Optimización Heurística

Optimizar una calificación

Daniel Tomás Sánchez
Aarón Cabero Blanco
Pablo Bautista Frías

03/01/2020

Índice

1 | Introducción

Durante la realización de esta práctica se ha planteado un problema de optimización para su posterior resolución. Se explicará en detalle el problema así como las operaciones realizadas para obtener la solución óptima con el algoritmo. Posteriormente se resolverá el problema con diferentes datos de entrada para así poder ver los distintos resultados obtenidos.

2 | Descripción

El problema planteado consiste en la optimización de la nota media obtenida en un conjunto de asignaturas por un alumno. Para que el caso sea lo más real posible, se han añadido distintas variables que afectan al resultado óptimo del problema, como por ejemplo, las horas de las que dispone el alumno para estudiar o la dificultad que tiene cada unas de las asignaturas.

Dado que se está intentando sacar la máxima nota posible, ha sido necesario cambiar nuestro anterior algoritmo DE implementado en la primera práctica (implementación de un algoritmo evolutivo) para que en lugar de minimizar, maximizase la nota obtenida. Al igual que se han realizado otros cambios como la forma de visualizar los resultados por pantalla y cómo se suman los ”arrays” en la clase ”Rand1MutationOperator()”, ya que ahora se realiza de forma directa y no sumando uno a uno cada elemento de este.

2.1 Variables del problema

Las distintas variables usadas en el problema son:

- **minimun_marks (array)**: Esta variable representa la nota mínima que se puede sacar en cada asignatura. En caso de no sacar la nota mínima en una asignatura, no se podrá pasar con éxito el curso.

- **point_per_hour (array)**: Cada elemento del array indica el número de horas que se deberá estudiar para sacar un punto en una asignatura. Esto también indica la dificultad de cada asignatura, cuantas más horas haya que estudiar para sacar un punto en una asignatura, mayor dificultad tendrá esta.

- **credits (array)**: Indica el valor de cada asignatura mediante un sistema de créditos. Una asignatura con más créditos tendrá más valor que una con menos créditos, de esta forma, a la hora de hacer la media, ponderará más una asignatura con más créditos.

- **revision_probability (array)**: Mediante esta variable se indica la probabilidad que tiene una asignatura de que en la revisión se le suba nota al alumno. Dado que

esta variable se combina con "student_luck" para determinar si finalmente al alumno se le subirá la nota, cuanto menor sea el número en "student_luck" más fácil será para el alumno subir nota en dicha asignatura. Los elementos del array deberán estar entre los valores 0 y 1.

- **student_luck (array)**: Tal y como hemos mencionado anteriormente, esta variable se combina con la anterior para determinar si un alumno subirá nota en una asignatura o no. Si "student_luck" es menor o igual que "revision_probability", el alumno subirá nota en dicha asignatura. Los elementos del array deberán estar entre los valores 0 y 1.

- textbfrevision_mark (array): En el caso de que, según lo anterior, el alumno suba nota en una asignatura, "revision_mark" indica la puntuación que subirá dicho alumno en la asignatura. Dado que las asignaturas se califican con una puntuación entre 0 y 10, un alumno nunca pondrá obtener una nota superior a 10 tras realizar la revisión. Para no variar demasiado los resultados obtenidos en las asignaturas, los valores entre los que se comprenden los elementos del array se encontrarán entre 0 y 0.5.

- **study_hours (int)**: La variable "study_hours" representa el número total de horas de las que dispone un alumno para estudiar. El resultado óptimo no podrá tener un número total de horas mayor que las horas de las que dispone el alumno.

Todas las variables anteriores se encuentran dentro del fichero "Data.py" y pueden ser modificadas. A la hora de modificar estas variables hay que tener en cuenta que el tamaño de los arrays debe ser el mismo, y este tamaño será igual al número de asignaturas con las que queramos plantear el problema. Además de las variables anteriores, existen otras variables que se introducirán por terminal, o que en su defecto, tendrán valores predeterminados. Estas variables son las siguientes:

- **population_size (int)**: Esta variable indica el tamaño de la población. En los casos donde no se introduzca mediante la terminal, su valor predeterminado será de 50.

- **iterations (int)**: Esta variable indica el número de iteraciones que se realizará el algoritmo. Al igual que la anterior, en los casos en los que no se introduzca por la terminal tendrá un valor predeterminado, y este valor será de 500.

- **reps (int)**: Indica el número de repeticiones que se realizará el algoritmo. Esta variable será obligatoria introducirla en los casos en los que se pida.

2.2 Bounds

En el problema que estamos tratando, el rango de los valores de los posibles candidatos puede variar dependiendo de los valores que obtengan las variables de entrada. Aún así, el

rango de valores una vez se han introducido las variables de entrada se obtiene según la siguiente fórmula matemática:

$$(minimum\_marks[i]/point\_per\_hour[i], 10/point\_per\_hour[i])$$

De esta forma, el rango irá desde las horas que necesita un alumno para sacar la nota mínima, a las horas que necesita un alumno estudiar para sacar un 10, que es la máxima nota.

3 | Función Fitness

Dado que nuestro problema consiste en maximizar la nota del alumno, cuanto mayor sea el "fitness" mejor será el resultado. A continuación mostraremos la parte del código utilizada para el "fitness" y posteriormente procederemos a explicarla.

```
def our_fitness(array):
    res = 0
    for i in range(len(array)):
        mark = (array[i] * point_per_hour[i])
        if student_luck[i] <= revision_probability[i]:
            mark += revision_mark[i]
        if mark >10:
            mark = 10
        res += mark * credits[i]
    nota = res / sum(credits)
    if sum(array) >study_hours:
        return -((sum(array) - study_hours) ** 2 / nota)
    return nota
```

La función "fitness" se encarga de calcular la nota media obtenida por el alumno. En primer lugar, se calcula la nota que sacaría en cada una de las asignaturas, multiplicando las horas que dedica a esa asignatura por los puntos que se obtienen en dicha asignatura por cada hora estudiada ($array[i] * point\_per\_hour[i]$). Una vez que el alumno obtiene la nota de la asignatura, tal y como hemos mencionado anteriormente, cabe la posibilidad de subir nota en la revisión. Si la suerte del estudiante en esa asignatura es menor que la probabilidad de revisión de esa misma asignatura ($student\_luck[i] <= revision\_probability[i]$), el alumno recibirá una subida de nota igual a la cantidad que el array "revision_mark" tenga en la posición en la que se encuentra la asignatura ($mark+ = revision\_mark[i]$). En caso de que tras la subida de nota, la calificación del alumno fuese mayor de 10 puntos ($mark > 10$), dicho alumno se quedaría con una nota igual a 10 ($mark = 10$). Posteriormente se guardará el valor que le ha otorgado esa nota al alumno en la variable "res", realizando la multiplicación de dicha nota por los créditos de la asignatura ($res+ = mark * credits[i]$) dado que una asignatura con más créditos tendrá más peso en la nota final. Para finalizar, se realizará la media con todos los resultados, esto se hace dividiendo el resultado final que hemos obtenido en "res" por los créditos totales de las asignaturas ($nota = res/sum(credits)$) y el resultado de esa nota será lo que devuelva nuestra función "fitness".

En el caso de habernos pasado de horas de estudio, es decir, que las horas de estudio que hemos empleado sea mayor que el número de horas de estudio de las que disponía el alumno ($sum(array) > study\_hours$), el resultado será un valor negativo ($-((sum(array) - study\_hours) * *2/nota)$), se ha elevado al cuadrado la diferencia entre las horas que ha estudiado y las que el alumno tiene disponible para que influya más ese aspecto en la penalización. De esta forma guiaremos al algoritmo al resultado más optimo sin sobrepasar las horas de estudio de las que el alumno dispone.

4 | Ejemplos

A continuación, vamos a realizar algunos ejemplos para visualizar los resultados obtenidos con el algoritmo y cómo se visualizan éstos.

4.1 Horas insuficientes para aprobar

En este primer ejemplo, el número de horas de las que dispone el alumno, es inferior a las horas mínimas necesarias para sacar la nota mínima en todas las asignaturas, lo que implica que el alumno no podrá aprobar el curso. Vamos a ver cómo se comporta el algoritmo en esta situación.

- **Variables de entrada**

```
minimum_marks = [3, 4, 5, 2, 1]
point_per_hour = [1, 2, 1, 2, 3]
credits = [3, 6, 1, 4, 2]
revision_probability = [0.4, 0.3, 0.2, 0.5, 0.3]
student_luck = [0.3, 0.2, 0.4, 0.5, 0.4]
revision_mark = [0.5, 0.1, 0.3, 0.4, 0.2]
study_hours = 9
```

4.1.1 Resultados con 1 repetición

Los resultados obtenidos al realizar una repetición del algoritmo son los siguientes:

```
Introduzca el tamaño de población, en caso de no hacerlo y pulsar la tecla "enter" el valor será "50": 50
Introduzca el número de iteraciones, en caso de no hacerlo y pulsar la tecla "enter" el valor será "500": 500
El número de asignaturas a evaluar son: 5
Las horas mínimas de estudio por cada asignatura son:
Horas mínimas: [3.    2.    5.    1.    0.333]
Best Genome: [ [3.    2.    5.    1.    0.333], fitness=-1.685 ]
Según el algoritmo, obtendría la mejor nota estudiando un total de: 11.333 horas
```

Como se puede observar en la imagen del resultado, el ”fitness” obtenido es un número negativo. Esto sucede debido al ejemplo en el que nos encontramos, donde el número de horas de las que dispone el alumno es insuficiente para llegar a la nota mínima de todas las

asignaturas y por tanto, tiene que sobrepasar esas horas (y ser penalizado) para alcanzar la nota mínima.

Esto también podemos observarlo en el vector ”Horas mínimas” donde nos indica las horas que deberíamos estudiar cada asignatura para llegar a su nota mínima y cuyo sumatorio es igual a 11.33, un número de horas mayor de las que dispone el alumno en el ejemplo (study_hour), que son 9 horas (tal y como se puede observar en la imagen de ”Variables de entrada”).

4.1.2 Resultados con varias repeticiones

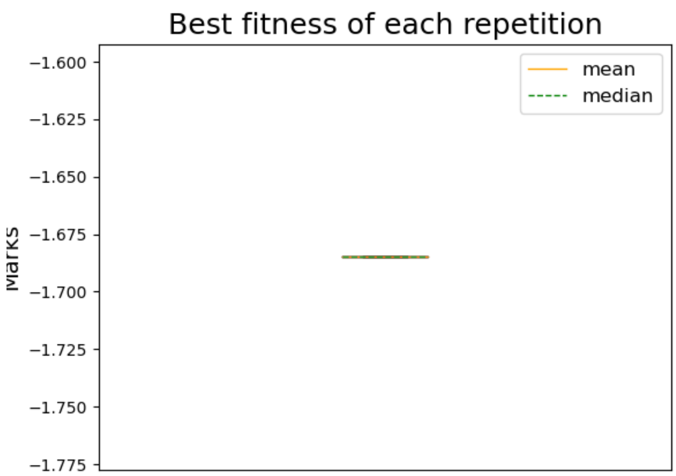
Ahora realizaremos varias repeticiones del mismo ejemplo para ver gráficamente las diferencias entre los distintos resultados. En esta ocasión, el número de repeticiones que realizaremos serán 10, pero estas repeticiones pueden introducirse en cada ejemplo mediante la terminal.

```
Ahora vamos a ajecutar el algoritmos varias veces para ver como se comporta con un tamaño de población = 50 y 500 iteraciones
Por favor, introduce el número de repeticiones que desea ejecutar el algoritmo: 10
Best Genome: [ [3.    2.    5.    1.    0.333], fitness=-1.685 ]
Best Genome: [ [3.    2.    5.    1.    0.333], fitness=-1.685 ]
Best Genome: [ [3.    2.    5.    1.    0.333], fitness=-1.685 ]
Best Genome: [ [3.    2.    5.    1.    0.333], fitness=-1.685 ]
Best Genome: [ [3.    2.    5.    1.    0.333], fitness=-1.685 ]
Best Genome: [ [3.    2.    5.    1.    0.333], fitness=-1.685 ]
Best Genome: [ [3.    2.    5.    1.    0.333], fitness=-1.685 ]
Best Genome: [ [3.    2.    5.    1.    0.333], fitness=-1.685 ]
Best Genome: [ [3.    2.    5.    1.    0.333], fitness=-1.685 ]
Best Genome: [ [3.    2.    5.    1.    0.333], fitness=-1.685 ]
```

En la imagen, podemos observar cómo todos los resultados coinciden con las horas mínimas de estudio para cada asignatura. Esto sucede porque el alumno no dispone de horas suficientes para alcanzar la nota mínima, por lo que no podría aprobar el curso con éxito, de ahí que todos los ”fitness” resultados sean negativos.

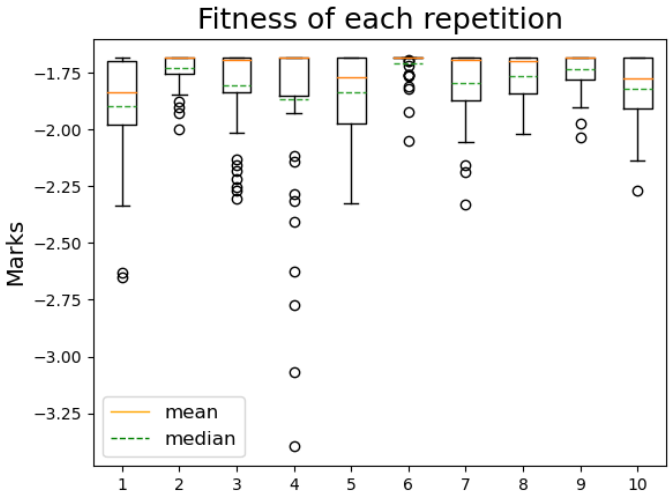
Ahora visualizaremos el resultado de los distintos ”fitness” mediante diagramas de cajas, para así tener una imagen más visual de los resultados que se han obtenido.

- **Mejor fitness de cada repetición**



En el gráfico podemos ver representado el resultado de la imagen anterior, en el que el mejor fitness de cada repetición es -1.685. Dado que todos los mejores fitness de cada repetición tienen el mismo valor, en este caso no podemos apreciar el diagrama de caja, pero si podemos apreciar que el resultado obtenido en el diagrama es el correcto.

- **Todos los fitness de cada repetición**



Este diagrama nos muestra todos los fitness que se han obtenido por cada una de las 10 repeticiones. Dado que las horas de estudio del alumno son inferiores a las horas mínimas necesarias para sacar la nota mínima de cada asignatura, todos los resultados obtenidos son resultados negativos, además de obtener muchos resultados dispares. Como podemos observar en el diagrama de caja, existen muchos resultados atípicos debido a la naturaleza del ejemplo en el que nos encontramos.

4.2 Horas suficientes para aprobar

Este ejemplo plantea un escenario totalmente distinto al anterior. En esta ocasión, el alumno dispondrá de las horas suficientes para poder sacar la nota mínima en todas las asignaturas, y partiendo de este punto, veremos cuantas horas debe dedicar a cada una de las asignaturas para sacar la mayor nota posible en el curso. El ejemplo se realizará con las mismas variables de entrada que el ejemplo anterior, pero aumentando el número de horas de estudio (study_hour) a 16.

- **Variables de entrada**

```
minimum_marks = [3, 4, 5, 2, 1]
point_per_hour = [1, 2, 1, 2, 3]
credits = [3, 6, 1, 4, 2]
revision_probability = [0.4, 0.3, 0.2, 0.5, 0.3]
student_luck = [0.3, 0.2, 0.4, 0.5, 0.4]
revision_mark = [0.5, 0.1, 0.3, 0.4, 0.2]
study_hours = 16
```

4.2.1 Resultados con 1 repetición

Los resultados obtenidos al realizar una repetición del algoritmo son los siguientes:

```
Introduzca el tamaño de población, en caso de no hacerlo y pulsar la tecla "enter" el valor será "50": 50
Introduzca el número de iteraciones, en caso de no hacerlo y pulsar la tecla "enter" el valor será "500": 500
El número de asignaturas a evaluar son: 5
Las horas mínimas de estudio por cada asignatura son:
Horas mínimas: [3.    2.    5.    1.    0.333]
Best Genome: [ [3.215 4.579 5.069 2.251 0.882], fitness=6.041 ]
Según el algoritmo, obtendría la mejor nota estudiando un total de: 15.995 horas
```

En los resultados se pueden observar varias cosas, por un lado, al igual que en el ejemplo anterior, nos muestra un vector con las horas mínimas de estudio necesarias para alcanzar la nota mínima en cada asignatura (Horas mínimas), y por otro lado nos muestra las horas optimizadas para cada asignatura (Best Genome) de tal forma que podamos alcanzar la nota máxima con las horas de estudio disponibles que tenemos (16).

El fitness obtenido es 6.041, según el algoritmo. Esta sería nuestra nota en caso de dedicar las hora obtenidas en "Best Genome" a cada una de las asignaturas, lo cual serían un total de 15.995 horas (aproximadamente las 16 horas de entrada).

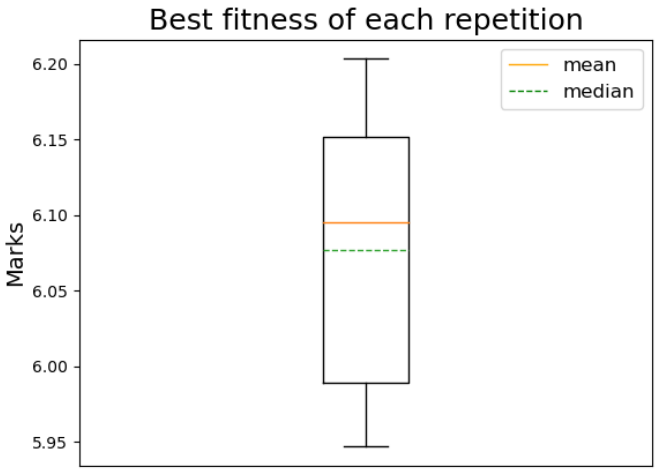
4.2.2 Resultados con varias repeticiones

En este apartado se realizarán 10 repeticiones del algoritmo para el mismo ejemplo. De esta forma podremos observar cómo se comporta el algoritmo al realizar varias repeticiones y cómo varían los resultados entre las distintas repeticiones.

```
Ahora vamos a ajecutar el algoritmos varias veces para ver como se comporta con un tamaño de población = 50 y 500 iteraciones
Por favor, introduce el número de repeticiones que desea ejecutar el algoritmo: 10
Best Genome: [ [3.176 4.563 5.    2.566 0.629], fitness=6.080 ]
Best Genome: [ [3.233 5.    5.    2.05  0.716], fitness=6.156 ]
Best Genome: [ [3.    5.    5.388 1.255 1.284], fitness=5.952 ]
Best Genome: [ [3.    5.    5.145 1.792 1.03 ], fitness=6.110 ]
Best Genome: [ [3.02  4.755 5.094 2.581 0.516], fitness=6.166 ]
Best Genome: [ [3.    4.876 5.    2.191 0.919], fitness=6.203 ]
Best Genome: [ [3.788 4.844 5.038 1.48  0.847], fitness=5.947 ]
Best Genome: [ [3.    4.744 5.069 1.798 1.265], fitness=6.042 ]
Best Genome: [ [3.519 4.704 5.    1.663 1.088], fitness=5.971 ]
Best Genome: [ [3.269 5.    5.023 2.142 0.532], fitness=6.141 ]
```

Aquí podemos observar los resultados de cada una de las 10 iteraciones. Como se puede ver en la imagen, no hay gran diferencia entre los "fitness" de cada repetición, lo que indica que el algoritmo funciona correctamente. En este ejemplo, el 'fitness" lo podemos encontrar entre los valores 5.9 y 6.2, pero eso se podrá observar mejor en los diagramas de cajas a continuación.

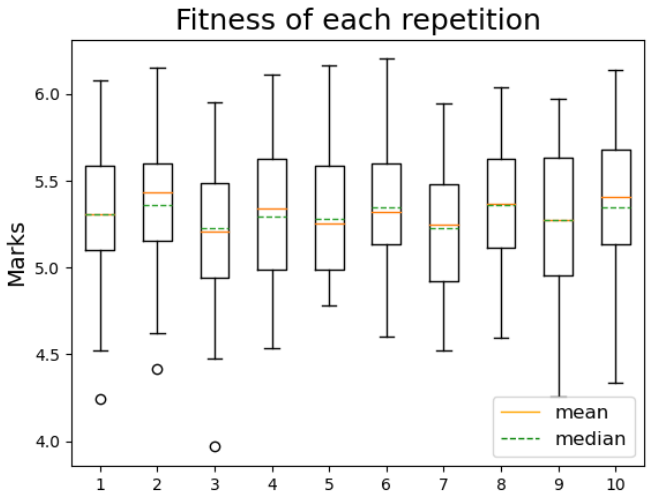
- **Mejor fitness de cada repetición**



Tal y como hemos mencionado antes, en este diagrama de cajas se puede apreciar con más claridad el rango en el que se encuentra el ”fitness” de cada una de las repeticiones. La media la podemos encontrar aproximadamente en el valor 6.1.

Por otro lado, podemos observar cómo el mejor ”fitness” de las 10 repeticiones se encuentra en el valor 6.2, lo que nos indicaría que tras realizar varias veces el algoritmo, la nota máxima que podemos alcanzar con 16 horas de estudio es 6.2 si las distribuimos de la siguiente forma: [3. 4.876 5. 2.191 0.919] ocupando cada posición del vector una asignatura.

- **Todos los fitness de cada repetición**



En este segundo diagrama de cajas podemos observar los valores ”fitness” que se han ido obteniendo en cada una de las repeticiones. Podemos observar cómo la distribución de los datos obtenidos en cada una de las repeticiones es muy similar, exceptuando algunos datos que se encuentran separados, los cuales podríamos tomar como anomalías.

En este diagrama se observa mejor el funcionamiento del algoritmo que en el ejemplo anterior. Ya que el alumno dispone de las horas suficientes para estudiar todas las asignaturas y alcanzar la nota mínima, el programa puede optimizar las horas para encontrar el resultado con el cual el alumno se acercará a la mejor nota, y esto se ve reflejado en la distribución de los datos obtenidos.

5 | Conclusiones

Este trabajo nos ha ayudado a afianzar los conocimientos obtenidos durante la asignatura. La ocasión de plantear nosotros mismos un problema, y más en este caso donde el problema nos toca tan de cerca, nos ha ayudado a poder seguir el proceso lógico del algoritmo con más facilidad.

También nos ha brindado la oportunidad de poder jugar con las variables de entrada y distintos valores, para ver de esta forma como se alteraban los resultados y comprender mejor el funcionamiento.

Por último, no solo nos ha servido para aprender, si no que al ser un ejemplo muy visual y como hemos mencionado anteriormente, un ejemplo cercano a nuestra situación, nos ha resultado curioso probar distintas variaciones para alterar de esta forma los resultados e imaginarnos como sería aplicar este problema a nuestro día a día en la facultad.