Yojhan Steven García Peña

Programación Evolutiva

Práctica 2 – Convocatoria extraordinaria

1. Gráficas

Problema:

Recorrer una serie de ciudades, emezando y terminado en Madrid, minimizando la distancia recorrida

Resultados:

Resultado: 5298

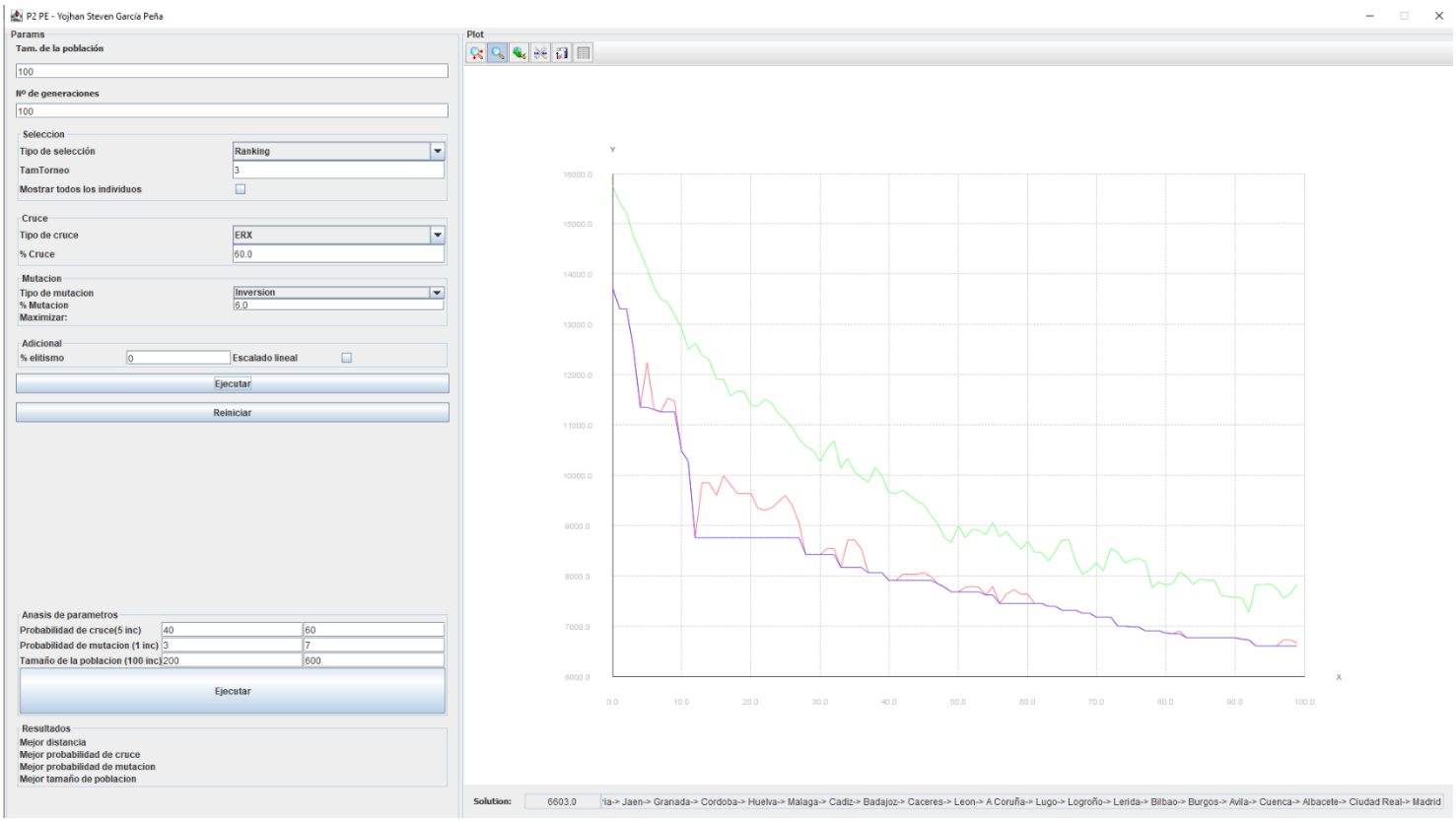
Recorrido:

Madrid-> Cuenca-> Albacete-> Murcia-> Alicante-> Castellón-> Gerona-> Barcelona-> Lérida-> Huesca-> Logroño-> Bilbao-> Burgos-> León-> A Coruña-> Lugo-> Ávila-> Cáceres-> Badajoz-> Huelva-> Cádiz-> Málaga-> Almería-> Granada-> Jaén-> Córdoba-> Ciudad Real-> Guadalajara-> Madrid

Este es el mejor resultado que he obtenido, y se ha logrado con una población de 500 individuos y a lo largo de 1.000 generaciones, consiguiendo el mejor resultado alrededor de la población 900. En este caso he utilizado la selección por truncamiento, cruce PMX con 55% de probabilidad de cruce y mutación por intersección con 3% de probabilidad de cruce. El algoritmo tenía aplicado un 10% de elitismo y activado escalado lineal.

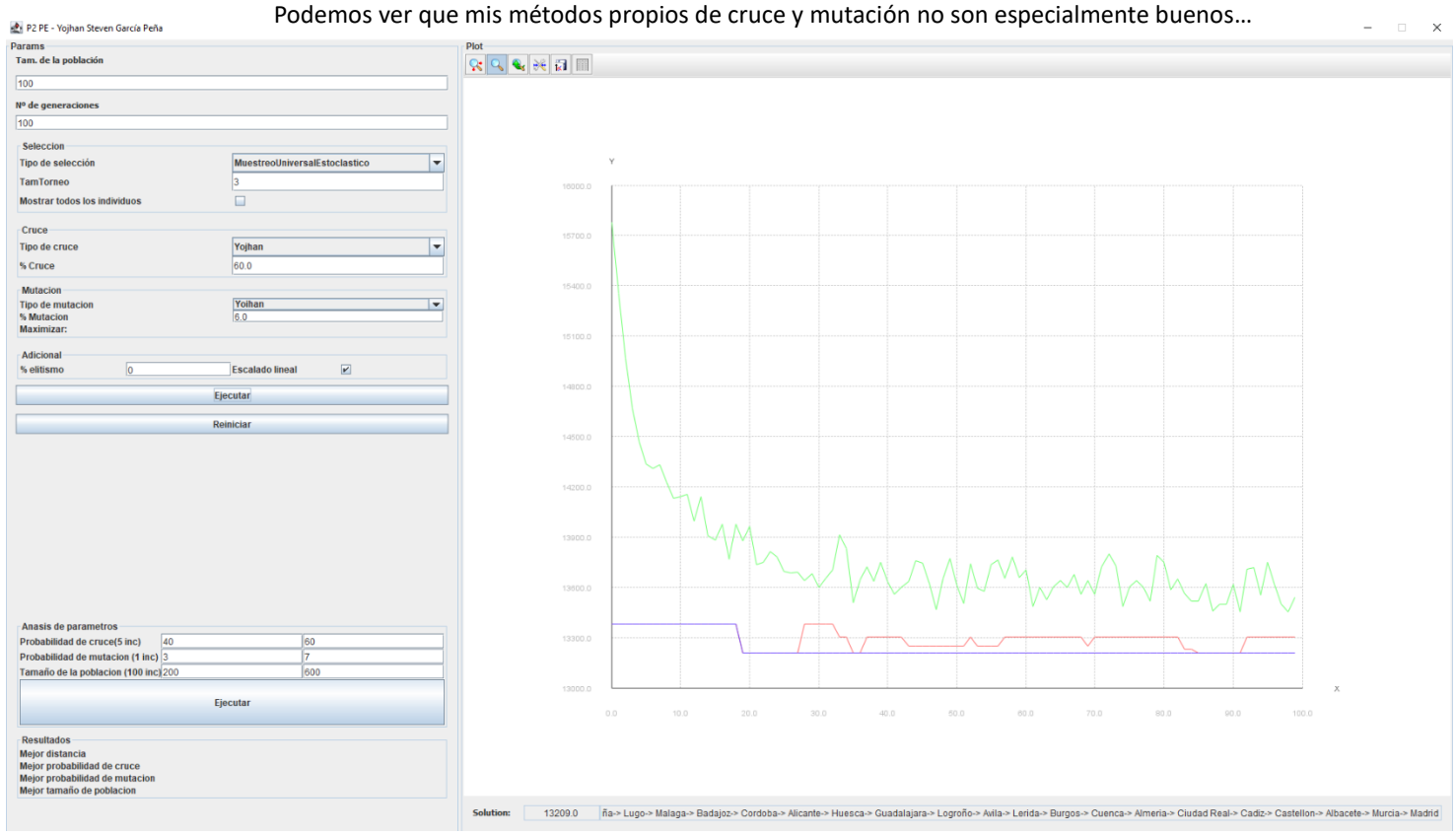
1.2 Otras gráficas con otros parámetros

Resultado: 6603



Resultado: 13209

Podemos ver que mis métodos propios de cruce y mutación no son especialmente buenos…



2 Implementación

Esta práctica ha sido hecha sobre la práctica 1, cambiando todas las cosas necesarias para su adaptación, por lo que gran parte del código es igual a la práctica anterior.

El primer cambio que tuve que hacer fue sobre el tipo de genes, eliminando la implementación anterior y haciendo una adaptada a genes enteros.

Otro cambio que he hecho con respecto a la implementación pedida es que por comodidad he hecho que las ciudades empiecen en 0 en vez de en 1, y terminen en N – 1.

Además, he eliminado a Madrid de la lista de ciudades, pues ya que siempre debemos comenzar y terminar en Madrid no me parecía necesario que Madrid formara parte de la codificación de los individuos.

Al igual que en la práctica anterior, he dejado que las gráficas se solapen pues creo que eso es bastante útil a la hora de comparar distintos parámetros en la interfaz. Si se pulsa el botón 'Reiniciar' la gráfica y la casilla de resultado se borran para poder lanzar otra ejecución desde cero.

Para manejar las ciudades, me he creado una clase estática auxiliar llamada 'DistanciaCiudades'. Esa clase contiene el array bidimensional con la matriz de distancias, y también tiene un array con las distancias de las otras ciudades hasta Madrid. También contiene un array con los nombres de las ciudades para poder escribir luego el recorrido con facilidad.

```
public static int Distancia(int A, int B) {

    if(A == B)
        return 0;

    int min = Math.min(A, B);
    int max = Math.max(A, B);

    return distancias[max][min];
}
```

Este es un ejemplo de cómo se calculan las distancias entre dos ciudades (a excepción de Madrid)

El resto de los cambios en el código es bastante sencillo, pues fue principalmente añadir el resto de los algoritmos de cruce y mutación, al igual que el método Ranking de selección.

```
info.fitnessFunction = (input) -> {

    int distanciaTotal = DistanciaCiudades.DistanciaMadrid(input[0]);


    for(int i = 0; i < input.length - 1; i++) {

        distanciaTotal += DistanciaCiudades.Distancia(input[i], input[i +1]);
    }

    distanciaTotal += DistanciaCiudades.DistanciaMadrid(input[input.length - 1]);

    return distanciaTotal;


};
```

Esta es la forma en la que se calcula la función de fitness. La variable input es el array de enteros que codifican al individuo. Como Madrid no forma parte de la lista de enteros tenemos que sumar su distancia con el primero y el último también.

Además, lo comentado anteriormente, he añadido también escalado lineal a la práctica, con la opción de poder activarla o desactivarla desde la interfaz.

```
private void EscaladoLineal(int idx) {

    if (idx == 0) {
        CorregirFitness();
        return;
    }

    double mejor = fitnessMejor[idx - 1];
    double media = fitnessMedio[idx - 1];

    double a = media / (media - mejor);
    double b = (1 - a) * media;

    for (int i = 0; i < poblacion.length; i++) {

        poblacion[i].SetFitness(a * poblacion[i].CalculateFitness() + b);
    }

    CorregirFitness();
}
```

Por último, el análisis de parámetros. He puesto en la esquina inferior izquierda de la interfaz un pequeño recuadro dónde se pueden analizar el valor de tres parámetros diferentes: el tamaño de la población, la probabilidad de cruce, y la probabilidad de mutación. Cada uno de esos parámetros cuenta con un valor mínimo y un valor máximo que podremos modificar. Si se pulsa el botón de ejecutar que se encuentra en ese recuadro se empezarán a ejecutar una secuencia de algoritmos con los valores entre los intervalos establecidos. Cada variable tiene su propio incremento (100 para número de generaciones, 1 para mutación y 5 para el cruce). El incremento es la diferencia que habrá en el valor entre ejecuciones.

Un añadido que le he puesto es ejecutar el análisis en un hilo separado, de esta forma se va actualizando la gráfica a medida que se van ejecutando distintos algoritmos.

Aquí un vídeo donde se puede ver cómo funciona la ejecución:

https://youtu.be/Heit4Yn6MRQ

Hacerlo en un hilo de vez en cuando genera errores en consola debido a las gráficas pero no sé exactamente a qué se debe, aunque no supone ningún problema adicional.

He añadido un recuadro donde se muestran el valor que tenían los distintos parámetros con los que se ha encontrado la mejor solución hasta el momento.

3 Conclusiones y curiosidades

Creo que he conseguido un buen resultado, no sé si se consigue la mejor solución, pero confío en que al menos conseguirá una que se le acerque.

He tenido una serie de problemas a la hora de implementar los algoritmos de cruce. Principalmente con el PMX pues conseguía individuos con ciudades repetidas constantemente. Otro problema que he tenido en general ha sido que mis ciudades al empezar desde el valor cero, cuando creaba un array nuevo no había contado con que se inician todos los valores a cero. Debido a eso he tenido algunos problemas con unos algoritmos, teniendo que después de crear el array, establecer todos los valores a menos uno.

Un problema que he tenido en concreto con el cruce ERX es que a veces se completaba el ciclo sin haber usado todos los hijos. Busqué una solución en internet y vi que habría que implementar un algoritmo de retroceso, pero me pareció bastante complejo así que opté por una solución mucho más cutre, en la que cuando se complete el ciclo, si quedan valores sin ser utilizados los relleno de forma manual en un bucle. Un ejemplo de un caso donde ocurre esto es: 1, 8, 7, 6, 5, 4, 3, 2.

Y, por último, haciendo la mutación heurística, tuve problemas porque no recordaba como conseguir todas las permutaciones de una secuencia de números y finalmente busqué en internet un método para conseguirlos de forma iterativa.

4 Reparto de tareas

Todo lo implementado en la práctica ha sido hecho por mí.