TSP: El problema del viajante de comercio ampliado – Metodología de la Programación

Valoración personal

Óscar Bermúdez Garrido
Francisco David Charte Luque

Para esta fase del desarrollo de la solución para el problema del viajante de comercio, ya contábamos con la base de lo trabajado en la práctica anterior, es decir, la estructura de las clases y los métodos básicos para lidiar con el problema. Además, se añadieron a las clases los destructores, para tratar la memoria dinámica reservada que debiera liberarse en el momento de la eliminación de un objeto, y se añadieron algunos métodos auxiliares (como es el caso de `insertar` en la clase `Recorrido`). Asimismo, se modificó la forma de llamar a algunos de los métodos por operadores, para mejorar la comprensión del código en las clases que trataran con objetos de `Recorrido` y `Problema`. También se alteró la función `main` para ajustarse a la tarea requerida.

La implementación de la heurística de inserción contó con dos partes: por un lado se debía escoger un criterio para la selección de las tres ciudades que formarían el triángulo inicial. Por otra parte se debía programar de forma eficiente un método iterativo que aumentara el número de ciudades de la forma menos costosa posible. Para la primera fase se estudió el comportamiento de varios criterios, entre ellos la elección de las ciudades más lejanas entre sí y la elección de las ciudades más al noroeste, noreste y sur. Finalmente se optó por la que daba mejores y más estables resultados, que era la elección de las ciudades más al oeste, al este y al norte.

Por otro lado, para la heurística propia, se experimentó con algunas heurísticas básicas de tipo evolutivo (generando caminos aleatorios y mejorándolos con "mutaciones" aleatorias), pero tras comprobar que nuestras implementaciones no daban buenos resultados, se decidió definir una heurística basada en los datos numéricos directos (las coordenadas) de forma que fuera muy rápida y generase un camino adecuado en el menor tiempo posible, ya que no habría que lidiar con distancias y demasiadas iteraciones. Se fue refinando esta idea hasta que se encontró una operación que, al realizarla a las coordenadas X, Y de un nodo, permitía ordenarlas generalmente en un camino bastante eficaz. Sin embargo, el comportamiento de esta heurística depende mucho de la distribución de las ciudades en la instancia, por ejemplo da malos resultados en `KROA200`.

En la siguiente tabla resumimos los datos extraídos de las soluciones generadas por cada heurística en cada instancia del TSP proporcionada. Se muestra el coste y el tiempo de ejecución (en segundos) en cada caso. Se señalan en verde los menores costes. Los gráficos asociados a cada solución se pueden consultar en el archivo `tablas.html` (generado por el guion `gentables.sh`).

Mapas	Vecino más cercano		Inserción más económica		Comparación de coordenadas	
small10	3256.13	0.005736	2839.94	0.005458	2826.5	0.006946
berlin52	8182.19	0.010284	9724.34	0.027486	15483.3	0.020499
eil101	736.368	0.017552	761.294	0.168332	1876.92	0.046158
KROA200	34547.7	0.083469	35213.7	1.94008	214857	0.143186
a280	3094.28	0.180689	3166.81	6.84771	2818.62	0.245479
pr1002	312237	7.89942	294606	964.007	349438	2.57201

Análisis de los resultados

Después de pruebas descartadas y modificaciones que mejoraron las heurísticas desarrolladas, tenemos tres heurísticas que tienen el comportamiento descrito a continuación:

- *Vecino más cercano.* La primera heurística implementada, se basa en una idea sencilla: pasar de cada ciudad a la más cercana no visitada. Además, se realiza un circuito de vecino más cercano comenzando por cada ciudad y se elige el mejor. La implementación es bastante eficiente y permite generar una solución adecuada en un tiempo razonable. De hecho, esta heurística es la que da mejores resultados en tres de las seis instancias del problema proporcionadas.

- *Inserción más económica.* La heurística de inserción más económica consta de dos fases diferenciadas: la elección del triángulo inicial sobre el cual se añadirán más ciudades y la inserción en sí, que trata de agregar ciudades al recorrido de la forma en que se aumente el coste lo menos posible. En este caso, la heurística da buenos resultados, mejorando en dos instancias (`small10` y `pr1002`) a *Vecino más cercano* y en las demás situándose en un coste muy cercano. Sin embargo, es la heurística más lenta en general, y sobre todo en instancias grandes como `pr1002`.

- *Comparación de coordenadas.* La heurística que hemos desarrollado parte de una idea simple: operar con las coordenadas de cada nodo/ciudad de forma que se puedan ordenar en dos caminos, uno de ida (que parte de un nodo escogido) y uno de vuelta. En la mayoría de los casos, la operación permite ordenar los nodos adecuadamente. Concretamente, genera mejores recorridos que las otras dos heurísticas en las instancias `small10` y `a280`. Sin embargo, en ocasiones, como en la instancia `KROA200`, genera un camino poco eficaz, alcanzando un coste mucho mayor que las otras. Por otro lado, es claro que para mapas muy grandes, como es el caso de `pr1002`, es la heurística más rápida, generando una solución en 2 segundos y medio, frente a los casi 8 segundos de *Vecino más cercano* y los 16 minutos de *Inserción más económica*.

En conclusión, tenemos que *Inserción más económica* es una heurística que genera resultados similares a *Vecino más cercano*, pero es algo más lenta. Además, la heurística de *Comparación de coordenadas* es algo errática, produce muy buenos resultados en poco tiempo o bien caminos que se alejan mucho de la solución óptima.

Creemos que la solución implementada para cada heurística es correcta, si bien se podría haber tratado de mejorar la eficiencia mediante algunos cambios, por ejemplo, si se gestionaran los índices de las ciudades en lugar de punteros a las mismas. También nos preocupa que la implementación de la clase Recorrido, a la que se añadieron varias sobrecargas de operadores, pueda ser confusa y se repita código. De hecho, las clases más básicas (`Ciudad` y `Recorrido`) en las que se sostienen las demás deberían ser las que tuvieran una implementación más cuidada y eficiente, y se ha tratado de hacer de esta forma, pero aún se podrían mejorar. Aún así, en general nuestro programa produce soluciones adecuadas al problema en tiempos razonables (puede que *Inserción más económica* tarde demasiado en `pr1002`), por lo que estamos satisfechos con nuestro trabajo.