



Tópicos Avanzados en Optimización Combinatoria y Teoría de Grafos

Docentes: Graciela Nasini, Daniel Severin, Paola Tolomei, Pablo Torres

TRABAJO PRÁCTICO: HEURÍSTICAS Y COTAS INFERIORES PARA EL PROBLEMA DEL VIAJANTE DE COMERCIO (TSP)

En este Trabajo Práctico evaluamos el comportamiento promedio respecto al valor óptimo del TSP de las heurísticas del *Vecino más cercano*, las heurísticas de *Inserción*, un algoritmo 2-aproximado y la cota inferior dada por un *1-árbol*, todas ellas vistas en el curso.

El estudio lo realizamos sobre 10 instancias de la biblioteca TSPLIB [TSPLIB] y 10 instancias aleatorias. Los valores óptimos para estas 20 instancias se deben obtener utilizando el software *Concorde* [CONC] y las soluciones brindadas por las heurísticas y la cota inferior dada por el 1-árbol se deben obtener con las siguientes herramientas en el entorno *Scilab* [SCILAB]:

- Herramientas desarrolladas por la cátedra:
 - `cargador.sce`: Carga el resto de las herramientas en la memoria de Scilab.
 - `lee_TSP.sci`: Lee un archivo con una instancia TSP y genera la matriz de distancias.
 - `vecino_mas_cercano.sci`: Dada la matriz de distancias de una instancia TSP y un vértice inicial, genera un tour y su valor aplicando la heurística del Vecino más cercano.
 - `insercion_mas_lejana.sci`: Dada la matriz de distancias de una instancia TSP, genera un tour y su valor aplicando la heurística de la Inserción más lejana.
 - `dibuja_tour.sci`: Muestra en pantalla la gráfica de un tour.
 - `genera_grafo_costos.sci`: Dada la matriz de distancias de una instancia TSP, genera un vector con las aristas del grafo y otro vector con los costos.
 - `kruskal.sci`: Dado un vector con las aristas del grafo y otro vector con los costos, genera un vector con las aristas del árbol de expansión mínimo y su valor.
- Herramientas a ser desarrolladas como parte del trabajo práctico:
 - `vecino_mas_cercano_mejorado.sci`: Dada la matriz de distancias de una instancia TSP, genera un tour y su valor con la heurística “mejorada” del Vecino más cercano detallada luego.
 - `insercion_mas_cercana.sci`: Dada la matriz de distancias de una instancia TSP, genera un tour y su valor con la heurística de la Inserción más cercana.
 - `alg2aprox.sci`: Dada la matriz de distancias de una instancia TSP, genera un tour y su valor con un algoritmo 2-aproximado.
 - `uno_arbol.sci`: Dada la matriz de distancias de una instancia TSP y un vértice inicial, genera una cota inferior del valor óptimo con el algoritmo 1-árbol.
 - `held_karp.sce`: Script que mejora la cota 1-árbol de una instancia.

Atención: la carga/optimización de las instancias más grandes suelen tardar un ratito, sugerimos comenzar con las instancias más pequeñas (por ejemplo, `berlin52` que tiene 52 nodos).

Pasos a seguir:

1. Cree en su máquina un directorio de trabajo. Descargue (de la página de la materia) y descomprima el archivo `TP.zip` en dicho directorio (contiene 10 instancias de TSPLIB y archivos de Scilab).
2. Ejecute el programa `Instalador-concorde1.1.exe` (también en la pág. de la materia) para instalar *Concorde* (utilice la configuración por defecto, eligiendo *Next* en todas las ventanas). Este instalador puede ser borrado luego.
3. Generación de instancias aleatorias y cálculo de valores óptimos, usando *Concorde*:
 - Cree 10 instancias aleatorias de 100 nodos utilizando los comandos de Concorde: *File* → *New*, *Edit* → *Add Random Nodes* y *File* → *Save*. Guarde las instancias en el directorio de trabajo, con extensión `tsp`.
 - Resuelva las 10 instancias de TSPLIB y las 10 instancias aleatorias generadas mediante el comando *Solve* con los parámetros por defecto. Tome nota de los valores óptimos obtenidos al final de la optimización de cada instancia. Por ejemplo, la instancia `att532` debería dar 86729.
4. Para cada instancia aleatoria generada, abra dicho archivo con un editor de texto (por ejemplo, *WordPad*), agregue una línea con la palabra `EOF` al final del archivo y guárdelo. El fin de esto es que las instancias aleatorias generadas puedan ser luego reconocidas por el lector de TSP de *Scilab*.
5. Abra *Scilab* y, en el Navegador de Archivos, configure su directorio de trabajo. Luego ejecute¹ `cargador.sce` (éste carga en memoria el resto de los programas).
6. Comportamiento de la heurística *Vecino más cercano*:

- Para cargar una instancia, escriba el siguiente comando en la consola:

```
[A, coordenadas] = lee_TSP('instancia.tsp');
```

- Para una instancia de n nodos, la matriz `A` tiene dimensión $n \times n$, contiene la distancia euclídea entre cada par de nodos, es simétrica y con valores ∞ en su diagonal. La matriz `coordenadas` tiene dimensión $2 \times n$ y contiene las coordenadas de cada vértice en el plano.
- El siguiente comando ejecuta la heurística sobre la instancia TSP partiendo del vértice inicial v_1 :

```
[tour, valor] = vecino_mas_cercano(A, v1)
```

- En `tour` se obtiene un vector que indica cómo deben ser recorridos los vértices, y en `valor` se obtiene el valor de dicho tour. Si se desea ver una gráfica del tour, se puede ejecutar:

```
dibuja_tour(tour, coordenadas)
```

- Teniendo en cuenta lo anterior, ejecute la heurística del Vecino más cercano sobre las 20 instancias, utilizando como vértice inicial $v_1 = 1$. Tome nota de los valores de los tours generados.

¹En Scilab existen dos maneras de ejecutar un archivo, una es apretando el botón derecho del mouse sobre el archivo y eligiendo el comando *Ejecutar*; la otra es escribiendo en la consola: `exec('archivo')`

7. Calcule el promedio en las 10 instancias de TSPLIB del cociente *valor heurística/valor óptimo*. Repita lo mismo con las 10 instancias aleatorias.
8. Grafique el tour óptimo y el tour de la heurística para aquella instancia donde la heurística se comportó peor (es decir, donde maximizó el cociente del ítem anterior).
9. Mejorando la heurística *Vecino más cercano*:
 - Implemente una función `vecino_mas_cercano_mejorado.sci` que toma una instancia TSP y calcula la heurística del Vecino más Cercano con $v_1 = v$ para cada nodo v del grafo, y luego devuelve el tour de menor valor.
 - Cada vez que modifique su función, recuerde volver a cargarla ejecutando `cargador.sce`.
 - Pruebe su función sobre instancias pequeñas generadas ad-hoc. Sugerimos usar la del Ejercicio 9 de la Práctica 1.
 - Calcule el comportamiento promedio sobre las 10 instancias de TSPLIB y sus 10 instancias aleatorias (es decir, repita el experimento del ítem 7).
10. Comportamiento de las heurísticas *de Insercion*:
 - Implemente una función `insercion_mas_cercana.sci` que, a partir de la matriz de distancias de una instancia TSP, genera un tour y su valor con la heurística de la Inserción más cercana. Sugerimos que se base en la función `insercion_mas_lejana.sci` brindada por la cátedra (con el fin de asegurarse que funciona, pruébela sobre instancias pequeñas generadas ad-hoc).
 - Calcule el comportamiento promedio de las heurísticas de Inserción más cercana y más lejana sobre ambos conjuntos de instancias.
11. Comportamiento de un algoritmo de aproximación con factor 2:
 - Implemente una función `alg2aprox.sci` que, a partir de la matriz de distancias de una instancia TSP, genera un tour y su valor de la siguiente manera:
 - Computar un árbol de expansión mínimo T (la siguiente sección explica cómo hacerlo).
 - Considerar un multigrafo \overline{G} que contiene los mismos vértices que G y cada arista de T está duplicada (observe que \overline{G} es euleriano y todo vértice tiene grado par).
 - Obtener un circuito euleriano C de \overline{G} . Luego, recorrer el circuito y cada vez que pasa por un vértice ya visitado saltarlo y pasar al siguiente (el concepto de *atajo* visto en teoría).
 - Calcule el comportamiento promedio del algoritmo sobre ambos conjuntos de instancias. Compare con los valores reportados para el Algoritmo de Christofides [LCO, pág. 246].
 - Pruebe que el algoritmo dado es de aproximación con factor 2. Es decir que, dado un tour óptimo H^* , el algoritmo genera un tour H tal que $c(H) \leq 2c(H^*)$.
12. Comportamiento de la cota dada por un 1-árbol óptimo:
 - Implemente una función `uno_arbol.sci` que, a partir de la matriz de distancias de una instancia TSP y un vértice inicial v_1 , genera una cota inferior del valor óptimo de dicha instancia con el algoritmo explicado en [LCO, pág. 253]. Para calcular el valor del árbol de expansión mínimo de un grafo cuya matriz de distancias es A puede realizar:

```
[G, costos] = genera_grafo_costos(A);  
[F, valor] = kruskal(G, costos);
```

- Calcule el comportamiento promedio de la heurística 1-árbol tomando $v_1 = 1$ sobre ambos conjuntos de instancias.
 - Explique el ejemplo mostrado en la pág. 254 de LCO (de qué forma se puede mejorar una solución dada por 1-árbol).
 - Escoja la instancia aleatoria donde la cota 1-árbol se comportó peor. Luego implemente un script `held_karp.sce` que lee dicha instancia, modifica sus costos (vea recuadro de [LCO, pág. 255]), aplica `uno_arbol` y obtiene una cota dual. Intente obtener una cota dual mejor que la original dada por 1-árbol y repórtela.
13. Escriba un **informe** que contenga el desarrollo del trabajo, incluyendo dos tablas: una cuyas columnas reporten los resultados generados por cada heurística y cuyas filas correspondan a cada instancia de TSPLIB, más una fila al final con los promedios; la otra tabla con la misma información pero sobre las instancias aleatorias. El informe luego debe analizar los valores de los promedios y compararlos con los valores reportados en [LCO, Cap. 7], y debe finalizar con los gráficos realizados en el ítem 8 y los códigos fuente de las implementaciones.

Referencias:

[LCO] W. Cook, W. Cunningham, W. Pulleyblank A. Schrijver. *Combinatorial Optimization*. Wiley-Interscience.

Para más información, puede consultar en:

[SCILAB] <http://www.scilab.org>

[CONC] <http://www.math.uwaterloo.ca/tsp/concorde>

[TSPLIB] <http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html>