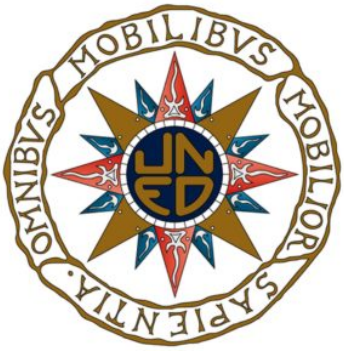
Máster en Investigación en Inteligencia Artificial

Computación evolutiva

Memoria Actividad 1

Autora:
Marta Rodríguez Sampayo



14 de diciembre de 2019

Índice

Índice de figuras

Índice de tablas

1. Introducción

Dentro de la computación evolutiva, la variante más ampliamente utilizada son los algoritmos genéticos. Estos son utilizados como métodos de optimización y búsqueda de soluciones, la información de cada solución se codifica en cadenas dentro de un alfabeto finito.

A lo largo de esta memoria se utilizarán con frecuencia los términos de exploración y explotación. La definición de la que se parte, extraída del libro base de la asignatura, es la siguiente:

- Exploración: generación de nuevos individuos en regiones aún no probadas del espacio de búsqueda.
- Explotación: concentración de la búsqueda en las proximidades de buenas soluciones conocidas.

En este documento se presenta la implementación de un algoritmo genético aplicado a la búsqueda de un óptimo que sirva como solución al problema del viajante. Tras la descripción del problema en la Sección 2, se definen las características particulares del algoritmo genético implementado en la Sección 3. Estas características son comunes a todo algoritmo genético y se presenta la solución específica utilizada para cada una de ellas, junto a una breve explicación del código desarrollado y la estructura del mismo, para facilitar su comprensión. Para poder realizar un análisis de la calidad de la implementación del algoritmo genético es necesario realizar una evaluación experimental del mismo, en este caso se realizan varias medidas de las prestaciones del AG para instancias concretas del problema con un cierto conjunto de parámetros, expuestas en la Sección 4. Finalmente, en la Sección 5 se presentan las conclusiones del trabajo realizado. A mayores, se añade un Anexo A donde se explica la estructura de ficheros, con el fin de realizar una correcta ejecución del programa

2. Descripción del problema del viajante

El problema del viajante o TSP (Travelling Salesman Problem) plantea un escenario con cierto número de ciudades y las distancias entre las mismas, donde se debe encontrar la ruta de distancia mínima que permita recorrerlas todas sin repetir ninguna de ellas, siendo esta ruta cíclica (comenzando y terminando en la misma ciudad).

Este problema de optimización, con sus múltiples variantes, es ampliamente conocido gracias a sus aplicaciones en campos muy diversos (planificación, logística, fabricación de circuitos electrónicos, secuenciación de ADN, etc.). Es por ello que resulta un caso de estudio muy interesante en el área de aplicación de los algoritmos evolutivos.

3. Implementación

A continuación, se describe la implementación del algoritmo genético propuesto para la obtención de un valor óptimo que sirva como solución al problema del viajante. Se presentan por separado cada una de las características de dicho algoritmo y la implementación escogida. El lenguaje de programación utilizado es Java.

3.1. Inicialización

Para la ejecución del programa son necesarios dos archivos de texto:

- GAParameters.txt: en este archivo se definen los parámetros necesarios para el funcionamiento del algoritmo, separados con comas.
  - Tamaño de la población: número entero.
  - Probabilidad de cruce: número decimal entre 0.5 y 1.0
  - Tamaño del torneo: número entero, menor o igual al tamaño de la población.

- Probabilidad de mutación: número decimal. En caso de no aparecer de forma explícita en el archivo, se calcula como 1/tamaño de la población.

- TSP-X.txt: definición de la instancia del TSP, en cada línea se especifican las coordenadas de cada ciudad. La X del nombre del archivo hace referencia al número de ciudades de una instancia determinada. Los archivos utilizados son: TSP-15, TSP-100, TSP-280 y TSP-2392.

Al comienzo de la ejecución se selecciona una instancia del TSP y se leen ambos archivos, creando un mapa donde se asocian las coordenadas de cada ciudad con un identificador entero (las coordenadas de la primera línea del archivo se asociarán con la ciudad con identificador '1' y así sucesivamente) y un objeto con los parámetros, que serán utilizados posteriormente.

3.2. Representación de cada individuo

El genotipo de cada individuo está asociado a una permutación de las ciudades, de forma que cada individuo se representa como una cadena de texto con la permutación de los identificadores de las ciudades.

3.3. Función de adaptación

La función de adaptación utiliza la distancia euclidiana entre ciudades a partir de sus coordenadas y asigna a cada individuo la longitud del recorrido asociado a su fenotipo. Un individuo tendrá mejor valor de adaptación cuánto menor sea dicha longitud.

3.4. Inicialización de la población

Una vez iniciada la ejecución y leídos los archivos con los parámetros del algoritmo, se crea la población inicial, de forma aleatoria y sin repetir ninguna ciudad en un mismo recorrido. Se crea un nuevo mapa, que representa un individuo, con valores aleatorios comprendidos entre los valores máximo y mínimo de las claves del mapa en el que se encuentran todas las ciudades; las claves de este mapa se convierten a cadena de texto y se almacenan en una lista con todos los individuos de la población.

3.5. Selección de padres

Se realiza mediante selección por torneo. En primer lugar, se escogen de la lista que representa a la población k candidatos aleatoriamente (el parámetro k se especifica en el archivo GAParameters.txt), una vez seleccionados, se calcula su valor de adaptación y el individuo con mejor valor de adaptación se almacena en otra lista como padre. Este proceso se repite hasta obtener tantos padres como el número de individuos de la población.

Se realiza esta selección con reemplazo, de forma que el individuo con mejor valor de adaptación de la población tiene mayor probabilidad de ser escogido varias veces como padre.

3.6. Cruce

Se realiza mediante *partially mapped crossover* con una probabilidad de cruce dada. El primer paso es seleccionar aleatoriamente dos padres entre la lista obtenida tras la selección por torneo y generar un número aleatorio comprendido entre 0 y 1, si este número es menor que la probabilidad de cruce del algoritmo (parámetro especificado en el archivo GAParameters.txt) se realiza el cruce, en caso contrario, se crean dos hijos idénticos a los dos padres seleccionados.

En el caso de que el número aleatorio sea mayor que la probabilidad de cruce, se generan dos números aleatorios comprendidos entre 0 y el número de ciudades del problema - 1, estos serán los puntos de corte y se realiza el cruce tal y como está definido en el libro base de la asignatura.

3.7. Mutación

Se realiza mediante *swap mutation* con una probabilidad de mutación dada. Por cada individuo de la población se genera un número aleatorio, si este número es menor que la probabilidad de mutación (parámetro especificado en el archivo GAParameters.txt), dicho individuo sufre mutación. En este caso, se selecciona aleatoriamente dos posiciones en la permutación asociada al individuo y se intercambian.

3.8. Selección de supervivientes

Se utiliza un modelo generacional, reemplazando toda la población actual por otra nueva. Además, se aplica elitismo, así el mejor individuo en la población actual permanece en la siguiente generación. Salvo si existe un individuo en la nueva población con igual o mejor valor de adaptación; de ser así, el individuo a reemplazar en la nueva población será el de menor valor de adaptación.

3.9. Estructura del código

El código está dividido en dos paquetes principales, en uno de ellos se encuentran las clases con los métodos que implementan las características del algoritmo genético y en el otro las clases necesarias para la correcta ejecución del programa.

- **GeneticAlgorithm**
  - **Selection.java**: incluye los métodos para llevar a cabo la selección de padres y la selección de supervivientes.
  - **Variation.java**: incluye los métodos para llevar a cabo la mutación y el cruce.
- **TSP_EC**
  - **Main.java**: clase principal del proyecto, en el método main() se ejecutan los pasos del algoritmo genético. El orden es el siguiente: lectura de ficheros con los datos de entrada, inicialización de la población, mutación, selección de padres, cruce y selección de supervivientes. Este proceso se repite o bien hasta que se alcance el óptimo global (para instancias del TSP donde este valor es conocido) o bien hasta un número determinado de generaciones. También se incluyen en este archivo un método para calcular el valor de adaptación de un individuo y un método para obtener el individuo con mejor valor de adaptación de la población.
  - **City.java**: objeto que representa a una ciudad según sus coordenadas. Incluye un método para calcular la distancia euclidiana entre dos ciudades.
  - **GAParametres.java**: objeto donde se almacenan los parámetros del algoritmo genético.
  - **Evaluation.java**: incluye los métodos para calcular SR, MBF y AES.
  - **FileManagement.java**: clase para lectura y escritura de ficheros.
  - **Utils.java**: incluye métodos para transformar las representaciones de individuos de un tipo a otro, adaptándola a las necesidades del código.

3.10. Evaluación

Durante la ejecución del programa se crean distintos ficheros de texto para poder realizar posteriormente la evaluación.

- **times.txt**: tiempo que tarda en ejecutarse el algoritmo y mejor valor de adaptación obtenido en cada ejecución.
- **evaluation.txt**: *Success Rate*, *Mean Best Fitness*, óptimo conocido para la instancia en cuestión del TSP y los parámetros utilizados.
- **progressCurvesX.txt**: tras cada generación se guarda el mejor valor de adaptación, para crear posteriormente las curvas de progreso. X es el número de ejecución.

Por motivos de comodidad a la hora de programar la implementación y para una mayor claridad del código, el procesado de dichos ficheros, necesario para la correcta evaluación del algoritmo, se ha implementado en scripts escritos con el lenguaje de programación Python.

- **Evaluation.py**: crea una gráfica con la curva de progreso, promediando los archivos progressCurvesX.txt obtenidos de todas las ejecuciones del programa.
- **AverageExecutionTime.py**: calcula el tiempo medio de ejecución, a partir del fichero times.txt
- **BestWorstFitness.py**: calcula el mejor y el peor valor de adaptación de entre todas las ejecuciones.
- **ProgressCurves.py**: genera en una misma gráfica las curvas de progreso de distintas ejecuciones del algoritmo con diferentes parámetros.

4. Evaluación experimental

A la hora de llevar a cabo la evaluación del algoritmo genético implementado se ha optado por utilizar instancias del problema con un óptimo conocido. De esta forma, el problema sencillo consta de 100 ciudades y el problema complejo de 2392, ambas extraídas del dataset de TSPLIB (kroA100 y pr2392).

Las medidas tomadas son el mejor fitness promedio, el mejor y el peor fitness y el tiempo medio de ejecución. Además se representan las curvas de progreso del algoritmo para cierto número de ejecuciones. Debido a que en ninguno de los dos problemas, sencillo y complejo, se consigue alcanzar el óptimo global, no se calculan la tasa de éxito ni el AES(*Average number of Evaluations to a Solution*).

4.1. Problema sencillo

Tal y como se ha comentado anteriormente, esta instancia del TSP define un escenario de 100 ciudades. Los parámetros del algoritmo genético se han elegido según los siguientes criterios:

- Tamaño de la población: 10; recomendado para la actividad.
- Probabilidad de cruce: puesto que el análisis de los resultados se debe hacer según este parámetro, se varía para cada experimento. Se utilizan los valores recomendados en el libro base de la asignatura, por tanto esta probabilidad se encuentra en el intervalo [0.5, 1.0].
- Tamaño del torneo: 10; para la instancia sencilla, teóricamente, se deben alcanzar mejores resultados, en cuanto a eficiencia, aplicando mayor explotación, por ello se fija este parámetro al mismo valor que el tamaño de la población.
- Probabilidad de mutación: 0.1; siguiendo las indicaciones del libro, se fija esta probabilidad al valor de la inversa del tamaño de la población, de forma que, en media, mutará un individuo en cada población.

Teniendo en cuenta el tiempo de ejecución y los resultados obtenidos, el número de generaciones en cada ejecución es 20.000 y se realizan 100 ejecuciones sobre las que se toman las medidas de prestaciones.

4.1.1. Resultados

El óptimo conocido para esta instancia del problema es 21282, sin embargo, con el algoritmo propuesto no se ha podido alcanzar dicho valor. Con el número de generaciones planteado, el algoritmo converge en todas las ejecuciones a un óptimo local comprendido entre 35.000 y 45.000, aproximadamente. La evaluación del algoritmo para esta instancia se encuentra resumida en la Tabla 1.

También en la Figura 1 se muestra el progreso del valor de adaptación de las distintas variantes del algoritmo a lo largo de las generaciones. Debido al elevado número de generaciones y al comportamiento

Probabilidad de cruce	MBF	Mejor fitness	Peor fitness	Tiempo medio de ejecución (s)
0.5	43155.36	34602	170768	37.86
0.6	43835.02	35372	167668	39.29
0.7	43375.89	35712	168978	36.07
0.8	43553.55	35887	167569	38.58
0.9	43872.51	35540	170066	37.72
1.0	44207.73	35178	170934	38.46

Tabla 1: Evaluación del problema de 100 ciudades

del algoritmo para este problema más sencillo, aunque cada curva está representada en un color diferente, correspondiente al valor de la probabilidad de cruce, estas son prácticamente indistinguibles.

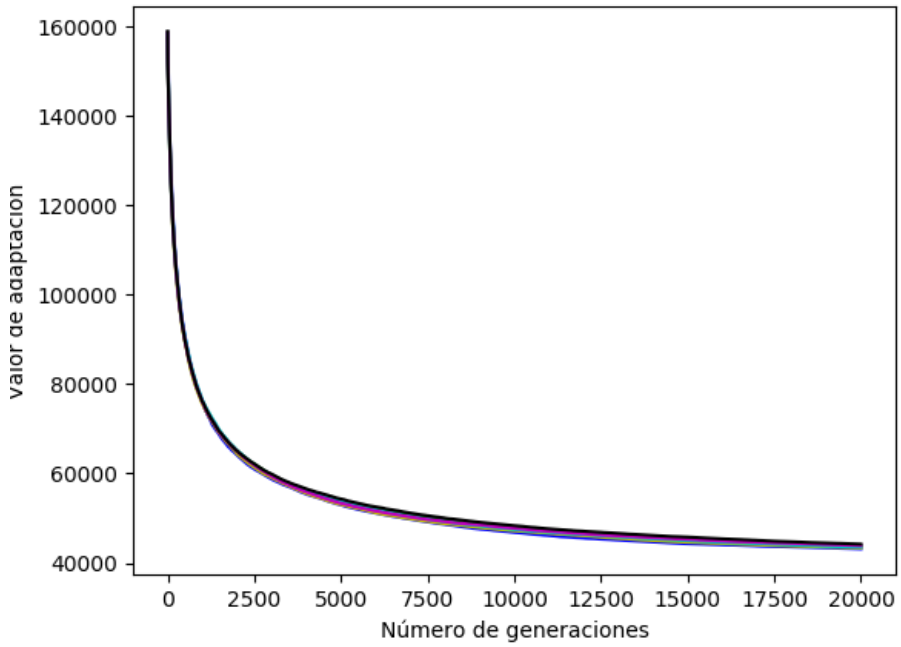


Figura 1: Curvas de progreso. Instancia de 100 ciudades, 100 ejecuciones.

A la vista de estos resultados, se puede concluir que el aumento de la probabilidad de cruce en esta instancia del problema no beneficia al cálculo del óptimo local. Además, al tratarse de tiempos medios de ejecución relativamente bajos, este parámetro no afecta a la eficiencia del sistema de forma significativa.

4.2. Problema complejo

Para la instancia compleja también se ha utilizado un problema con un óptimo global conocido. Se han realizado los experimentos con los siguientes parámetros:

- Tamaño de la población: 100; recomendado para la actividad.
- Probabilidad de cruce: siguiendo el mismo razonamiento que en el problema sencillo, esta probabilidad se encuentra en el intervalo [0.5, 1.0].
- Tamaño del torneo: 20; en el caso de la instancia compleja, teóricamente, se deben alcanzar mejores resultados, en cuanto a eficacia, aplicando mayor exploración, por tanto, tras comparar los resultados de distintas ejecuciones, se escoge este valor.

- Probabilidad de mutación: 0.01; de nuevo se aplicará mutación a un individuo en cada generación.

Puesto que el número de ciudades es considerablemente mayor, de la misma forma que las distancias de los recorridos, se ha escogido como condición de terminación un número de generaciones que cumpla un compromiso entre eficacia y tiempo de ejecución. Por tanto, se han realizado 40 ejecuciones con 300 generaciones cada una.

4.2.1. Resultados

En este caso, la distancia mínima conocida para recorrer las 2392 ciudades es 378032. Debido al elevado tiempo de ejecución, el algoritmo no llega a converger a un óptimo local, quedando el valor alcanzado en la última generación muy por encima del óptimo global conocido. Sin embargo, los resultados obtenidos son suficientes para apreciar el comportamiento del algoritmo para esta instancia del problema.

Probabilidad de cruce	MBF	Mejor fitness	Peor fitness	Tiempo medio de ejecución (s)
0.5	14025980.575	13733880	15011217	396.07
0.6	13945079.175	13582493	14975592	396.27
0.7	13858767.2	13583265	15015784	405.69
0.8	13820480.875	13583423	14985276	406.82
0.9	13759930.45	13431000	15015303	417.71
1.0	13725980.55	13334053	15019888	421.87

Tabla 2: Evaluación del problema de 2392 ciudades

En la Tabla 2 se puede apreciar el efecto de la variación de la probabilidad de cruce en los distintos experimentos. Por un lado, al aumentar su valor, aumenta el tiempo medio de cada ejecución y, por otro, el valor del MBF y del mejor fitness obtenido disminuyen.

En la Figura 2 se visualiza este comportamiento. Se pueden observar las curvas de progreso de las seis variantes del algoritmo, desde la azul (0.5) a la negra (1.0). A medida que aumenta la probabilidad de cruce, el descenso inicial del valor de adaptación es más pronunciado y se alcanzan valores menores al cumplirse la condición de terminación.

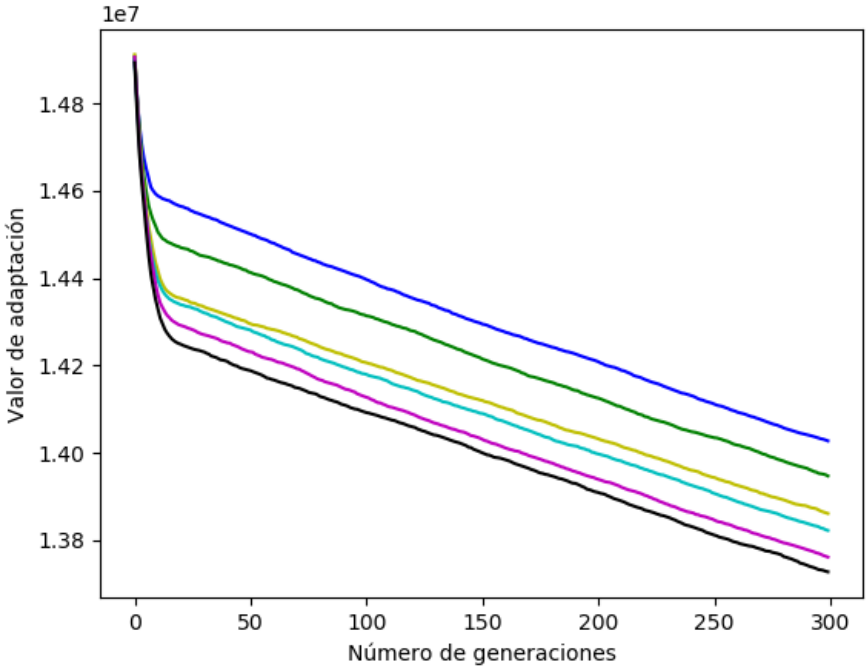


Figura 2: Curvas de progreso. Instancia 2392 ciudades, 40 ejecuciones. (Azul: 0.5, Verde: 0.6, Amarillo: 0.7, Cián: 0.8, Rosa: 0.9, Negro: 1.0)

En base a los resultados obtenidos, se puede afirmar que la eficacia del algoritmo aumenta con el valor de la probabilidad de cruce. Es decir, aplicando más explotación, la calidad del resultado obtenido es mayor. Sin embargo, a la vista de los tiempos medios de ejecución, la eficiencia desciende. Esto es debido a la realización del método de cruce, al aumentar la probabilidad de que se produzca, son más las operaciones que debe realizar el algoritmo sobre una población en cada generación, afectando esto al rendimiento del mismo.

Con el objetivo de mostrar mejor el comportamiento del algoritmo, se han llevado a cabo dos ejecuciones (para valores de la probabilidad cruce de 0.5 y 1.0) de 4000 generaciones (Figura 3)

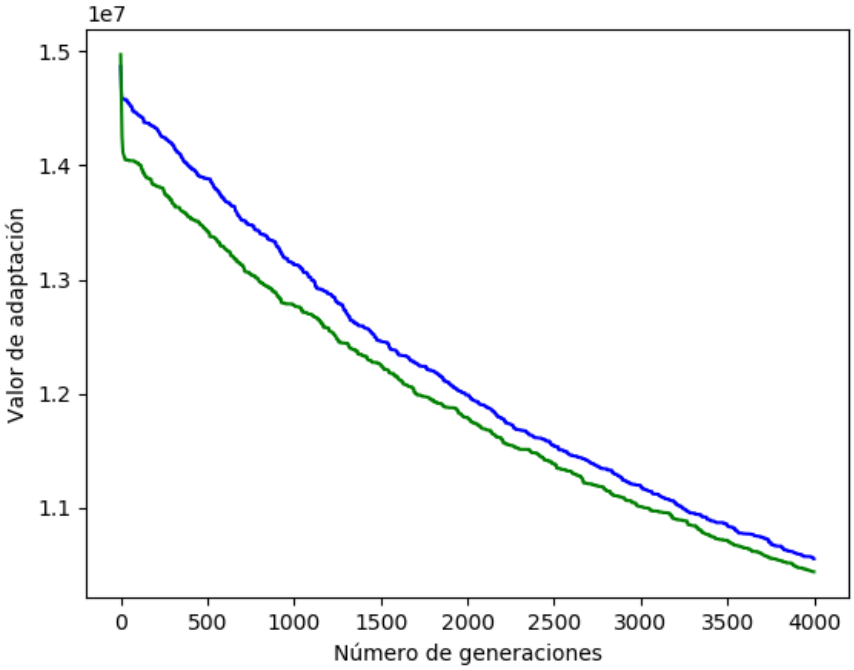


Figura 3: Curvas de progreso para una única ejecución (Azul: 0.5, Verde: 1.0)

Al ser el resultado de ejecuciones aisladas, los valores no están suavizados y por ello existen picos en las gráficas. No se pueden extraer conclusiones estadísticamente relevantes, pero estos resultados no contradicen la evaluación del algoritmo durante 300 generaciones.

4.3. Comparativa

Al aumentar la complejidad del problema se deben adaptar las características del algoritmo genético, pero también se debe tener en cuenta el sistema físico en el que se lleven a cabo los experimentos. Aunque hubiese sido deseable utilizar la misma condición de terminación para ambas instancias del problema, los resultados no habrían sido igual de descriptivos y se ha priorizado la evaluación individual de cada problema frente a la comparativa entre ambos.

Con el objetivo de realizar una comparación más visual se han realizado dos nuevos experimentos, ambos de 40 ejecuciones del algoritmo genético aplicado al problema sencillo (mismos parámetros especificados anteriormente) y 300 generaciones, uno con una probabilidad de cruce de 0.5 y el otro de 1.0. Los resultados se pueden observar en la Tabla 3 en contraposición a los obtenidos anteriormente para la instancia compleja del problema.

Ciudades	P. de cruce	MBF	Mejor fitness	Peor fitness	Tiempo medio de ejecución (s)
100	0.5	101142.875	89696	169040	0.5919
100	1.0	100590.025	92791	167301	0.5762
2392	0.5	14025980.575	13733880	15011217	406.82
2392	1.0	13725980.55	13334053	15019888	417.71

Tabla 3: Problema sencillo vs. problema complejo

Las diferencias más destacables entre ambas instancias son los tiempos de ejecución, la calidad de la solución encontrada y los efectos al variar la probabilidad de cruce.

- Tiempo de ejecución: al aumentar el tamaño del problema, se aumenta la longitud de las permutaciones que representan a cada individuo y, en este caso, el número de individuos de la población. Es razonable deducir que el algoritmo reducirá su eficiencia al tener que trabajar con cantidades de datos más elevadas. Así lo muestran los resultados, con tiempos de ejecución muy superiores para la instancia compleja.

- Calidad de la solución: al aplicar diferentes condiciones de terminación, no es lógico comparar de forma directa la calidad del óptimo local alcanzado, puesto que para igualar el número de generaciones sería necesaria una ejecución durante un gran número de horas del algoritmo implementado para la instancia compleja. Sin embargo, observando la evolución de ambos problemas, se puede afirmar que el algoritmo alcanza una solución mejor durante un menor número de generaciones para la instancia sencilla. Basándonos en los datos de la Tabla 3, el mejor fitness obtenido en las 40 ejecuciones del problema sencillo es, aproximadamente, 4 veces mayor que el óptimo global conocido, mientras que, para la instancia compleja, es del orden de 35 veces más elevado.

- Variación de la probabilidad de cruce: aunque se han explicado en cada apartado correspondiente los efectos al variar este parámetro, a modo de resumen, al aumentar esta probabilidad se genera un mayor número de descendientes diferentes a los padres utilizados en el método de cruce, de forma que se exploran nuevas posibles soluciones y esto beneficia a los problemas de mayor tamaño ya que hay un número mucho más elevado de permutaciones posibles entre las que encontrar un óptimo. Al contrario, en la instancia con un menor número de ciudades, al reducirse el espacio de búsqueda de soluciones, resulta mucho más eficiente explotar los individuos con mejor valor de adaptación ya conocido. En cuanto a cómo afecta este parámetro al tiempo de ejecución, los cambios en el problema sencillo no son significativos, mientras que en el problema complejo, salta a la vista que al aumentarla se requiere más tiempo para ejecutar el algoritmo.

5. Conclusiones

En definitiva, el problema del viajante es un conocido problema de optimización con múltiples aplicaciones y posibles métodos para encontrar una solución válida, siendo la implementación de un algoritmo genético una de ellas.

Tras realizar el desarrollo de este sistema y evaluar los resultados en ambas instancias del problema, se pueden deducir las siguientes conclusiones:

1. A la hora de implementar un algoritmo genético se debe tener en cuenta el tipo del problema para el cual se requiere una solución y el tamaño del mismo. Es claro, por tanto, que para distintas instancias del TSP el conjunto de parámetros se debe adaptar a dichas condiciones.

2. En el caso de la instancia sencilla escogida para el desarrollo de esta actividad (100 ciudades) aumentar el grado de exploración no mejora la calidad de las soluciones ni la velocidad con la que se obtienen las mismas.

3. Por otro lado, para el problema complejo (2392 ciudades), es clara la mejoría al aplicar mayor explotación, en términos de eficacia, ya que el valor del recorrido disminuye a medida que se aumenta la probabilidad de cruce. Al tratarse de un problema de dimensiones mayores, el rendimiento global del sistema es mucho menor, aumentando los tiempos de ejecución y limitando el tamaño de los experimentos (número de ejecuciones y generaciones).

4. Es relevante comentar el rendimiento general del algoritmo, ya que, aunque hubiese sido deseable, no se ha alcanzado el óptimo global en la instancia sencilla del problema. La realización de distintos experimentos variando los parámetros, lleva a concluir que con el algoritmo genético desarrollado, partiendo de las características requeridas para esta actividad, no resulta posible alcanzar dicho óptimo. Esto puede ser debido a la instancia seleccionada o a que las características del propio algoritmo no son las adecuadas para dicha instancia en particular. Sí se han alcanzado soluciones óptimas para problemas de menor tamaño (entre 10 y 20 ciudades).

A. Entrega

A.1. Estructura de ficheros

- **doc/**: documentación sencilla de la aplicación (generada con javadoc).
- **Evaluation/**: esta carpeta debe situarse en el mismo directorio que el archivo .jar para la correcta ejecución del programa. En ella se almacenarán tras la ejecución del programa los archivos necesarios para realizar la evaluación del algoritmo, también se encuentran los scripts de python que completan la evaluación.
- **ProblemFiles/**: esta carpeta también debe situarse en el mismo directorio que el archivo .jar. En ella se encuentran 4 archivos de texto con diferentes instancias del problema del viajante y otro archivo con los parámetros del algoritmo. En caso de querer utilizar otros ficheros, también deben incluirse en esta carpeta.
- **src/**: código del programa, cuya estructura se explica en la Sección 3.9.
- **TSP.jar**: archivo ejecutable del programa.

A.2. Instrucciones de ejecución

1. Es posible ejecutar el programa introduciendo los siguientes argumentos directamente desde un terminal:
   - Nombre del archivo donde se encuentren las coordenadas de las ciudades (debe estar situado en la carpeta ProblemFiles)
   - Nombre del archivo donde se encuentren los parámetros del algoritmo genético (situado en la carpeta ProblemFiles)
   - Número de ejecuciones del algoritmo que se desea realizar.
   - Número máximo de generaciones que se utilizará como condición de terminación.
   - Un 1 en caso de que se desee realizar la evaluación o un 0 en caso contrario.

   En caso de no introducir los 5 argumentos, pero sí alguno, el programa no se ejecutará.

   Ejemplo de ejecución: java -jar TSP.jar TSP-100.txt GAParameters.txt 10 200 1
2. En caso de ejecutar el programa sin argumentos se abrirá un menú en la consola que pedirá sucesivamente los valores necesarios para la ejecución.