Seminario de Heurísticas de Optimización Combinatoria
Recocido Simulado con Aceptación por Umbrales

Andrea Itzel González Vargas

Facultad de Ciencias UNAM

En este proyecto se implementó la heurística del recocido simulado con aceptación por umbrales para la resolución del problema del agente viajero. Se cuenta con una base de datos en donde se guardan las ciudades a tomar en cuenta junto con sus respectivas conexiones con otras ciudades.
Para implementar el proyecto se utilizó en lenguaje de programación Go junto con el manejador de bases de datos SQLite3.
El repositorio del proyecto está en `https://github.com/andreagonz/recocido`

Ejecución del programa

Para poder ejecutar el programa se adjunta la imagen de máquina virtual de QEMU manjaro.img. Lo único que se tiene que hacer es correr la máquina con

```
qemu-system-x86_64 -hda manjaro.img -m <memoria en bytes>
```

Si de algo sirve, la contraseña de root es "chepe". Una vez que se haya abierto la máquina, se deberá de ir al directorio ∼/go/src/github.com/andreagonz/recocido, donde se debe de correr go build, lo cual creará el archivo ejecutable recocido.
El comando para correr el programa es el siguiente:

```
./recocido <archivo.tsp> <params.txt> [ops]
```

donde <archivo.tsp> es el archivo con el conjunto de ciudades cuya ruta mínima quiere encontrarse. El formato de éste archivo es escribir los índices de las ciudades como están en la base de datos separados por una coma y un espacio. Por ejemplo:

```
26, 37, 14, 7, 1, 27, 31, 2, 33
```

<params.txt> es el archivo donde se especifica que parámetros se usarán para la ejecución. El formato del archivo es el siguiente:

```
{int: Semilla}
{int: Tamaño del lote}
{double: P}
```

{double: ϵ_p}
{double: ϵ_t}
{double: ϵ}
{double: ϕ}
{int: c}

donde {t: X} es la representación numérica de X, es decir de cada parámetro, si t es int sólo se aceptarán números enteros, si es double se permite también decimales. Ejemplo:

```
30
500
0.9
0.001
0.0001
0.001
0.9
5
```

Por último, [ops] son los parámetros opcionales del programa, hay dos de estos:

-g: Permite que se creé la gráfica de soluciones aceptadas. Se hará un archivo costos.txt y un archivo costos.png.

-m: Crea la representación en mapa de la ruta en el archivo mapa.html, donde se utiliza Google Maps.

Ejemplo de ejecución:

```
./recocido archivos/prueba.tsp archivos/params/1.txt -m -g
```

Al final de su ejecución, el programa crea un archivo llamado ruta.tsp donde se imprime la ruta de la mejor solución encontrada.

Si se quiere ver la documentación del proyecto, basta con ejecutar

```
godoc -http=:6060
```

y acceder desde el navegador a localhost:6060/pkg/github.com/andreagonz/recocido/

Diseño

Se dividió el diseño en la parte genérica del recocido y la parte de implementación específica para el problema del agente viajero, de manera que se pueda reutilizar el código para algún otro problema. La parte genérica está implementada en heuristica/heuristica.go, donde se hace uso de las interfaces de Go para representar las soluciones de un lote y se implementaron los algoritmos del recocido para calcular el lote, la temperatura y la aceptación por umbrales. La parte orientada al problema del agente viajero está en implementacion/solucion.go, donde se implementaron los métodos de la interfaz Solucion.

Por otro lado se tiene otras funciones en util/ que permiten manipular archivos, crear las gráficas y hacer los mapas.

Experimentación

Se toma en cuenta al directorio ∼/go/src/github.com/andreagonz/recocido como directorio base. Para probar el programa fue proporcionado un conjunto de ciudades que están guardadas en archivos/prueba.tsp.

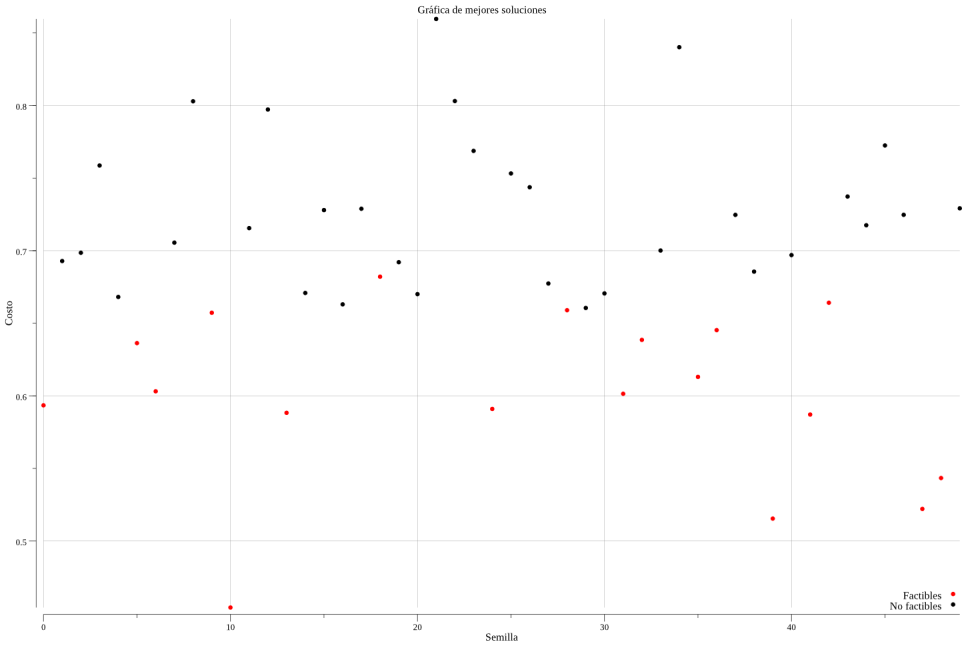
Se utilizaron 4 conjuntos de parámetros y se corrieron 50 semillas (de la 0 a la 49) distintas para cada uno.

Parámetro	Params1	Params2	Params3	Params4
$Lote$	500	500	500	1000
P	0.9	0.9	0.9	0.9
ϵ_p	0.01	0.001	0.001	0.001
ϵ_t	0.01	0.01	0.001	0.001
ϵ	0.01	0.001	0.001	0.001
ϕ	0.9	0.9	0.9	0.9
c	5	5	5	5

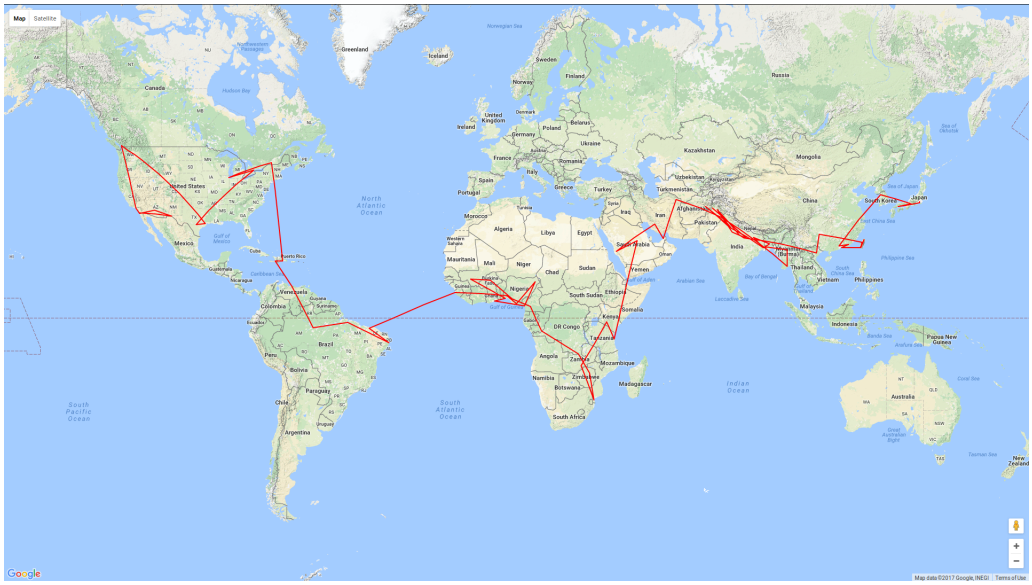
A continuación se mostrará los resultados que se obtuvo con cada conjunto de parámetros.

Params1

Con este conjunto de parámetros se obtuvo un %36 de soluciones factibles. Se muestra la gráfica de número de semilla contra costo de la mejor solución obtenida con esa semilla. Tambien se puede ver si la solución es factible o no.



La mejor solución encontrada tuvo un costo de **0.454207349**, lo cuál es decente pero más alto de lo que nos gustaría y como se puede apreciar, es una ruta algo alejada de la óptima. Se obtuvo con la semilla **10**.



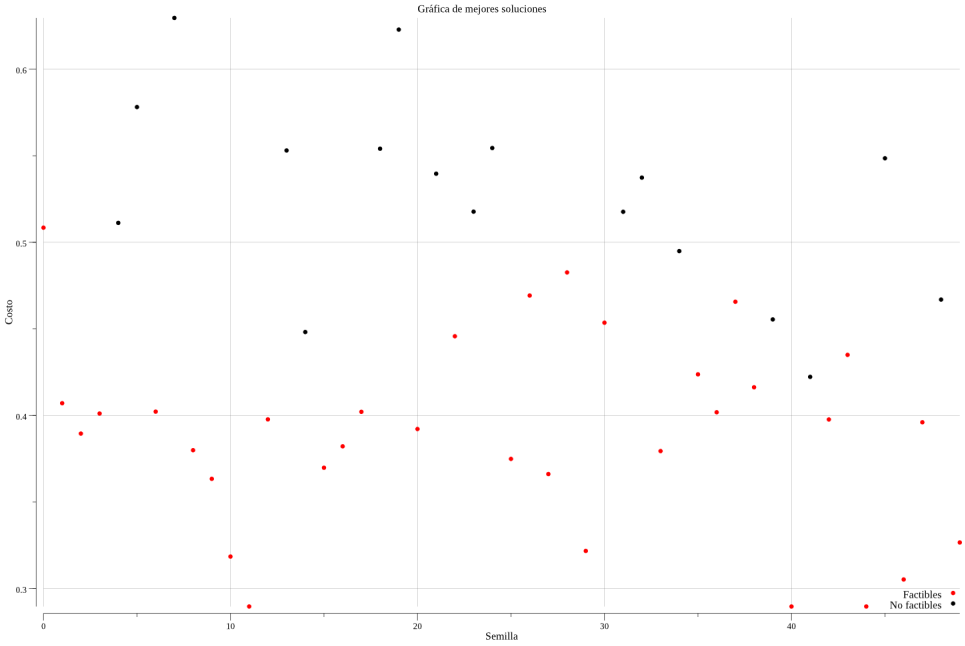
La gráfica del costo de las soluciones aceptadas de la mejor solución es la siguiente (se omitieron varias soluciones ya que eran demasiadas):

El promedio de los costos de las mejores soluciones de todas las semillas fue 0.681062436

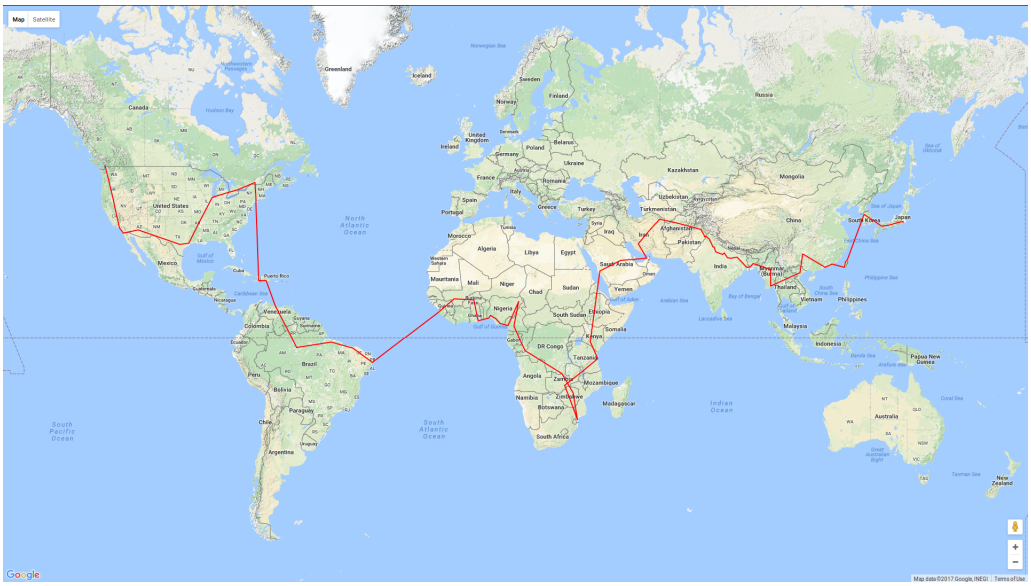
La mejor solución encontrada con Params1 se puede obtener con los parámetros archivos/params/1.txt.

Params2

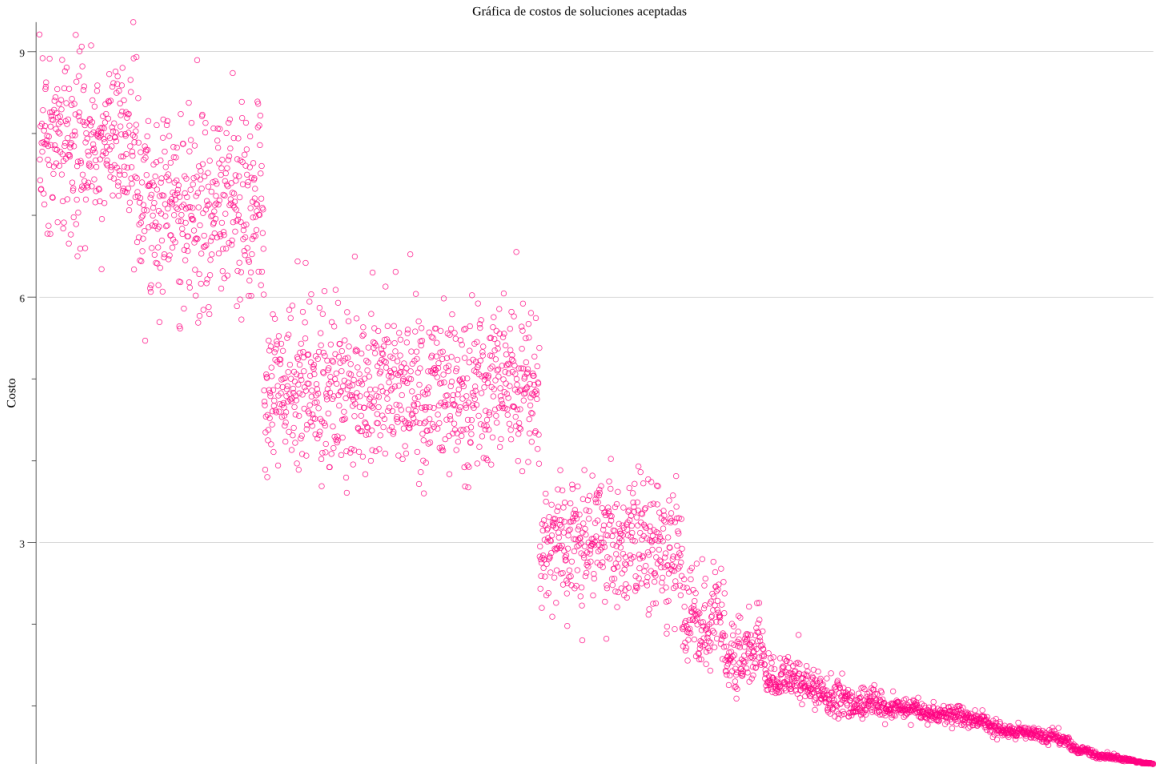
Se obtuvo un %66 de soluciones factibles, muchas más que en el caso anterior.
Las gráficas obtenidas son las siguientes:



La mejor solución encontrada tuvo un costo de **0.289669266**, se obtuvo con la semilla **40**. Por lo que podemos ver en el mapa ésta pareciera ser la ruta óptima, sin embargo como veremos adelante se encontró una mejor.



Gráfica del costo de las soluciones aceptadas de la mejor solución:

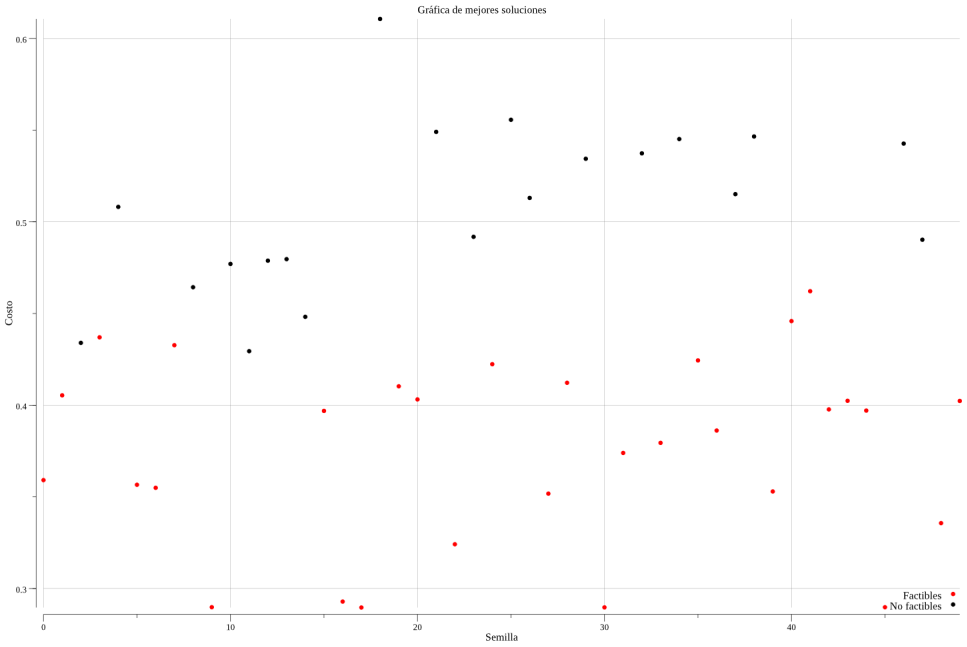


El promedio de los costos de las mejores soluciones de todas las semillas fue 0.435944361, lo cuál es una gran mejora del caso anterior.

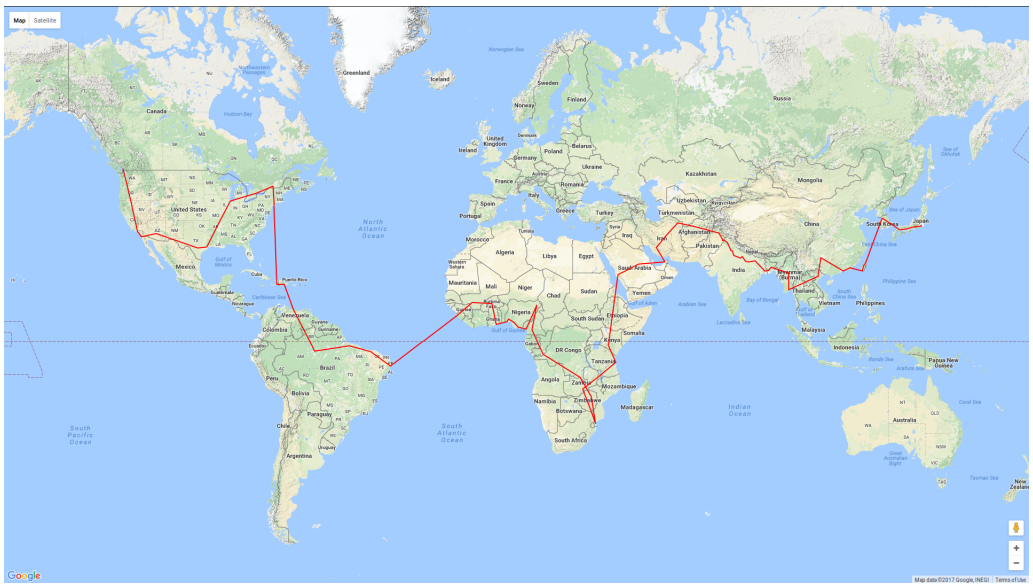
La mejor solución encontrada con Params2 se puede obtener con los parámetros archivos/params/2.txt.

Params3

Se obtuvo un %60 de soluciones factibles, poco menos que en el caso anterior.



La mejor solución encontrada tuvo un costo de **0.289633223**, se obtuvo con la semilla **17**. Ésta es mejor que la anterior y pareciera ser la óptima, aunque queda la duda.



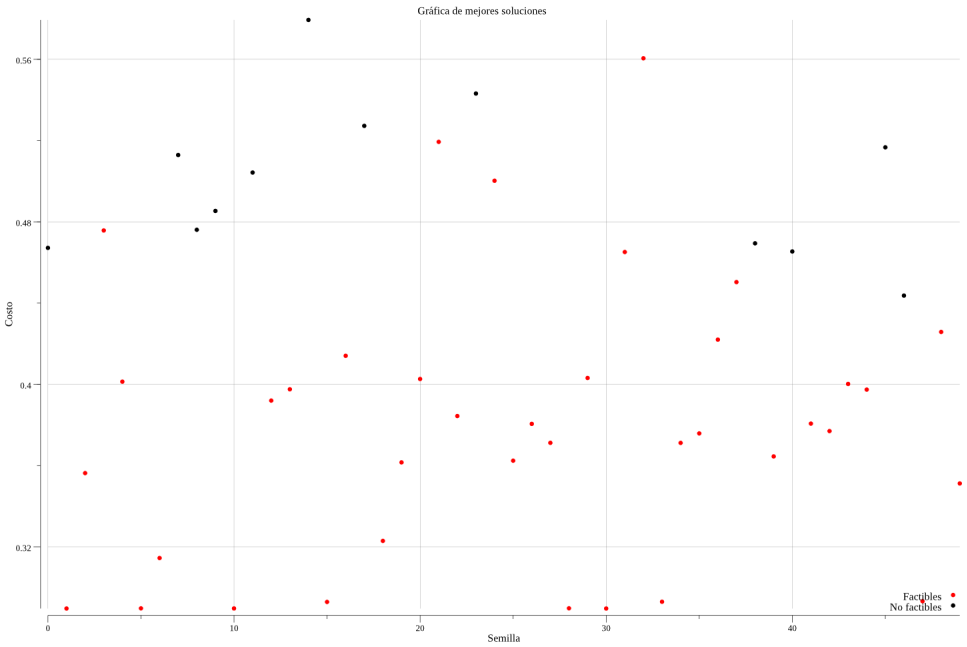
Gráfica del costo de las soluciones aceptadas de la mejor solución:

El promedio de los costos de las mejores soluciones de todas las semillas fue 0.428595471, muy parecido al caso anterior.

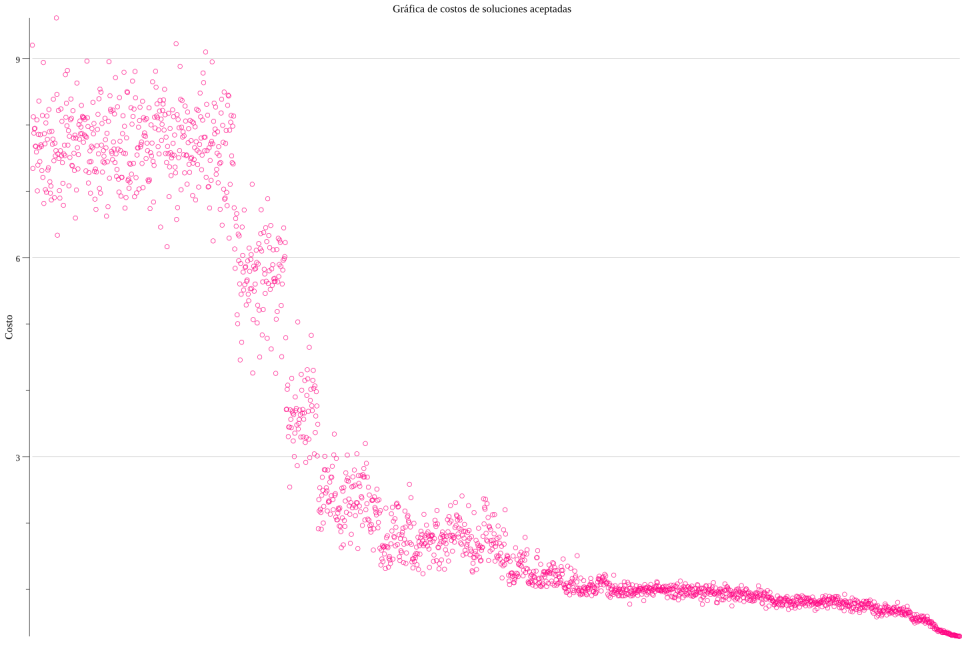
La mejor solución encontrada con Params3 se puede obtener con los parámetros archivos/params/3.txt.

Params4

Se obtuvo un %76 de soluciones factibles, lo cuál ya es bastante más alto que en los casos anteriores.



La mejor solución encontrada tuvo un costo de **0.289633223**, se obtuvo con la semilla **1** y la **30**. Ésta es la misma que en el caso anterior (por ésto se omite el mapa) y la mejor solución obtenida con estos conjuntos de parámetos.
Gráfica del costo de las soluciones aceptadas de la mejor solución:



El promedio de los costos de las mejores soluciones de todas las semillas fue 0.408394459, lo cuál es una pequeña mejora comparada con los dos casos anteriores.

La mejor solución encontrada con Params4 se puede obtener con los parámetros archivos/params/4.txt.

Nota: Todas las gráficas se encuentran en archivos/graficas y los mapas en archivos/mapas

Conclusiones

Params1 resultó en soluciones decentes pero lejos de ser las mejores, sin embargo el tiempo de ejecución para cada solución fue bastante menor que con los otros parámetros, ya que las *epsilon* utilizadas son más grandes en comparación.

Con Params2 se pudo obtener una solución muy buena, muy cercana a la óptima y en un número mucho mayor de soluciones factibles, la desventaja es que el programa se tarda más en ejecutarse que con Params1.

Params3 y Params4 lograron llegar a una solución que podría ser la óptima, y si no lo es se acerca demasiado. Como es el caso con Params2, la ejecución del programa es bastante más tardada, en especial con Params4, ya que el lote tiene el doble de tamaño que el de los otros casos, sin embargo Params4 logró obtener un porcentaje bastante alto de soluciones factibles. Como podemos ver, mientras perdemos en tiempo ganamos en soluciones, y al contrario.

El recocido simulado sirvió muy bien para resolver el problema del agente viajero con el conjunto de ciudades que se nos propuso, el cuál no era muy pequeño pero tampoco muy grande, falta ver que sucede con conjuntos más grandes.

La implementación del recocido fue bastante sencilla, sin embargo hay un muchos lugares en los que te puedes perder sin darte cuenta, lo cuál fue mi caso. Go ayudó a que me perdiera en un principio, pero una vez que se le encuentra sentido a este lenguaje, resulta ser una buena herramienta, salvo por algunas cosas subjetivas.