Complutense de Madrid

Práctica Programación Evolutiva.

Segunda práctica.

Raúl Torrijos & Lukas Häring

April 1, 2019

Tabla de contenidos

1 Introducción **2**
1.1 Mejoras implementadas 2
1.2 Aclaraciones 3

2 Algoritmos Propios **4**
2.1 Algoritmo de Cruce 4
2.2 Algoritmo de Mutación 4

3 Análisis de ejecuciones **5**
3.1 Ruleta 5
3.2 Contractividad 6
3.3 Elitismo 6

1 Introducción

En está versión hemos implementado una modificación del Algoritmo Genético para aproximarnos a las posibles soluciones de El problema del viajante de comercio, utilizando mejoras sobre el esquema básico del Algoritmo Genético Simple.

1.1 Mejoras implementadas

Las mejoras implementadas sobre el AGS que hemos incluido en esta versión, y que por lo tanto no se encontraban implementadas en la Práctica 1 son:

1. Algoritmos de Selección
    - Selección por Ranking
    - Selección por Truncamiento
2. Algoritmos de Cruce
    - Emparejamiento Parcial (PMX)
    - Cruce por Orden (OX)
    - Cruce por Orden con posiciones prioritarias (OXPP)
    - Cruce por Orden con orden prioritario (OXOP)
    - Ciclos (CX)
    - Recombinación de Rutas (ERX)
    - Codificación Ordinal (CO)
    - Método Propio 1 (explicado más adelante)
3. Algoritmos de Mutación
    - Inserción
    - Intercambio
    - Inversión
    - Heurística
    - Método Propio 1 (explicado más adelante)
4. Otras mejoras
    - Contractividad
    - Mapa de España con el mejor recorrido representado gráficamente

1.2 Aclaraciones

Nos gustaría puntualizar un par de pequeñas decisiones que hemos tomado a la hora de implementar el ejercicio:

- El número de ciudades que introducimos en el algoritmo es 27 (contando con Madrid), teniendo en cuenta que Madrid (id: 25) aparece tanto al principio como al final, nuestro cromosoma tiene por lo tanto 28 parámetros de entrada y cuenta con identificadores de ciudades desde el id: 0 (Albacete) hasta id: 26 (Málaga), por lo tanto nos dejamos fuera Murcia ya que entonces serían 28 ciudades y no 27.

En cualquier caso nuestra implementación está preparada para ejecutar el problema con más o menos ciudades.

- Hemos descubierto una errata en la tabla de distancias usada en la función de evaluación para evaluar el *fitness* de las ciudades; La distancia de **Guadalajara** a **Cuenca** es incorrecta, aparecen 486km cuando en realidad son 136km. Hemos actualizado al valor correcto en nuestra tabla.

De este error nos hemos podido dar cuenta gracias a nuestra representación del mapa de España tras observar en reiteradas ocasiones que siempre trataba de evitar el camino entre estas ciudades.

2 Algoritmos Propios

2.1 Algoritmo de Cruce

Nuestro algoritmo de cruza dos posibles soluciones al problema utilizando dos puntos de corte iguales en ambos padres y una componente aleatoria.

1. Se generan dos enteros aleatorios que servirán como puntos de corte comunes para los padres.
2. Se intercambian los subsegmentos entre los puntos.
3. Para ambos hijos, se empieza desde la primera posición sin rellenar y se elige un candidato aleatoriamente.
4. Si el gen elegido no está presente aún en el hijo, se inserta, si lo está se elige de nuevo un candidato aleatoriamente.
5. Se repite el proceso para el otro hijo.

Padre 1	$a1_0$	$a1_1$	...	$s1_{c1}$	...	$1s_{c2-1}$	...	$a1_{n-1}$	$a1_n$

Padre 2	$a2_0$	$a2_1$	...	$s2_{c1}$	...	$s2_{c2-1}$	...	$a2_{n-1}$	$a2_n$

Hijo 1	$a1_0$	$a1_1$	...	$s2_{c1}$	...	$s2_{c2-1}$	...	$a1_{n-1}$	$a1_n$

Hijo 2	$a2_0$	$a2_1$	...	$s1_{c1}$	...	$1s_{c2-1}$	...	$a2_{n-1}$	$a2_n$

Donde $c1, c2$ son los puntos de corte generados, $s1, s2$ los elementos de los subsegmentos a intercambiar, y $a1, a2$ los elementos generados aleatoriamente.

2.2 Algoritmo de Mutación

El algoritmo consiste en evaluar dos posibles mutaciones, estas mutaciones son las que se producen al intercambiar alguno de los vecinos más cercanos. Algoritmo:

1. Se genera un entero aleatorio desde 2 hasta $n-2$, donde n es la cantidad de genes.
2. Se realiza un intercambio del elemento que apunta este puntero con el vecino de la izquierda.
3. Si su evaluación es mejor, intercambiamos. Por el contrario, lo dejamos como está.
4. Teniendo en cuenta el cromosoma sin modificar, realizamos un intercambio con el vecino de la derecha.

5. De la misma forma, si la evaluación es mejor, intercambiamos.

Como vemos, el número aleatorio se elige entre 2 y $n-2$. Esto es, claro está, que si intercambiamos el puntero fuera 1, este cogería como vecinos 0 y 2, haciendo que pudiera mutar el gen inicial (Y eso no debe ocurrir).

Cromosoma 1	a_0	$\dots$	a_{k-1}	a_k	a_{k-1}	$\dots$	a_n

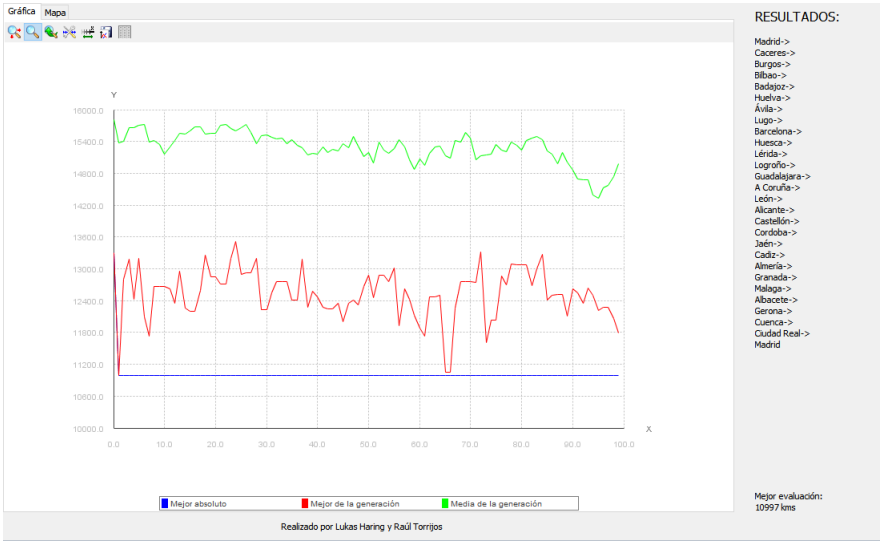
Donde $k \in \{2, \dots, n-2\} \subseteq \mathbb{N}$

3 Análisis de ejecuciones

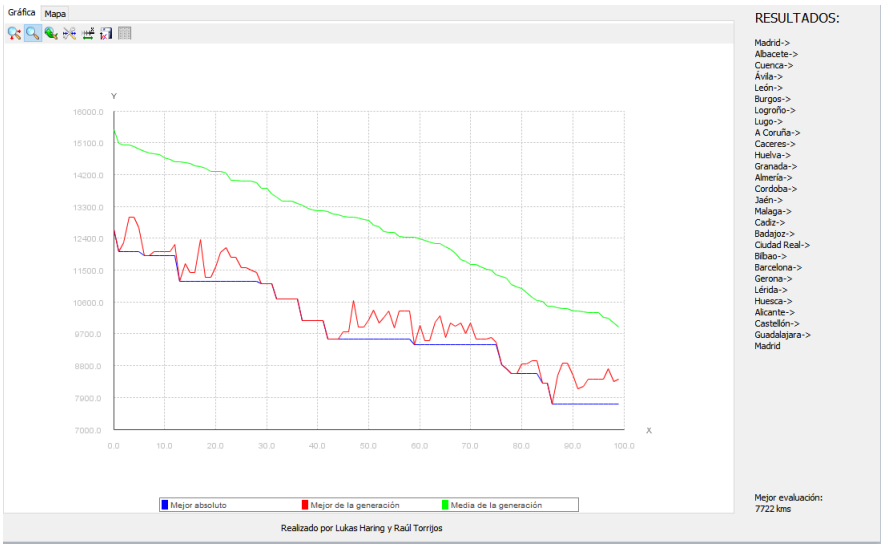
Para realizar un análisis exhaustivo del comportamiento de cada uno de los algoritmos usados hemos realizado una gran cantidad de ejecuciones y hemos obtenido las siguientes conclusiones.

3.1 Ruleta

Ruleta es el método de selección que menos favorece a la evolución ya que tiene una gran componente aleatoria y resuelve el ejercicio con valores muy lejanos al recorrido mínimo posible.



La única manera de obtener resultados mas razonables utilizando ruleta es usando contractividad y/o elitismo y/o ampliar el número de generaciones.



3.2 Contractividad

La contractividad hace que no pueda haber una generación con la media peor que la anterior, y esto en principio es algo positivo, pero puede suceder que cada vez sea más complicado sacar individuos mejores, y esto puede estancar el algoritmo

Para ello es necesario añadir otra condición de salida además del número de generaciones y hemos optado por un criterio de terminación enfocado al coste, cuando se produzcan 1000 fallos seguidos al intentar conseguir una generación, para la ejecución

```
while ((currentGeneration<total_generations) && (failureCount < 1000)) {
```

3.3 Elitismo

Hemos observado sobre este problema concreto el elitismo no adopta un papel demasiado determinante, ya que hemos conseguido valores muy próximos al recorrido mínimo posible sin elitismo activado