Proyecto Final
Problema del Agente Viajero con Algoritmos Genéticos

Emmanuel Cruz Hernández
314272588

13 de Junio de 2019

1 Objetivo

El objetivo principal de este proyecto es mostrar una de las aplicaciones de inteligencia artificial para resolver problemas. En particular, se muestra como resolver el problema del viajero usando algoritmos genéticos.

2 Obtención de datos

Los datos para obtener las coordenadas e identificadores de cada ciudad se pueden contrar en la página:

http://www.math.uwaterloo.ca/tsp/world/countries.html#DJ

En esta dirección se encuentra una serie de países con ciudades y las coordenadas de cada una de ellas con un identificador. Esta práctica usa el pais Djibouti con 38 ciudades para mostrar en la interfaz gráfica el procedimiento de búsqueda. Sin embargo, se puede usar cualquiera de las ciudades con esta práctica pero el procedimiento no es visible en la interfaz gráfica, sino en una ventana de la terminal en forma de vector y su correspondiente distancia.

Para modificar la ciudad con la que se va a ejecutar el algoritmo se crea el archivo con extensión .txt con las ciudades del pais deseado y se cambia el archivo de lectura en la clase anidada *Camino* en el método llamado *readCities()*.

3 Introducción

3.1 Problema del Agente Viajero

Dado un conjunto finito de ciudades o puntos con coordenadas 'x' y 'y', se quiere encontrar el recorrido con menos costo para visitar todas las ciudades sin

repetirlas y terminar en la misma ciudad o punto de inicio.

Sea C={c | c es una ciudad}, $\forall$ c_1,c_2 $\in$ C, $\exists$ un costo r entre las ciudades c_1 y c_2.

Como se mencionó anteriormente, una ciudad tiene coordenadas 'x' y 'y'. Sean C_1 y C_2 con coordenadas x_1, y_1 y x_2, y_2, respectivamente. Este costo estará definido como:

R: C $\times$ C $\rightarrow$ $\Re = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

Dado un camino entre las ciudades (c_1, c_2, ..., c_n), cumpliendo las características definidas, lo que se pretende encontrar es el camino con menor costo. Este recorrido está definido como:

$$r_{recorridoFinal} = R(c_1, c_n) + \sum_{i=1}^{n-1} R(c_i, c_{i+1}) \quad (1)$$

Cabe mencionar que este problema es conocido por ser de la clase NP-Completo. Esto quiere decir que no se puede resolver en un tiempo polinomial por algoritmos deterministas. Con algoritmos genéticos, se aproxima a la mejor solución para recorrer todas las ciudades en un tiempo no determinado.

3.2 Algoritmos Genéticos

Los Algoritmos Genéticos están basados en el proceso genético de los organismos vivos, es decir, se pretende simular el comportamiento de los mismos a través del paso de generaciones. Estas generaciones son el resultado de la combinación de genes de generaciones anteriores, en las que van a prevalecer las mejores características del individuo de la generación anterior. Además de que en cada individuo puede ocurrir una mutación o deformación de un individuo con cierta probabilidad, la cuál debe ser muy baja, ya que esto no sucede frecuentemente. En este caso, se usa una probabilidad de 0.01 para que pueda ocurrir una mutación entre los individuos de la población actual. Para saber cuales son las características que van a conservarse y cuáles se van a descartar se recurre a los principios de la selección natural y la supervivencia de los más fuertes, postulados por Darwin (1859).

Otro concepto importante para este tipo de algoritmos es la evolución. La evolución nos permite encontrar soluciones hacia valores óptimos del problema que depende en buena medida de la codificación y las variables con las que se trabaja. Tales como el tamaño de la población, la cantidad de iteraciones para correr el algoritmo o la probabilidad de que exista una mutación.

4 Desarrollo e Implementación

Para esta práctica, cada uno de los individuos tendrá una aptitud, que es el costo o ganancia que nos aporta este. Este costo será el recorrido de las ciudades dado un camino que será calculado con la ecuación 1.

A continuación, se define cada una de las operaciones que van a ser necesarias para llevar a cabo cada una de las iteraciones del aloritmo genético y se explica cuál es el funcionamiento de cada una de ellas.

4.1 Inicialización

La primera generación de población consta de 50 individuos de tipo Camino. Con esta población generada aleatoriamente se van a desprender las siguientes poblaciones, pasando siempre a la siguiente el mejor individuo (*Camino*) encontrado hasta el momento. Como se había dicho, el mejor camino es aquel que tiene aptitud con el menor valor entre todos los caminos d ella población actual.

El algoritmo que genera poblaciones se itera cierta cantidad de veces para actualizar los datos en pantalla. Este valor es fijo durante todo el proceso de búsqueda. Se actualiza la el mejor camino encontrado en la interfaz gráfica cada 100 iteraciones.

4.2 Método de Selección

Este método permite obtener un individuo de la población actual para recombinarlo con otro. Sin embargo, la obtención no este no es aleatoriamente, ya que se debe cumplir con cierta probabilidad de elección, esto con el fin de seleccionar a los individuos que tienen mejor aptitud y pueden mejorar la población con mayor velocidad. A este procedimiento o tipo de selección se le conoce como *selección proporcional de aptitud*.

Si r_i es la aptitud del individuo (la suma del recorrido de ciudades) la probabilidad de selección es:

$$p_i = \frac{r_i}{\sum_{j=1}^{50} r_i} \tag{2}$$

Esto es lo que nos permite tener a los mejores individuos con mayor probabilidad de ser seleccionados, dejando aquellos no tan aptos o con aptitud baja con menos oportunidad de recombinarse.

4.3 Recombinación

Esta operación permite tomar dos caminos seleccionados de la población actual y a partir de estos crea un nuevo camino. Para crear este nuevo camino, se toma un valor Random entre 1 y la cantidad de ciudades que tienen los caminos (en el caso de la ciudad muestra 38).

Una vez generado el valor Random r, se toman las primeras r ciudades del primer camino en ese orden. Este subconjunto de ciudades serán la primera parte del nuevo camino. Ahora, se toman las ciudades en el intervalo r al total de las ciudades del segundo camino y se añade a la lista de ciudades.

Sin embargo, esto no resuelve del todo el problema de recombinación de caminos, ya que la segunda parte del camino puede contener ciudades que ya están contenidas en la primera creación del nuevo camino. Recordemos que una

de las condiciones del problema del agente viajero es que no se deben de repetir las ciudades en un camino. Así que se debe verificar con un método *equals()* que no se repitan las ciudades.

Dicho lo anterior, se agregan todas las ciudades que no han sido visitadas del segundo camino. Pero para que se cumpla la condición de que se deben recorrer todas las ciudades, agregamos el resto que no se encontró en la primera parte ni en la segunda para completar todas las ciudades en el camino.

4.4 Mutación

La mutación es una operación que altera los genes de un individuo que recién fue creado a partir de la recombinación. Hablando de la práctica y los caminos en particular, esta operación altera el orden del recorrido de las ciudades. Cabe destacar, que esta alteración solo se aplica si se cumple cierta probabilidad que está fija. Recordemos que la mutación en los seres vivos es muy poco probable, así que fijamos este valor con una probabilidad de 0.01.

Lo que se hace primero es crear un valor Random en el intervalo 0 a 1. A partir de esto, verificamos si se cumple la probabilidad. Si la probabilidad no se cumple, regresamos el individuo sin alteraciones, es decir, regresamos el camino sin modificación alguna.

Por otro lado, en caso de que si se cumpla la probabilidad. Tomamos 2 ciudades aleatorias del camino y hacemos un intercambio de estas. Es decir, tomamos la ciudad en el índice i y la ponemos en el índice j y a la ciudad que estaba en el índice j la ponemos en el índice i. Este cambio, ya representa en sí, una alteración de los genes del individuo.

4.5 Algoritmo Genético

A continuación se muestra el pseudocódigo para generar cada una de las generaciones (nuevas poblaciones) para encontrar el camino con menor costo para recorrer todas las ciudades del país.

Este pseudocódigo está basado del código de la práctica en el método *algoritmoGenetico(n)*.

```
1:  poblacion = new Poblacion(50)
2:  poblacion.calculaAptitud()
3:  while iteraciones < 100 do
4:      nuevaPoblacion = new Camino[50]
5:      nuevaPoblacion.add(mejorSolucion())
6:      while lleno < 50 do
7:          camino1 = selecciona()
8:          camino2 = selecciona()
9:          caminoHijo = recombina(camino1, camino2)
10:         caminoHijo.mutacion()
11:         nuevaPoblacion.add(caminoHijo)
12:     end while
```

```
13:     poblacion = nuevaPoblacion
14:     caminoBest = daMejorSolucion()
15: end while
```

5 Conclusión

El inicio del algoritmo muestra un camino como el que aparece en la Figura 1. Se muestra un camino generado aleatoriamente en la iteración 0.

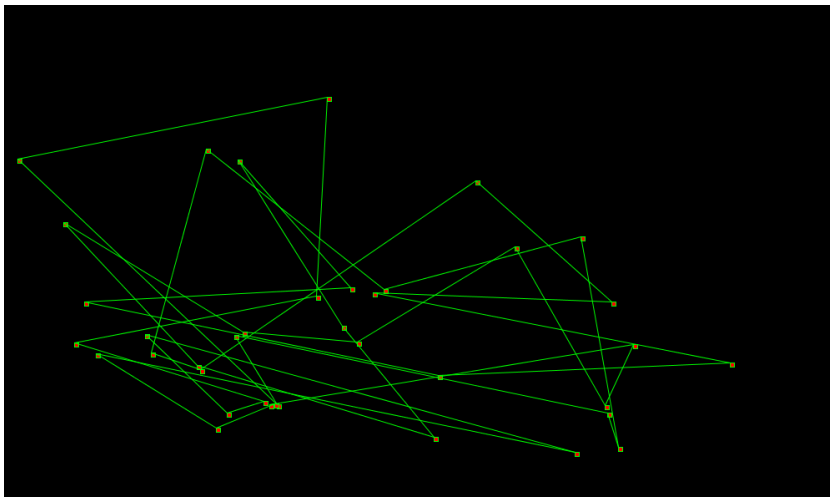

Figure 1: Camino inicial.

El algoritmo se ejecuta una cantidad infinita de iteraciones. Entonces, el programa se detiende hasta que el usuario lo desea y cierra la ventana de ejecución. Sin embargo, conforme aumenta el número de iteraciones, se aproxima más el mejor camino encontrado. Por lo que, entre más iteraciones se deje correr el programa mejor será la aproximación encontrada.

Para la iteración 100000 tendremos un camino similar al que aparece en la Figura 2

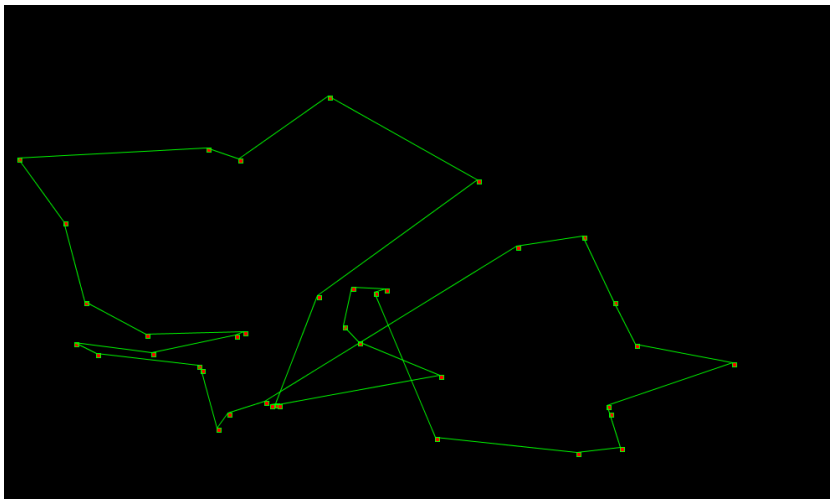

Figure 2: Mejor camino encontrado en la iteración 100000.

A comparación de este problema resuelto con Recocido Simulado, con Algoritmos Genéticos se resuelve más rápido. Las soluciones que ecuentra se acercan a una aproximación más cercana al camino perfecto.

La aproximación con Algoritmos Genéticos a este problema depende mucho de ciertas variables que se mencionaron como iniciales. El tiempo en que se acerca a la mejor solución está determinado por la probabilidad de mutación y el tamaño de la población. Entonces los parámetros iniciales son muy importantes para poder hacer que el algoritmo funcione bien.

6 Notas Adionales

Para poder correr el proyecto se hace uso de Processing, que es un lenguaje para codificar el contexto de las artes visuales. Esto lo usamos para poder generar puntos (ciudades) y líneas (caminos entre ciudades) en una interfaz gráfica. Por lo que debemos importar esta biblioteca.

Podemos obtener Processing de la siguiente página:

https://www.processing.org/download/