

# GRUPO 13

28 de marzo de 2023 -  
Programación Evolutiva



## INTEGRANTES

Andrés Cardenal Antón
Rafael Alonso García



## CONTENIDO

1. Guía de Uso, Arquitectura de la Aplicación y detalles de la implementación .....	4
2. Cruces y mutaciones.....	6
3. Elitismo.....	12
4. ¿Qué métodos de cruce y mutación son mejores? .....	12
5. Mejoras.....	14
6. Análisis de Convergencia.....	14
7. Problemas / Curiosidades encontrados .....	14
8. Reparto de tareas.....	15

# 1. Guía de Uso, Arquitectura de la Aplicación y detalles de la implementación

## EJECUCIÓN DEL PROGRAMA

Para probar el programa se arrancará el ejecutable o tras haber importado el proyecto, desde el IDE Eclipse.

Se introducirán los parámetros de la izquierda antes de empezar la prueba, tal como funcionaba en nuestra práctica 1 anterior. Para la ejecución del *Travelling Salesman Problem*, se pondrá el apartado “Function” al valor CITIES.

## DETALLES DE IMPLEMENTACIÓN

A la hora de realizar la práctica 2 nos hemos asegurado de que siga siendo compatible con las funciones de la práctica 1, ya que hemos adaptado el código al nuevo enunciado. Dentro de la Arquitectura se encuentran como clases clave en esta nueva práctica, al gen **CityGen** (implementa a *GenI*) que es una simplificación del anterior *BoundedGen* (este trabaja con límites inferiores y superiores en los valores). La función **FunctionTSP** (implementa *Fitness*, interfaz genérica de funciones) realiza el cálculo del valor de un cromosoma. Dicho cálculo se realiza accediendo desde el cromosoma **TravellerChromosome** a sus genes y desde cada gen a la variable asociada (nombre del gen, de “x1” a “x26”) y con las variables, obtenemos un valor del cromosoma (fenotipo) con la función *FunctionTSP*.

En la clase *TravellerChromosome* se implementa la inicialización de los individuos (junto con el **Initializer**, del paquete model.iniciatizer) y se realiza un traductor *getGenesToString()* con el fin de mostrar la solución la secuencia de ciudades.

A la hora de codificar la solución, entendemos “que siempre se empieza por Madrid”, así el gen “x1” corresponde a la siguiente ciudad a Madrid, el gen “x2” corresponde al siguiente al gen “x1” y así hasta el gen “x26”, el cual precede a Madrid. El valor que toma cada gen es un *Double* (por temas de retrocompatibilidad) que toma valores enteros entre 0 y 27, exceptuando el 25 (identificador de Madrid). En todo cromosoma, tal como se requiere en la solución al TSP, no se repite el mismo identificador de ciudad en 2 genes distintos.

En el paquete model.util podemos observar a las clases *Pair*, a los iteradores *MapIterator* (utilizado en nuestro método de cruce propio *Alternative Order*, AO), *Iterator* (utilizado en los métodos de cruce de orden (OX) y de orden prioritario (POX) y *ConnectionTable* (utilizado en el método de cruce de recombinación de rutas, para crear las conexiones de cada ciudad).

A la hora de realizar la mutación heurística (en *CityHeuristicMutation*) se utilizan las clases internas *Backtracking* y *ChromosomeVariator*, donde se hace uso del algoritmo “vuelta atrás” para implementar la mutación.

El model.debug se encuentra el **TravellerChromosomeChecker**, clase que sirve para comprobar que un cromosoma cumple la precondition de solución válida, y que sirve para depurar errores.

## ARQUITECTURA

Se sigue el patrón **Vista-Controlador-Modelo**. Así desde la clase *Launcher* se carga el cliente *Client* que podrá ser **Window**, para la ejecución usual en forma de ventana o *StatisticGenerator* que hemos creado con el fin de obtener una estadística acerca de qué métodos de cruce y mutación son mejores, aparte de obtener la

mejor solución TSP utilizando 10 ejecuciones de cada terna, método de cruce, método de mutación y elitismo (valores de 0% y de 5%).

En el paquete `src/statistics` también está la clase **StatisticsWriter** que genera un fichero `"solucion.txt"` con las 210 combinaciones que hemos generado. El **StatisticsGenerator** también obtiene la mejor solución TSP a la vez que genera la media del mejor individuo intergeneracional de los métodos de cruce y mutación. Se combinan estos dos últimos junto con los métodos de selección ruleta, ranking y probabilístico (70% de elección del campeón del torneo) y elitismo (valores de 0% y de 5%); utilizamos 10 ejecuciones por cada cuarteto para que la media sea más representativa. Para probar esa funcionalidad, será necesario pasar por parámetro el valor `"test"` y borrar el archivo `"statistics.txt"`.

Para ambos clientes, se utiliza a la clase **Request** para transportar la configuración de la ejecución hacia el **Builder** que es el encargado de traducir todos los parámetros en los objetos necesarios (funciones, métodos de selección, cruce o mutación) o el molde (**Mold**), el cuál almacena una referencia a la función y al tipo de genes que ha de contener cada cromosoma (todos los cromosomas contienen el mismo molde en cada ejecución).

Los nuevos métodos de cruce implementan a la clase abstracta *Crossover* que se encarga de los emparejamientos y los nuevos métodos de mutación implementan a **CityMutationI** que a su vez extiende a *MutationI*. Dado que la mutación se realiza a nivel de cromosoma y no a nivel de gen, diferencia con respecto a la práctica 1, el *TravellerChromosome* contiene a la mutación, y el *CityGen* ya no la contiene. Las clases *MutationBuilder* y *CrossoverBuilder* se han refactorizado para incluir a los nuevos métodos, que serán generados por el *Builder*, antes de que el **Controller**, ejecute el programa a través del *Executor*.

El **Executor** contiene el bucle principal del programa y no ha sufrido apenas modificaciones con respecto a la primera práctica.

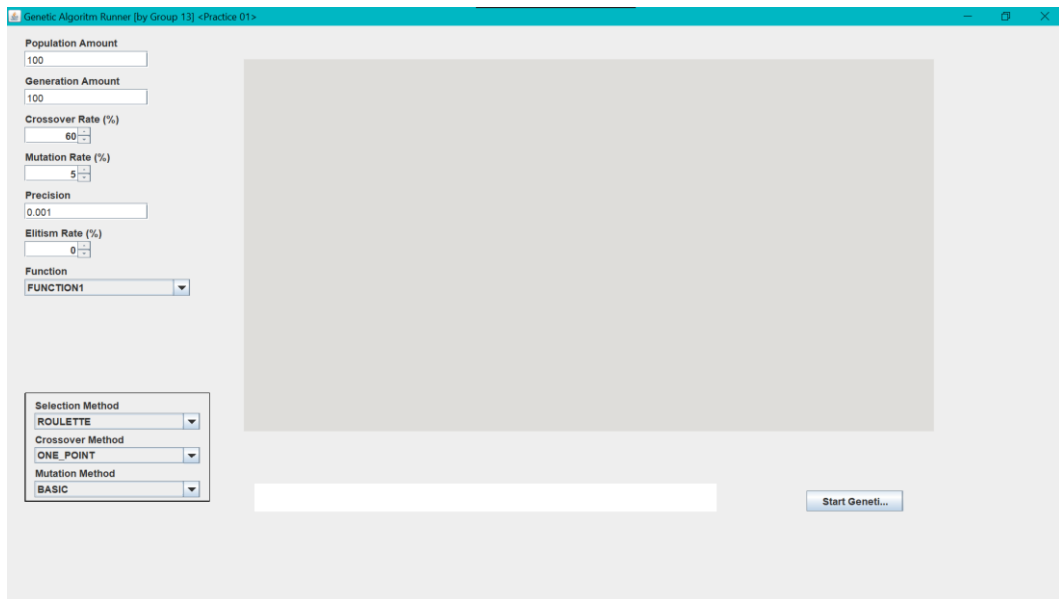
En el paquete `model.util` se encuentran clases de apoyo utilizadas por los nuevos métodos de cruce y mutación.

## NUEVOS MÉTODOS

Para el desarrollo de la práctica, además de los métodos pedidos, hemos creado 1 adicional tanto para el cruce como la mutación.

En el caso del cruce, tenemos el método de Orden Alternativo (AO), el cuál funciona de la siguiente manera: Empezamos en cualquiera de los 2 padres a cruzar, y vamos alternando los genes a añadir al hijo de tal manera que se añadirá el gen que le sigue (ya sea porque se sigue un orden de izquierda a derecha o inverso) en el padre opuesto siempre y cuando este no pertenezca aún al hijo. En cuyo caso, se buscará al sucesor de ese gen existente.

Para la mutación, se aplica el método eugenésico, por el cual se va a dar un valor a cada gen de un individuo. Este valor será definido por la distancia a la que está cada gen con respecto a sus adyacentes. Después de eso, se intercambiarán de posición los 2 genes con peor valoración.



*Menú de Inicio*

## 2. Cruces y mutaciones

A continuación, se presentarán ejemplos de ejecución para los nuevos métodos de cruce y para los nuevos métodos de mutación. En general, se ha intentado que la ejecución sea representativa al comportamiento del método. Se ha utilizado una probabilidad de cruce del 60%, una probabilidad de mutación del 5% en todas las ejecuciones en orden de facilitar la comparación y una población de 300 individuos. No se ha hecho uso de elitismo. En algunos casos se ha ejecutado con 200 generaciones y en otras 300 para evidenciar la convergencia en caso de duda. Los ejemplos han utilizado el método de selección de ruleta.

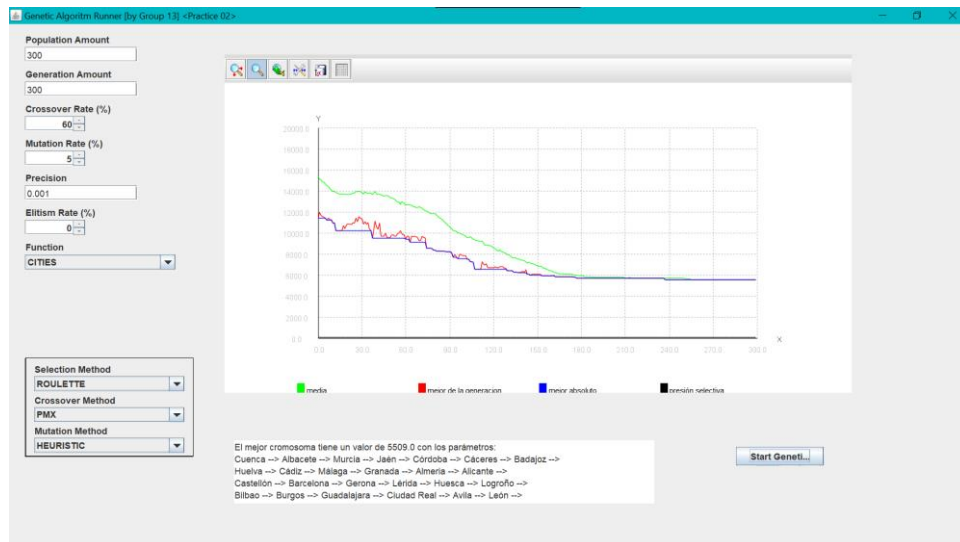
### 2.1 Cruces

Para los cruces hemos decidido utilizar el método de mutación heurístico.

#### **PMX**

Como podemos observar en la gráfica, la mejora es sustancial hasta la generación 100, estancándose a partir de esta. La media se estabiliza a partir de la generación 165 casi solapándose con el mejor de la generación y con el mejor absoluto por lo que la población ha llegado a la convergencia.

## Programación Evolutiva - Práctica 1



*Emparejamiento parcial junto a la mutación heurística*

### ERX

En la gráfica se observa que no se ha llegado a la convergencia al final de 200 generaciones puesto que hay una variación de unos 3000 puntos entre el mejor de cada generación a la media de cada generación. Además, a partir de la generación 153 hay cierto empeoramiento del mejor de la generación con respecto al mejor global no produciéndose ninguna mejora. El ERX posee la mayor diferencia entre el mejor generacional y el mejor absoluto de todos los métodos de cruce estudiados.

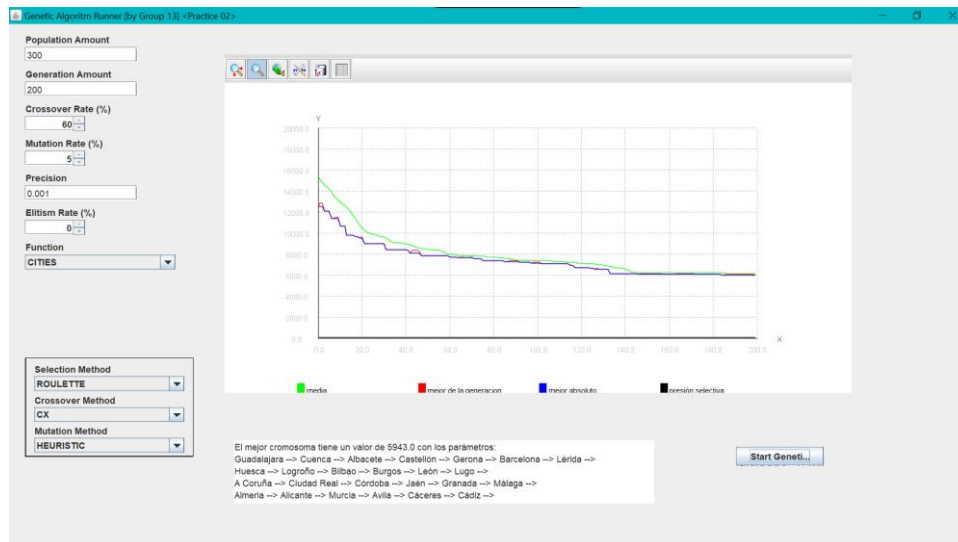


*Recombinación de rutas*

### CX

En esta gráfica observamos un decrecimiento tanto de la media, tanto del mejor absoluto. Es relevante el hecho de que el mejor generacional se solapa mayoritariamente con el mejor absoluto, lo que indica que hay una mejora continua. Por otra parte, no debe haber mucha diversidad puesto que en la generación 60 disminuye la distancia entre el mejor generacional y la media generacional a la mínima expresión. Podemos hablar de convergencia a partir de la generación 145 puesto que la media generacional se solapa con la media generacional, manteniéndose constantes en el tiempo.

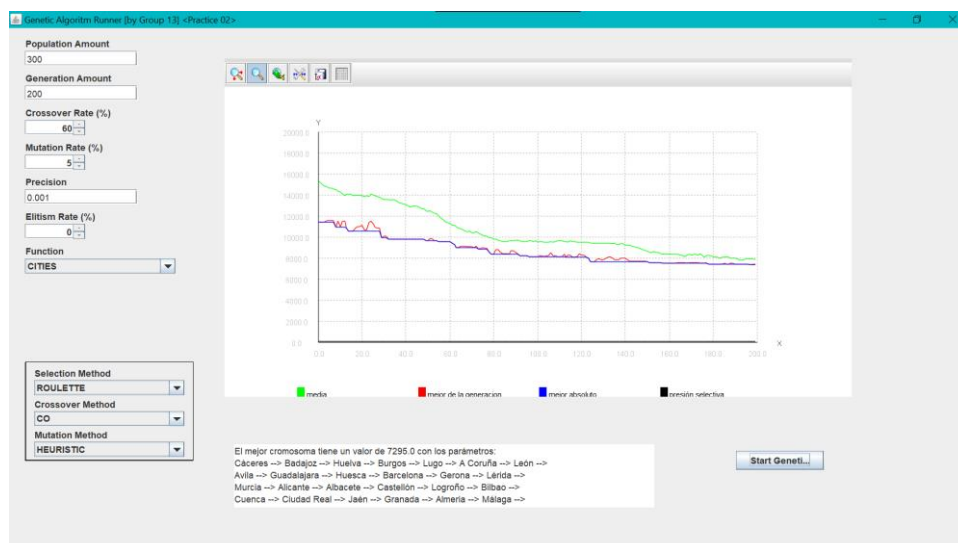
## Programación Evolutiva - Práctica 1



*Ciclos*

CO

En la gráfica en la que se evalúa la codificación ordinal, vemos una mejora constante, sobre todo entre las generaciones 40 y 80. Se llega a la convergencia, pero es necesario utilizar a las 200 generaciones para que la media se acerque significativamente al mejor absoluto.



*Codificación ordinal*

OX

Se observa convergencia media puesto que hay bastante diferencia entre el valor del mejor generacional con la media de la población, pero hay poca variación en el valor del mejor generacional puesto que no hay una mejora significativa del mejor generacional N.º 55 al N.º 300.

En torno a la generación número 200, hay una mejora general apreciable a lo largo de la población.



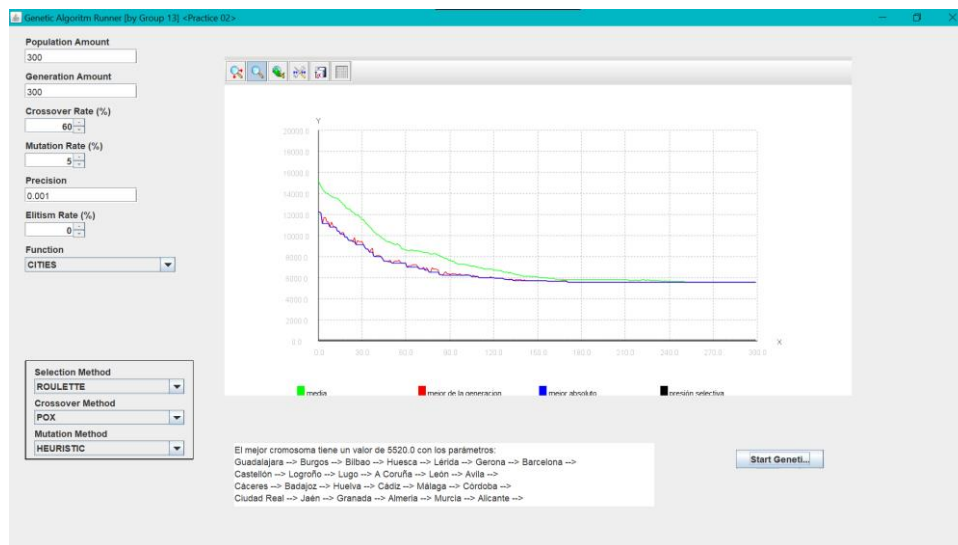
## Programación Evolutiva - Práctica 1



Orden

### POX

La gráfica es semejante a la observada en la ejecución del método de ciclos. Se diferencia con esta gráfica en que aumenta la diferencia de la media generacional y el mejor de cada generación en las primeras 100 generaciones. Pero al igual que en ciclos hay una rápida convergencia que si sitúa sobre la generación 170. Pese a la rápida convergencia no hablaría de convergencia prematura puesto que el resultado 5520 es muy bueno.



Orden prioritario

### AO (Método de mutación propio)

El método de orden alternativo en ocasiones no presenta mucha mejora a partir de la generación N.º 40, sin embargo, en esta gráfica podemos observar una mejoría con poca pendiente pero constante a lo largo de las generaciones analizando el mejor absoluto. La media mejora en gran medida entre la generación 120 y la 160, pero a partir de la 160 empeora significativamente. Pese a la complejidad del algoritmo (en *Explicación de métodos propios*, se detalla) es bastante peor al método de cruce de Orden.



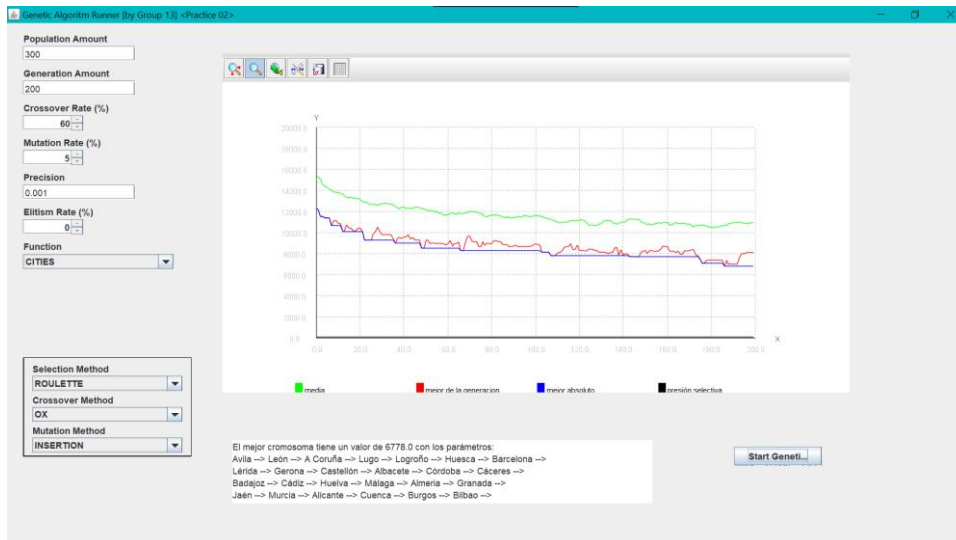
Orden alternativo (alternative order, método propio)

## 2.1 Mutaciones

Para las mutaciones hemos utilizado al método de orden (OX) como constante.

### Inserción

Con el método de inserción, vemos un resultado parecido al del resto de gráficas con mejora constante pero leve, aunque en este caso, parece que justo al final empeoran ligeramente los valores de los últimos mejores cromosomas. No hay convergencia observable dada la desviación de la media con respecto al mejor generacional.



Inserción

### Intercambio

Se aprecia una mejora a lo largo del primer tercio de la prueba, la cual se queda relativamente estancada más adelante. Hay una significativa diferencia entre la media y el mejor generacional a lo largo de la fase de estancamiento (desde la generación 50).



*Intercambio*

### Inversión

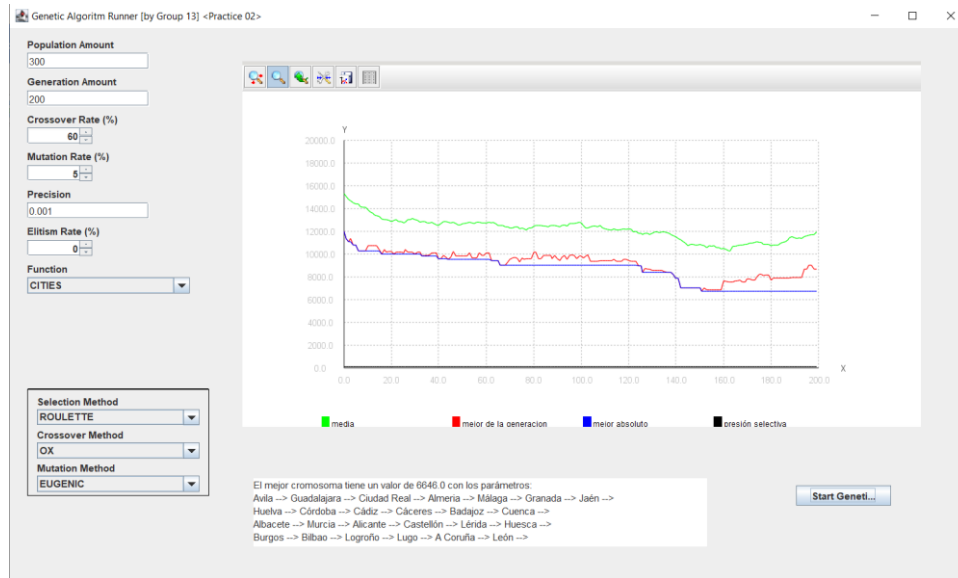
Pese a seguir el camino común en comparación con el resto de las pruebas, en este caso, tanto la media como los mejores individuos de las últimas generaciones se alejan del resultado ideal (empeoramiento de la media desde la generación N.º 150, dando a entender que hubiese sido necesaria un reinicio de la población).



*Inversión*

### Eugenesia (Método de mutación propio)

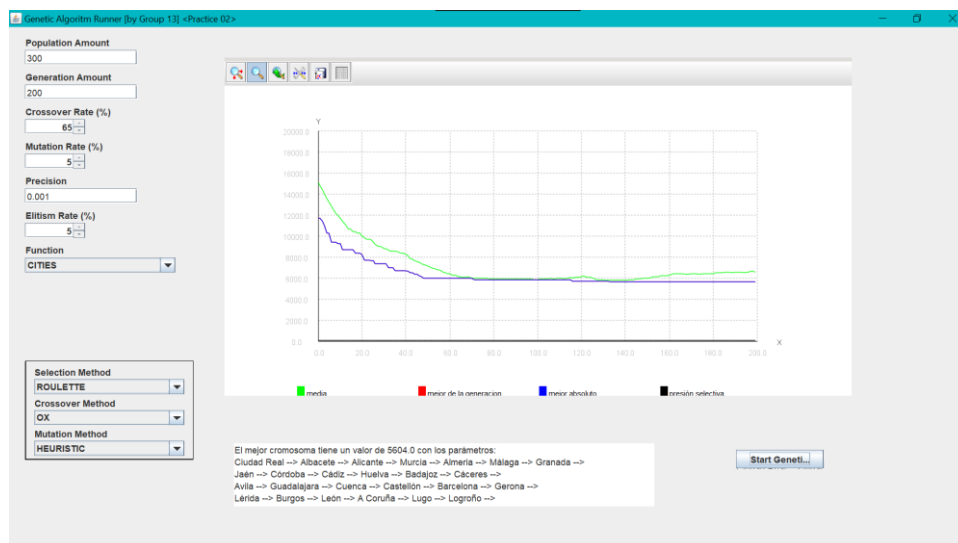
En este último caso, la media desciende con menor intensidad en comparación a otros ejemplos, sin embargo, incluso estando en la última generación, esta está cerca de alcanzar un mejor valor absoluto.



*Eugenesia (Método de mutación propio)*

## 3. Elitismo

Una pequeña muestra de elitismo, en torno al 5%, ha permitido mejorar considerablemente los resultados de la prueba, por lo que podemos decir que ha sido positivo conservar los mejores individuos a lo largo de la simulación, evitando que se pierdan por el camino



*Función Elitismo*

## 4. ¿Qué métodos de cruce y mutación son mejores?

Para valorar la calidad de cada método, hemos construido un programa anexo (paquete **statistics**) que ejecute automáticamente nuestra práctica con el fin de obtener estadísticas acerca de que combinaciones son mejores. Hemos variado las ejecuciones con respecto a los siguientes parámetros:

- Selección: Métodos de ruleta, ranking y probabilístico (70%).

- Cruce: Orden (OX), orden prioritario (POX), emparejamiento parcial (PMX), ciclos (CX), recombinación de rutas (ERX), codificación ordinal (CO) y el método propio, orden alternativo (AO).
- Mutación: Inversa, inserción, intercambio, heurístico y nuestro método propio, la mutación eugenésica.
- Elitismo: 0% y 5%.

Para toda ejecución del algoritmo se han usado 200 de población y tan solo 100 generaciones. Se ha obtenido la media de 10 ejecuciones por cada configuración (cuarteto). Los rankings se han obtenido de la media entre todas las ejecuciones de cada método.

	Ranking de cruce		Ranking de mutación	
1.	POX	7038	Heurística	7177
2.	OX	7185	Inversa	7418
3.	PMX	7269	Intercambio	7618
4.	ERX	7508	Eugenésica	8685
5.	AO	8206	Inserción	8686
6.	CO	8567		
7.	CX	9645		

*Ranking de métodos de cruce y mutación (más arriba, mejor resultado siendo menor la distancia en el TSP)*

El mejor método de cruce es el **POX** y el mejor método de mutación es el **heurístico**.

### MEJOR CONFIGURACIÓN (cuarteto)

Elitismo: **5%**. Método de cruce: **PMX**. Método de mutación: **Inversa**.

Método de selección: **Torneo probabilista** (grupos de 3 individuos, probabilidad del 70% de elegir al campeón).

Resultado: **5662.7** de media

```

421
422----- RANKING DE CRUCES -----
4231ª Posición: POX. Valor: 7038.52
4242ª Posición: OX. Valor: 7185.8
4253ª Posición: PMX. Valor: 7269.283333333335
4264ª Posición: ERX. Valor: 7508.546666666666
4275ª Posición: AO. Valor: 8206.433333333334
4286ª Posición: CO. Valor: 8567.820000000002
4297ª Posición: CX. Valor: 9645.766666666666
430-----
431
432----- RANKING DE MUTACIONES -----
4331ª Posición: HEURISTIC. Valor: 7177.595238095236
4342ª Posición: INVERSE. Valor: 7418.200000000003
4353ª Posición: EXCHANGE. Valor: 7618.599999999998
4364ª Posición: EUGENIC. Valor: 8685.933333333332
4375ª Posición: INSERTION. Valor: 8686.935714285717
438-----
439
440-----
441Con el elitismo al 5.0%, la selección PROBABILISTIC TOURNAMENT, el cruce PMX
442 y la mutación INVERSE hemos obtenido el mejor valor, el cual es 5662.7
443-----
444

```

*Resultados del test*

## 5. Mejoras

No hemos implementado ninguna mejora más allá de las ya realizadas en la Práctica 1.

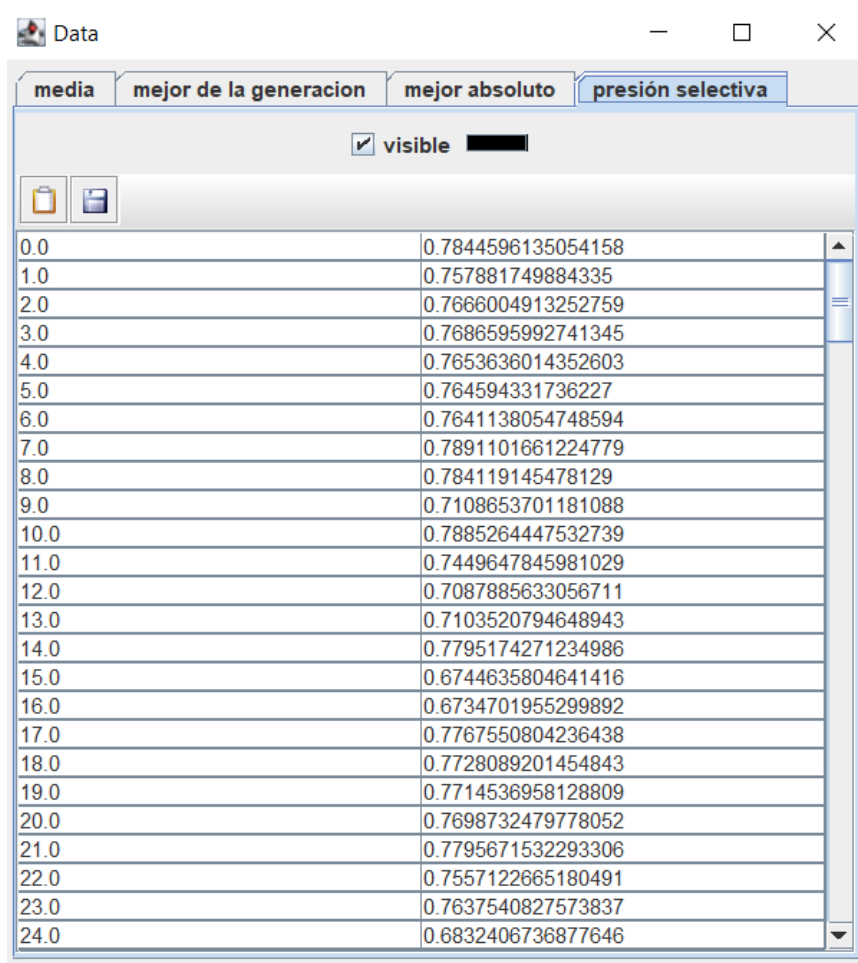
## 6. Análisis de Convergencia

A diferencia de la práctica 1, en este caso se puede apreciar un gran cambio de resultados entre las primeras iteraciones con respecto a las siguientes, reduciendo la velocidad de mejora cada vez que aumentan las generaciones. En la mayoría de los métodos tanto de mutación se mantiene la media bastante alejada del mejor generacional con excepción del heurístico que iguala sus valores en ciertas situaciones. En “2. Cruces y Mutaciones” se puede observar el análisis de convergencia para cada método estudiado.

## 7. Problemas / Curiosidades encontrados

Como curiosidad hemos observado que para todas las ejecuciones la presión selectiva está por debajo de 1.

Esta tabla ha sido generada ejecutando el método de selección ranking, con el método propio de cruce, orden alternativo y con el método de mutación heurística.



0.0	0.7844596135054158
1.0	0.757881749884335
2.0	0.7666004913252759
3.0	0.7686595992741345
4.0	0.7653636014352603
5.0	0.764594331736227
6.0	0.7641138054748594
7.0	0.7891101661224779
8.0	0.784119145478129
9.0	0.7108653701181088
10.0	0.7885264447532739
11.0	0.7449647845981029
12.0	0.7087885633056711
13.0	0.7103520794648943
14.0	0.7795174271234986
15.0	0.6744635804641416
16.0	0.6734701955299892
17.0	0.7767550804236438
18.0	0.7728089201454843
19.0	0.7714536958128809
20.0	0.7698732479778052
21.0	0.7795671532293306
22.0	0.7557122665180491
23.0	0.7637540827573837
24.0	0.6832406736877646

*Tabla de Presión selectiva*

## 8. Reparto de tareas

**Rafael Alonso:** Cruce por emparejamiento parcial, cruce por ciclos, cruce por recombinación de rutas, codificación ordinal, mutación por inversión, mutación por intercambio, mutación por inserción, mutación eugenésica (método propio), clase ConnectionTable (recombinación de rutas) y muestra de resultados por pantalla.

**Andrés Cardenal:** Métodos de cruce orden alternativo (método propio), orden y orden prioritario , método de mutación heurística, refactorización de Request y Builder más creación de Available (patrón ViewHelper de la vista), TravellerChromosomeChecker (comprobación de validez), clases MapIterator e Iterator (para los cruces de orden) e interfaces de mutación.

**Ambos (laboratorio y google meet):** StatisticsGenerator y StatisticsWriter (tablas de estadísticas que clasifican a los métodos de cruce y mutación), inicialización de población, gen CityGen, FunctionTSP (codificación del cromosoma y obtención del fenotipo).