dicha estrategia el flujo se envía de forma colectiva a todos los nodos terminales.

La estrategia de flujo simple nace con el fin de solucionar una problemática que tenía el anterior, y es que éste, al usar cotas en sus restricciones, se podía demorar en la obtención del óptimo. En esta estrategia el flujo se envía individualmente a cada nodo terminal.

A continuación se presentan ambos modelos y se procede a su explicación detallada.

ESTRATEGIA DE FLUJO MÚLTIPLE

ELEMENTOS PARTICIPANTES EN EL PROBLEMA

A continuación se explicarán los elementos que participan en el sistema. Son de naturaleza diversa, desde personas, herramientas, lugares, tiempo, etc.

Suelen tener asociada información y las llamaremos variables asociadas. Estas variables tienen que contener información numérica. Participan en las acciones que se producen en el sistema y también soportan las restricciones del mismo.

Están íntimamente relacionadas con las acciones que se producen en el sistema, y en ocasiones, resulta eficaz la identificación conjunta de ambos componentes.

En este caso los elementos y sus correspondientes variables asociadas serían las siguientes:

ELEMENTOS	TIPO	VARIABLES ASOCIADAS
Nodos	i,j = 1…N	• Ti (Ti=1 si i es terminal), **binario**, i=r (nodo raíz)
Arcos	(i,j)	• Cij (coste de i a j), **entero**

Tabla 1. Elementos Estrategia Flujo Múltiple

Fuente: Elaboración propia.

Llamamos N al conjunto de todos los nodos que conforman el grafo.

Llamamos A al conjunto de todos los nodos que conforman el grafo.

En el grupo de elementos que se han definido se han fijado una serie de variables necesarias para la resolución del problema. A continuación se explican una a una:

- Ti: Es una variable binaria que indica que el nodo es terminal, si vale 1, y que no lo es si vale 0.
- r: Variable que sirve para diferenciar del resto al nodo raíz. El nodo raíz es aquel que se usa como origen del camino solución. Cuando el valor de i sea r, significará que estamos ante el nodo raíz.
- Cij: Valor de coste que se le asigna a cada arco originariamente. Indica qué coste se sumaría al coste final si este arco formara parte de la solución.

VARIABLES DE DECISIÓN

Acciones directas e indirectas que se producen en el sistema para las que es necesario decidir su valor, el cual no está determinado. Se encuentran asociadas a los elementos.

Las variables son acciones simples, es decir, no pueden ser fruto de un cálculo, función simple o combinación de otras variables de decisión.

En cambio, sí es posible poseer variables de decisión cuyos valores estén condicionados al valor de otras variables de decisión.

Las variables de decisión definen las variables principales del modelo. Los valores que pueden asignarse a éstas son las que podríamos asignar a una variable cualquiera (valor binario (1/0), valor entero o valor continuo).

En este caso la actividad que vamos a realizar será la de seleccionar un arco del conjunto para que forme parte de la solución. Esta variable será binaria y valdrá cero si no seleccionamos el arco y uno si, por el contrario, sí lo seleccionamos.

- Seleccionar ARCO (i,j) para el conjunto:

α_{ij} → 1 si el arco (i,j) se selecciona para la solución

α_{ij} → 0, si no

RESTRICCIONES

Son normas y acciones definidas que se deben cumplir en el sistema.

Las restricciones de este problema generan las fórmulas matemáticas, que más adelante se explicarán con detalle una por una:

1) $\sum_k X_{ik} = \sum_j X_{ji} - 1$ $, \forall i / i \neq r, T_i = 1$

2) $\sum_k X_{ki} = \sum_j X_{ij}$ $, \forall i / \ T_i = 0$

3) $X_{ij} \leq CS * \alpha_{ij}$ $, \forall (i,j) \in A$

4) $X_{ji} \leq CS * \alpha_{ij}$ $, \forall (j,i) \in A$

5) $\sum_j X_{rj} = CS$

6) $(\sum_{i=0}^{N} T_i) - 1 = CS$ $, \forall i / i \in N$

7) $\alpha_{ij} \in [0,1]$ $, \forall (i,j) \in A$

A continuación se procede a explicar las restricciones individualmente.

1) La primera restricción indica que para cualquier nodo terminal que no sea el nodo raíz, el flujo que entra por dicho nodo debe de ser al que sale menos una unidad. Esto significa que los nodos terminales siempre se quedan con una unidad de flujo, por lo que el flujo total disminuye cada vez que pasa por un nodo de estas características.

2) La segunda restricción nos indica que el flujo que entra en un nodo Steiner es exactamente el mismo que el que sale. Por lo tanto, los nodos Steiner no se quedan con ninguna unidad de flujo y el flujo total no varía cuando pasa por estos nodos.

3) La tercera restricción dice que el flujo que pasa por cada arco (i,j) debe de ser menor o igual a una cota superior (la cual luego vemos cuánto vale) multiplicada por el alpha, que indica si el arco se ha activado o no. Con esto se consigue que, si el arco forma parte de la solución, nunca pase por el nodo más flujo que el que ha enviado el nodo raíz.

4) Esta restricción es exactamente la misma que la anterior pero para arcos de sentido contrario (j,i).

5) La quinta restricción indica que la cantidad de flujo total que envía el nodo raíz al resto es exactamente la cota superior. Con esto se consigue que se envíen tantas unidades como requieran los nodos terminales, sabiendo ya que cada nodo terminal se queda con una unidad de flujo.

6) La sexta nos define la cota superior como el número de terminales menos uno. Esto no es más que un cálculo que sirve de apoyo a la hora de solucionar el problema y que resulta muy útil si es utilizado en restricciones como la 5). Con esto se consigue que el total de flujo enviado por el nodo raíz sea exactamente el flujo que necesitan los nodos terminales.

7) La séptima no es más que una definición de la variable alfa como binaria.

FUNCIÓN OBJETIVO

Es el criterio que guía la resolución del sistema. El objetivo será una función de costes asociados directa o indirectamente a las funciones que marcan las variables que se producen en el sistema. Los costes que se presentan en un sistema guían a una correcta definición de las variables de decisión.

En este caso la función objetivo busca minimizar el coste total de transportar una unidad de flujo por el camino obtenido como solución.

$$\sum_{(i,j)\in A} c_{ij} * \alpha_{ij}$$

ESTRATEGIA DE FLUJO SIMPLE

ELEMENTOS PARTICIPANTES EN EL PROBLEMA

La tabla de elementos en este caso es la misma que la anterior, aunque luego

ELEMENTOS	TIPO	VARIABLES ASOCIADAS
Nodos	i,j = 1…N	• Ti (Ti=1 si i es terminal), **binario**, i=r (nodo raíz)
Arcos	(i,j)	• Cij (coste de i a j), **entero**

Tabla 2. Elementos Estrategia Flujo Simple

Fuente: Elaboración propia.

Llamamos N al conjunto de todos los nodos que conforman el grafo.

Llamamos A al conjunto de todos los nodos que conforman el grafo.

En el grupo de elementos que se han definido se han fijado una serie de variables asociadas necesarias para la resolución del problema. A continuación se explican una a una:

- Ti: Es una variable binaria que indica que el nodo es terminal si vale 1 y que no lo es si vale 0.
- r: Variable que sirve para diferenciar del resto al nodo raíz. El nodo raíz es aquel que se usa como origen del camino solución. Cuando el valor de i sea r, significará que estamos ante el nodo raíz.
- Cij: Valor de coste que se le asigna a cada arco originariamente. Indica qué coste se suma al coste final si este arco formara parte de la solución.

VARIABLES DE DECISIÓN

En este caso la actividad que vamos a realizar vuelve a ser la de seleccionar un arco del conjunto para que forme parte de la solución. Esta variable será binaria y valdrá cero si no seleccionamos el arco y uno si, por el contrario, sí lo seleccionamos.

- Seleccionar ARCO (i,j) para el conjunto:

α_{ij} → 1 si el arco (i,j) se selecciona para la solución

α_{ij} → 0, si no

RESTRICCIONES

Las restricciones de este problema generan las fórmulas matemáticas, que más adelante se explicarán con detalle una por una:

1) $\sum_{j:(i,j)\in A}(x_{ij}^k - x_{ji}^k) = 0 \quad ,\forall j \neq \{k,r\}, \forall k \neq r / T_k = 1$

2) $\sum_{j:(r,j)\in A} x_{rj}^k = 1 \quad ,\forall j = r / j \neq k, \forall k \neq r / T_k = 1$

3) $x_{ij}^k \leq \alpha_{ij} \quad ,\forall (i,j), \forall k \neq r / T_k = 1$

4) $x_{ji}^k \leq \alpha_{ij} \quad ,\forall (i,j), \forall k \neq r / T_k = 1$

5) $x_{ij}^k \in \{0,1\} \quad ,\forall (i,k) / T_k = 1, \forall j \in N$

6) $\alpha_{ij} \in [0,1] \quad ,\forall (i,j) \in A$

A continuación se procede a explicar las restricciones individualmente.

1) 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 de ser el mismo que sale desde i hacia el resto de los nodos conectados, pasando siempre por k. Es decir, lo que llega tiene que ser igual que lo que sale.

2) La segunda restricción nos dice que el nodo raíz debe enviar una unidad de flujo a cada nodo terminal.

3) La tercera indica que el flujo que tiene que pasar por cada arco tiene que ser menor o igual que uno.

4) La cuarta es igual que la anterior pero para arcos de sentido contrario.

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

6) La sexta nos fija el alfa como una variable binaria.

FUNCIÓN OBJETIVO

La función objetivo busca de nuevo minimizar el coste total de transportar una unidad de flujo por el camino obtenido como solución.

$$\sum_{(i,j)\in A} c_{ij} * \alpha_{ij}$$

LINGO

INTRODUCCIÓN A LINGO

Lingo es una herramienta informática utilizada para resolver modelos de optimización tanto lineales como no lineales automáticamente y de forma rápida, ofreciendo una hoja de resultados muy detallada que nos permitirá interpretar estos de forma cómoda.

Para programar en Lingo es necesario utilizar su propio lenguaje, el cual describiremos brevemente a continuación, así como el formato necesario para escribir el modelo.

El formato consta de 5 partes:

- **Encabezado**

  Sirve para indicarle a Lingo que vamos a escribir un modelo.
  MODEL:

- **Función Objetivo**

  Maximizar o minimizar con los valores de los costes de todas las variables.
  *MIN = 3*X1 + 4*X2 + X3 - 7*X4 + 2*X5 + 7;*
  Es importante indicar que, a fin de evitar errores, es necesario poner punto y coma (;) al final de cada línea, de modo que indiquemos al programa que en la siguiente línea se va a escribir otra indicación diferente a la anterior.

- **Restricciones del problema**

  Al igual que el todos los modelos de optimización, existen una serie de restricciones a cumplir que hay que reflejarlas manteniendo la siguiente forma.
  *X1+3*X4<=7;*
  *X3-7*X2>=1;*
  …

- **Restricciones asociadas al tipo de variable**

  - En Lingo, por defecto, las variables son continuas (≥ 0)
  - Las variables enteras se escriben como @GIN(X1);
  - Las variables binarias se escriben como @BIN(X2);
  - Las variables libres se escriben como @FREE(X3);

- **Fin**

  Para indicarle a Lingo que hemos finalizado la escritura del modelo.
  END

Hay que tener en cuenta ciertas consideraciones como por ejemplo:

- Lingo ignora las líneas en blanco.
- Si queremos insertar un comentario dentro de la misma programación del modelo debemos preceder éste del símbolo ¡ y al final de un punto y coma (;).
- Lingo no distingue entre mayúsculas y minúsculas.
- La longitud máxima de una línea es de 512 caracteres.

A continuación se muestra un modelo de optimización que será programado en lenguaje Lingo de forma que tengamos un ejemplo de modelo completo para poder ver más claramente las partes de las que constan los modelos en este lenguaje.

$$Max\ x_1 + x_2 + y$$
$$s.a.$$
$$x_1 \leq 2$$
$$x_1 + y \leq 3$$
$$y + 2x_2 \leq 4$$
$$x_1 \geq 0, x_2 \geq 0\ Entera$$
$$y\ binaria$$

Fig.6 Modelo Ejemplo Lingo

Fuente: Resolución de Problemas (Librerías de Optimización)

Si escribimos el modelo anterior en lenguaje Lingo quedaría como lo siguiente:

```
MODEL:

[OBJETIVO] MAX = x1  +  x2  +  y  ;

[PRIMERA] x1 <=2; !ejemplo de comentario;

[SEGUNDA] x1 + y <=3;

[TERCERA] y + 2*x2 <=4;

[Signo_x1] x1>=0;
[Signo_x2] x2>=0;

@GIN(X2);
@BIN(y);

END
```

Fig.7 Código Ejemplo Lingo

Fuente: Resolución de Problemas (Librerías de Optimización)

Aparece una función objetivo seguida de cinco restricciones, de las cuales dos de ellas sólo sirven para indicar que X1 y X2 son positivos.

También aparecen dos restricciones asociadas al tipo de variable que nos indican que X2 es una variable entera y que 'y' representa una variable binaria.

En el caso de no haber indicado nada en las restricciones asociadas al tipo de variable, Lingo asumirá que éstas pueden tomar cualquier valor real.

AMPLIACIÓN DE LINGO

USO DE CONJUNTOS

Cuando queremos expresar un modelo de forma simplificada (abstracta), es necesario el uso de conjuntos. Los conjuntos tienen una equivalencia directa con los elementos del problema, con una diferencia. Lingo distingue dos tipos de conjuntos:

-Conjuntos primitivos: Recogen los elementos conjunto del problema y sus variables asociadas independientes o relacionadas con elementos no conjuntos

-Conjuntos derivados: Recogen las variables asociadas relacionadas entre elementos conjunto. Están formados, por tanto, por los elementos primitivos ya definidos.

Las variables asociadas sólo se definen, por tanto, una única vez.

Los elementos concepto o unitarios simples no es necesario definirlos. Sus variables asociadas independientes se definen directamente en la sección de valores (DATA).

LINGO posee dos secciones para la definición de los conjuntos.

-Sección SET: Se definen los nombres y número de elementos de los conjuntos

-Sección DATA: Se definen los valores de las variables asociadas. Es posible también definir aquí el número de elementos del conjunto.

Por ejemplo, en el caso de un grafo, la tabla de elementos estaría compuesta por nodos y arcos. Es esta tabla la que habría que definir en la sección SETS.

Si suponemos un grafo con 9 nodos cuyas variables asociadas son ser terminal o no, y 13 arcos cuyas variables asociadas son un coste y un valor de tránsito, su modelado en Lingo sería el siguiente:

```
SETS:
Nodos/1..9/:Ti;
Arcos(Nodos,Nodos)/1 2,2 4,2 5,4 5,4 7,7 5,8 5,7 8,5 3,3 6,5 6,5 9,6
9/:C,X;
ENDSETS
```

Podemos ver cómo hemos definido un conjunto de 9 nodos (/1..9/) y posteriormente hemos indicado la característica como Ti.

Lo mismo hemos hecho con los arcos, aunque aquí hemos indicado que estos están formados por dos valores dados por dos nodos (Nodos,Nodos), especificando posteriormente qué nodos forman cada arco. Por ejemplo, para el primer arco (1 2), su origen sería el nodo 1 y el destino sería el nodo 2, y así para todos los arcos.

Finalmente hemos definido sus variables asociadas que son C para el coste de cada arco y X para el valor de tránsito.

Si queremos fijar valores para las variables asociadas usaremos la sección DATA. En el caso de los problemas que se tratan en este trabajo, el enunciado nos facilita cuáles son los nodos terminales, así como el coste de cada arco. Por lo tanto, tendríamos que fijar estos valores para trabajar con ellos.

Podemos considerar la variable Ti como un valor binario que valdrá 1 si el nodo es terminal y 0 si no. El valor de coste de cada arco es un valor entero que nos proporciona el enunciado. Todo esto lo definiríamos en Lingo de la siguiente forma:

```
DATA:
Ti=0,1,1,1,0,1,1,0,1;
C=2,1,3,2,1,3,2,3,1,1,3,2,3;
ENDDATA
```

Ahora tenemos 9 valores de Ti. El nodo 1, por ejemplo, no es terminal, ya que es el primer valor de Ti el que corresponde al nodo 1.

Este mismo orden se sigue para el coste. El coste del primer arco definido (1 2) es de 2 y así sucesivamente.

RESTRICCIONES

A la hora de representar restricciones en Lingo usamos las funciones @FOR y @SUM.

Función	Uso
@FOR	*@FOR* es principalmente usado par generar restricciones sobre los miembros de un conjunto.
@SUM	*@SUM* calcula la suma de una expresión sobre los miembros de un conjunto.

Aplicación: $\forall i: \sum_{j=1}^{12} x_{ij} \leq A_i$

@FOR @SUM

Tabla 3. Restricciones en Lingo

Fuente: Resolución de Problemas (Librerías de Optimización)

El formato utilizado en las restricciones para los signos es el siguiente:

#EQ#	=
#NE#	≠
#GE#	≥
#GT#	>
#LT#	<
#LE#	≤

Fig.8 Signos en Lingo

Fuente: Resolución de Problemas (Librerías de Optimización)

Por poner un ejemplo para aclarar un poco más estos conceptos trataremos un sistema con 100 proveedores a los que compro unidades de un producto.

Hay que imponer que a cada proveedor no se le compren más de 2500 unidades.

```
SETS:
Proveedores/1..100/:X;
Producto;
ENDSETS
@FOR(Proveedores(i):X(i)<=2500);
```

Como se puede ver, hemos definido los 100 proveedores con una variable X que indica las unidades que se compran a cada proveedor.
El @FOR indica que para **cada proveedor** no podemos comprar más de 2500 unidades.

Un ejemplo para la función @SUM sería:

```
SETS:
Clientes/1..12/: Demanda;
ENDSETS
DATA:
Demanda = 20, 25, 25, 30, 10, 10 ,16, 23, 25, 35, 30, 13;
ENDDATA
DemandaTotal = @SUM( Clientes(j): Demanda(j) );
```

En este último ejemplo se han definido doce clientes cada cual tiene una demanda que se fija en la sección DATA.

Al final definimos un valor “DemandaTotal” que es la suma de las demandas de todos los clientes, lo que se definiría matemáticamente así: $\sum_{j \in Clientes} Dj$

INTERFAZ DE USUARIO

Una vez ejecutamos la aplicación Lingo lo primero que nos debe aparecer es la siguiente ventana:

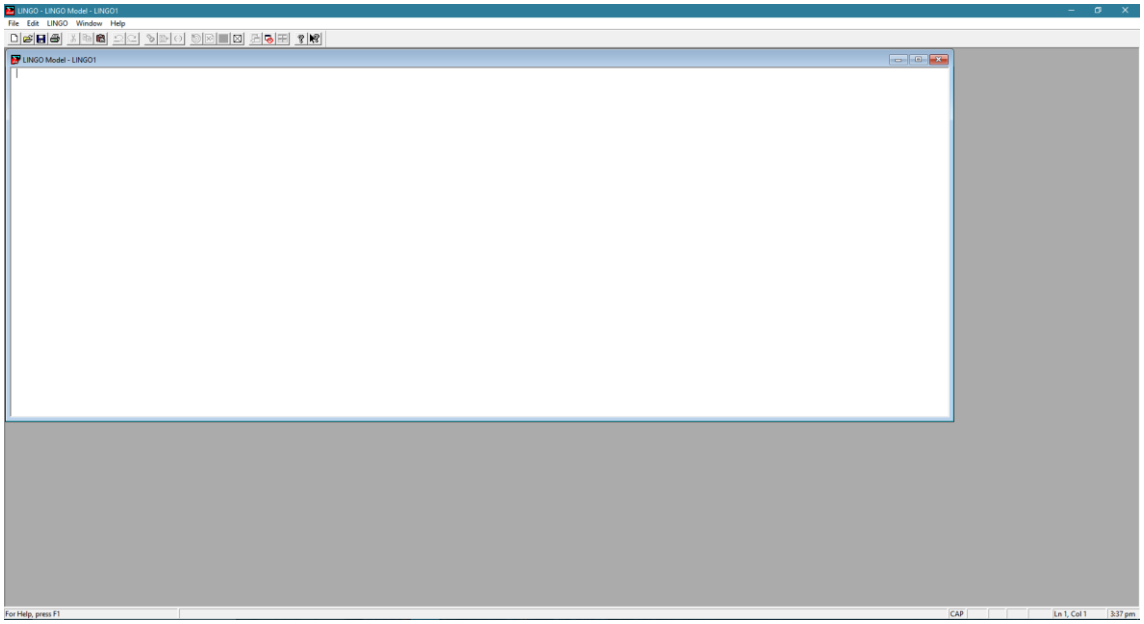


Fig.9 Interfaz de Inicio Lingo

Fuente: Elaboración Propia

La misma ventana de la aplicación contiene a otra que nos deja ver una página en blanco. Es ahí donde escribiremos el código del modelo para Lingo lo resuelva.

En la parte superior de la ventana aparecen una serie de herramientas, algunas necesarias para la resolución del modelo y otras que pueden servir de ayuda.

Una vez escribimos el código sólo tenemos que pulsar en el botón “solve” para que Lingo lo resuelva automáticamente:

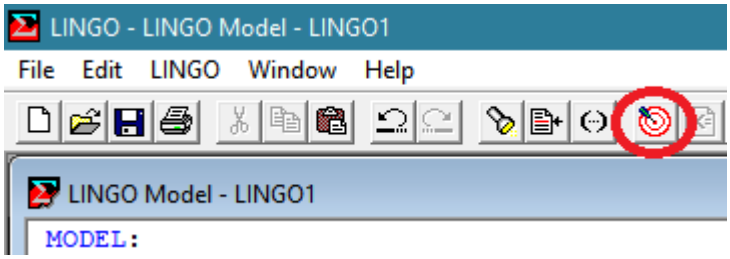


Fig.10 Barra de Herramientas Lingo

Fuente:Elaboración Propia

Es entonces cuando se pueden dar dos situaciones:

- Lingo nos indica que existe algún fallo en nuestro código, ya sea porque no hemos seguido correctamente la ortografía del lenguaje Lingo, porque estamos usando variables no definidas, etc.
- Lingo resuelve el problema y nos abre una nueva ventana donde nos indica el número total de variables, número de restricciones, valor de la función objetivo, etc.

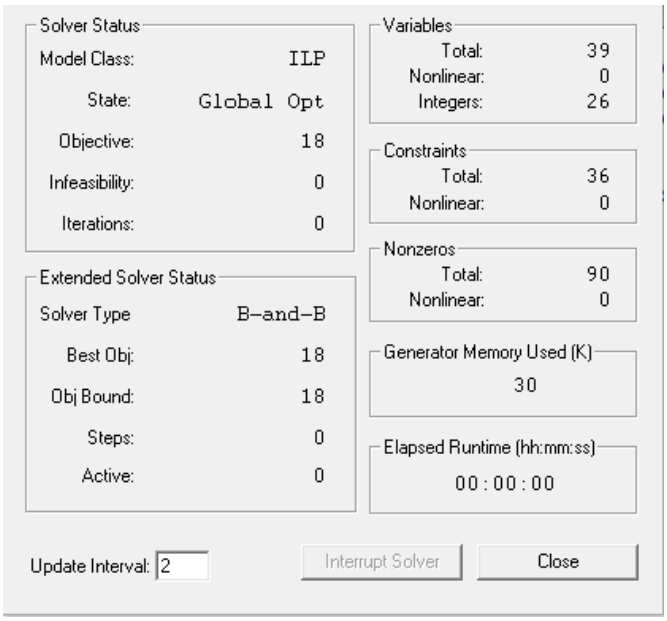


Fig.11 Ventana de Soluciones Lingo

Fuente: Elaboración Propia

Finalmente se nos abre otra ventana que contiene todos los datos obtenido en la solución como la siguiente:

```
Global optimal solution found.
Objective value:                              18.00000
Extended solver steps:                        0
Total solver iterations:                      0

                     Variable           Value        Reduced Cost
                           CS        5.000000            0.000000
                        TI( 1)       0.000000            0.000000
                        TI( 2)       1.000000            0.000000
                        TI( 3)       1.000000            0.000000
                        TI( 4)       1.000000            0.000000
                        TI( 5)       0.000000            0.000000
                        TI( 6)       1.000000            0.000000
                        TI( 7)       1.000000            0.000000
                        TI( 8)       0.000000            0.000000
                        TI( 9)       1.000000            0.000000
                         R( 1)       0.000000            0.000000
                         R( 2)       1.000000            0.000000
                         R( 3)       0.000000            0.000000
                         R( 4)       0.000000            0.000000
                         R( 5)       0.000000            0.000000
                         R( 6)       0.000000            0.000000
                         R( 7)       0.000000            0.000000
                         R( 8)       0.000000            0.000000
                         R( 9)       0.000000            0.000000
                      C( 1, 2)       2.000000            0.000000
                      C( 2, 4)       1.000000            0.000000
                      C( 2, 5)       3.000000            0.000000
                      C( 4, 5)       2.000000            0.000000
                      C( 4, 7)       1.000000            0.000000
                      C( 7, 5)       3.000000            0.000000
                      C( 8, 5)       2.000000            0.000000
                      C( 7, 8)       3.000000            0.000000
                      C( 5, 3)       1.000000            0.000000
                      C( 3, 6)       1.000000            0.000000
```

Fig.12 Fichero Soluciones Lingo

Fuente: Elaboración Propia

La imagen que muestra la Fig. 12 no está completa.

A partir de los datos que obtenemos de esta última imagen podemos interpretar las soluciones, ya que no sólo nos muestra el valor de la función objetivo en el óptimo, sino que también nos ofrece la posibilidad de saber el valor de todas las variables en todas las situaciones posibles.

CONVERSIÓN DE LOS ENUNCIADOS .TXT A LINGO CON C

Los datos que se han utilizado para crear problemas de grafos Steiner proceden de archivos de texto (.txt) los cuales tiene ordenados los datos por filas de la siguiente forma:

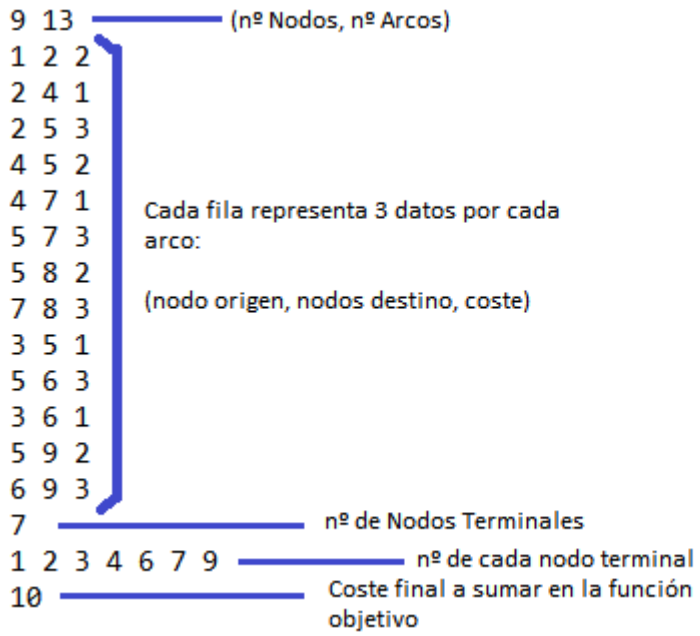


Fig.13 Enunciados Problemas de Steiner

Fuente: Elaboración Propia

Para la conversión de estos datos a un modelo resoluble en Lingo se ha utilizado el lenguaje de programación C. Se ha hecho uso de este lenguaje, y no otro, ya que ha sido el que se ha impartido en algunas asignaturas del grado.

La programación se ha hecho con el fin de leer los datos proporcionados por los archivos .txt de los enunciados por filas, guardar estos datos en unas variables definidas dentro del programa en C y finalmente mostrar en la pantalla de comandos los caracteres que ha leído indicando a qué dato del problema corresponde cada uno. Así mismo, esos datos se usan para escribir en un fichero llamado “enunciadolingo.txt”, el código del modelo en Lingo, de forma que solo habría que copiarlo y pegarlo en el programa Lingo para que este lo resolviera sin necesidad de hacer ninguna modificación posterior.

Se han generado dos programas:

- Programasimple: Genera el código en Lingo para el problema de flujo simple.
- Programamultiple: Genera el código en Lingo para el problema de flujo múltiple.

Ambos programas siguen el mismo patrón, sólo que cambian ciertos parámetros con el fin de generar los datos correctamente para el fichero "enunciadolingo.txt".

La programación comienza declarando la variable ***char*** *nombrearchivo*, como una cadena de caracteres con una longitud de 50, de manera que en ella podamos introducir el nombre del archivo de enunciado que queremos abrir.

Seguidamente se programa de manera que podamos interactuar con la ventana de comandos. Ésta nos va a pregunta qué archivo queremos abrir. Lo programamos con la función *printf* de forma que se nos muestre por pantalla una frase que invite al usuario escribir el nombre del archivo que quiere abrir.

Con la función *scanf* leemos lo que el usuario ha escrito lo asignamos a la variable *nombrearchivo.*

Posteriormente usamos la función *fopen* para abrir el fichero *nombrearchivo* al que hemos asignado la variable *f* dándole permisos de lectura. Se ha hecho con el siguiente código:

*FILE *f = fopen(nombrearchivo,"r");*

A continuación asignamos al fichero enunciadolingo.txt permisos de escritura para que podamos insertar ahí el código de Lingo. Se hizo de la siguiente forma:

*FILE *f2 = fopen("enunciadolingo.txt","w");*

Ya hemos abierto el fichero que queremos leer y hemos asignado uno que tenemos ya creado para escribir en él. Lo siguiente será programar la lectura de datos del fichero de los enunciados y la escritura en *enunciadolingo.txt*.

Para ello, empezamos definiendo las variables *nodos*, *arcos*, *i* y *j* como números enteros.

La variable *nodos* guardará el número de nodos que tiene el grafo y la variable *arcos* el número de arcos. Las variables *i* y *j* servirán de índices a la hora de realizar bucles.

Vamos a leer el fichero de datos fila a fila, ya que en cada una hay un conjunto de datos relacionado.

La primera fila tiene los datos del número de nodos y número de arcos, por lo tanto, la leemos con la función *fscanf* para leer ficheros, asignando el primer valor de la fila a la variable *nodos* y el segundo a la variable *arcos.*

Una vez los tenemos en la memoria podemos utilizarlos para escribirlos donde queramos, tanto si los queremos sacar por la ventana de comandos con *printf* como si los queremos tener plasmados en el fichero del enunciado con *fprintf*.

Para trabajar con los datos de las siguientes filas (origen, destino, coste), definimos tres vectores los cuales tienen el mismo nombre que el dato al que representan. Estos vectores tienen, cada uno, una longitud de *Nº de arcos – 1*, ya que hemos de tener en cuenta que, en C, el primer valor de un vector es el cero.

Con un bucle que ser repite tantas veces como arcos haya, leemos los datos de origen, destino y coste y, a la vez que los leemos, van estando a nuestra disposición para el uso que le queramos dar.

A continuación se lee la siguiente fila, cuyo dato representa el número de nodos terminales, y se guarda en la variable *numeroterminales.*

La siguiente fila tiene como datos los nodos que son terminales. Este dato se guardará en un array que se llamará *terminales* y tendrá como longitud *Nº de terminales – 1* por el mismo motivo que se dio con los datos de origen, destino y coste.

Los datos se van leyendo con un bucle *for* al igual que en el caso anterior. Se itera tantas veces como número de terminales haya.

Por último leemos la fila definitiva cuyo dato representa un coste a sumar en la función objetivo. Este dato se guardará en una variable llamada *costeasumar*.

Tanto la programación seguida para escribir en la ventana de comandos como la que se ha usado para escribir en el archivo de texto enunciadolingo.txt se detallan en el *ANEXO Iy II*, donde aparece el código completo.

Se procede a continuación a mostrar un diagrama de flujo que representa de forma esquemática el procedimiento que se sigue desde que se tiene el archivo de texto hasta que finalmente se consigue el código en Lingo.

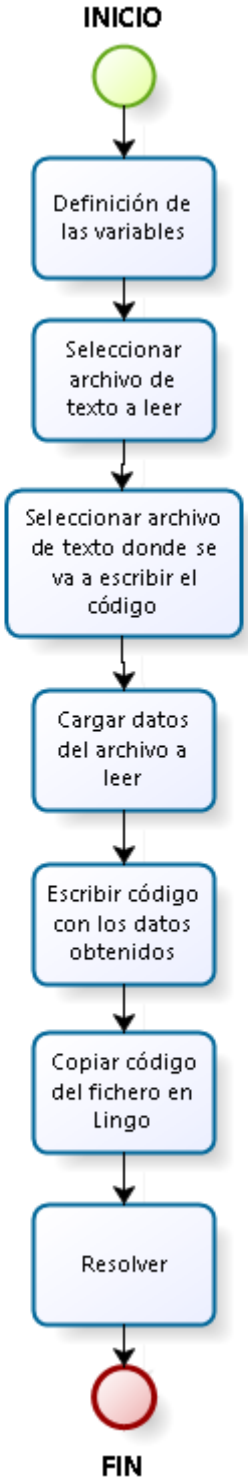


Fig.14 Diagrama de Flujo Conversión C-Lingo

Fuente: Elaboración Propia (Bizagi)

CÓDIGOS GENERADOS PARA LINGO

Como bien es sabido, Lingo posee un lenguaje de programación propio, de ahí la necesidad de codificar el programa en C.

Una vez ves los archivos de texto de los enunciados y conociendo además el lenguaje de Lingo, salta a la vista la necesidad de automatizar el proceso de codificación del problema.

Se va a disponer a continuación a describir detalladamente la codificación el Lingo de los problemas de flujo simple y múltiple.

CODIFICACIÓN PARA ESTRATEGIA DE FLUJO MÚLTIPLE

El código para la estrategia de flujo múltiple es más escueto que el simple y, por consiguiente, más fácil de comprender, por lo tanto se ha considerado conveniente explicar el múltiple primero, para ir allanando el terreno antes de explicar la estrategia de flujo simple.

Como bien se explicó con antelación, los datos se obtienen de los ficheros steinxxx.txt, donde los datos que se leen en la primera fila del fichero es el número de nodos y el número de arcos.

Esto se plasma en el modelo de Lingo a la hora de definir los conjuntos de componentes que participan en el modelo.

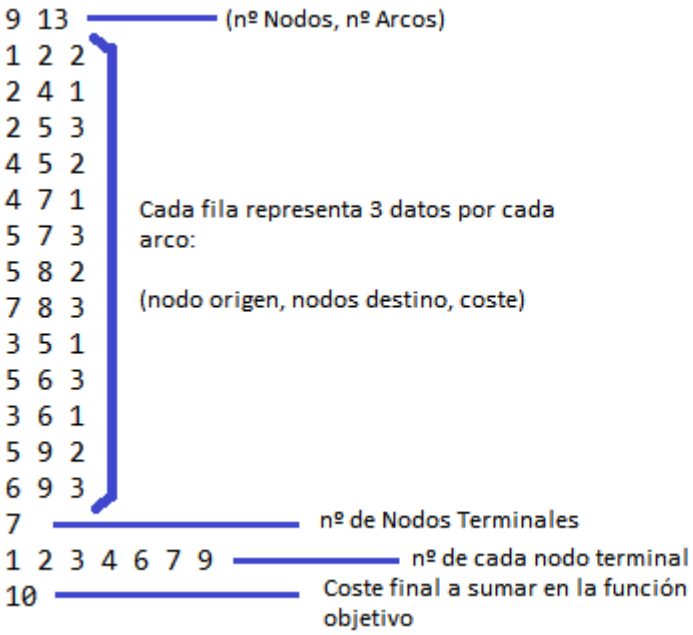


Fig.15 Enunciados Problemas de Steiner II

Fuente: Elaboración Propia

Por ejemplo, el número de nodos que nos fija la Fig. 15 es 9 y 13 para los arcos.

Empezamos definiendo el modelo en Lingo tal y como exige la programación, esto es, escribiendo MODEL y SETS en las dos primeras líneas. A continuación definimos el conjunto Nodos:

MODEL:

SETS:

Nodos/1..9/:Ti,R;

Como podemos ver, definimos Nodos diciendo que es un conjunto de recorrido 1 a 9 y posteriormente le asignamos las variables asociadas a estos, Ti y R.

- Ti nos va a indicar cuándo el nodo es terminal. Es un binario que valdrá 1 si es terminal y 0 si es nodo Steiner, tal y como se indicaba en la tabla de elementos participantes en el problema.

- R nos indicará cuál es el nodo raíz, es decir, el nodo por el cual se enviará el flujo total que circule por el grafo. Vuelve a ser un binario que valdrá 1 sola y exclusivamente para el nodo raíz.

Acto siguiente procedemos a definir el conjunto Arcos, pero esta vez no lo hacemos igual que el de los nodos, ya que el conjunto de los nodos no requiere de otro conjunto para definirse, al contrario que los arcos.

Los arcos están definidos según dos nodos, ya que necesitan de un origen y un destino, pues es a través de ellos por donde se envía el flujo de nodo a nodo.

Es necesaria, por lo tanto, la inclusión del nodo origen y el nodo destino en la definición de los arcos en Lingo. Esa definición quedaría de la siguiente manera, e iría justo a continuación del fragmento de modelo que escribimos antes:

Arcos1(Nodos,Nodos)/1 2,2 4,2 5,…/:C,Selección;

Arcos2(Nodos,Nodos)/1 2,2 1,2 4,4 2,2 5,5 2,…/:X;

ENDSETS

Como se ha podido comprobar se han definido dos conjuntos de arcos:

- Arcos1, el cual contiene todo el conjunto de los 13 arcos que componen el grafo del ejemplo de la Fig. 15.

Estos arcos tienen los nodos ordenados de forma que el del número de menor valor está a la izquierda, y el de mayor valor a la derecha. Esto no es más que una mera modificación para conseguir codificar correctamente el modelo de la estrategia de flujo múltiple en Lingo.

Tras esto, se definen las variables siguientes:

- C, es el coste que se suma al usar ese arco. Cada arco tiene un coste asociado. Si pasa flujo por éste, se añade el coste del mismo.

- Selección, corresponde a la variable binaria alpha que se explicó en el capítulo donde se hablaba de la al detalle de las estrategias de flujo simple y múltiple. Dicha variable vale 1 si el arco se usa, es decir, si pasa flujo por éste, y vale 0 si no.

- Arcos2, el cual contiene todos los arcos del grafo, tanto en un sentido como en otro. Es decir, este conjunto incluye tanto al arco que va de *a* a *b* como el que va de *b* a *a.* Se deduce por tanto que en este conjunto hay el doble de arcos que los que indica el enunciado, pero la posibilidad de coger el arco tanto en un sentido como en sentido contrario existe, ya que estamos ante un grafo no dirigido.

  De primeras puede carecer de sentido la definición de este último conjunto, pero es necesaria para la correcta codificación del modelo en Lingo.

  Por último tenemos la variable asociada a cada elemento de este conjunto, a la que hemos llamado X. Esta variable corresponde a la cantidad de flujo que pasa por cada arco.

  Se deduce por lo tanto que el flujo puede pasar por cualquier arco en los dos sentidos, pero el modelo, al buscar la optimización, sólo lo pasará en uno de los dos sentidos que marca cada arco.

Ya hemos definido los conjuntos o SETS, ahora toca fijar los datos en la sección DATA de Lingo.

En este caso fijaremos:

1. El valor de Ti para cada nodo

2. El valor de R para cada nodo

3. El valor de C para cada arco

4. El valor de una cota superior necesaria para la resolución del problema

Todos estos datos se van a obtener de los enunciados, excepto el valor de R, que lo fijaremos nosotros mismos para todos los problemas.

R será siempre el nodo terminal con el subíndice más pequeño. Esto quiere decir, si tenemos al nodo 1 como terminal, éste será el nodo raíz siempre, pero si el nodo 1 es Steiner, el nodo raíz será el siguiente siempre que sea terminal. Si el nodo 2 es terminal, en este caso sería el raíz, pero si no lo fuera tendríamos que irnos al nodo 3 y así sucesivamente.

En el caso del enunciado de la Fig. 15 el nodo raíz sería el nodo 1.

Con el ejemplo de la Fig. 15 la sección DATA nos quedaría de la siguiente forma:

DATA:

Ti=1,1,1,1,0,1,1,0,1;

R=1,0,0,0,0,0,0,0,0;

C=2,1,3,2,1,…;

CS=6;

ENDDATA

Como se puede ver, las variables Ti y R tienen tantos valores como nodos haya, y cada valor está ordenado de izquierda a derecha. El valor de la izquierda corresponde al primer nodo, y el de la derecha al último.

Lo mismo ocurre con los valores de C.

El valor de CS se obtiene definiendo un contador en el código de C. Se recorre un bucle leyendo la fila que corresponde al número de cada nodo terminal y se va sumando uno al contador cada vez que lee un número en dicha fila. De esta manera conseguimos contar el número de nodos terminales y fija el valor de CS que, como se definió en los capítulos anteriores, se corresponde con el número de terminales menos uno.

La función objetivo y las restricciones que componen el modelo ya no es necesario obtenerlas del enunciado, salvo la excepción del número que se encuentra en la última fila de estos, el cual se introduce como una variable en C, se le fija dicho valor, y se plasma en el código de la función objetivo de Lingo.

La función objetivo en el ejemplo de la Fig. 15 sería:

*MIN=10+@SUM(Arcos1(i,j):Seleccion(i,j)*C(i,j));*

Como queremos minimizar, usamos el MIN y luego vamos definiendo la función objetivo.

El valor 10 es el de la última fila de los enunciados, que es un valor fijo para cada problema.

@SUM realiza el sumatorio de todos los valores Selección(i,j)*C(i,j) de cada arco. Al fin y al cabo lo que marca la función objetivo en el modelo matemático.

La primera restricción del modelo define Selección(i,j) como un binario, tal y como se explicó en capítulos anteriores. A cada arco del conjunto Arcos1 le corresponde una variable binaria Seleccion:

@FOR(Arcos1(i,j):@BIN(Seleccion(i,j)));

La siguiente restricción fija que cada nodo terminal se queda con una unidad de flujo:

@FOR(Nodos(i)|Ti(i)#EQ#1 #AND# R(i)#NE#1:
@SUM(Arcos2(i,k)|R(k)#NE#1:X(i,k))=@SUM(Arcos2(j,i):X(j,i))-1);

Con la función @FOR recorremos todos los nodos, con la condición de que sean terminales y ninguno de ellos sea el nodo raíz (*Ti(i)#EQ#1 #AND# R(i)#NE#1*).

Con la función @SUM sumamos el flujo total que sale de cada nodo y forzamos que sea igual que el flujo que entra menos 1, por el motivo que antes se dijo (*@SUM(Arcos2(i,k)|R(k)#NE#1:X(i,k))=@SUM(Arcos2(j,i):X(j,i))-1*).

La siguiente restricción obliga a que ningún nodo Steiner se quede con una unidad de flujo.

@FOR(Nodos(i)|Ti(i)#EQ#0:@SUM(Arcos2(k,i):X(k,i))=@SUM(Arcos2(i,j):X(i,j)));

Con la función @FOR recorremos todos los nodos, con la condición de que no sean terminales (*@FOR(Nodos(i)|Ti(i)#EQ#0:*).

Con la función @SUM sumamos el flujo total que sale de cada nodo y forzamos que sea igual que el flujo que entra, por el motivo que antes se dijo (*@SUM(Arcos2(k,i):X(k,i))=@SUM(Arcos2(i,j):X(i,j))*).

A continuación aparecen dos restricciones cuyas condiciones tuvieron que ser modificadas para que Lingo pudiera entender el código y resolver el problema.

La primera restricción fuerza que el flujo que pasa por cada arco del conjunto Arcos2 tiene que ser menor o igual que la multiplicación de la cota superior, anteriormente definida, y la suma de todos los valores binarios Selección, correspondientes a los arcos del conjunto Arcos1, por ello la necesidad de relacionar ambos conjuntos.

@FOR(Arcos2(i,j)|i#LT#j:X(i,j)<=CS@SUM(Arcos1(s,k)|s#EQ#i #AND# k#EQ#j:Seleccion(s,k)));*

En este caso seleccionamos los arcos del conjunto Arcos2 cuyo primer valor sea menor que el segundo. Con esto barremos la primera mitad de los arcos.

En la segunda restricción hacemos lo mismo pero seleccionando los arcos del conjunto Arcos2 cuyo primer valor es mayor que el segundo:

@FOR(Arcos2(i,j)|i#GT#j:X(i,j)<=CS@SUM(Arcos1(s,k)|s#EQ#j #AND# k#EQ#i:Seleccion(s,k)));*

Al corresponder la variable Selección, exclusivamente a los arcos del conjunto Arcos1, y tener todos estos el primer valor menos que el segundo, tenemos que hacer una modificación en los índices de la variable selección, de forma que los cambiemos de posición y sigan así correspondiéndose con los del conjunto Arcos1.

Con esto conseguimos abarcar la totalidad de los arcos y evitar la problemática de que la variable Selección sólo se asociara a la mitad de arcos del grafo.

Por último la restricción que fuerza a que, del nodo raíz, salga una cantidad de flujo igual al número de terminales menos uno o, lo que es lo mismo, igual a la cota superior (CS).

@FOR(Nodos(i)|R(i)#EQ#1:@SUM(Arcos2(i,j):X(i,j))=CS);

Con la función @FOR barremos la totalidad de los nodos del grafo buscando aquel que cumple que R=1.

Una vez lo tenemos sumamos los flujos de salida de este nodo y forzamos a que el total sea igual a la cota superior.

Para concluir la codificación en Lingo hay que comunicarle al programa que hemos acabado de codificar:

END

A continuación se muestra, a modo de ejemplo, cómo quedaría la codificación en Lingo para la estrategia de flujo múltiple con el problema steinb1.txt, el cual es el más básico de todos los de la relación de problemas y puede resultar fácil y rápida la identificación y casación de los valores del enunciado con los del código Lingo.

Primero mostramos el enunciado:

```
13 19
1 5 7
3 9 1
3 1 7
4 8 2
6 4 2
6 5 2
6 3 8
8 10 4
9 2 8
9 7 5
10 2 6
11 1 8
11 2 14
12 6 8
12 10 9
12 11 2
12 13 7
12 7 15
13 9 11
8
4 5 6 7 8 10 11 13
 41
```

Al haber explicado anteriormente qué valores son los que marca cada fila del documento de texto, se ha considerado innecesario volverlo a reflejar en la imagen anterior.

Ante cualquier duda se recomienda revisar la Fig. 15.

A continuación se muestra el código de Lingo completo para el problema steinb1.txt que se corresponde con el enunciado anterior.

Los pasos a seguir para la realización de dicho código son los explicados con detalle en este mismo capítulo.

```
MODEL:
SETS:
Nodos/1..13/:Ti,R;
Arcos1(Nodos,Nodos)/1 5,3 9,1 3,4 8,4 6,5 6,3 6,8 10,2 9,7 9,2 10,1 11,2 11,6 12,10 12,11 12,12 13,7 12,9 13/:C,Seleccion;
Arcos2(Nodos,Nodos)/5 1,1 5,9 3,3 9,1 3,3 1,8 4,4 8,4 6,6 4,5 6,6 5,3 6,6 3,10 8,8 10,2 9,9 2,7 9,9 7,2 10,10 2,1 11,11 1
,2 11,11 2,6 12,12 6,10 12,12 10,11 12,12 11,13 12,12 13,7 12,12 7,9 13,13 9/:X;
ENDSETS
DATA:
Ti=0,0,0,1,1,1,1,1,0,1,1,0,1;
R=0,0,0,1,0,0,0,0,0,0,0,0,0;
C=7,1,7,2,2,2,8,4,8,5,6,8,14,8,9,2,7,15,11;
CS=7;
ENDDATA
MIN=41+@SUM(Arcos1(i,j):Seleccion(i,j)*C(i,j));
@FOR(Arcos1(i,j):@BIN(Seleccion(i,j)));
@FOR(Nodos(i)|Ti(i)#EQ#1 #AND# R(i)#NE#1:@SUM(Arcos2(i,k)|R(k)#NE#1:X(i,k))=@SUM(Arcos2(j,i):X(j,i))-1);
@FOR(Nodos(i)|Ti(i)#EQ#0:@SUM(Arcos2(k,i):X(k,i))=@SUM(Arcos2(i,j):X(i,j)));
@FOR(Arcos2(i,j)|i#LT#j:X(i,j)<=CS*@SUM(Arcos1(s,k)|s#EQ#i #AND# k#EQ#j:Seleccion(s,k)));
@FOR(Arcos2(i,j)|i#GT#j:X(i,j)<=CS*@SUM(Arcos1(s,k)|s#EQ#j #AND# k#EQ#i:Seleccion(s,k)));
@FOR(Nodos(i)|R(i)#EQ#1:@SUM(Arcos2(i,j):X(i,j))=CS);
END
```

Fig.16 Código de Lingo Steinb1 Estrategia Flujo Múltiple

Fuente: Elaboración Propia

Todos los códigos de cada uno de los problemas que se han resuelto en este trabajo siguen el mismo patrón que el que se muestra en la imagen anterior.

La longitud del código dependerá en gran parte del número de arcos, así también como del número de nodos, pero en menor medida.

CODIFICACIÓN PARA ESTRATEGIA DE FLUJO SIMPLE

El código para la estrategia de flujo simple es más complejo de comprender, y bastante más largo que el simple y, es por ello que parece mejor proceder a explicar esta estrategia ahora, una vez comprendido el código del múltiple.

Vamos a volver a utilizar como ejemplo el enunciado que se mostró en la Fig. 15.

Suponiendo que ya sabemos que valores representan los recogidos en cada fila de los archivos de texto del enunciado, procedemos a repetir dicha explicación y vamos directos al código de Lingo.

Como en todo código de Lingo procedemos a indicar al programa que vamos a escribir un modelo que ha de resolver.

Le indicamos también que vamos a definir una seria de conjuntos los cuales participan en el problema y que sabemos que se indican con el SETS.

Definimos entonces el primer conjunto que va a representar a los nodos, de la misma forma que se hico en el modelo de estrategia de flujo múltiple:

MODEL:

SETS:

Nodos/1..9/:Ti,R;

Volvemos a definir Nodos, diciendo que es un conjunto de recorrido 1 a 9 y posteriormente le asignamos las variables asociadas a estos, Ti y R, exactamente igual que en el múltiple.

- Ti nos va a indicar cuándo el nodo es terminal. Es un binario que valdrá 1 si es terminal y 0 si es nodo Steiner, tal y como se indicaba en la tabla de elementos participantes en el problema.

- R nos indicará cuál es el nodo raíz, es decir, el nodo por el cual se enviará el flujo total que circule por el grafo. Vuelve a ser un binario que valdrá 1 sola y exclusivamente para el nodo raíz.

Vamos con el siguiente conjunto, el de los arcos, y que en este caso se llamará Arcos, pues no es necesario diferenciar dicho conjunto con otro que representa lo mismo.

Volvemos a definir cada arco con origen en el nodo menor y destino en el nodo mayor, para la realización de un ajuste posterior con el fin de poder resolver el modelo con Lingo.

Arcos(Nodos,Nodos)/1 2,2 4,2 5,…/ :C,Seleccion;

Vemos que se vuelven a definir las mismas variables que en el conjunto Arcos1 del apartado anterior:

- C, es el coste que se suma al usar ese arco. Cada arco tiene un coste asociado. Si pasa flujo por éste, se añade el coste del mismo.

- Selección, corresponde a la variable binaria alpha que se explicó en el capítulo donde se hablaba de la al detalle de las estrategias de flujo simple y múltiple. Dicha variable vale 1 si el arco se usa, es decir, si pasa flujo por éste, y vale 0 si no. La condición vale para el arco en ambos sentidos, es decir, si se usa el arco (2,1), el valor de Selección(1,2) será el que pase a valer uno.

Acto seguido definimos un nuevo conjunto que en este caso se va a llamar Conj(Nodos,Nodos,Nodos), donde el primer nodo que aparece corresponde, en X_{ik}^{k} , al valor i, el segundo nodo corresponde al valor j y el tercero al valor k:

Conj(Nodos,Nodos,Nodos)/2 1 1,1 2 1,2 1 2,…/:X;

ENDSETS

Como ya se explicó en su momento, los valores de este conjunto indican el flujo que va desde i a j pasando por k.

Pasamos a la sección DATA, donde fijaremos esta vez los valores siguientes:

1. El valor de Ti para cada nodo

2. El valor de R para cada nodo

3. El valor de C para cada arco

La regla a seguir vuelve a ser la misma que se usó en el caso anterior.

Todos estos datos se van a obtener de los enunciados, excepto el valor de R, que lo fijaremos nosotros mismos para todos los problemas.

R será siempre el nodo terminal con el subíndice más pequeño. Esto quiere decir, si tenemos al nodo 1 como terminal, éste será el nodo raíz siempre, pero si el nodo 1 es Steiner, el nodo raíz será el siguiente siempre que sea terminal. Si el nodo 2 es terminal, en este caso sería el raíz, pero si no lo fuera tendríamos que irnos al nodo 3 y así sucesivamente.

DATA:

Ti=1,1,1,1,0,1,1,0,1;

R=0,1,0,0,0,0,0,0,0;

C=2,1,3,2,1,3,2,3,1,3,1,2,3;

ENDDATA

Como se puede ver, las variables Ti y R tienen tantos valores como nodos haya, y cada valor está ordenado de izquierda a derecha. El valor de la izquierda corresponde al primer nodo, y el de la derecha al último.

Lo mismo ocurre con los valores de C.

En este caso no es necesario el uso de una cota superior para la resolución del problema, como bien se pudo observar en el modelo matemático, por lo tanto nos lo ahorramos y directamente cerramos la sección DATA con el correspondiente ENDDATA.

Justo después colocamos la función objetivo.

Ésta vuelve a ser exactamente la misma que la anterior, lo que es totalmente lógico ya que busca el mismo fin en cada problema.

*MIN=10+@SUM(Arcos(i,j):Seleccion(i,j)*C(i,j));*

La primera restricción del modelo define de nuevo a la variable Selección(i,j) como un binario. A cada arco del conjunto Arcos le corresponde una variable binaria Seleccion:

@FOR(Arcos(i,j):@BIN(Seleccion(i,j)));

La siguiente restricción es muy parecida a la restricción de balance de flujo que se fijó en el apartado anterior, pero con la diferencia de que ahora tenemos un nuevo conjunto:

@FOR(Nodos(k)|R(k)#NE#1 #AND# Ti(k)#EQ#1:@FOR(Nodos(s)|s#NE#k #AND# R(s)#EQ#0:@SUM(Conj(i,s,k):X(i,s,k))=@SUM(Conj(s,i,k):X(s,i,k))));

En este caso se indica que todo lo que llegue al nodo *s* pasando por *k* tiene que ser lo mismo que salga de ese mismo nodo pasando por *k.*

Hay que recalcar que el nodo k no puede ser el nodo raíz al igual que el nodo s. Así mismo, el nodo k también ha de ser terminal.

En la siguiente restricción nos centramos en el nodo raíz, al que hemos llamado “rr”. Aquí se busca que el nodo raíz envíe una unidad de flujo a cada nodo terminal. Por lo tanto, la suma de todas las unidades de flujo que se envían desde el nodo raíz deben ser el número de terminales menos uno. Este uno es el mismo nodo raíz que, como es lógico, no se va a enviar una unidad a sí mismo.

@FOR(Nodos(k)|R(k)#NE#1 #AND# Ti(k)#EQ#1:@FOR(Nodos(rr)|R(rr)#EQ#1: @SUM(Conj(i,rr,k):X(i,rr,k))+1=@SUM(Conj(rr,i,k):X(rr,i,k))));

Si leemos el código podemos ver que indica que la suma de todas las unidades de flujo que salen desde el nodo raíz pasando por k, menos la suma de todas las unidades que llegan al nodo raíz pasando por k, tiene que ser uno.

Esto implica que se envíe una unidad a cada nodo terminal.

Las dos últimas restricciones fuerzan a que, si no se activa el arco, no pase ninguna unidad de flujo por él, tanto para un sentido del arco (primera restricción) como para otro (segunda restricción).

@FOR(Arcos(i,j)|i#LT#j:@FOR(Nodos(k)|R(k)#EQ#0 #AND# Ti(k)#EQ#1:
@SUM(Conj(i,j,k):X(i,j,k))<=Seleccion(i,j)));

@FOR(Arcos(i,j):@FOR(Nodos(k)|R(k)#EQ#0 #AND# Ti(k)#EQ#1:
@SUM(Conj(i,j,k)|i#LT#j:X(j,i,k))<=Seleccion(i,j)));

Esto lo hace para cada arco y para cada nodo k siempre que éste sea terminal, y nunca sea el nodo raíz.

Para cerrar el modelo, como siempre tenemos que indicárselo a Lingo con el correspondiente END:

END

A continuación se muestra, a modo de ejemplo, cómo quedaría la codificación en Lingo para la estrategia de flujo simple con el problema steinb1.txt.

Para seguir el modelo de la resolución con su correspondiente enunciado véase el enunciado del problema steinb1.txt.

Los pasos a seguir para la realización de dicho código son los explicados con detalle en este mismo capítulo.

```
MODEL:
SETS:
Nodos/1..13/:Ti,R;
Arcos(Nodos,Nodos)/1 5,3 9,1 3,4 8,4 6,5 6,3 6,8 10,2 9,7 9,2 10,1 11,2 11,6 12,10 12,11 12,12 13,7 12,9 13/:C,Seleccion;
Conj(Nodos,Nodos,Nodos)/5 1 4,1 5 4,5 1 5,1 5 5,5 1 6,1 5 6,5 1 7,1 5 7,5 1 8,1 5 8,5 1 10,1 5 10,5 1 11,1 5 11,5 1 13,1 5 13,9 3 4,
3 9 4,9 3 5,3 9 5,9 3 6,3 9 6,9 3 7,3 9 7,9 3 8,3 9 8,9 3 10,3 9 10,9 3 11,3 9 11,9 3 13,3 9 13,1 3 4,3 1 4,1 3 5,3 1 5,1 3 6,3 1 6,
1 3 7,3 1 7,1 3 8,3 1 8,1 3 10,3 1 10,1 3 11,3 1 11,1 3 13,3 1 13,8 4 4,4 8 4,8 4 5,4 8 5,8 4 6,4 8 6,8 4 7,4 8 7,8 4 8,4 8 8,8 4 10,
4 8 10,8 4 11,4 8 11,8 4 13,4 8 13,4 6 4,6 4 4,4 6 5,6 4 5,4 6 6,6 4 6,4 6 7,6 4 7,4 6 8,6 4 8,4 6 10,6 4 10,4 6 11,6 4 11,4 6 13,
6 4 13,5 6 4,6 5 4,5 6 5,6 5 5,5 6 6,6 5 6,5 6 7,6 5 7,5 6 8,6 5 8,5 6 10,6 5 10,5 6 11,6 5 11,5 6 13,6 5 13,3 6 4,6 3 4,3 6 5,6 3 5,
3 6 6,6 3 6,3 6 7,6 3 7,3 6 8,6 3 8,3 6 10,6 3 10,3 6 11,6 3 11,3 6 13,6 3 13,10 8 4,8 10 4,10 8 5,8 10 5,10 8 6,8 10 6,10 8 7,8 10 7,
10 8 8,8 10 8,10 8 10,8 10 10,10 8 11,8 10 11,10 8 13,8 10 13,2 9 4,9 2 4,2 9 5,9 2 5,2 9 6,9 2 6,2 9 7,9 2 7,2 9 8,9 2 8,2 9 10,9 2 10,
2 9 11,9 2 11,2 9 13,9 2 13,7 9 4,9 7 4,7 9 5,9 7 5,7 9 6,9 7 6,7 9 7,9 7 7,7 9 8,9 7 8,7 9 10,9 7 10,7 9 11,9 7 11,7 9 13,9 7 13,
2 10 4,10 2 4,2 10 5,10 2 5,2 10 6,10 2 6,2 10 7,10 2 7,2 10 8,10 2 8,2 10 10,10 2 10,2 10 11,10 2 11,2 10 13,10 2 13,1 11 4,11 1 4,
1 11 5,11 1 5,1 11 6,11 1 6,1 11 7,11 1 7,1 11 8,11 1 8,1 11 10,11 1 10,1 11 11,11 1 11,1 11 13,11 1 13,2 11 4,11 2 4,2 11 5,11 2 5,
2 11 6,11 2 6,2 11 7,11 2 7,2 11 8,11 2 8,2 11 10,11 2 10,2 11 11,11 2 11,2 11 13,11 2 13,6 12 4,12 6 4,6 12 5,12 6 5,6 12 6,12 6 6,
6 12 7,12 6 7,6 12 8,12 6 8,6 12 10,12 6 10,6 12 11,12 6 11,6 12 13,12 6 13,10 12 4,12 10 4,10 12 5,12 10 5,10 12 6,12 10 6,10 12 7,
12 10 7,10 12 8,12 10 8,10 12 10,12 10 10,10 12 11,12 10 11,10 12 13,12 10 13,11 12 4,12 11 4,11 12 5,12 11 5,11 12 6,12 11 6,11 12 7,
12 11 7,11 12 8,12 11 8,11 12 10,12 11 10,11 12 11,12 11 11,11 12 13,12 11 13,13 12 4,12 13 4,13 12 5,12 13 5,13 12 6,12 13 6,13 12 7,
12 13 7,13 12 8,12 13 8,13 12 10,12 13 10,13 12 11,12 13 11,13 12 13,12 13 13,7 12 4,12 7 4,7 12 5,12 7 5,7 12 6,12 7 6,7 12 7,12 7 7,
7 12 8,12 7 8,7 12 10,12 7 10,7 12 11,12 7 11,7 12 13,12 7 13,9 13 4,13 9 4,9 13 5,13 9 5,9 13 6,13 9 6,9 13 7,13 9 7,9 13 8,13 9 8,
9 13 10,13 9 10,9 13 11,13 9 11,9 13 13,13 9 13/:X;
ENDSETS
DATA:
Ti=0,0,0,1,1,1,1,1,0,1,1,0,1;
R=0,0,0,1,0,0,0,0,0,0,0,0,0;
C=7,1,7,2,2,2,8,4,8,5,6,8,14,8,9,2,7,15,11;
ENDDATA
MIN=41+@SUM(Arcos(i,j):Seleccion(i,j)*C(i,j));
@FOR(Arcos(i,j):@BIN(Seleccion(i,j)));
@FOR(Nodos(k)|R(k)#NE#1 #AND# Ti(k)#EQ#1:@FOR(Nodos(s)|s#NE#k #AND# R(s)#EQ#0:@SUM(Conj(i,s,k):X(i,s,k))=@SUM(Conj(s,i,k):X(s,i,k))));
@FOR(Nodos(k)|R(k)#NE#1 #AND# Ti(k)#EQ#1:@FOR(Nodos(rr)|R(rr)#EQ#1:@SUM(Conj(i,rr,k):X(i,rr,k))+1=@SUM(Conj(rr,i,k):X(rr,i,k))));
@FOR(Arcos(i,j)|i#LT#j:@FOR(Nodos(k)|R(k)#EQ#0 #AND# Ti(k)#EQ#1:@SUM(Conj(i,j,k):X(i,j,k))<=Seleccion(i,j)));
@FOR(Arcos(i,j):@FOR(Nodos(k)|R(k)#EQ#0 #AND# Ti(k)#EQ#1:@SUM(Conj(i,j,k)|i#LT#j:X(j,i,k))<=Seleccion(i,j)));
END
```

Fig.17 Código de Lingo Steinb1 Estrategia Flujo Simple

Fuente: Elaboración Propia

La longitud del código dependerá en gran parte del número de arcos, así también como del número de nodos, pero en menor medida.

En este caso, estamos ante el código Lingo más corto para la formulación de flujo simple.

La problemática viene con el conjunto “Conj”. Al incorporar este todos los arcos en ambos sentidos y pasando por todos los nodos terminales, se obtiene un conjunto bastante pesado.

Si tenemos en cuenta un problema con el doble de nodos terminales, tendríamos el doble de valores para dicha variable, y eso que sólo estaríamos hablando de un problema con 16 nodos terminales.

EXPERIMENTACIÓN

Los experimentos realizados tienen como fin la comparación de ambas estrategias, la estrategia de flujo simple y la estrategia de flujo múltiple.

Cada experimento se basa en la resolución de las dos primeras tandas de problemas de Steiner (steinbXX y steincXX) planteados en la OR library [7] mantenida por J.Beasley, en la página web del departamento de investigación operativa, del Imperial College de Londres.

Estos problemas se clasifican en función de su complejidad y dimensión, en las clases "b","c","d" y "e", siendo estos últimos los más complejos y por consiguiente los que mayor tiempo de resolución requerirán.

En este trabajo sólo se experimentará con los problemas de clase "b" y "c", pues eran los que requerían de un tiempo de resolución que hacía posible la interpretación de la mayor parte de los problemas de forma total.

En la siguiente tabla se puede observar la lista de problemas que se van a someter a estudio, así como algunos parámetros del problema: Nº de nodos, nº de arcos y nº de nodos terminales.

Problema	Nº Nodos	Nº Arcos	Nº Terminales
Steinb1	13	19	8
Steinb2	15	21	11
Steinb3	20	25	15
Steinb4	40	80	9
Steinb5	39	80	12
Steinb6	45	87	25
Steinb7	22	33	11
Steinb8	26	38	15
Steinb9	27	35	23
Steinb10	55	121	13
Steinb11	63	129	19
Steinb12	63	125	36
Steinb13	36	56	14
Steinb14	42	65	21
Steinb15	48	69	38
Steinb16	77	166	17
Steinb17	74	153	23
Steinb18	82	166	45
Steinc1	143	260	5
Steinc2	128	234	10
Steinc3	178	295	75
Steinc4	193	314	102
Steinc5	223	341	180
Steinc6	366	837	5
Steinc7	383	866	10
Steinc8	387	867	79
Steinc9	418	903	124
Steinc10	427	891	242

Steinc11	499	2005	5
Steinc12	499	2065	10
Steinc13	498	2026	83
Steinc14	499	1968	125
Steinc15	500	1815	250
Steinc16	500	3517	5
Steinc17	500	3463	10
Steinc18	500	3496	83
Steinc19	500	3352	125
Steinc20	500	3121	250

Tabla 4. Datos Enunciados Problemas Steiner

Fuente: Elaboración Propia

A continuación se presenta un gráfico en el que se representa el número de nodos para los problemas steinb, de manera que podamos apreciar la evolución del número de estos a medida que avanzamos de problema:

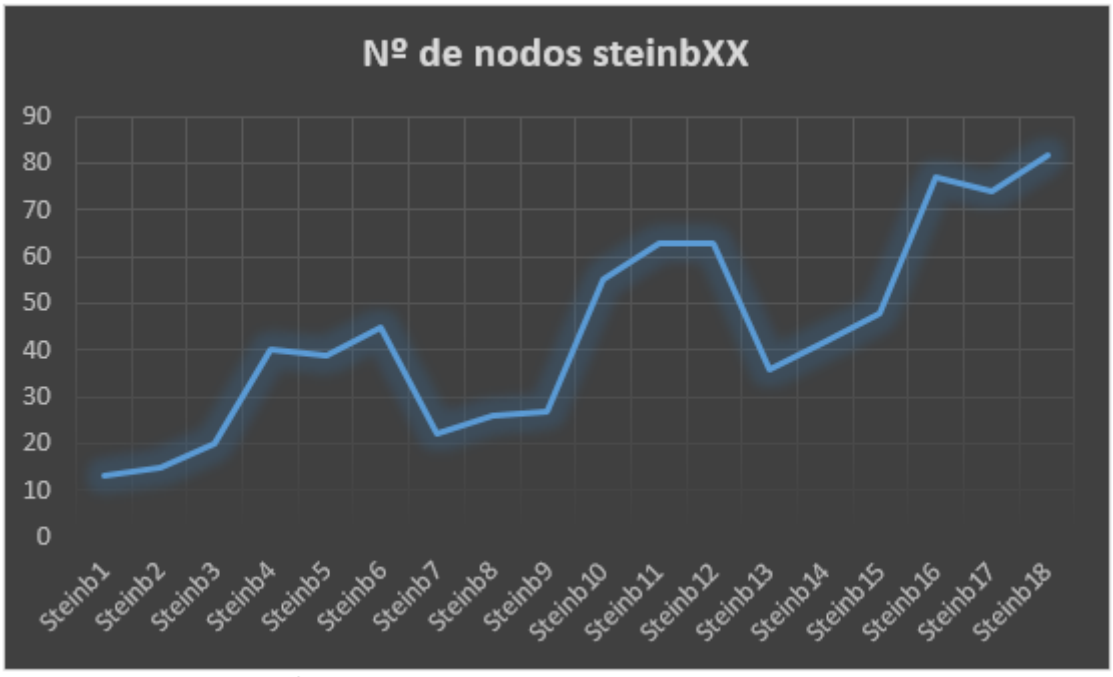


Fig.18 Número de Nodos Steinb

Fuente: Elaboración Propia

Se puede apreciar que el número de nodos crece y decrece con un intervalo de seis, esto quiere decir que el número de nodos crece progresivamente hasta llegar al steinb6 y a partir de ahí decrece cuando pasamos al steinb7, entonces vuelve a crecer hasta el steinb12 y así sucesivamente.

Esto veremos posteriormente si influye o no a la hora de resolver los problemas.

El siguiente gráfico refleja el número de nodos de los problemas del grupo “c”. Se prevé que la cantidad de estos aumente, pues es sabido que la dificultad de los problemas del grupo “c” es mayor que la de los del grupo “b”.

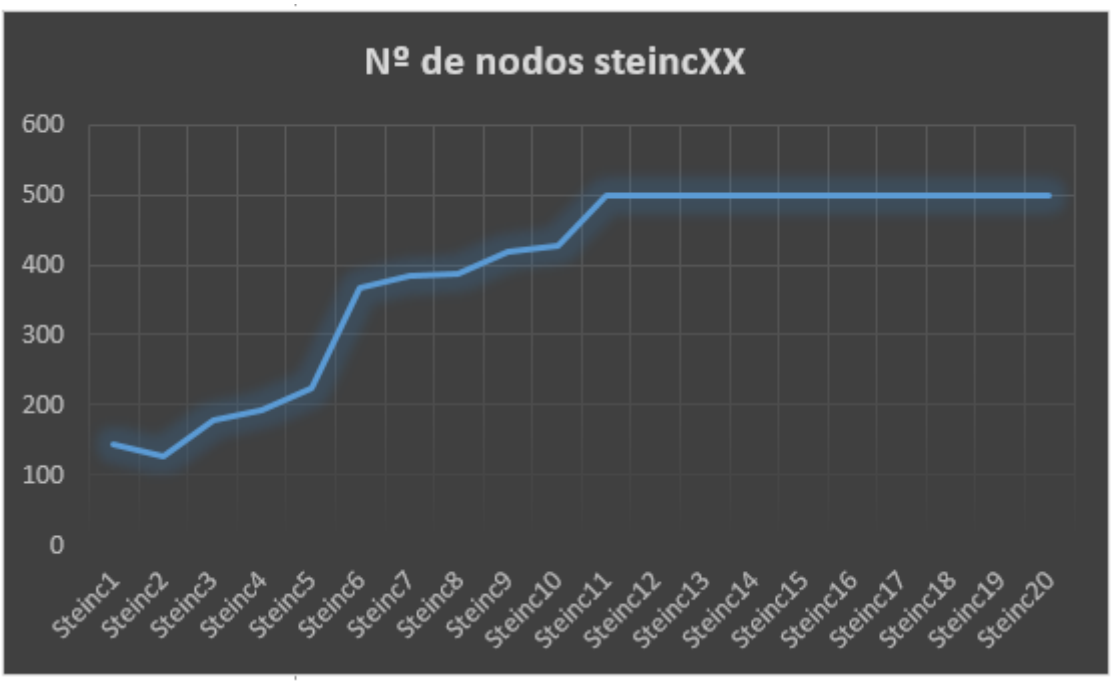


Fig.19 Número de Nodos Steinc

Fuente: Elaboración Propia

Se puede apreciar que, en este caso, no existe una oscilación en el número de nodos conforme avanzamos de problema.

La cantidad de nodos es mayor ya en el steinc1 que en el steinb18, lo que nos hace pensar que el problema previsiblemente más fácil del grupo “c” va a llevar un tiempo de resolución mayor que el problema más difícil del grupo “b”.

Todo esto son especulaciones que posteriormente se pondrán a prueba con los datos obtenidos experimentalmente.

A continuación se muestra el gráfico donde se representa el número de arcos de los problemas del grupo “b”:

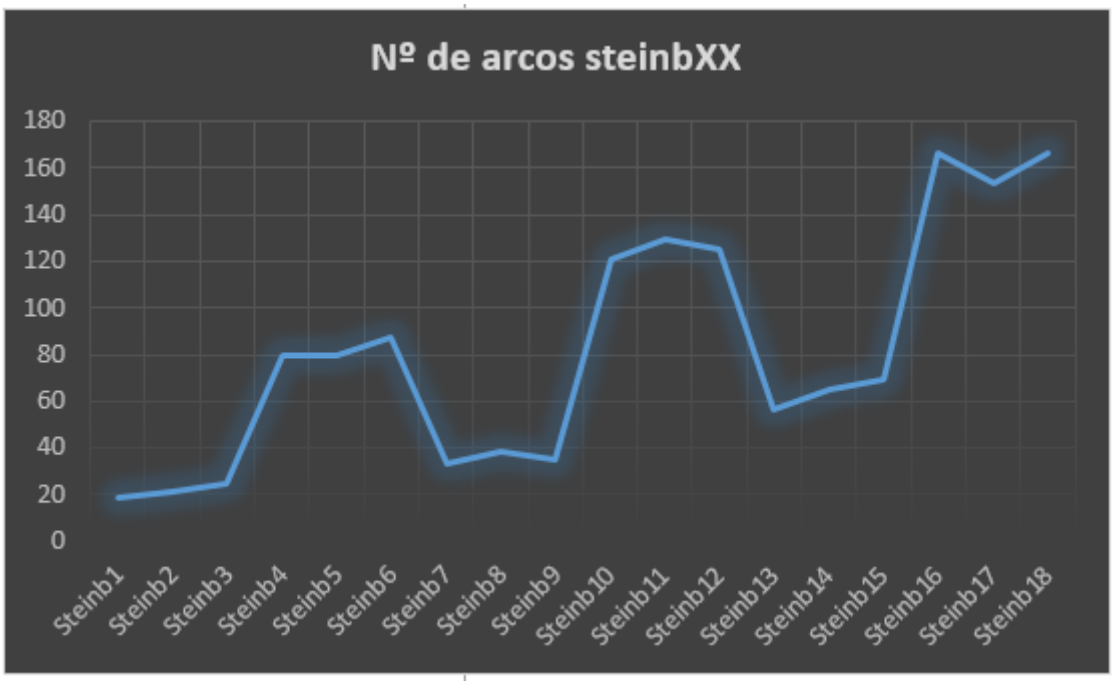


Fig.20 Número de Arcos Steinb

Fuente: Elaboración Propia

Se puede observar un gráfico muy parecido al de los nodos del grupo “b”.

Esto hace pensar que los problemas que más tiempo de resolución van a requerir son el steinb6, el steinb12 y el steinb18, pues son los que acumulan mayor número de nodos y arcos.

En la siguiente gráfica se refleja el número de arcos de los problemas del grupo “c” que, como es de esperar, serán mayores que los del grupo “b”:

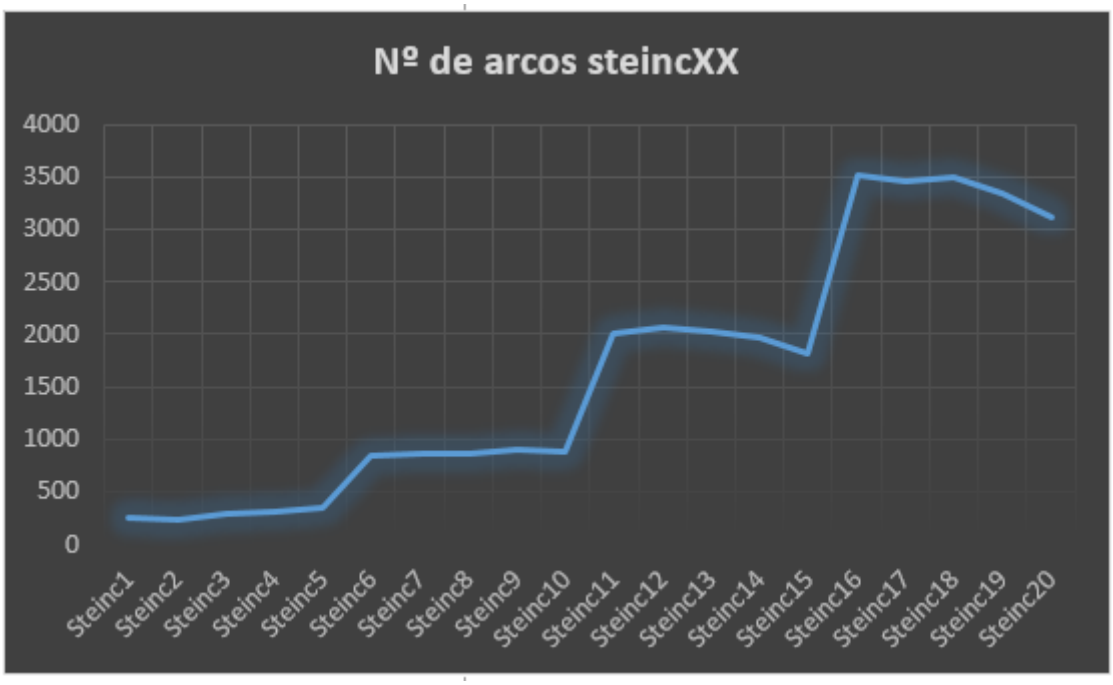


Fig.21 Número de Arcos Steinc

Fuente: Elaboración Propia

Resulta llamativo que el número de arcos del problema que teóricamente debería encontrar el óptimo más rápidamente en este grupo (Steinc1), sea casi el doble que el número de arcos del problema supuestamente más lento de solucionar del grupo “b” (Steinb18).

El gráfico presenta una forma escalonada, donde la subida vuelve a producirse en los steinc6, steinc12 y steinc18 aproximadamente.

Después de dichas subidas el número se mantiene estable en los 4 problemas siguientes pero acto seguido se incrementa en casi el doble de lo que había en el problema anterior.

Analizando las gráficas anteriores y comparando los del grupo “b” y “c” parece que los problemas del último grupo pueden llevar un tiempo de resolución considerablemente superior a los del grupo “b”.

Se presenta, a continuación, un gráfico que representa la evolución del número de nodos terminales conforme avanzamos en los problemas del grupo “b”:

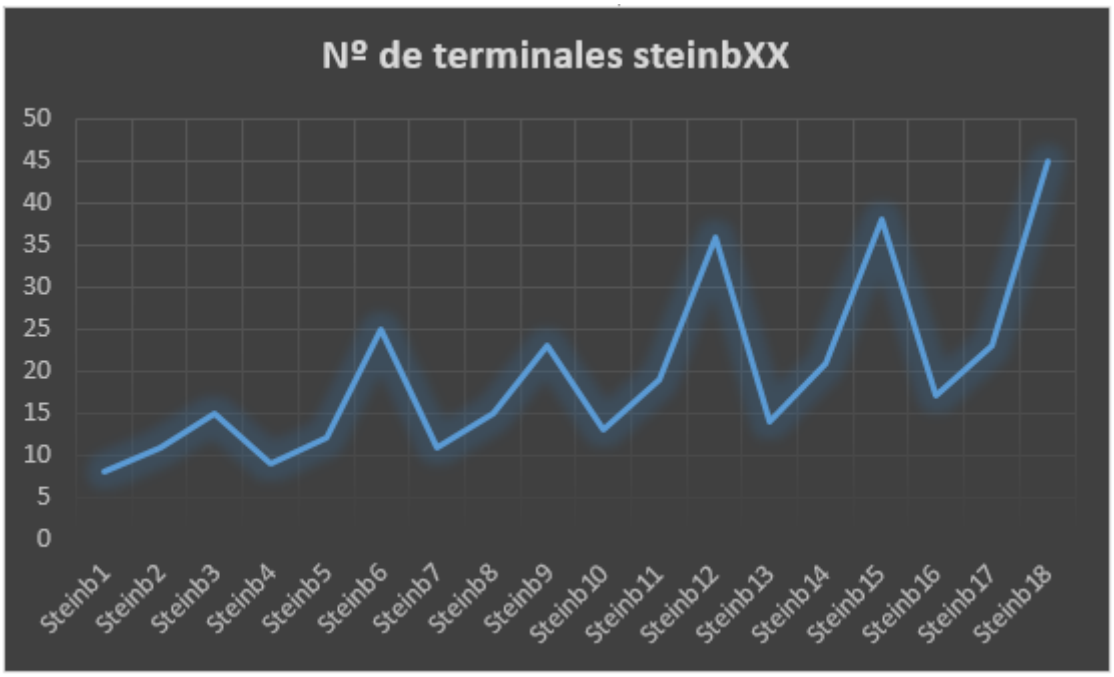


Fig.22 Número de Nodos Terminales Steinb

Fuente: Elaboración Propia

Se comprueba que se vuelven a producir picos en los steinb6, steinb12 y steinb18.

Parece evidente que serán estos los problemas que necesiten un mayor tiempo de proceso para encontrar su solución óptima.

Por lo demás, se ve que se forma un gráfico en forma de dientes de sierra, que aumentan el tamaño cada dos y, acto seguido, reducen el tamaño en los siguientes dos problemas.

Se muestra ahora la gráfica para el número de nodos terminales en los problemas del grupo “c”:

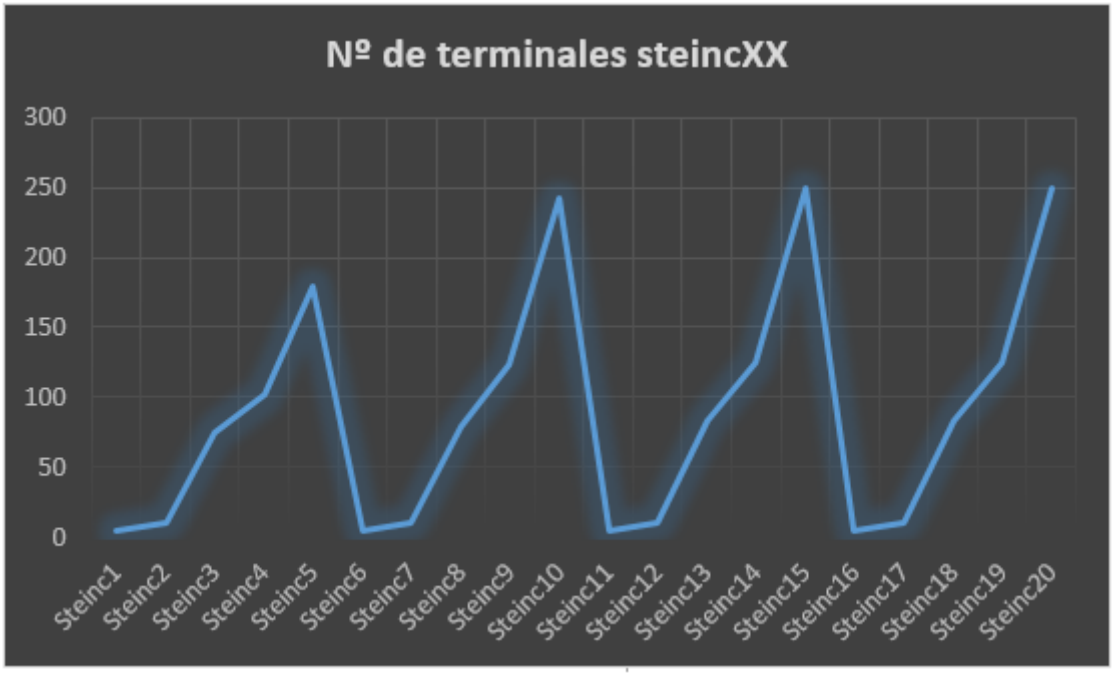


Fig.23 Número de Nodos Terminales Steinc

Fuente: Elaboración Propia

Volvemos a tener un gráficos con dientes de sierra, aunque ahora los problemas donde aparecen los picos no son los mismos.

La previsión del tiempo de resolución de estos problemas se hace ahora más complicada, debido a que los problemas que tienen mayor número de nodos y de arcos no coinciden con los problemas que tienen mayor número de nodos terminales.

El análisis de estos problemas del grupo “c” puede ser clave para determinar si es el número de nodos terminales, o el número de arcos y nodos el factor determinante a la hora de medir el tiempo de resolución de un problema.

HIPÓTESIS INICIALES

Después de haber analizado todos los enunciados y haber podido comparar los problemas uno a uno, sabemos cuáles tienen un mayor número de arcos y nodos totales, así como cuáles tienen un número mayor de nodos terminales y, por consiguiente, menor de nodos Steiner.

Como concluimos en el capítulo donde se presentaron los modelos de lingo para la resolución de los problemas con estrategias tanto de flujo simple como de flujo múltiple, en ambos el número total de nodos y de arcos era determinante para que el código del problema tuviera una longitud mayor o menor.

Pero era en el código de la estrategia de flujo simple donde parecía crucial el número de nodos terminales, pues el conjunto "Conj" crecía muy notablemente conforme aumentaba el número de nodos de estas características.

Parece ser, entonces, que el tiempo de resolución para los problemas con estrategia de flujo múltiple dependerá, en gran parte, del número total de nodos y arcos del problema, y puede que, en menor medida, del número de nodos terminales.

Sin embargo, el tiempo de resolución para los problemas con estrategia de flujo simple, parece ser que va de la mano junto con el número de nodos terminales. Todo esto considerando que, a mayor longitud del código de lingo, mayor tiempo de resolución del problema.

Se considera conveniente recalcar, de nuevo, que todo lo anteriormente expuesto son meras hipótesis, que serán finalmente comparadas con los resultados computacionales.

RESULTADOS COMPUTACIONALES

Para la comparación de ambas estrategias se han realizado experimentos que consisten en resolver los modelos de Lingo obtenidos con el programa en C, dejando la aplicación corriendo durante aproximadamente media hora.

Se decidió acotar el tiempo cuando, al intentar obtener el óptimo con Lingo del problema steinb6.txt, éste aún seguía iterando pasadas las 6 horas y media. Esto llevó a concluir que no sería posible realizar un experimento si intentábamos obtener el óptimo en todos los problemas, debido a que no tendríamos datos suficientes para comparar.

Para solucionar esta problemática se definió un indicador que evaluaba el error entre la solución obtenida pasada media hora de ejecución con Lingo, y la solución óptima del problema.

Estas soluciones óptimas son datos obtenidos de experimentos anteriores ajenos a este trabajo.

Con el indicador podemos saber cuán alejada del óptimo está nuestra solución, lo que a la par da una idea, aproximadamente, del tiempo restante para que Lingo encuentre la solución óptima.

Por lo tanto se deduce que, a mayor error entre la solución obtenida y el óptimo, mayor tiempo de resolución del problema y, por consiguiente, la estrategia utilizada para dicha resolución nos resultará menos favorable.

En las tablas que se muestran a continuación se recogen los siguientes datos:

- Nombre del problema a resolver, con título ”Modelo” en la columna
- Número de iteraciones realizadas
- Tiempo empleado en el proceso de resolución (alrededor de 30 minutos)
- Steps o pasos realizados
- Número de variables existentes en dicho problema
- Número de restricciones consecuencia del desarrollo del problema
- Memoria usada para resolver el problema
- Solución obtenida en treinta minutos de ejecución del Lingo
- Óptimo del problema

- Error en porcentaje entre el óptimo y la solución obtenida

El error se ha medido con la siguiente fórmula:

$$Error = \frac{Solución\ Obtenida - Óptimo}{Óptimo}$$

Todos estos datos se han recogido para los problemas del grupo “b” y “c”.

Se tomó esa decisión ya que se disponía de un tiempo limitado, por lo que se concluyó que no había tiempo suficiente para resolver problemas de los siguientes grupos.

Además, sabida de antemano la dificultad que los problemas de los siguientes grupos poseían, con media hora no iba a ser suficiente para obtener datos concluyentes con los que poder trabajar.

La tabla con los resultados obtenidos para los problemas con la estrategia de flujo múltiple se presenta a continuación:

Modelo	Iteraciones	Tiempo	Steps	Variables	Restricciones	Memory Used (k)	Solución Obtenida	Óptimo	Error
Steinb1	2407	0:00:00	148	57	52	34	82	82	0%
Steinb2	3819	0:00:00	60	63	58	35	83	83	0%
Steinb3	33390	0:00:03	592	75	71	39	138	138	0%
Steinb4	427522	0:00:22	10937	240	201	74	59	59	0%
Steinb5	1823717	0:01:58	33834	240	200	74	61	61	0%
Steinb6	24846207	0:30:58	398867	261	220	76	122	122	0%
Steinb7	63004	0:00:04	958	99	89	43	111	111	0%
Steinb8	149365	0:00:13	3663	114	103	47	104	104	0%
Steinb9	265990	0:00:18	8280	105	98	45	220	220	0%
Steinb10	17719752	0:34:10	357163	363	298	95	89	86	3%
Steinb11	19272598	0:31:45	157477	387	322	106	88	88	0%
Steinb12	19089243	0:36:39	282324	375	314	100	180	174	3%
Steinb13	212758	0:02:04	63762	168	149	60	165	165	0%
Steinb14	23693295	0:30:06	433242	195	173	64	236	235	0%
Steinb15	17183635	0:30:00	503748	207	187	68	318	318	0%
Steinb16	10907445	0:30:00	125366	498	410	129	127	127	0%
Steinb17	24731557	0:31:46	503238	459	381	120	134	131	2%
Steinb18	12717686	0:30:09	221907	498	415	129	233	218	7%
Steinc1	461544	0:01:12	2903	780	664	212	85	85	0%
Steinc2	7522469	0:33:47	60007	702	597	184	148	144	3%
Steinc3	13127440	0:30:06	154557	885	769	236	834	754	11%
Steinc4	13192245	0:31:36	123166	942	822	251	1181	1079	9%
Steinc5	11185107	0:30:08	115703	1023	906	276	1698	1579	8%
Steinc6	962155	0:30:20	26107	2511	2041	621	56	55	2%
Steinc7	4152641	0:31:02	12623	2598	2116	643	114	102	12%
Steinc8	4104355	0:30:51	19526	2601	2122	645	651	509	28%
Steinc9	6391311	0:31:19	42734	2709	2225	675	891	707	26%
Steinc10	8407215	0:30:01	49998	2673	2210	670	1365	1093	25%
Steinc11	5026574	0:30:51	32998	6015	4510	1448	33	32	3%
Steinc12	6911284	0:36:08	37092	6195	4630	1488	49	46	7%
Steinc13	3546882	0:34:02	30218	6078	4551	1462	351	258	36%
Steinc14	6400010	0:32:06	15481	5904	4436	1423	420	323	30%
Steinc15	5320214	0:30:04	18461	5445	4131	1321	662	556	19%
Steinc16	2706966	0:30:02	8475	10551	7535	2454	12	11	9%
Steinc17	3469283	0:30:02	42815	10389	7427	2418	22	18	22%
Steinc18	4097388	0:30:33	6362	10488	7493	2440	158	113	40%
Steinc19	2540591	0:30:10	4572	10056	7205	2344	210	146	44%
Steinc20	1730527	0:30:14	746	9363	6743	2190	353	267	32%

Tabla 5. Resultados Estrategia Flujo Múltiple

Fuente: Elaboración Propia

Como se puede observar, aparecen celdas de color verde en la columna de los tiempos. Esto es porque el problema asociado a ese tiempo ha sido resuelto por completo en Lingo, habiendo encontrado su óptimo.

En la columna Error aparecen en verde los valores iguales a cero, a los que les corresponde una solución obtenida igual al óptimo.

Se da el caso en que hay valores cero en dicha columna pero la celda de la columna tiempo asociada al mismo modelo no está en color verde. Esto quiere decir que la solución encontrada por Lingo era el óptimo, pero el programa aun no lo sabía ya que todavía seguía realizando iteraciones.

Si seguimos en la misma columna "Error" vemos que hay valores en rosa. Esto es porque son valores mayores que cero, pero no son muy grandes (<10%), por lo que se puede considerar como una buena solución la que hemos obtenido.

En el otro extremo están las celdas en rojo de la columna "Error". Esto ocurre cuando el valor obtenido es mayor de 10%. Estaríamos entonces ante lo que hemos tomado como una mala solución.

Según el criterio que se ha fijado, una estrategia es mejor que otra cuanto menor sea el valor del error.

A continuación se expone un gráfico que representa el valor del error entre la solución obtenida y el óptimo para los problemas del grupo "b" con la estrategia de flujo múltiple, a fin de hacer más sencilla la interpretación de los datos obtenidos:

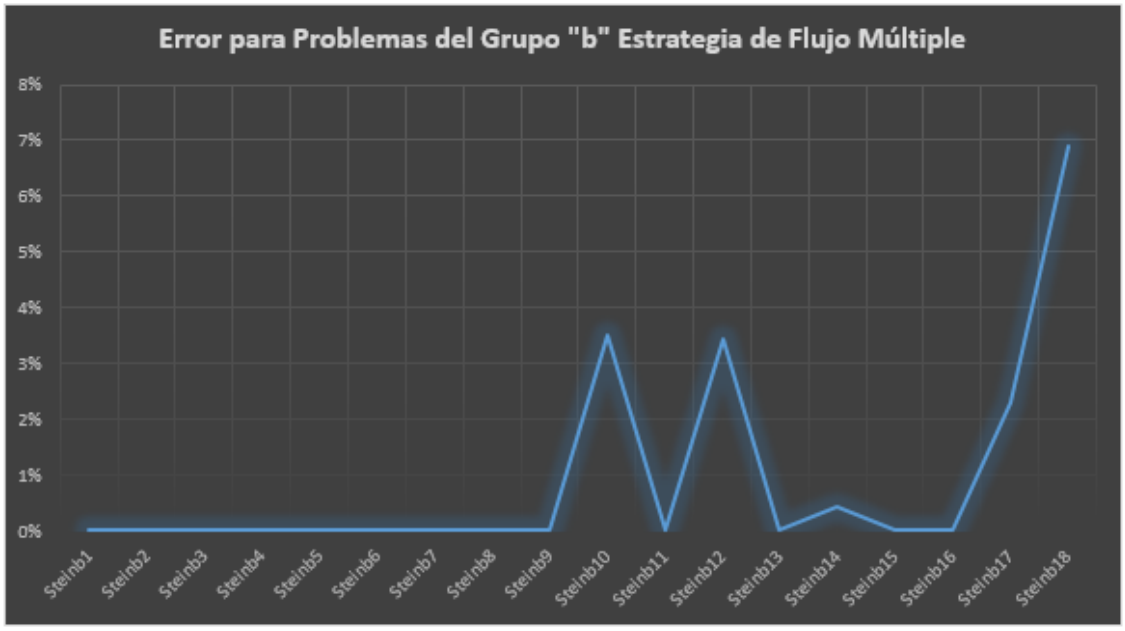


Fig.24 Error Steinb para Flujo Múltiple

Fuente: Elaboración Propia

En el múltiple, es evidente que en los nueve primeros modelos del grupo “b” resueltos con dicha estrategia, éste se comporta francamente bien, ya que todas las soluciones obtenidas son el óptimo, de ahí la línea recta pegada al cero.

Es a partir del steinb10 cuando empieza a desestabilizarse, y todo ello parece estar ligado a un aumento repentino del número de nodos y de arcos desde el steinb9 al steinb10, como se puede ver en el siguiente fragmento:

Problema	Nº Nodos	Nº Arcos	Nº Terminales
Steinb1	13	19	8
Steinb2	15	21	11
Steinb3	20	25	15
Steinb4	40	80	9
Steinb5	39	80	12
Steinb6	45	87	25
Steinb7	22	33	11
Steinb8	26	38	15
Steinb9	27	35	23
Steinb10	55	121	13
Steinb11	63	129	19
Steinb12	63	125	36

Tabla 6. Fragmento tabla 4

Fuente: Elaboración Propia

Echando la vista atrás, a la Tabla 4, podremos comprobar como las oscilaciones en el gráfico anterior parecen estar estrechamente ligadas al número total de nodos y arcos del problema.

La estrategia parece comportarse muy bien con valores bajos de nodos y arcos independientemente del número de nodos terminales que posea el problema.

Si vemos el mismo gráfico pero ahora para los problemas del grupo “c” pasa lo siguiente:

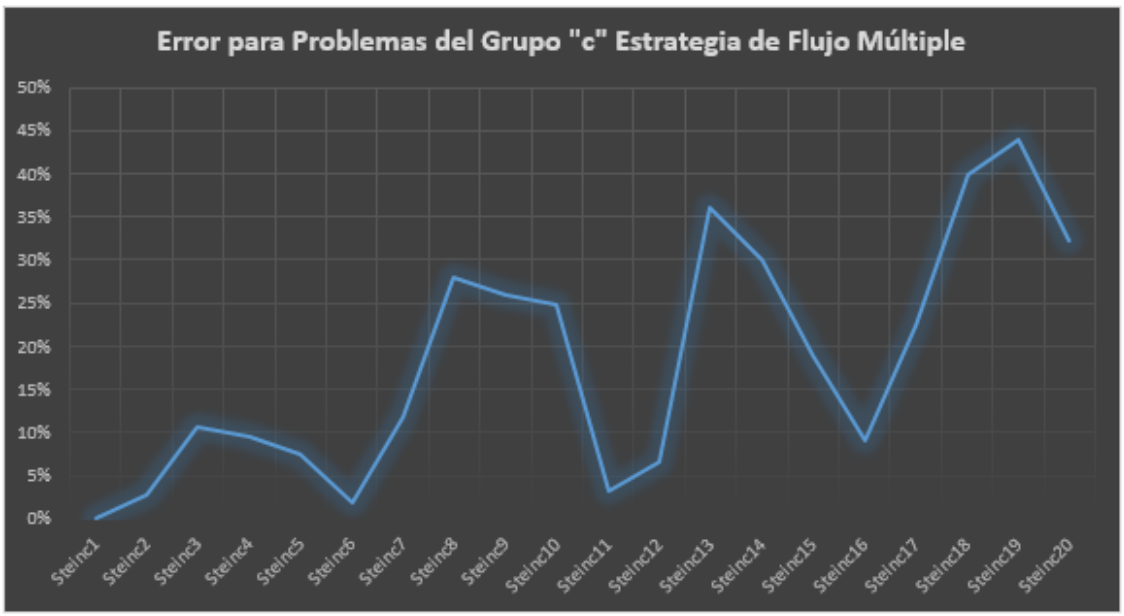


Fig.25 Error Steinc para Flujo Múltiple

Fuente: Elaboración Propia

La gráfica anterior parece seguir un ritmo de oscilaciones muy parecido al de las gráficas de las Fig. 18 y Fig. 20, pero éstas no corresponden a los problemas del grupo “c”. Sin embargo, las oscilaciones que se muestran en la gráfica anterior sí parecen estar ligadas al número de nodos terminales de problema:

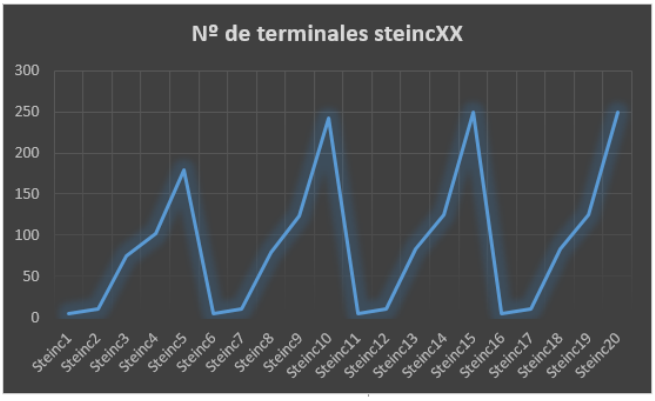


Fig.26 Número de Nodos Terminales Steinc II

Fuente: Elaboración Propia

Una vez interpretados los datos para la estrategia de flujo múltiple nos disponemos a analizar los resultados para la estrategia de flujo simple, cuya tabla de resultados se muestra a continuación:

Modelo	Iteraciones	Tiempo	Steps	Variables	Restricciones	Memory Used (k)	Solución Obtenida	Óptimo	Error
Steinb1	7822	0:00:01	20	323	351	91	82	82	0%
Steinb2	8022	0:00:02	14	483	561	133	83	83	0%
Steinb3	35792	0:00:07	50	775	967	215	138	138	0%
Steinb4	128340	0:00:31	158	1520	1593	394	59	59	0%
Steinb5	364741	0:01:28	1201	2000	2179	504	61	61	0%
Steinb6	2407562	0:31:16	2072	4437	5233	1139	135	122	11%
Steinb7	32418	0:00:08	49	759	871	205	111	111	0%
Steinb8	173842	0:00:25	257	1178	1415	319	104	104	0%
Steinb9	300745	0:00:43	456	1645	2113	453	220	220	0%
Steinb10	626855	0:05:03	367	3267	3553	817	86	86	0%
Steinb11	2465580	0:30:11	1172	5031	5761	1281	88	88	0%
Steinb12	1081057	0:30:04	554	9125	10921	2353	200	174	15%
Steinb13	990298	0:04:03	1050	1624	1912	437	165	165	0%
Steinb14	4623793	0:30:25	6115	2795	3421	732	239	235	2%
Steinb15	1486653	0:30:03	984	5313	6846	1411	332	318	4%
Steinb16	1552644	0:30:03	675	5810	6529	1468	142	127	12%
Steinb17	2014169	0:30:07	4731	7191	8339	1837	136	131	4%
Steinb18	807506	0:34:54	47	15106	18173	3906	274	218	26%

Tabla 7. Resultados Estrategia Flujo Simple

Fuente: Elaboración Propia

En este caso se puede comprobar que se han excluido los problemas del grupo “c” en este experimento. Esto es así debido a que no se encontraron soluciones en la primera media hora para aproximadamente el 75% de los problemas de este grupo, por lo que estos resultados no se consideraron susceptibles de análisis.

Los colores se interpretan siguiendo las indicaciones dadas en la Tabla 5.

A continuación se exponen un gráfico que representa el valor del error entre la solución obtenida y el óptimo para los problemas del grupo “b” con la estrategia de flujo simple, a fin de hacer más sencilla la interpretación de los datos obtenidos:

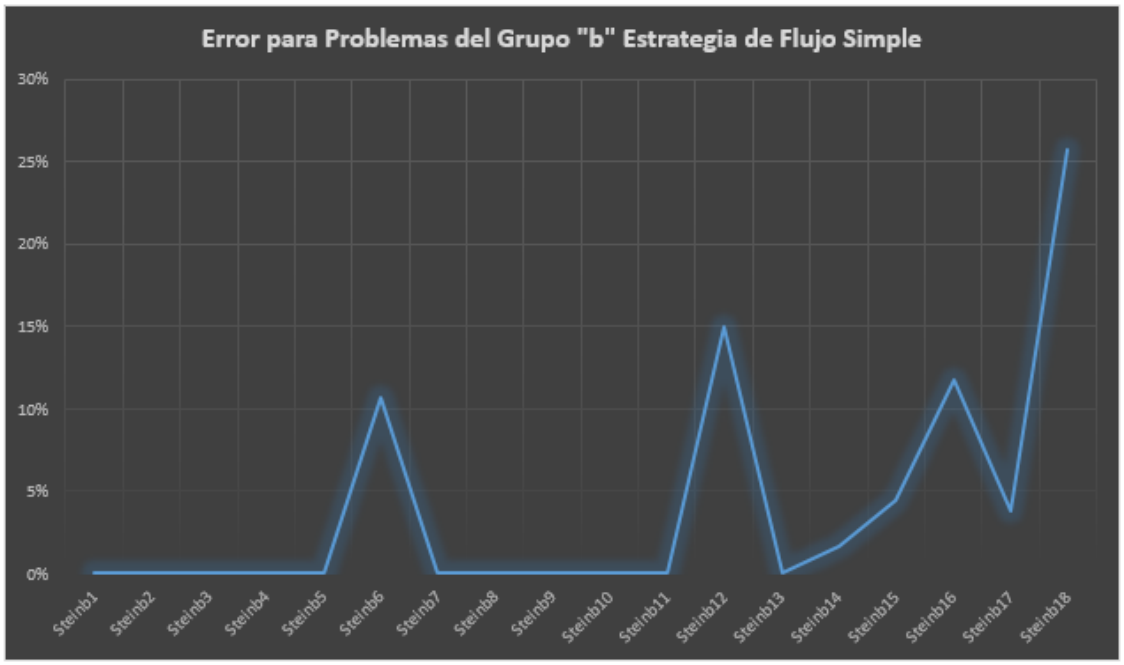


Fig.27 Error Steinb para Flujo Simple

Fuente: Elaboración Propia

Aparece un gráfico muy parecido al de la estrategia múltiple.

Aun así, esta vez el pico que aparece en el 6 antes era plano. De todas formas es difícil interpretar la gráfica en este caso, pues hay ocasiones en las que parece que una subida brusca del número de nodos terminales produzca un incremento en el error, pero en otras, como es el caso del problema 16 y 17, el problema 16 tiene un error mayor que el 17 teniendo este último más nodos terminales que el anterior.

RESULTADOS DEL PROBLEMA CONTINUO

Una muy buena forma de saber si un problema se comporta bien ante una determinada estrategia de resolución es buscar el óptimo del problema continuo o bound.

El bound es el valor de la solución del problema continuo, pero es también, al mismo tiempo, el primer valor con el que empezamos a probar en el Brach & Bound del problema entero.

Esto quiere decir que, si sabemos que el óptimo de un problema entero es 80, y el del mismo problema continuo es 10’34, este problema necesitará mucho tiempo para acercarse al óptimo, pues su bound está muy lejos de éste.

Sin embargo, si el óptimo entero es 80 y el continuo es 75’65, este problema tardará mucho menos que el anterior en encontrar el óptimo, por lo tanto, podemos considerar que se comporta mejor que el anterior **si suponemos que se ha usado la misma estrategia de resolución en ambos**.

Para calcular la solución continua de cada problema lo único que se ha tenido que hacer es modificar la restricción que fijaba el valor “Selección” como binario, y hacer que este sea continuo. El cambio es el siguiente:

- Restricción original: *@FOR(Arcos1(i,j):@BIN(Seleccion(i,j)));*
- Nueva restricción: *@FOR(Arcos1(i,j): (Seleccion(i,j)<=1));*

Después de realizar esta modificación sólo hay que poner a correr Lingo y resolverá el problema con el óptimo continuo.

Es destacable el tiempo de resolución de los problemas continuos, ya que es prácticamente inmediato en el caso, sobre todo, de la estrategia de flujo múltiple.

En la siguiente tabla se muestran los óptimos del problema continuo resueltos con la estrategia de flujo múltiple, representados en la columna “Bound”, el número de iteraciones realizadas, los óptimos enteros de cada problema y la diferencia entre estos últimos y el bound.

Sólo se representan los resultados de los problemas del grupo “b” ya que los del grupo “c” requerían un tiempo mayor de resolución, por lo que se considerará su análisis para un siguiente trabajo.

Modelo	Iteraciones	Bound	Óptimos Enteros	Diferencia
Steinb1	28	49,4286	82	32,57
Steinb2	40	36,3	83	46,70
Steinb3	61	85,3571	138	52,64
Steinb4	176	15,125	59	43,88
Steinb5	148	13,2727	61	47,73
Steinb6	246	16,5	122	105,50
Steinb7	74	56,3	111	54,70
Steinb8	98	53,6429	104	50,36
Steinb9	71	160,364	220	59,64
Steinb10	295	29,1667	86	56,83
Steinb11	332	23,3333	88	64,67
Steinb12	389	32,3714	174	141,63
Steinb13	130	52,2308	165	112,77
Steinb14	156	100,2	235	134,80
Steinb15	177	117	318	201,00
Steinb16	363	26,6875	127	100,31
Steinb17	390	45,6818	131	85,32
Steinb18	472	49,3182	218	168,68

Tabla 8. Soluciones Continuas Flujo Múltiple

Fuente: Elaboración Propia

Acto seguido se muestra la gráfica que representa los valores de la diferencia para cada problema:

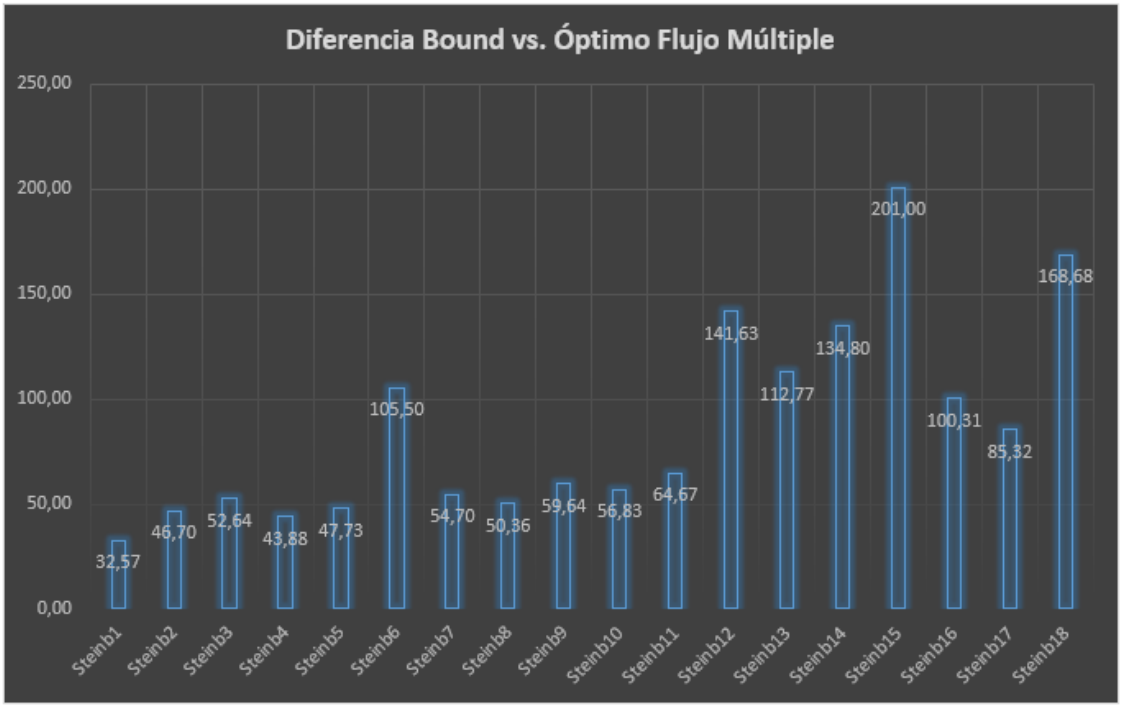


Fig.28 Diferencias Óptimos Continuos vs. Enteros Flujo Múltiple

Fuente: Elaboración Propia

Como era de esperar, los problemas con un valor más alto de "Diferencia Bound-Óptimo" son los que más problemas nos han dado a la hora de resolverlos, ya sea porque la solución que encontraban en treinta minutos no era muy cercana al óptimo, porque se tardó mucho en encontrar una solución admisible, etc.

A continuación se muestra la misma tabla Tabla 8 pero para las soluciones obtenidas con la estrategia de flujo simple:

Modelo	Iteraciones	Bound	Óptimos Enteros	Diferencia
Steinb1	178	72	82	10,00
Steinb2	318	74	83	9,00
Steinb3	640	124,5	138	13,50
Steinb4	1034	49,5	59	9,50
Steinb5	1731	49,5	61	11,50
Steinb6	5519	89	122	33,00
Steinb7	547	96,5	111	14,50
Steinb8	1018	82,5	104	21,50
Steinb9	1406	200	220	20,00
Steinb10	3254	71,5	86	14,50
Steinb11	6851	72,5	88	15,50
Steinb12	16869	128	174	46,00
Steinb13	1749	137	165	28,00
Steinb14	3313	198	235	37,00
Steinb15	11424	250,5	318	67,50
Steinb16	7551	103,5	127	23,50
Steinb17	10683	105,5	131	25,50
Steinb18	61283	170	218	48,00

Tabla 9. Soluciones Continuas Flujo Simple

Fuente: Elaboración Propia

La gráfica de los valores diferencia de la tabla anterior es la siguiente:

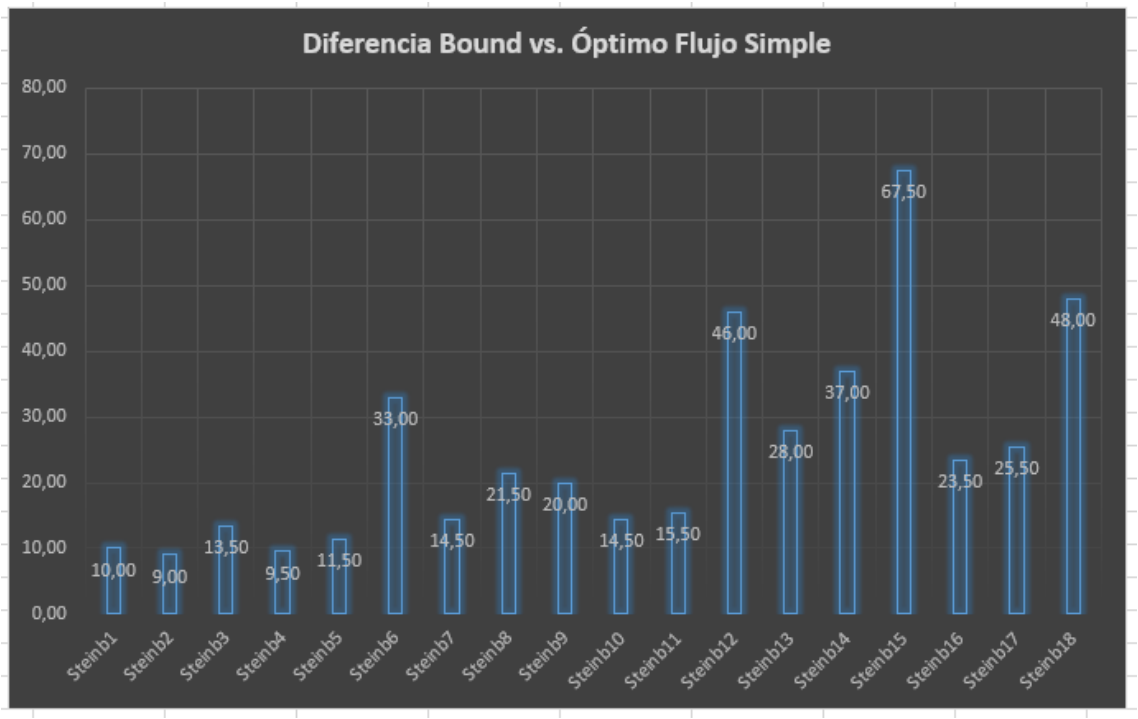


Fig.29 Diferencias Óptimos Continuos vs. Enteros Flujo Simple

Fuente: Elaboración Propia

Vuelve a ocurrir lo mismo que en la gráfica de la Fig. 28, aunque en este caso los valores son mucho más bajos, lo que nos llevaría a pensar en que la estrategia de flujo simple puede ser mejor que la múltiple.

Esto hemos visto ya que no es así pues, aunque el bound en el simple sea mucho más cercano al óptimo que en el múltiple, la convergencia de éste hasta el mejor valor posible es mucho más lenta.

Para poder ver con más claridad las diferencias entre los bounds obtenidos para las dos estrategias se ha realizado una gráfica comparativa que se muestra a continuación:

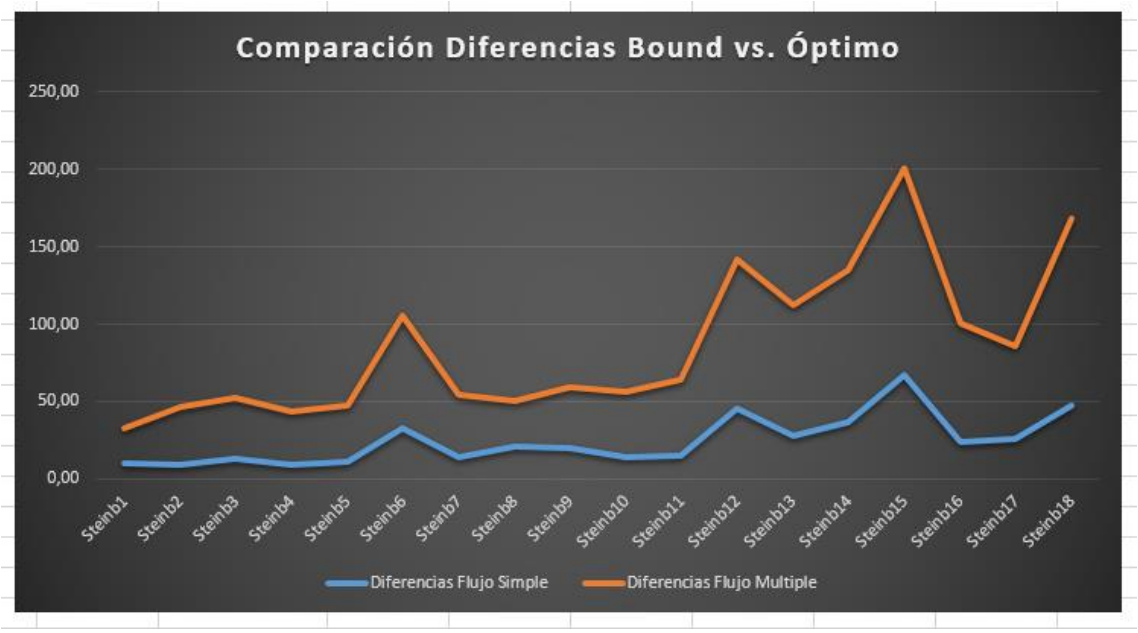


Fig.30 Comparación Diferencias Bound vs. Óptimo

Fuente: Elaboración Propia

CONCLUSIONES

Hemos analizado ambas estrategias, considerando como factor clave a la hora de decir qué estrategia utilizar el error producido entre la solución admisible que nos brinda lingo pasados unos treinta minutos y la solución óptima del problema.

Esta decisión se tomó, como bien se dijo anteriormente, tras comprobar que había problemas que requerían de varias horas, incluso días para obtener el óptimo con Lingo.

Con el fin de tener un amplio abanico de datos y poder interpretar y comparar, al menos, dos categorías de problemas diferentes, se realizaron los experimentos siguiendo el criterio antes especificado.

Ciñéndonos exclusivamente al dato de "Error", es más que evidente que la estrategia de flujo múltiple se comporta mejor que la estrategia de flujo simple.

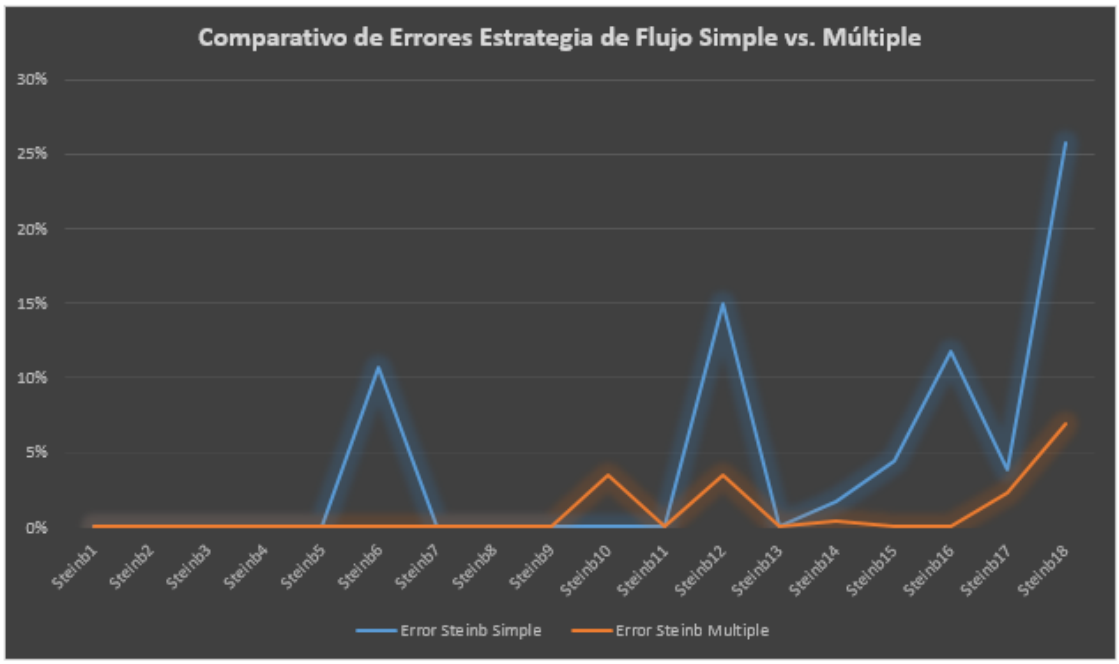


Fig.31 Comparación Errores Simple vs. Múltiple

Fuente: Elaboración Propia

Se hace más claro aún si tenemos en cuenta que Lingo no pudo obtener soluciones admisibles de gran parte de los problemas del grupo "c" pasada media hora con la estrategia de flujo simple, cosa que sí consiguió sin problemas la estrategia de flujo múltiple.

Si queremos apoyarnos en los datos obtenidos del estudio de las soluciones continuas, podríamos decir que la estrategia de flujo simple es mejor que la múltiple, pero la

experiencia nos dice que es al contrario si tenemos en cuenta lo expuesto anteriormente.

Del estudio de las soluciones continuas sólo se puede deducir que **no es indicado para comparar los mismos problemas resueltos con estrategias diferentes**, a fin de comprobar cuál es mejor, al contrario de lo que se pensaba a priori. Sin embargo, se deduce que **sí es útil para saber cómo se comportan ciertos problemas al resolverlos con una misma estrategia**.

Como conclusión final qué mejor que responder a la pregunta que nos hacíamos al principio:

¿Qué estrategia he de utilizar para resolver problemas de Steiner si pretendo usar la formulación de flujo?

La respuesta es que, prácticamente sin excepción, la mejor opción siempre será la estrategia de flujo múltiple, la cual nos ofrecerá una solución óptima en un menor tiempo.

La estrategia de flujo simple tendría que quedar completamente descartada si queremos ser eficientes a la hora de resolver problemas, pero nunca está de más en el ámbito investigador.

BIBLIOGRAFÍA

1. Diseño de Redes de Comunicaciones Confiables. El Problema de Steiner Generalizado.
   Centro de Cálculo – Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay.
   Autor: Sergio Nesmachnow
   Año: 2002

2. Resolución del Problema Generalizado de Steiner Utilizando Simulated Annealing.
   Lab. de Computación de Alto Rendimiento (LCAR), Univ. Nacional Experimental Del Táchira, Táchira (Venezuela)
   Autores: Javier Maldonado, Jhon Amaya
   Año: Sin especificar

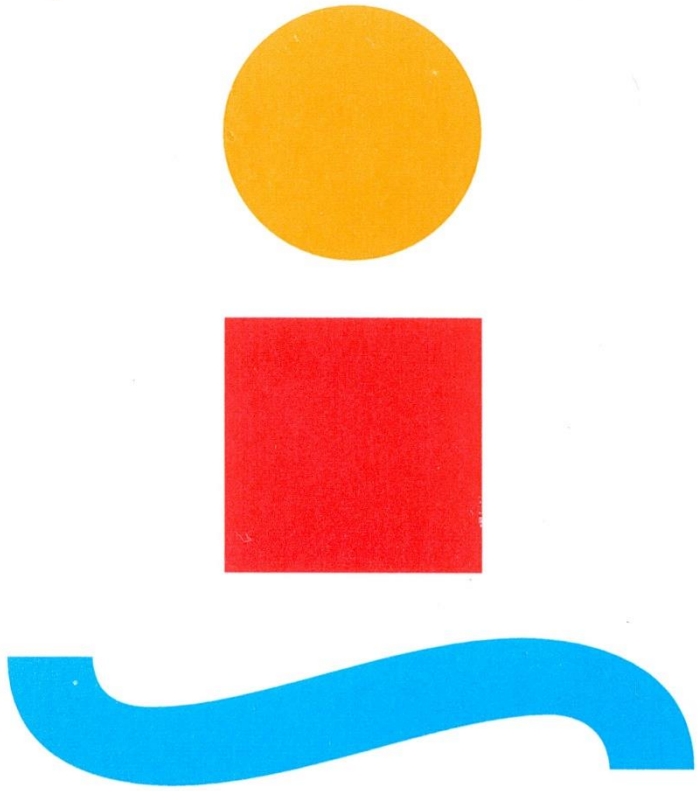
3. Análisis de estrategias de modelado para la resolución del problema de Steiner en grafos.
   Trabajo Fin de Grado. Universidad de Sevilla.
   Autor: Daniel Guzmán Moreno
   Tutor: José Manuel García Sánchez
   Año: Sin Especificar

4. Resolución de Problemas. Librerías de Optimización. Métodos Cuantitativos de Gestión.
   Apuntes Asignatura Métodos de Optimización.
   Autor: José Manuel García Sánchez
   Año: Sin Especificar

5. 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

6. Archivos de texto de los Enunciados de los Problemas de Steiner
   URL: http://www.brunel.ac.uk/~mastjjb/jeb/info.html (OR-Library J E Beasley)

7. Optimal Trees
   Handbooks in Operations Research and Managament Science
   Autores: Thomas L. Magnanti and Laurence A. Wolsey
   Año: 1994



escuela superior de
INGENIEROS DE SEVILLA

ANEXOS

ANEXO I (Código en C para Flujo Múltiple)

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

int main()
{
    char nombrearchivo[50];
    printf("Escriba el nombre completo (xxxx.txt) del archivo del enunciado: ");
    scanf("%s",&nombrearchivo);
    FILE *f = fopen(nombrearchivo,"r");
    FILE *f2 = fopen("enunciadolingo.txt","w");
    if (f==NULL)
    {
    perror("Error al abrir el archivo de lectura");
    return -1;
    }
    if (f2==NULL)
    {
    perror ("Error al abrir enunciadolingo.txt");
    return -1;
    }
    int nodos;
    int arcos;
    int i;
    int j;
    fscanf(f, "%d %d\n",&nodos,&arcos);
    printf("Nodos = %d ; Arcos = %d",nodos,arcos);
    printf("\n\n");
    fprintf(f2,"MODEL:\nSETS:\nNodos/1..%d/:Ti,R;\nArcos1(Nodos,Nodos)/",nodos);
    int origen[arcos-1];
    int destino[arcos-1];
    int coste[arcos-1];
    for(i=0;i<arcos;i++)
    {
    fscanf (f, "%d %d %d\n",&origen[i],&destino[i],&coste[i]);
    printf("ORIGENARCO%d = %d ; DESTINOARCO%d = %d ; COSTEARCO%d = %d\n",i+1,origen[i],i+1,destino[i],i+1,coste[i]);
    if(i==0)
    {
        if(origen[i]<destino[i])
        {
            fprintf(f2,"%d %d",origen[i],destino[i]);
        }
        else
        {
            fprintf(f2,"%d %d",destino[i],origen[i]);
        }
    }
    else
    {
        if(origen[i]<destino[i])
        {
            fprintf(f2,",%d %d",origen[i],destino[i]);
        }
        else
        {
            fprintf(f2,",%d %d",destino[i],origen[i]);
        }
    }
    }
    fprintf(f2,"/:C,Seleccion;\nArcos2(Nodos,Nodos)/");
    for(i=0;i<arcos;i++)
    {
    if(i==0)
    {
        fprintf(f2,"%d %d",destino[i],origen[i]);
        fprintf(f2,",%d %d",origen[i],destino[i]);
    }
```

```
else
{
    fprintf(f2,",%d %d",destino[i],origen[i]);
    fprintf(f2,",%d %d",origen[i],destino[i]);
}
}
fprintf(f2,"/:X;\nENDSETS\nDATA:\nTi=");
printf("\n\n");
int numeroterminales;
fscanf(f,"%d\n",&numeroterminales);
int terminales[numeroterminales-1];
printf("NUMERO DE NODOS TERMINALES = %d\n\n",numeroterminales);
printf("Los nodos terminales son: ");
int Ti[nodos-1];
for(j=0;j<nodos;j++)
{
    Ti[j]=0;
}
for(j=0;j<numeroterminales;j++)
{
    fscanf(f,"%d",&terminales[j]);
    printf("[%d] ",terminales[j]);
    Ti[terminales[j]-1]=1;
}
for(j=0;j<nodos;j++)
{
    if(Ti[j]==1)
    {
        if(j==0)
        {
            fprintf(f2,"1");
        }
        else
        {
            fprintf(f2,",1");
        }
    }
    else
    {
        if(j==0)
        {
            fprintf(f2,"0");
        }
        else
        {
            fprintf(f2,",0");
        }
    }
}
fprintf(f2,";\nR=");
int R[nodos-1];
for(j=0;j<nodos;j++)
{
    R[j]=0;
}
int contador=0;
j=0;
do
{
    R[j]=0;
    contador++;
    j++;
}while(Ti[j]==0);
R[contador]=1;
for(j=0;j<nodos;j++)
{
    if(j==0)
```

```
        {
            fprintf(f2,"%d",R[j]);
        }
        else
        {
            fprintf(f2,",%d",R[j]);
        }
    }
    fprintf(f2,";\nC=");
    for(i=0;i<arcos;i++)
    {
        if(i==0)
        {
            fprintf(f2,"%d",coste[i]);
        }
        else
        {
            fprintf(f2,",%d",coste[i]);
        }
    }
    fprintf(f2,";\n");
    int CS=numeroterminales-1;
    fprintf(f2,"CS=%d;\nENDDATA\nMIN=",CS);
    printf("\n\n");
    int costesumar;
    fscanf(f,"%d",&costesumar);
    printf("El coste a sumar en la FO es de %d",costesumar);
    printf("\n\n");
    fprintf(f2,"%d+@SUM(Arcos1(i,j):Seleccion(i,j)*C(i,j));\n",costesumar);
    //fprintf(f2,"@FOR(Arcos1(i,j):@GIN(X(i,j)));\n");
    fprintf(f2,"@FOR(Arcos1(i,j):@BIN(Seleccion(i,j)));\n");
    fprintf(f2,"@FOR(Nodos(i)|Ti(i)#EQ#1 #AND# R(i)#NE#1:@SUM(Arcos2(i,k)|R(k)#NE#1:X(i,k))=@SUM(Arcos2(j,i):X(j,i))-1);\n");
    fprintf(f2,"@FOR(Nodos(i)|Ti(i)#EQ#0:@SUM(Arcos2(k,i):X(k,i))=@SUM(Arcos2(i,j):X(i,j)));\n");
    fprintf(f2,"@FOR(Arcos2(i,j)|i#LT#j:X(i,j)<=CS*@SUM(Arcos1(s,k)|s#EQ#i #AND# k#EQ#j:Seleccion(s,k)));\n");
    fprintf(f2,"@FOR(Arcos2(i,j)|i#GT#j:X(i,j)<=CS*@SUM(Arcos1(s,k)|s#EQ#j #AND# k#EQ#i:Seleccion(s,k)));\n");
    fprintf(f2,"@FOR(Nodos(i)|R(i)#EQ#1:@SUM(Arcos2(i,j):X(i,j))=CS);\nEND");
    fclose(f);
    fclose(f2);
}
```

ANEXO II (Código en C para Flujo Simple)

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

int main()
{
    char nombrearchivo[50];
    printf("Escriba el nombre completo (xxxx.txt) del archivo del enunciado: ");
    scanf("%s",&nombrearchivo);
    FILE *f = fopen(nombrearchivo,"r");
    FILE *f2 = fopen("enunciadolingo.txt","w");
    if (f==NULL)
    {
    perror("Error al abrir el archivo de lectura");
    return -1;
    }
    if (f2==NULL)
    {
    perror ("Error al abrir enunciadolingo.txt");
    return -1;
    }
    int nodos;
    int arcos;
    int i;
    int j;
    fscanf(f, "%d %d\n",&nodos,&arcos);
    printf("Nodos = %d ; Arcos = %d",nodos,arcos);
    printf("\n\n");
    fprintf(f2,"MODEL:\nSETS:\nNodos/1..%d/:Ti,R;\nArcos(Nodos,Nodos)/",nodos);
    int origen[arcos-1];
    int destino[arcos-1];
    int coste[arcos-1];
    for(i=0;i<arcos;i++)
    {
    fscanf (f, "%d %d %d\n",&origen[i],&destino[i],&coste[i]);
    printf("ORIGENARCO%d = %d ; DESTINOARCO%d = %d ; COSTEARCO%d = %d\n",i+1,origen[i],i+1,destino[i],i+1,coste[i]);
    if(i==0)
    {
        if(origen[i]<destino[i])
        {
            fprintf(f2,"%d %d",origen[i],destino[i]);
        }
        else
        {
            fprintf(f2,"%d %d",destino[i],origen[i]);
        }
    }
    else
    {
        if(origen[i]<destino[i])
        {
            fprintf(f2,",%d %d",origen[i],destino[i]);
        }
        else
        {
            fprintf(f2,",%d %d",destino[i],origen[i]);
        }
    }
    }
    fprintf(f2,"/:C,Seleccion;\nConj(Nodos,Nodos,Nodos)/");
    int numeroterminales;
    fscanf(f,"%d\n",&numeroterminales);
    int terminales[numeroterminales-1];
    printf("NUMERO DE NODOS TERMINALES = %d\n\n",numeroterminales);
    printf("Los nodos terminales son: ");
    int Ti[nodos-1];
    int cont=1;
    for(j=0;j<nodos;j++)
    {
        Ti[j]=0;
    }
    for(j=0;j<numeroterminales;j++)
    {
        fscanf(f,"%d",&terminales[j]);
        printf("[%d] ",terminales[j]);
        Ti[terminales[j]-1]=1;
    }
    for(i=0;i<arcos;i++)
    {
        for(j=0;j<numeroterminales;j++)
        {
            if(i==0 && cont==1)
            {
                fprintf(f2,"%d %d %d",destino[i],origen[i],terminales[j]);
                fprintf(f2,",%d %d %d",origen[i],destino[i],terminales[j]);
            }
```

```
            else
            {
                fprintf(f2,",%d %d %d",destino[i],origen[i],terminales[j]);
                fprintf(f2,",%d %d %d",origen[i],destino[i],terminales[j]);
            }
            cont=cont+1;
        }
    }
    fprintf(f2,"/:X;\nENDSETS\nDATA:\nTi=");
    for(j=0;j<nodos;j++)
    {
        if(Ti[j]==1)
        {
            if(j==0)
            {
                fprintf(f2,"1");
            }
            else
            {
                fprintf(f2,",1");
            }
        }
        else
        {
            if(j==0)
            {
                fprintf(f2,"0");
            }
            else
            {
                fprintf(f2,",0");
            }
        }
    }
    printf("\n\n");
    fprintf(f2,";\nR=");
    int R[nodos-1];
    for(j=0;j<nodos;j++)
    {
        R[j]=0;
    }
    int contador=0;
    j=0;
    do
    {
        R[j]=0;
        contador++;
        j++;
    }while(Ti[j]==0);
    R[contador]=1;
    for(j=0;j<nodos;j++)
    {
        if(j==0)
        {
            fprintf(f2,"%d",R[j]);
        }
        else
        {
            fprintf(f2,",%d",R[j]);
        }
    }
    fprintf(f2,";\nC=");
    for(i=0;i<arcos;i++)
    {
        if(i==0)
        {
            fprintf(f2,"%d",coste[i]);
        }
        else
        {
            fprintf(f2,",%d",coste[i]);
        }
    }
    fprintf(f2,";\nENDDATA\nMIN=");
    int costesumar;
    fscanf(f,"%d",&costesumar);
    printf("El coste a sumar en la FO es de %d",costesumar);
    printf("\n\n");
    fprintf(f2,"%d+@SUM(Arcos(i,j):Seleccion(i,j)*C(i,j));\n",costesumar);
    //fprintf(f2,"@FOR(Nodos(k)|R(k)#NE#1:@FOR(Conj(i,j,k):X(i,j,k)<=1));\n");
    fprintf(f2,"@FOR(Arcos(i,j):@BIN(Seleccion(i,j)));\n");
    fprintf(f2,"@FOR(Nodos(k)|R(k)#NE#1 #AND# Ti(k)#EQ#1:@FOR(Nodos(s)|s#NE#k #AND# R(s)#EQ#0:@SUM(Conj(i,s,k):X(i,s,k))=@SUM(Conj(s,i,k):X(s,i,k))));\n");
    fprintf(f2,"@FOR(Nodos(k)|R(k)#NE#1 #AND# Ti(k)#EQ#1:@FOR(Nodos(rr)|R(rr)#EQ#1:@SUM(Conj(i,rr,k):X(i,rr,k))+1=@SUM(Conj(rr,i,k):X(rr,i,k))));\n");
    fprintf(f2,"@FOR(Arcos(i,j)|i#LT#j:@FOR(Nodos(k)|R(k)#EQ#0 #AND# Ti(k)#EQ#1:@SUM(Conj(i,j,k):X(i,j,k))<=Seleccion(i,j)));\n");
    fprintf(f2,"@FOR(Arcos(i,j):@FOR(Nodos(k)|R(k)#EQ#0 #AND# Ti(k)#EQ#1:@SUM(Conj(i,j,k)|i#LT#j:X(j,i,k))<=Seleccion(i,j)));\nEND");
    fclose(f);
    fclose(f2);
}
```