Resolviendo el problema del TSP con Recocido Simulado

Víctor Zamora Gutiérrez

El recocido simulado es una heurística de optimización combinatoria que sirve para resolver problemas NP duros. En esta ocasión, utilizamos dicha heurística para resolver un caso modificado del Traveling Salesman Problem (TSP): dado un conjunto de ciudades, encontrar el camino de menor peso que pase por todas las ciudades exactamente una vez.

1. Para correr el programa

Para correr el programa, se necesita una distribución de Linux con java 8, ant y sqlite3. Para instalar dichos programas desde ArchLinux, ejecutar:

```
$sudo pacman -S jre8-openjdk
$sudo pacman -S jdk8-openjdk
$sudo pacman -S apache-ant
$sudo pacman -S sqlite
```

Teniendo esto, desde la carpeta raíz del proyecto (la que tiene el build.xml) ejecutar:

```
$ant tsp.jar
$java -jar tsp.jar <semilla> <ciudades>
```

Donde semilla es la semilla con la que se quiere correr el recocido y ciudades es un archivo que contiene una lista de ids de ciudades, separadas por una coma y un espacio. Por ejemplo, *ciudades* puede tener la siguiente línea:

1, 5, 9, 12, 16, 22, 23, 29, 30, 31, 39, 48, 52, 56, 58, 62, 65, 66, 70, 75, 80, 84, 86, 90, 92, 94, 95, 101, 107, 117, 119, 122, 133, 135, 143, 144, 146, 147, 150, 158, 159, 160, 166, 167, 176, 178, 179, 185, 186, 188, 190, 191, 194, 198, 200, 203, 207, 209, 213, 215, 216, 220, 221, 224, 227, 232, 233, 235, 238, 241, 244, 248, 250, 254, 264, 266, 274, 276

Alternativamente, en la carpeta raíz se encuentra el jar, por lo que teniendo java puede correrse el programa sin pasar por la compilación.

Para correr las pruebas unitarias, utilizar:

```
$ant test
```

También desde la carpeta raíz.

Y para generar la documentación de *javadoc*:

```
$ant doc
```

Ésta se generará en la carpeta *doc* del directorio raíz.

Por último, para borrar el .jar, la documentación y los archivos compilados:

```
$ant clean
```

2. Parámetros utilizados

Para el programa, utilicé los siguientes parámetros:

$\epsilon = 0{,}0001$

$\epsilon_p = 0{,}0001$

$\varphi = 0{,}9$

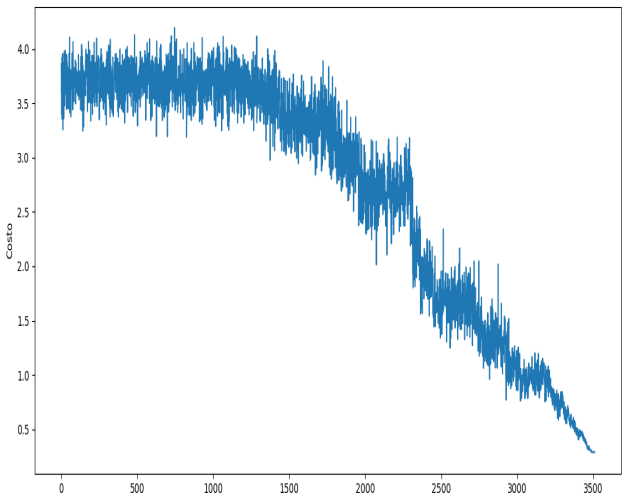
$C = 2$

Para más información sobre los parámetros, leer sobre la heurística en los documentos de Canek Pelaez Valdés.

3. Gráfica de soluciones

La siguiente gráfica muestra los costos de las soluciones aceptadas respecto al tiempo con el conjunto de entrada

1, 5, 9, 12, 16, 22, 23, 29, 30, 31, 39, 48, 52, 56, 58, 62, 65, 66, 70, 75, 80, 84, 86, 90, 92, 94, 95, 101, 107, 117, 119, 122, 133, 135, 143, 144, 146, 147, 150, 158, 159, 160, 166, 167, 176, 178, 179, 185, 186, 188, 190, 191, 194, 198, 200, 203, 207, 209, 213, 215, 216, 220, 221, 224, 227, 232, 233, 235, 238, 241, 244, 248, 250, 254, 264, 266, 274, 276



Esta fue la mejor solución obtenida con dicho conjunto. El costo final es de 0.28963322352405946 y la solución es factible. Se obtuvo con la semilla 72. El programa para generar estas gráficas se encuentra en la carpeta *graficas*. Está escrito en python y toma como entrada un archivo con el costo de una solución por línea con el formato

```
E: <costo>
```

4. Conclusiones

- Aunque la heurísitica no es difícil de programar en sí, pueden ocurrir varios errores durante el proceso, así que hay que hacerlo con cuidado.
- Me falta hacer más experimentación sobre los parámetros.
- El proyecto fue muy divertido.