Práctica 4

Algoritmos Genéticos

Javier Pellejero Ortega & Zhaoyan Ni

Inteligencia Artificial

Grupo 11

Doble grado Matemáticas e ingeniería informática

1. **Descripción de como habéis representado los individuos**.

En esta práctica, tenemos que asignar *turnsToAssign* turnos de examen CFI a los profesores siendo *turnsToAssign* un entero positivo menor o igual al 16 usando el algoritmo genético. Cada profesor tiene dos listas, una de restricciones y otra de preferencias de turnos y los turnos están enumerados desde 1 a 16.

Para representar los individuos (las posibles soluciones), hemos utilizado una lista de enteros y hemos enumerado los profesores por su orden de llegada, por ejemplo, el profesor 0 es el primer profesor de la entrada. La lista utilizada tiene un tamaño de *TOTAL_TURNS*, constante que indica el número total de turnos que existe, en este caso, 16 y los índices de esta lista representan los turnos, por ejemplo, el índice 0 representa al turno 1. Los valores de esta lista son unos enteros que representan a los profesores. Si algún índice *i* de la lista tiene valor -1, indica que el turno *i+1* no está asignado a ningún profesor. Nótese que en esta lista aparece *TOTAL_TURNS – turnsToAssign* veces el valor -1.

2. **Descripción de la generación de población inicial.**

Para obtener la población inicial, hemos generado 50 individuos aleatoriamente. Si alguno de ellos es imposible de generar, mostramos un mensaje por la pantalla y terminamos el proceso.

Para generar los individuos aleatoriamente, llamamos la función *generateRandomIndividual* de la clase *CFIGenAlgoUtil* que recibe dos argumentos: *turnsToAssign,* número de turnos necesarios a asignar y *restrictionsList,* lista que contiene la lista de restricciones de cada profesor. En esta función, comprobamos si *turnsToAssign* es mayor que *TOTAL_TURNS*, en caso afirmativo, no es posible generar un individuo y devolvemos *null*. Luego, veamos si el número total de restricciones de todos los profesores sumado al *turnsToAssign* es mayor o igual que *TOTAL_TURNS* multiplicado al número de profesores, en caso afirmativo, devolvemos *null* también puesto que en este caso no es posible asignar *turnsToAssign* turnos a los profesores sin que a ningún profesor le toca un turno que está en su lista de restricciones. Después de excluir los casos en los que no podemos generar los individuos, iniciamos los valores de la representación del individuo, una variable del tipo *List<Integer>* de tamaño *TOTAL_TURNS*, a todos -1 y luego generamos un número aleatorio para el turno y el otro número aleatorio para el profesor para asignarle el turno que acabamos de obtener. Si el turno generado está en la lista de restricciones del profesor elegido aleatoriamente, generamos de nuevo estos dos números todas las veces necesarias. Repetimos el proceso anterior *turnsToAssign* veces para tener un individuo con exactamente *turnsToAssign* turnos asignados.

3. Descripción de la implementación del operador de cruce

Para generar los individuos de la siguiente generación, hacemos el cruce de dos individuos de esta generación. Para ello, llamamos la función *reproduce* de la clase *CFIGeneticAlgorithm* que implementa la clase *GeneticAlgorithm<Integer>*.

La función *reproduce* recibe dos argumentos: el individuo *x* y el individuo *y* que son de misma generación y van a reproducir. Una vez seleccionados dos individuos para cruzar se cortan sus cromosomas por un punto seleccionado aleatoriamente para generar dos segmentos diferenciados en cada uno de ellos: la cabeza y la cola. Para generar los nuevos descendientes, se recorre primero el individuo *x* hasta el punto seleccionado aleatoriamente y copiar los valores en el individuo hijo que inicialmente no tiene ningún turno asignado. Después, recorremos el individuo *y* desde el punto seleccionado aleatoriamente y copiar los valores en el individuo hijo. Durante este proceso, tenemos en cuenta el número de turnos asignados. Cuando el número de turnos asignados actualmente es igual a *turnsToAssign* y no hemos terminado el cruce, terminamos el proceso. Si el número de turnos asignados actualmente es menor que *turnsToAssign* y hemos terminado el cruce, recorremos el individuo *y* desde principio para encontrar un turno que está asignado en el individuo *y* pero no en el individuo hijo y copiar el valor en el individuo hijo. Repetimos el proceso último para tener *turnsToAssign* turnos asignados en el individuo hijo.

4. Descripción de la implementación del operador de mutación

Para hacer la mutación de un individuo, llamamos la función *mutate* de la clase *CFIGeneticAlgorithm.*

La función *mutate* recibe como argumento un individuo, sobre este individuo vamos a hacer la mutación. Para hacer la mutación, seleccionamos aleatoriamente un número *m* entre 0 y *TOTAL_TURNS* y veamos si el alelo *m* del individuo es -1 o no, en caso afirmativo actualizamos el valor *m* sumándole 1 y dividirlo entre *TOTAL_TURNS* y el valor nuevo de *m* será el resto de la división. Una vez obtenemos el valor *m*, guardamos todos los profesores disponibles en una lista, es decir, aquellos que no tienen al turno *m* en su lista de restricciones y no tenía asignado al turno *m*. Cambiamos el valor del alelo *m* del individuo a un número que representa a uno de los profesores disponibles.

5. Descripción de la función de fitness empleada

La función de fitness calcula la aptitud de cada individuo, es decir, indica como de bueno es cada individuo a la hora de resolver el problema. Esta función siempre toma valores positivos, penalizar a las malas soluciones y premiar a las buenas.

En la función de fitness que hemos implementado, damos más importancia a la asignación correcta de los turnos, es decir, si a algún profesor le asignamos un turno que está en su lista de restricciones, la función de fitness devuelve 0.1. Si la asignación de los turnos es correcta, vamos a ver cuántos profesores tienen asignados turnos de su preferencia y a este número le restamos el número de profesores que su número de turnos no está equilibrado. La aptitud de cada individuo es la suma de *turnsToAssign* y la solución de la resta más 1 para evitar que la función de fitness devuelve un número nulo. Nótese que este número es mayor cuando mejor es la solución pues el número de profesores que tienen asignados turnos que ellos prefieren es mayor y el número de profesores que tienen número de turnos desequilibrado es menor, así la resta da un número mayor que otras soluciones peores y la función de fitness devuelve un número mayor, es decir, premia a las soluciones buenas.

6. Descripción de la función objetivo empleada

La función objetivo está implementada en la clase *CFIGoalTest* que extiende de la clase *GoalTest.* Esta función recibe un individuo y devuelve una variable del tipo booleano.

Para ver si hemos llegado al estado objetivo, calculamos el valor de fitness del individuo y veamos si es igual a *turnsToAssign* multiplicado al 2 y sumando 1 a la solución de la multiplicación. Esto es porque en el estado objetivo, el número de turnos asignados es igual a *turnsToAssign,* cada profesor tiene asignado turnos que los prefieren y el número de turnos de cada profesor está equilibrado. En este caso, la función de fitness devuelve un valor que es igual a *turnsToAssign* multiplicado al 2 y sumando 1 a la solución de la multiplicación.

Antes de continuar con los siguientes puntos de la memoria, presentamos la tabla comparativa de los resultados obtenidos con los 6 algoritmos.

La probabilidad de mutación escogida ha sido de 0,15, el tiempo límite 500 milisegundos y se han realizado 100 ejecuciones de cada algoritmo por cada configuración.

Algoritmo	Max. Fitness	Media Fitness	Min. tiempo	Media tiempo	Objetivo
CONFIGURACIÓN ARCHIVO 1					
Estándar	9	8,97	0	25	Sí
Probabilidad 0,7	9	8.97	3	4	Sí
Probabilidad 0,8	9	8.96	2	10	Sí
Probabilidad 0,9	9	8.99	2	14	Sí
Dos hijos, prob. 0,8	9	8.87	1	71	Sí
No destructivo, dos hijos, prob. 0,8	9	8.83	0	89	Sí
CONFIGURACIÓN ARCHIVO 2					
Estándar	20	19,06	501	501	No
Probabilidad 0,7	20	19,03	501	501	No
Probabilidad 0,8	20	19,02	501	501	No
Probabilidad 0,9	20	19,14	501	501	No
Dos hijos, prob. 0,8	20	18,92	501	501	No
No destructivo, dos hijos, prob. 0,8	20	19,96	501	501	No
CONFIGURACIÓN ARCHIVO 3					
Estándar	20	18,88	501	501	No
Probabilidad 0,7	20	19,2	501	501	No
Probabilidad 0,8	20	19,12	501	501	No
Probabilidad 0,9	20	19,12	501	501	No
Dos hijos, prob. 0,8	20	18,91	501	501	No
No destructivo, dos hijos, prob. 0,8	20	19,99	501	501	No

7. Resultados obtenidos por el algoritmo desarrollado para las configuraciones proporcionadas.

El algoritmo estándar consigue encontrar una solución objetivo (todos los turnos asignados de manera equilibrada y satisfaciendo las preferencias de todos los profesores a los que les hemos asignado turno) sólo en la primera configuración.

Deducimos, ya que ningún otro algoritmo consigue dar con la solución objetivo, que ni la configuración 2 ni la 3 tienen dicha solución. Sin embargo, si que podemos afirmar a partir de los resultados de otros algoritmos (todos son coincidentes con el estándar) en que encuentra una de las soluciones óptimas en los casos 2 y 3, ergo podemos suponer que no existe una mejor.

Poco podemos hablar del tiempo empleado por el algoritmo estándar en los casos 2 y 3 por obvias razones, pero sí podemos hacerlo del caso 1.

Tenemos el mejor tiempo mínimo, de suponemos, poco más de 0 milisegundos, aunque bien ha podido deberse a una casualidad, ya que la media no es tan buena, siendo superado con creces por los algoritmos de probabilidad de cruce.

Por último, la media de Fitness es la segunda mejor en la configuración 1, la cuarta en la configuración 2 y la tercera en la configuración 3. Esto nos viene a decir que en la mayor parte de las ejecuciones ha conseguido llegar a una solución óptima y no se ha quedado atascada en torno a una solución local, haciendo un papel notable frente al resto de algoritmos.

8. Influencia de la probabilidad de cruce en la aplicación.

En cuanto a estados objetivos no hay variación: al igual que el estándar, ha encontrado al menos una de las soluciones óptimas y a juzgar por sus medias, que ahora analizaremos, lo ha conseguido en la gran mayoría de casos.

Podemos decir que sus medias de fitness son las mejores en la configuración 1 y las segundas mejores en la 2 y 3. Además también tiene las mejores medias en tiempo en la configuración 1. A través de estos datos podemos decir que es el mejor algoritmo cuando existe una solución objetivo y el segundo mejor tras el no destructivo cuando no la hay.

En definitiva, podemos señalar este algoritmo como el más estable.

9. Influencia de obtener dos individuos en el cruce en lugar de uno en la aplicación.

En el caso de obtener dos individuos en el cruce, generamos más individuos de la siguiente generación que en el caso de obtener un individuo solo en el cruce. En teoría, esto da más diversidad en la siguiente generación.

Sin embargo, observando en las tres tablas anteriores, si comparamos el caso de obtener un individuo en el cruce y con probabilidad de cruce 0,8 con el caso de obtener dos individuos en el cruce y con probabilidad de cruce 0,8, vemos que el valor máximo de fitness es la misma en las tres tablas, el tiempo mínimo de obtener dos individuos en el cruce es mejor que el de obtener un individuo en el cruce en la primera tabla y son iguales en las tablas 2 y 3. Pero al observar la media de fitness y la media de tiempo, obtener dos individuos en el cruce es peor que obtener un individuo en el cruce. Es decir, obtener dos individuos en el cruce es menos eficiente que obtener un individuo en el cruce.

10. Influencia de la estrategia no destructiva frente a la estrategia destructiva.

Para ver la influencia de la estrategia no destructiva frente a la estrategia destructiva en la aplicación, comparamos el caso de estrategia no destructiva, obtener dos hijos en el cruce y la probabilidad de cruce es 0,8 con el caso de estrategia destructiva, obtener dos hijos en el cruce y la probabilidad de cruce es 0,8, es decir, observamos las dos últimas filas de las tres tablas.

En la primera tabla, observamos que el valor máximo de fitness es igual en los dos casos. El caso de estrategia no destructiva es mejor en tiempo mínimo, pero es peor en la media de fitness y en la media de tiempo. En las otras dos tablas, el caso de estrategia no destructiva es mejor en media de fitness y en otros campos, las dos estrategias tienen valores iguales. En conclusión, la estrategia no destructiva es menos eficaz que la estrategia destructiva en estos casos pero mejora la media de fitness.