March 1, 2024

Optimization Problem: Book Scanning

Artificial Intelligence
MSc Data Science and Engineering
Teacher Luís Paulo Reis

By Igor Diniz, Ingrid Diniz and Paula Ito

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Contents

Introduction to the Book Scanning Problem

Online Qualification Round of Hash Code 2020

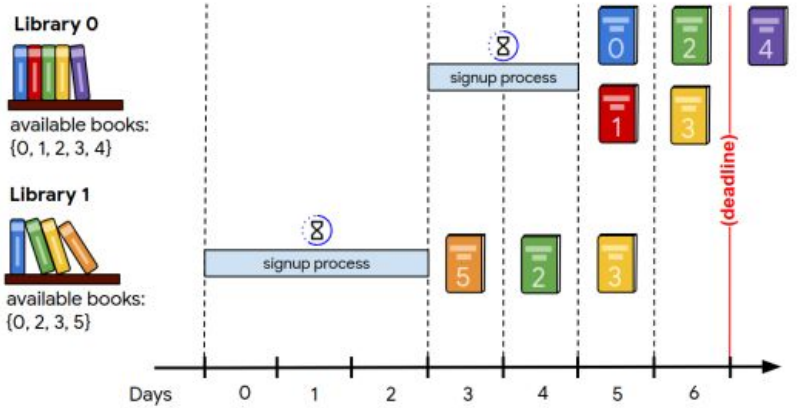
Goal: Maximize the total score of all scanned books in a given period of time (D days).

Book

score

Library

books
sign up days
check out no. per day



Problem Formulation

Solution: composed by list of libraries and list of books.

Evaluation/fitness function: sum of solution's book scores.

```
class Solution:
    def __init__(self, libraries: list = [], books: list = []):
        self.libraries = libraries
        self.books = books

    def evaluate(self):
        return sum(map(lambda book: book.score, self.books))
```

Constraints:

- Only **one library can undergo sign up** at a time.
- Books from a library can be scanned only after the sign up process is complete.
- Each library has a **daily limit** on the number of books that can be scanned.
- Books must be shipped, scanned, and returned within the specified time frame.
- Each book is only scanned once.

Input Data

The original data consists of 5 files that have the following information:

- **Number of Books (B)**
- **Number of Libraries (L)**
- **Number of Days (D)**
- **Book Scores (S)**
- **Library Information:**
  - **Number of Books in the Library (N)**
  - **Sign Up Time (T)**
  - **Scanning Rate (R)**
  - **Books in the Library:** A list of book IDs

File Format

Input file	Description
`6 2 7`	There are 6 books, 2 libraries, and 7 days for scanning.
`1 2 3 6 5 4`	The scores of the books are 1, 2, 3, 6, 5, 4 (in order).
`5 2 2`	Library 0 has 5 books, the signup process takes 2 days, and the library can ship 2 books per day.
`0 1 2 3 4`	The books in library 0 are: book 0, book 1, book 2, book 3, and book 4.
`4 3 1`	Library 1 has 4 books, the signup process takes 3 days, and the library can ship 1 book per day.
`3 2 5 0`	The books in library 1 are: book 3, book 2, book 5 and book 0.

Note:

For this first demonstration of results we created 4 files with the same format.

Neighbour, Mutate and Cross-over

General Approach: Change solution's list of libraries, *then* get best books for each library.

Neighbour/Mutate Functions:

- "Internal" neighbours: swap and deletion of libraries.
- "External" neighbours: addition of new library.

Cross-over Functions: Mid and random point.

Implementation Details

- **Object-oriented:** books, libraries, solution and solver.


```
optimization_book_scanning
├── classes
│   ├── book.py
│   └── library.py
├── data
│   ├── a_example.in
│   ├── b_read_on.in
│   ├── c_incunabula.in
│   ├── d_tough_choices.in
│   ├── e_so_many_books.in
│   └── f_libraries_of_the_world
├── helpers
│   ├── file_reader.py
│   ├── solution.py
│   ├── helpers.py
│   └── app_utils.py
├── metaheuristics
│   ├── solver.py
│   ├── genetic_algorithm.py
│   ├── tabu_search.py
│   ├── hill_climbing.py
│   └── simulated_annealing.py
├── app.py
└── README.md
```

Optimization Algorithms

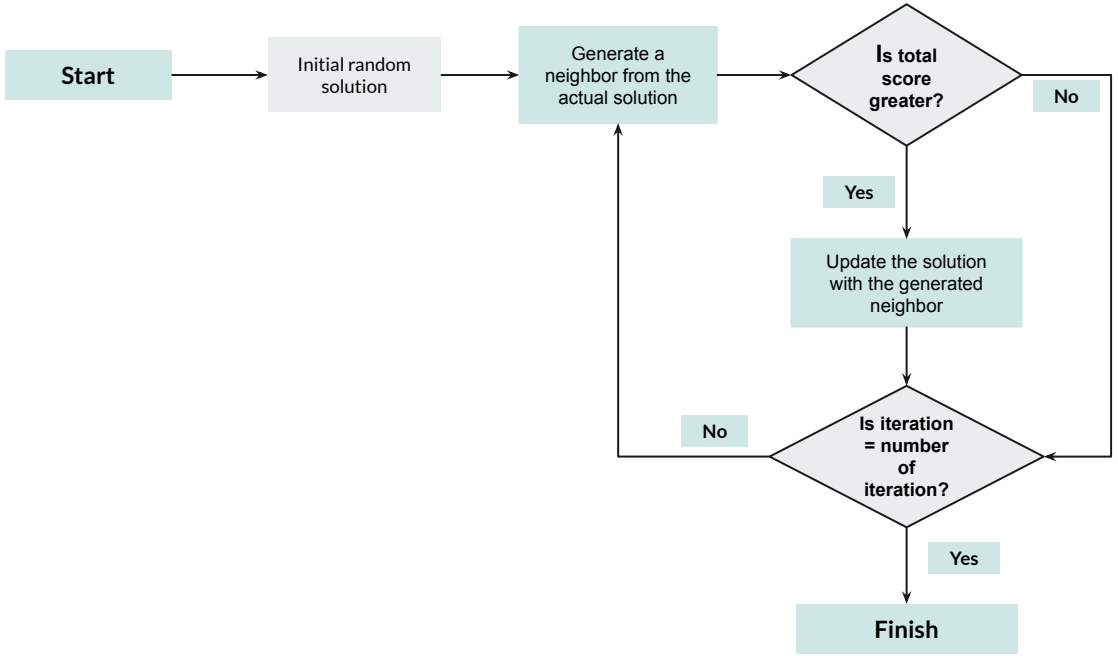
Hill Climbing Algorithm

Local Search Algorithm

Do not accept worse solutions

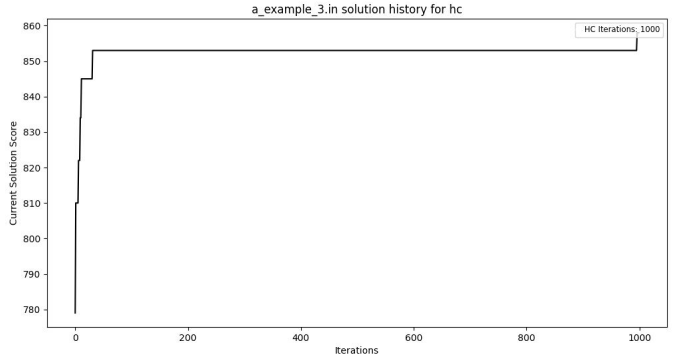
Parameters:

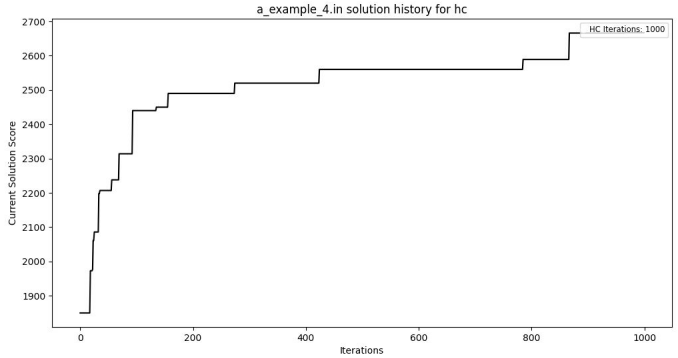
- Initial solution
- Number of iteration



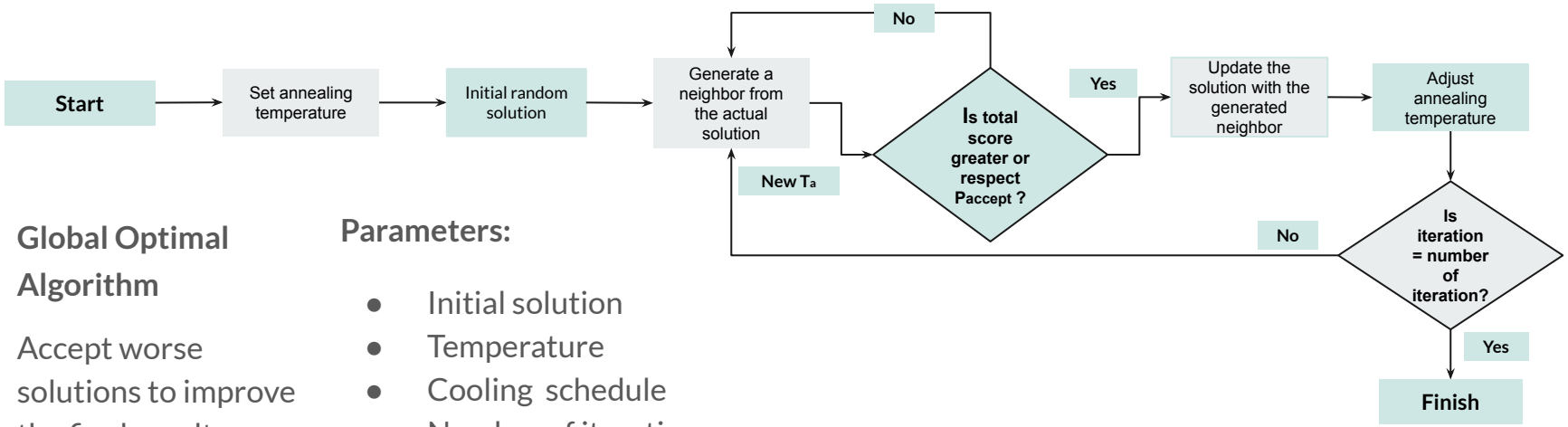
Hill Climbing Algorithm

file	max iterations	best score	time (s)	memory (kB)
a_example.in	1000	21	0.0056	111
a_example_2.in	1000	173	1.54	168
a_example_3.in	1000	205	3.67	160
a_example_4.in	1000	1663	26.06	1057





Simulated Annealing Algorithm



Global Optimal Algorithm

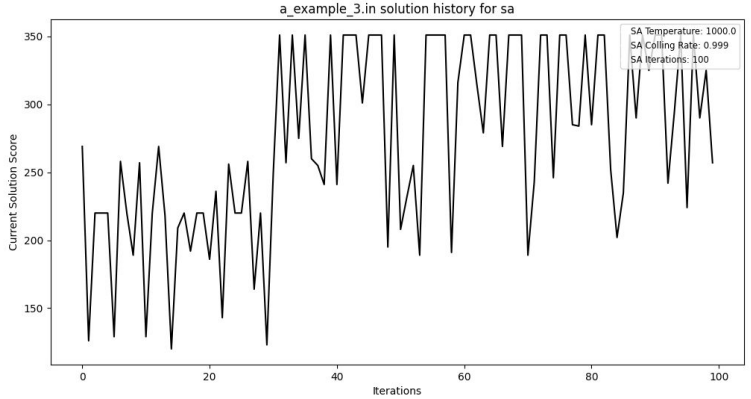
Accept worse solutions to improve the final result

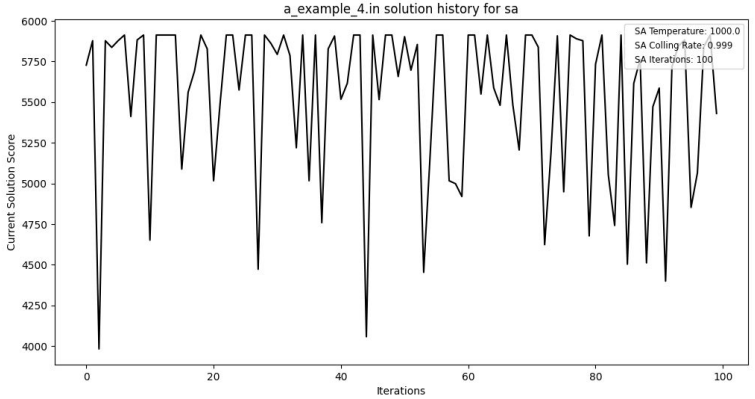
Parameters:

- Initial solution
- Temperature
- Cooling schedule
- Number of iteration

Simulated Annealing

file	temperature	cooling rate	best score	time (s)	memory (kB)
a_example.in	1000	0.999	21	0.002	70
a_example_2.in	1000	0.999	248	2.02	180
a_example_3.in	1000	0.999	536	4.83	237
a_example_4.in	1000	0.999	2676	30.36	1299





Tabu Search

Tabu Tenure: number of iterations in tabu list.

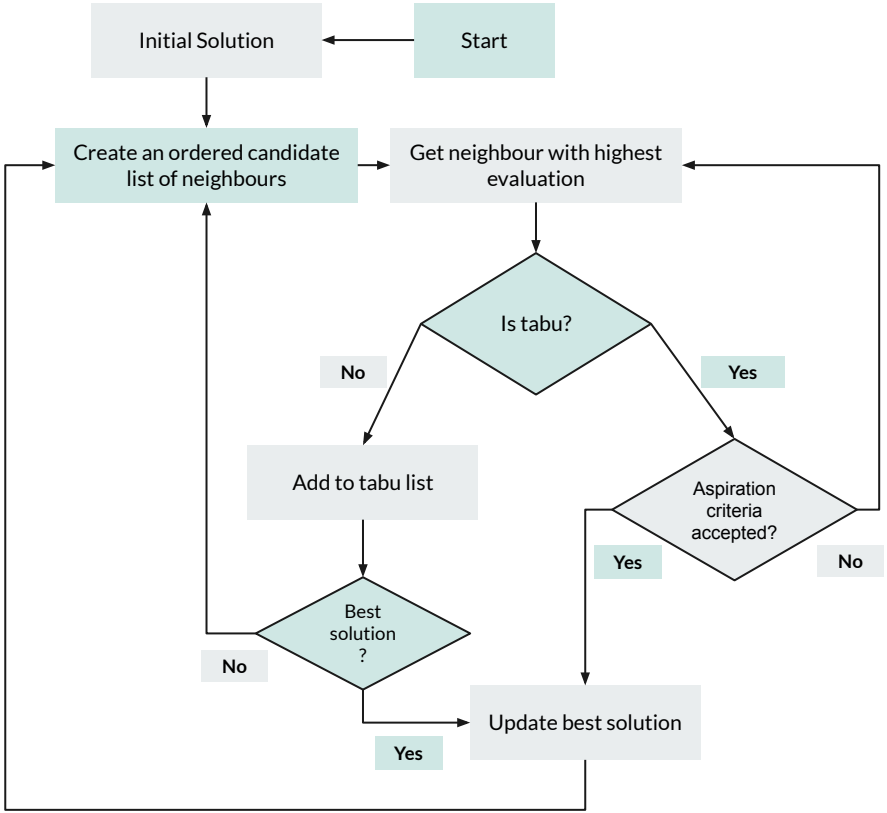
Candidate List: number of neighbours to be generated.

Aspiration Criterion: if current solution is better than the best one known.

Tabu Attributes: Swaps between libraries.

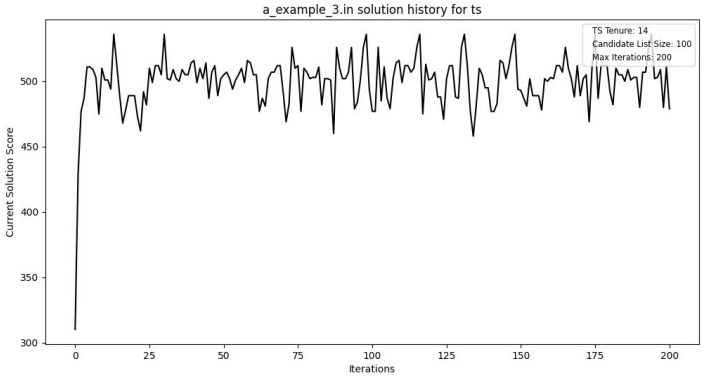
Parameters:

- Initial Solution
- Tabu Tenure
- Number of Neighbours (Candidate List Size)
- Maximum Number of Iterations



Tabu Search

file	tabu tenure	candidate list size	best score	time (s)	memory (kB)
a_example.in	11	all	21	0.024	25
a_example_2.in	11	50	248	4.22	226
a_example_3.in	11	100	536	66.04	1831
a_example_4.in	11	30	2188	227.16	4137



Genetic Algorithm

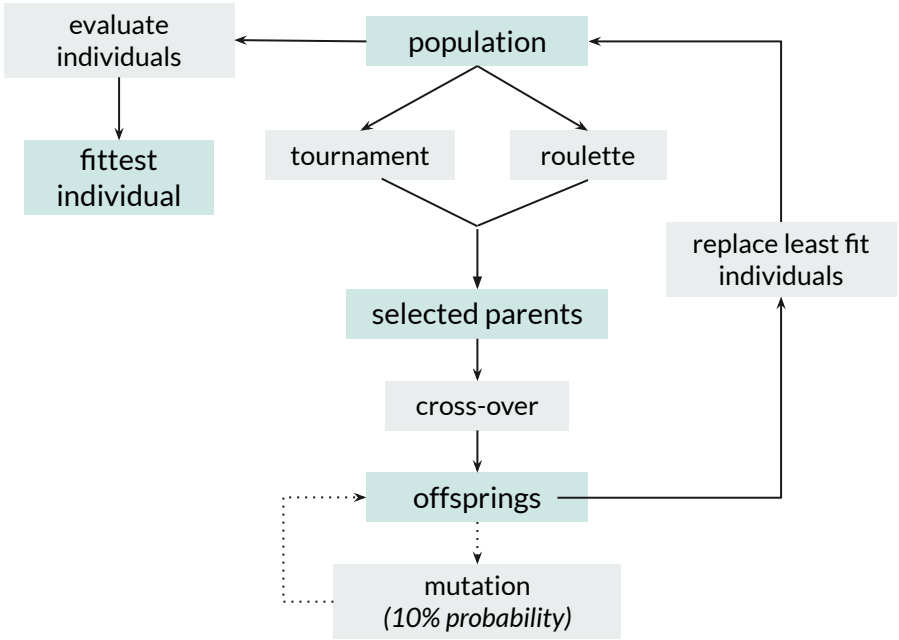
Population: set of individuals (i.e. solutions).

Steady-state: least fit individuals replace each generation.

Parameters:

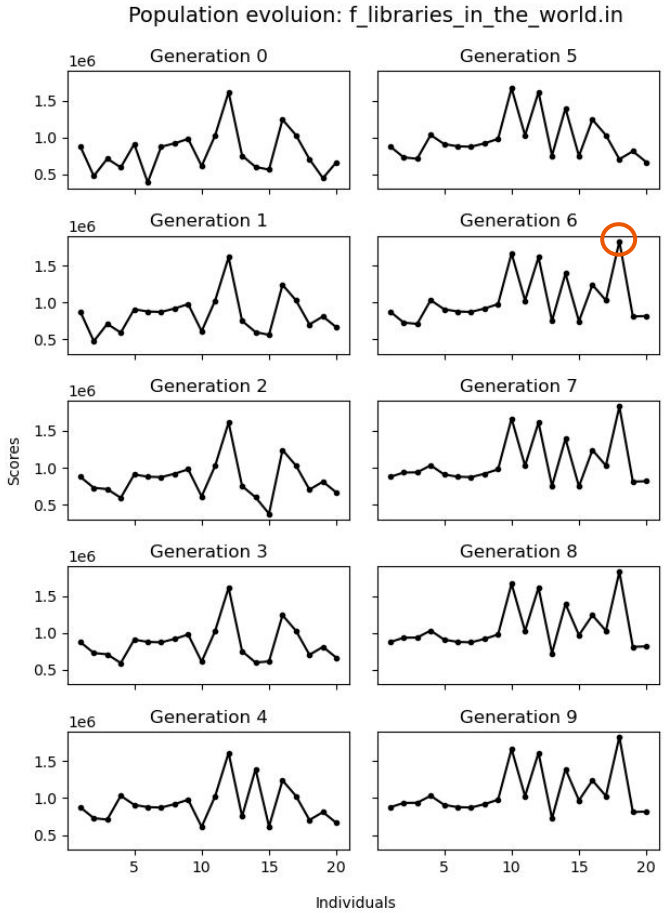
- Population size → *20*
- No. of generations → *10*
- Mutation mode:
  - *swap*, *deletion* or *addition*
- Crossover mode
  - *mid* or *random* point

} *grid search*



Genetic Algorithm

file	mutation mode	crossover mode	best score	time (min)
a_example.in	swap	random	21	0.03
a_example_2.in	swap	random	206	0.36
a_example_3.in	addition	mid	594	13.49
a_example_4.in	deletion	random	3181	9.08
b_read_on	swap	random	4770900	7.17
c_incunabula	addition	mid	1014099	1.19
d_tough_choices	addition	random	4362930	17.99
e_so_many_books	deletion	random	1258925	2.66
f_libraries_of_the_world	deletion	mid	1831208	1.34



Comparing Results

Parameters

Initial Solution: random (HC, SA, TS)
Max. iterations: 100 (HC, SA, TS)

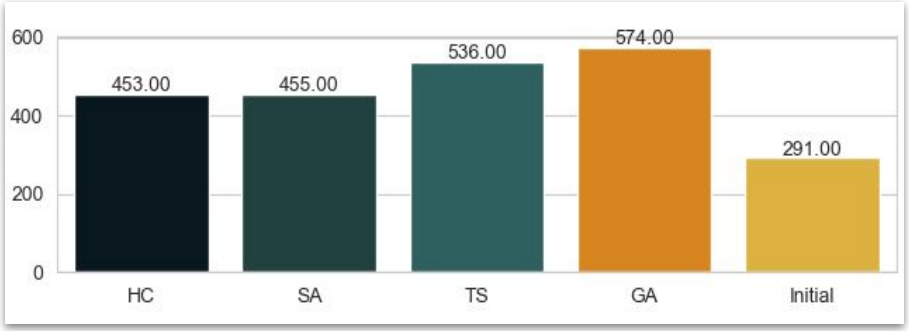
Simulated Annealing:
- Temperature: 1000
- Cooling schedule: 0.99

Tabu Search:
- Tabu tenure: 15
- Num. neighbours: 100

Genetic Algorithm:
- Population size: 100
- Num. generations: 20
- Mutate: addition
- Crossover: mid point

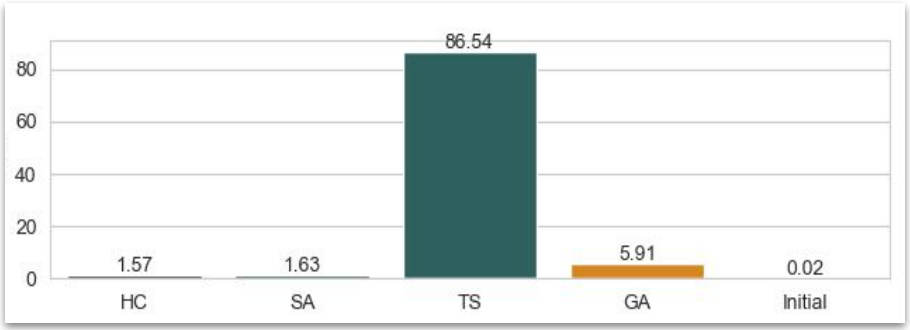
Score Analysis

Comparison for different algorithms



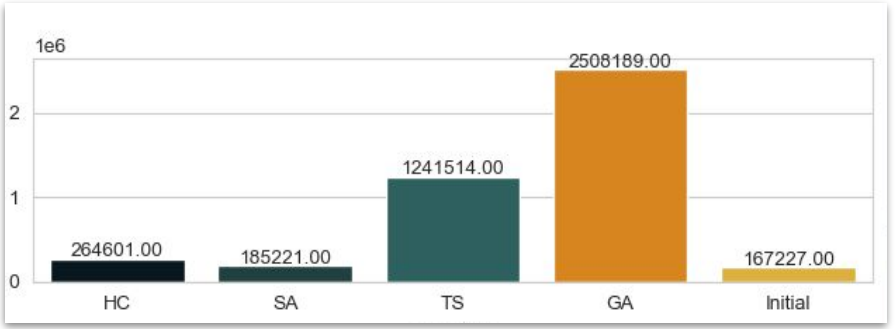
Time Analysis

Comparison for different algorithms



Memory Usage Analysis

Comparison for different algorithms



Conclusion

Problem complexity → Algorithm Choice!

Hill Climbing and Simulated Annealing:

- Work well on small, simple datasets
- Less memory and time to process

Tabu Search and Genetic Algorithm:

- Solve big, complex datasets (better scores)
- More memory and time to process

DEMO TIME!

Future Works and Improvements for Next Week

1. Run the code for the real datasets provided by Hash Code
2. Find the best parameters for each algorithm with the real datasets
3. Add a time limit parameter in the implementations
4. Add penalty using the Long Term Frequency Based Memory for Tabu Search
5. Generate initial population using greedy solution for Genetic Algorithm

References

Soares, A., & Costa, J. P. (2013). Principled Modeling of the Google Hash Code Problems for Meta-Heuristics [University of Coimbra]. Retrieved from https://estudogeral.uc.pt/retrieve/265403/tese_final_pedro_rodrigues.pdf.

John, K. (2020, October 30). How we achieved 98.27% of the best score at Google HashCode 2020. [Medium]. Retrieved from https://medium.com/@kevinjohn1999/how-we-achieved-a-98-27-of-the-best-score-at-google-hashcode-2020-94314a904190.

March 1, 2024

Thank You!

Artificial Intelligence | MSc Data Science and Engineering
Teacher Luís Paulo Reis
By Igor Diniz, Ingrid Diniz and Paula Ito

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO