Práctica 2 MARP
Problema del viajante (TSP)

Autor: Javier Cortés Tejada

1. Introducción

La práctica desarrollada se trata de un algoritmo resolutor para el problema del viajante (TSP). Este ha sido implementado usando un MVC junto con observadores para el correcto paso de mensajes entre objetos. Además, la parte gráfica ha sido desarrollada usando el API *Processing* dado su fácil intgración con el entorno de desarrollo Eclipse y los buenos resultados que aporta, pues es muy intuitiva y fácil de usar

2. Guía sobre la práctica

Los ficheros entregados se componen de un archivo *core.jar* que debe añadirse al path para poder ejecutar la práctica, los ficheros *txt* para la realización de pruebas y una carpeta *src* con todo el codigo desarrollado para la práctica. En ella encontramos varios fichero, aunque toda la lógica de la práctica como tal (el algoritmo) está en el fichero *TSP.java*, el cual a diferencia del resto del código ha sido más comentado dado que era lo solicitado en la entrega.

En cuento a la ejecución de la práctica, tenemos que irnos a la clase *Main.java* y lanzar el programa. Nada más hacerlo se nos abrirá una pequeña ventana donde seleccionaremos el tipo de estimación a aplicar, una *naive* o más débil y otra más costosa a nivel computacional pero que estima bastante mejor, además de dos botones, uno de ellos nos permitirá cargar un fichero de texto y el otro lanzará la ejecución.

Una vez hayamos pulsado el botón *run* esa ventana se cerrará y se abrirá otra a pantalla completa donde tendremos dos representaciones de la matriz, una en forma de grafo donde las soluciones que vaya computando el algoritmo durante su ejecución serán mostradas, una matriz de adyacencia para tener constancia de los costes de cada arista y una pequeña consola en la parte inferior derecha donde se mostrarán los resultados de ejecución pedidos en la práctica.

3. Notas importante

- Cabe destacar que la práctica necesita el archivo *core.jar*, luego para su funcionamiento es imprescindible agregarlo al proyecto (este *jar* se proporciona junto con la entrega).
- La aplicación ha sido desarrollada tomando las dimensiones de una pantalla estándart de resolución 16 : 9 luego para ver la práctica correctamente se recomienda un monitor de estas características, aunque esto no influye para nada en el funcionamiento del algoritmo.
- En la pantalla donde se pide al usuario el archivo y el tipo de estimación, si el usuario no modifica ningún campo y lanza la ejecución, se aplicará una estimación débil sobre los datos del fichero *matrix17.txt*, luego es necesario que este fichero esté al mismo nivel que la carpeta *src* del proyecto, ya que se toma la ruta relativa de dicho fichero. En caso de no estar dicho archivo en el lugar indicado, la ejecución podría acabar en estado de fallo (este fichero se proporciona en la entrega).
- Los ficheros entregados tienen matrices de adyacencia de tamaños 15, 16 y 17. Dichos ficheros proporcionan ejecuciones con solución. En caso de que se quieran hacer más pruebas con otros archivos, en el fichero *Main.java* se ha incluido un segundo *main* comentado, el cual permite generar ficheros con matrices de adyacencia modificando el tamaño de las mismas.
- Dado el problema que se ha desarrollado, no se asegura que haya una solución para cualquier fichero de entrada, de manera que en caso de no haber un camino de coste mínimo la solución mostrada por pantalla constará de un vector de ceros indicando que no hay solución.

4. Fuentes de recursos

En cuanto al desarrollo de la parte gráfica, he usado la documentación de la siguiente fuente https://processing.org/, y para comprobar la optimalidad de mi solución he comparado los resultados obtenidos con los proporcionados por herramientas específicamente creadas para este problema, las cuales se pueden consultar en http://tspsg.info/.

5. Conclusiones

Tras la realización de la práctica, he sacado en claro la importancia de hacer estimaciones correctas y precisas en este tipo de programación, ya que en ciertos casos he llegado a obtener una disminución del 30 % de los nodos tras pasar de una estimación débil a una más fuerte. También he podido observar que las técnicas de ramificación y poda admiten (al menos en este caso) un número de nodos bajo para las ejecuciones debido a la complejidad computacional del problema, por tanto, creo que hubiese sido más interesante haber aplicado otro tipo de algoritmos bien genéticos o heurísticos con el fin de poder trabajar con un volumen de datos mayor, aunque no siempre proporcionen soluciones óptimas.