Análisis del software Grafos

Métodos Cuantitativos III
Curso 2007-2008 Q2

Autores:

Carlos Paredes Egea

Dídac Gonzalez Sánchez

Josep María Borrell Serra

Eva Jurjo Mosoll

Luis Gámiz Fernandez

INDICE

1. Introducción:

Este trabajo pretende ayudar a entender cómo trabaja el programa Grafos, así como intentar explicar en qué consisten todos los algoritmos que incorpora. Se pretende explicar cómo se ha de utilizar éste programa para poder resolver toda la serie de algoritmos de éste programa. Con esto queremos decir que es una especie de tutorial básico, pero que también profundiza en el conocimiento del método de resolución de cada algoritmo. Es decir, alguien que no sepa cómo funcionan los algoritmos que incorpora el programa, sabrá para que se ha de utilizar cada tipo de algoritmo. En el caso que quiera profundizar un poco más en dicho algoritmo, también se adjunta unas series de ejemplos paso a paso, sin utilizar éste software que ayudan a la interpretación de dicho algoritmo.

Es un software para la construcción, edición y análisis de grafos. Grafos pretende ser de utilidad para la docencia y el aprendizaje de la Teoría de Grafos, y otras disciplinas relacionadas como la ingeniería de organización industrial, la logística y el transporte, investigación operativa, diseño de redes, etc. Grafos se puede usar perfectamente para el modelado y resolución de problemas reales.

Un grafo representa un modelo de una realidad empresarial en forma de red. Este modelo podrá ser analizado desde distintos puntos de vista gracias a los algoritmos y funciones incorporados en el software Grafos. Independientemente de sus conocimientos actuales sobre la materia, la información recogida en estas páginas será un buen punto de partida para el aprendizaje en mayor profundidad de la Teoría de Grafos (Graph Theory) y su aplicación en la realidad empresarial e industrial.

La filosofía de Grafos es la siguiente: "dibujar, modelar, resolver y analizar". Con esto se pretende que el usuario tenga libertad absoluta para tratar y abordar los problemas de grafos. Podrá dibujar libremente el grafo sin preocuparse del análisis o algoritmo que utilizará posteriormente. Grafos le avisará en caso de no factibilidad o de cualquier otro requerimiento para un análisis en particular. Los estudiantes que usen Grafos experimentarán un proceso de aprendizaje basado en su libertad y en etapas de prueba-error. Otros programas existentes, a diferencia de este, guían al usuario paso a paso, descartando de entrada su libertad de elección y construcción.

La versión actual del programa 1.2.8 es totalmente funcional, y se puede usar para diseñar grafos y para los análisis descritos. Todo el material recopilado en este sitio web (software, textos, imágenes, casos, etc.), se distribuye bajo licencia Creative Commons License.

Actualmente, el proyecto Grafos sigue en activo, en proceso de programación de nuevos Algoritmos, incorporación de nuevas funciones y generación de documentación actualizada.

1.1. Algoritmos que resuelve:

Este es un listado con todos los posibles algoritmos o análisis implementados actualmente en Grafos:

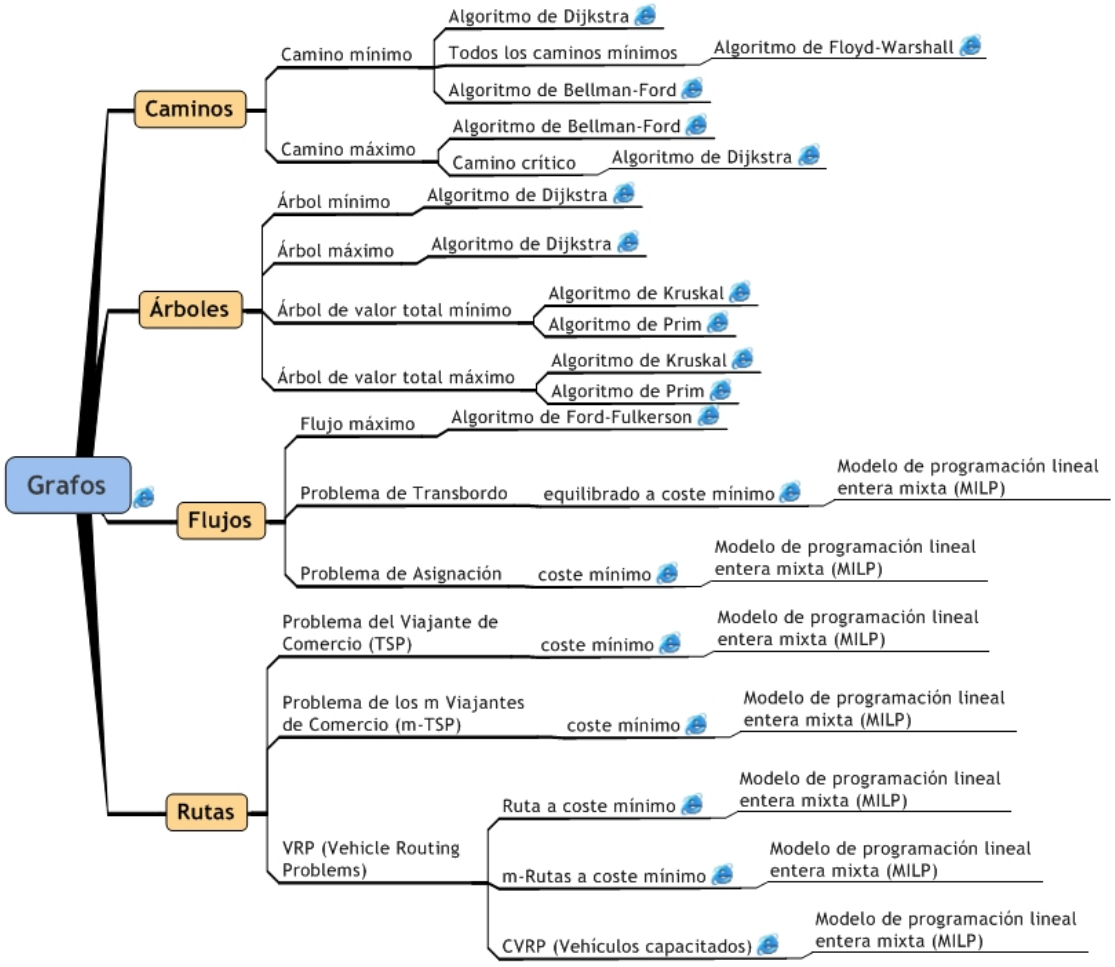


Ilustración 1 - Posibilidades De Resolución Del Software

1.2. Principales Funciones:

Aquí se indican las principales características funcionales del programa:

- Arcos bidireccionales (directed graphs).
- Posibilidad de arco con origen y destino sobre un mismo nodo.
- Valores en arcos (mínimo, máximo y coste).
- Valores en nodos (de coste o de capacidad/demanda, y etiqueta).
- Edición gráfica WYSIWYG ('lo que usted ve, es lo que usted obtiene').
- Deshacer y rehacer operaciones de edición gráfica.
- Cursores de dibujo contextuales.
- Menús y barras de herramientas sensibles al contexto.
- Dibujar arcos mediante arrastrar y soltar (botón central del ratón).

- Edición de arcos y nodos con doble-clic (botón izquierdo del ratón).
- Edición en modo tabla del grafo.
- Etiquetado automático de nodos.
- Guardar y abrir fichero (formato propietario .grf, formato estándar .graphML).
- Exportar datos del grafo de manera personalizada a ficheros .txt, .csv.
- Importar/Añadir datos al grafo desde ficheros .txt, .csv.
- Definición de estilo gráfico.
- Personalización del aspecto gráfico de cada nodo y arco (incluye puntas de flecha).
- Reordenación automática del grafo en formato (aleatorio, árbol, circular, tabular, flujo, orgánico, radial).
- Zoom (ampliar, reducir, ajustar) y control del Zoom con rueda del ratón.
- Girar y contraer/extender parte del grafo seleccionado con rueda del ratón.
- Alineación de nodos (horizontal, vertical, a rejilla).
- Rejilla de dibujo.
- Centrar grafo a tapiz, o ajustar tapiz al grafo.
- Incluir imagen de fondo ajustada al tapiz del grafo (de gran utilidad para representar grafos sobre mapas).
- Exporta la imagen del grafo a diferentes formatos de gráficos (.gif, .tif, .png, .bmp, .svg).
- Copiar imagen del grafo al portapapeles de Windows.
- Selección de impresora y Configuración de página.
- Función 'Imantar' nodos para su alineado a la rejilla.
- Crear un nuevo grafo aleatorio (indicando número de nodos y densidad de arcos).
- Visión preliminar e Impresión.
- Selección de nodos con el ratón (mover, alinear H/V, alinear a rejilla, borrar).
- Utilidades (auto radio-nodo, trazo-arco, coste-arco según distancia).
- Formato automático de radio de nodo y trazo de arco en función de valores.
- Configuración del Solver y log del proceso de optimización.
- Configuración del formato de representación de resultados.
- Ventana de solución del análisis (visualizar, copiar, exportar e imprimir solución). Permutar la visualización (grafo original - grafo solución tras análisis).
- Exportar/Visualizar/Imprimir los modelos MILP utilizados en los análisis. Modelos en formatos (.lp, .mps) que pueden ser usados en otros solvers.
- Estructura de datos extensible para problemas VRP en formato (.vrpxml).

2. Análisis de los algoritmos implementados en el software:

En este apartado explicaremos cada algoritmo que implementa el software. Cada tipo de algoritmo, mantiene la siguiente estructura: primero se explica que tipos de problemas resuelve dicho algoritmo, luego se expone el pseudo código o condiciones de programación lineal, a continuación se adjunta un problema hecho paso a paso a mano, es decir siguiendo el pseudo código y sin utilizar el software Grafos. Al final se complementa con la resolución de dicho problema mediante el software Grafos, analizando los datos que este nos devuelve. Hay algoritmos que debido a su complejidad sólo constan de la resolución clara y concisa de un ejemplo mediante el software Grafos.

2.1. Análisis de caminos:

2.1.1. Dijkstra

2.1.1.1. Camino mínimo

El problema de la ruta más corta se puede resolver utilizando programación lineal sin embargo, debido a que el método simplex es de complejidad exponencial, se prefiere utilizar algoritmos que aprovechen la estructura en red que se tiene para estos problemas.

Para ello, el algoritmo mantiene un conjunto S de nodos cuyos pesos finales de camino mínimo desde el nodo origen ya han sido determinados.

A continuación se muestra el pseudo código del Algoritmo:

```
        Dijkstra (G,s)

        Inicializar
  for cada v perteneciente a V[G]
    do d[v] = infinito
     p[v] = nulo
     d[s] = 0

        S =  vacío
Q = V[G]

        mientras Q no vacío
do u = nodo v con min d[v]
 S = S unión u 'se añade al conjunto de nodos
finalizados

        for cada v perteneciente Adyacente u
  Relajación
    if d[v] > d[u] + w(u,v) then
      d[v] = d[u] + w(u,v)
      p(v) = u
```

2.1.1.2. Ejemplo hecho a mano camino mínimo

A partir del algoritmo de Dijkstra se realizará un ejemplo paso por paso para su mejor comprensión, a continuación se resolverá con el programa objeto de estudio y se comprobará que realmente da el mismo resultado. El grafo a estudiar será el siguiente:

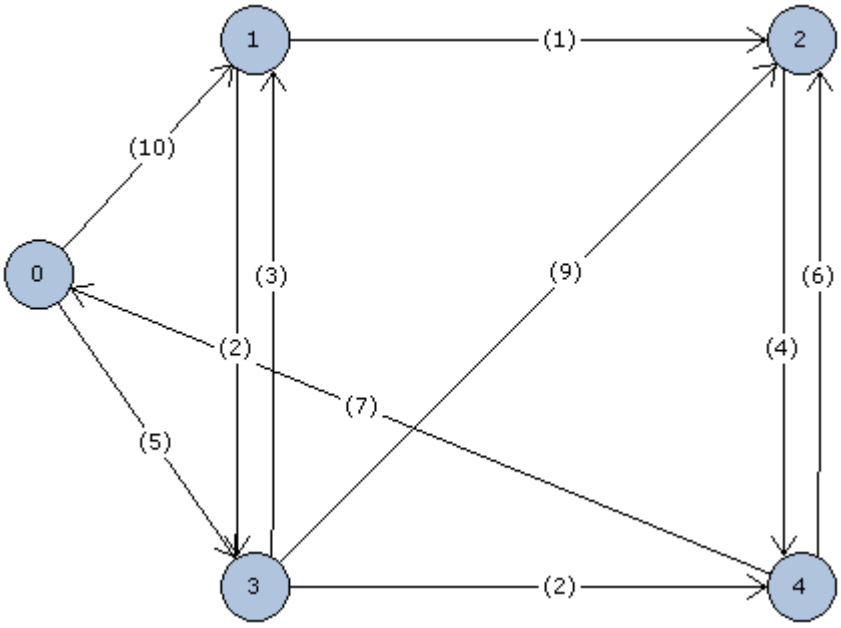


Ilustración 2- Grafo Camino Mínimo A Mano Dijkstra

1.- El primer paso a realizar será buscar la distancia mínima desde el nodo origen (0) a los posibles nodos donde se pueda ir, en este caso al 1 y al 3. Observamos que la distancia mínima nos lleva al nodo 3, con un valor de $d_1 = 5$.

2.- El siguiente paso a seguir es hacer exactamente igual que en el paso anterior, pero esta vez el origen lo tomamos desde el nodo 3 (aunque la distancia que hemos obtenido antes la conservamos para poder hacer la suma total). Observamos que desde 3 tenemos la posibilidad de ir al nodo 1, con un coste de 3; al nodo 2, con un coste de 9; y al nodo 4 (nodo destino), con un coste de 2. En este caso se ve con mucha claridad que el algoritmo nos llevará al nodo 4 debido a que es el nodo destino y a su vez el nodo que tiene la distancia más corta desde 3.

3.- Finalmente, unimos el nodo 3 con el nodo 4, y obtenemos un valor de $d_2 = 2$. Así pues, la distancia total recorrida será de $D = d_1 + d_2 = 5 + 2 = 7$.

2.1.1.3. Ejemplo resuelto por el software camino mínimo

A partir del siguiente grafo encontraremos algunos caminos mínimos según este algoritmo:

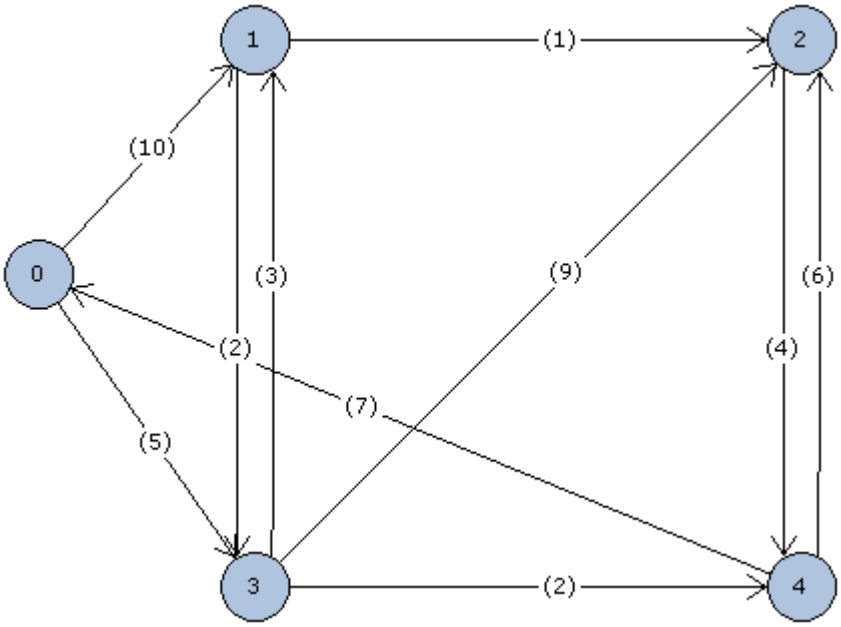


Ilustración 3 - Grafo Camino Mínimo Software Dijkstra 1

Aplicando el algoritmo, para ir, por ejemplo, del nodo 0 al 4 obtenemos la siguiente solución:

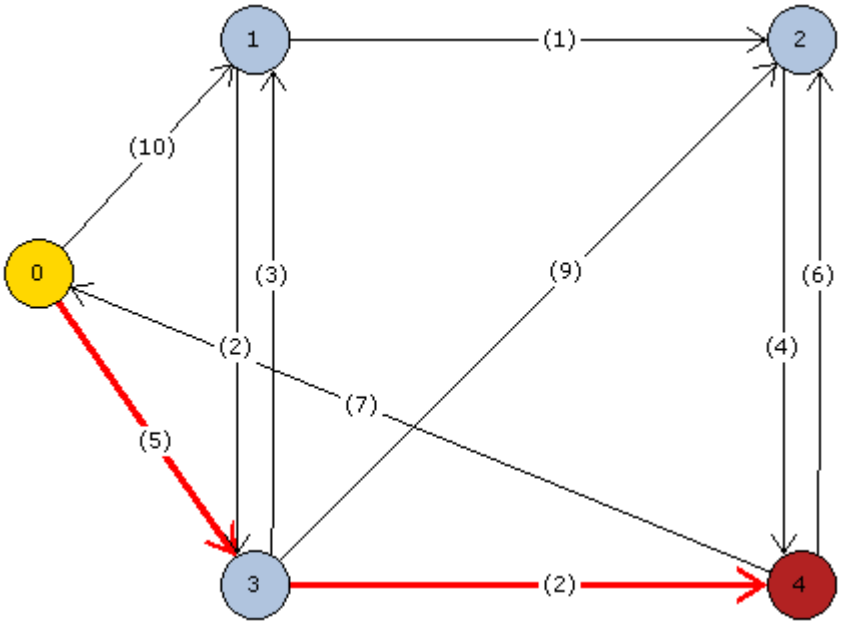


Ilustración 4 - Grafo Camino Mínimo Software Dijkstra 2

CAMÍNO MÍNIMO – ALGORITMO DE DIJKSTRA

Arcos calculados desde el nodo origen (0) hasta el nodo destino (4):

* 0 ----(5)→ 3

* 3 ----(2)→ 4

Coste total = 7

Matriz de Arcos con coste mínimo:

N1\N2	0	1	2	3	4
0	0	0	0	1	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	1
4	0	0	0	0	0

2.1.1.4. Ejemplo hecho a mano camino máximo

Para realizar un ejemplo del algoritmo de Dijkstra con camino crítico o máximo, utilizaremos el siguiente grafo, con caminos de ida y vuelta entre los nodos. El nodo inicial será el 1, mientras que el final será el 3:

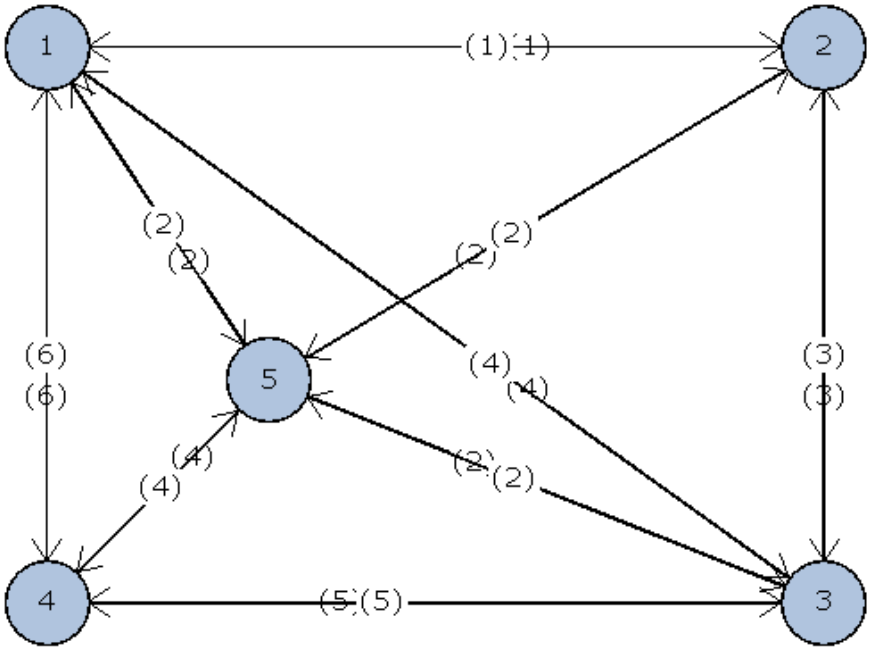


Ilustración 5 - Grafo Camino Máximo A Mano Dijkstra

1.- El primer paso a realizar será buscar la distancia máxima desde el nodo origen (1) a los demás nodos posibles donde se pueda ir, en este caso al 2, 3, 4 y al 5. Observamos que la distancia máxima nos lleva al nodo 4, con un valor de $d_1 = 6$.

2.- El siguiente paso a seguir es hacer exactamente igual que en el paso anterior, pero esta vez el origen lo tomamos desde el nodo 4 (aunque la distancia que hemos obtenido antes la conservamos para poder hacer la suma total). Observamos que desde 4 tenemos la posibilidad de ir al nodo 5, con un coste de 4; al nodo 3, con un coste de 5; o volver al nodo 1 (nodo origen), pero este último caso no es posible debido a la naturaleza del algoritmo. Se supone que el camino crítico para llegar desde 1 hasta 3 pasa por el mínimo numero de nodos, pero con máximo coste.

3.- Finalmente, unimos el nodo 4 con el nodo 3, y obtenemos un valor de $d_2 = 5$. Así pues, la distancia total recorrida será de $D = d_1 + d_2 = 6 + 5 = 11$.

El camino crítico estará formado por tareas críticas (nodos) cuya duración (coste del arco sucesor) determina la duración total de un proyecto. Si una tarea crítica se retrasa o su duración cambia durante la realización del proyecto, afectaría directamente a la duración total del proyecto y a su fecha de finalización.

Encontrar el camino crítico de la planificación de un proyecto es lo mismo que encontrar el camino más largo desde el nodo inicial (tarea inicial) al nodo final (última tarea); esto es, la mínima cantidad de tiempo necesaria para finalizar un proyecto.

El algoritmo de Dijkstra aunque fue diseñado para encontrar la ruta más corta se puede transformar fácilmente para encontrar la ruta más larga (camino crítico), cambiando simplemente su función objetivo. Del mismo modo, se encuentra el árbol máximo desde un nodo origen.

2.1.1.5. Ejemplo hecho por el software camino máximo

Utilizamos el mismo grafo que anteriormente:

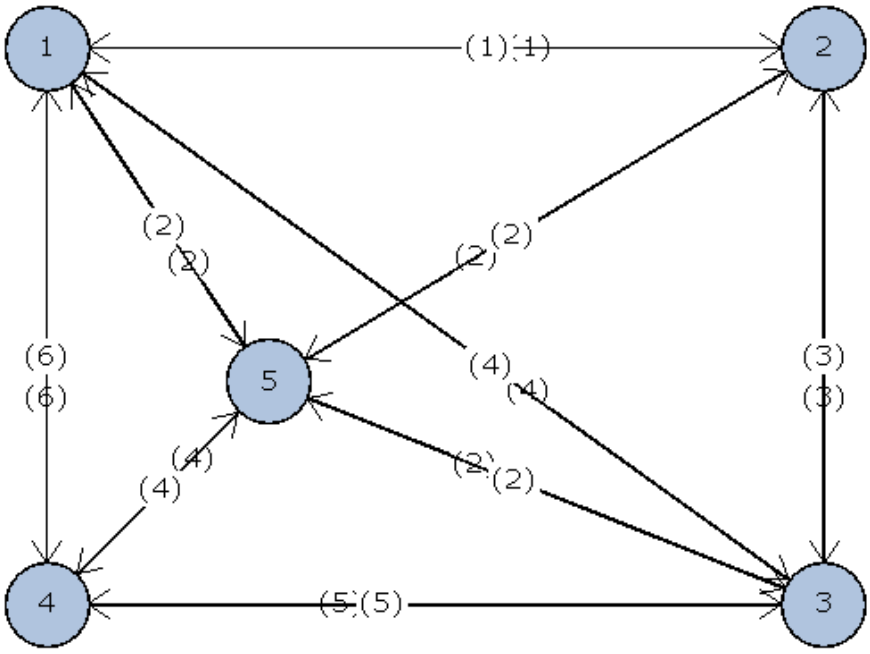


Ilustración 6 - Grafo Camino Máximo Software Dijkstra I

Seleccionando nodo origen y nodo destino obtenemos la siguiente solución:

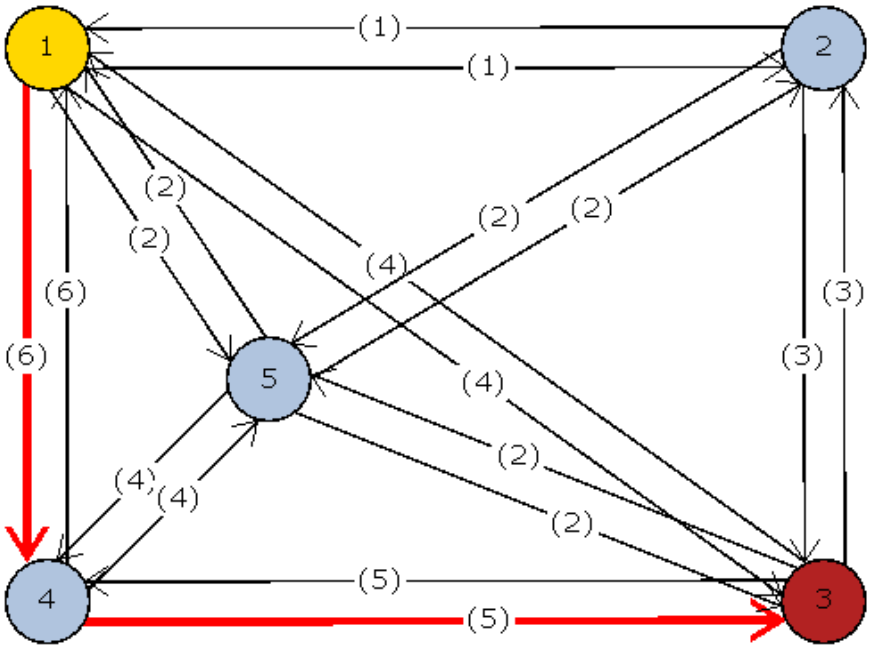


Ilustración 7 - Grafo Camino Máximo Software Dijkstra 2

CAMINO CRÍTICO - ALGORITMO DE DIJKSTRA

Arcos calculados desde el nodo origen (1) hasta el nodo destino (3):

* 1 --(6)→ 4
* 4 --(5)→ 3

Coste total = 11

Matriz de Arcos con coste máximo:

N1\N2	1	2	3	4	5
1	0	0	0	1	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	1	0	0
5	0	0	0	0	0

2.1.2. Floyd-Warshall

El problema que intenta resolver este algoritmo es el de encontrar el camino más corto entre todos los pares de nodos o vértices de un grafo. Esto es semejante a construir una tabla con todas las distancias mínimas entre pares de ciudades de un mapa, indicando además la ruta a seguir para ir de la primera ciudad a la segunda. Este es uno de los problemas más interesantes que se pueden resolver con algoritmos de grafos.

Existen varias soluciones a este problema y los algoritmos a aplicar dependen también de la existencia de arcos con pesos o costes negativos en el grafo. En el caso de no existir pesos negativos, sería posible ejecutar V veces el Algoritmo de Dijkstra para el cálculo del camino mínimo, donde V es el número de vértices o nodos del grafo. Esto conllevaría un tiempo de ejecución de O(V3) (aunque se puede reducir). Si existen arcos con pesos negativos, se puede ejecutar también V veces el Algoritmo de Bellman-Ford, una vez para cada nodo del grafo. Para grafos densos (con muchas conexiones o arcos) esto conllevaría un tiempo de ejecución de O(V4).

El algoritmo de Floyd-Warshall ('All-Pairs-Shortest-Path' - Todos los caminos mínimos) ideado por Floyd en 1962 basándose en un teorema de Warshall también de 1962, usa la metodología de Programación Dinámica para resolver el problema. Éste puede resolver el problema con pesos negativos y tiempos de ejecución iguales a O(V3); sin embargo, para ciclos de peso negativo el algoritmo tiene problemas.

A continuación se muestra el pseudo código del Algoritmo:

```
        Floyd-Warshall (G)

        Inicializar
   D = A ' matriz de distancias = matriz de arcos
    si i=j o Dij= infinito entonces Pi,j = nulo sino Pi,j=i 'matriz de
caminos

        for k = 1 to V
    for i = 1 to V
       for j = 1 to V
          Di,j = min(Di,j , Di,k + Dk,j )
             si min = Di,k + Dk,j entonces
                Pi,j = Pk,j
fin
```

Este algoritmo se puede aplicar a multitud de problemas, incluyendo el diseño de circuitos, el diseño de rutas de transporte, aproximaciones al problema del viajante de comercio, o como base de otros algoritmos más complejos.

2.1.2.1. Ejemplo hecho a mano

Con el algoritmo de Floyd-Warshall se realizará otro ejemplo paso por paso. En este caso el objetivo es encontrar los mínimos caminos desde cada nodo con la finalidad de encontrar un camino que pase por todos los nodos. El grafo a estudiar será el siguiente:

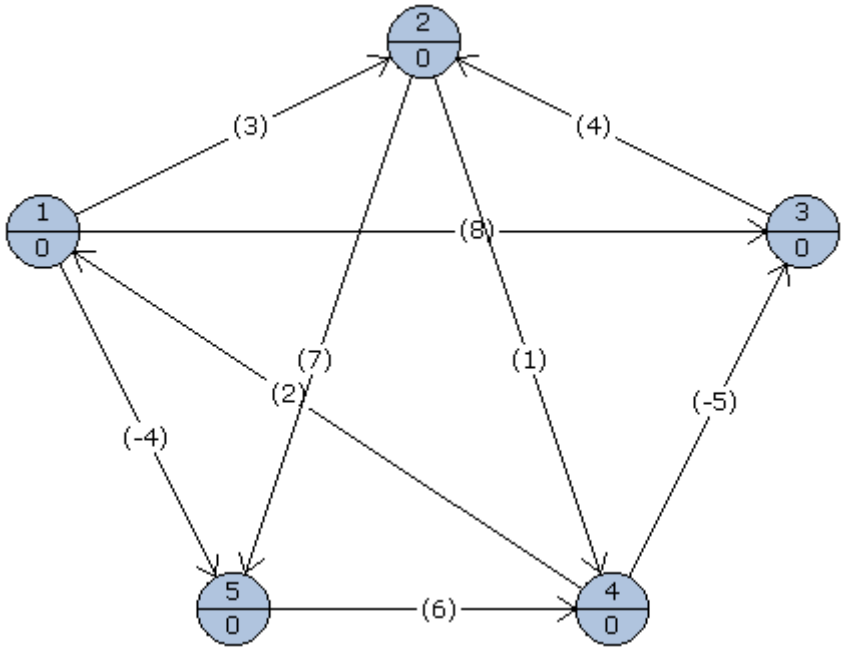


Ilustración 8 - Grafo A Mano Floyd Warshall

1.- El primer paso que hay que realizar es encontrar la matriz de distancias mínimas, eso es el camino con mínimo valor que va desde el nodo X al nodo Y, por ejemplo, para ir del nodo 2 al 5, comprobamos la distancia 2-4 y la distancia 2-5 (directamente), se observa que la mínima distancia nos la da 2-4, con un valor $d_1 = 1$.

La siguiente comprobación será la de ir desde 4 hasta otro nodo. Se observa que la mínima distancia desde el nodo 4 es ir al nodo 3. La marcamos, aunque temporalmente. Acto seguido comprobamos que desde 3 sólo podemos ir a 2, con lo que estamos en la misma situación que al inicio. Por tanto, borraremos esta solución, y en este paso iremos desde 4 hasta 1, con una distancia $d_2 = 2$.

Finalmente, observamos que desde 1 podemos ir al nodo 2, con un valor de 3 y volviendo a la situación inicial; y también podemos ir al nodo 5, nodo destino, con un valor $d_3 = -4$. Escogemos esta segunda opción y ya hemos realizado el camino mínimo desde 2 hasta 5.

El camino con valor mínimo es el 2-4-1-5, con un recorrido total de $d = 1 + 2 - 4 = -1$.

* Nota: La matriz de distancias mínimas nos la da la solución del programa, con lo cual, no la pondré en este apartado.

2.- El segundo y último paso a realizar es el listado de caminos, esto se refiere a realizar un estudio para obtener el mínimo recorrido desde el nodo I al nodo J. Voy a escribir los caminos mínimos para ir desde el nodo 1 y desde el nodo 2 hacia los demás nodos, puesto que esta es una tarea larga, ardua y pesada como para hacerla totalmente a mano.

- Para ir del nodo 1 con un recorrido de valor (1) hasta el nodo 2 hay que pasar por los nodos, 1, 5, 4, 3 y, finalmente, 2.

- Para ir del nodo 1 con un recorrido de valor (-3) hasta el nodo 3 hay que pasar por los nodos, 1, 5, 4 y, finalmente, 3.

- Para ir del nodo 1 con un recorrido de valor (2) hasta el nodo 4 hay que pasar por los nodos, 1, 5 y, finalmente, 4.

- Para ir del nodo 1 con un recorrido de valor (-4) hasta el nodo 5 hay que pasar por los nodos, 1 y 5.

- Para ir del nodo 2 con un recorrido de valor (3) hasta el nodo 1 hay que pasar por los nodos, 2, 4 y, finalmente, 1.

- Para ir del nodo 2 con un recorrido de valor (-4) hasta el nodo 3 hay que pasar por los nodos, 2, 4 y, finalmente, 3.

- Para ir del nodo 2 con un recorrido de valor (1) hasta el nodo 4 hay que pasar por los nodos, 2 y 4.

- Para ir del nodo 2 con un recorrido de valor (-1) hasta el nodo 5 hay que pasar por los nodos, 2, 4, 1 y, finalmente, 5.

4.- Finalmente, una vez tengamos todos estos recorridos (desde cada nodo a cada nodo), los marcamos en el grafo y obtenemos así el resultado final de todos los caminos mínimos.

2.1.2.2. Ejemplo hecho por el software

Tenemos esta serie de nodos, los mismos que para el ejemplo hecho a mano, unidos entre ellos según la distancia marcada en el grafo siguiente:

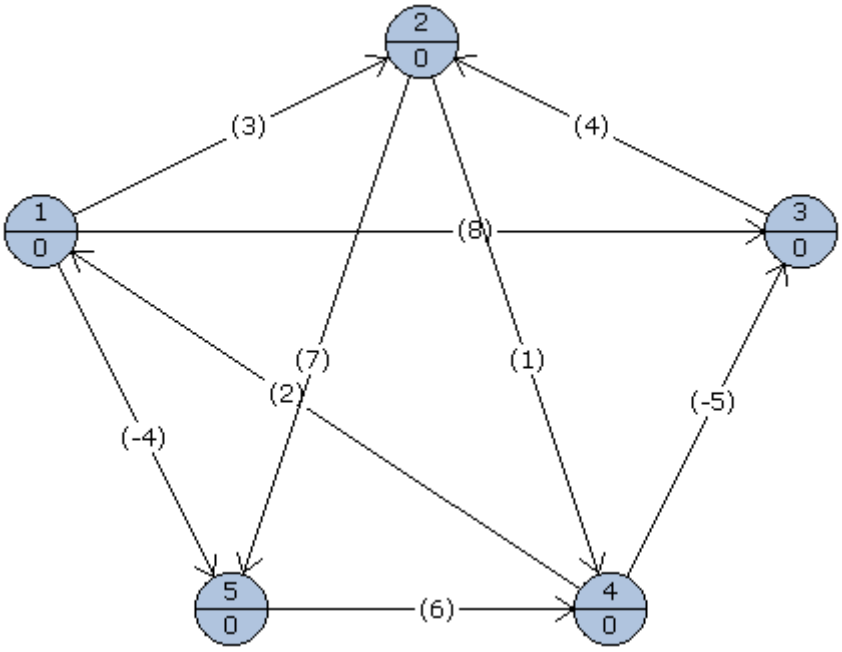


Ilustración 9 - Grafo Software Floyd Warshall 1

Si aplicamos el algoritmo con el software, obtenemos el siguiente resultado:

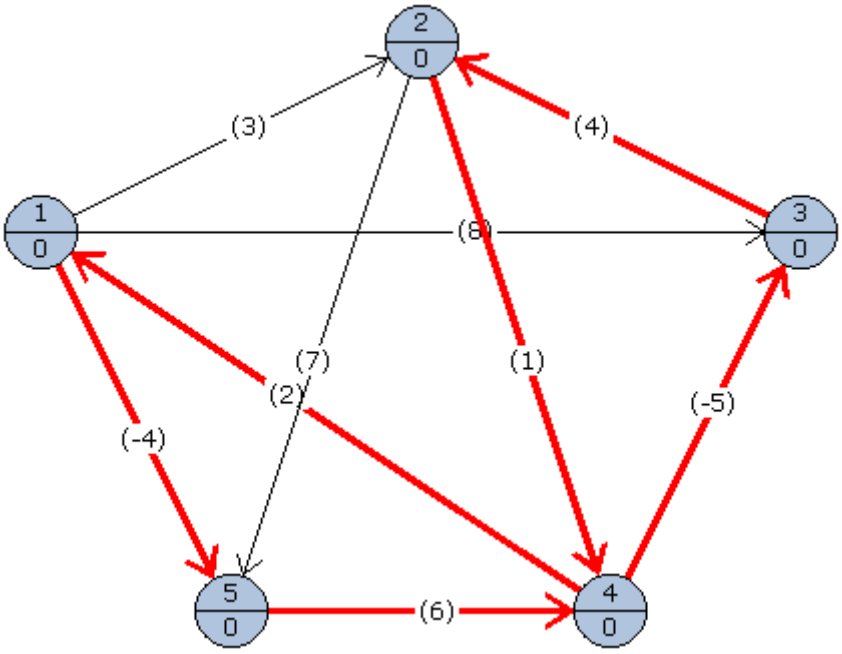


Ilustración 10 - - Grafo Software Floyd Warshall 2

ÁRBOL DE CAMINOS MÍNIMOS - ALGORITMO DE FLOYD WARSHALL

Matriz de Distancias mínimas:

N1\N2	1	2	3	4	5
1	0	1	-3	2	-4
2	3	0	-4	1	-1
3	7	4	0	5	3
4	2	-1	-5	0	-2
5	8	5	1	6	0

Listado de Caminos:

1 -- (0) → 1 =

1 -- (1) → 2 = 1, 5, 4, 3, 2

1 -- (-3) → 3 = 1, 5, 4, 3

1 -- (2) → 4 = 1, 5, 4

1 -- (-4) → 5 = 1, 5

2 -- (3) → 1 = 2, 4, 1

2 -- (0) → 2 =

2 -- (-4) → 3 = 2, 4, 3

2 -- (1) → 4 = 2, 4

2 -- (-1) → 5 = 2, 4, 1, 5

3 -- (7) → 1 = 3, 2, 4, 1

3 -- (4) → 2 = 3, 2

3 -- (0) → 3 =

3 -- (5) → 4 = 3, 2, 4

3 -- (3) → 5 = 3, 2, 4, 1, 5

4 -- (2) → 1 = 4, 1

4 -- (-1) → 2 = 4, 3, 2

4 -- (-5) → 3 = 4, 3

4 -- (0) → 4 =

4 -- (-2) → 5 = 4, 1, 5

5 -- (8) → 1 = 5, 4, 1

5 -- (5) → 2 = 5, 4, 3, 2

5 -- (1) → 3 = 5, 4, 3

5 -- (6) → 4 = 5, 4

5 -- (0) → 5 =

2.1.3. Bellman Ford

2.1.3.1. Camino mínimo

Soluciona el problema de la ruta más corta o camino mínimo desde un nodo origen, de un modo más general que el Algoritmo de Dijkstra, ya que permite valores negativos en los arcos.

El algoritmo devuelve un valor booleano si encuentra un circuito o lazo de peso negativo. En caso contrario calcula y devuelve el camino mínimo con su coste.

Para cada vértice v perteneciente a V, se mantiene el atributo d[v] como cota superior o coste del camino mínimo desde el origen s al vértice v.

A continuación se muestra el pseudo código del Algoritmo:

```
        Bellman-Ford (G,s)

        Inicializar
  for cada v perteneciente a V[G]
    do d[v] = infinito
       p[v] = nulo
  p[s] = 0

        for i=1 to V[G]-1
  do for cada arco (u,v) perteneciente a
A[G]
     Relajación
     if d[v] > d[u] + w(u,v) then
       d[v] = d[u] + w(u,v)
       p(v) = u

        for cada arco (u,v) chequea lazo de
peso negativo
  do if d[v] > d[u] + w(u,v) then
     return FALSO 'el algoritmo no converge
return VERDADERO
```

2.1.3.2. Ejemplo hecho a mano camino mínimo

El siguiente grafo es el que utilizaremos para mostrar el algoritmo:

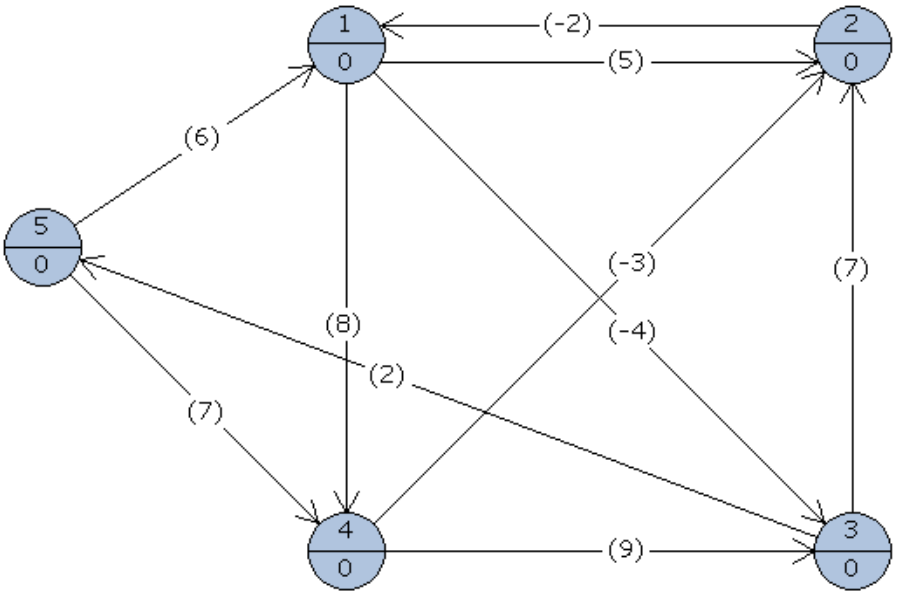


Ilustración 11 - Grafo Camino Mínimo A Mano Bellman Ford

1.- El primer paso que hay que realizar es seleccionar el nodo origen y el nodo destino, nuestro nodo inicial será el 1 mientras que el final será el 5. Seguidamente chequearemos cuál es el nodo con mínima distancia desde el nodo inicial. Vemos que podemos ir desde 1 hasta 2, con un peso de 5; podemos ir también al nodo 3, con un peso de -4; y podemos ir también al nodo 4, con un peso de 8. Seleccionaremos el recorrido 1-3 con un valor $d_1 = -4$.

2.- El segundo paso es igual que el primero, sólo que esta vez el nodo inicial lo ponemos en el nodo 3, manteniendo el valor de d_1 para el posterior cálculo de distancia total. Aquí vemos que desde 3 el camino mínimo es ir hasta el nodo 5, con un peso $d_2 = 2$.

3.- La distancia total será de $D = d_1 + d_2 = -4 + 2 = -2$.

2.1.3.3. Ejemplo hecho por el software camino mínimo

El siguiente grafo es el que utilizaremos para mostrar el algoritmo:

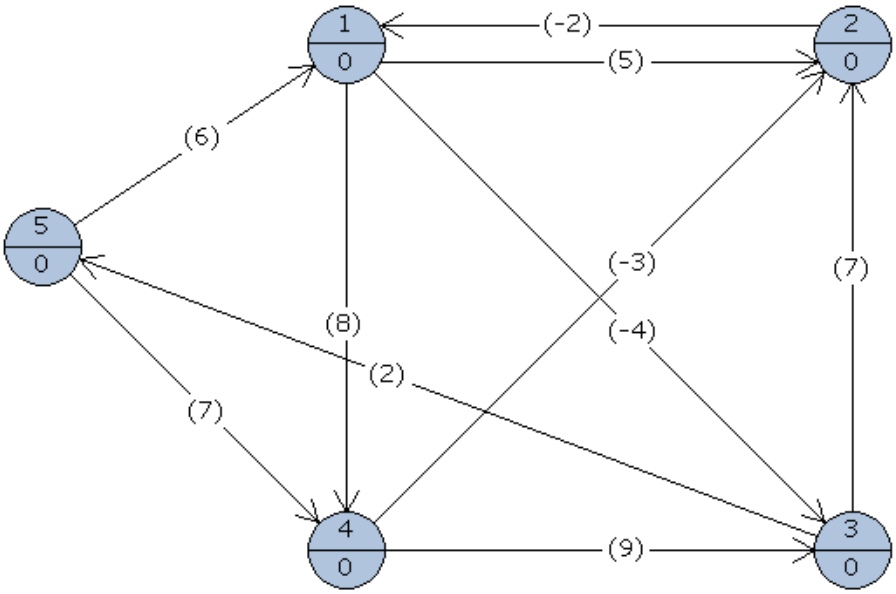


Ilustración 12 - Grafo Camino Mínimo Software Bellman Ford 1

La solución que da el programa, para ir del nodo 1 al 5, es:

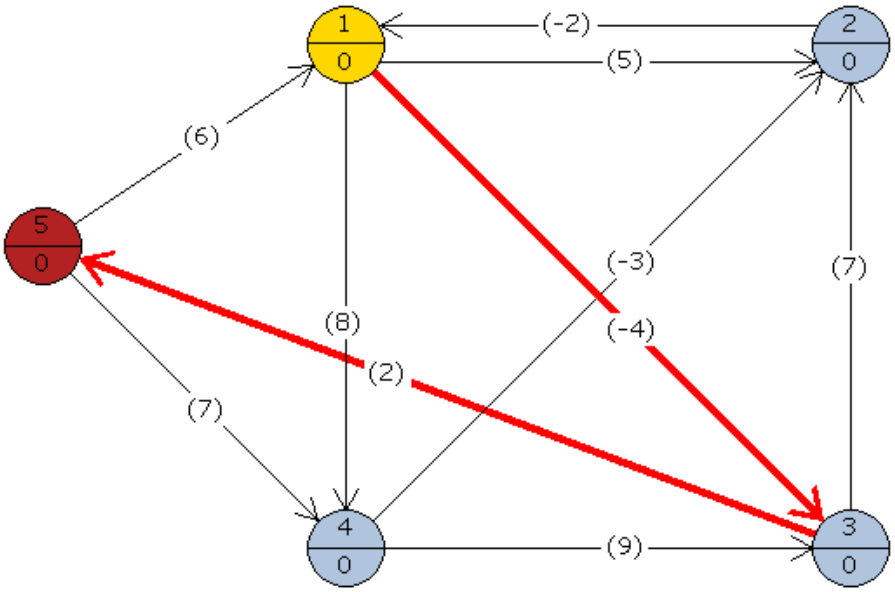


Ilustración 13 - Grafo Camino Mínimo Software Bellman Ford 2

CAMINO MÍNIMO - ALGORITMO DE BELLMAN-FORD

Arcos calculados desde el nodo origen (1) hasta el nodo destino (5):

* 1 -- (-4)→ 3
* 3 -- (2) → 5

$$Coste\ total = -2$$

Matriz de Arcos con coste mínimo:

N1\N2	1	2	3	4	5
1	0	0	1	0	0
2	0	0	0	0	0
3	0	0	0	0	1
4	0	0	0	0	0
5	0	0	0	0	0

2.1.3.4. Camino máximo

El problema de la ruta más larga puede ser transformado en el de ruta más corta cambiando el signo de los costes de los arcos.

De manera alternativa se puede transformar también cambiando los procesos de inicialización y relajación. En este caso el problema es inconsistente para circuitos de peso positivo.

```
        Inicializar
for cada v perteneciente a V[G]
   do d[v] = - infinito
      p[v] = nulo
p[s] = 0

        Relajación
   if d[v] < d[u] + w(u,v) then
      d[v] = d[u] + w(u,v)
      p(v) = u
```

2.1.3.5. Ejemplo hecho a mano camino máximo

El grafo con el que explicaremos el ejemplo es el siguiente, no se puede usar el del modelo de camino mínimo puesto que no es compatible para el cálculo de máximo camino:

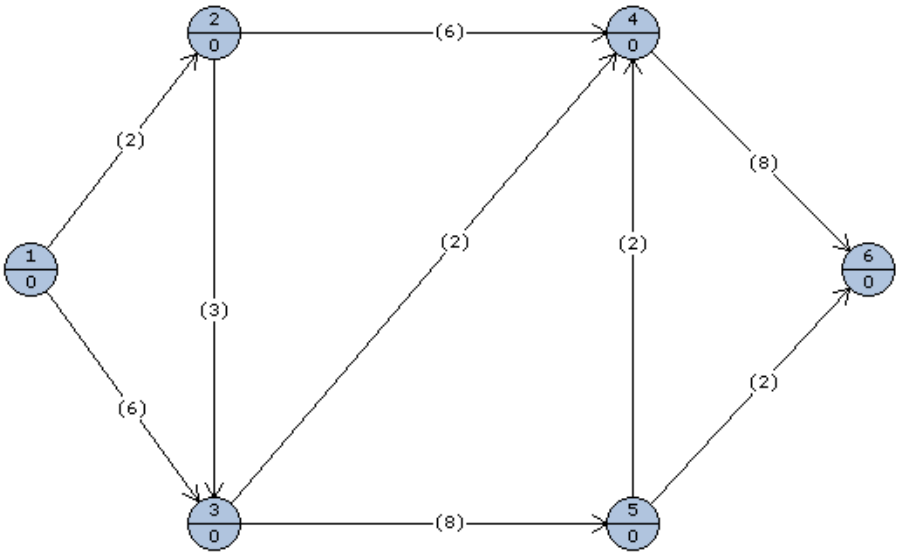


Ilustración 14 - Grafo Camino Máximo A Mano Bellman Ford

* Nota: El algoritmo se resuelve idénticamente al de camino mínimo, sólo que se toman valores máximos.

1.- El primer paso que hay que realizar es seleccionar el nodo origen y el nodo destino, nuestro nodo inicial será el 1 mientras que el final será el 6. Seguidamente chequearemos cuál es el nodo con mínima distancia desde el nodo inicial. Vemos que podemos ir desde 1 hasta 2, con un peso de 2; podemos ir también al nodo 3, con un peso de 6. Seleccionaremos el recorrido 1-3 con un valor $d_1 = 6$.

2.- El segundo paso es igual que el primero, sólo que esta vez el nodo inicial lo ponemos en el nodo 3, manteniendo el valor de d_1 para el posterior cálculo de distancia total. Aquí vemos que desde 3 el camino máximo es ir hasta el nodo 5, con un peso $d_2 = 8$.

El siguiente arco con valor máximo, desde 5, es ir al nodo 4 o al 6 directamente, pero como estamos en el caso de que buscamos un máximo camino, iremos al nodo 4, con un peso $d_3 = 2$.

Finalmente desde 4 ya si que podemos llegar a 6, con un recorrido de $d_4 = 8$.

3.- La distancia total será de $D = d_1 + d_2 + d_3 + d_4 = 6 + 8 + 2 + 8 = 24$.

2.1.3.6. Ejemplo hecho por el software

El grafo con el que explicaremos el ejemplo es el siguiente, no se puede usar el del modelo de camino mínimo puesto que no es compatible para el cálculo de máximo camino:

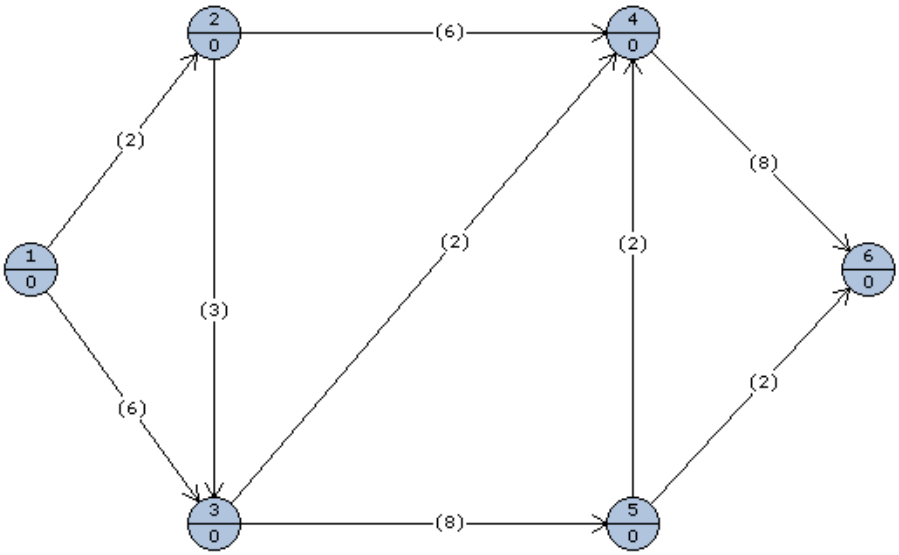


Ilustración 15 - Grafo Camino Máximo Software Bellman Ford 1

La solución que da el programa, para ir del nodo 1 al 6, con el máximo camino, es:

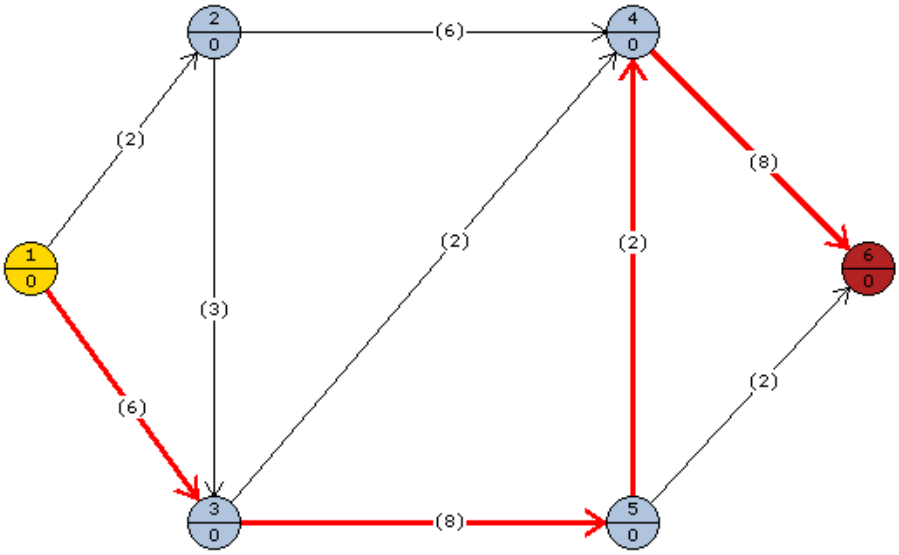


Ilustración 16 - Grafo Camino Máximo Software Bellman Ford 2

CAMINO MÁXIMO – BELLMAN FORD

Arcos calculados desde el nodo origen (1) hasta el nodo destino (6):

* 1 --(6) → 3
* 3 --(8) → 5
* 4 --(8) → 6
* 5 --(2) → 4

Coste total = 24

Matriz de Arcos con coste máximo:

N1\N2	1	2	3	4	5	6
1	0	0	1	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	1	0
4	0	0	0	0	0	1
5	0	0	0	1	0	0
6	0	0	0	0	0	0

2.2. Análisis de árboles:

2.2.1. Dijkstra

El algoritmo de Dijkstra resuelve tanto árboles máximos como árboles mínimos. A continuación veremos los dos métodos:

2.2.1.1. Algoritmo de Dijkstra (árbol mínimo)

Un algoritmo de trayectoria más corta, rutea cada vehículo a lo largo de la trayectoria de longitud mínima (ruta más corta) entre los nodos origen y destino. Hay varias formas posibles de seleccionar la longitud de los enlaces. La forma más simple es que cada enlace tenga una longitud unitaria, en cuyo caso, la trayectoria más corta es simplemente una trayectoria con el menor número de enlaces. De una manera más general, la longitud de un enlace puede depender de su capacidad de transmisión y su carga de tráfico.

La solución es encontrar la trayectoria más corta. Esperando que dicha trayectoria contenga pocos enlaces no congestionados; de esta forma los enlaces menos congestionados son candidatos a pertenecer a la ruta. Hay algoritmos de ruteo especializados que también pueden permitir que la longitud de cada enlace cambie en el tiempo, dependiendo del nivel de tráfico de cada enlace. De esta forma un algoritmo de ruteo se debe adaptar a sobrecargas temporales y rutear paquetes alrededor de nodos congestionados. Dentro de este contexto, el algoritmo de ruta más corta para ruteo opera continuamente, determinando la trayectoria más corta con longitudes que varían en el tiempo.

2.2.1.2. Algoritmo de Dijkstra (árbol máximo)

El camino crítico estará formado por tareas críticas (nodos) cuya duración (coste del arco sucesor) determina la duración total de un proyecto. Si una tarea crítica se retrasa o su duración cambia durante la realización del proyecto, afectaría directamente a la duración total del proyecto y a su fecha de finalización.

Encontrar el camino crítico de la planificación de un proyecto es lo mismo que encontrar el camino más largo desde el nodo inicial (tarea inicial) al nodo final (última tarea); esto es, la mínima cantidad de tiempo necesaria para finalizar un proyecto.

El algoritmo de Dijkstra aunque fue diseñado para encontrar la ruta más corta se puede transformar fácilmente para encontrar la ruta más larga (camino crítico), cambiando simplemente su función objetivo. Del mismo modo, se encuentra el árbol máximo desde un nodo origen.

A continuación se muestra el pseudo código del Algoritmo:

```
        Dijkstra (G,s)

        Inicializar
   for cada v perteneciente a V[G]
     do d[v] = infinito
      p[v] = nulo
      d[s] = 0

        S =  vacío
Q = V[G]

        mientras Q no vacío
do u = nodo v con min d[v]
 S = S unión u 'se añade al conjunto de nodos
finalizados

        for cada v perteneciente Adyacente u
   Relajación
     if d[v] > d[u] + w(u,v) then
        d[v] = d[u] + w(u,v)
        p(v) = u
```

2.2.1.3. Ejemplo Dijkstra:

Tenemos el siguiente grafo y queremos encontrar el árbol mínimo según el algoritmo de Dijsktra:

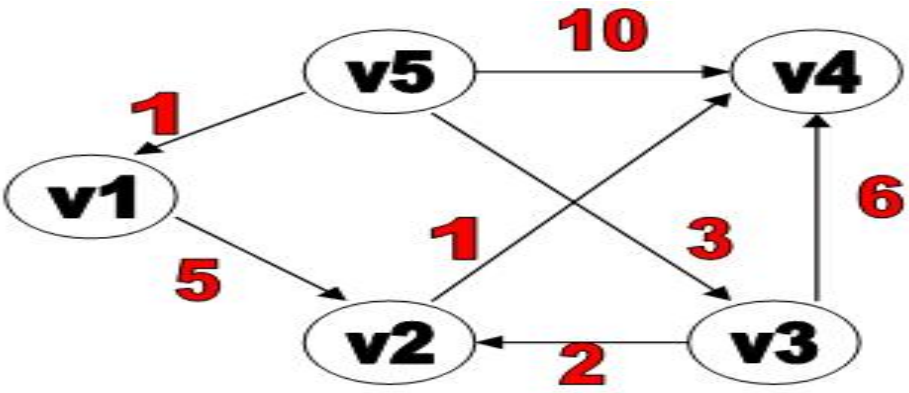


Ilustración 17 - Enunciado de Dijsktra (Árboles)

Dibujo del grafo:

Lo primero que hemos de hacer es dibujar los nodos que se indican en el grafo según el tutorial anterior, con lo que obtenemos el siguiente grafo (Previamente en el menú formato hemos de indicar que de los arcos sólo queremos su peso, y que de los nodos solo la etiqueta). Podemos observar que aquí si que importa el sentido de los arcos, por eso van con su flecha correspondiente:

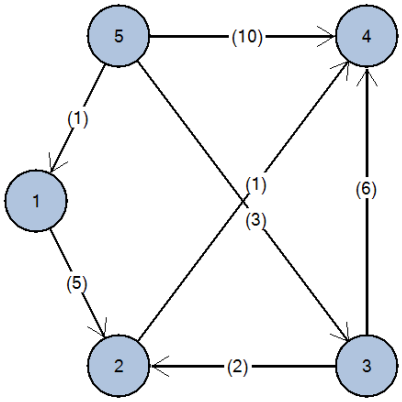


Ilustración 18 - Grafo Ejemplo Dijsktra (Árboles)

2.2.1.4. Solución:

Para solucionarlo mediante el algoritmo de Dijkstra, hemos de ir al icono de cálculo de arboles.

Resolviendo de forma manual, sería de la siguiente manera:

Siguiendo el algoritmo de Kruskal, y cogiendo el nodo 1 como nodo inicial, tenemos:

- Partimos del nodo 1, y calculamos la longitud al resto de los nodos (2 3 4 5), y nos da 5-inf- inf- inf. Esto nos indica, que del nodo 1 al nodo 2 nos "cuesta" 5, del 1 al 3 infinito y el resto lo mismo.

- En la siguiente iteración (iteración 2) seleccionamos el nodo 2 en W, y a partir del nodo 1 y 2 calculamos la distancia mínima hasta el resto de los nodos, es decir, del nodo 1 y 2 conjuntamente al nodo 3, nos da infinito por que nos podemos llegar a el de ninguna manera, del nodo 1 y 2 al nodo 4 si se modifica con una distancia 6, esto quiere decir que para ir al nodo 4 a partir del 1, nos cuesta 6 pasando previamente por el nodo 2. Esto ocurre en el vector P, en el vector D se modifica la posición 4 con un 2 indicando que pasamos por el nodo 2 para llegar al nodo 4.

- En la siguiente iteración (iteración 3) ya no seguimos por que no se ha modificado el vector P, lo que nos indica que no ponemos ninguna distancia mínima nueva y por tanto el algoritmo acaba.

- Para obtener la solución visitamos el vector P partiendo de la posición 1 y es: {inf 5 inf 6 inf} que nos indica que para ir del nodo 1 al 1 cuesta infinito, para ir al 2 5, para el 3 infinito, para el 4 cuesta 6 y para el 5 infinito. Para obtener el camino consultamos el vector D que contiene lo siguiente: {-- 1 -- 2 --}, que nos indica que para ir al nodo 1 a través del 1 no hay camino, para ir al 2 vamos por el 1, para ir al 3 no hay camino, para ir al 4 vamos a través del nodo 2, y para ir al 5 no hay camino.

iteracion	S	W	2 3 4 5	2 3 4 5
1	1	--	5 inf inf inf	1 -- -- --
2	1,2	2	5 inf 6 inf	1 -- 2 --
3	1,2,4	4	5 inf 6 inf	1 -- 2 --

Ilustración 19 - Resolución Ejemplo Dijkstra (Árboles)

Si resolvemos con el software Grafos, obtenemos el siguiente Report. Para poderlo solucionarlo mediante el programa antes hemos de clicar con el botón izquierdo sobre el nodo inicial y éste se volverá de color amarillo, con esto ya podemos ir al icono de árboles y solucionar.

ÁRBOL MÍNIMO - ALGORITMO DE DIJKSTRA

Tiempo de proceso = 0 segundos

Arcos calculados desde el nodo origen (1):

1 ----(5)---> 2

2 ----(1)---> 4

Matriz de Arcos con coste mínimo:

N1\N2	1	5	2	4	3
1	0	0	1	0	0
5	0	0	0	0	0
2	0	0	0	1	0
4	0	0	0	0	0
3	0	0	0	0	0

Aquí podemos analizar en la primera parte que del nodo 1 vamos al 2, con un peso de 5. Del nodo 2 vamos al 4 con un peso de 1.

Tenemos también que el valor del árbol de coste mínimo es la suma de los dos que es 6.

En la matriz lo que vemos es con cuantos nodos comunica cada nodo.

El programa también te devuelve el árbol mínimo sobre el grafo original que hemos dibujado, obteniendo el siguiente grafo:

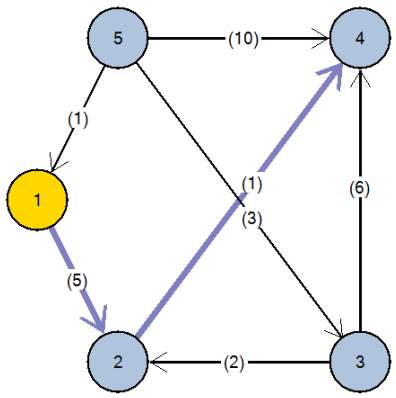


Ilustración 20 - Report Grafos Ejemplo Dijkstra (Árboles) Mínimo

Claramente en éste ejemplo es lo mismo el árbol de mínimo que el de máximo. Ya que la configuración de este grafo partiendo desde 1 solo permite una única solución. Pero si cogemos otro ejemplo podemos ver que diferente:

Ejemplo 2 Mínimo:

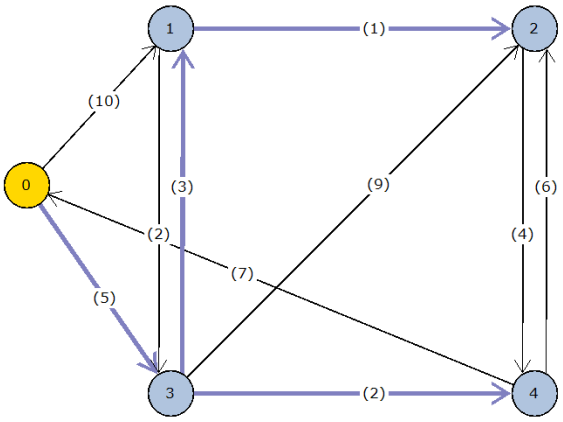


Ilustración 21 - REPORT GRAFOS EJEMPLO DIJKSTRA (ÁRBOLES) MÍNIMO2

Ejemplo 2 Máximo:

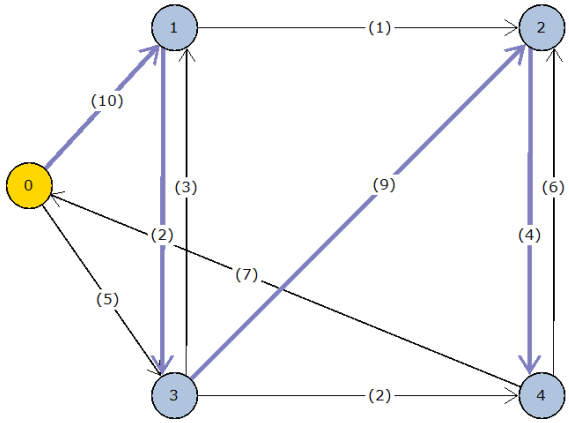


Ilustración 22 - REPORT GRAFOS EJEMPLO DIJKSTRA (ÁRBOLES) MÁXIMO 2

2.2.2. Kruskal

El objetivo del algoritmo de Kruskal es construir un árbol (subgrafo sin ciclos) formado por arcos sucesivamente seleccionados de mínimo peso a partir de un grafo con pesos en los arcos. Un árbol (spanning tree) de un grafo es un subgrafo que contiene todos sus vértices o nodos. Un grafo puede tener múltiples árboles.

La aplicación típica de este problema es el diseño de redes telefónicas. Una empresa con diferentes oficinas, trata de trazar líneas de teléfono para conectarlas unas con otras. La compañía telefónica le ofrece esta interconexión, pero ofrece tarifas diferentes o costes por conectar cada par de oficinas. Cómo conectar entonces las oficinas al mínimo coste total.

Otra aplicación menos obvia es que el árbol de coste total mínimo puede ser usado como solución aproximada al problema del viajante de comercio (traveling salesman problem). La manera formal de definir este problema es encontrar la trayectoria más corta para visitar cada punto al menos una vez. Nótese que si se visitan todos los puntos exactamente una vez, lo que se tiene es un tipo especial de árbol. Si se tiene una trayectoria que visita algunos vértices o nodos más de una vez, siempre se puede soltar algunos nodos del árbol. En general el peso del árbol total mínimo es menor que el del viajante de comercio, debido a que su minimización se realiza sobre un conjunto estrictamente mayor.

2.2.2.1. Algoritmo de Kruskal (árbol de coste total mínimo)

Dado un grafo G con nodos conectados por arcos con peso (coste o longitud): el peso o coste total de un árbol será la suma de pesos de sus arcos. Obviamente, árboles diferentes tendrán un coste diferente. El problema es entonces ¿cómo encontrar el árbol de coste total mínimo?

Una manera de encontrar la solución al problema del árbol de coste total mínimo, es la enumeración completa. Aunque esta forma de resolución es eficaz, no se puede considerar un algoritmo, y además no es nada eficiente.

El algoritmo se basa en una propiedad clave de los árboles que permite estar seguros de si un arco debe pertenecer al árbol o no, y usar esta propiedad para seleccionar cada arco. Nótese en el algoritmo, que siempre que se añade un arco (u,v), éste será siempre la conexión más corta (menor coste) alcanzable desde el nodo u al resto del grafo G. Así que por definición éste deberá ser parte del árbol.

Este algoritmo es de tipo greedy, ya que a cada paso, éste selecciona el arco más barato y lo añade al subgrafo. Este tipo de algoritmos pueden no funcionar para resolver otro tipo de problemas, por ejemplo para encontrar la ruta más corta entre los nodos a y b.

Para simplificar, se asumirán que existe un único árbol de coste total mínimo, aunque en muchos problemas puede existir más de una solución óptima de igual valor total mínimo.

De la misma manera que podemos encontrar el árbol de mínimo peso, podemos encontrar el de mayor peso cogiendo los arcos de mayor coste.

A continuación se muestra el pseudocódigo del Algoritmo:

```
Kruskal (G)

E(1)=0, E(2)= todos los Arcos del grafo
G
  Mientras E(1) contenga menos de n-1 arcos
y E(2)=0 do
    De los arcos de E(2) seleccionar el de
menor coste -->e(ij)
    E(2)= E(2) - {e(ij)}
      Si V(i), V(j) no están en el mismo árbol
entonces
        juntar los árboles de  V(i) y de V(j) en
uno sólo
      end Si
  end do
Fin del algoritmo
```

2.2.2.2. Ejemplo Kruskal:

Tenemos el siguiente grafo y queremos encontrar el árbol mínimo según el algoritmo de Kruskal:

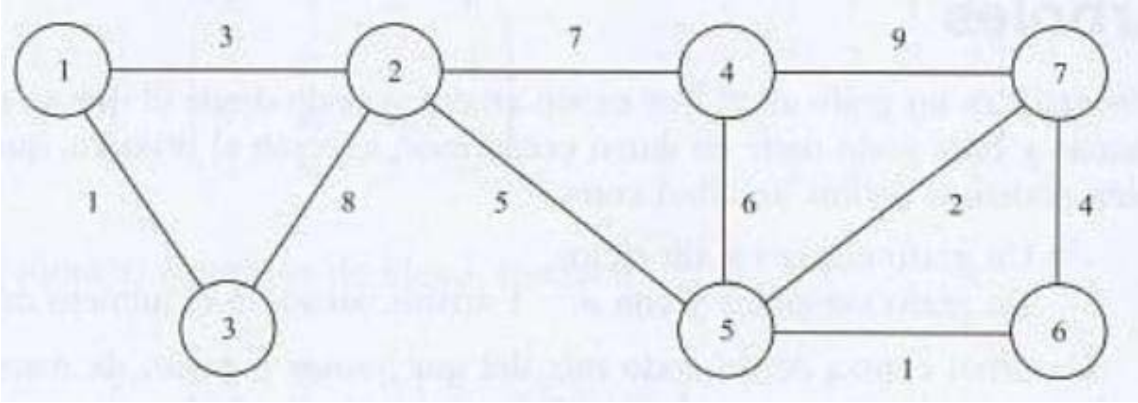


Ilustración 23 - Enunciado Ejemplo Kruskal

Dibujo del grafo:

Lo primero que hemos de hacer es dibujar los nodos que se indican en el grafo según el tutorial anterior, con lo que obtenemos el siguiente grafo (Previamente en el menú formato hemos de indicar que de los arcos sólo queremos su peso, y que de los nodos solo la etiqueta):

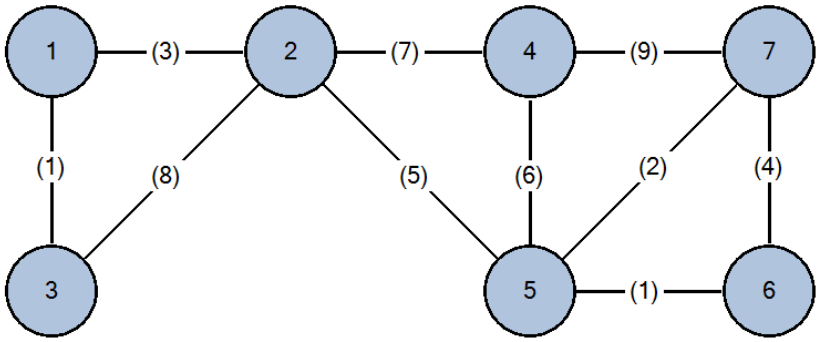


Ilustración 24 - Grafo Enunciado Ejemplo Kruskal

2.2.2.3. Solución:

Para solucionarlo mediante el algortimo de Kruskal, hemos de ir al icono de cálculo de arboles.

En este caso podemos ver cómo podemos calcular el árbol mínimo, y el máximo, en nuestro caso calculamos el mínimo. El árbol máximo se calcularía de la misma manera que el mínimo.

Resolviendo de forma manual, sería de la siguiente manera:

Siguiendo el algoritmo de Kruskal, y cogiendo la arista de menor valor, tenemos:

- Elegimos, por ejemplo, la arista ***(5, 6)*** *= 1* (menor valor) y la marcamos.
- Elegimos la siguiente arista con menor valor ***(1, 3)*** *= 1* y la marcamos.
- Elegimos la siguiente arista con menor valor ***(5, 7)*** *= 2* y la marcamos, ya que no forma ciclos con ninguna arista de las marcadas anteriormente.
- Elegimos la siguiente arista con menor valor ***(1, 2)*** *= 3* y la marcamos, ya que no forma ciclos con ninguna arista de las marcadas anteriormente.
- Elegimos la siguiente arista con menor valor *(6, 7) = 4* y la desechamos, ya que forma ciclos con las aristas *(5, 7)* y *(5, 6)* marcadas anteriormente.
- Elegimos la siguiente arista con menor valor ***(2, 5)*** *= 5* y la marcamos, ya que no forma ciclos con ninguna arista de las marcadas anteriormente.
- Elegimos la siguiente arista con menor valor ***(4, 5)*** *= 6* y la marcamos, ya que no forma ciclos con ninguna arista de las marcadas anteriormente.
- FIN. Finalizamos dado que los *7* nodos del grafo están en alguna de las aristas, o también ya que tenemos marcadas *6* aristas (*n-1*).

La suma de las aristas elegidas nos da el árbol de peso mínimo, que vale 1+1+2+3+5+6=18.

Si resolvemos con el software Grafos, obtenemos el siguiente Report:

ÁRBOL DE VALOR TOTAL MÍNIMO - ALGORITMO DE KRUSKAL

Tiempo de proceso = 0 segundos

* 1 ----(3)---> 2
* 1 ----(1)---> 3
* 2 ----(5)---> 5
* 5 ----(6)---> 4
* 5 ----(2)---> 7
* 5 ----(1)---> 6

Coste total = 18

Matriz de Arcos del árbol con coste mínimo:

N1\N2	1	2	3	4	7	5	6
1	0	1	1	0	0	0	0
2	0	0	0	0	0	1	0
3	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0
5	0	0	0	1	1	0	1
6	0	0	0	0	0	0	0

Aquí podemos analizar en la primera parte que del nodo 1 vamos al 2, con un peso de 3. Del nodo 1 vamos al 3 con un peso de 1. Del nodo 2 vamos al 5 con un peso de 5. Del nodo 5 vamos al 4 con un peso de 6. Del nodo 5 vamos al 7 con un peso de 2, y del nodo 5 vamos al 6 con un peso de 1.

Tenemos también que el valor del árbol de coste mínimo es de 18.

En la matriz lo que vemos es con cuantos nodos comunica cada nodo.

El programa también te devuelve el árbol mínimo sobre el grafo original que hemos dibujado, obteniendo el siguiente grafo:

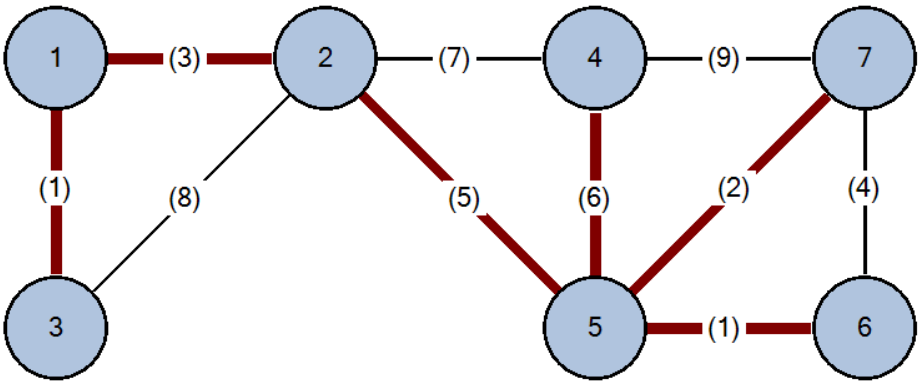


Ilustración 25 - Report Ejemplo Kruskal Mínimo

En el caso del árbol máximo:

ÁRBOL DE VALOR TOTAL MÁXIMO - ALGORITMO DE KRUSKAL

Tiempo de proceso = 0 segundos

* 1 ----(3)---> 2
* 2 ----(8)---> 3
* 2 ----(7)---> 4
* 4 ----(9)---> 7
* 7 ----(4)---> 6
* 5 ----(6)---> 4

Coste total = 37

Matriz de Arcos del árbol con coste máximo:

N1\N2	1	2	3	4	7	5	6
1	0	1	0	0	0	0	0
2	0	0	1	1	0	0	0
3	0	0	0	0	0	0	0
4	0	0	0	0	1	0	0
7	0	0	0	0	0	0	1
5	0	0	0	1	0	0	0
6	0	0	0	0	0	0	0

El programa también te devuelve el árbol máximo sobre el grafo original que hemos dibujado, obteniendo el siguiente grafo:

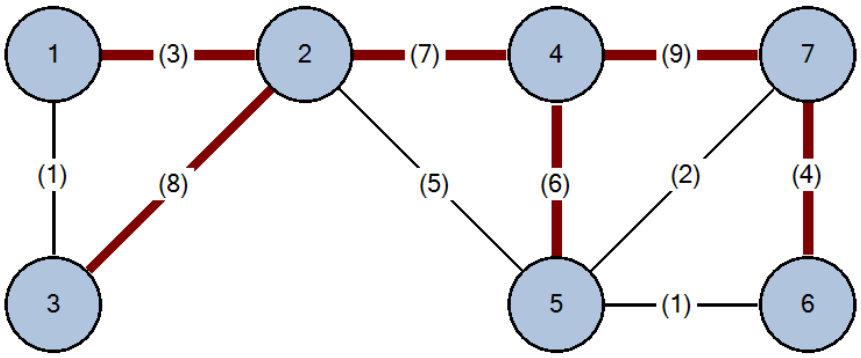


Ilustración 26 - Report Ejemplo Kruskal Máximo

2.2.3. Prim

2.2.3.1. Algortimo Prim

Consiste en dividir los nodos de un grafo en dos conjuntos: procesados y no procesados. Al principio, hay un nodo en el conjunto procesado que corresponde al equipo central; en cada interacción se incrementa el grafo de procesados en un nodo (cuyo arco de conexión es mínimo) hasta llegar a establecer la conexión de todos los nodos del grafo a procesar.

De la misma manera podemos calcular el árbol de coste máximo.

A continuación se muestra el pseudocódigo del Algoritmo:

```
Prim ( L [1..n , 1..n ]) : 'conjunto de arcos

'Inicialización: sólo el nodo 1 se encuentra en B

T =NULL 'T contendrá los arcos del árbol de extensión
mínima Distmin[1]=-1

para i=2 hasta n hacer
  más_próximo [ i ]=1
  distmin [ i ]=L [ i , 1]

para i=1 hasta n -1 hacer
  min=infinito

      para j=2 hasta n hacer
    si 0 <= distmin [ j ] < min entonces
    min=distmin [ j ]
    k=j

      T=T union {{mas_próximo [ k ], k }}
  distmin [ k ]= -1 'se añade k a B

      para j=2 hasta n hacer
   si L [ j , k ] < distmin [ j ] entonces
   distmin [ j ]=L [ j , k ]
   más_próximo [ j ]=k

  devolver T
```

2.2.3.2. Ejemplo Prim:

Tenemos el siguiente grafo y queremos encontrar el árbol mínimo según el algoritmo de Prim:

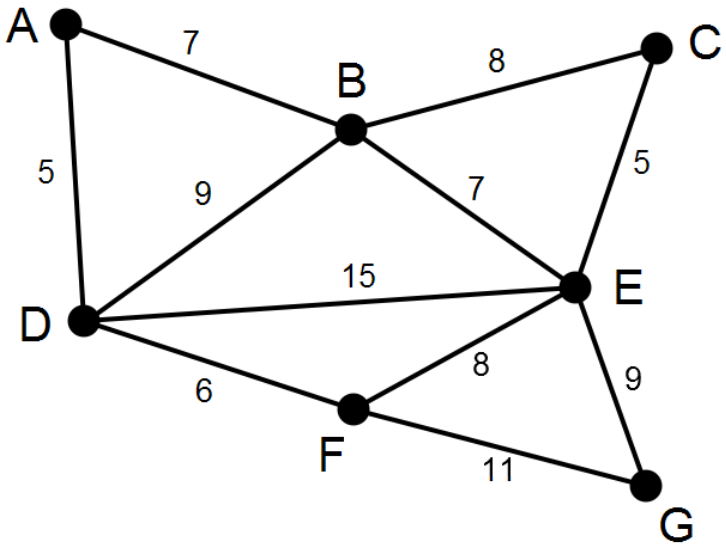


Ilustración 27 - Enunciado Ejemplo Prim

Dibujo del grafo:

Lo primero que hemos de hacer es dibujar los nodos que se indican en el grafo según el tutorial anterior, con lo que obtenemos el siguiente grafo (Previamente en el menú formato hemos de indicar que de los arcos sólo queremos su peso, y que de los nodos solo la etiqueta):

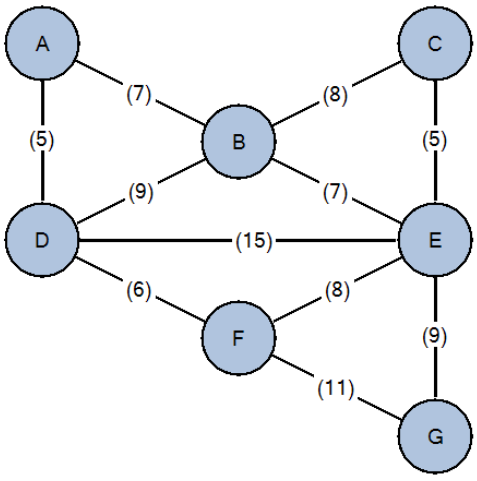


Ilustración 28 - Grafo Ejemplo Prim

2.2.3.3. Solución:

Para solucionarlo mediante el algortimo de Prim, hemos de ir al icono de cálculo de arboles.

En este caso podemos ver cómo podemos calcular el árbol mínimo, y el máximo, en nuestro caso calculamos el mínimo. El árbol máximo se calcularía de la misma manera que el mínimo.

Resolviendo de forma manual, sería de la siguiente manera:

Siguiendo el algoritmo de Prim, y cogiendo el nodo D como nodo inicial, tenemos:

- El segundo vértice es el más cercano a **D**: **A** está a 5 de distancia, **B** a 9, **E** a 15 y **F** a 6. De estos, 5 es el valor más pequeño, así que marcamos la arista **DA**.
- El próximo vértice a elegir es el más cercano a **D** o **A**. **B** está a 9 de distancia de **D** y a 7 de **A**, **E** está a 15, y **F** está a 6. 6 es el más chico, así que marcamos el vértice **F** y a la arista **DF**.
- El algoritmo continua. El vértice **B**, que está a una distancia de 7 de **A**, es el siguiente marcado. En este punto la arista **DB** es marcada.
- Aquí hay que elegir entre **C**, **E** y **G**. **C** está a 8 de distancia de **B**, **E** está a 7 de distancia de **B**, y **G** está a 11 de distancia de **F**. **E** está más cerca, entonces marcamos el vértice **E** y la arista **EB**.
- Sólo quedan disponibles **C** y **G**. **C** está a 5 de distancia de **E**, y **G** a 9 de distancia de **E**. Se elige **C**, y se marca con el arco **EC**. El arco **BC** también se marca.
- **G** es el único vértice pendiente, y está más cerca de **E** que de **F**, así que se agrega **EG** al árbol. Todos los vértices están ya marcados.

  En este caso, la suma de pesos tiene un valor de 39.

Si resolvemos con el software Grafos, obtenemos el siguiente Report:

ÁRBOL DE VALOR TOTAL MÍNIMO - ALGORITMO DE PRIM

Tiempo de proceso = 0 segundos

* A ----(5)---> D
* A ----(7)---> B
* D ----(6)---> F
* B ----(7)---> E
* C ----(5)---> E
* E ----(9)---> G

Coste total = 39

Matriz de Arcos del árbol con coste mínimo:

N1\N2	A	D	B	C	E	F	G
A	0	1	1	0	0	0	0
D	0	0	0	0	0	1	0
B	0	0	0	0	1	0	0
C	0	0	0	0	1	0	0
E	0	0	0	0	0	0	1
F	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0

Aquí podemos analizar en la primera parte que del nodo A vamos al D, con un peso de 5. Del nodo A vamos al B con un peso de 7. Del nodo D vamos al F con un peso de 6. Del nodo B vamos al E con un peso de 7. Del nodo C vamos al E con un peso de 5, y del nodo E vamos al G con un peso de 9.

Tenemos también que el valor del árbol de coste mínimo es de 18.

En la matriz lo que vemos es con cuantos nodos comunica cada nodo.

El programa también te devuelve el árbol mínimo sobre el grafo original que hemos dibujado, obteniendo el siguiente grafo:

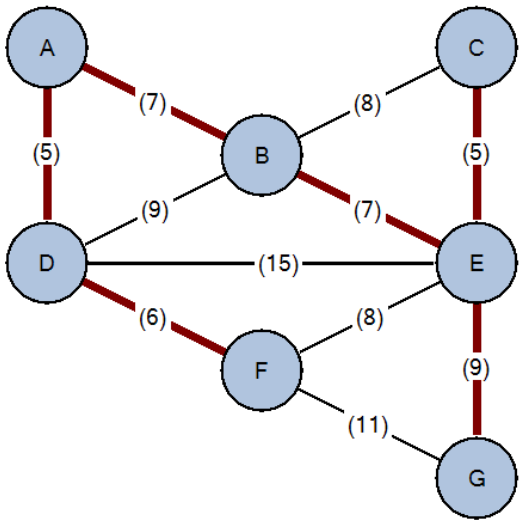


Ilustración 29 - Report Ejemplo Prim (Mínimo)

Para el caso del árbol máximo tenemos:

ÁRBOL DE VALOR TOTAL MÁXIMO - ALGORITMO DE PRIM

Tiempo de proceso = 0 segundos

* A ----(7)---> B
* D ----(15)---> E
* B ----(9)---> D
* B ----(8)---> C
* E ----(9)---> G
* F ----(11)---> G

Coste total = 59

Matriz de Arcos del árbol con coste máximo:

N1\N2	A	D	B	C	E	F	G
A	0	0	1	0	0	0	0
D	0	0	0	0	1	0	0
B	0	1	0	1	0	0	0
C	0	0	0	0	0	0	0
E	0	0	0	0	0	0	1
F	0	0	0	0	0	0	1
G	0	0	0	0	0	0	0

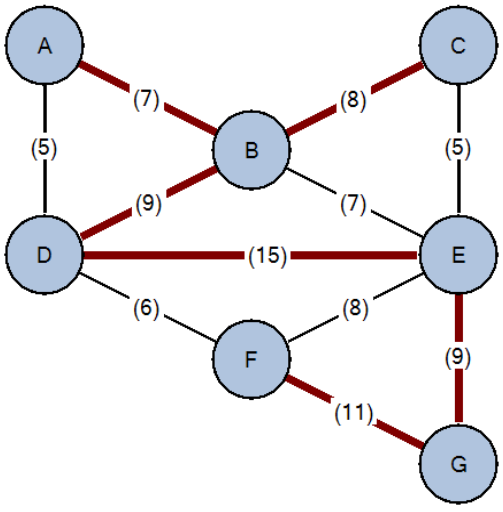


Ilustración 30 - Report Ejemplo Prim (Máximo)

2.3. Análisis de flujos:

2.3.1. Flujo Máximo:

El problema a resolver en esta edición es el problema del flujo máximo en una red de flujo. Este es un problema de enorme interés que encuentra aplicación en numerosas situaciones de la vida real. Por ejemplo, el problema del flujo máximo aparece en problemas de redes de distribución de mercancías. Un empresa tiene diferentes puntos de producción, diferentes puntos de almacenamiento y diferentes puntos de venta. Las mercancías de trasladan desde los puntos de producción a los puntos de almacenamiento, y posteriormente se distribuyen entre los puntos de venta. Todo este proceso se realiza a través de una red distribución en la cual cada conexión tiene una capacidad determinada. El problema de flujo máximo permite determinar cual es el flujo de mercancías que se debe establecer entre los diferentes puntos de conexión de tal manera que no falten productos en los puntos de venta.

Otras posibles aplicaciones relacionadas son redes de transmisión de datos, control de tráfico, problemas de planificación, problemas de geometría que encuentran aplicación en sistemas de radar, localización de puntos débiles en un red de distribución, de suministro, etc.

Una red de flujo G = (V, E) es un grafo dirigido en el cual cada arista (u, v) $\in$ E tiene asociada una capacidad c(u, v) $\geq$ 0. Si una arista no pertenece a la red de flujo, entonces se supone que su capacidad es 0. Una red de flujo tiene dos nodos especiales denominados fuente y sumidero. Se supone que todo nodo está en un camino que va de la fuente al sumidero.

Un flujo en una red de flujo es un conjunto de pesos no negativos de aristas que satisfacen las condiciones de que ningún peso es mayor que la capacidad de la arista y que el flujo total que entra en un nodo es igual al flujo total que sale del nodo (excepto para los nodos fuente y sumidero).

De manera más formal, sea G = (V, E) una red de flujo con una fuente s y un sumidero t. Un flujo en G es un función f : V×V → R que cumple tres propiedades:

Restricción de capacidad: para todo u, v $\in$ V, se debe cumplir que f (u, v) $\leq$ c(u, v).

Simetría inversa: para todo u, v $\in$ V, se debe cumplir que f (u, v) = - f (v, u).

Conservación de flujo: para todo u $\in$ V - {s, t}, se debe cumplir que $\sum_{\in v}$ f (u, v) = 0.

El valor f (u, v) se denomina flujo del nodo u al nodo v. El valor de un flujo f (denotado como | f|) se define como: | f| = $\sum_{\in v}$ f (s, v), es decir, el flujo total que sale de la fuente.

Dada una red de flujo G con una fuente s y un sumidero t, el problema del flujo máximo consiste en encontrar un flujo de valor máximo.

El problema del flujo máximo puede resolverse con el método de Ford-Fulkerson. A continuación describimos este método.

2.3.1.1. Algoritmo de Ford-Fulkersson

Se puede considerar un grafo como una red de flujo. Donde un nodo fuente produce o introduce en la red cierta cantidad de algún tipo de material, y un nodo sumidero lo consume. Cada arco, por tanto, puede considerarse como un conducto que tiene cierta capacidad de flujo. De igual modo que en redes eléctricas (Ley de Kirchhoff), la suma de flujos entrantes a un nodo, debe ser igual a la suma de los salientes (principio de conservación de energía), excepto para el nodo fuente y el nodo sumidero.

Por tanto, el problema de flujo máximo se enuncia como: ¿cuál es la tasa a la cual se puede transportar el material desde el nodo fuente al nodo sumidero, sin violar las restricciones de capacidad?.

Este algoritmo se puede usar para resolver modelos de: transporte de mercancías (logística de aprovisionamiento y distribución), flujo de gases y líquidos por tuberías, componentes o piezas en líneas de montaje, corriente en redes eléctricas, paquetes de información en redes de comunicaciones, tráfico ferroviario, sistema de regadíos, etc.

Una red de flujo es un grafo dirigido G=(V,E) donde cada arco (u,v) perteneciente a E tiene una capacidad no negativa. Se distinguen dos nodos: la fuente o nodo s, y el sumidero o nodo t. Si existen múltiples fuentes y sumideros, el problema se puede simplificar añadiendo una fuente común y un sumidero común.

Este algoritmo depende de tres conceptos principales:

- Un camino de aumento, es una trayectoria desde el nodo fuente s al nodo sumidero t que puede conducir más flujo.
- La capacidad residual es la capacidad adicional de flujo que un arco puede llevar cf (u,v) = c(u,v) - f(u,v)
- Teorema de Ford-Fulkerson (1962): En cualquier red, el flujo máximo que fluye de la fuente al destino es igual a la capacidad del corte mínimo que separa a la fuente del destino.

El algoritmo de Ford-Fulkerson propone buscar caminos en los que se pueda aumentar el flujo, hasta que se alcance el flujo máximo.

La idea es encontrar una ruta de penetración con un flujo positivo neto que una los nodos origen y destino.

Consideraremos las capacidades iniciales del arco que une el nodo i y el nodo j como Cij y Cji. Estas capacidades iniciales irán variando a medida que avanza el algoritmo, denominaremos capacidades residuales a las capacidades restantes del arco una vez pasa algún flujo por él, las representaremos como cij y cji.

Para un nodo j que recibe el flujo del nodo i, definimos una clasificación [aj,i] donde aj es el flujo del nodo i al nodo j.

A continuación se muestra el pseudocódigo del Algoritmo:

```
        Ford-Fulkerson (G,s,t)

        para cada arco (u,v) de E

        f(u,v) = 0
f(v,u) = 0

        mientras exista un camino p desde s a t en la red
residual Gf

        cf (p) = min {cf (u,v) : (u,v) está sobre p}

        para cada arco (u,v) en p

        f(u,v) = f(u,v) + cf (p)
f(u,v) = -f(u,v)
```

Los pasos del algoritmo se definen como sigue:

Paso 1: Inicializamos las capacidades residuales a las capacidades iniciales, hacemos (cij,cji)=(Cij,Cji) para todo arco de la red. Suponiendo el nodo 1 como el nodo origen, hacemos a1=∞ y clasificamos el nodo origen con [∞,-]. Tomamos i=1 y vamos al paso 2.

Paso 2: Determinamos Si como un conjunto que contendrá los nodos a los que podemos acceder directamente desde i por medio de un arco con capacidad positiva, y que no formen parte del camino en curso. Si Si contiene algún nodo vamos al paso 3, en el caso de que el conjunto sea vacío saltamos al paso 4.

Paso 3: Obtenemos k Є Si como el nodo destino del arco de mayor capacidad que salga de i hacia un nodo perteneciente a Si. Es decir, cik = max{cij} con j Є Si. Hacemos ak = cik y clasificamos el nodo k con [ak,i]. Si k es igual al nodo destino o sumidero, entonces hemos encontrado una ruta de penetración, vamos al paso 5. En caso contrario continuamos con el camino, hacemos i = k y volvemos al paso 2.

Paso 4 (retroceso): Si i=1, estamos en el nodo origen, y como Si es vacío, entonces no podemos acceder a ningún nodo, ni encontrar algún nuevo camino, hemos terminado, vamos al paso 6. En caso contrario, i≠1, le damos al valor i el del nodo que se ha clasificado inmediatamente antes, eliminamos i del conjunto Si actual y volvemos al paso 2.

Paso 5: Llegados a este paso tenemos un nuevo camino: Np={1,k1,k2,...,n}, esta será la p-ésima ruta de penetración desde el nodo origen al nodo destino. El flujo máximo a lo largo de esta ruta será la capacidad

mínima de las capacidades residuales de los arcos que forman el camino, es decir: fp=min{a1,ak1,ak2,...,an}. La capacidad residual de cada arco a lo largo de la ruta de penetración se disminuye por fp en dirección del flujo y se incrementa por fp en dirección inversa, es decir, para los nodos i y j en la ruta, el flujo residual se cambia de la (cij,cji) actual a (cij-fp,cji+fp) si el flujo es de i a j, o (cij+fp,cji-fp) si el flujo es de j a i . Inicializamos i=1 y volvemos al paso 2 para intentar una nueva ruta de penetración.

Paso 6 (solución): Una vez aquí, hemos determinado m rutas de penetración. El flujo máximo en la red será la suma de los flujos máximos en cada ruta obtenida, es decir: F=f1+f2+...+fm. Teniendo en cuenta que las capacidades residuales inicial y final del arco (i, j) las dan (Cij,Cji) y (cij,cji) respectivamente, el flujo máximo para cada arco se calcula como sigue: sea (α, β)=(Cij-cij, Cji-cji), si α>0, el flujo óptimo de i a j es α, de lo contrario, si β>0, el flujo óptimo de j a i es β. Es imposible lograr que tanto α como β sean positivas.

2.3.1.2. Ejemplo:

Determinar el flujo máximo en la red siguiente:

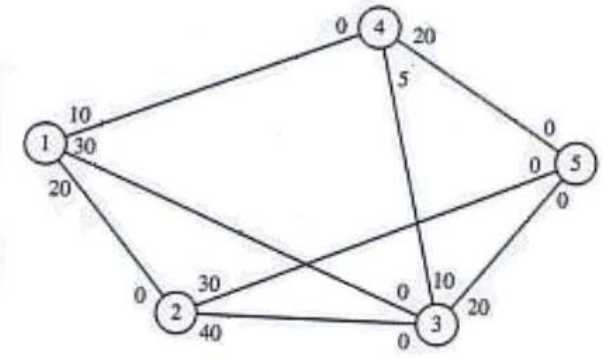


Ilustración 31 - Enunciado Ejemplo Ford-Furkelson

2.3.1.3. Solución:

- Iteración 1:

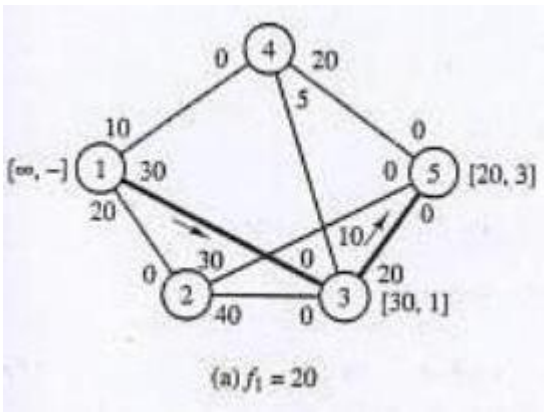


Ilustración 32 - Iteración I Ejemplo FORD-FURKELSON

Determinamos las residuales iniciales (cij,cji) iguales a las capacidades iniciales (Cij,Cji).

- Paso 1: Hacemos ai=∞, y clasificamos el nodo 1 con [a1,-]. Tomamos i=1.
- Paso 2: S1={2,3,4} (no vacío).
- Paso 3: k=3 ya que c13=max{c12,c13,c14}={20,30,10}=30. Hacemos a3=c13=30 y clasificamos el nodo 3 con [30,1]. Tomamos i=3 y repetimos el paso 2.
- Paso 2: S3={4,5}
- Paso 3: k=5 y a5=c35=max{10,20}=20. Clasificamos el nodo 5 con [20,3]. Logramos la penetración, vamos al paso 5.
- Paso 5: La ruta de la penetración se determina de las clasificaciones empezando en el nodo 5 y terminando en el nodo 1, es decir, 5→[20,3]→3→[30,1]→1.

Entonces la ruta es N1={1,3,5} y f1=min{a1,a3,a5}={∞,30,20}=20. Las capacidades residuales a lo largo de esta ruta son:

(c13,c31)=(30-20, 0+20)=(10,20)

(c35,c53)=(20-20, 0+20)=(0,20)

- Iteración 2:

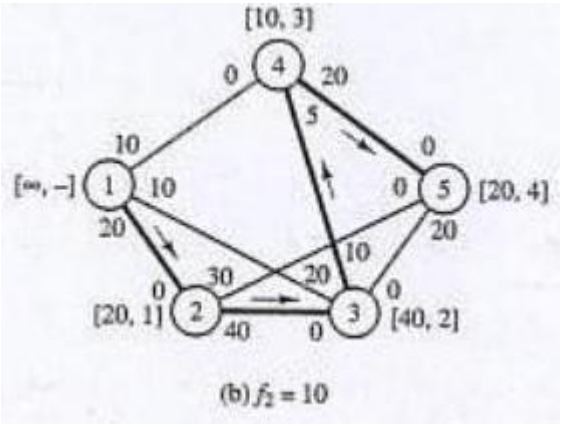


Ilustración 33 - Iteración 2 Ejemplo FORD-FURKELSON

- Paso 1: Hacemos ai=∞, y clasificamos el nodo 1 con [a1,-]. Tomamos i=1.
- Paso 2: S1={2,3,4}.
- Paso 3: k=2 y a2=c12=max{20,10,10}=20. Clasificamos el nodo 2 con [20,1]. Tomamos i=2 y repetimos el paso 2.
- Paso 2: S2={3,5}
- Paso 3: k=3 y a3=c23=max{30,40}=40. Clasificamos el nodo 3 con [40,2]. Tomamos i=3 y repetimos el paso 2.

- Paso 2: S3={4} (c35=0, el nodo 1 ya ha sido clasificado y el nodo 2 cumple ambas condiciones, por tanto los nodos 1, 2 y 5 no pueden ser incluidos en S3).
- Paso 3: k=4 y a4=c34=10. Clasificamos el nodo 4 con [10,3]. Tomamos i=4 y repetimos el paso 2.
- Paso 2: S4={5}
- Paso 3: k=5 y a5=c45=20. Clasificamos el nodo 5 con [20,4]. Logramos la penetración, vamos al paso 5.
- Paso 5: La ruta de la penetración es: 5→[20,4]→4→[10,3]→3→[40,2]→2→[20,1]→1.

Entonces la ruta es N2={1,2,3,4,5} y f2=min{∞,20,40,10,20}=10. Las capacidades residuales a lo largo de esta ruta son:

(c12,c21)=(20-10, 0+10)=(10,10)

(c23,c32)=(40-10, 0+10)=(30,10)

(c34,c43)=(10-10, 5+10)=(0,15)

(c45,c54)=(20-10, 0+10)=(10,10)

- Iteración 3:

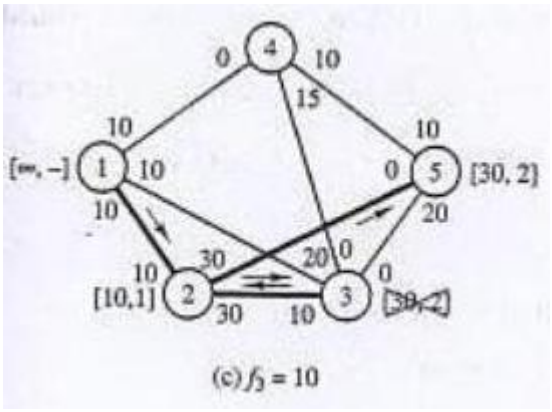


Ilustración 34 - Iteración 3 Ejemplo FORD-FURKELSON

- Paso 1: Hacemos ai=∞, y clasificamos el nodo 1 con [a1,-]. Tomamos i=1.
- Paso 2: S1={2,3,4}.
- Paso 3: k=2 y a2=c12=max{10,10,10}=10, rompemos el empate arbitrariamente. Clasificamos el nodo 2 con [10,1]. Tomamos i=2 y repetimos el paso 2.
- Paso 2: S2={3,5}
- Paso 3: k=3 y a3=c23=max{30,30}=30. Clasificamos el nodo 3 con [30,2]. Tomamos i=3 y repetimos el paso 2.
- Paso 2: S3 vacío ya que c34=c35=0. Vamos al paso 4 para retroceder.

- Paso 4: La clasificación [30,2] nos dice que el nodo inmediatamente precedente es el 2. Eliminamos el nodo 3 de una consideración posterior en esta iteración. Tomamos i=2 y repetimos el paso 2.
- Paso 2: S2={5}
- Paso 3: k=5 y a5=c25=30. Clasificamos el nodo 5 con [30,2]. Logramos la penetración, vamos al paso 5.
- Paso 5: La ruta de la penetración es: 5→[30,2]→2→[10,1]→1.

Entonces la ruta es N2={1,2,5} y f3=min{∞,10,30}=10. Las capacidades residuales a lo largo de esta ruta son:

(c12,c21)=(10-10, 10+10)=(0,20)

(c25,c52)=(30-10, 0+10)=(20,10)

- Iteración 4:

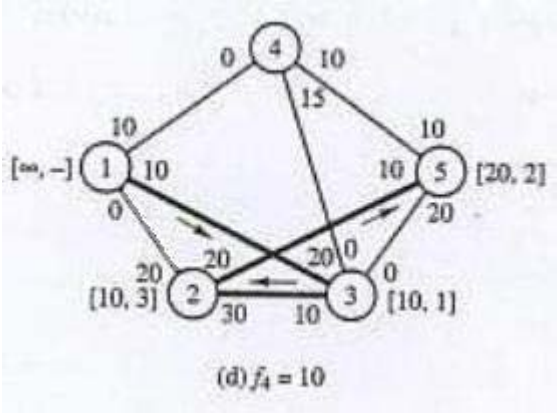


Ilustración 35 - Iteración 4 Ejemplo FORD-FURKELSON

- Paso 1: Hacemos ai=∞, y clasificamos el nodo 1 con [a1,-]. Tomamos i=1.
- Paso 2: S1={3,4}.
- Paso 3: k=3 y a3=c13=max{10,10}=10. Clasificamos el nodo 3 con [10,1]. Tomamos i=3 y repetimos el paso 2.
- Paso 2: S3={2}
- Paso 3: k=2 y a2=c32=10. Clasificamos el nodo 2 con [10,3]. Tomamos i=2 y repetimos el paso 2.
- Paso 2: S2={5}
- Paso 3: k=5 y a5=c25=20. Clasificamos el nodo 5 con [20,2]. Logramos la penetración, vamos al paso 5.
- Paso 5: La ruta de la penetración es: 5→[20,2]→2→[10,3]→3→[10,1]→1.

Entonces la ruta es N4={1,3,2,5} y f4=min{∞,10,10,20}=10. Las capacidades residuales a lo largo de esta ruta son:

(c13,c31)=(10-10, 20+10)=(0,30)

(c32,c23)=(10-10, 30+10)=(0,40)

(c25,c52)=(20-10, 10+10)=(10,20)

- Iteración 5:

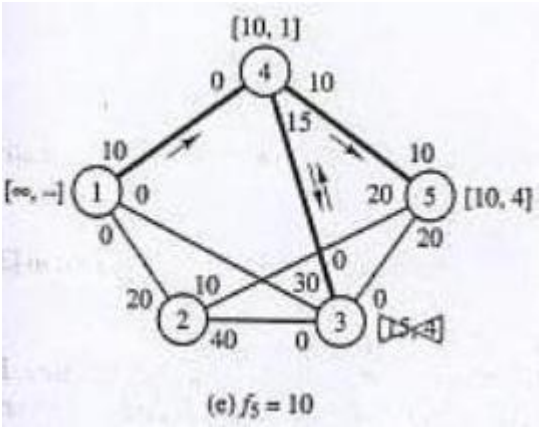


Ilustración 36 - Iteración 5 Ejemplo FORD-FURKELSON

- Paso 1: Hacemos ai=∞, y clasificamos el nodo 1 con [a1,-]. Tomamos i=1.
- Paso 2: S1={4}.
- Paso 3: k=4 y a4=c14=10. Clasificamos el nodo 4 con [10,1]. Tomamos i=4 y repetimos el paso 2.
- Paso 2: S4={3,5}
- Paso 3: k=3 y a3=c23=max{15,10}=15. Clasificamos el nodo 3 con [15,4]. Tomamos i=3 y repetimos el paso 2.
- Paso 2: S3 vacío ya que c32=c34=c35=0. Vamos al paso 4 para retroceder.
- Paso 4: La clasificación [15,4] nos dice que el nodo inmediatamente precedente es el 4. Eliminamos el nodo 3 de una consideración posterior en esta iteración. Tomamos i=4 y repetimos el paso 2.
- Paso 2: S4={5}
- Paso 3: k=5 y a5=c45=10. Clasificamos el nodo 5 con [10,4]. Logramos la penetración, vamos al paso 5.
- Paso 5: La ruta de la penetración es: 5→[10,4]→4→[10,1]→1.

Entonces la ruta es N2={1,4,5} y f3=min{∞,10,10}=10. Las capacidades residuales a lo largo de esta ruta son:

(c14,c41)=(10-10, 0+10)=(0,10)

(c45,c54)=(10-10, 10+10)=(0,20)

- Iteración 6:

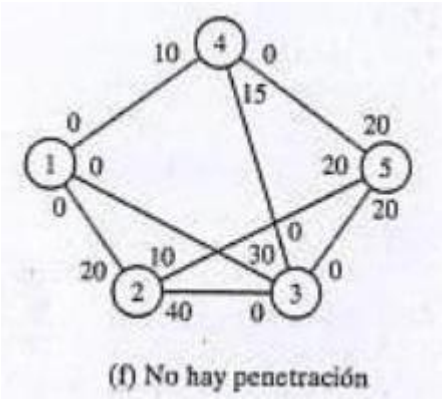


Ilustración 37 - Iteración 6 Ejemplo FORD-FURKELSON

No son posibles más penetraciones, debido a que todos los arcos fuera del nodo 1 tienen residuales cero. Vayamos al paso 6 para determinar la solución.

- Paso 6: El flujo máximo en la red es F=f1+f2+...+f5=60 unidades. El flujo en los diferentes arcos se calcula restando las últimas residuales obtenidas en la última iteración de las capacidades iniciales:

Arco	(Cij, Cji) - (cij, cji)en it. 6	Cantidad de flujo	Dirección
(1,2)	(20, 0) - (0, 20) = (20, -20)	20	1→2
(1,3)	(30, 0) - (0, 30) = (30, -30)	30	1→3
(1,4)	(10, 0) - (0, 10) = (10, -10)	10	1→4
(2,3)	(40, 0) - (40, 0) = (0, 0)	0	-
(2,5)	(30, 0) - (10, 20) = (20, -20)	20	2→5
(3,4)	(10, 5) - (0, 15) = (10, -10)	10	3→4
(3,5)	(20, 0) - (0, 20) = (20, -20)	20	3→5
(4,5)	(20, 0) - (0, 20) = (20, -20)	20	4→5

Resolución del ejemplo anterior con el programa Grafos

ÁRBOL DE FLUJO MÁXIMO - ALGORITMO DE FORD FULKERSON

Tiempo de proceso = 0 segundos

Flujos calculados desde el nodo origen (1) hasta el nodo destino (5)

Flujo máximo = 60

Matriz de Arcos con flujo máximo:

N1\N2	1	2	3	4	5
1	0	20	30	10	0
2	-20	0	0	0	20
3	-30	0	0	10	20
4	-10	0	-10	0	20
5	0	-20	-20	-20	0

Matriz de Capacidades Residuales:

N1\N2	1	2	3	4	5
1	0	0	0	0	0
2	20	0	40	0	10
3	30	0	0	0	0
4	10	0	10	0	0
5	0	20	20	20	0

En el report podemos ver que el programa también nos adjunta el gráfico de la solución. Al lado de cada arco de flujo tenemos dos valores, el primero es la capacidad que circulará por el grafo y el segundo valor es la capacidad máxima que podríamos hacer circular por él. Esta información también la podemos ver en la matriz de flujo máximo. La matriz de capacidades residuales nos indica las capacidades restantes del arco una vez pasa algún flujo por él.

El flujo máximo que podrá circular por el grafos tiene un valor máximo de 60 unidades que se corresponde al mismo valor encontrado con el ejemplo solucionada teóricamente.

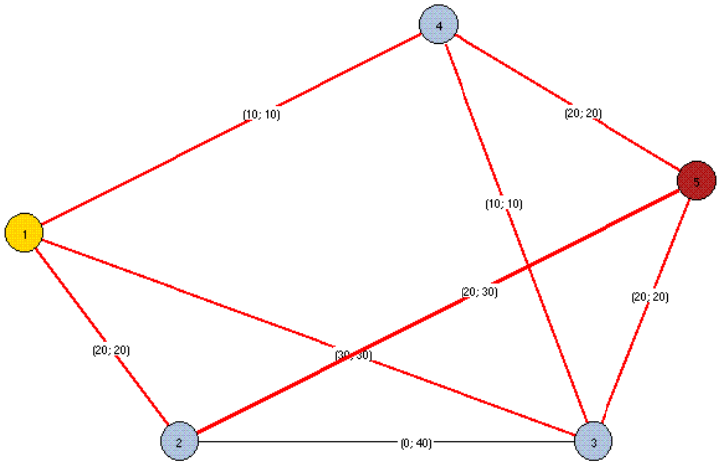


Ilustración 38 - Report Ejemplo Ford-Furkelson

2.3.2. Problema de transbordo:

Al ser los dos métodos iguales, el de transbordo y el de asignación, explicamos un solo método.

En este apartado se comentará la aplicación de la teoría de grafos para la resolución y optimización del flujo en una red. Concretamente se hablará de los Problemas de Suministro y Demanda.

Dado un grafo G con vértices o nodos V conectados entre sí y clasificados en dos conjuntos X e Y. Los nodos x se considerarán como elementos de suministro (por ejemplo plantas productivas o proveedores), mientras que los nodos y se entenderán como elementos de demanda (clientes cuya demanda hay que satisfacer). El objetivo será encontrar el flujo (transporte de mercancías a través de la red) con origen en los nodos x y destino en los y, que satisfaga una serie de restricciones que dependerán del problema particular a resolver.

En la literatura, existen multitud de heurísticas y algoritmos optimizadores para resolver algunos de estos casos particulares (por ejemplo una versión especial del Algoritmo Simplex para Programación Lineal llamado Network Simplex Algorithm). Asimismo, también existen generalizaciones del problema ('multiterminal'), o incluso variantes con perdida de flujo en los arcos (por ejemplo la evaporación de agua en canales de riego).

En cualquier caso, este tipo de problemas es uno de los más interesantes de la teoría de grafos y de la investigación operativa. Su aplicación es visible y de gran importancia para la resolución de problemas reales en la Dirección de Operaciones y Logística.

2.3.2.1. Problema del Transbordo (coste mínimo)

Se puede considerar un grafo como una red de flujo. Donde los nodos proveedor producen o introducen en la red cierta cantidad de algún tipo de material, y los nodos cliente lo consumen. Cada arco, por tanto, puede considerarse como un conducto que tiene cierta capacidad de flujo (mínima y/o máxima) y un coste de transporte por unidad de material.

Dado un grafo dirigido G=(V,E) con capacidades c (mínima y/o máxima) no negativas y también con coste de transporte no negativo (expresado por unidad de flujo) para los arcos E.

Dado un conjunto de nodos X proveedores (con una capacidad b<0 de suministro), y también un conjunto Y de nodos clientes (con una demanda b>0). En este caso particular, además se consideran nodos de transbordo (con valor de b=0), esto es nodos, que ni aportan ni consumen mercancías y donde éstas simplemente cambian de arco o hacen un transbordo.

Además, se exige la condición de continuidad en los nodos. Esto es, la suma de flujos entrantes a un nodo menos la suma de los salientes, debe ser mayor o igual a la capacidad b del nodo.

Por tanto, el problema de flujo máximo se enuncia como: ¿cuál es el flujo de transporte de cada arco con el cual se puede transportar el material desde los nodos proveedores a los nodos cliente, sin violar las restricciones de capacidad y continuidad y todo ello minimizando el coste total de transporte?.

$$\min \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij}$$

sujeto a :

$$\sum_{k=1}^{m} x_{ki} - \sum_{j=1}^{m} x_{ij} = b_i, j = 1\cdots m \begin{cases} b_i > 0, \text{demanda} \\ b_i < 0, \text{oferta} \\ b_i = 0, \text{transbordo} \end{cases}$$

$$x_{ij} \geq 0, x_{ij} \in Z$$

Ilustración 39 - Modelo PL de Transbordo

En el caso de Grafos, este problema se aborda a través del modelado y resolución mediante Programación Lineal. Igualmente, se pueden resolver problemas de: transporte de mercancías (logística de aprovisionamiento y distribución), flujo de gases y líquidos por tuberías, componentes o piezas en líneas de montaje, corriente en redes eléctricas, paquetes de información en redes de comunicaciones, tráfico ferroviario, sistema de regadíos, etc.

El problema de transbordo, se puede transformar en un problema clásico de flujo mínimo coste añadiendo un único nodo proveedor s y un único nodo de demanda t (ver Hitchcock-Koopmans, método Out-of-Kilter). Para que dicho problema sea factible, la suma del suministro total debe ser mayor o igual que el total de demanda (equilibrado).

2.3.2.2. Problema del Transporte (coste mínimo)

El problema del transporte es un problema de transbordo pero con un grafo dividido G=(S,T,E), con todos los nodos suministradores en S y los de demanda en T. Además, todos los arcos van dirigidos de S a T y no existen nodos de transbordo.

El Problema de Asignación es un caso especial del problema de transporte, en el cual hay el mismo número de nodos suministradores que de demanda y los valores de nodos b= ± 1.

2.3.2.3. Ejemplo

NTN es un operador logístico intermodal suizo localizado en Lausanne. Cuando un cliente tiene que transportar productos de un sitio a otro, NTN le suministra uno o más contenedores en los cuales el cliente cargará sus productos. Una vez que el cargamento ha llegado a su destino y los productos han sido descargados, el contenedor vacío debe ser transportado a un punto de recogida de un nuevo cliente.

Como consecuencia de esto, la gestión de NTN necesita reubicar periódicamente los contenedores vacíos (en la práctica una vez a la semana). El transporte de contenedores vacíos es muy caro (su coste es aproximadamente del 35% del total de los costes operativos).

El caso aquí propuesto muestra un supuesto donde varios contenedores ISO 20 vacíos deben reubicarse entre las terminales de Amsterdam, Berlín, Munich, Paris, Milán, Barcelona y Madrid.

Para modelar este problema se requieren los datos de disponibilidad y demanda de contenedores en cada uno de los terminales. Tenga en cuenta que los valores negativos deben entenderse como oferta, mientras que los positivos corresponden a la demanda. Se debe construir un grafo que relacione las diferentes terminales, cada terminal puede estar comunicada en ambos sentidos. En el grafo resultante se expresa el coste de transporte de los contenedores en euros/contenedor.

N1\N2	Coste	Amsterdam	Berlin	Munich	Paris	Milan	Barcelona	Madrid
Amsterdam	10		30	40	20			
Berlin	-20	30		30				
Munich	-50	40	30		55	30		
Paris	-20	20		55		30	50	70
Milan	50			30	30		30	
Barcelona	20				50	30		25
Madrid	10				70		25	

Ilustración 40 - Tabla Ejemplo Transbordo

2.3.2.4. Solución

Se trata por tanto de averiguar el flujo de contenedores óptimo entre las diferentes terminales, tal que la oferta cubra toda la demanda, y de manera que el coste total de transporte sea mínimo. En este problema, no se tienen en cuenta las restricciones de flujo mínimo o de flujo máximo. Además como se puede observar los datos de oferta y demanda, se trata de un problema equilibrado (demanda total = oferta total) por lo que no habrá oferta o demanda residual.

Para resolverlo, utilizaremos la opción de análisis de problema de transbordo, que mediante un modelo de programación lineal entera mixta (ver modelo .lp para este caso concreto) encontrará la solución óptima al problema.

La siguiente figura muestra el flujo de contenedores desde unas terminales a otras.

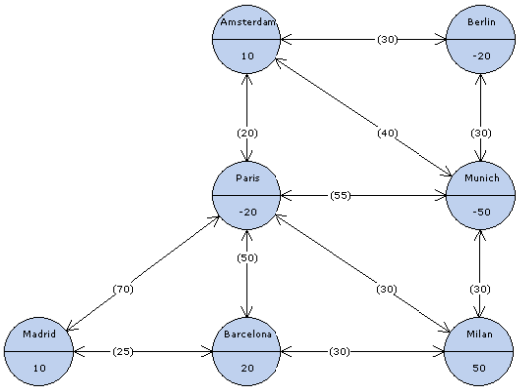


Ilustración 41 - Grafo Ejemplo Transbordo

Report del programa:

PROBLEMA DEL TRANSBORDO

Tiempo de modelado = 0 segundos

Tiempo de proceso = 0 segundos

SOLUCION OPTIMA ENCONTRADA

lp_solve -> 0

Valor de la función objetivo = 3900

Valor actual de las variables:

x_1_0:: Berlin --> Amsterdam = 10

x_2_0:: Munich --> Amsterdam = 0

x_3_0:: Paris --> Amsterdam = 0

x_0_1:: Amsterdam --> Berlin = 0

x_2_1:: Munich --> Berlin = 0

x_0_2:: Amsterdam --> Munich = 0

x_1_2:: Berlin --> Munich = 10

x_3_2:: Paris --> Munich = 0

x_4_2:: Milan --> Munich = 0

x_0_3:: Amsterdam --> Paris = 0

x_2_3:: Munich --> Paris = 0

x_4_3:: Milan --> Paris = 0

x_6_3:: Madrid --> Paris = 0

x_2_4:: Munich --> Milan = 60

x_3_4:: Paris --> Milan = 0

x_5_4:: Barcelona --> Milan = 0

x_4_5:: Milan --> Barcelona = 10

x_6_5:: Madrid --> Barcelona = 0

x_3_6:: Paris --> Madrid = 10

x_5_6:: Barcelona --> Madrid = 0

x_3_5:: Paris --> Barcelona = 10

x_5_3:: Barcelona --> Paris = 0

Valor actual de las restricciones:

r1	10
r2	-20
r3	-50
r4	-20
r5	50
r6	20
r7	10
r8	10
r9	0
r10	0
r11	0
r12	0
r13	0
r14	10
r15	0
r16	0
r17	0
r18	0
r19	0
r20	0
r21	60
r22	0
r23	0
r24	10
r25	0
r26	10
r27	0

r28 10

r29 0

Sensibilidad de los coeficientes de la función objetivo:

	Desde	Hasta	Coste reducido
x_1_0:: Berlin --> Amsterdam =	20	60	-1e+030
x_2_0:: Munich --> Amsterdam =	0	1e+030	0
x_3_0:: Paris --> Amsterdam =	-10	1e+030	0
x_0_1:: Amsterdam --> Berlin =	-30	1e+030	0
x_2_1:: Munich --> Berlin =	-30	1e+030	0
x_0_2:: Amsterdam --> Munich =	0	1e+030	0
x_1_2:: Berlin --> Munich =	0	40	-1e+030
x_3_2:: Paris --> Munich =	-10	1e+030	0
x_4_2:: Milan --> Munich =	-30	1e+030	0
x_0_3:: Amsterdam --> Paris =	10	1e+030	0
x_2_3:: Munich --> Paris =	10	1e+030	0
x_4_3:: Milan --> Paris =	-20	1e+030	0
x_6_3:: Madrid --> Paris =	-70	1e+030	0
x_2_4:: Munich --> Milan =	0	40	-1e+030
x_3_4:: Paris --> Milan =	20	1e+030	0
x_5_4:: Barcelona --> Milan =	-30	1e+030	0
x_4_5:: Milan --> Barcelona =	20	40	-1e+030
x_6_5:: Madrid --> Barcelona =	-20	1e+030	0
x_3_6:: Paris --> Madrid =	25	75	-1e+030
x_5_6:: Barcelona --> Madrid =	20	1e+030	0
x_3_5:: Paris --> Barcelona =	45	60	-1e+030
x_5_3:: Barcelona --> Paris =	-50	1e+030	0

Sensibilidad RHS de las restricciones:

	Precio sombra	Desde	Hasta
r1	30	0	10
r2	0	-1e+030	1e+030
r3	30	-60	-50
r4	40	-30	-20
r5	60	40	50
r6	90	10	20

r7 110 0 10

r8 0 -1e+030 1e+030

r9 0 -1e+030 1e+030

r10 0 -1e+030 1e+030

r11 0 -1e+030 1e+030

r12 0 -1e+030 1e+030

r13 0 -1e+030 1e+030

r14 0 -1e+030 1e+030

r15 0 -1e+030 1e+030

r16 0 -1e+030 1e+030

r17 0 -1e+030 1e+030

r18 0 -1e+030 1e+030

r19 0 -1e+030 1e+030

r20 0 -1e+030 1e+030

r21 0 -1e+030 1e+030

r22 0 -1e+030 1e+030

r23 0 -1e+030 1e+030

r24 0 -1e+030 1e+030

r25 0 -1e+030 1e+030

r26 0 -1e+030 1e+030

r27 0 -1e+030 1e+030

r28 0 -1e+030 1e+030

r29 0 -1e+030 1e+030

x_1_0:: Berlin --> Amsterdam = 0 -1e+030 1e+030

x_2_0:: Munich --> Amsterdam = 40 0 10

x_3_0:: Paris --> Amsterdam = 30 0 10

x_0_1:: Amsterdam --> Berlin = 60 0 1e+030

x_2_1:: Munich --> Berlin = 60 0 1e+030

x_0_2:: Amsterdam --> Munich = 40 0 10

x_1_2:: Berlin --> Munich = 0 -1e+030 1e+030

x_3_2:: Paris --> Munich = 65 0 10

x_4_2:: Milan --> Munich = 60 0 1e+030

x_0_3:: Amsterdam --> Paris = 10 0 10

x_2_3:: Munich --> Paris = 45 0 10

x_4_3:: Milan --> Paris = 50 0 10

x_6_3:: Madrid --> Paris = 140 0 1e+030

x_2_4:: Munich --> Milan = 0 -1e+030 1e+030

x_3_4:: Paris --> Milan = 10 0 10

x_5_4:: Barcelona --> Milan = 60 0 1e+030

x_4_5:: Milan --> Barcelona = 0 -1e+030 1e+030

x_6_5:: Madrid --> Barcelona = 45 0 10

x_3_6:: Paris --> Madrid = 0 -1e+030 1e+030

x_5_6:: Barcelona --> Madrid = 5 0 10

x_3_5:: Paris --> Barcelona = 0 -1e+030 1e+030

x_5_3:: Barcelona --> Paris = 100 0 1e+0

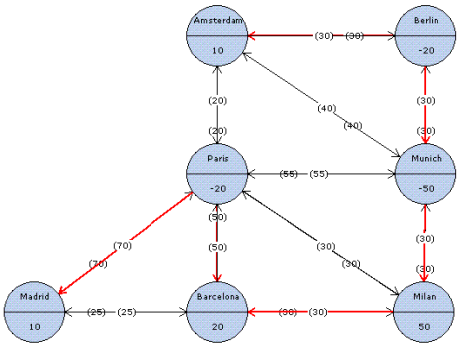


Ilustración 42 - Report Ejemplo Transbordo

Analizando el report, podemos ver que el flujo óptimo de contenedores es:

Berlín-->Amsterdam=10

Berlín-->Munich=10

Munich-->Milán=60

Milán-->Barcelona=10

Paris-->Madrid=10

Paris --> Barcelona = 10

El coste total de esta operación de transbordo es de 3.900 € (coste mínimo).

Si bien, el caso aquí expuesto es una situación concreta del operador logístico a una determinada fecha, la gestión del operador consiste en resolver este problema de manera dinámica periódicamente a lo largo del tiempo. Será necesario además enlazar este modelo de transbordo con otros modelos de previsión de la demanda.

2.4. Análisis de rutas

Dentro de la denominación de Problemas de Rutas o Recorridos realmente se engloba todo un amplio conjunto de variantes y personalizaciones de problemas. Desde aquellos más sencillos (como el que se comentará en este apartado) hasta algunos mucho más complejos que incluso hoy en día son materia de investigación. Todos ellos sin embargo, además del reto computacional que representan, tienen en común su gran importancia en investigación operativa por su aplicación práctica en la realidad.).

Podemos hacer una primera clasificación de los problemas de rutas atendiendo al lugar donde se produce la demanda, distinguiendo entre los Problemas por Vértices y los Problemas por Arcos.

Nosotros trataremos los problemas por vértices, dónde el elemento esencial a visitar por los vehículos son los clientes situados en los nodos o vértices de un grafo, tal es el caso del problema de obtener un ciclo que pase por todos los vértices de un grafo de forma que el recorrido/distancia total sea el mínimo posible. Este es el famoso TSP (Traveling Salesman Problem o Problema del Viajante), en el que un único vehículo debe realizar todo el trabajo.

El VRP (Vehicle Routing Problem) es una generalización natural del TSP en donde la demanda total requiere más de un vehículo. Este problema es todavía más difícil que el anterior pues se hace necesario particionar el conjunto de clientes para que sean atendidos separadamente por cada vehículo y después determinar el orden de servicio de cada vehículo.

2.4.1. El viajante de comercio:

2.4.1.1. Definición del problema:

Este problema se puede representar como un grafo completo dirigido G = (N,A), donde N es un conjunto de n nodos y A es el conjunto de aristas del grafo. Al igual que en el caso anterior, los nodos representan las ciudades y las aristas las distancias entre ellas. Por lo tanto, se supone que las aristas nunca toman valores negativos y como en este caso, se trata de un grafo completo, siempre existe una arista entre cualquier par de nodos (es decir, desde cualquiera de las ciudades se puede llegar directamente hasta todas las demás). Como se supone de nuevo que se tiene un grafo dirigido, hay que tener en cuenta que la distancia para ir desde i hasta j no tiene por qué ser igual a la distancia para llegar desde j hasta i. Esto ocurre muchas veces en la realidad al desplazarse entre ciudades.

Un viajante de comercio desea salir de una de las ciudades O, para visitar todas las demás ciudades exactamente una vez y regresar al punto de partida habiendo recorrido la mínima distancia posible. Hablando en términos de grafos, se dice que el objetivo de este problema es encontrar el ciclo hamiltoniano más corto posible de un grafo dado. A modo de recordatorio, se dice que un ciclo hamiltoniano es aquel que pasa por todos los nodos del grafo exactamente una vez. Formulando el problema matemáticamente, se dice que el

objetivo del mismo consiste en encontrar el camino C formado por la secuencia de n+1 ciudades, C = {1, 2,…, n ,n+1}, donde $\forall i \in N, i \in C$, que consiga:

Minimizar $\sum_{i=1}^{n} A[C_i, C_{i+1}]$ tal que $C_1 = C_{n+1} = 0$

Sea xij una variable binaria que indica si el viajante utilizará el arco de la ciudad i a la j en su recorrido solución. Para el modelado matemático, la ciudad de comienzo es irrelevante. A continuación se muestra el modelo completo del problema:

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}$$

sujeto a:

$$\sum_{j=1}^{n} x_{ij} = 1, i = 1 \cdots n$$

$$\sum_{i=1}^{n} x_{ij} = 1, j = 1 \cdots n$$

$$x_{ij} \in \{0,1\}$$

condiciones de Tucker:

$$u_i - u_j + n \cdot x_{ij} \leq n-1, 2 \leq i \neq j \leq n$$

Ilustración 43 - Modelo del Viajante

Supongamos que las ciudades a ser visitadas están numeradas 1,2,…,n. Utilizamos las variables xij con el siguiente significado:

xij=1 si el tour va directamente de la ciudad i a la ciudad j

xij=0 si el tour no va directamente de la ciudad i a la ciudad j

El concepto directamente es muy importante porque quiere decir que la ciudad xij solo tomará valor si el viajante va desde la ciudad i a la j sin pasar por ninguna otra ciudad por el medio.

El objetivo será minimizar

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}$$

Hay dos condiciones obvias que deben cumplirse:

Exactamente una ciudad debe ser visitada inmediatamente después de la ciudad i

Exactamente una ciudad debe ser visitada inmediatamente antes de la ciudad j.

Esto es lo que quiere decir

$$\sum_{j=1}^{n} x_{ij} = 1, i = 1 \cdots n$$

$$\sum_{i=1}^{n} x_{ij} = 1, j = 1 \cdots n$$

$$x_{ij} \in \{0,1\}$$

Sin embargo, con estas restricciones y la condición binaria de las variables xij, no es suficiente para garantizar que las soluciones factibles son recorridos. Es posible por tanto, que aparezca una solución formada por subrutas (no conectadas entre sí) y que cumplan las restricciones anteriormente comentadas.

Es por ello, por lo que es necesario añadir más restricciones que eviten la formación de subrutas. Una de las posibles formas de hacer esto (ya que existen varias), es la propuesta por Tucker:

Condiciones de Tucker:

$$U_i - U_j + n \cdot x_{ij} \leq n - 1 \qquad 2 \leq i \neq j \leq n$$

La Ui son variables enteras. El significado de la variable indica su naturaleza entera pero no es necesario definirlas así. Al estar relacionadas con variables y coeficientes enteros, el algoritmo de resolución de Programación Linial Entera dará valores enteros a dichas variables.

Secuencia: dijimos que le conjunto de variables xij definen una secuencia de visitas a las ciudades del problema. Por secuencia entendemos que el valor de la variable asociada a la ciudad de partida de un tramo es menor que la variable asociada a la ciudad de llegada del mismo tramo.

Esto queda reflejado en los vínculos en los que xij = 1. Esto quiere decir que la ciudad i precede a la ciudad j.

Esto se demuestra a partir de aquí:

$$U_i - U_j + n \cdot x_{ij} \leq n - 1$$

Cuando los xij =1

$$U_i - U_j + n \cdot \leq n - 1$$

$$U_i - U_j \leq n - 1 - n$$

$$U_i - U_j \leq 1$$

$$U_i \leq U_j - 1$$

$$U_j \leq U_i + 1$$

Es decir, el valor de la variable de llegada debe ser mayor que la variable de partida en, por lo menos, una unidad. A continuación se demuestra que difieren exactamente una unidad. A esto se le llama correlación unitaria (el significado de las variables Ui es el número de secuencia en el cual la ciudad i es visitada).

Para que esto ocurra, debe cumplirse que Uj = Ui + 1

En el conjunto de variables Ui habrá una Ul que indique que es la primera ciudad visitada y una Uk que indique cual es la última ciudad visitada antes de retornar a la ciudad de partida (ciudad 0)

El planteamiento no involucra la ciudad cero y desde la última ciudad solo se puede llegar a la ciudad 0 (la última ciudad y la primera después de la cero no se relacionan).

Por lo tanto

xkl = xlk = 0

Si planteamos

$$U_k - U_l + n \cdot x_{kl} \leq n - 1$$

Como Xkl= 0

$$U_k - U_l \leq n - 1$$

La diferencia entre dos ciudades en secuencia es unitaria, ya que, de ser de otra manera, habrá dos ciudades con el mismo número de secuencia, y eso, como hemos visto, es imposible.

La clave es que, al considerar un subtour, una de las restricciones reflejará la llegada a la primera ciudad partiendo de la última y la diferencia Ui – Uj será positiva y no cumplirá la condición. Entonces demostraremos que al agregar estas condiciones se impide cualquier subtour que quiera plantearse.

¿Porqué no evita el tour completo? Porque incluye la ciudad cero y, como las UI están definidas para cualquier ciudad salvo para la cero, ésa es la última ciudad a la que se puede llegar después de haber salido de ella.

El Problema del Viajante de Comercio (Traveling Salesman Problem - TSP), es un problema de complejidad NP-completo. Esto es así, porque el número de posibles soluciones crece exponencialmente con el número de nodos del grafo (ciudades), y rápidamente sobrepasa las capacidades de cálculo de los ordenadores más potentes.

2.4.1.2. Ejemplo (Utilizando la Heurística del Vecino más próximo):

Sea el siguiente grafo:

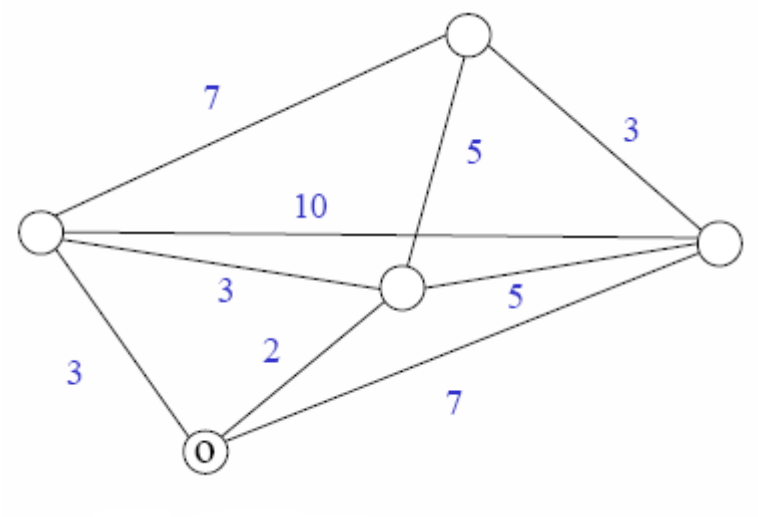


Ilustración 44 - Ejemplo Viajante 1

Donde el nodo marcado como nodo cero es el nodo origen

Buscaremos la ciudad o nodo más próximo a nuestra ciudad origen:

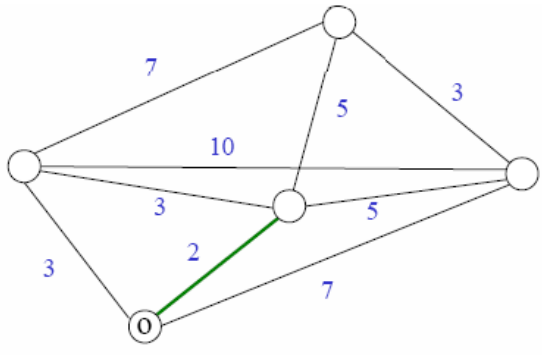


Ilustración 45 - Ejemplo Viajante 2

Marcamos el camino que seguimos y, a continuación, volvemos a buscar la ciudad vecina más próxima a la que estamos:

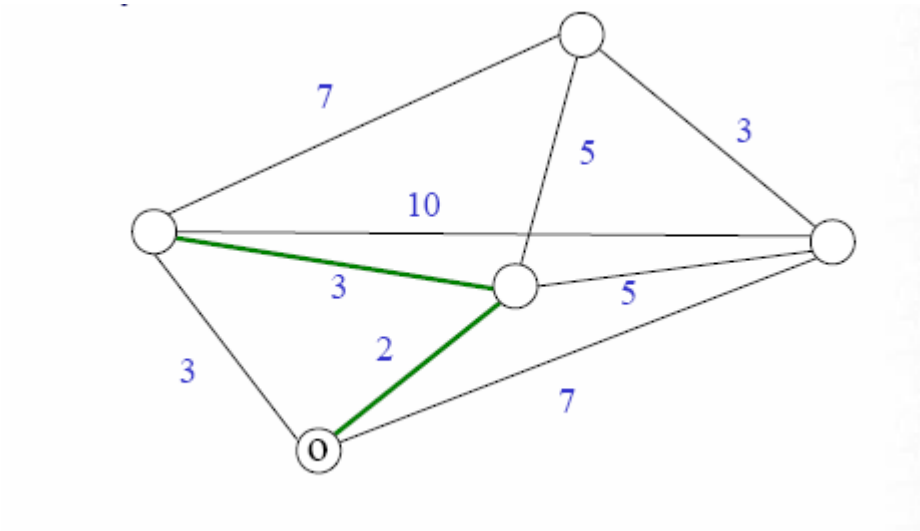


Ilustración 46 - Ejemplo Viajante 3

Se debe vigilar pues no se admite que la ciudad que escogemos haga un ciclo o una horquilla con las que ya hemos visitado.

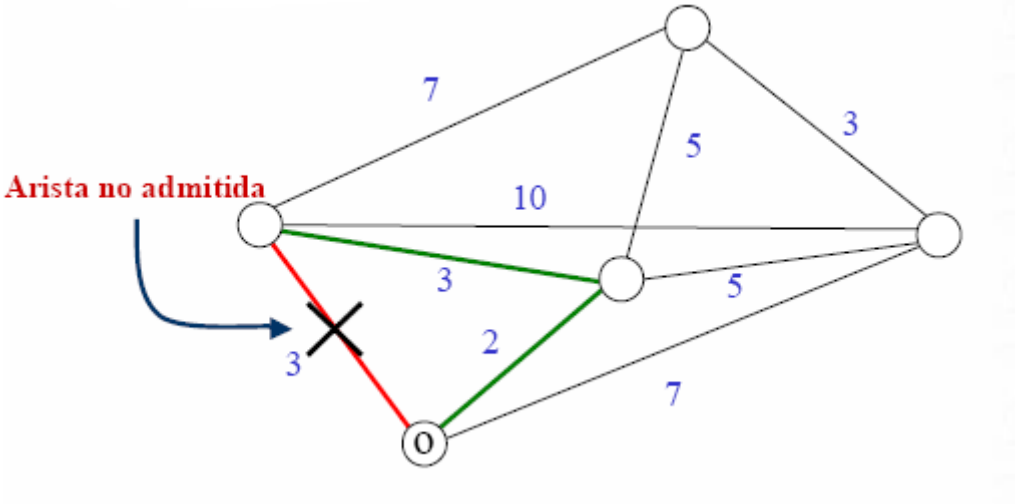


Ilustración 47 - Ejemplo Viajante 4

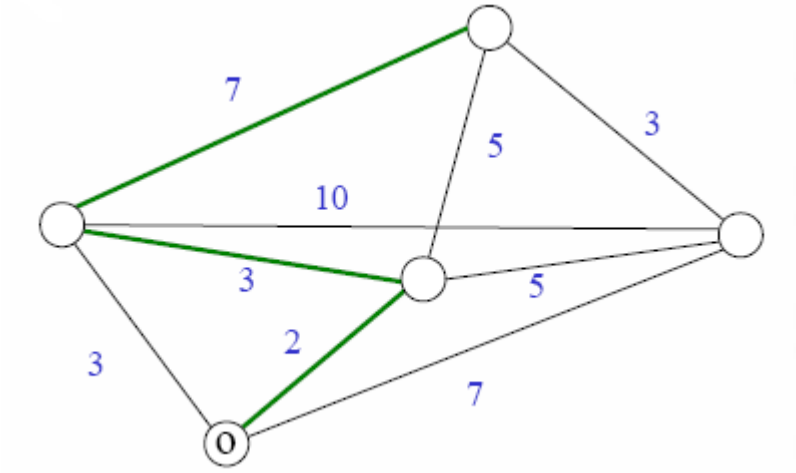


Ilustración 48 - Ejemplo Viajante 5

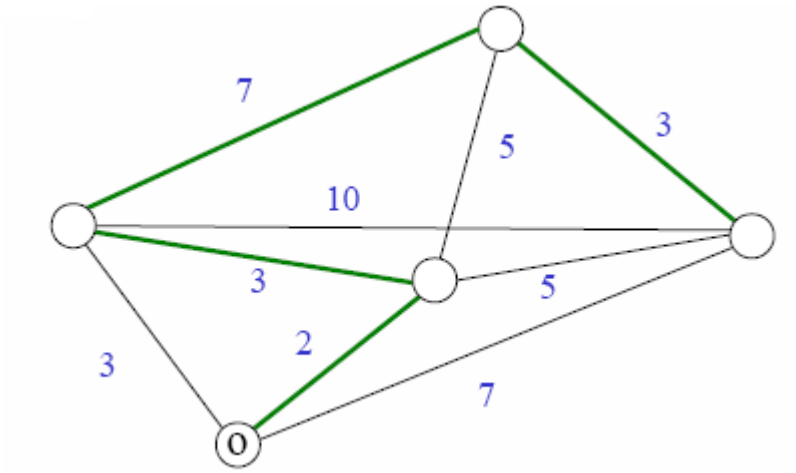


Ilustración 49 - Ejemplo Viajante 6

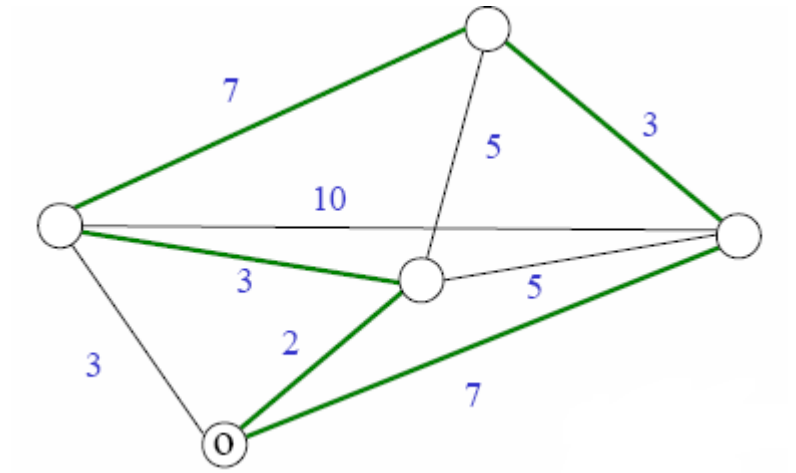


Ilustración 50 - Ejemplo Viajante 7

Finalmente, hemos llegado a una solución. Esta solución nos da un costo de 22 (2+3+7+3+7)

RESOLUCIÓN UTILIZANDO EL SOFTWARE

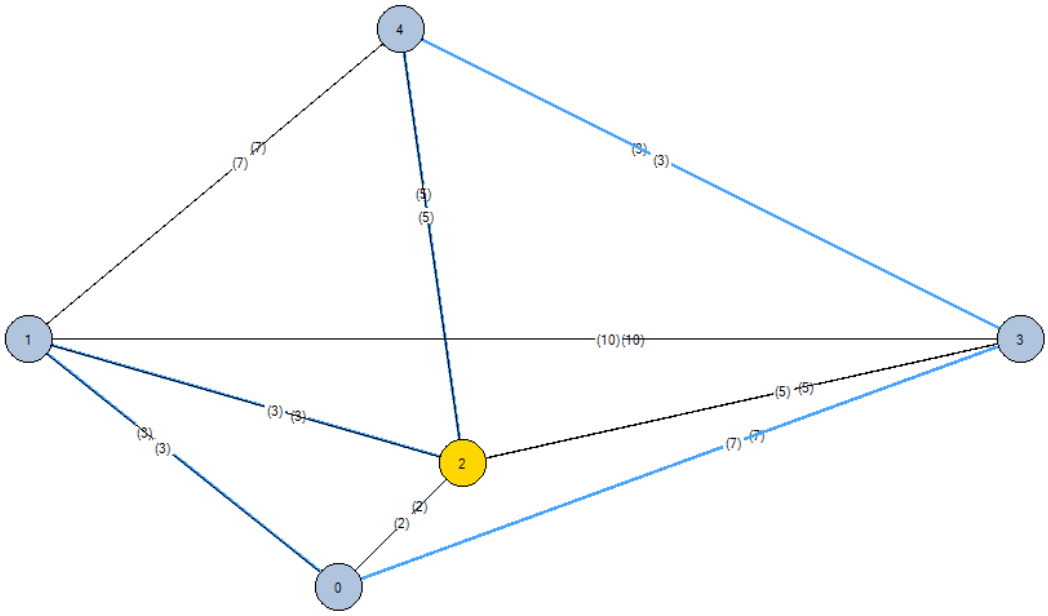


Ilustración 51 - Report Ejemplo Viajante

REPORT

PROBLEMA DEL VIAJANTE DE COMERCIO

Tiempo de modelado = 0 segundos

Tiempo de proceso = 0 segundos

SOLUCION ÓPTIMA ENCONTRADA

lp_solve -> 0

Valor de la función objetivo = 21

Valor actual de las variables:

x_1_0:: 1 --> 0 = 1

x_2_0:: 2 --> 0 = 0

x_3_0:: 3 --> 0 = 0

x_0_1:: 0 --> 1 = 0

x_2_1:: 2 --> 1 = 1

x_3_1:: 3 --> 1 = 0

x_4_1:: 4 --> 1 = 0

x_0_2:: 0 --> 2 = 0

x_1_2:: 1 --> 2 = 0

x_3_2:: 3 --> 2 = 0

x_4_2:: 4 --> 2 = 1

x_0_3:: 0 --> 3 = 1

x_1_3:: 1 --> 3 = 0

x_2_3:: 2 --> 3 = 0

x_4_3:: 4 --> 3 = 0

x_1_4:: 1 --> 4 = 0

x_2_4:: 2 --> 4 = 0

x_3_4:: 3 --> 4 = 1

u1 4

u0 5

u2 3

u3 0

u4 2

Valor actual de las restricciones:

r1 1

r2 1

r3 1

r4 1

r5 1

r6 1

r7 1

r8 1

r9 1

r10 1

r11 4

r12 1

r13 4

r14 2

r15 -2

r16 4

r17 3

r18	1
r19	-5
r20	-4
r21	-3
r22	3
r23	-2
r24	4
r25	2
r26	1
r27	0
r28	0
r29	0
r30	1
r31	0
r32	0
r33	0
r34	0
r35	0
r36	1
r37	1
r38	0
r39	0
r40	0
r41	0
r42	0
r43	1
r44	1
r45	0
r46	0
r47	0
r48	1
r49	0
r50	0
r51	0
r52	0

r53	0
r54	1
r55	1
r56	0
r57	0
r58	0
r59	0
r60	0
r61	1

Ver explicación del Modelo Lineal (pág. 43) para entender el significado de cada una de las restricciones y de las variables que intervienen en este Report.

2.4.2. PROBLEMA DE LOS m-viajantes del comercio

Se trata de m vendedores o viajantes de comercio que debe visitar n ciudades para vender u ofertar sus productos. Cada viajante de comercio podrá visitar como máximo a p clientes o ciudades. Cada par de ciudades puede estar comunicada o no, su distancia se define mediante cij. El problema es por tanto, decidir el recorrido que comenzando por una determinada ciudad { 0 } pase por todas las demás una sola vez y vuelva finalmente a la primera, de manera que se minimice la distancia total recorrida de todas las rutas.

2.4.3. Problemas de rutas de vehículos VRP (vehicle routing problema):

2.4.3.1. Introducción:

En la literatura científica, Dantzig y Ramser fueron los primeros autores en 1959, en interesarse en los problemas de rutas de vehículos VRP, cuando estudiaron la aplicación real en la distribución de gasolina para estaciones de carburante.

Este tipo de problemas pertenece a los problemas de optimización combinatoria1, dentro de los cuáles reciben la denominación de Problemas de Rutas o Recorridos. Éstos engloban todo un amplio conjunto de variantes y personalizaciones de problemas. Desde aquellos más sencillos, hasta algunos mucho más complejos que incluso hoy en día son materia de investigación. Todos ellos sin embargo, además del reto

[1] Los algoritmos de optimización combinatoria resuelven instancias de problemas que se creen ser difíciles en general, explorando el espacio de soluciones (usualmente grande) para estas instancias. Estos algoritmos combinatoria logran esto reduciendo el tamaño efectivo del espacio, y explorando el espacio de búsqueda eficientemente.

computacional que representan, tienen en común su gran importancia en investigación operativa por su aplicación práctica en la realidad.

La mayoría de los problemas VRP son de complejidad NP-completo, es decir, el número de posibles soluciones crece exponencialmente con el número de nodos del grafo (ciudades o puntos de paso) y rápidamente sobrepasa las capacidades de cálculo de los ordenadores más potentes. Es por esa misma razón, que esta clase de problemas no se resuelven de manera óptima, ya que, como hemos mencionado anteriormente, a medida que n va siendo más grande, esto puede llegar a ser imposible, así que los problemas combinatorios son solucionados con heurísticas las cuales dan una buena solución subóptima con un consumo moderado de recursos.

En ellos en general, se trata de averiguar las rutas de una flota de transporte para dar servicio a unos clientes. Su aplicación es visible y de gran importancia para la resolución de problemas reales en la Dirección de Operaciones y Logística, como por ejemplo:

- Problemas de preparación de pedidos en un almacén (picking).
- Rutas de vehículos.
- Planificación de transporte urbano.
- Planificación de recogida de residuos o de aprovisionamiento.
- Problemas de reparto o distribución.
- Sistemas de navegación GPS.
- Planificación de movimientos de robots, vehículos autoguiados (AGV)

Problema del Agente Viajero (TSP), el cual se ha tratado anteriormente, y el VRP, están relacionados estrechamente, ya que el TSP sería equivalente a tener un VRP con un único vehículo de capacidad suficiente y sin límite de tiempo para realizar la tarea.

El VRP básico consiste en establecer las rutas para una flota de vehículos que deben salir de un depósito y recorrer determinados puntos (clientes) una sola vez y regresar al depósito, de manera que se minimice la distancia total recorrida por todos los vehículos.

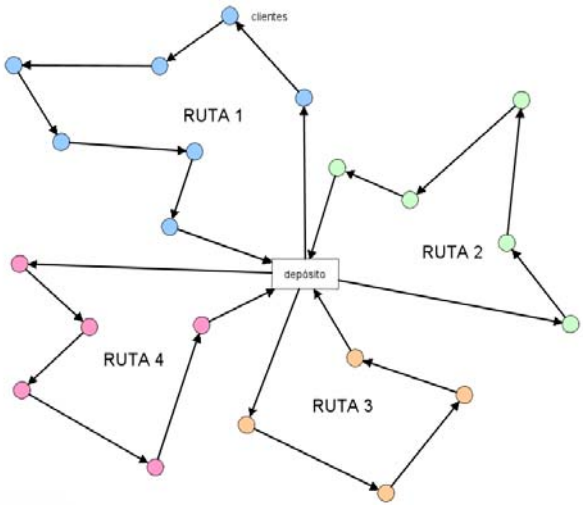


Ilustración 52 - Esquema básico VRP

2.4.4. Variaciones del VRP Clásico:

Existen distintas variaciones del VRP original, de las que destacamos las siguientes:

- Restricciones de Capacidad. Cada vehículo posee una limitación de capacidad o carga. Esto de lugar a una variación del VRP que se llama CVRP-Vehicle Routing Problem o Problema de Rutas de vehículos restringidos por Capacidad.
- Restricciones de Distancia. Es decir, cada vehículo además de tener una limitación de capacidad o carga, también está limitado por el tiempo o distancia máxima que puede emplear. Esto da lugar a una variación del VRP que se llama DCVRP-Distance Capacitated Vehicle Routing Problem o Problema de Rutas de Vehículos restringidos por Capacidad yDistancia).
- Ventanas de Tiempo. En este caso, los clientes exigen que el servicio se realice dentro de un horario o ventana de tiempo. Un cliente puede tener más de una ventana. El vehículo, no puede atender la demanda de un cliente fuera de su horario, aunque en ocasiones se permite incumplimientos de horarios pero ello conlleva a una penalización que es función del retraso. El problema básico dentro de esta categoría es el VRPTW-Vehicle Routing Problem with Time Windows. El orden de visita de los clientes dentro de una ruta deja de ser el que minimiza el coste o distancia recorrida por el vehículo, pues la existencia de ventanas de tiempo altera este criterio de optimalidad.
- Demanda Compartida. En este tipo de problemas es posible compartir la demanda de un cliente por más de un vehículo a diferencia de lo que ocurre en el VRP y VRPTW. Esta relajación puede significar la reducción del número de vehículos necesarios para atender la demanda con el consiguiente ahorro. El problema básico de esta familia se llama SDVRP-Split Delivery Vehicle Routing Problem o Problema de Rutas de Vehículos con Demanda Compartida.
- Restricciones de precedencia. En ocasiones, la distribución de la mercancía en el vehículo obliga a que las cargas má pesadas se sirvan después que las más ligeras. Esto supone una secuenciación en el orden de visita de los clientes. De nuevo, el orden de visita de los clientes está afectado por la presencia de estas restricciones invalidando muchas técnicas de optimizaci¶on que no las tienen en cuenta.
- Límites en el tiempo de transporte de cada mercancía. Tal es el caso del DARP Dial-A-Ride Problem, en el que las rutas contienen tramos de transporte de personas discapacitadas o mayores, de forma que aparte de que las rutas deben satisfacer unas ventanas de tiempo en los tiempos de recogida y entrega, las personas no deben permanecer en el vehículo más de un tiempo límite (ride time).
- Recogida y Entrega de mercancías. En ocasiones el mismo vehículo que entrega la mer-canca también debe recoger del cliente cierta mercancía (pensemos en el problema de repartir

refrescos embotellados y simultaneamente recoger los envases vacíos). Esta familia de problemas se llama Pick Up and Delivery Problems.

- Diversidad y existencia de varios objetivos. No solamente se puede aspirar a minimizar la distancia o tiempo utilizado por los veh¶³culos, sino en conseguir, por ejemplo, unas rutas equilibradas tanto en carga como en distancia o ambas cosas. Pensemos en las rutas de autobuses escolares, en donde se busca no sólo la optimalidad de las rutas en cuanto a tiempo global sino a minimizar el tiempo/distancia máxima que debe recorrer cada autobús, lo que puede suponer un aumento en el número de autobuses y mayor coste economico pero a cambio un mayor confort de los estudiantes trasladados.Las rutas no son circuitos sino caminos que visitan clientes, sin necesidad de retornar al depósito. Como por ejemplo el reparto de periódicos de suscripción en bicicleta en algunas ciudades.
- Múltiples depósitos. En este caso la flota de vehículos está repartida entre varios depósitos. Se debe decidir cuántos vehículos y desde que depósitos satisfacer las demandas de los clientes.

2.4.5. VRP's que pueden resolverse con Grafos:

2.4.5.1. VRP – ruta a coste mínimo

Se trata de un vehículo que debe visitar n lugares de una red geográfica para aprovisionarse o distribuir productos. Cada par de lugares (nodos del grafo) puede estar comunicada o no, su distancia o coste para ir de uno a otro se define mediante cij. El problema es por tanto, decidir el recorrido que comenzando por un determinado lugar pase por todos las demás al menos una vez (en la solución el vehículo puede pasar más de una vez por los lugares señalados, pero una de las veces será una parada para cargar o descargar mercancía; y finalmente vuelva finalmente al lugar de origen, de manera que se minimice la distancia/coste total recorrida/incurrido. La diferencia esencial con el TSP, es que en el caso del VRP, el vehículo puede, o ha de volver a un almacén o depósito en el caso de que exista una restricción de carga.

2.4.5.2. VRP – m rutas a coste mínimo

El Problema de m-Rutas trata de determinar los recorridos de m vehículos de capacidad homogénea que partiendo de un origen común deben pasar por un conjunto de lugares de interés para recoger o distribuir mercancías, y volver de nuevo al origen de manera que la distancia total recorrida, el coste o el tiempo empleado por el conjunto de vehículos sea mínima. En el tipo de problema más sencillo no se tiene en cuenta la capacidad de carga de los vehículos ni la demanda o cantidad de producto a recoger o entregar en cada lugar de interés; sin embargo, se añade la restricción de que como máximo un vehículo no podrá visitar más de p lugares de interés.

Este problema, es en realidad una combinación del Problema de Rutas y del Problema de los m-Viajantes de Comercio (m-Traveling Salesman Problem m-TSP).

2.4.5.3. VRP - Problema de rutas con vehículos capacitados CVRP

El Problema CVRP básico trata de determinar los recorridos de k vehículos de capacidad Ck que partiendo de un origen común deben pasar por un conjunto de lugares de interés (clientes) para recoger o distribuir mercancías según una demanda di, y volver de nuevo al origen de manera que la distancia total recorrida (el coste o el tiempo empleado) por el conjunto de vehículos sea mínima. En el tipo de problema más sencillo no se tiene en cuenta el horario de entrega o recogida en cada lugar de interés (ventanas horarias).

A partir de este problema básico aparecen todo un conjunto de extensiones o particularizaciones. Por ejemplo, la función objetivo podría ser:

- Minimizar el número total de vehículos (o conductores) requeridos para dar servicio a todos los clientes.
- Minimizar los costes fijos asociados con el uso de los vehículos o los conductores.
- Minimizar el coste total de transporte (coste fijo más variable de la ruta)
- Balancear las rutas, por tiempo de viaje o carga de vehículo.
- Minimizar las penalizaciones asociadas para un servicio parcial a los clientes

2.4.6. Algoritmo de programación lineal del CVRP

La siguiente ilustración muestra el modelo matemático de tres subíndices para el problema básico CVRP.

$$\min \sum_{i \in V} \sum_{j \in V} c_{ij} \sum_{k=1}^{K} x_{ijk}$$

$$s.a.$$

$$\sum_{k=1}^{K} y_{ik} = 1 \quad \forall i \in V^{\cdot}\cdot.\{0\}$$

$$\sum_{k=1}^{K} y_{0k} = K$$

$$\sum_{j \in V} x_{ijk} = \sum_{j \in V} x_{jik} = y_{ik} \quad \forall i \in V, k = 1 \ldots K$$

$$\sum_{i \in V} d_i y_{ik} \leq c_k \quad \forall k = 1 \ldots K$$

$$\sum_{i \in S} \sum_{j \notin S} x_{ijk \geq} y_{hk} \quad \forall S \subseteq V^{\cdot}\cdot.\{0\}, h \in S, k = 1 \ldots K$$

$$x_{ijk} \in \{0,1\} \quad \forall i, j \in V, k = 1 \ldots K$$

$$y_{ik} \in \{0,1\} \quad \forall i \in V, k = 1 \ldots K$$

Ilustración 53 - Pseudocódigo CVRP

Para un conjunto i, j de nodos del grafo, se expresa la función objetivo que intentará minimizar el coste total de todos los arcos recorridos en la solución. La variable binaria Xijk indica si el vehículo k tendrá una ruta utilizando el arco ij. Mientras, la variable binaria Yik indica si el nodo i con demanda di será atendido por el

vehículo k con capacidad Ck. Como se puede ver en la primera restricción cada nodo cliente deberá ser atendido únicamente por un vehículo (en el problema básico CVRP). En cambio del nodo origen 0 pueden partir todos los K vehículos de la flota. A continuación aparecen las restricciones de continuidad donde el vehículo que llegue a un cliente deberá también partir desde él. Tan sólo faltan las restricciones de capacidad: la demanda atendida por un vehículo (suma de di) no debe exceder su capacidad Ck. En el caso en que todos los vehículos tengan la misma capacidad, los valores Ck serán iguales. Por último aparecen condiciones de Miller y Tucker , y la definición de variables binarias.

2.4.7. Ejemplo problema de Rutas CVRP

2.4.7.1. Enunciado del caso

La atención sanitaria pública está dividida en áreas de salud. Alcoi es la capital de la zona o área 13. el área de salud de Alcoi puede considerarse constituida por las comarcas naturales de l'Alcoià y el Comtat, abarcando una población total en torno a los 150.000 habitantes. Estos se distribuyen en 43 entidades municipales (entre municipios y pedanías), que oscilan entre los 65.000 habitantes de Alcoi y los menos de 100 habitantes de Famorca, Tollos, Benillup, Ares, etc. Es un área con una fuerte dispersión demográfica, concentrándose aproximadamente el 65% de la población en tan sólo dos municipios: Alcoi e Ibi.

Ilustración 54 - Ambulancias De Servicio

El área 13 se encuentra situada en un conjunto montañoso del prebético valenciano, en el que abundan los valles. Ocupa una extensión de 786.6 km2, de los cuales el 65,13% del territorio está ocupado por montes. En su conjunto supera los 500 m de altitud, llegando algunas cimas a alcanzar entre los 900 y 1500 m sobre el nivel del mar. Su montañosa orografía, ha condicionado las dificultades de sus comunicaciones, internas y externas.

Dentro de la atención especializada que se realiza en el Hospital comarcal de Alcoi, los ciudadanos/as disponen del servicio de rehabilitación. Para el traslado de los pacientes, el hospital dispone de un servicio de ambulancias concertado con una empresa privada, la cual recoge a los pacientes en sus domicilios y los traslada al hospital para recibir el tratamiento. Posteriormente los devuelve a sus domicilios.

Este servicio dispone de 3 ambulancias. Todos los días, según los pacientes que se deben recoger, se deben calcular las rutas de las ambulancias para informar a los conductores de las mismas. Cada ambulancia

puede trasladar hasta 7 pacientes. Tendrán que realizar los servicios necesarios para atender a todos los pacientes.

El objetivo no es asignar ambulancias a diferentes zonas geográficas, sino que de manera dinámica, se calculen y asignen las rutas para que el servicio sea lo más eficiente posible.

2.4.7.2. Modelado

Para modelar este problema, lo primero que se ha hecho es introducir en el programa todos los pueblos del área con sus respectivas distancias (matriz de distancias entre nodos). Los datos se han obtenido del mapa oficial de carreteras (edición 32) del Ministerio de Obras Públicas, Transporte y Medio Ambiente. Con ello se construyó una matriz de datos en MsExcel que fue importada a Grafos.

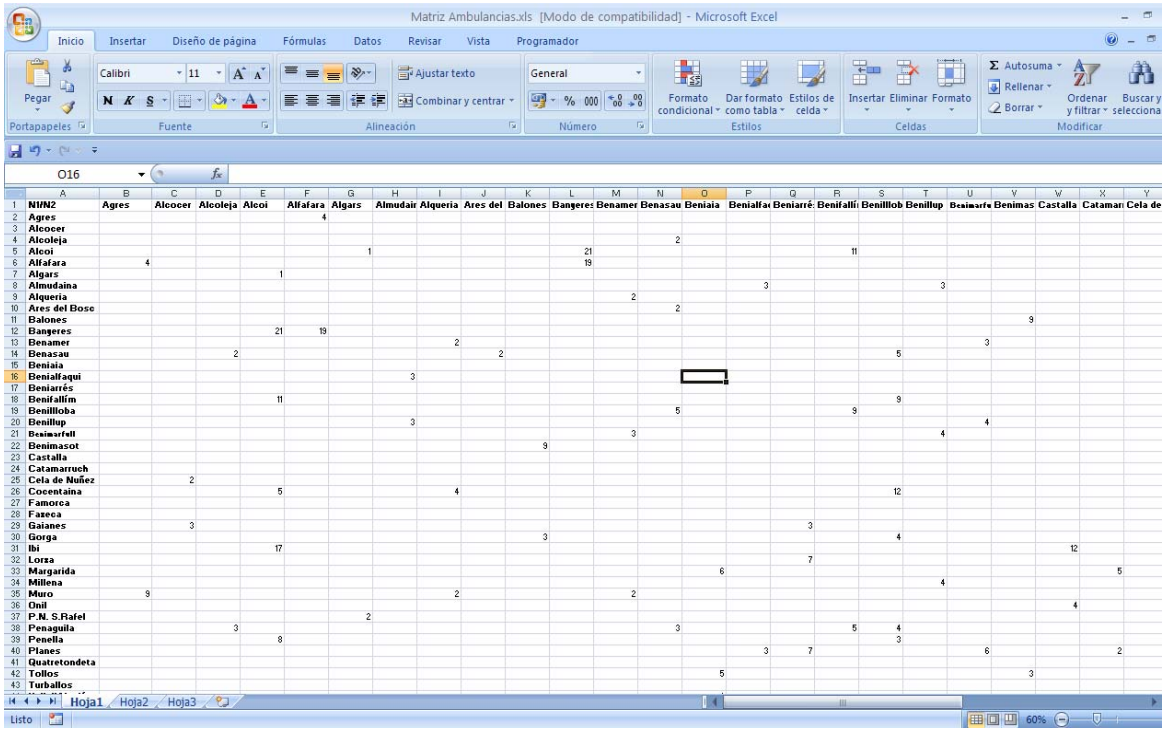


Ilustración 55 - Matriz de datos MS Excel

Una vez realizado esto, se edita el grafo en formato gráfico y se le va dando formato, de manera que refleje con mayor claridad la ubicación geográfica de las diferentes localidades. También se podría haber añadido un mapa de fondo para ilustrar de una manera más realista la geografía de la red, aunque no debemos olvidar que precisamente una de las cualidades de los grafos es su capacidad para simplificar la realidad, lo que permite una comprensión y análisis más sencillo y rápido.

La figura de la siguiente hoja, muestra como quedaría el grafo, una vez tratado los datos con el software Grafos.

Como se puede observar analizando el enunciado del problema, se trata de un problema transporte para vehículos capacitados (CVRP). El problema CVRP básico trata de determinar los recorridos de k vehículos de capacidad Ck que partiendo de un origen común deben pasar por un conjunto de lugares de interés (clientes) para recoger o distribuir mercancías según una demanda establecida di y volver de nuevo al origen de manera que la distancia total recorrida, el coste o el tiempo empleado por el conjunto de vehículos sea mínimo.

En el tipo de problema más sencillo no se tiene en cuenta el horario de entrega o recogida en cada lugar de interés (ventanas horarias). Las siguientes ilustraciones muestran el proceso de modelado de datos para la resolución de este caso práctico, dónde un conjunto de 3 ambulancias deben pasar por diferentes poblaciones cercanas a recoger pacientes de diálisis y llevarlos al hospital situado en la población de Alcoi. La restricción principal es la capacidad de servicio (en número de pacientes) de cada ambulancia.

En este tipo de problema el objetivo es minimizar la distancia total recorrida o coste del servicio, aunque otro posible objetivo sería minimizar el número de vehículos utilizados.

Se deben introducir los datos del enunciado (restricciones) en el software clickando en el icono que se muestra en la siguiente figura. A continuación aparecerá un desplegable dónde deben utilizarse los datos del enunciado:

Nº de ambulancias (3), capacidad de éstas (7 pacientes), nodo origen (Alcoi)…

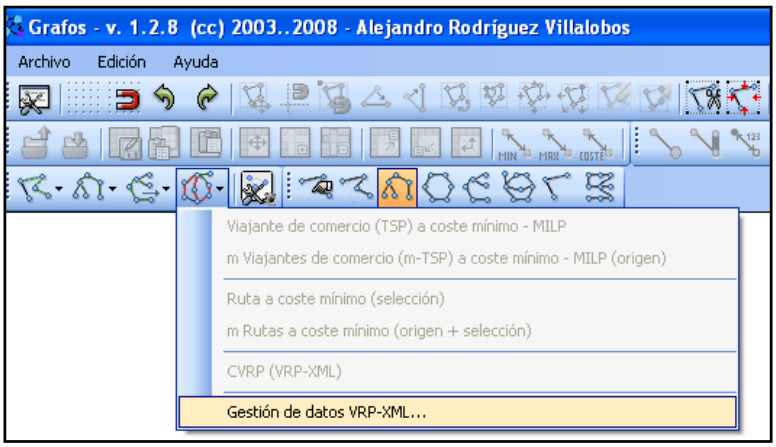


Ilustración 56 - Gestión de datos enunciado

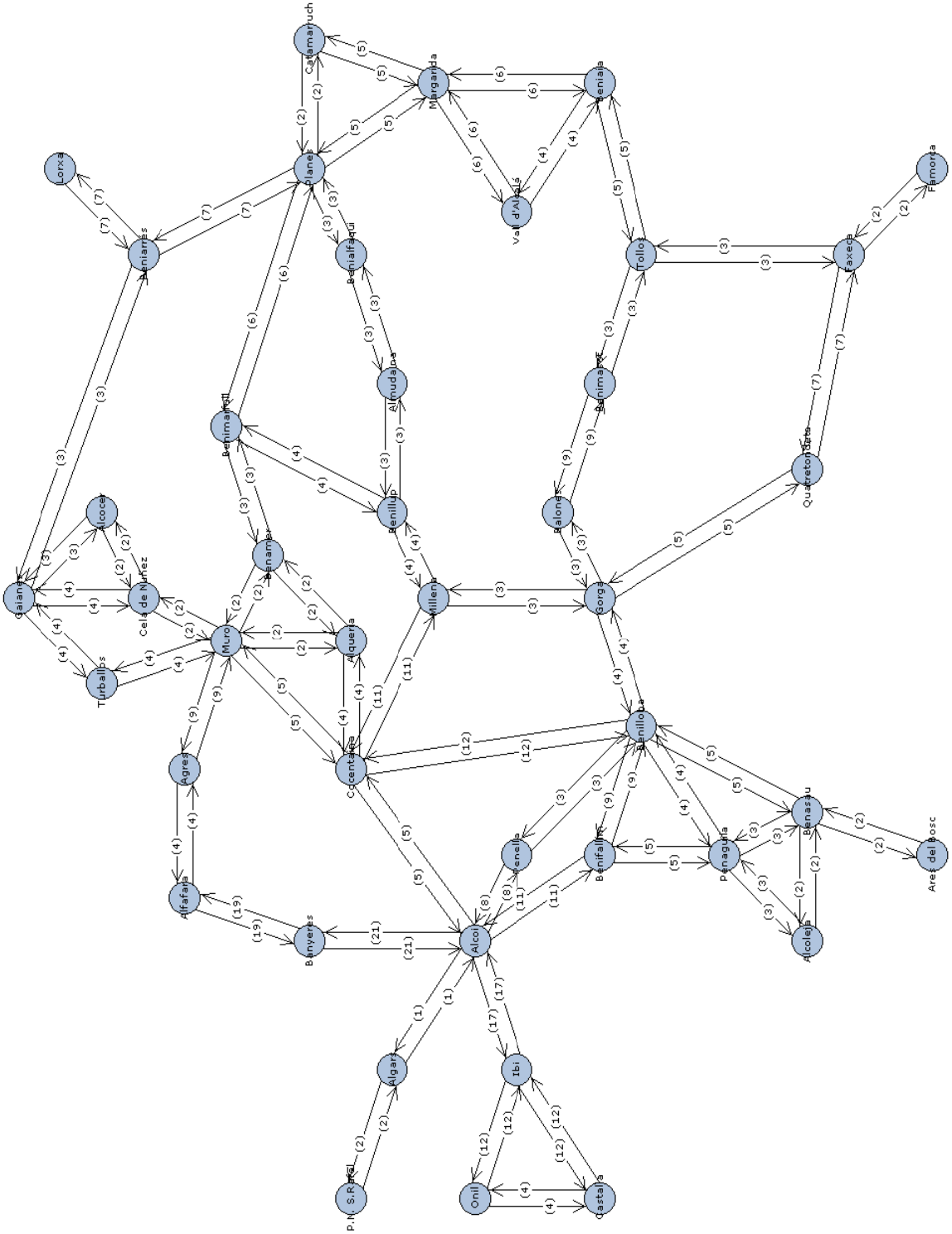


Ilustración 57 - Grafo de la Matriz de Datos

Posteriormente nos aparece la ventana siguiente en la que introduciremos los datos mencionados anteriormente, en cada una de las pestañas correspondientes de la ventana.

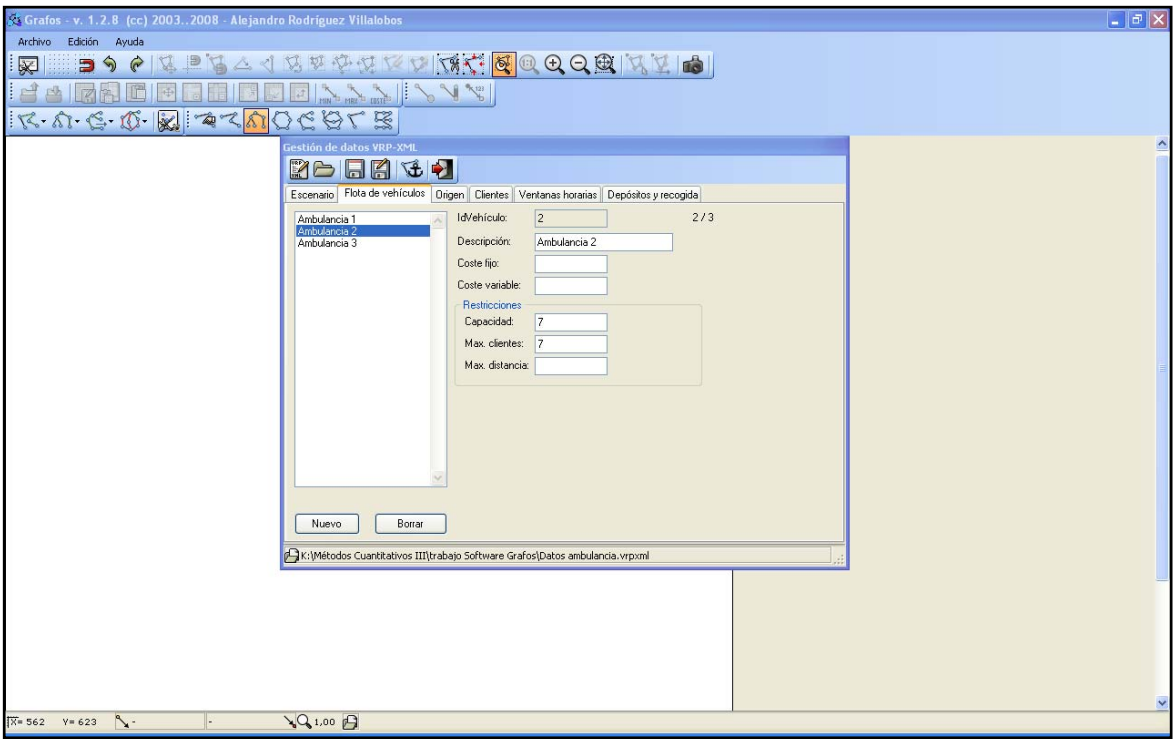


Ilustración 59 - Pantalla introducción de datos

2.4.7.3. Resolución

Una vez introducidos todos los datos del enunciado y las restricciones estamos en condiciones de darle la orden al programa para que nos resuelva el problema de rutas CVRP.

Grafos resuelve este problema mediante algoritmo de reducción del grafo, un modelo matemático lp[2] y una interpretación de los caminos y construcción de rutas para el grafo completo.

Una vez construida la solución final, ésta se puede analizar desde la ventana de resultados. Desde ella se obtiene toda la información detallada de todas las rutas solución. Además, se puede activar o desactivar la visualización de cada una de las rutas solución. De esta manera se puede observar con mayor claridad cada una de las rutas por separado.

A continuación podemos observar el report que nos ofrece el software grafos con la solución encontrada para su posterior interpretación.

[2] Lp: siglas de linear programmig.

DISTANCIA TOTAL MíNIMA - PROBLEMA DE RUTAS CON VEHíCULOS CAPACITADOS (CVRP)

Tiempo de proceso = 6 segundos

RUTA 1 :: IdVehÃculo 1: ambulancia 1

Distancia = 42

(21) > Alcoi, Banyeres

(21) > Banyeres, Alcoi

Servicio/Capacidad = Aprovechamiento (%): 2/7 = 28,57143 %

(Demanda) Cliente > UbicaciÃ³n:

(2) Banyeres > Banyeres

RUTA 2 :: IdVehÃculo 2: ambulancia 2

Distancia = 50

(11) > Alcoi, BenifallÃm

(24) > BenifallÃm, Benilloba, Gorga, Millena, Benillup, Benimarfull

(5) > Benimarfull, Benamer, Muro

(10) > Muro, Cocentaina, Alcoi

Servicio/Capacidad = Aprovechamiento (%): 6/7 = 85,71429 %

(Demanda) Cliente > UbicaciÃ³n:

(2) Benifallim > BenifallÃm

(2) Benimarfull > Benimarfull

(2) Muro > Muro

RUTA 3 :: IdVehÃculo 4: ambulancia 3

Distancia = 58

(17) > Alcoi, Ibi

(12) > Ibi, Onil

(29) > Onil, Ibi, Alcoi

Servicio/Capacidad = Aprovechamiento (%): 6/7 = 85,71429 %

(Demanda) Cliente > UbicaciÃ³n:

(4) Ibi > Ibi

(2) Onil > Onil

Distancia total = 150 unidades

```
Model name:  'PROBLEMA DE RUTAS CON VEHÃCULOS CAPACITADOS (CVRP) (grafo reducido)' - run #1
Objective:   Minimize(R0)

SUBMITTED
Model size:      796 constraints,     165 variables,         1257 non-zeros.
Sets:                                   0 GUB,                   0 SOS.

Using DUAL simplex for phase 1 and PRIMAL simplex for phase 2.
The primal and dual simplex pricing strategy set to 'Devex'.

Relaxed solution                  98 after         61 iter is B&B base.

Feasible solution                151 after        169 iter,          9 nodes (gap 53.5%)
Improved solution                150 after        542 iter,         67 nodes (gap 52.5%)

+Optimal solution                150 after      36187 iter,       4284 nodes (gap 51.5%).
Excellent numeric accuracy ||*|| = 4.44089e-016

MEMO: lp_solve version 5.5.0.2 for 32 bit OS, with 64 bit REAL variables.
      In the total iteration count 36187, 0 (0.0%) were bound flips.
      There were 2207 refactorizations, 0 triggered by time and 1 by density.
       ... on average 16.4 major pivots per refactorization.
      The largest [LUSOL v2.2.1.0] fact(B) had 1727 NZ entries, 1.0x largest basis.
      The maximum B&B level was 34, 0.1x MIP order, 9 at the optimal solution.
      The constraint matrix inf-norm is 7, with a dynamic range of 7.
      Time to load data was 0.015 seconds, presolve used 0.031 seconds,
       ... 6.468 seconds in simplex solver, in total 6.514 seconds.
```

Para finalizar con la resolución del problema se muestran una imagen con las "rutas solución" que nos ofrece el software Grafos.

- Ambulancia 1
- Ambulancia 2
- Ambulancia 3

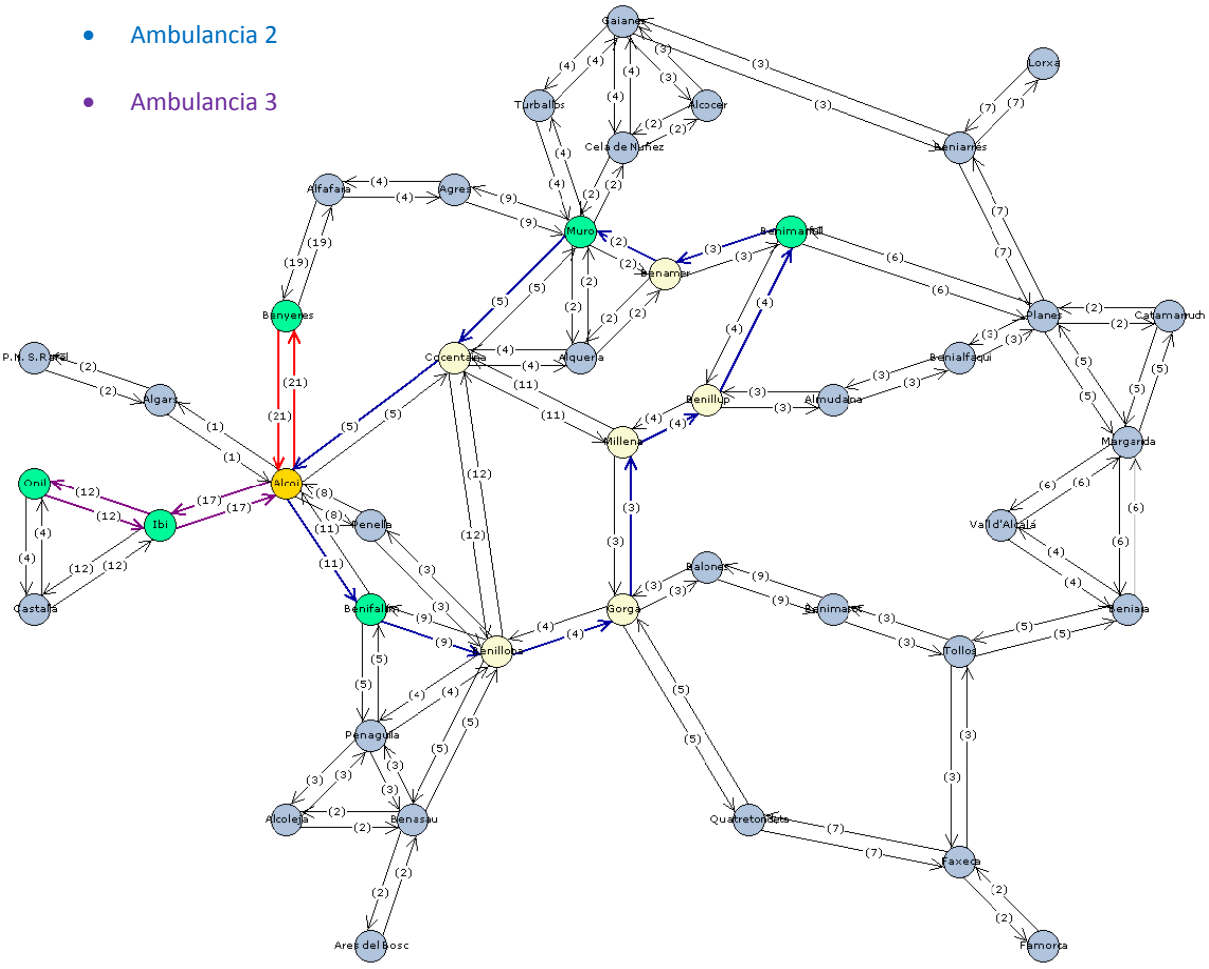


ILUSTRACIÓN 60 - SOLUCIÓN CON LAS RUTAS DE LAS AMBULANCIAS

Este tipo de análisis podrían realizarse a diario para optimizar la asignación de rutas y el coste del transporte. O bien, de manera planificada, si se dispone de la información de demanda de servicio con suficiente anterioridad.

3. Biografias Autores:

3.1. Dijkstra

Edsger Wybe Dijkstra nació en Rotterdam, (Holanda) en 1930. Sus padres eran ambos intelectuales y él recibió una excelente educación. Su padre era químico y su madre matemática. En 1942, cuando Dijkstra tenía 12 años, entró en Gymnasium Erasminium, una escuela para estudiantes especialmente brillantes, donde dio clases, fundamentalmente, de Griego, Latín, Francés, Alemán, Inglés, biología, matemáticas y química. En 1945, Dijkstra pensó estudiar Derecho y trabajar como representante de Holanda en las Naciones Unidas.

Sin embargo, debido a su facilidad para la química, las matemáticas y la física, entró en la Universidad de Leiden, donde decidió estudiar física teórica. Durante el verano de 1951, asistió a un curso de verano sobre programación en la Universidad de Cambridge. A su vuelta empezó a trabajar en el Centro Matemático en Amsterdam, en marzo de 1952, donde se incrementó su creciente interés en la programación. Cuando terminó la carrera se dedicó a problemas relacionados con la programación. Pero uno de los problemas con que se encontró es que ser programador no estaba oficialmente reconocido como una profesión. De hecho, cuando solicitó una licencia de matrimonio en 1957, tuvo que señalar que su profesión era físico teórico.

Dijkstra continuó trabajando en el Centro Matemático hasta que aceptó un trabajo como desarrollador en Burroughs Corporation, en los Estados Unidos, a principio de la década de los 70. En 1972 ganó el Premio Turing ACM, y ,en 1974, el AFIPS Harry Good Memorial. Dijkstra se trasladó a Austin, Texas a principio de los 80. En 1984, se le ofreció un puesto en Ciencias de la Computación en la Universidad de Texas, donde permaneció desde entonces hasta que recientemente el 6 Agosto del 2002, Edsger W. Dijkstra, Professor Emeritus of Computer Sciences and Mathematics at The University of Texas en Austin, falleció en su hogar de Nuenen, (Netherlands). Es miembro honorario de la Academia Americana de Artes y Ciencias y de Real Academia Holandesa de Artes y Ciencias. Además es miembro distinguido de la Sociedad de Computación Británica. Finalmente es Doctor Honoris Causa en Ciencias por la Queen's University Belfast.

3.2. Bellman-Ford

Lester Randolph Ford es uno de los pioneros en el campo de la programación de flujos en grafos. Es el hijo de L.R. Ford Sr. (quién también es un matemático distinguido) y nació el 23 de septiembre de 1927. L. R. Ford Sr es elogiado por su ejemplar trabajo en matemáticas al inventar una interpretación geométrica absolutamente maravillosa de la serie de Farey. También le acredita su trabajo 'Pointwise Discontinuous Functions' que era la base de su trabajo para un grado de M.S. del departamento de matemáticas en la universidad de Missouri-Colombia en 1912. Tal fue su contribución a las matemáticas, que en 1964 se estableció el Lester R. Ford Award para reconocer la contribución a las matemáticas de excelentes autores matemáticos publicados en The American Mathematical Monthly o Mathematics Magazine. Fue redactor de American Mathematical Monthly, de 1942-1946, y el presidente de Mathematical Association of America,

1947-1948. Ford Sr. y Ford Jr. son co-autores de Automorphic Functions cuál fue publicado cerca por McGraw-Hill en 1963.

Al continuar los pasos de su padre Ford Jr. también hizo una enorme contribución al campo de las matemáticas. Su trabajo con Delbert Ray Fulkerson (14 agosto de 1924 - 10 Enero de 1976) ha puesto la base de casi toda la investigación en flujos de grafos. El artículo de Ford y de Fulkerson (1956) con el problema de flujo máximo estableció el famoso teorema del flujo máximo - mínimo corte.

Mientras trabajó en RAND CORPORATION, Ford Jr publicó numerosos artículos que no solo establecieron la base de los flujos de red sino también la futura investigación en este campo. En 1962 Priceton University Press publicó su libro Flow in Networks con D. R. Fulkerson como co-autor. Este libro contiene todo su trabajo sobre redes.

La mayoría del trabajo de Ford lo hizo en la colaboración con Fulkerson, al parecer los dos hacían una buena asociación. Sin embargo, en 1956 presentó varios artículos firmados por él sólo. Ha sido el autor de diversos algoritmos que se han refinado con los años y que todavía se utilizan para solucionar la mayoría de problemas de grafos.

3.3. Floyd-Warshall

Robert W. Floyd nació el 8 de junio de 1936 en Nueva York, es profesor de la Stanford University (B.A. Chicago 1955 B.S. Chicago 1958), y en 1978 fue galardonado con el prestigioso premio A.M. Turing que otorga la ACM para reconocer las contribuciones de naturaleza técnica realizadas a la comunidad informática. El premio le fue concedido por tener una influencia clara en las metodologías para la creación de software eficiente y fiable, y por ayudar a fundar las siguientes áreas de la informática: teoría de análisis sintáctico, semántica de los leguajes de programación, verificación automática de programas, síntesis automática de programas y análisis de algoritmos.

Fue uno de los inventores del deterministic linear time selection algorithm. También introdujo mejoras en los algoritmos quicksort y quickselect.

3.4. Kruskal

Grafos forma parte de un proyecto de investigación y desarrollo de aplicaciones informáticas de diseño modular orientadas hacia la docencia, investigación y labores profesionales de Ingeniería de Organización. Grafos contiene entre otras una DLL con el Algoritmo de Kruskal:

Joseph B. Kruskal investigador del Math Center (Bell-Labs), que en 1956 descubrió su algoritmo para la resolución del problema del Árbol de coste total mínimo (minimum spanning tree - MST) también llamado árbol recubridor euclídeo mínimo. Este problema es un problema típico de optimización combinatoria, que fue considerado originalmente por Otakar Boruvka (1926) mientras estudiaba la necesidad de electrificación rural en el sur de Moravia en Checoslovaquia.

El objetivo del algoritmo de Kruskal es construir un árbol (subgrafo sin ciclos) formado por arcos sucesivamente seleccionados de mínimo peso a partir de un grafo con pesos en los arcos.

Un árbol (spanning tree) de un grafo es un subgrafo que contiene todos sus vértices o nodos. Un grafo puede tener múltiples árboles. Por ejemplo, un grafo completo de cuatro nodos (todos relacionados con todos) tendría 16 árboles.

La aplicación típica de este problema es el diseño de redes telefónicas. Una empresa con diferentes oficinas, trata de trazar líneas de teléfono para conectarlas unas con otras. La compañía telefónica le ofrece esta interconexión, pero ofrece tarifas diferentes o costes por conectar cada par de oficinas. Cómo conectar entonces las oficinas al mínimo coste total.

La formulación del MST también ha sido aplicada para hallar soluciones en diversas áreas (diseño de redes de transporte, diseño de redes de telecomunicaciones - TV por cable, sistemas distribuidos, interpretación de datos climatológicos, visión artificial - análisis de imágenes - extracción de rasgos de parentesco, análisis de clusters y búsqueda de superestructuras de quasar, plegamiento de proteínas, reconocimiento de células cancerosas, y otros).

Otra aplicación menos obvia es que el árbol de coste total mínimo puede ser usado como solución aproximada al problema del viajante de comercio (traveling salesman problem), recuerde que encontrar la solución óptima a este problema es NP-Hard. La manera formal de definir este problema es encontrar la trayectoria más corta para visitar cada punto al menos una vez. Nótese que si se visitan todos los puntos exactamente una vez, lo que se tiene es un tipo especial de árbol. En el ejemplo anterior, 12 de los 16 árboles son trayectorias de este tipo. Si se tiene una trayectoria que visita algunos vértices o nodos más de una vez, siempre se puede soltar algunos nodos del árbol. En general el peso del árbol total mínimo es menor que el del viajante de comercio, debido a que su minimización se realiza sobre un conjunto estrictamente mayor. Existen diferentes algoritmos y maneras de usar el árbol de coste total mínimo para encontrar la solución al problema del viajante de comercio (con resultados cercanos al óptimo).

3.5. Prim

Robert Prim en 1957 descubrió un algoritmo para la resolución del problema del Árbol de coste total mínimo (minimum spanning tree - MST). Este problema es un problema típico de optimización combinatoria, que fue considerado originalmente por Otakar Boruvka en 1926 mientras estudiaba la necesidad de electrificación rural en el sur de Moravia en Checoslovaquia. Este problema también fue resuelto por Joseph B. Kruskal en 1956.

En los años 60 estos científicos del Math Center (Bell Labs) fueron los pioneros de la moderna teoría de secuenciación, particularmente en el análisis de algoritmos de aproximación y en secuenciación multiprocesador (Ed Coffman, Ron Graham, David Johnson y Mike Garey). En las siguientes tres décadas, se añadieron multitud de contribuciones que mejoraron la teoría general.

El algoritmo de Prim encuentra un árbol de peso total mínimo conectando nodos o vértices con arcos de peso mínimo del grafo sin formar ciclos.

Al igual que el algoritmo de Kruskal, el de Prim también ha sido aplicado para hallar soluciones en diversas áreas (diseño de redes de transporte, diseño de redes de telecomunicaciones - TV por cable, sistemas distribuidos, interpretación de datos climatológicos, visión artificial - análisis de imágenes - extracción de rasgos de parentesco, análisis de clusters y búsqueda de superestructuras de quasar, plegamiento de proteínas, reconocimiento de células cancerosas, y otros). También se ha utilizado para encontrar soluciones aproximadas a problemas NP-Hard como el del 'viajante de comercio'.

3.6. Ford-Furkelson

Lester Randolph Ford es uno de los pioneros en el campo de la programación de flujos en grafos. Es el hijo de L.R. Ford Sr. (quién también es un matemático distinguido) y nació el 23 de septiembre de 1927. L. R. Ford Sr es elogiado por su ejemplar trabajo en matemáticas al inventar una interpretación geométrica absolutamente maravillosa de la serie de Farey. También le acredita su trabajo 'Pointwise Discontinuous Functions' que era la base de su trabajo para un grado de M.S. del departamento de matemáticas en la universidad de Missouri-Colombia en 1912. Tal fue su contribución a las matemáticas, que en 1964 se estableció el Lester R. Ford Award para reconocer la contribución a las matemáticas de excelentes autores matemáticos publicados en The American Mathematical Monthly o Mathematics Magazine. Fue redactor de American Mathematical Monthly, de 1942-1946, y el presidente de Mathematical Association of America, 1947-1948. Ford Sr. y Ford Jr. son co-autores de Automorphic Functions cuál fue publicado cerca por McGraw-Hill en 1963.

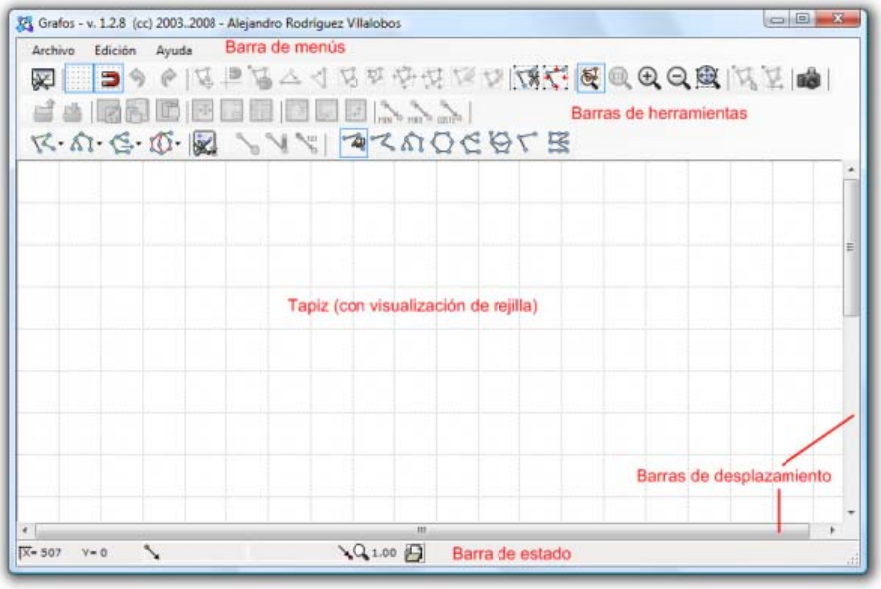
Al continuar los pasos de su padre Ford Jr. también hizo una enorme contribución al campo de las matemáticas. Su trabajo con Delbert Ray Fulkerson (14 agosto de 1924 - 10 Enero de 1976) ha puesto la base de casi toda la investigación en flujos de grafos. El artículo de Ford y de Fulkerson (1956) con el problema de flujo máximo estableció el famoso teorema del flujo máximo - mínimo corte. Mientras trabajó en RAND CORPORATION, Ford Jr publicó numerosos artículos que no solo establecieron la base de los flujos de red sino también la futura investigación en este campo. En 1962 Priceton University Press publicó su libro Flow in Networks con D. R. Fulkerson como co-autor. Este libro contiene todo su trabajo sobre redes.

La mayoría del trabajo de Ford lo hizo en la colaboración con Fulkerson, al parecer los dos hacían una buena asociación. Sin embargo, en 1956 presentó varios artículos firmados por él sólo. Ha sido el autor de diversos algoritmos que se han refinado con los años y que todavía se utilizan para solucionar la mayoría de problemas de grafos.

4. Manual para la utilización de Grafos:

4.1. Conocer la interfaz de usuario

Es importante familiarizarse con la interfaz de usuario del programa. Grafos tiene una interfaz sencilla e intuitiva que fácilmente aprenderá a utilizar. La siguiente imagen muestra es aspecto general de la interfaz y sus principales elementos.



En futuras versiones del programa, esta interfaz podría cambiar, incorporando nuevos elementos o variando su aspecto estético.

A continuación se describen con mayor detalle estos elementos.

4.2. El tapiz

El tapiz debe entenderse como una hoja de papel o una superficie donde el usuario puede construir y editar un grafo de manera gráfica. Algunas de las principales características del tapiz son:

- Se puede cambiar el ancho y alto del tapiz a voluntad (pixeles).
- Se puede elegir un color sólido personalizado para el fondo del tapiz.

En lugar de un color sólido, se puede incluir una imagen de mapa de bits como fondo, sobre la cual construir o mostrar el grafo.

El tapiz tiene un sistema de coordenadas X Y (en pixeles) que ayuda a la localización de los elementos del grafo. El origen de coordenadas X=0 Y=0 se encuentra en la esquina superior izquierda del tapiz. Los ejes X e Y son crecientes hacia abajo y a la derecha de la pantalla respectivamente.

Se puede cambiar la escala (mediante el zoom) a la cual se muestra el tapiz y el grafo construido sobre él. El tapiz tiene una rejilla (personalizable) que facilita la alineación de los elementos del grafo. Esta rejilla se puede mostrar u ocultar a voluntad.

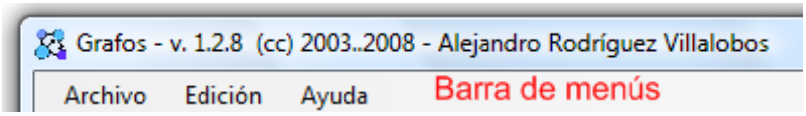
Las barras de desplazamiento horizontal y vertical, permiten desplazarse por el tapiz en el caso de que sus dimensiones sean mayores que la ventana de Grafos.

Esta superficie de trabajo permite una edición gráfica WYSIWYG ('lo que usted ve, es lo que usted obtiene'). Esto quiere decir que podrá exportar la imagen del grafo construido, imprimir o copiar la imagen resultante a la misma escala en pixeles que usted lo visualiza en pantalla.

Más adelante se explicará (al tratar los menús y barras de herramientas) como trabajar sobre el tapiz y personalizar sus características.

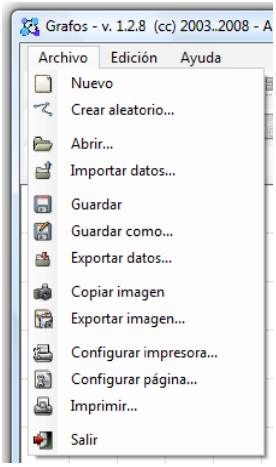
4.3. Los menús

Tal y como se comentaba en el apartado de intefaz de usuario, en la parte superior de la ventana se encuentra la barra de menús.



Ésta barra contiene tres menús principales tal y como muestran las siguientes imágenes:

4.4. Archivo

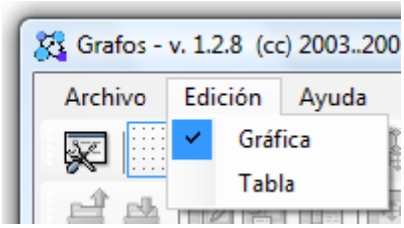


El menú Archivo está estructurado en seis bloques de opciones. Si usted ya ha trabajado en el entorno Windows estará ya familiarizado con muchas de estas opciones. A través de este menú usted podrá:

- Crear un nuevo grafo partiendo de cero.
- Crear un nuevo grafo aleatorio.
- Abrir un grafo existente y previamente creado y guardado.
- Importar datos a su grafo actual.

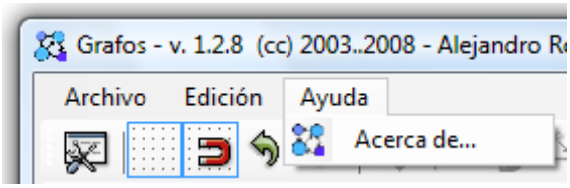
- Guardar el grafo actual con el nombre y la ubicación deseada.
- Exportar los datos del grafo a otros formatos de fichero.
- Copiar la imagen del grafo al 'portapapeles de Windows'.
- Exportar la imagen del grafo a diferentes formatos de fichero gráfico.
- Configurar la impresora y el modo de impresión.
- Imprimir la imagen del grafo que está visualizando.
- Cerrar y abandonar el programa.
- Posteriormente se explicarán estas opciones con mayor detalle.

4.5. Edición



El menú Edición indica cuál es el modo de edición actual (algo evidente a simple vista) y permite permutar entre el modo de edición gráfica WYSIWYG y el modo de edición tabular (o edición matricial). La construcción y edición en ambos modos se explicará más adelante detenidamente.

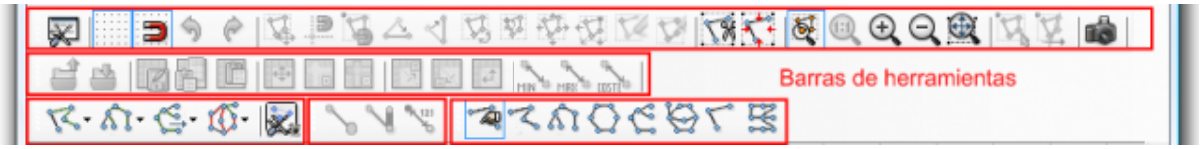
4.6. Ayuda



El menú de Ayuda de momento no muestra un manual de usuario o un acceso a una ayuda contextual.

4.7. Las barras de herramientas

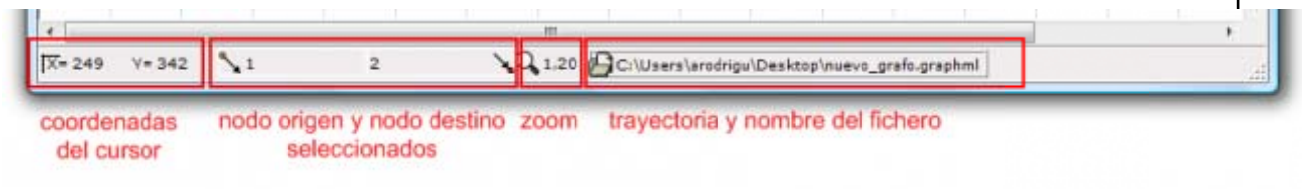
Las barras de herramientas están situadas en la parte superior de la interfaz de usuarios. En la actualidad (versión 1.2.8) existen 5 barras de herramientas:



En ellas se encuentran botones de comandos, botones de opciones y menús desplegables para realizar todo un conjunto de tareas (tanto en el modo de edición gráfica, como en el modo de edición tabular). Más adelante se explicarán cada una de las barras de herramientas y sus botones.

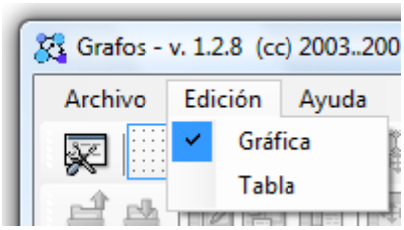
4.8. La barra de estado

La barra de estado está situada en la parte inferior de la ventana de Grafos. En esta barra se muestra información adicional que ayuda al usuario a situarse en la intarfaz.



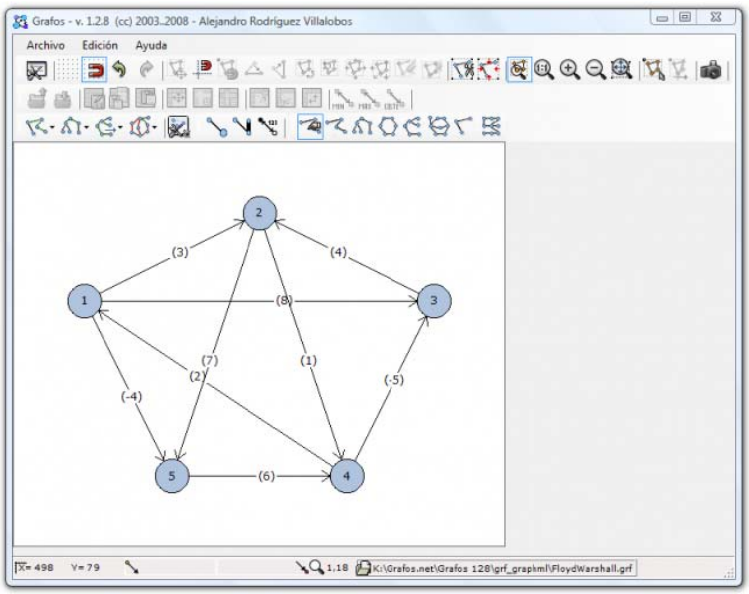
El primer bloque de información muestra las coordenadas X e Y del ratón sobre el tapiz. A continuación se muestra el identificador de los nodos origen y destinos seleccionados. Si no se selecciona alguno de estos nodos, no aparecerá su identificador. Seguidamente se muestra el factor de ampliación o zoom, un factor de 1,20 como muestra la imagen, quiere decir que se está ampliando la imagen original un 20%. Por último, se muestra el nombre y trayectoria del fichero creado (una vez guardado o abierto).

4.9. Diferencias entre el modo gráfico y el modo tabular

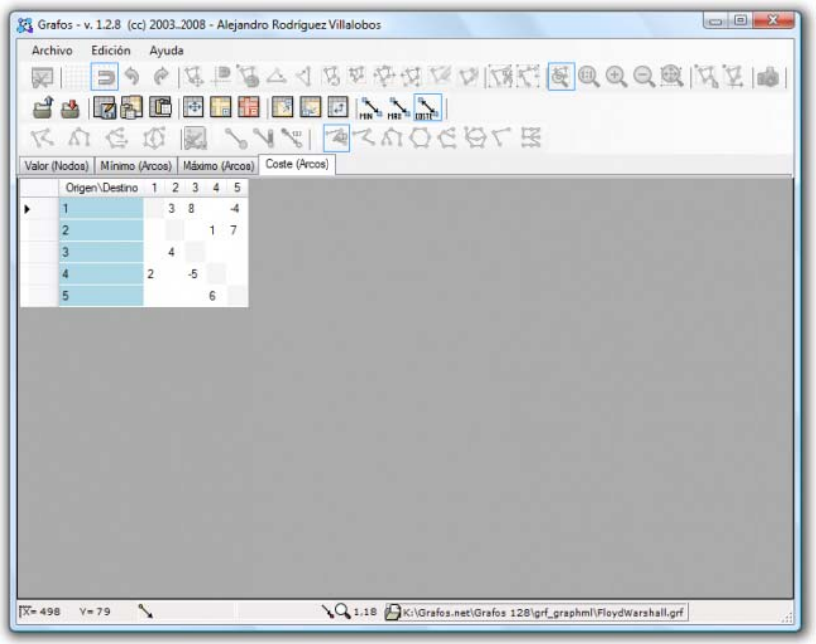


El menú Edición indica cuál es el modo de edición y permite cambiar entre ambos modos.

El modo de edición gráfica, trabaja en un entorno WYSIWYG. Usted podrá editar o construir un grafo como si estuviera dibujándolo gráficamente.



El modo de edición tabular en cambio sustituye el entorno de trabajo sobre el tapiz, por un entorno en forma de tabla. A diferencia del anterior, aquí se introducen y modifican los datos en las tablas correspondientes a cada matriz (edición matricial).

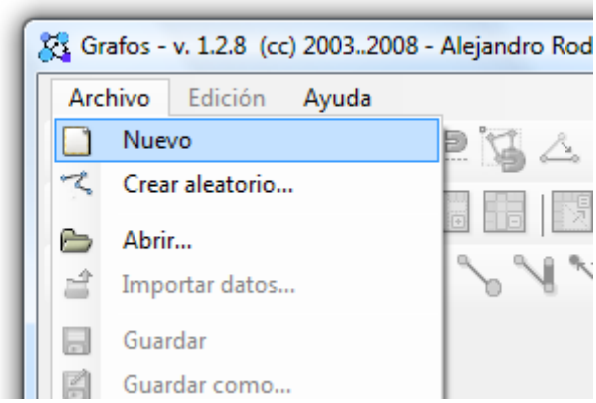


Hay que subrayar que en realidad Grafos trabaja con 4 matrices de datos. La primera de ellas corresponde a los Nodos y será la encargada de guardar el identificador o nombre de un nodo y un valor numérico asociado. Las otras tres matrices se refieren a los Arcos, y realmente son las responsables de la construcción de grafo propiamente dicho, ya que los arcos son las relaciones entre el conjunto de nodos. Cada arco tiene asociado tres posibles valores numéricos: valor mínimo, valor máximo y coste. Son como tres dimensiones de un mismo arco; es por ello, por lo que existen tres matrices o tablas. Según la tabla que se edite, se estará cambiando o modificando una vista del grafo, aunque las dimensiones del grafo serán las mismas para todas las vistas; es decir, siempre existirán el mismo número de nodos y de arcos, sea cuál sea la vista editada. Si en una vista se añaden o borran nodos, esto afectará al resto de vistas. Si en una vista se añade un arco, éste aparecerá en el resto aunque con valores cero.

Otra de las diferencias entre ambos modos, es que mientras que en el modo gráfico se presta especial atención a la posición de los elementos del grafo y su estilo gráfico (colores, grosor del trazo, localización, etc.); en el modo tabular, se favorece la rapidez en la edición de los valores numéricos, dejando de lado la representación gráfica y su estilo. En cualquier caso, las modificaciones de los datos en modo gráfico afectarán a las matrices de datos observadas en modo tabular, y viceversa. Si usted tiene pensado trabajar con gran cantidad de datos o matrices grandes, es recomendable comenzar a construir y editar el grafo en modo tabular, dejando para más adelante el formato gráfico del mismo (perfeccionamiento del estilo antes de imprimir o exportar).

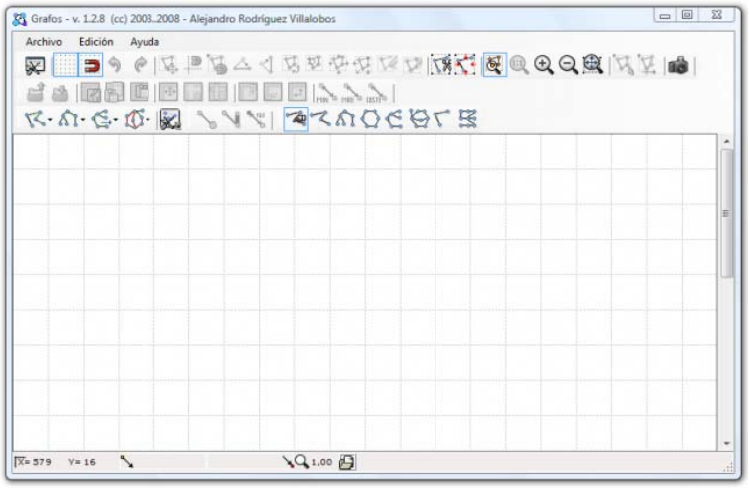
Más adelante se explicará con ejemplos cómo editar los datos de las diferentes vistas, tanto en modo gráfico como en modo tabular.

4.10. Cómo crear un nuevo grafo (modo gráfico)



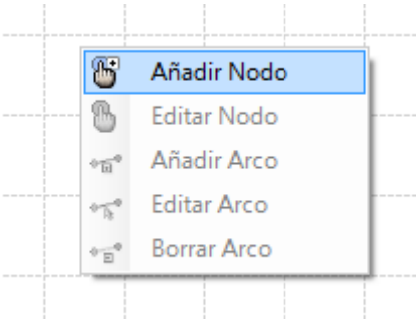
Cuando inicie Grafos por primera vez, observará que no existe ningún grafo cargado y ni siquiera se visualiza el tapiz de fondo (fondo de color gris sin tapiz). Para crear un nuevo grafo y comenzar a trabajar partiendo de cero, vaya al menú Archivo y seleccione la opción Nuevo.

A continuación observará que inicia el modo de edición gráfica, y ya dispone de un tapiz, en blanco donde comenzar a trabajar. Todavía no existe un grafo propiamente dicho, solamente una superficie sobre la cual comenzar a crearlo.



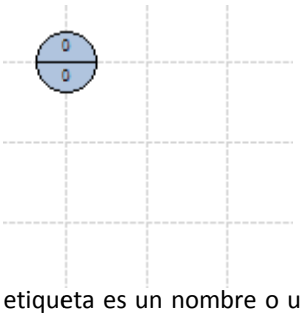
Seguidamente veremos cómo se puede construir un grafo, añadiendo Nodos y las relaciones entre ellos: los Arcos.

4.11. Añadir nodos



Para crear o añadir nodos a su grafo, lo que debe hacer es pulsar el botón derecho del ratón sobre un espacio libre del tapiz. Rápidamente aparecerá un menú contextual con diversas opciones, entre ellas Añadir Nodo.

Si pulsa esta opción del menú se dibujará un nodo en la posición del ratón, tal y como muestra la siguiente figura.  Si está activada la opción Modo imantar a rejilla de la barra de herramientas, el nodo creado será colocado en la cuadrícula de la rejilla más próxima a la posición del ratón.



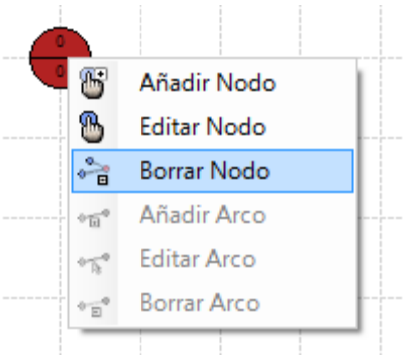
4.12. Etiqueta y Valor de un nodo

Cada nodo tiene dos propiedades importantes, la etiqueta y el valor. La etiqueta es un nombre o un número que permite identificarlo. La etiqueta se muestra en la parte superior del nodo (sobre su ecuador). El primer nodo creado es el nodo cero, por eso su etiqueta muestra el valor cero encima del ecuador. El segundo tendrá una etiqueta 2, el tercero 3 y así sucesivamente. Posteriormente, usted podrá editar la etiqueta a su voluntad.

El valor de un nodo es utilizado en algunos algoritmos como una cifra numérica que indica por ejemplo, el consumo o demanda de un nodo. Por defecto, el valor de todos los nuevos nodos creados es cero, y éste se muestra en la parte inferior al ecuador.

4.13. Borrar sólo un nodo

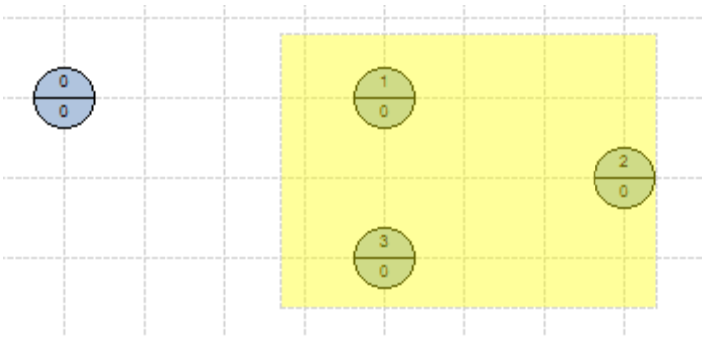
El proceso de borrar un nodo es muy sencillo, sobre el nodo que desea borrar pulse el botón derecho del ratón. El nodo se coloreará de color rojo y aparecerá el siguiente menú contextual. A continuación seleccione la opción Borrar Nodo y el nodo desaparecerá. Tenga presente que si el nodo seleccionado está conectado con arcos a otros nodos, dichos arcos de conexión también serán borrados automáticamente.



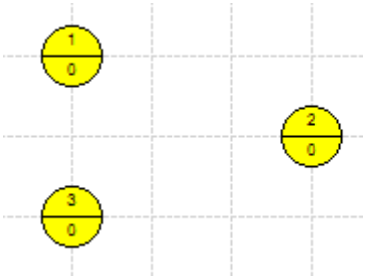
4.14. Borrar varios nodos a la vez

Si lo que desea es borrar varios nodos a la vez, primero deberá seleccionarlos. Para ello, pulse el botón izquierdo sobre el tapiz (cerca de los nodos a seleccionar) y arrastre el ratón. Formará una región de selección tal y como muestra la siguiente pantalla.

Cuando levante el dedo del botón del ratón, la operación de selección terminará, subrayando en color amarillo los nodos seleccionados. 



Para borrar los nodos seleccionados, pulse el botón Borrar todos los nodos seleccionados y sus arcos de la barra de herramientas. A continuación se borrarán todos los nodos de la selección, y también sus conexiones (arcos entrantes o salientes de dichos nodos).

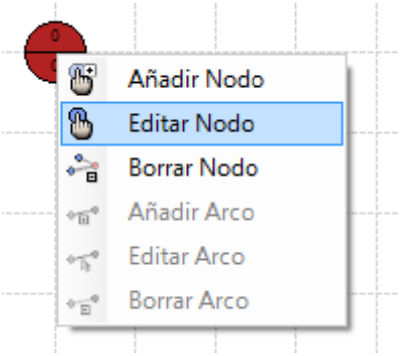


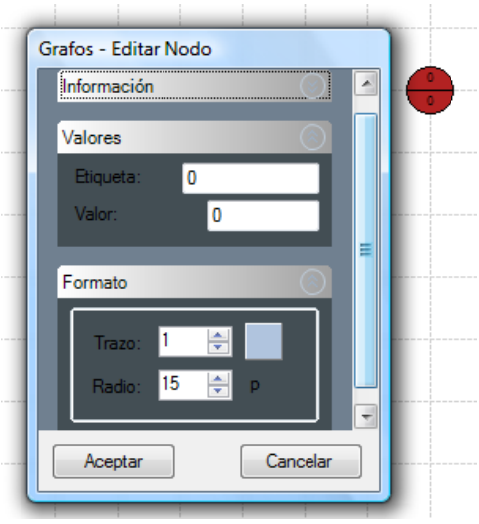
4.15. Añadir o quitar nodos de la selección

Para añadir o quitar nodos de la selección, vuelva a crear una zona de selección sobre los nodos deseados, pulsando el botón izquierdo y arrastrando el ratón. Si el nodo no estaba seleccionado, se añadirá a la selección y viceversa. Puede repetir esta operación hasta tener el conjunto deseado.

4.16. Editar nodos

Editar un nodo es cambiar sus Valores y/o Formato. De modo similar a borrar un nodo, para editarlo haga click con el botón derecho del ratón sobre él. Aparecerá el siguiente menú contextual donde debe seleccionar la opción Editar Nodo. Seguidamente aparecerá una pequeña ventana de opciones sobre las propiedades del Nodo.

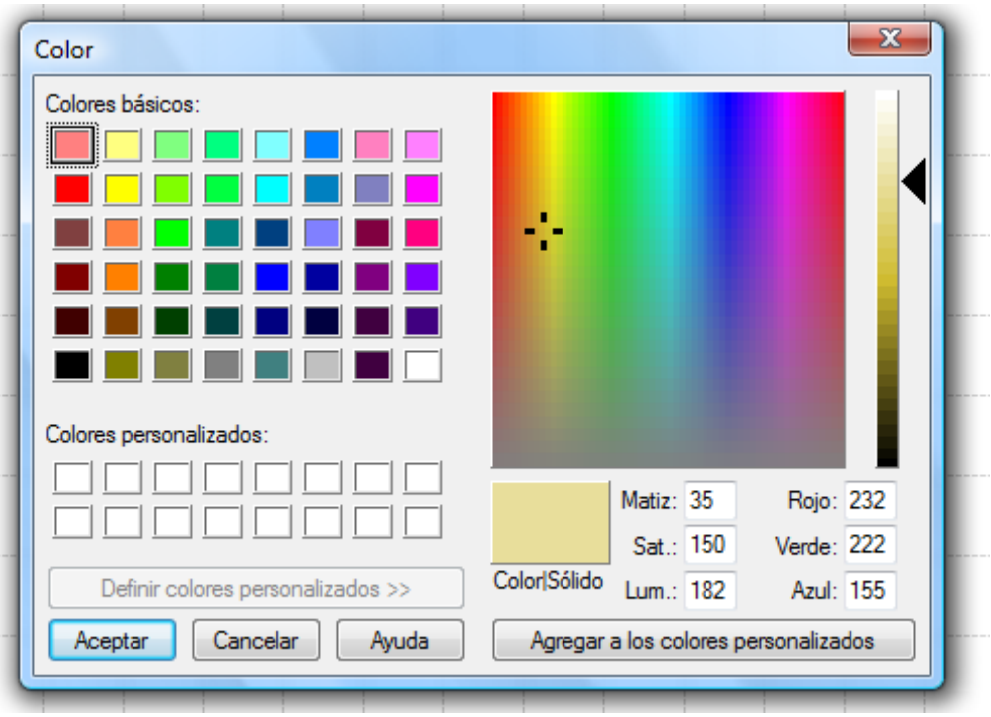




Desde esta ventana, se puede:

- Cambiar la etiqueta del nodo
- Cambiar el valor del nodo (para algunos algoritmos)
- Cambiar el radio (tamaño del nodo en pixels)
- Cambiar el color de relleno del nodo

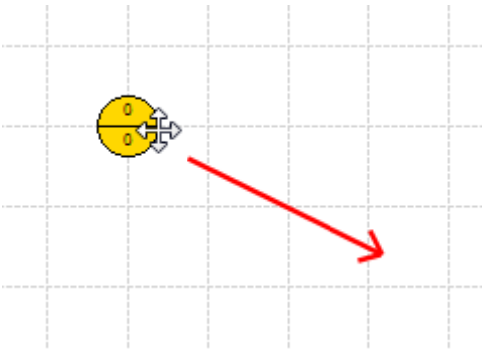
Para cambiar el color de relleno del nodo, pulse sobre el cuadro de color, aparecerá una ventana de diálogo de selección de color.



Se puede personalizar el aspecto de cada nodo individualmente o de todos en su conjunto. A continuación se puede ver un ejemplo de un nodo personalizado.

4.17. Mover nodos

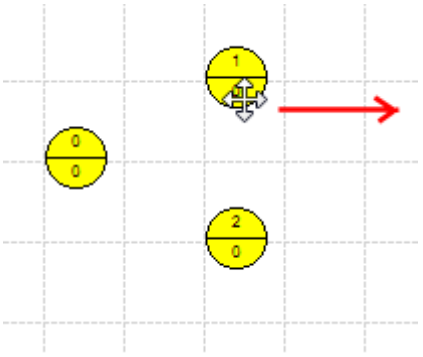
Para mover un nodo a otra parte del tapiz, simplemente coloque el cursor del ratón sobre él, a continuación pulse el botón derecho del ratón, y manténgalo pulsado mientras mueve el ratón. De este modo arrastrará el nodo a la posición deseada. Al soltar el botón del ratón, el nodo quedará ubicado en su nueva posición.



Recuerda que si está activa la opción de Imantar nodos a la rejilla, el nodo quedará ubicado en la intersección de la rejilla más próxima. Si esta opción no está activa, usted puede mover libremente el nodo. Cambiando el espaciado o densidad de la rejilla,

puede controlar las distancias entre nodos.

4.18. Mover varios nodos



 Si lo que desea es mover varios nodos, deberá activar la opción Mover nodos seleccionados (arrastre ratón) de la barra de herramientas.

Una vez seleccionada esta opción, seleccione todos los nodos que desea mover y arrástrelos a la posición deseada. Recuerde que para seleccionar varios nodos debe de pinchar y arrastrar con el ratón sobre el tapiz para hacer una región de selección, tal y como se explicaba en el apartado Borrar nodos.

Cuando se mueven los nodos, también se redibujarán los arcos existentes.

De esta manera tendremos el grafo totalmente definido y es aquí donde empieza la parte de resolución según el tipo de heurística que queramos aplicar. El paso siguiente es elegir el tipo de algoritmo que queremos que nos solucione. Esto se eligen en la última fila de los iconos visibles en el programa.

5. Bibliografía:

- Página web del software: http://personales.upv.es/~arodrigu/grafos/
- Apuntes y libro de la asignatura de Métodos Cuantitativos del segundo ciclo de Ingeniería en Organización Industrial de la ETSEIAT.
- Conceptos y material sobre los análisis de árboles (Dijkstra):
  - http://www.alumnos.unican.es/uc900/Algoritmo.htm
  - http://neo.lcc.uma.es/evirtual/cdd/applets/distancia%20corta/Example2.html
  - http://www.it.uc3m.es/~pablo/asignaturas/rysc1/alumnos/04-EncEjerciciosSolucion.pdf
- Conceptos y material sobre los análisis de árboles (Prim):
  - http://www.dma.fi.upm.es/gregorio/grafos/PrimKruskal/prim/prim.html
  - http://decsai.ugr.es/~ta_ii/algoritmos/seccion.php?id=teoria&subseccion=applets/prim
  - http://tracer.lsi.upc.es/kp/SimpleAlgorithmsPage.htm
  - decsai.ugr.es/~verdegay/tema-08.pdf
- Conceptos y material sobre los análisis de árboles (Kruskal):
  - http://www.dma.fi.upm.es/gregorio/grafos/PrimKruskal/kruskal/kruskal.html
  - http://lear.inforg.uniovi.es/ioperativa/TutorialGrafos/minimaexp/minimaexp.htm
  - http://www.dsic.upv.es/asignaturas/facultad/eda/practicas/pr6/practica6.pdf
- Conceptos y material sobre flujos (Ford-Furkelson):
  - http://www.dsic.upv.es/asignaturas/facultad/eda/concurso/concuOld/0506/concurso0506web/node6.html
  - http://www.cidse.itcr.ac.cr/revistamate/Contribucionesv3n2002/horario/pag4.html
  - http://www.cs.princeton.edu/~wayne/kleinberg-tardos/07demo-maxflow.ppt
- Conceptos y material sobre flujos (coste mínimo):
  - http://webdelprofesor.ula.ve/ingenieria/gbriceno/IO-B2004/costo%20minimo.pdf
  - http://mit.ocw.universia.net/15.053/s02/pdf/s02-lec11.pdf

- Conceptos y material sobre el VRP:
  - http://personales.upv.es/arodrigu/IDI/VRPXML.pdf
  - http://karnak.upc.es/~elena/Tutoriales/rutas.pdf
  - http://ciruelo.uninorte.edu.co/pdf/ingenieria_desarrollo/13/planeacion_del_transporte_y_enrutamiento_de_vehiculos.pdf
  - http://www.fing.edu.uy/inco/cursos/geneticos/ae/2007/Clases/clase1.pdf
  - www.fing.edu.uy/inco/grupos/invop/mh/material/presentG3-GRASP
  - http://personales.upv.es/arodrigu/Grafos/MTSP.htm
  - http://karnak.upc.es/~elena/Tutoriales/rutas

6. Licencia del software:

Aquí se describe el tipo de licencia de Grafos y de todo el material contenido en este sitio web. Por favor, lea atentamente los términos de la licencia.

El software Grafos y todo el contenido (documentos, imágenes, ficheros, textos, programas, etc.). alojado en esta web (dominio: /~arodrigu/grafos/,,,) se distribuye bajo las condiciones: Reconocimiento-NoComercial-CompartirIgual 3.0. (Creative Commons License) a menos que se indique lo contrario o sea propiedad de otro autor. Si bien está permitida su copia libre y distribución gratuita, no está permitida su comercialización (bajo ningún soporte y condición) sin el consentimiento escrito del autor. Además debe citar al autor de todo el material aquí expuesto. Este programa está protegido por las leyes de derechos de autor y otros tratados internacionales. Así que la distribución o comercialización ilícita de este programa o de cualquier parte del mismo y de este sitio web (incumpliendo las condiciones anteriores), pueden dar lugar a responsabilidades civiles y criminales, que serán perseguidas.

7. Índice de ilustraciones: