**MEMORIA**

**PRÁCTICA 4**

**ADINA HAN**

**YOUSSEF EL FAQIR EL RHAZOUI**

1. **NQueens**

La función de fitness está en la clase interna NQueensFitnessFunction de NQueensGenAlgoUtil que implementa la interfaz FitnessFunction. La función de fitness está relacionada con la variable booleana nonAttackingPair que se pone a false si hay parejas de reinas que se ataca y si no hay se le suma 1 a la función de fitness.

La función objetivo esta implementada en la clase NQueensGoalTest que implementa la interfaz GoalTest. El objetivo es representado por un valor booleano que devuelve el método isGoalState que indica si en el tablero de las n reinas no hay ninguna pareja de reinas que se ataquen.

La población inicial se genera mediante el método generateRandomIndividual, que devuelve una representación de los individuos. Los individuos se representan mediante una lista de enteros del tamaño del tablero. Esta lista está formada por enteros comprendidos entre 0 y el tamaño del tablero y son generados aleatoriamente.

El problema de las n reinas se representa en un tablero, que es un array bidimensional de enteros de 8x8 y las reinas se representa con el entero “1”, mientras las posiciones que están libres con “0”. La población está representada por un conjunto de individuos de tamaño 50 y la representación de los individuos se hace mediante una lista de enteros con tamaño igual al tamaño del tablero.

El algoritmo genético utiliza la selección por ruleta y esta implementada en el método randomSelection de la clase GeneticAlgorithm. Este método devuelve el individuo que cumple que al escoger un numero aleatorio, este sea menor que el valor de la función de fitness acumulado para cada individuo.

Una vez que estén seleccionados los individuos, se realiza el cruce para producir nuevos individuos que se insertaran en la siguiente generación. Se utiliza el cruce de un punto y la representación del hijo está dada por una parte que representa la cromosoma de uno de los padres desde el principio hasta ese punto que se ha elegido aleatoriamente, a la que se le concatena otra parte de la cromosoma del otro padre desde ese punto elegido hasta el final. El algoritmo genético inserta los hijos en la siguiente generación sin tener en cuenta si son mejores o no que sus padres porque utiliza el cruce de tipo destructivo.

El tipo de mutación que utiliza aima es el reemplazo aleatorio de un punto. La mutación se implementa en el método mutate de la clase GeneticAlgorithm y devuelve un individuo de la población después de que se le ha aplicado la mutación. Este método elige aleatoriamente el individuo a mutar de la población y también se elige aleatoriamente el valor por el cual será reemplazado.

Los valores que se pueden configurar para el problema de las n reinas en el algoritmo genético son: el tamaño del tablero, que en nuestro caso es 8, la probabilidad de mutación para el cual se ha elegido el valor de 0.15, el tamaño de la población se ha establecido en 50 y el tiempo máximo en milisegundos durante el cual el algoritmo se puede ejecutar (1000mls).

1. **Asignación de turnos a los exámenes CFI**

La población está constituida por un conjunto de individuos. Cada uno de estos tiene una representación, que es una lista de String y un entero, que sería el número de descendientes que tiene cada uno de los individuos. La posición que tiene cada uno dentro de la lista es el turno que le toca.

La generación de la población inicial se hace aleatoriamente, pero cada uno de los individuos que se genera tiene que ser un individuo válido y el número de turnos asignados coincide con el número de turnos necesarios expresados en el archivo. Para que un individuo sea válido tiene que cumplir las restricciones que aparecen en el archivo (esta comprobación la realiza la función testRestrictions de la clase TimetableGenAlgoUtil.java). El tamaño de la población la hemos fijado en 50.

El operador de cruce que hemos implementado es el de un punto, elegido aleatoriamente. Independientemente de este hecho el individuo resultante después del cruce tiene que ser válido. Una vez encontrado un individuo valido se introduce en la siguiente generación sin tomar en cuenta la función de fitness del hijo, por lo tanto hemos utilizado la estrategia destructiva que permite que tanto los mejores, como los peores individuos pasen a la siguiente generación.

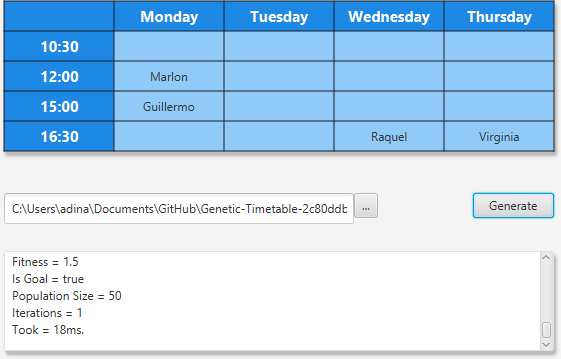
El tipo de mutación que hemos implementado para el algoritmo genético básico ha sido el reemplazo aleatorio. Este método consiste en elegir un gen (ya que la mutación utilizada es de un punto) del cromosoma aleatoriamente y cambiarlo por otro valor. El individuo que devuelve el método mutate siempre es un individuo valido y no hace falta comprobarlo.

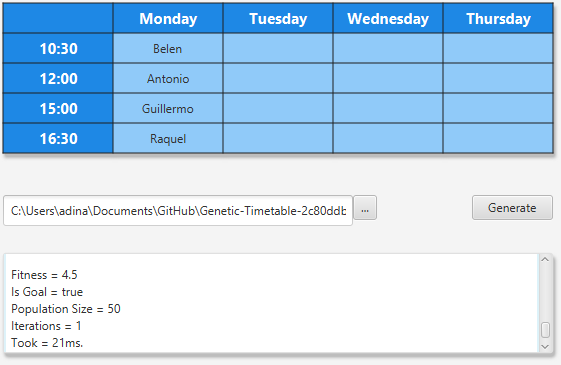
La función de fitness esta implementada en la clase interna TimetableFitnessFunction de TimetableGenAlgoUtil.java. El valor de la función de fitness es la resta entre las preferencias del individuo y las repeticiones. Al número total de preferencias de cada individuo le sumamos 0.5, ya que premiamos esto. Si para distintos turnos se repiten profesores entonces el número de repeticiones lo penalizamos multiplicándole 0.5, lo que va a hacer que disminuya la función de fitness. Si el valor que devuelve la función testRepetitions es -1 significa el que individuo no es válido porque se le ha asignado un turno que estaba en las restricciones.

La función objetivo determina si un individuo es objetivo o no. Un individuo es objetivo si se cumplen las restricciones indicadas para el con respecto a los turnos y también que el número de profesores que se repite en un individuo sea distinto de -1 (ya que eso significaría que se ha incumplido una restricción).

Con la implementación básica del algoritmo genético obtenemos los siguientes datos:

Convocatoria 1: para el primer archivo hemos obtenido una media de la función de fitness de 2.38. El valor mínimo de esta de 1.5

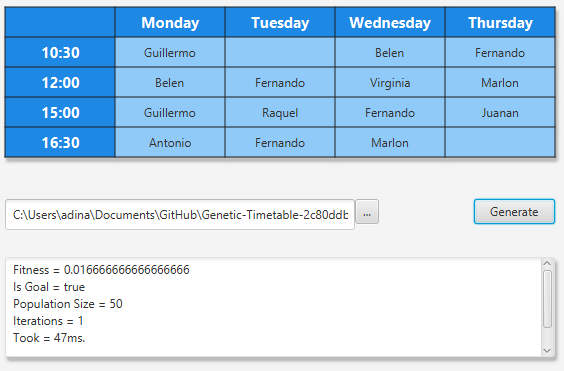


El valor máximo de la función de fitness es de 4.5. 

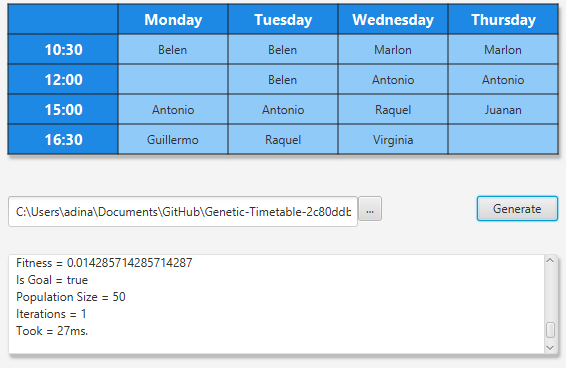
El tiempo máximo que ha tardado el algoritmo en ejecutarse ha sido 194 ms y el mínimo 16ms.

Convocatoria 2: para el segundo archivo la media es 0.015.

La función de fitness tiene como valor máximo 0.016.

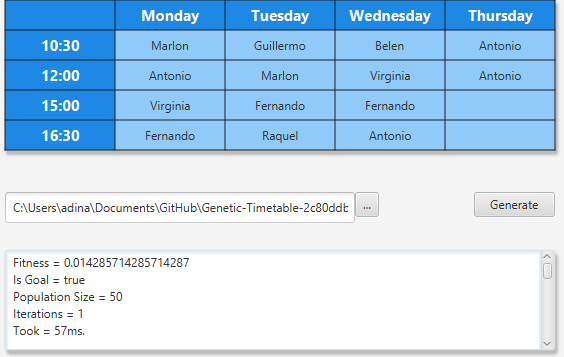


Y como valor mínimo tiene 0.14.



El tiempo máximo en el cual el algoritmo termina es 71 ms y el mínimo 12 ms.

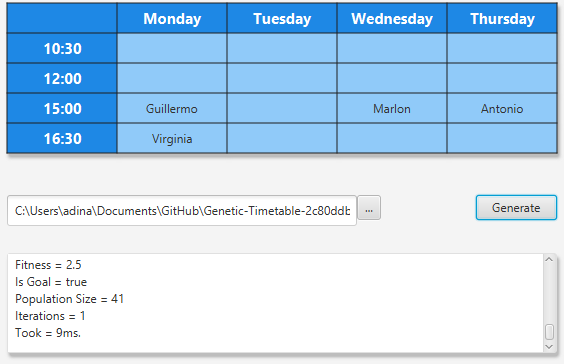
Convocatoria 3: para el tercer archivo la media, el mínimo y el máximo tienen el mismo valor 0.014.Con respecto a las anteriores la media ha bajado y también el máximo. El tiempo máximo de ejecución del algoritmo se establece en 57 ms y como vemos ha bajado también en comparación con el tiempo alcanzado en las convocatorias anteriores.



1. **Modificación del algoritmo genético de AIMA**

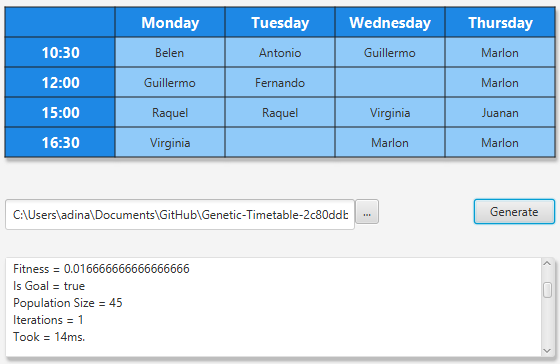
Hemos realizado pruebas después de introducir la probabilidad de cruce y los resultados cambian.

Para la convocatoria 1 con una probabilidad de 0.7 la media de la función de fitness sigue siendo bastante alta (2.117). El mínimo se mantiene en 1.5, pero el máximo baja su valor quedándose en 3.5. En este caso el tamaño máximo de la población es 41.

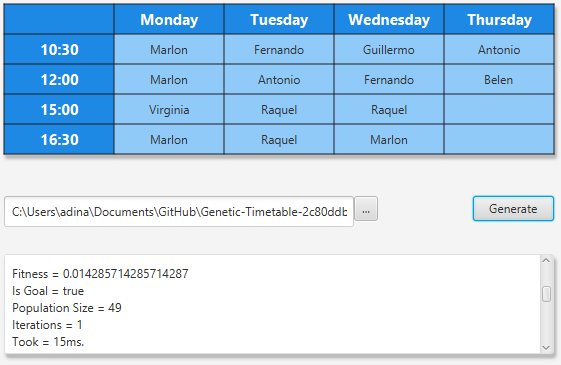


Tanto si introducimos una probabilidad de cruce de 0.8, como una de 0.9 para esta convocatoria tanto el mínimo, como el máximo mantienen su valor, pero la función de fitness tiene su máximo de 2.26 para la de 0.8 y 2.5 para la de 0.9.

En el caso de la segunda convocatoria el valor mínimo (0.014) de la función de fitness y el máximo (0.016) mantienen su valor independientemente del valor de la probabilidad. En cambio, la media cambia siendo 0.256, 0.0151 y 0.0152 para una probabilidad de 0.7, 0.8 y 0.9, respectivamente. En el segundo caso de la probabilidad el tamaño máximo de la población es 45.

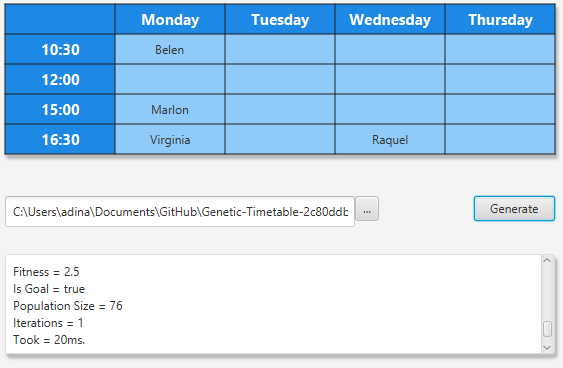


En el caso de la última convocatoria los 3 valores (media, mínimo, máximo de la función de fitness) que hemos tomado en cuenta mantienen su valor de 0.014 para las 3 probabilidades posibles que hemos considerado. En el caso de la probabilidad de 0.9 el tamaño de la población es el más alto de las 3 convocatorias.



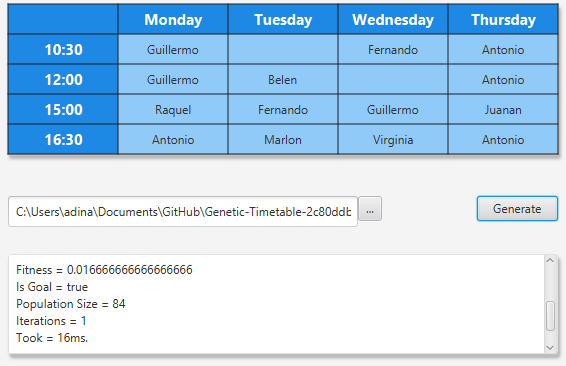
Hemos cambiado la función de cruce para que nos devuelva 2 individuos en vez de uno y estos son los cambios que hemos notado al realizar pruebas:

Para la primera convocatoria y con una probabilidad de cruce de 0.7 la media del fitness ha aumentado, siendo esta 2.38. El valor mínimo de esta función sigue siendo 1.5 y el máximo ahora es solo 2.5.

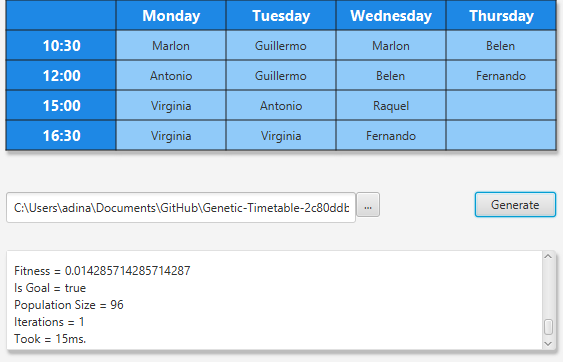


Como vemos también el tamaño de la población ha incrementado, llegando a un máximo de 76.

Con una probabilidad de cruce de 0.8 y tomando en cuenta la segunda convocatoria tenemos el mismo valor para la media, el mínimo y el máximo de la función de fitness (0.016). Esto demuestra que teniendo en cuenta solo la probabilidad de cruce el valor de la media es menor que la media calculada si el cruce devuelve 2 individuos.

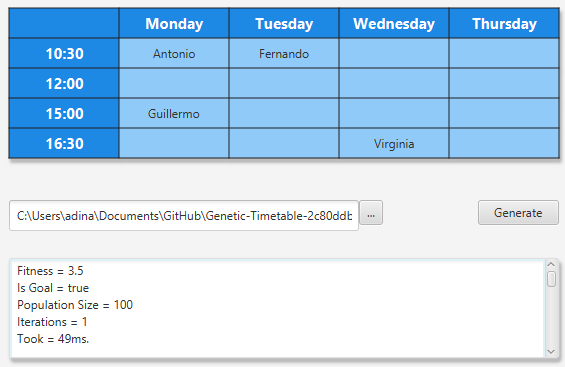


En el último caso de la tercera convocatoria y dándole el valor 0.9 a la probabilidad de cruce obtenemos los mismos resultados si no podríamos la condición que se devuelvan 2 individuos. Este valor es igual para la media de la función de fitness, para su mínimo y su máximo (0.014) y el tamaño de la población alcanza su máximo (96).

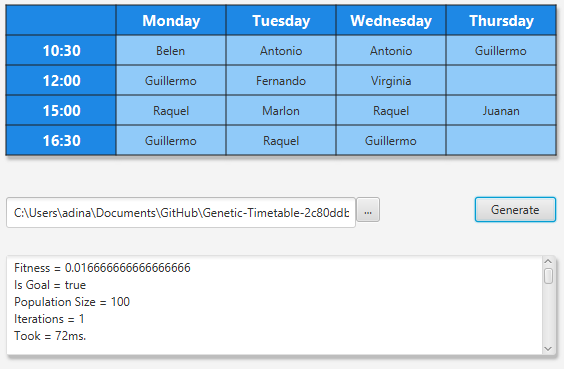


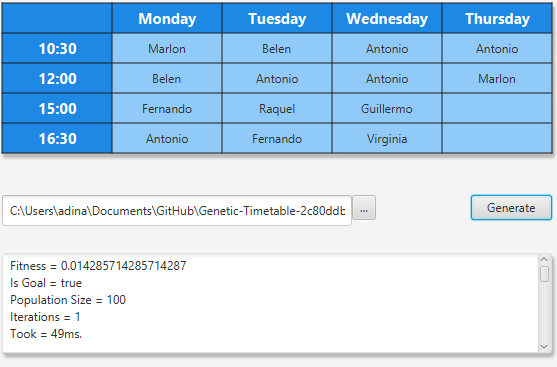
Hemos cambiado la estrategia destructiva del cruce por la no destructiva, teniendo en cuenta la función de fitness de los padres para la introducción de los individuos en la nueva generación. Fijamos la probabilidad de cruce en 0.7 para realizar las pruebas con los 3 archivos.

Después de las pruebas realizadas sobre el primer archivo hemos sacado la media de los valores de la función de fitness y el resultado es 2.32. El valor más bajo que ha tomado esta función ha sido 1.5 y el más alto 3.5. El algoritmo necesita un tiempo entre 9 ms y 49 ms para terminar de ejecutarse.



Para los otro 2 archivos hemos obtenido resultados iguales para la media, el máximo y el mínimo. En el caso del segundo archivo este valor es 0.016 y para el tercero es 0.014. Por último, el tiempo máximo para el segundo es 72 ms y 49 para el tercero.





1. **Parte opcional**