Parte 2 Análisis de algoritmos Voraces

El Problema del viajante

El Problema del Viajante de Comercio (TSP) es uno de los problemas de optimización combinatoria más conocidos. En su formulación más general, dadas una serie de ciudades, el objetivo consiste en encontrar el circuito de menor coste que parta de una ciudad concreta, pase por todas las demás una sola vez y retorne a la ciudad de origen, es decir, calcular la ruta más corta posible que visita cada ciudad exactamente una vez y al finalizar regresa a la ciudad origen.

Para el caso de n ciudades, existen n! rutas posibles, aunque se puede simplificar ya que dada una ruta nos da igual el punto de partida y esto reduce el número de rutas a examinar en un factor n quedando (n-1)! rutas posibles.

En la práctica, para un problema del viajante con 10 ciudades hay (10-1)! = 362.880 rutas diferentes y apenas aumentamos el número de ciudades las posibilidades crecen factorialmente haciendo inviable encontrar la solución óptima probando todas las posibilidades (para 20 ciudades hay más de $1 \cdot 10^{17}$ rutas posibles. Un ordenador que calcule un millón de rutas por segundo necesitaría 3.817 años para resolverlo. Con 25 ciudades, al haber $6 \cdot 10^{23}$ rutas posibles se tardarían 19.674 millones de años en probar todas las posibilidades).

Por cada ciudad nueva que incorporemos, el número de rutas se multiplica por el factor n y crece factorialmente. Por ello el problema pertenece a la clase de problemas NP-completos.

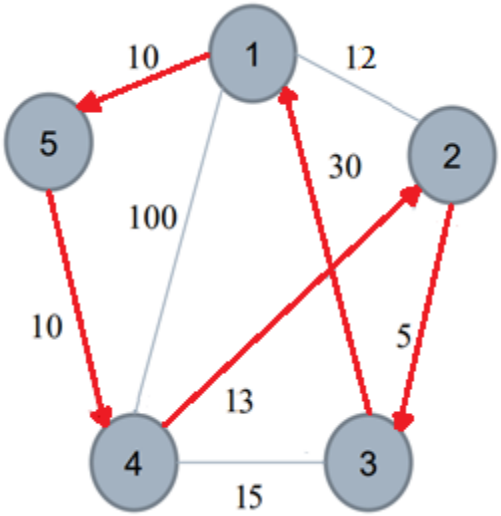
Dado que es muy difícil (por no decir imposible) encontrar la solución óptima, para este tipo de problemas se suelen emplear técnicas que den soluciones aceptables en un tiempo computacionalmente prudencial.

Primera Aproximación: Estrategia de búsqueda voraz (unidireccional)

Una primera solución es realizar una búsqueda voraz considerando como criterio voraz escoger en cada momento la ciudad más cercana a la última añadida al circuito (siempre que no haya sido visitada antes): partimos de una ciudad elegida aleatoriamente y nos desplazamos a la ciudad más cercana a ella, una vez en ella volvemos a desplazarnos a la ciudad (que aún no haya sido visitada) más cercana a ésta última y repetimos el proceso hasta visitar todas las ciudades.

El programa resultante es corto, fácil de codificar y muy rápido y por regla general, para n ciudades aleatoriamente distribuidas el algoritmo retorna un camino que es un 25% más largo que el óptimo.

A modo de ejemplo, en el siguiente grafo (los círculos representan ciudades y los arcos la distancia entre las ciudades) hemos resaltado en color rojo la solución que devolvería el algoritmo voraz planteado, partiendo de la ciudad 1.

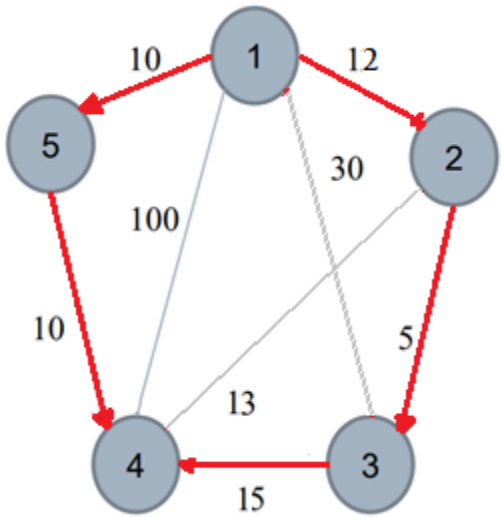


Como podemos comprobar, la ruta devuelta por el algoritmo voraz (1,5,4,2,3,1) tiene un coste de 10+10+13+5+30 = 68 mientras que la óptima es la ruta (1,2,3,4,5,1) cuyo coste es de 12+5+15+10+10 = 52.

Segunda Aproximación: Estrategia de búsqueda voraz (bidireccional)

Una segunda solución es realizar una búsqueda voraz considerando como criterio voraz escoger en cada momento la ciudad (que no haya sido visitada) más cercana a uno de los extremos del camino que llevamos recorrido: partimos de una ciudad elegida aleatoriamente y la unimos a la ciudad más cercana a ella. A continuación, elegimos la ciudad más cercana (que aún no haya sido visitada) de cada extremo y seleccionamos el extremo con distancia más corta y repetimos el proceso hasta visitar todas las ciudades.

A modo de ejemplo, en el mismo grafo anterior hemos resaltado en color rojo la solución que devolvería el algoritmo con este criterio planteado, partiendo de la ciudad 1.



Como podemos comprobar, la ruta devuelta por este algoritmos voraz (1,2,3,4,5,1) cuyo coste es de 12+5+15+10+10 = 52 y coincide (por casualidad) con el óptimo, es mejor que el devuelto por el algoritmo voraz anterior (1,5,4,2,5,1) que tenía un coste de 10+10+13+5+30 = 68.

Desarrollo de la práctica

El alumno deberá desarrollar un proyecto en lenguaje JAVA que **resuelva el problema del viajante siguiendo las dos estrategias voraces comentadas**.

El programa desarrollado deberá permitir seleccionar como entrada un conjunto de datos generados de forma aleatoria, o bien un conjunto de datos importados mediante la carga de cualquier fichero de la biblioteca TSPLIB. Para ello el programa debe proporcionar una opción de menú o una ventana de diálogo que permita al usuario indicar la ruta del fichero que desea abrir.

Nota: Al facilitar sólo el conjunto de puntos, el programa deberá generar todas los posibles caminos (conectando cada punto con todos los demás), asociando a cada arista como peso la distancia euclídea.

El programa deberá calcular cual es el coste de la ruta calculada (suma del coste de los caminos desde la ciudad origen al resto de ciudades) con ambas estrategias voraces y deberá mostrar por pantalla, para cada estrategia, el recorrido de dicha ruta, permitiendo guardar en disco un fichero (uno con cada estrategia) con la solución obtenida.

El formato del fichero generado por la aplicación debe ser el siguiente (se muestra el fichero solución del algoritmo voraz unidireccional para el grafo de ejemplo de la página anterior):

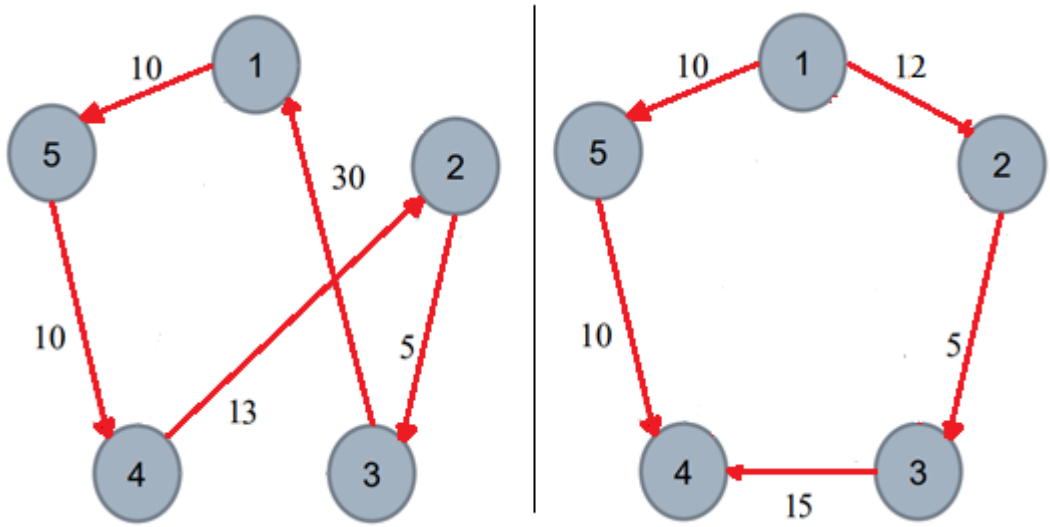
```
NAME : huelva4.opt.tour
TYPE : TOUR
DIMENSION : 5
SOLUTION: 68
TOUR_SECTION
1,5,4,2,3,1
10 - 1,5
10 - 5,4
13 - 4,2
 5 - 2,3
30 - 3,1
EOF
```

Deseamos comprobar empíricamente si una estrategia es mejor que otra o ambas son equivalentes, por lo que el programa deberá comparar ambas estrategias con diferentes tamaños de datos generados de forma aleatoria (estudio experimental realizado sobre grupos aleatorios de Puntos de tamaños 500, 1.000, 1.500, …, 5.000, realizándose para cada una de las tallas indicadas **100 ejecuciones** diferentes). Se debe devolver cuantas veces una estrategia obtiene mejor resultado que la otra y el tiempo medio que tarda en ejecutarse.

Opcional (2 puntos):

El programa deberá realizarse en modo gráfico (mediante un formulario con botones de comandos, etiquetas, cuadros de texto, ventanas de diálogo, etc,) **y mostrar la solución del problema** (la ruta que empieza y acaba en un mismo Punto pasando una única vez por todos los demás) **en formato gráfico por medio de un panel en el que se representen los puntos del dataset y los enlaces que forman la ruta calculada.**

A modo de ejemplo, se muestra la solución en formato gráfico de ambas estrategias para el grafo de ejemplo anterior.



Opcional (2 puntos):

Implementar un algoritmo que calcule la solución óptima probando todas las combinaciones posibles (el algoritmo deberá generar todas las permutaciones posibles para el conjunto de entrada dado).

El algoritmo sólo podrá aceptar como entrada conjuntos con un máximo de 10-12 Puntos, ya que para tamaños superiores la cantidad de combinaciones posibles hace que el algoritmo tarde un tiempo inviable.

Documentación a entregar

Para esta segunda parte de la práctica 1, el alumno NO TIENE que realizar ningún estudio teórico, aunque deberá entregar en la documentación una tabla resumen con los resultados obtenidos por ambas estrategias sobre los datasets proporcionados (**berlin52, ch130, ch150, d493 y d657).**

Además, en dicha documentación deberá mostrar el resultado de comparar los resultados obtenidos por ambas estrategias en el estudio experimental descrito en la página 16, devolviendo, para cada talla cuantas veces una estrategia obtiene mejor resultado que la otra y el tiempo medio que tarda en ejecutarse.

Los resultados deberán mostrarse en una tabla y de forma gráfica (una gráfica para el tiempo de ejecución y otra para mostrar cuantas veces una estrategia es mejor que la otra).