Google Books

Joaquim Carneiro
University of Porto
Faculty of Engineering
Porto, Portugal
up200203159@up.pt

João Patrício
University of Porto
Faculty of Engineering
Porto, Portugal
up202100812@up.pt

João Pedro Pêgo
Dept. of Civil Engineering
University of Porto
Faculty of Engineering
Porto, Portugal
up199502401@up.pt

***Abstract*—This article presents the work of the authors of implementing and testing three meta-heuristic search algorithms for a combinatory problem, namely of scheduling a set of libraries to sign up books for scanning. The problem was proposed by Google in their Online Qualification Round 2020 of team programming competition Hash Code. The problem consists of scheduling a set of libraries, whose goal is to maximize the total score obtained from scanning different books over a defined period of days. Two determinist solutions were found for the 6 problem files proposed during the competition. Furthermore, three optimization algorithms (simulated annealing, tabu search and genetic algorithm) were implemented to solve the problem starting from a random initial (non-optimal) solution. All three algorithms were able to find better solutions in a reasonable interval of time, though with a variety of performances. The experimental results demonstrate that at least two of the problems have a solution which exceeds the deterministic solutions found.**

Keywords— meta-heuristics search algorithms, optimization algorithms, simulated annealing, tabu search, genetic algorithm

I. INTRODUCTION

Hash Code is a team programming competition, organized by Google, for students and professionals around the world [1]. It allows participants to share skills and connect with other coders as they work together to solve a problem modeled off a real Google engineering challenge! In small teams of two to four, coders all over the world tackle the first problem through Qualification Round. Though this round is hosted online, teams can come together virtually to compete side-by-side in locally coordinated Hash Code Hubs. The top teams from this round are invited to join the virtual World Finals.

The problem that the authors present here was proposed as the challenge for the Qualification Round of 2020 [2] and is related to Google Books [3]. "Google Books is a project that embraces the value books bring to our daily lives. It aspires to bring the world's books online and make them accessible to everyone. In the last 15 years, Google Books has collected digital copies of 40 million books in more than 400 languages, partly by scanning books from libraries and publishers all around the world" [4].

II. SPECIFICATION

The book scanning problem is a scheduling problem whose goal is to maximize the total score obtained from scanning different Books from different Libraries over a defined period. The key decisions to be made are:

- What Libraries to be considered for scanning and in what order.
- What Books of a given library are considered for scanning and in what order.

Below are key additional aspects that influence the decision-making process:

- Sign up: The number of days a Library takes before it can start scanning Books. A Library can start the Sign Up process only after its preceding Library is signed up.
- Scan Daily Rate: The number of Books a Library can scan per day. The number of signed up libraries that can scan books on a given day is not constrained.
- Time: Total number of days for scanning books.
- Score: Value attributed to each scanned Book. A Book's 'score' is equal across libraries. A 'score' is not considered more than once, i.e. repeated scans only count as one. Equally, a score is not considered if a Book is dispatched for scanning after the established deadline.

The dataset provided during the competing included 6 different files, corresponding to 6 different problems, with the characteristics presented in TABLE I. .

TABLE I. PROBLEMS MAIN INDICATORS

File	B	L	D	Avg N	Total N
a	6	2	7	4	8
b	10^5	100	10^3	10^3	10^5
c	10^5	10^4	10^5	15	$15x10^4$
d	$78.6x10^3$	$30x10^3$	30001	7	$21x10^4$
e	10^5	10^3	200	491	$491\ x10^3$
f	10^5	10^3	700	509	$5.09\ x10^5$

L - the number of libraries

B - the number of different books in all libraries

D - the number of days available to ship books

Avg N – average number of books per library

Total N – total number of books to scan

A full specification of this problem can be found in [4].

III. BIBLIOGRAPHIC RESEARCH

The proposed challenge is one of scheduling a set of libraries for sign up and shipping books for scanning. Yet, since the scanning units have an unlimited capacity to scan books, that is, they can receive in the same day whatever number of books to scan, the problem simplifies. The solution will need to, in the first step, define a sequence of libraries to

sign up and, in the second step, define which of the books from each library will sum their score to the total score.

Three search algorithms were considered and studied for the current project: simulated annealing (SA), tabu search (TS) and genetic algorithms (GA). Each of the algorithms uses a meta heuristic to guide the algorithm in the search for possible solutions which shall be a local optimum, but not necessarily a global optimum. The three algorithms will be briefly explained next.

A. Simulated Annealing

SA is a stochastic global search optimization metaheuristic. described in 1983 by Kirkpatrick, Gelatt Jr. and Vecchi [5] for a solution of the traveling salesman problem.

Like other search methods, SA searches the neighbourhood of a given initial/current solution/state, however it may move to a worse solution. The general idea is that by applying a 'temperature schedule', SA uses a slow descending likelihood of accepting worse solutions, the metropolis criterion, high at the beginning (as in random walk) and decreasing with the progress of the search (as in hill climbing), allowing for a more extensive search for the global optimal solution.

Below is the pseudocode of a simple version of SA:

```
t0 ← initial temperature
s ← a random initial solution
k ← maximum number of iterations

for i in 1 to k do
   T ← schedule (i), for example t0 / (i+1)
   sneighor ← neighbour(s)
   ΔE ← evaluation (sneighbour) - evaluation (s)
   if ΔE > 0
        s ← sneighbour
        sbest ← sneighbour
   else
        s ← sneighbour if e^(ΔE/T) ⩾ random(0,1)

return best
```

where:

- schedule() – returns a decreasing temperature;
- neighbour() – finds a random neighbour;
- evaluation() – evaluates the solution.

This is a memoryless, efficient, and complete algorithm (as long as the schedule lowers T slowly enough).

B. Tabu Search

Tabu search is a metaheuristic search method that employs local search (LS) methods for the mathematic optimization of a problem. It was proposed by Fred W. Glover ([6],[7],[8]) in the 1980's. Tabu search attempts to escape local optima by searching a neighbourhood of the current candidate solution for better solutions, while tracking previously searched solutions (so called tabus) in a tabu list. Because of this feature, it is memory demanding. To limit the amount of memory used, the tabu search method deletes some of the previous solutions from the tabu list. A solution that belongs to the tabu list is not admissible, unless it fulfils a given aspiration criteria.

The pseudocode for a simplified version of the TS algorithm is provided below according to [9]:

```
sBest ← s0
bestCandidate ← s0
tabuList ← []
tabuList.push(s0)
while (not stoppingCondition())
  sNeighbourhood ← getNeighbours(bestCandidate)
  bestCandidate ← sNeighbourhood[0]
  for (sCandidate in sNeighbourhood)
    if ((not tabuList.contains(sCandidate)) and
(fitness(sCandidate) > fitness(bestCandidate)))
      bestCandidate ← sCandidate
    end if
  end for
  if (fitness(bestCandidate) > fitness(sBest))
    sBest ← bestCandidate
  end if
  tabuList.push(bestCandidate)
  if (tabuList.size > maxTabuSize)
    tabuList.removeFirst()
  end if
end while
return sBest
```

where:

- S0 – initial solution;
- s_{Best} – best solution;
- $best_{Candidate}$ – candidate to best solution;
- tabuList – list of already visited solutions;
- $s_{Neighbourhood}$ – list of neighbourhood solutions to search;
- $s_{Candidate}$ – candidate solution;
- fitness () – function to evaluate the solution.

In this pseudocode a short-term memory in the form of the tabu list is included. It also assumes the tabu list stores the complete information of previously visited solutions, which is not necessarily the case, and will not be the type of implementation used in the present work.

C. Genetic Algorithm

Although the idea of an artificial selection was primarily proposed by the geneticist Alex Fraser in 1957, it was only during the 1960s that optimization methods based on artificial evolution were widely recognized. This was mainly due to the work performed by Ingo Rechenberg and Hans-Paul Schwefel, which were able to apply evolution strategies to solve complex engineering problems. It was only on the early 1970's when genetic algorithms became popular, due to the work developed by John Holland, resulting on his book "Adaptation in Natural and Artificial Systems". [10]

GAs are inspired in natural selection behaviours, where an initial population of individuals is generated randomly. Then, a group of population's individuals are selected based on their fitness function (objective function). At the end, the offspring is generated by the reproduction of two different parents, based on crossover and mutation reproduction processes. The idea of crossover is to bring diversification to the population in an early stage, resulting (like simulated annealing) in large steps in the search space. Then, as the search process evolves smaller steps are adopted, as the similarity between individuals increases. At the end, the cycle is restarted till the number of generations is reached. [11]

The pseudocode for a standard GA application is stated below according to [12].

```
Randomly initialize population
for (g=1 to nGenerations)
  for (i=1 to nPopulation)
     evaluate fitness function of individual i
  end for
  save best individual to population g+1
  for (i=2 to nPop)
     select two individuals
     crossover: creates two new individuals
     mutate the new individuals
     move new individuals to population g+1
  end for
end for
```

where:

- nGenerations – number of generations;
- nPopulation – population size.

In the following section it will be described how the GA's main hyperparameters – population size, number of generations, selection method, crossover rate, and mutation rate – could work together to increase the explored search space, to look for better problem's solutions.

IV. Formulation of the Problem

A. Solution Representation

Reflecting the key decisions of this problem, the ideal solution representation should indicate

- The order in which each Library will be scanned; and
- The order in which each Book of a given Library will be scanned.

This would possibly require all libraries and all books represented in the solution.

To simplify this, a solution design constraint has been adopted, i.e., books within a library as assumed to be scanned prioritized by respective scores. In other words, books with higher score will be scanned first.

This simplifies the above points into a solution representation that indicates

- The order in which each Library will be scanned (same as before); and
- In which book position (ordered by score) the scanning will start for a given library.

With this simplification the solution representation does not need to represent all the books but simply a position where to start scanning. The solution representation will take the following form:

[(library_a, book_pos_a), (library_b, book_pos_b), (library_c, book_pos_c), …]

In this representation library_a will start scanning first, followed by library_b, etc… In library_a, the first book being scanned will be the one in position book_pos_a. The picture below illustrates this with an example.

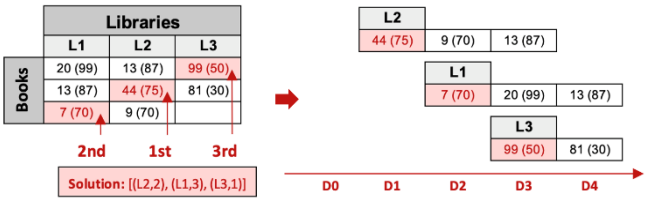

Fig. 1 Solution representation

B. Evaluation Function

The evaluation function consists in the summation of the scores of the books scanned for the time set for the problem. This considers that:

- A book has the same score across libraries;
- A score is not considered more than once in case of repeated books;
- Each library scans a different number of books each day;
- Each library needs a few sign up days before scanning starts.

In the example from the previous section, the 'Scan Daily Rate' of each individual library is 1 book and the 'Sign Up' days of each individual library is 1 day. If the scanning 'Time' for the problem was 4 days, the scores for Book 13 from Library L1 and Book 81 from Library L3 wouldn't be considered (total score of 451). On the other hand, if this scanning 'Time' was 5 days, the score for Book 13 from Library L1 wouldn't be considered again, this time because this was a repeated book.

C. Simulated Annealing

Neighbourhood

A new solution S2 is neighbour to a current solution S1 when S1 can be transformed into S2 by:

Swapping two libraries: if S1is [(L2,2), (L1,2), (L3,1)] and S2 is [(L1,2), (L2,2), (L3,1)], S1 and S2 are neighbours because S2 is equivalent to swapping the first and second libraries in S1.

Rotating libraries: if S1is [(L2,2), (L1,2), (L3,1)] and S2 is [(L1,2), (L3,1), (L2,2)], S1 and S2 are neighbours because S2 is the equivalent to moving the first library of S1 to the end of the sequence.

Rotating books: if S1 is [(L2,2), (L1,2), (L3,1)] and S2 is [(L2,1), (L1,2), (L3,1)], S1 and S2 are neighbours because S2 is the equivalent to starting the scanning of library L2 books from the first book, instead of the second.

A concept of neighbour distance is also applied and is defined based on i) the number of rotations applied to libraries, ii) the number of rotations applied to books and iii) how far two libraries are in the sequence of libraries.

D. Tabu Search

The implemented tabu search algorithm uses the same base formulation as the other two algorithms in what concerns

the solution representation (a list of tuples with pairs (Library, First Book to scan)) and the evaluation function.

The neighbourhood used to search for new solutions was defined in a different way. The function to generate the neighbourhood takes two approaches, addressing the permutation of the position of libraries (without changing the book order) or changing the order of the books list for a given library.

The first approach takes the following steps:

1. Select a tuple (L1, B1) at random, that is, computes an index idxL1 in the solution list, between 0 and L-1.
2. Compute the distance to the second library, L2, whose position is to be permutated with L1. The distance is computed at random from a minimum and a maximum limit, which allows for diversification of solutions.
3. At random select the library L2, either to the left or right of L1. That is, idxL2 = idxL1 +/- distance.

For the second approach, the permutations of initial book of a given library is done by randomly selecting a distance between books in the list book that is between 1 and a maximum distance provided to the algorithm.

Each time it is called, the neighbour generating function chooses at random between the first and second approach, with a probability of LBbal for the first approach. The reason for this is because exchanging libraries positions has a more significant impact in the total score of the problem than exchanging book orders, which only affects the libraries close to the deadline for book shipping. The maximum number of neighbours to visit was set to 25.

The neighbour generating function returns not only the candidate solution, but also the pair of exchanged nodes, (L1, B1) and (L2, B2), together with the exchanged positions (idxL1, idxL2).

Once the neighbourhood of the current solution is defined the local search algorithm uses a “Best accept” approach, that is it will the select the neighbour that has the highest score among all neighbour candidate candidates. If none found, it returns the current solution.

Each time a new candidate is found, the LS algorithm tests if it is inside the tabu list, both in terms of the nodes ((L1, B1) and (L2, B2)) and the positions (idxL1, idxL2). If it is, the candidate solution is not considered.

The tabu list consists of the nodes and indices of the form (((L1, B1), (L2, B2), (idxL1, idxL2)), which represent the permutations done in each iteration. Each item in the tabu list is initialized with a given penalty (tenure), which is the number of iterations such tabu or its inverse cannot be considered in future solutions. If a new candidate solution contained nodes that already existed in the tabu list, for all solutions in the tabu list that used the same nodes the penalty was reset to the tenure value. In each iteration of the TS algorithm, the penalty of the tuples in the tabu list was reduced of 1 unit and when it reached zero, the tuple was removed from the tabu list.

The following hyper parameters were used in the TS algorithm:

- max_iter – maximum number of iterations of the TS algorithm;
- max_iter_LS – maximum number of iterations of the LS algorithm;
- dist_L_min – minimum distance between neighbour libraries;
- dist_L_max – maximum distance between neighbour libraries;
- dist_B – maximum distance between neighbour books;
- tenure – number of iterations a solution remains in the tabu list;
- LBbal – threshold above which book list permutations occur when generating neighbours.

When searching the tabu list, it is recommended to include not only relation between node A and node B, but also between node B and node A. For example, if a new neighbour is the permutation involving library L1 and book B1 (L1, B1) with library L2 and book B2 (L2, B2) then the tabu list should be searched both for ((L1, B1), (L2, B2)) and ((L2, B2), (L1, B1)). To speed up the optimization, for each candidate solution tested which was not in the tabu list, both ((L1, B1), (L2, B2)) and ((L2, B2), (L1, B1)) were stored in the tabu list. This meant that the double of the memory is necessary, but with the advantage of reducing the search time for the next iterations.

The maximum expected memory usage corresponds to the maximum size of the tabu list, which was not limited, since the aspiration criterion would allow for a tuple to remain in the tabu list for longer than the tenure. The experimental results showed that the maximum size of the tabu list rarely exceeded the double of the tenure value.

The maximum number of candidate solutions searched is given by max_iter * max_iter _LS.

E. Genetic Algorithm

Although many hyperparameters could be used to develop a more complex GA approach, our mainly goal was to reach a trade-off between algorithm complexity and output robustness. Therefore, seven hyperparameters were stablished, and their detailed description is described below:

- Population size – the number of individuals in a population, in other words, the number of starting solutions. For instance, a population size with two individuals could be described as, [[(L1,2), (L2,4), (L3,6)], [[(L1,1), (L2,2), (L3,3)]]. Using high population’s sizes, increases the probability to obtain additional better solutions, along the algorithm run. Nevertheless, the algorithm’s computation time increases with the population size. Therefore, a trade-off between population size and computation time needs also to be evaluated previously. The GA implemented considers that the population size is static along the algorithm run (there is no variation in the population size along the algorithm run).
- Number of generations – the number of algorithm’s iterations. Using a high number of generations increases the probability to obtain additional better solutions, since

the algorithm has additional opportunities to perform crossover and mutation operations in their population's individuals, resulting in an increased explored search space. As expected, the algorithm computation time increases with the number of generations.

- Selection type – the method used to perform the individual's selection, to participate in the reproduction phase. The GA implemented considers that the individuals' selection type is done by tournament, where a percentage of the population is chosen to participate (selectionRate hyperparameter), and a number of participants in each tournament is also previously defined (nParti hyperparameters). The winner of each tournament is defined based on his fitness function value (solution score). In the selection phase, the goal is to increase the probability of having the best individuals (with higher solution scores) selected for the reproduction phase, while the worst individuals (with lower solution scores) are disregarded [13].
- Crossover rate – determines whether a crossover operation is performed (or not) between two parents (P1 and P2), when compared with a randomly generated value. A random crossover point is stablished (random library position). The GA implement considers one-point crossover operations, where the children (C1 and C2) will receive the related parent's genetic information till the crossover point (represented in the example by "|"), and the second parent's genetic information (in the same order of occurrence) after the crossover point, for instance:

  P1: [(L1,2), | (L3,6), (L2,4)] → C1: [(L1,2), (L2,2), (L3,3)]

  P2: [(L1,1), | (L2,2), (L3,3)] → C2: [(L1,1), (L3,6), (L2,4)]

  Crossover only uses information that is already embedded in the population (from parents), and tries different combinations (generating children), by changing the solution order (libraries order), without specifically bringing new information into the population [13]. New information could be achieved by mutation operations.
- Mutation rate – determines whether a mutation operation is performed (or not) in a child (C1 and C2), by rotating the ID of the book in a randomly chosen library, as presented in the two examples below:

  C1: [(L1,2), (L2,4), (L3,6)] → C1: [(L1,2), (L2,4), (L3,7)]

  C2: [(L1,1), (L2,2), (L3,3)] → C2: [(L1,1), (L2,3), (L3,3)]

  By performing a gene modification (book rotation), the mutation process is able to bring new information to the population [13].

V. Implementation Details

The algorithms were programmed in Python (version 3.9), using Deepnote [14] as development environment. Deepnote allows for collaborative programming and provides sufficient computational power in a device independent manner to allow for relative comparison between different solutions and algorithms.

A. Input files

The input files of the problem are structured according to the specifications of the competition as follows:

The first line of the data set contains:

- B - the number of different books (0 <= B <= 1E5);
- L - the number of libraries (0 <= L <= 1E5);
- D - the number of days available to ship books for scanning (0 <= D <= 1E5).

This is followed by one line containing B integers, S[i], i = 0... B-1, describing the score of individual books, from book 0 to book B-1.

This is followed by L sections describing individual libraries from library 0 to library L-1. Each such section contains two lines:

The first line, which contains:

- N_j- number of books in library j, j=0… L;
- T_j (0 <= T_j <= 1E5) - the time in days that it takes to sign up library j for scanning, j=0… L;
- M_j (0 <= M_j <= 1E5) - the number of books that can be shipped from library j to the scanning facility, once the library is signed up, j=0… L;

The second line, which contains N_j integers, describing the IDs of the books in the library. Each book ID is listed at most once per library.

The total number of books in all libraries does not exceed 1E6.

B. Search space

The search space will be given by the product of possible combinations of library sequences with the number of possible combination of books order inside each library. Using the previous notation, the search space, S, will be

$$S = L! \cdot \prod_{j=0}^{L-1} N_j! \quad (1)$$

Yet, since in our approach to the problem we opted to sort each library list of books by decreasing score value, the reduced search space, S_{red}, will be given by:

$$S_{red} = L! \cdot \prod_{j=0}^{L-1} N_j \quad (2)$$

TABLE II. lists the size of the search space for each file.

TABLE II. Search space

File	S	S_{red}	α
a	$10^{3.76}$	$10^{1.6}$	0.19
b	10^{256918}	10^{457}	0.34
c	10^{158576}	10^{47328}	0.75
d	10^{244680}	10^{146863}	0.83
e	$10^{1156803}$	10^{5124}	0.50
f	$10^{1203551}$	10^{5141}	0.50

C. Data structure

Below are the key data entities used to represent this problem:

- Book - This entity represents a book; it is defined with an id and a score used in the evaluation / fitness of a given solution.
- Library - This entity represents a library. It is defined with an id, a sign up number of days, a scanning daily rate and an ordered collection of Books. The ordering of the books is performed as part of the data loading process.
- Problem - This entity represents a specific scenario where Libraries and Books give shape to the problem's search space; it is defined with a total number of days for scanning books and a collection of Libraries. It is responsible for i) defining viable random solutions, ii) evaluating a solution and iii) identifying viable neighbours for a given solution.

A Problem must be loaded before a search algorithm can start looking for a solution.

VI. Algorithms Implemented

As stated before, three search algorithms were implemented for the current problem: simulated annealing, tabu search and genetic algorithm. The details of each individual implementation are provided next. Whenever possible, libraries and routines were shared among the three algorithms.

A. Simulated Annealing

The SA algorithm implemented is an adaptation of [15]. The main adjustments relate to i) the usage of neighbours at varying distances, ii) adaptive search repetitions throughout the temperature cooling and iii) the use of different temperature schedules.

Packaged alternatives were considered but discarded given the simplicity of the algorithm and the complexity of the solution representation.

B. Tabu Search

The pseudocode for the implemented TS algorithm is as follows:

```
sBest ← s0
bestCandidate ← s0
tabuList ← []
tabuList.push(s0, tenure)
while (not stoppingCondition())
  sNeighbourhood ← getNeighbours(bestCandidate)
  bestCandidate ← sNeighbourhood[0]
  for (sCandidate in sNeighbourhood)
    if tabuList.contains(sCandidate)
      end while
    end if
    if(fitness(sCandidate)>fitness(bestCandidate))
      bestCandidate ← sCandidate
    end if
  end for
  if tabuList.contains(sCandidate.nodes)
    tabuList.push(nodes, tenure)
  end if
  if (fitness(bestCandidate) > fitness(sBest))
    sBest ← bestCandidate
  end if
  tabuList.push(bestCandidate, tenure)
  if (tabuList[i] < 0)
    tabuList.remove(tabuList[i])
  end if
end while
return sBest
```

The TS algorithm uses a short-term memory, with an aspiration criterion to handle previously visited solutions. The neighbourhood is searched using a "Best Accept" approach to select the next candidate solution. The present implementation of the TS algorithm combined not only the knowledge acquired during the course unit on Artificial Intelligence and the bibliographic survey but also by analysing publicly available implementations of the algorithm. For this work the implementations of [16] and [17] were of great inspiration. Unfortunately, a generic implementation of the TS algorithm was not readily available and, in general, the available implementations are also very much problem dependent.

The reduced search space of equation (2) can be seen as product of two constants,

$$S_{red} = a \cdot b, \tag{3}$$

where $a = L!$ only depends on the number of permutations of the libraries sequence and $b = \prod_{j=0}^{L-1} N_j$ depends on the number of permutations of the starting position of the books list in each library. If one applies the logarithmic function to equation (3), one obtains equation (4).

$$\log_2(S_{red}) = \log_2(a) + \log_2(b) \tag{4}$$

We can now define the right-hand side terms as a function of the search space logarithm as follows.

$$\log_2(a) = \alpha \cdot \log_2(S_{red}) \tag{5}$$

$$\log_2(b) = \beta \cdot \log_2(S_{red}) \tag{6}$$

Substituting equations (4) and (5) in equation (3) we obtain:

$$\log_2(S_{red}) = \alpha \cdot \log_2(S_{red}) + \beta \cdot \log_2(S_{red}) \tag{7}$$

From equation (7) results equation (8),

$$\alpha + \beta = 1 \tag{8}$$

We can interpret the values of α and β as the relative weight of each type of permutations (a or b) in the construction of the search space. If $\alpha > \beta$ this means that the permutations in the libraries sequence have a higher weight on the total combinations of the search space. The parameter α, which is presented in TABLE II. can thus be used as threshold level, LBbal, for the neighbours generating function.

C. Genetic Algorithm

The pseudocode for the implemented GA algorithm is as follows:

```
Randomly initialize population
for (g=1 to nGenerations)
 evaluate fitness function of all population
 individuals
 for (i=1 to nSelection)
```

```
    Tournament of nParti individuals randomly
    chosen
  end for
  for (i=1 to selectionWinners)
    Parents1 = selectionWinners
    Parents2 = randPopulation
    Create pairs (parents1, parents2)
  end for
  for (i=1 to Parents1+Parents2)
    if (crossoverRate > randomNumber)
    crossover
    end if
      if (mutationRate > randomNumber)
      mutation
      end if
  end for
  for (i=1 to (nPop-(Parents1+Parents2))
    Add best population individuals till dimension
    equals to nPopulation
  end for
end for
```

where:

- nGenerations – number of generations;
- nPopulation – population size;
- nSelection – number of individuals chosen for selection (tournament);
- nParti – number of individuals listed in the tournament;
- selectionWinners – tournament winner individuals (parents1);
- randPopulation – individuals randomly selected from the population (parents2);
- crossoverRate – crossover percentage of occurrence;
- mutationRate – mutation percentage of occurrence.

The present implementation is therefore able to apply the key principles of a GA, where a population is randomly generated and their evaluation using a fitness function is performed. Then, a selection of individuals is made by tournament and groups of parents are generated (where the first are the tournament winners, and the second are randomly selected population's individuals). At this point reproduction can take place, where crossover and mutation operations generate new children. The offspring is completed by adding the best population individuals to the children already available, using an elitist approach, which guarantee that the best population individuals found so far, are kept as members of the population. After that the loop is reinitiated [18].

Different GAs implementations could be adopted since a high diversity of approaches can be found in the literature. The one presented was mainly inspired by the course unit on Artificial Intelligence, as well as, the related bibliography [11] and additional content available online ([19],[18],[13]).

VII. EXPERIMENTAL RESULTS

A. *Deterministic solutions exploratory analysis*

The best solution to this problem is the one that maximizes the total score of books shipped. The constrains to the problem are the sign up period of each library, the maximum daily scan rate of each library and the number of days available to send books to the scanning facility. The libraries that are placed at the end of the queue will, eventually, not contribute to the total score because the deadline to scan books is already passed. So, one strategy would be to place as many libraries as possible at the beginning of the queue sending books in parallel, that is, placing at the beginning of the queue the libraries with short sign up processes. On the other hand, libraries that have a list of books with higher scores will have, in principle, a higher contribution to the total score, if they are placed at the beginning of the queue. Based on these assumptions, two "deterministic solutions" were prepared for evaluation:

1. Libraries sorted by decreasing ratio of "total score of library's books" / "sign up period of the library". In case of tie, the libraries are sorted by decreasing "total score of library's books", followed by decreasing "maximum daily scan rate".
2. Libraries sorted by increasing "sign up period of the library". In case of tie, the libraries are sorted by decreasing "total score of library's books", followed by decreasing "maximum daily scan rate".

In both cases, it is assumed the list of books of each library is sorted in decreasing order of book score. The total score of the two "deterministic solutions" for each of the 6 problems is provided in TABLE II.

TABLE III. DETERMINISTIC SOLUTIONS SCORES

File	Sorted by "books score / sign up period"	Sorted by 1 / "sign up period"
a	21	21
b	5 822 900	5 822 900
c	5 645 747	5 467 966
d	4 815 395	4 815 395
e	3 714 416	3 977 298
f	5 227 905	2 703 359

The maximum score of the set of problems totals 25 489 266. This compares well with some solutions made public by participants in this challenge ([20],[21],[22],[23]).

Problems a to d have similar or identical values for the total scores. Problem e has a more significant deviation in both cases with a higher score when sorted by "sign up period", while problem f shows clearly that there are significant differences among the two solutions, with a higher value when sorted by "books score / sign up period".

Since these solutions do not depend on any optimization algorithm, they can be used to benchmark optimized solutions. If they happen to be optimal, that is, there is no better solution to the problem than one of these, you can clearly state the problem is deterministic. Due to its size and convergence behaviours, file b was selected to compare the three algorithms. Files c and d result in search spaces so big that convergence to the solution was not found before completing this article.

B. *Algorithms benchmark*

To assess the algorithms capacity to correctly optimize solutions, the authors created a simple problem with four libraries, each with a signup period of 1 day, 12 books to scan and 5 days available to scan books, the solution of which is represented in Fig. 2. The three algorithms successfully reached the optimum solution within few iterations.

	D0	D1		D2		D3		D4		D5
L0	sign-up	B0	B1	B2	B3	B4				
		12	11	10	9	8				
L1	wait	sign-up		B5	B6	B7	B0			
				7	6	5	12			
L2	wait	wait		sign-up		B8	B9	B0		
						4	3	12		
L3	wait	wait		wait		sign-up		B10	B11	B5
								2	1	5

deadline

Fig. 2 Benchmark problem solution

C. *Comparative tests and performance indicators*

The three algorithms were tested for the same initial solution, so to make the different performance criteria comparable. The initial solution was generated randomly for file b. All algorithms were run through enough iterations or generations to test 20 000 solutions for each set of hyper-parameters. The performance indicators selected were the total score obtained (best value), the number of iterations until the optimum was found and the time taken for the calculations. The latter is a rather subjective comparative term, since different algorithms were programmed by different persons and also because some algorithms use more memory than others. For example, the tabu search algorithm requires that in each iteration the candidate solution must not be in the tabu list, requiring a search procedure which consumes time and is exponentially proportional to the size of the problem.

D. *Simulated Annealing*

Simulated Annealing was performed in different conditions. Those with more impact on the solution evaluation relate to variations in i) the temperature schedule, ii) the neighbour distance and used of iii) adaptive repetitions.

Both a slow and fast cooling temperature schedules where considered.

$$T_{fast} = \frac{T_0}{i} \quad (9)$$

$$T_{slow} = T_0 \times \alpha^i \quad (10)$$

where:

T_0 – initial temperature;

i – iteration number;

To model the decreasing probability of accepting worse solutions, according to the Metropolis criterion,

$$e^{(\frac{\Delta E}{T})} \quad (11)$$

the following parameters were considered,

- Initial temperature T_0;
- Maximum number of iterations i;
- Average ΔE, determined based on a 'random walk' in the search space.

resulting in the probability curves below.

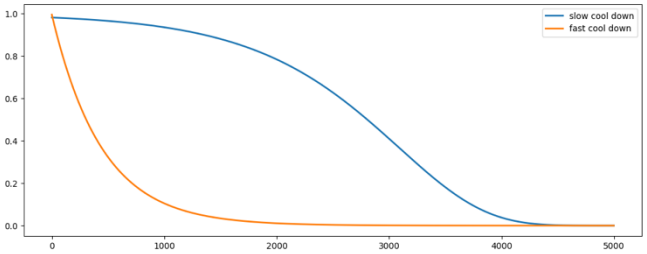

Fig. 3 Probability curves according to the Metropolis criterion for $i_{max} = 5000$, $T_0 = 5000$, $\alpha = 0.998$ for the slow cool down and $i_{max} = 5000$, $T_0 = 40000$ for the fast cool down.

The table below lists the different parameters tested with the SA algorithm.

TABLE IV. SCENARIOS TESTED FOR THE SA ALGORITHM

Parameter	Scenario 1	Scenario 2	Scenario 3	Scenario 4
Initial Solution	Random Solution	Random Solution	Random Solution	Determ. Solution
Temperature Schedule	Fast	Slow	Slow	Slow
Neighbour Distance	30	30	100	1
Max. no. of Repetitions	10	10	10	10
Best Score	4 615 200	4 763 800	5 318 300	5 822 900

In Fig. 4, with a fast schedule, the algorithm is locked in a local optimum very early.

In Fig. 5, the scenario changes to a slow schedule, the algorithm searches for worse alternative solutions for longer which leads to an improved best solution.

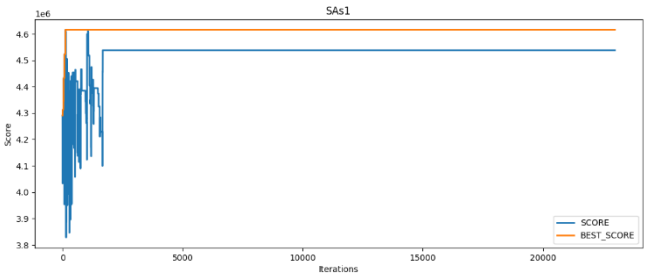

Fig. 4 SA evaluation using 5k iterations (with up to 10k repetitions results in above 20k iterations), fast temperature schedule, up to 30 steps distance neighbours, starting from a random solution.

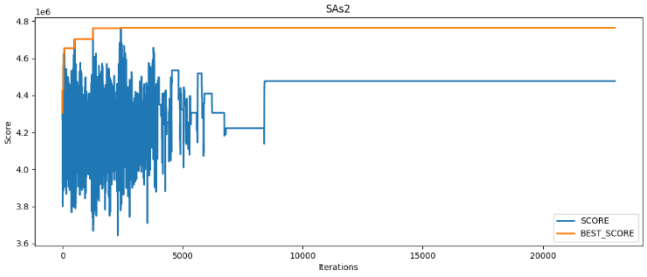

Fig. 5 SA evaluation using 5k iterations (with up to 10k repetitions results in above 20k iterations), slow temperature schedule and up to 30 steps distance neighbours, starting from a random solution.

In Fig. 6, the scenario changes so that the algorithm searches for farther neighbours, further allowing the algorithm to escape local optima which leads to an even better solution.

Interestingly, and because of the adaptive number of repetitions, small increases in the best solution are observed as the algorithm progresses towards the stop criteria.

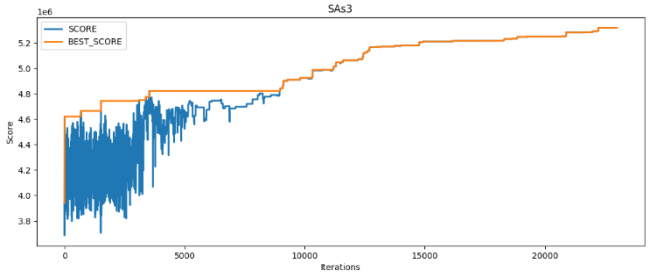

Fig. 6 SA evaluation using 5k iterations (with up to 10k repetitions results in above 20k iterations), slow temperature schedule and up to 100 steps distance neighbours, starting from a random solution.

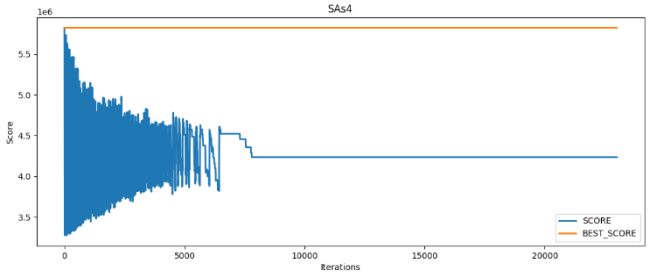

Fig. 7 SA evaluation using 5k iterations (with up to 10k repetitions results in above 20k iterations), slow temperature schedule and up to 1 steps distance neighbours, starting from the best 'deterministic solution'.

Finally, and just as a reference, applying the same parameters but using the deterministic solution as the initial solution, even when moving only to the closest possible neighbours, the algorithm is not able to find a solution better than the SA 'deterministic solution'.

Additional scenarios have also been tested with inferior results. In detail

- Scenarios with shorter distance neighbours, given the dimension of search space, had always considerable inferior solutions.
- No search repetitions resulted in an early stabilization of the best solution.

E. Tabu Search

The set of parameters presented in TABLE V. were used as basis for the computations using the tabu search algorithm.

TABLE V. TABU SEARCH HYPERPARAMETERS

Parameter	Value
max_iter	800
max_iter_LS	25
dist_L_min	coef_Lmin * nr_libs
dist_L_max	coef_Lmax * nr_libs
dist_B	0.5 * $nbooks$
tenure	$\sqrt{nr\_libs}$
LBbal	threshold

Here, nr_libs is the number of libraries of the problem and:

$$coef_{Lmin} \in [0.05, 0.15, 0.25] \quad (12)$$

$$coef_{Lmax} \in [0.25, 0.50, 0.75] \quad (13)$$

$$threshold \in [0.1, \alpha,\ 0.9] \quad (14)$$

When $threshold$ = 0.1, neighbour generation was mainly achieved by permutations of the book order inside the libraries, while when $threshold$ = 0.9, neighbours are generated mainly by permutations of the libraries order. The value of α is given by equation (5).

Fig. 8 shows the convergence to the deterministic solution starting with a random initial solution, where the orange line represents the best solution obtained until that iteration and the blue line corresponds to the evaluation of the solution in that iteration. Fig. 9 to Fig. 11 show the graphs of convergence for the three hyper-parameters mentioned before, when all the others remain constant. In Fig. 9 it is clear to see that the permutations in the book order led to a slower convergence, while a higher number of permutations in the libraries order will lead to a faster convergence. On the other hand, the assumption that $LBbal = \alpha$ would better "explain" the problem seems to be misleading, as it did not produce better results than when $LBbal = 0.9$.

Fig. 10 shows that the minimum distance between libraries when creating neighbours does not seem to significantly affect the convergence in the earlier stages but that will affect the convergence of the solution. If dist_L_min is set too high, this will lead to a situation where the algorithm does not converge to the global optimum or does it much slower. This could be easily explained because when the algorithm is close to the optimum solution, this factor hinders the search in solutions nearby.

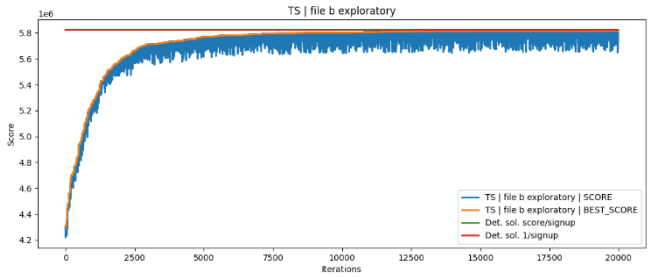

Fig. 8 TS score convergence for the solution with standard hyper-parameters ($coef_{Lmin}$ = 0.05, $coef_{Lmax}$ = 0.50, $LBbal$ = 0.34). Score = 5 819 200

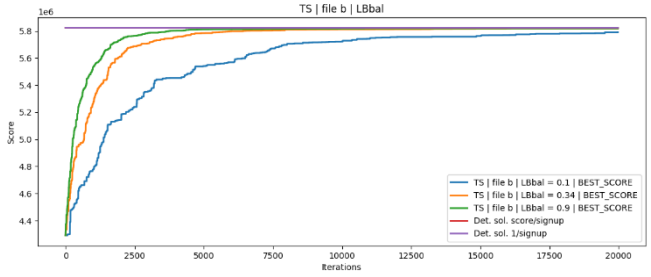

Fig. 9 TS score convergence for different values of $LBbal$ ($coef_{Lmin}$ = 0.05, $coef_{Lmax}$ = 0.50)

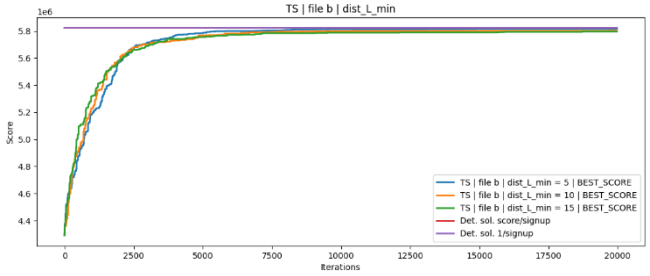

Fig. 10 TS score convergence for different values of $coef_{Lmin}(coef_{Lmax} = 0.50, LBbal = 0.34)$

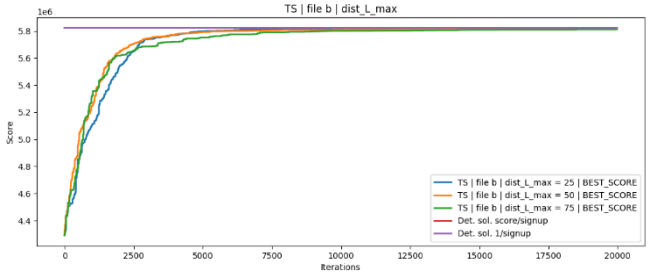

Fig. 11 TS score convergence for different values of $coef_{Lmax}(coef_{Lmin} = 0.05, LBbal = 0.34$)

Finally, Fig. 11 demonstrates the influence of dist_L_max. Setting up this value too low can lead to a convergence to a local optimum, as the global optimum is missed by the algorithm. Yet, setting up this value to much larger than 50% of nr_libs will not lead to much better results, as with this size of the search distance it is possible to cover the whole range of positions of the libraries, from 0 to $L - 1$.

Based on the experience obtained with the previous tests, modifications were introduced to the tabu search algorithm which improve its performance in terms of convergence to the optimum solution. Since the first iterations of the algorithm are used to find the best sequence of libraries, it is advantageous to have large values of dist_L_max, dist_L_min and LBbal in the first iterations. This setting allows to search for neighbour solutions which produce permutation of libraries positions that are situated in the extremes of the queue, some of which are placed past the deadline and so do not contribute to the total score of the solution. As the algorithm converges to the optimum solution, finding neighbours that produce permutations between libraries that are close by in the queue and having more permutations in the book order will conduct to a refined search algorithm. Following this idea, a new function was introduced that allowed for large values of the three hyper-parameters for the first iterations, while progressively decreasing their value the further the process progressed. The function used an exponential decay from a maximum/minimum value at the first iteration to a minimum/maximum value at the last iteration, according to equation (14)

$$f_{iter} = f_a + (f_i - f_a) \cdot b^{c(iter-i)} \qquad (15)$$

Where f_{iter} is the value of the function at iteration $iter$, f_a is the asymptotic value of the function for infinity, f_i is the value of the function at the initial iteration, i, b is a base and c is the exponential constant. To find the value of the c, knowing two points of the curve, such as the initial and the point past which the function becomes asymptotic, it is possible to use equation (15).

$$c = \frac{\log_b(|f_n - f_a|) - \log_b(|f_i - f_a|)}{n - i} \qquad (16)$$

Here, n is the iteration after which the function flattens, that is $f_n \cong f_a$. Fig. 12 shows the exponential decay function that was used to compute the value of dist_L_max for each of the TS iterations, applied to file b. The function has a constant that creates of range for dist_L_max between 100, at iteration 1, and 10 at iteration 200, with an asymptotic value of 3.

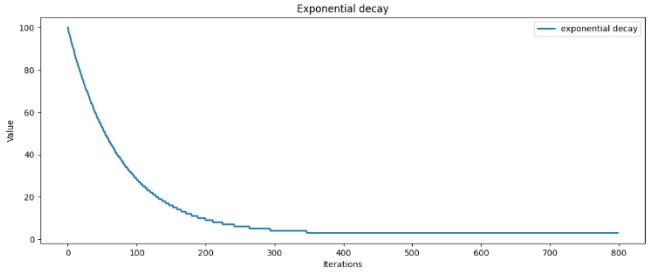

Fig. 12 Exponential decay function for the adaptative algorithm

The values used to define the exponential decay functions of dist_L_max, dist_L_min and LBbal are provided in TABLE VI. As can be seen in Fig. 13, the optimized algorithm produced similar results to those of Fig. 8, while converging much faster to the optimum solution. It also used a stopping criterion which would check for the convergence of the value by taking the normalized derivative of the last solutions values, where the number of solutions to consider was equal to 10% of the maximum number of iterations.

$$sc = \text{iter} < \max\_iter \wedge \frac{\sigma_{\Delta f}}{\mu_{\Delta f}} > 10^{-6} \qquad (17)$$

Here, sc is the stopping criterion, max _iter is the maximum number of iterations allowed, Δf is the difference in the function value between two consecutive iterations, $\mu_{\Delta f}$ and $\sigma_{\Delta f}$ are the average and standard deviation of said differences. The algorithm stops when the stopping criterion becomes false.

TABLE VI. EXPONENTIAL DECAY FUNCTION COEFFICIENTS

Parameter	f_a	f_i	f_n	n
dist_L_min	2	$25\% nr\_libs$	$10\% f_i$	$\frac{\max\_iter}{2}$
dist_L_max	1	nr_libs	$10\% f_i$	$\frac{\max\_iter}{2}$
LBbal	0.0	α	0.01	$\max\_iter$

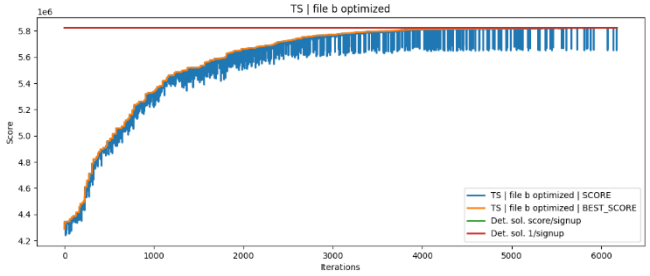

Fig. 13 TS score convergence for the solution with adaptative algorithm (hyper-parameters at iteration 1: $coef_{Lmin} = 0.25$, $coef_{Lmax} = 1.0$, $LBbal = 0.34$). Score = 5 819 200

F. Genetic Algorithm

Since many hyperparameters are involved in the GA deployment, it was given priority to the study of crossover and mutation rates hyperparameters, in order to understand how the conjugation between both may influence the performance of the solution. More particularly, how an increment in the solutions' diversity (increasing crossover and mutation rates), may result in better problem's solutions.

According to all the knowledge collected along the performed projected, it is expected that increasing the solution's diversity, by increasing both crossover and mutation rates, better and faster solutions will be achieved, with an asymptotic behaviour being evidenced.

The set of hyperparameters presented in TABLE VII. were used as basis for the GAs' computations and were obtained by primarily developed simulations. The obtained results are presented below. Note that, due the stochastic nature of the GA, future results may differ from the ones presented.

TABLE VII. GENETIC ALGORITHM HYPERPARAMETERS

Parameter	Value
nPopulation	100
nGenerations	100
selectionType	tournament
selectionRate	0.5
nParti	20
crossoverRate	[0.2, 0.5, 0.9]
mutationRate	[0.1, 0.4, 0.8]

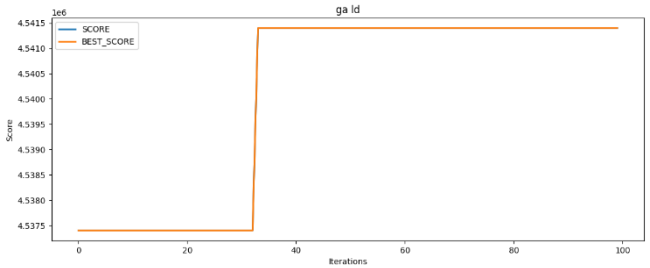

Fig. 14 GA implementation in a "low diversity" scenario, with a crossover rate of 0.2 and a mutation rate of 0.5.

From the analysis of the Fig. 14 to 16, it is possible to conclude that the implementation from the Fig. 14 was only able to generate solutions with one improved score of 4.54×10^6. On the other hand, the implementations from the figures 15 and 16, were able to consecutively generate solutions with improved scores, which the ultimate improved scores were 4.77×10^6 and 4.76×10^6, respectively. In both figures an asymptotic behaviour is also observed. Additionaly, it is also possible to conclude that an increment in the crossover and mutation operations, did not directly resulted in a solution score improvement, although it will for sure increase the propability of better solution score may be found.

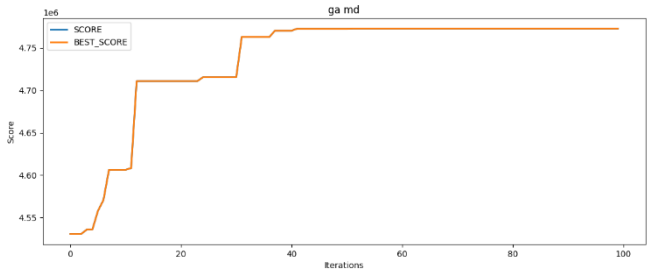

Fig. 15 GA implementation in a "medium diversity" scenario, with a crossover rate of 0.5 and a mutation rate of 0.4.

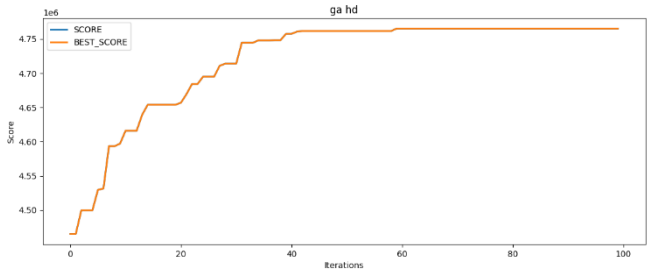

Fig. 16 GA implementation in a "high diversity" scenario, with a crossover rate of 0.9 and a mutation rate of 0.8.

Based on that, an additional implementation was performed for an extended scenario with 300 generations, crossover rate of 0.5, and mutation rate of 0.4. The results are presented on the Fig. 17. With an additional increment in the number of generations was therefore possible to obtain an ultimate maximum solution score of 4.93×10^6 for the GA implementation. Note that (unfortunatelly), it was not possible to run the GA for a more extended implementation (20k, for instance), since computational memory issues related with the Deepnote platform surged.

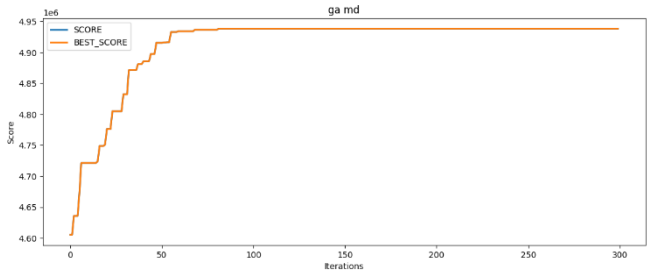

Fig. 17 GA implementation in a "medium diversity" scenario, with a crossover rate of 0.5 and a mutation rate of 0.4, with an extented number of generations of 300.

G. Comparative analysis

TABLE VIII. shows the indicators for the comparative performance analysis of the three algorithms, namely: the number of solutions tested, the number of iterations necessary to reach the best score, the time used in the computation and the best score obtained with that algorithm with the optimized hyper parameters.

The results below presented were obtained in conditions as similar as possible, i.e., same computation power, same number of iterations and same initial random solution (except for the GA).

TABLE VIII. PERFORMANCE COMPARISON

Alg.	Sol. tested	Iter. to optimum	Time [s]	Best score
SA	20k	2 431	368	5 318 300
TS	20k	19 483	431	5 819 200
GA	300	82	6 747	4 938 400

VIII. CONCLUSIONS

One of the most striking findings was how much time it is necessary to search the optimum solution by starting with a random solution. This means the Google Hash Code challenge must be started with some informed solution, such as those proposed in section VII. It was possible to prove that for the case of file b the algorithms converge to the deterministic solution. Even when started with the "score"/"sign up" as the initial solution, the total score would not increase. It is thus safe to say that file b leads to a deterministic solution. Though not presented here the other files also have similar behaviour'. The exception being for files e and d that, when initialized with the best deterministic solution it was possible to improve the total score, as is demonstrated in Fig. 18 and Fig. 19.

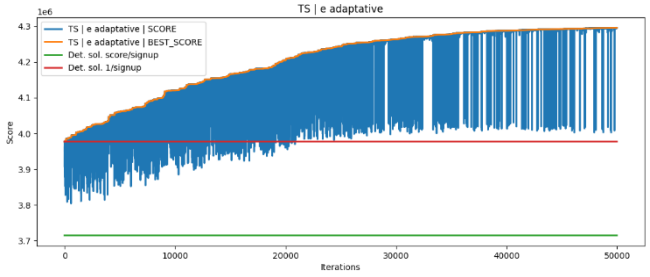

Fig. 18 TS score convergence for the problem of file e, when initialized with the deterministic solution of ordering the libraries by the 1/"sign up period" (hyper-parameters at iteration 1: $coef_{Lmin} = 0.25$, $coef_{Lmax} = 1.0$, $LBbal = 0.34$). Score = 4 295 559

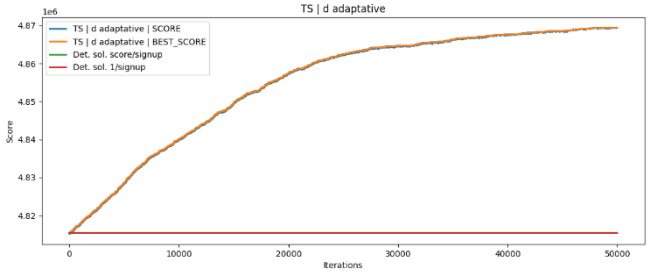

Fig. 19 TS score convergence for the problem of file d, when initialized with the deterministic solution of ordering the libraries by the "score"/"sign up period" ($coef_{Lmin} = 0.05$, $coef_{Lmax} = 1.0, LBbal = 0.83$) Score = 4 869 410

For reasons of simplicity, the list of books of each library was ordered by decreasing value of book score, which significantly limits the number of solutions we can explore. It is possible that considering a random order of books might lead to improvements in the total score of the solutions. Another limitation resulted from the time available to conclude this work. The definition of the nodes of the solution to be pairs of ("library", "initial book") meant the implementation of the algorithms was rather unusual, so they had to be programmed from scratch. No library of optimization algorithms was used in the implementation of the solutions presented in this article.

REFERENCES

[1] Google Hash Code https://codingcompetitions.withgoogle.com/hashcode/ (visited on the 2022/03/19)

[2] Google Hash Code, Archive, https://codingcompetitions.withgoogle.com/hashcode/archive (visited on the 2022/03/19)

[3] Google books https://books.google.com/ (visited on the 2022/03/19)

[4] Google Hash Code, Archive 2020, Google Books https://storage.googleapis.com/coding-competitions.appspot.com/HC/2020/hashcode_2020_online_qualification_round.pdf (visited on the 2022/03/19)

[5] Kirkpatrick, S.; Gelatt Jr, C. D.; Vecchi, M. P. (1983). "Optimization by Simulated Annealing". Science.

[6] Fred Glover (1986). "Future Paths for Integer Programming and Links to Artificial Intelligence". Computers and Operations Research. 13 (5): 533–549. doi:10.1016/0305-0548(86)90048-1.

[7] Fred Glover (1989). "Tabu Search – Part 1". ORSA Journal on Computing. 1 (2): 190–206. doi:10.1287/ijoc.1.3.190.

[8] Fred Glover (1990). "Tabu Search – Part 2". ORSA Journal on Computing. 2 (1): 4–32. doi:10.1287/ijoc.2.1.4.

[9] https://en.wikipedia.org/wiki/Tabu_search (visited on the 2022/03/19)

[10] https://en.wikipedia.org/wiki/Genetic_algorithm (visited on the 2022/03/20)

[11] Russel, S. (2010). *Artificial Intelligence A Modern Approach.* Pearson Education.

[12] Judson, R. S. (2008). *Genetic Algorithms.* Springer.

[13] https://gistbok.ucgis.org/bok-topics/genetic-algorithms-and-artificial-genomes (visited on the 2022/03/21)

[14] https://deepnote.com (visited on the 2022/03/19)

[15] https://machinelearningmastery.com/simulated-annealing-from-scratch-in-python

[16] Edgar Sanchez, https://github.com/edgarsmdn/TS (visited on the 2022/03/19)

[17] https://python.algorithms-library.com/searches/tabu_search

[18] https://pygad.readthedocs.io/en/latest/README_pygad_ReadTheDocs.html#life-cycle-of-pygad (visited on the 2022/03/17)

[19] https://machinelearningmastery.com/simple-genetic-algorithm-from-scratch-in-python (visited on the 2022/03/17)

[20] https://towardsdatascience.com/google-hash-code-2020-a-greedy-approach-2dd4587b6033 (visited on the 2022/03/24)

[21] https://betterprogramming.pub/google-hash-code-2020-how-we-took-98-5-of-the-best-score-e5b6fa4abc1b (visited on the 2022/03/24)

[22] https://github.com/Thanasis1101/Hashcode-2020-solution-Qualification-round (visited on the 2022/03/24)

[23] https://www.linkedin.com/pulse/google-hash-code-2020-discussing-my-solution-scored-223-prateek/ (visited on the 2022/03/24)