Can AI students beat hashcode2020 qualification round high score?

Tymon Dydowicz 151936 Group: 1 **Eryk Ptaszyński 151950 Group: 2**

Abstract

Probably not.

COMBINATORIAL OPTIMIZATION

I) Instruction:

- As our development environment we used Google Collaboratory and then we extracted it to Python script
- Therefore the program is written in Python 3.8
- We also used 4 Python libraries namely: Sys, NumPy, Random and Time
- To run the script you need to use CAT command to read the input file and then pipe that into a Python script using "|", then the output of the script has to be redirected somewhere, preferably to output.txt file

```
cat a_example.txt | python3 ./hash_code_151936_151950.py > output.txt
```

Figure 1 Example of running our script, reading the a_example.txt and saving the output to output.txt file

II) Explanation:

As the heuristic algorithm we have used a combination of GA (*Genetic algorithm*) and Greedy. At first we only implemented GA however it performed very poorly so

```
a_example : |score = 21 |time = 0.27918338775634766
b_read_on : |score = 5658900 |time = 240.09250164031982
c_incunabula : |score = 957607 |time = 241.02282285690308
d_tough_choices : |score = 4359225 |time = 246.15873646736145
e_so_many_books : |score = 1861355 |time = 240.49368977546692
f_libraries_of_the_world : |score = 3111619 |time = 240.38900780677795
```

Figure 2 Generic Genetic Algorithm Scores

we decided to add some elements of greedy approach, more precisely we made the initial population be chosen with Greedy approach that later would be explored and exploited by the GA. After that we experimented with multiple heuristics for the Greedy initialization, some of them brought bad results but some of them were quite good. Some of those heuristics were:

$$\gamma(x) = \frac{1}{x.signuptime}$$

$$\omega(x) = \frac{x.booksperday}{len(x.books)} \cdot \sum_{book \in x.books} book.score$$

$$\varepsilon(x) = \sum_{ub \in x.UB} ub.score + \frac{x.bpd}{len(x.B - x.UB)} \cdot \sum_{b \in x.B} b.score$$

where UB – unique books, B – books, bpd – books per day

Finally we settled for:

$$\varphi(x) = \frac{1}{x.signuptime} \cdot \sum_{book \in x.books} book.score$$

	γ	ω	ε	φ
a_example	21	21	21	21
b_read_on	5 822 900	4 126 100	4 126 100	5 822 900
c_incunabula	5 467 966	1 132 105	1 132 105	5 646 269
d_tough_choices	4 109 300	4 109 300	4 109 300	4 816 110
e_so_many_books	4 262 125	647 814	647 814	4 601 961
f_libraries_of_the_world	2 430 950	104 370	104 370	5 240 161
Total	22 093 262	10 119 710	10 119 710	26 127 422
Time (s)	4.38	5.13	703.10	5.51

Table 1 Results for given files while using particular heuristic

Our decision on why we chose GA was based on our previous experience with implementing such algorithms. GA are very versatile and elastic as long as you find a good way to represent the problem which also contributed to our choice. GAs are also very easy to control I.e. keep track of how much time or iterations has passed

and we can terminate it easily had it been taking too much time or wasting it by looking at the same solutions over and over.

III) Implementation:

First our program loads the data from input file, reads the first line and saves the values as “NUMBER_OF_BOOKS”, “NUMBER OF LIBRARIES”, “NUMBER OF DAYS”, then it loads the next line in generator using enumerate function to initialize all the books with correct id and score. In our approach we used Object Oriented Programming, where all libraries and books are objects with their respective fields. Each book object has it’s id and score which are stored as integers. This is a very simple object however it helps the code be more readable and explainable. We approached libraries in similar way where each object has fields: days_left, id, number_of_books, books which is a filed containing chosen books for given library in given solution, signup_time, books_per_day. Most of those fields are self-explainable. After initializing and creating all objects our GA can finally start. First the GA initializes it’s population using greedy approach, It sorts all Library objects using one of the aforementioned heuristics, and then for each library it assigns its days_left field to initial number of days and then decreases the number of days by its signup_time for the next library. Then the library calls its choose_greedy_books method which sorts all its books by the score and takes only those that have not been previously added until it has chosen $daysleft * bookPerDay$ books then a tuple (library, chosenbooks) is created and added to the solution, that creates a permutation of libraries each with its chosen books, and finally population is created in such a way that it holds the permutation of libraries with their books and the evaluation of the solution. Evaluate works in like this: it iterates over all books in solution and if a book was not yet read it adds it’s score if it was read then it just skips over it. After the population has been initialized the algorithms starts its evolution. In each evolution step is chooses some number of solutions which is determined by hyperparameters ($tournament\_size * population\_size$) and then we perform tournament rounds until the new population is created, this way we ensure elitism i.e. we make sure that

Figure 3 example book object

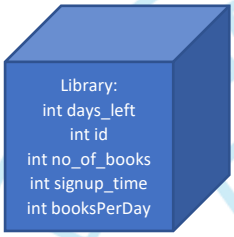

Figure 4 example library object

some number of best evaluated solutions always survives to the next population. New populations is created using aforementioned tournament rounds which mean that we sample whole population and chose the 2 best solutions from that sample, then we Crossover them and mutate each one of them, how exactly that happens will be explained shortly. This is repeated until either the GA did not found a better solution for 480 iterations or the time limit of 4 minutes got exceeded. This way the GA explores and exploits the neighborhood of very good solutions in order to find new better ones.

Mutation:

The mutation method is trying to swap the places of two libraries in the solution. The expected number of swaps is mutation probability * 5 this number seems super small, but it proved to be the perfect balance between keeping a decent amount of randomness but also making sure that each solution keeps its characteristics to some extent, then the chosen books must be fixed due to the possibility of library gaining days or losing them allowing for bigger or smaller amount of chosen books which must be taken care of. In what way we deal with that I will explain in the upcoming bullet point.

Crossover:

To crossover solutions we used the Merge Crossover approach. Which works in the following way, we iterate through both solutions and check the indices of libraries, if a library index is not in new solution1 then we add it and if it is then we add it to solution2. This way we ensure that no index appears twice which is desired for permutation type representations in GA. Whilst we are at it we create a new dictionary which consists of pairs idx1, idx2 which we use later on. Then we crossover the books between libraries with same id. The program, for each pair of libraries, takes the intersection of chosen books and fills it with sample

Figure 5 Example of merge crossover

from library from solution 1 and solution 2 so that the length of chosen books stays the same. Then after the crossover is done the books are fixed due to the possibility of library gaining or losing days.

Fixing books:

Fixing books is a process we do after any new solution is created out of neccesity to check whether all libraries scan the correct amount of books. Days_checker variable is set to the initial number of days, then the program iterates through all libraries in a solution each time assigning days_left field to days_checker variable and decreasing the days_checker variable by signup_time field. Then each library checks whether there are more or less books then $books\_per\_day * days\_left$ and if there are less then it adds random sample of books not yet chosen to the chosen books, if there are more books then it removes a random sample of chosen books. This way we ensure that each library scans correct amount of books while also introducing some slight randomness for exploration as well as keeping the main characteristics of solution in tact.

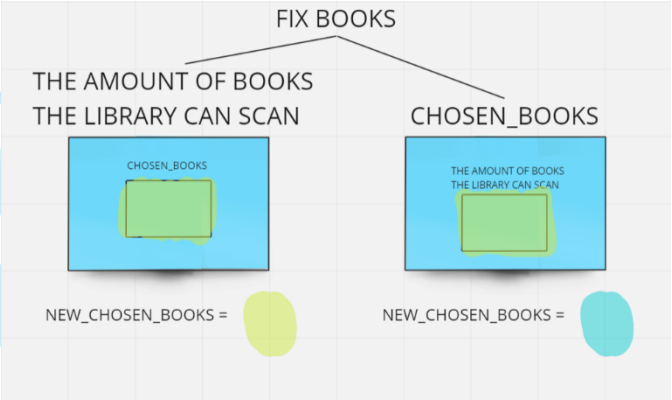

Figure 6 Visualization of fixing books

IV) Conclusion:

While trying to solve this problem we encountered many problems with GA approach both with representation and evolution. When we finally managed to make it work, its result were a let down to say the least. After introducing simple greedy population to hopefully improve its perfomence it jumped to producing very good solutions however its exploration and exploitation diminished vastly. We also tried optimizing hyperparameters of GA such as mutation probabilty, crossover probabilty, tournament size and population size however even though it seemed to make some difference it was negligible . Therefore we conclude that GA is not a good algorithm of choice for this problem and settling for just greedy or some other type of heuristic approach would be prefferable.