Book Scanning

Group: Group_A1_81
Bruno Fernandes (up202108871)
Catarina Canelas (up202103628)
Vasco Oliveira (up202108881)

Problem Description

The problem involves planning an efficient scanning process for books from various libraries to maximize the total score of all scanned books. Each book has a unique ID and can be found in multiple libraries, but only needs to be scanned once to count towards the score. Libraries must be signed up before they can send books to be scanned. The challenge is to determine which books to scan from which library within a given timeframe to achieve the highest possible score.

The way we decided to structurize our data was the following: The library object has a queue of books, the time to sign up and the number of books that it can scan each day. The book object has it’s score and an identifier.

Formulation of the problem

Solution Representation:

A list of Tuples, with each of this tuples having the library and the list of books that it will scan. Example of a solution: [(1,[5,2,3]), (0, [0,1,2,3,4])]

The score will be calculated by putting all scanned books in a set, so there are no duplicates, and then iterate through them to calculate the total sum of scores.

Neighborhood/Mutation functions and Crossover functions:

Library Swap: Swap the order in which two libraries are signed up for scanning.

Formulation of the problem

Hard Constraints:

- **Time limit**: There will be no action after the deadline, this meaning that, for example, we will never choose to sign up a library that won't have the possibility to scan any book before the deadline.
- **Overlapping sign ups**: There can't be multiple libraries signing up at the same time.
- **Only signed up can scan**: Only signed up libraries can scan books.
- **No book score repetitions**: The total score will only include each scanned book once, even if it is scanned multiple times.

Evaluation functions:

To evaluate the value of a solution we simply calculate the score iterating through all the books that were scanned on that solution. We take into account the deadline and the fact that a book can only add score for the first time that it has been scanned.

Heuristics / Implemented Algorithms

Starting Solution:

To calculate a starting solution used a Greedy approach, that sorts the book of each library from highest score to lowest using a priority queue, and picks the next library to be signed up by calculating the maximum possible score that that library can generate, taking into account the number of days left until the deadline. This algorithm gave us a good head start because it generated really good starting solutions. We also add 10% of the libraries left after reaching the deadline so that there are more neighbor variety.

In order to find the optimal solution, we implemented the following algorithms/heuristics:

Basic Hill Climbing:

In this Heuristic, we look through the current solution's neighbors until we find a better one, and than repeat that process for that neighbor until no better solution is found.

Steepest Ascent Hill Climbing:

In this heuristic, we look through all the neighbors of the current solution and find the one with the highest score from all of them. If that neighbor is better than the current solution, we them repeat this process for that neighbor until no better solution is found.

Heuristics / Implemented Algorithms

Simulated Annealing:

This heuristic selects a random neighbour and creates a new one, so if the score of this new solution turns out to be better, then the current solution gets updated. If the new score is not better, it can still be updated based on the probability of the difference between the new score and the current score (delta) and the current temperature. This is the key of Simulated Annealing, since it can allow bad moves in order of escaping local optima. Then, the temperature gets decreased, reducing the probability of accepting worse solutions.

Tabu Search combined with Simulated Annealing:

This heuristic consists in an implementation of the simulated annealing described above, with the addition of the concept used in a tabu search. Basically, we mark visited nodes (solutions) as tabu, so that we do not visit them again. This prevents the algorithm from getting in a loop. It also limits the number of tabu nodes (tenure) so that the node can be expanded once again later.

Results

The first two heuristics (hill climbing), did not have a good performance in general because it takes to much time to calculate every single neighbor of a solution. This was also the reason for us not implementing a simple tabu search algorithm.

So in general, the 2 simulated annealing implementations had good results, but there are no clear best because they both pick random neighbors, so the luck factor makes it so that they are both sometimes really good, and sometimes not that great.

In our interface, there is the possibility to run the optimization algorithms from the initial solution or from the current best solution, so it is always improving. In the next slide the is a print of our interface, showing our current results.

One factor that may be limiting our results is that not all libraries are being used because of our generation of the starting solution, that picks (in a greedy way) a limited number of libraries. One way we tried to combat this problem was to add extra libraries to the first solution, even though they were not signed up on time, more specifically 10% of the libraries left.

Results

The improvements aren't very significant, because our initial (greedy) solution is already optimized.

We believe that our initial solutions for the files *a_example.txt* and *b_read_on.txt* are already the optimal solutions, because we could not find a better solution in any of our tries using the several algorithms.

Scores

File	Initial Score	Best Score	Improvement (%)
a_example.txt	21	21	0.00%
b_read_on.txt	5822900	5822900	0.00%
c_incunabula.txt	5645747	5651078	0.09%
d_tough_choices.txt	4813510	4850885	0.78%
e_so_many_books.txt	4135783	4284645	3.60%
f_libraries_of_the_world.txt	5227124	5322062	1.82%
Total Score	25645085	25931591	1.12%

Conclusions

With the development of this project, we concluded that there are infinite ways to solve a problem with such high complexity, and that it is really important to make a plan and structure the problem before trying to solve it.

Given the obtained results, we also realized that algorithms based on random selections end up being more effective in finding optimal solutions.

References

- Official problem statement
- Python documentation
- Lecture Slides available in Moodle
- Streamlit (used for the user interface)
- Pandas Library (used for the dataframes showcased)