upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Genéticos
Practica Final - TSP

Jhonny Fabricio, Chicaiza Palomo - https://github.com/jhonnyfc
Computación

Índice

Desarrollo V0:

Desarrollo V1:

Link Repositorio Git Hub: **https://github.com/jhonnyfc/Geneticos_TSP_Viajante**

Explicación del Código - V0

El sistema consta de un fichero llamado **PracFin.m** este es el que llama y ejecuta el simulador. En este archivo se ha fijado una seed para que las ciudades para una simulación sean las mismas y se añaden las rutas de todas las carpetas necesarias.

La función que se encarga de recopilar los datos para la ejecución del sistema esta en el archivo **simulador.m** en la carpeta Simulación. Para realizar los experimentos se considerado oportuno que se repita más de una vez el mismo problema a resolver para elegir la mejor para una determina configuración, por ello se le pide al usuario que introduzca el numero de experimentos. Dependiendo del número de experimentos y de la configuración elegida puede tardar mucho en ejecutarse.

En la carpeta **Simulación** podemos encontrar las funciones: **algoGeneti.m** que ejecuta un experimento al pesarle los parámetros apropiados, **csvToAdya.m** lee el fichero con los datos de un país y obtiene su matriz de adyacencia, **fitnessPob.m** calcula el fitness para una población se ha elegido la suma de la distancia, **generaPob.m** según los parámetros revividos genera un población para realizar el experimento, se puede elegir cual es la ciudad inicial aunque con la gran dimensión de los data set se ha dejado este valor fijo si se quiere cambiar este valor está en la línea 32 de archivo **simulador.m**.

En la carpeta **Selección** se encuentra la función que se utiliza para la selección de progenitores. El método seleccionado ha sido el del torneo **selecTorneo.m** debido a que es mejor que el método de la ruleta dado que consideramos el fitness de la población y este es un factor importante para la evolución de esta. El tipo de torneo que se esta haciendo es con emplazamiento un mismo padre puede ser elegido varias veces.

En la carpeta **Mutaciones** se encuentran los métodos referentes a la mutación. Se ha implementado la mutación por inserción (**mut_inser.m**), intercambio (**mut_Intercambio.m**), inversión (**mut_Inver.m**) y sacudida (**nut_Sacu.m**).

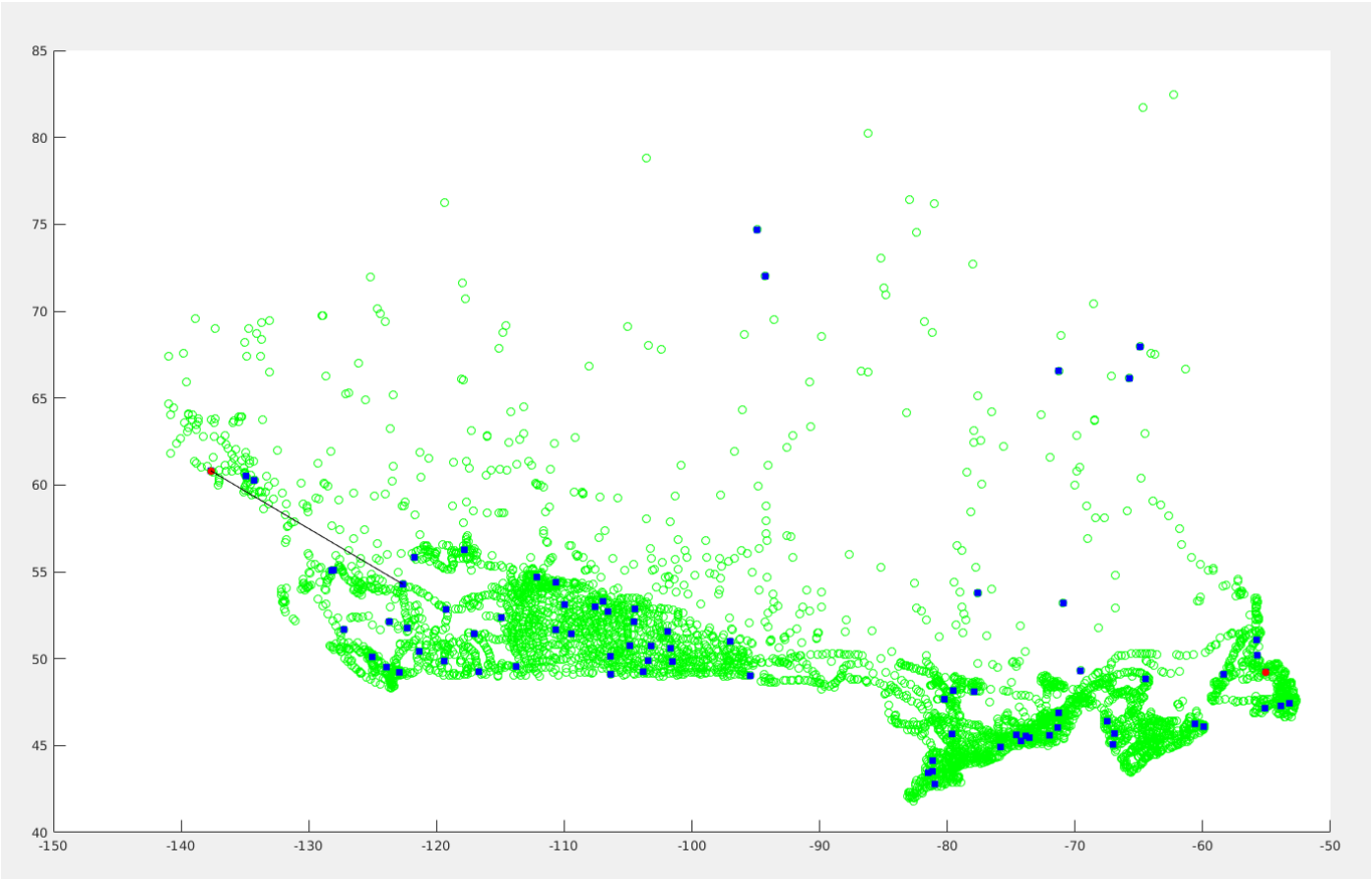
En la carpeta **Cruces** se encuentran los cruces implementados, disponemos de los siguientes tipos: ciclo (**cruz_Ciclo.m**), mapeado (**cruz_Map.m**), orden (**cruz_Orden.m**).

En el archivo **readPais.ipynb** se ha implementado un archivo en Python notebook para adaptar las coordenadas obtenidas de la página web dada.

En el archivo **algoGeneti.m** en la línea 7 se encuentra la variable **nRep** que mide las veces que se a repetido un mínimo local, se puede cambiar manualmente si se desea.

Experimentación - V0

El problema propuesto es el de encontrar el camino más corto para recorrer las ciudades de un país. Este problema es conocido como el problema del viajante (Travelling Salesman Problem). En la imanen siguiente se muestra el problema a resolver donde se marcan en azul los puntos par los que buscaremos el mejor camino.



Para que los experimentos sean contrastables se ha fijado una la semilla 12345 al principio. Gracias a fijar esto la elección de las ciudades al principio son las mismas. Como parámetros comunes para los distintos experimentos son los siguientes:

- Tipo de problema: 2
- Como ciudad Canadá: 1
- El número de ciudades: 80
- Ciudad Inicial 4
- Tamaño de la población: 90
- Número de generaciones: 500
- Número de Experimentos: 100 (Dentro del bucle de los experimentos se cambia el seed para que salgan resultados distintos y buscar el mínimo)
- Probabilidad de mutación: 0,5

- Cruzamiento: 1 (Mapeado), Mutación: 1 (Intercambio)
    Mínimo encontrado: 166933.538 km
- Cruzamiento: 1 (Mapeado), Mutación: 2 (Inversión)
    Mínimo encontrado: 164862.228 Km
- Cruzamiento: 1 (Mapeado), Mutación: 3 (Sacudida)
    Mínimo encontrado: 165348.098 Km
- Cruzamiento: 1 (Mapeado), Mutación: 4 (Inserción)
    Mínimo encontrado: 161907.493 Km

- Cruzamiento: 2 (Orden), Mutación: 1 (Intercambio)
    Mínimo encontrado: 168376.225 Km
- Cruzamiento: 2 (Orden), Mutación: 2 (Inversión)
    Mínimo encontrado: 153439.985 Km
- Cruzamiento: 2 (Orden), Mutación: 3 (Sacudida)
    Mínimo encontrado: 150020.713 Km
- Cruzamiento: 2 (Orden), Mutación: 4 (Inserción)
    Mínimo encontrado: 162226.870 Km

- Cruzamiento: 3 (Ciclo), Mutación: 1 (Intercambio)
    Mínimo encontrado: 162408.224 Km
- Cruzamiento: 3 (Ciclo), Mutación: 2 (Inversión)
    Mínimo encontrado: 153889.166 Km
- Cruzamiento: 3 (Ciclo), Mutación: 3 (Sacudida)
    Mínimo encontrado: 167678.017 Km
- Cruzamiento: 3 (Ciclo), Mutación: 4 (Inserción)
    Mínimo encontrado: 165371.790 Km

Análisis - V0

Después de las experiencias realizadas anteriormente podemos concluir los siguientes enunciados.
Al utilizar el Ciclo como método de cruzamiento hemos obtenido los mejore tiempos independientemente de los métodos de mutación utilizados. Por el contrario, con el método de cruzamiento Mapeado se ha tardado mucho más tiempo en ejecución.
En cuanto a la calidad de la solución se puede ver en los resultados que la mínima distancia obtenida se ha hecho con la configuración: {Cruzamiento: 2 (Orden), Mutación: 3 (Sacudida)} donde la distancia mínima ha sido de 150020.713 Km.

Para confirmar que este método obtuvo una buena soluciona se ha vuelto repetir ese mismo experimento, pero no se ha vuelto a obtener el mismo mínimo, se ha obtenido 163978.184 Km.

Se probo a subir el número generaciones y se bajó la probabilidad y no se consiguió mejorar la ruta, por eso se ha decidido hacer una nueva versión con algunas mejoras.

En el archivo **algoGeneti.m** en la línea 7 se encuentra la variable **nRep** que mide las veces que se a repetido un mínimo local, se puede cambiar manualmente si se desea.

Explicación del Código – V1 (modificaciones)

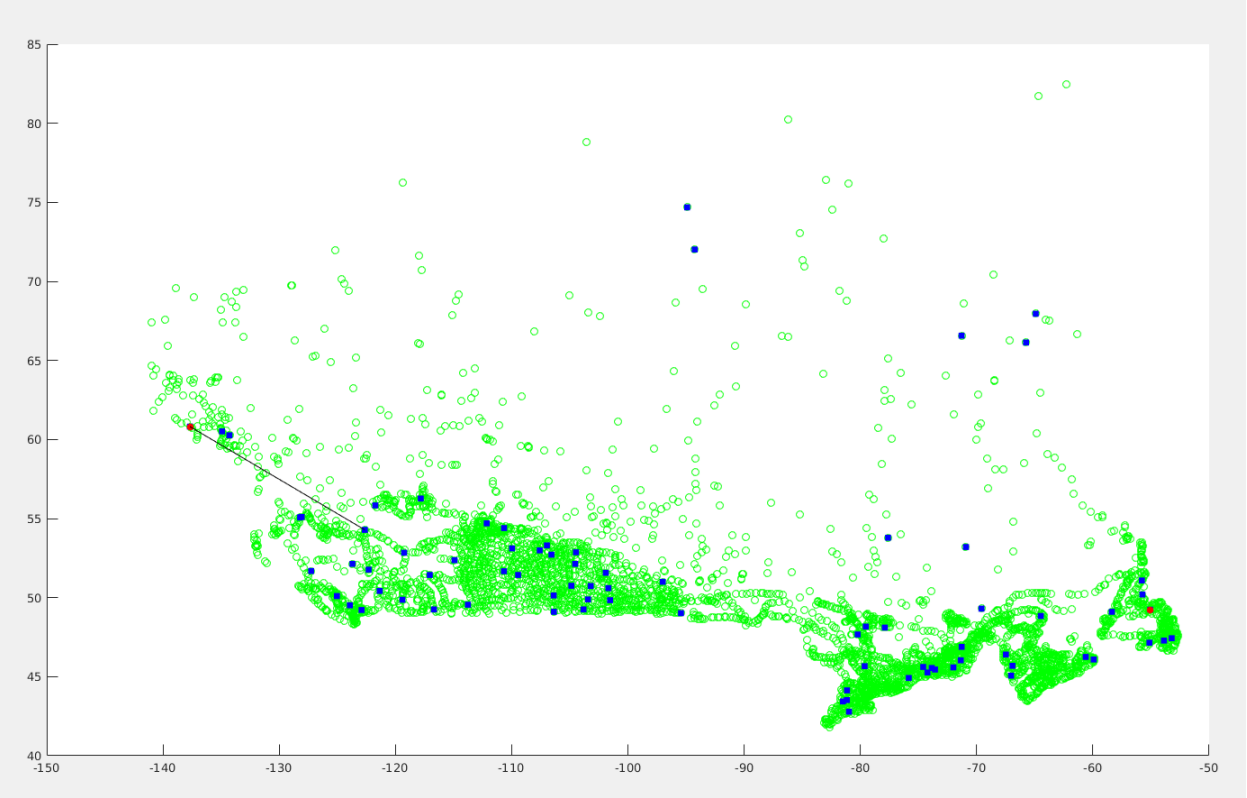
En la carpeta **Simulación** hemos modificado: **algoGeneti.m** se ha añadido que a la hora de generar hijos se permita elegir el número de cruces mediante la función (**realizaCruce.m**), a la hora de insertar los nuevos hijos se ha creado una función (**insertHijos.m**) que nos permite insertar los nuevo hijos quitando los de mayor fitness (mayor distancia de recorrido) dependiendo una probabilidad que la pone el usuario, además se hecho que el contador que cuneta el número de veces que se repiten se repiten mínimos locales se ponga a 0 cunado hay nuevo mínimo. Se han creado la función **realizaMut()** para que este más claro el código.

Se han creado La carpeta **Tools** para distribuir las funciones que no están implicadas directamente en el algoritmo genético **csvToAdya.m** y **plotCaminoSol.m**.

Se ha hecho control de errores por si el usuario introduce una variable mal.

Experimentación - V1

El problema a resolver es el mismo que en la V0 para poder certificar si ha habido mejoras o no.



Para que los experimentos sean contrastables se ha fijado una la semilla 12345 al principio. Se ha decidido modificar los siguientes parámetros debido que se han hecho varia pruebas y se han encontrado otros que se adaptan mejor:

- Tipo de problema: 2
- Como ciudad Canadá: 1
- El número de ciudades: 80
- Ciudad Inicial: 4
- Tamaño de la población: 55 ← Se ha reducido este valor
- Número de generaciones: 5050 ← Se ha aumentado considerablemente este valor para mejorar la evolución debido a que se ha visto que al reducir la población y aumentar las generaciones aumenta la calidad de la solución.
- Número de Experimentos: 150 ← (Dentro del bucle de los experimentos se cambia el seed para que salgan resultados distintos y buscar el mínimo)
- Numero de cruces: 3 ← En cada cruce se crean 2 hijos.
- Probabilidad de Inserción: 0,7 ← (Esta variable es para decidir si cambiar el hijo nuevo por uno de mayor fitness (mayor distancia de recorrido) o por uno aleatorio
- Probabilidad de mutación: 0,2

- Cruzamiento: 1 (Mapeado), Mutación : 1 (Intercambio)
    Mínimo encontrado: 48965.750 Km
- Cruzamiento: 1 (Mapeado), Mutación: 2 (Inversión)
    Mínimo encontrado: 27859.843 Km
- Cruzamiento: 1 (Mapeado), Mutación: 3 (Sacudida)
    Mínimo encontrado: 28295.062 Km
- Cruzamiento: 1 (Mapeado), Mutación: 4 (Inserción)
    Mínimo encontrado: 40591.784 Km

- Cruzamiento: 2 (Orden), Mutación: 1 (Intercambio)
    Mínimo encontrado: 43815.843 Km
- Cruzamiento: 2 (Orden), Mutación: 2 (Inversión)
    Mínimo encontrado: 28829.846 Km
- Cruzamiento: 2 (Orden), Mutación: 3 (Sacudida)
    Mínimo encontrado: 28254.974 Km
- Cruzamiento: 2 (Orden), Mutación: 4 (Inserción)
    Mínimo encontrado: 37428.566 Km

- Cruzamiento: 3 (Ciclo), Mutación: 1 (Intercambio)
    Mínimo encontrado: 47386.605 Km
- Cruzamiento: 3 (Ciclo), Mutación: 2 (Inversión)
    Mínimo encontrado: 28588.310 Km
- Cruzamiento: 3 (Ciclo), Mutación: 3 (Sacudida)
    Mínimo encontrado: 28947.805 Km
- Cruzamiento: 3 (Ciclo), Mutación: 4 (Inserción)
    Mínimo encontrado: 34640.992 Km

Análisis - V1 & Conclusiones

Después de las experiencias realizadas anteriormente podemos concluir los siguientes enunciados.

En cuanto al tiempo de ejecución después ella nuevas modificaciones hechas en el código todos los experimentos han durado más o menos lo mismo.
En cuanto a la calidad de la solución se puede ver en los resultados que la mínima distancia obtenida se ha hecho con la configuración: **{Cruzamiento: 1 (Mapeado), Mutación: 2 (Inversión)}** donde la **distancia mínima** ha sido de 27859.843 Km.

Para confirmar que este método obtuvo una buena soluciona se ha vuelto repetir ese mismo experimento y se obtuvo 28218.105 Km con este dato se puede decir que si se obtuvo un buen resultado.

Los demás resultados no son malos, en concreto en los que se ha conseguido cerca de 28000km son bastante buenos dado que según el grafo de la mejor solución obtenida no se podía mejorar la solución mucho más.

Se puede decir que el algoritmo desatollado en la **v1** es mucho mejor que el de la **v0** dado que los resultado son mucho mejores e incluso se han conseguido con diferentes problemas no registrados **grafos óptimos**.

Las mejoras que pueden haber dado estos buenos resultado son la modificación de la inserción de los nuevos hijos, en las posiciones de peor fitness según un umbral o una posición aleatoria si el valor random es mayor que el umbral. El otro factor clave ha sido el modo de contar los mínimos locales que se ha decidido reiniciar el contador si se encuentra un mínimo distinto.