Seminario
Heurísticas de optimización combinatoria
2017-2

Carlos Gerardo Acosta Hernández

Facultad de Ciencias UNAM
3/04/17

Como primer proyecto para el seminario de Heurísticas de optimización combinatoria, se implementó un sistema para resolver instancias del **Problema del Agente Viajero** [1] en su versión simétrica, con la variación de ser un camino Hamiltoniano sin ciclo, como se describe usualmente. Para lograr dicho cometido, utilizamos la heurística de **Recocido Simulado** [2].

1. Sistema

El sistema consta de los siguientes componentes:

- **Lenguaje de programación:** Scala 2.12.1
- **Sistema de construcción:** SBT (*Scala Build Tool*) 0.13.13
- **Documentación:** ScalaDoc 2.12.1
- **Graficación:** Gnuplot 5.0
- **Insumos:** Los insumos (datos de entrada) fueron proporcionados por el profesor mediante una base de datos relacional *SQL*. El sistema manejador de la base de datos utilizado es SQLite 3.16.2. El controlador del *SMBD* es una biblioteca para *Java* compatible con *Scala*, SQLite-JDBC 3.16.1.
- **Control de versiones:** Para mantener el control de versiones se utilizó Git 2.11.0 y el repositorio en línea se encuentra alojado en GitHub.

[1] Del inglés *Traveling Salesman Problem (TSP)*
[2] Del inglés *Simulated Annealing*

1.1. Estructura

El proyecto está organizado con la jerarquía de directorios que se especifica para proyectos de *SBT* (el sistema de construcción). El árbol desde el directorio raíz, debe verse como:

```
RecocidoSimulado-TSP/
|__ doc/
|__ lib/
|__ src/
|__ project/
|__ src/
|__ README.md
|__ build.sbt
|__ hi.db
|__ conf.txt
```

En la carpeta **doc**, se encuentra este documento que estás leyendo, y su código fuente en LaTeX; también un subdirectorio llamado **graficas/** donde se almacena la gráfica de la última ejecución, reproducible con el archivo de captura *graficas/gnuplot/***exec.txt**. Para volver a generarla es necesario ejecutar el programa *gnuplot* con *graficas/gnuplot/***commands.gp** de entrada.

En la carpeta de **lib**, se incluyen las bibliotecas externas al sistma de construcción y al lenguaje de programación de las que depende el proyecto. Sólo debe encontrarse por el momento el driver del *SMBD SQLite* para *Java*.

Todo el código fuente referido a la implementación de la heurística se encuentr en el directorio **src/**. Desde ahí es posible exploar el código. Es importante recalcar que se incluye una carpeta para código escrito en *Java*, sin embargo, se encontrará vacía, pues la implementación está realizada por completo con la sintáxis de *Scala*. Hay dos subdirectorios en dicha sección, pero lo retomaremos más adelante en diseño (Sección 1.2).

En **project** están definidas las dependencias del proyecto en un archivo de nombre **Dependencies.scala**, donde se definen las características del sistema de construcción (puede señalarse, por ejemplo, una versión específica de *SBT* que será descargada en cuanto se inicialice el proyecto por la versión actual que tenga el sistema). Asimismo pueden agregarse todas las dependencias que sean necesarias, por el momento sólo se define la de *ScalaTest* para pruebas unitarias.

Los archivos libres en la raíz se refieren a:

- *README.md* es el archivo escrito en *markdown* para la descripción del proyecto para el repositorio en línea.
- *build.sbt* es el archivo de configuración del sistema, provee un nombre al proyecto, define la versión del lenguaje a utilizar, un empaquetador y conecta las dependencias externas del proyecto.

- *hi.db* es el archivo de la base de datos que *SQLite* genera a partir de los insumos. Decidí incluirla para señalar dónde se espera que esté ubicada la base de datos en la ejecución del programa.
- *conf.txt* es un archivo de configuración para una ejecución de la heurística. Por defecto contiene la instancia de 78 ciudades presentada en clase y la configuración de semillas y parámetros necesarios para obtener la mejor solución encontrada en la experimentación con el sistema, puede cambiarse su contenido, siempre que se respete el formato (su sintáxis será especificada en la Sección 1.2). De cualquier forma, es la entrada del programa, por lo que podemos utilizar cualquier otro con la misma sintáxis.

1.2. Diseño

Para diseñar la heurística del Recocido Simulado, consideré implementar interfaces para hacer la implementación independiente del problema *NP-Duro* con el que se fuera a trabajar. Sin embargo, el uso de *traits* en *Scala* no es exactamente como las interfaces de *Java*, por lo que puede apreciarse en el código una dependencia importante con el *TSP*. De cualquier forma, es posible intercambiar varias implementaciones de los traits sin que el Recocido se vea afectado del todo en su lógica -la función de costo, la condición de terminación, entre otros.

Como se mencionaba en la sección anterior, la implementación está dividida en dos paquetes principales, subdirectorios del paquete raíz.

```
src/main/scala/
|__ hoc/
|__ autsp/
```

El paquete **hoc** contiene todos los *traits* del proyecto y el paquete **autsp**[3] contiene todas las implementaciones de dichos traits. A continuación una lista enumerada de la relación entre cada implementación y el *trait* que extiende.

1. CondicionDeTerminacion.scala
2. FitFun.scala
3. GeneradorVerificador.scala
4. Lote.scala
5. RecocidoSimulado.scala
6. Solucion.scala
7. Temperatura.scala

1. MaximoFallidos.scala
2. FuncionDeCosto.scala
3. GenVer.scala
4. Lote.scala
5. AceptacionPorUmbrales.scala
6. Camino.scala
7. Temperatura.scala

Para información más detallada de la implementación, favor de revisar *target/scala-2.12/api/***index.html** con la *API* generada por *ScalaDoc*.

[3]autsp se refiere a: Aceptación por umbrales TSP

Uso

Descarga del proyecto

El proyecto se encuetntra alojado en línea en *GitHub*, para descargarlo basta ejecutar *Git* en cualquier directorio del sistema de archivos.

```
$ git clone https://github.com/Pernath/RecocidoSimulado-TSP.git
```

Se creará un directorio con la estructura definida en la sección 1.2.

Compilación y ejecución

Para las operaciones siguientes se supone que el sistema operativo ya cuenta con el sistema de construcción *sbt* instalado.

En primer lugar llamamos al sistema de construcción, que buscará el archivo **build.sbt** y establecerá la ruta actual como la ruta del proyecto.

```
[user@host RecocidoSimulado-TSP]$ sbt
```

Esto nos llevará a un *prompt* desde el que podremos llamar los siguientes comandos de acuerdo a lo que necesitemos.

Para compilar el proyecto y verificar que todo está listo para las pruebas, ejecutamos

```
> compile
```

Posteriormente, será posible ejecutar el programa con el archivo de configuración como argumento.

```
> run conf.txt
```

Si deseamos guardar la ejecución para poder graficarla con *gnuplot* o algún otro programa que acepte el formato "<x> <y>" en cada línea, ejecutamos con un segundo argumento 'w'[4].

```
> run conf.txt w
```

Empaquetado

Una manera de empacar la implementación es ejecutar el siguiente comando:

```
> package
```

[4]Es muy importante señalar que si empaquetamos el proyecto y ejecutamos el *JAR*, esta función dejará de estar disponible, pues la ruta establecida para escribir la ejecución es dependiente de la estructura de directorios del sistema. No lo recomiendo y definitavemente pensaré en una alternativa, pero siempre se puede reproducir la ruta que realizará el programa para buscar el archivo **exec.txt**.

Sin embargo, este no incluirá las dependencias externas en el *JAR* generado bajo la ruta *target/scala-2.12/***AUTSP.jar**, por lo que no será portátil y dependerá siempre de la estructura de directorios del sistema.

Por fortuna, también incluí la opción para generar un *stand-alone JAR* que nos permitirá mover líbremente el ejecutable de *Java* y ejecutarlo desde cualquier directorio. Esto se logra con el siguiente comando y podrá encontrarse el ejecutable en la misma dirección mencionada anteriormente:

```
> assembly
```

JAR y requisitos

Cómo se había mencionado anteriormente, para poder ejecutar el sistema como producto de *assembly*, es necesario que mantengamos tanto la base de datos **hi.db** al mismo nivel de directorio y que conseve ese nombre, pues a diferencia de **conf.txt** no es un argumento del programa y siempre se referirá a este en sus ejecuciones.

```
[user@host scala-1.12]$ java -jar AUTSP.jar conf.txt
```

Archivo de configuración

El archivo de configuración por defecto que se incluye desde el repositorio en línea es **conf.txt**. Puede utilizarse cualquier otro archivo o modificar este para que el sistema produzca nuevos resultados, si la entrada es la misma, el sistema reproducirá su ejecución. El archivo debe contar con dos líneas únicamente, al menos sólo dos serán leídas.

En la primera, separadas por comas y sin espacios, deben ir las ciudades de la instancia de *TSP* con la que se trabajará:

```
[id_city_a],[id_city_b],...,[id_city_w]
```

Donde $a, b, w \leq 278$.

La segunda y última línea, contiene los parámetros de la ejecución, igualmente separados por comas sin espacios:

```
[semilla],[temp_inicial],[phi],[tamaño_lote],[epsilon],[epsilonP],[constante_costo]
```

2. Experimentación

Para la experimentación, utilicé la instancia de *TSP* que nos proporcionó el profesor de 78 ciudades. Aquí la lista de los id's de las ciudades incluídas en la instancia:

[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]

La experimentación se llevó a cabo en dos computadoras distintas:

1. Memoria RAM: 7.7GiB
Procesador: Intel Core i5 CPU @ 2.40GHz x 4

2. Memoria RAM: 3.9GiB
Procesador: AMD E-3 CPU @ 1.60GHz x 2

Se probaron alrededor de 1000 semillas distintas con resultados con de muy mala evaluación aunque factibles y 20 pruebas más, una vez encontrada la configuración de parámetros más favorable con evaluaciones a lo más de 0,5.

Parámetros elegidos

Los parámetros elegidos son los siguientes:

- Semilla: 14
- Temperatura inicial: 4
- φ: 0.9
- Tamaño de lote: 10,000
- ε: 0.001
- ε_p: 0.04
- C: 3

Mejor solución encontrada

A continuación se presenta una gráfica con de las soluciones aceptadas y su evaluación para la mejor configuración de parámetros.

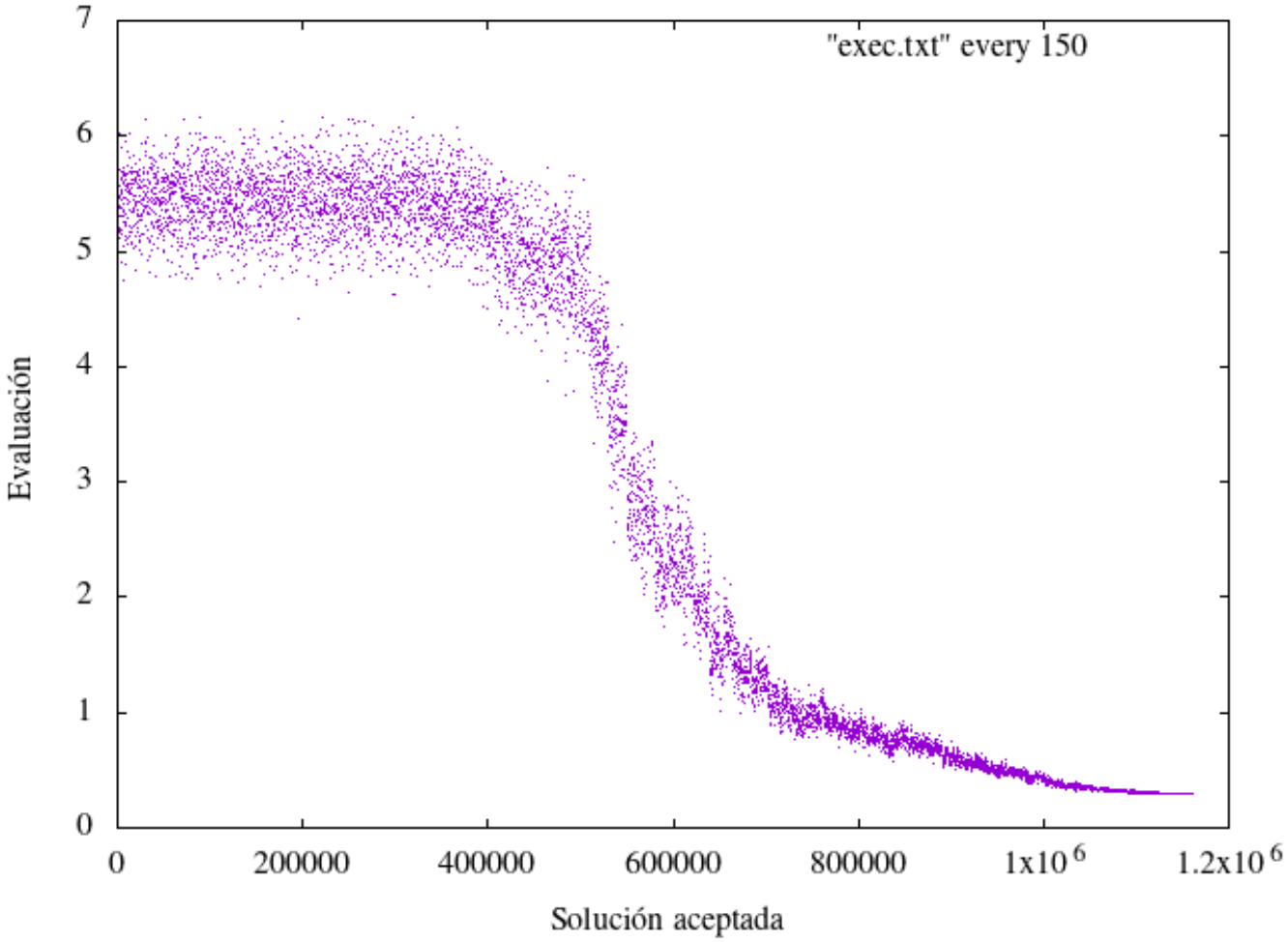

Figura 1: Gráfica de la mejor configuración encontrada.

El producto final de dicha corrida es un camino factible con evaluación de 0.28967526729032445:

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

3. Conclusiones

Aunque tardé bastante más en la experimentación de lo que tenía contemplado -provocando que el refinamiento del sistema no fuera el deseado- estoy bastante satisfecho con el resultado final obtenido. A decir verdad la experimentación que realicé en un principio no parecía ser suficiente por los resultados encontrados. Esto resultó ser bastante frustrante. No fue hasta que hice con detenimiento las pruebas, considerando los cambios que podía experimentar el sistema, que por fin di con una configuración apropiada y hubo oportunidad de probar un número significativamente menor que en las primeras pruebas (de 1000 a sólo 20). Aún con la mejora visible en la evaluación de los resultados (los exhibidos en la última clase eran de entre 0.7 a 1), y ahora contar con uno muy cercano a la mejor encontrada por el grupo, creo que es necesario continuar la experimentación de mi sistema y mejorar la interfaz del usuario.

Como primer proyecto de clase, creo que fue muy importante llevarlo a término para tener más herramientas y planificar con mayor eficiencia el siguiente. La oportunidad de implementar una nueva heurística, me proveerá de una más amplio panorama de elección para el nuevo proyecto.