Delivery Scheduling

Metaheuristics for Optimization/Decision Problems

- Delivery system composed of **one truck** with **constant speed**.
- For a given set of packages, each with their own delivery coordinates, design algorithms to **optimize** the **delivery order of packages**.
- Packages may be of different types: **normal**, **fragile** or **urgent**.
- Must **minimize travelling costs**.
- Must maximize reputation, by **minimizing** both **damage to fragile packages** and **delayed deliveries of urgent packages**.

Class 9 Group 2 João Ramos Marco Costa Tiago Viana

Problem Formulation

$\textbf{PackageSet} = \{p_0, \ldots, p_{n-1}\}$, where $|\textbf{PackageSet}| = n$

$\textbf{Solution} = (s_0, \ldots, s_{n-1}) \in \textbf{PackageSet}^n$, where $\forall_{i \neq j}\ (0 \leq i < j < n),\ s_i \neq s_j$, and $|\textbf{Solution}| = n$

$neighbour(\mathbf{S}) \rightarrow$ For a given $s_i, s_j \in \mathbf{S}$, swap s_i and s_j

$mutation(\mathbf{S}) \rightarrow$ For random pair $s_{2i}, s_{2i+1} \in \mathbf{S}$, swap s_{2i} and s_{2i+1} with probability P_{swap}

$crossover(\mathbf{S_1}, \mathbf{S_2}) \rightarrow$ Iterate through $\mathbf{S_1}$ and $\mathbf{S_2}$, picking a package from either one with equal probability
Add it to the child if not already present

Hard Constraints

- Delivery truck **starts at origin**.
- Only **one location** can be visited at a time.
- **Routes** between **all** delivery locations are **available**.
- The driver drives at **60km per hour** and takes **0 seconds** to deliver the goods.
  - Both these constants are **parameters** of the problem and can be changed for each instance.

Problem Formulation

Evaluation Function

For a given **Solution**, $TotalCost = w_{TravellingCost} \cdot TravellingCost + w_{DamageCost} \cdot DamageCost + w_{DelayCost} \cdot DelayCost$

$$TravellingCost = C_{km} \cdot \sum_{i=0}^{n-2} d(s_i, s_{i+1}), \text{ where: } s_i, s_{i+1} \in \textbf{Solution}$$

C_{km} is the travelling cost per km

$d(s_i, s_j)$ is the distance between delivery locations of packages s_i and s_j

$$DamageCost = \sum_{i=0}^{n-1} d_i \cdot Z_i, \quad \text{where: } Z_i \text{ is the cost of damaging package } s_i, \begin{cases} Z_i > 0, & \text{if } s_i \text{ is fragile} \\ Z_i = 0, & \text{otherwise} \end{cases}$$

$$\begin{cases} d_i = 1, & \text{with probability } P_{damage} \\ d_i = 0, & \text{otherwise} \end{cases}$$

$$P_{damage} = 1 - (1 - X)^{d_{s_i}}$$

d_{s_i} is the distance travelled in kms by package s_i

X is the probability of a fragile package being damaged per each km travelled

$$DelayCost = C_{delay} \cdot \sum_{i=0}^{n-1} delay_{s_i} \cdot u_i, \quad \text{where: } s_i \in \textbf{Solution}, \begin{cases} u_i = 1, & \text{if } s_i \text{ is urgent} \\ u_i = 0, & \text{otherwise} \end{cases}$$

C_{delay} is the cost per minute of delay

$delay_{s_i}$ is the delay of package s_i in minutes

Implementation Details

- **Programming Language -** Python.
- **Development Environment -** Python scripts in VSCode.
- **Core Data Structures**
  - **Package -** Represents a package to be delivered. Stores the coordinates of the package's delivery location and the package type. For fragile packages, the breaking change and breaking cost are set. For urgent packages, a maximum delivery time is set.
  - **Delivery Schedule -** Represents an instance of the problem to be solved. Stores the set of packages to be delivered and the constants related to the evaluation function.
- **Libraries -** NumPy, MatplotLib.

Neighbour Operators

- **First Operator -** Swaps two consecutive deliveries.
- **Second Operator -** Swaps two random deliveries.
- **Third Operator -** Randomly picks one of the previous operators with equal probability.

Crossover and Mutation Operators

- **Crossover Operator -** Iteratively selects genes from one of the parents with equal probability.
- **Mutation Operator -** Swaps a random pair of contiguous genes with a certain probability.

Optimization Algorithms

Hill Climbing

- ❑ Acceptance Criterion
  - ❑ First Accept
  - ❑ Best Accept
- ❑ Neighbour Operator

Simulated Annealing

- ❑ Initial Temperature
- ❑ Cooling Rate
- ❑ Fixed Temperature Iterations
- ❑ Neighbour Operator

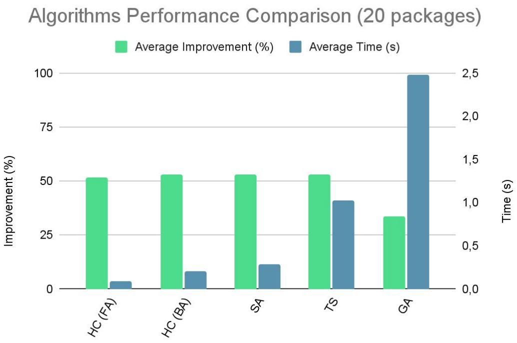
Tabu Search

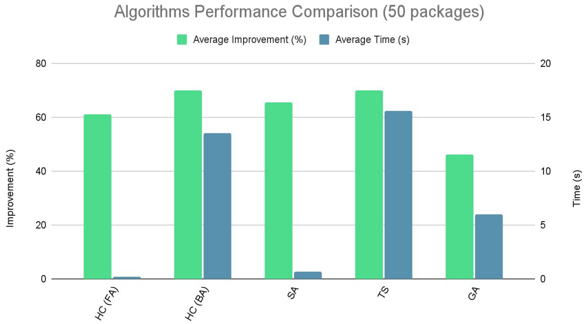
- ❑ Initial Tenure
- ❑ Tenure Mode
  - ❑ Constant
  - ❑ Random
  - ❑ Iteration-based
  - ❑ Dynamic
- ❑ Tenure Parameter
- ❑ Tabu Criteria
  - ❑ Solution Repetition
  - ❑ Move Repetition
  - ❑ Package Repetition
- ❑ Shuffle Probability
- ❑ Neighbour Operator

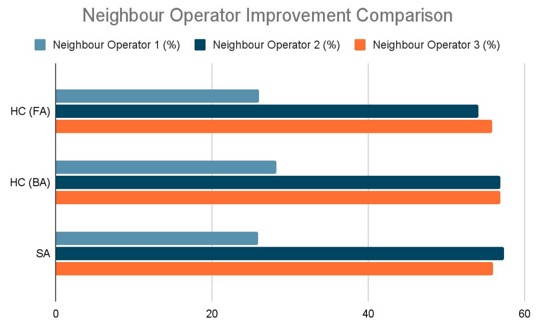
Genetic Algorithm

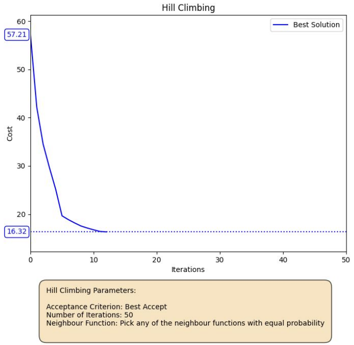
- ❑ Tournament Size
- ❑ Population Size
- ❑ Mutation Probability
- ❑ Number of Offsprings
- ❑ Evolution Strategy
  - ❑ Replacement
  - ❑ Growth
- ❑ Replacement Strategy
  - ❑ Random
  - ❑ Fitness-based

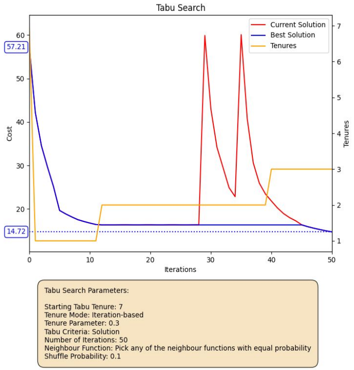
Experimental Results

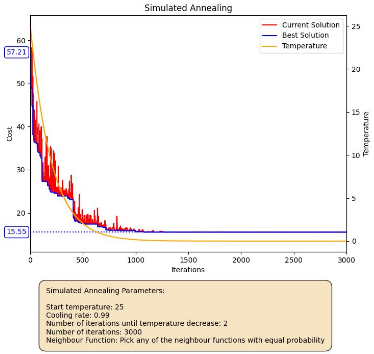


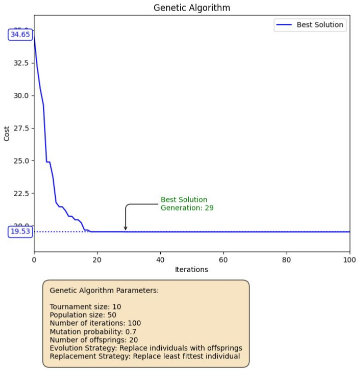


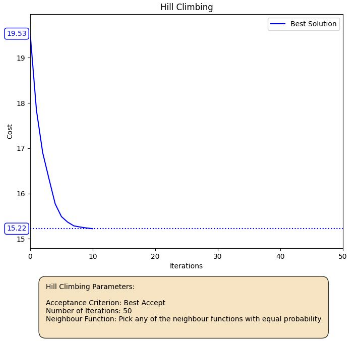




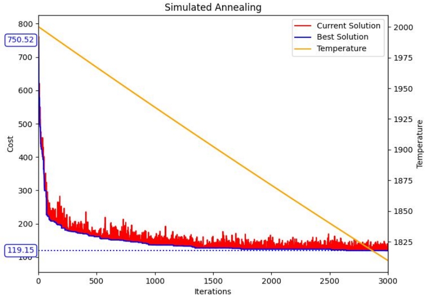






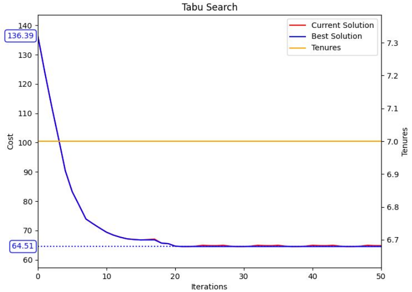


Experimental Results



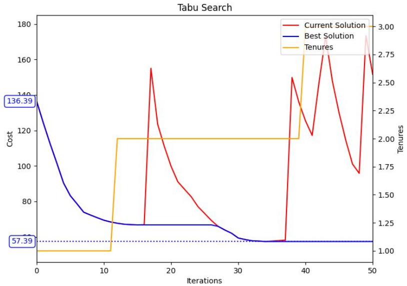
Simulated Annealing Parameters:

Start temperature: 2000
Cooling rate: 0.9999
Number of iterations until temperature decrease: 3
Number of iterations: 3000
Neighbour Function: Pick any of the neighbour functions with equal probability



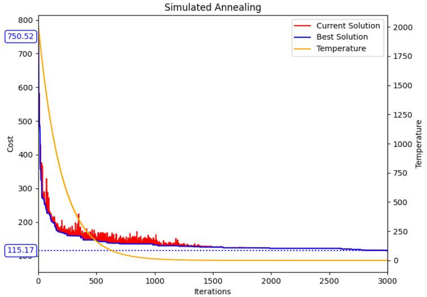
Tabu Search Parameters:

Starting Tabu Tenure: 7
Tenure Mode: Constant
Tabu Criteria: Solution
Number of Iterations: 50
Neighbour Function: Pick any of the neighbour functions with equal probability
Shuffle Probability: 0.0



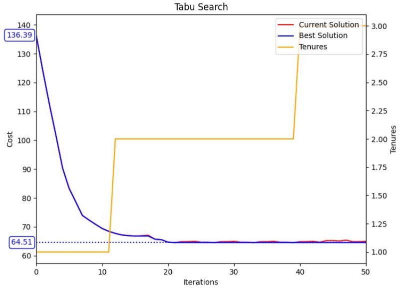
Tabu Search Parameters:

Starting Tabu Tenure: 1
Tenure Mode: Iteration-based
Tenure Parameter: 0.3
Tabu Criteria: Package
Number of Iterations: 50
Neighbour Function: Pick any of the neighbour functions with equal probability
Shuffle Probability: 0.1



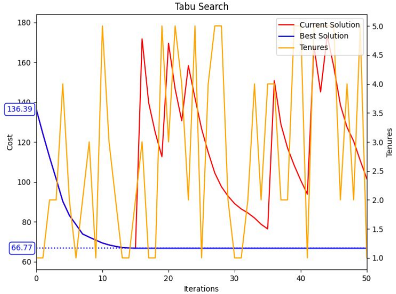
Simulated Annealing Parameters:

Start temperature: 2000
Cooling rate: 0.99
Number of iterations until temperature decrease: 2
Number of iterations: 3000
Neighbour Function: Pick any of the neighbour functions with equal probability



Tabu Search Parameters:

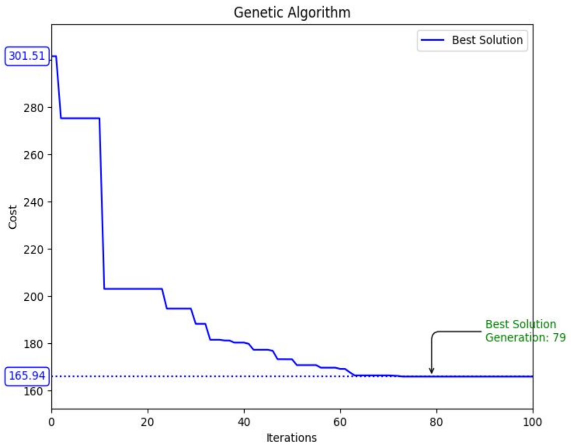
Starting Tabu Tenure: 1
Tenure Mode: Iteration-based
Tenure Parameter: 0.3
Tabu Criteria: Package
Number of Iterations: 50
Neighbour Function: Pick any of the neighbour functions with equal probability
Shuffle Probability: 0.0



Tabu Search Parameters:

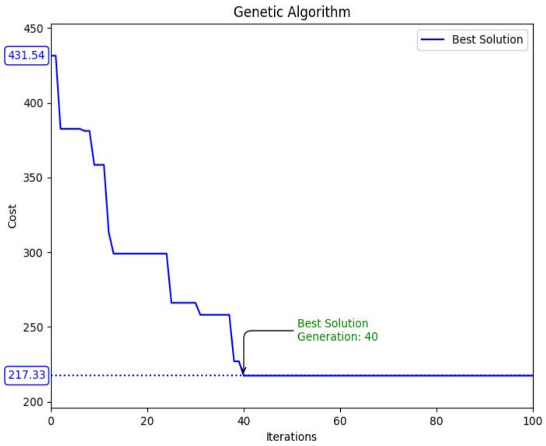
Starting Tabu Tenure: 1
Tenure Mode: Random
Tabu Criteria: Package
Number of Iterations: 50
Neighbour Function: Pick any of the neighbour functions with equal probability
Shuffle Probability: 0.1

Experimental Results



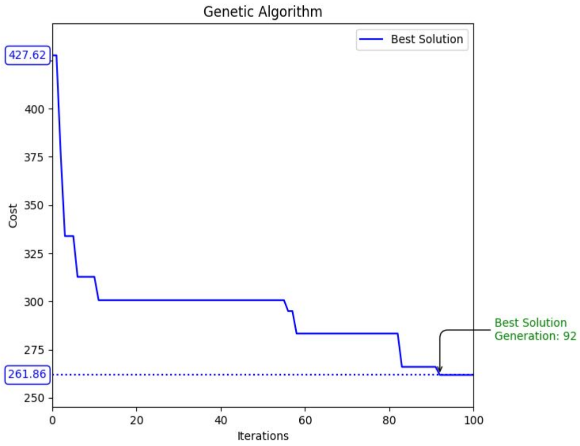
Genetic Algorithm Parameters:

Tournament size: 10
Population size: 50
Number of iterations: 100
Mutation probability: 0.5
Number of offsprings: 10
Evolution Strategy: Replace individuals with offsprings
Replacement Strategy: Replace least fittest individual



Genetic Algorithm Parameters:

Tournament size: 10
Population size: 50
Number of iterations: 100
Mutation probability: 0.5
Number of offsprings: 10
Evolution Strategy: Replace individuals with offsprings
Replacement Strategy: Replace random individual



Genetic Algorithm Parameters:

Tournament size: 10
Population size: 50
Number of iterations: 100
Mutation probability: 0.5
Number of offsprings: 10
Evolution Strategy: Add offsprings to the population
Replacement Strategy: Replace least fittest individual

Conclusions

- **Metaheuristics** are a powerful tool to seek near-optimal solutions to **optimization/decision problems**. However, they require a large amount of **configuration** and **fine tuning** in order to achieve good results.

- This project allowed us to deepen our knowledge of each of the metaheuristics implemented, granting us awareness of the pros and cons of each of them:
  - **Hill Climbing First-Accept** and **Best-Accept** proved to be simple approaches to optimization problems, but are highly susceptible to getting trapped in local optima. The First-Accept approach is more computationally efficient. On the other hand, Best-Accept nearly guarantees better solutions are achieved.
  - **Simulated Annealing** often achieves better results than Hill Climbing Best-Accept, showcasing high flexibility in solution space exploration and escaping local optima. Furthermore, it keeps computational demands low and is straightforward to tune according to different problem instances.
  - **Tabu Search** provides excellent results at the cost of high memory and computational power consumption, being recommended only for small to medium-sized problems. Despite its higher execution times, it is capable of achieving better and more novel solutions when compared to the other approaches.
  - **Genetic Algorithms** are very flexible and can be approached in many different ways. The **crossover** and **mutation operators** are key to the improvement of the overall fitness of the population, albeit the chosen evolution strategy is the ultimate determinant of the quality of the solutions that persist throughout the generations. Conversely, despite escaping worse local optima, the latest generations fail to converge to better optima, a problem which can be remedied by subsequently applying Local Search.

- In conclusion, one of the main takeaways of this project was that determining which metaheuristic is better suited for a given optimization/decision problem heavily depends on the problem's constraints, size, and particularities.

References

- Stuart Russell, Peter Norvig - Artificial Intelligence: A modern Approach.
- Delivery Scheduling. Retrieved from https://drive.google.com/file/d/1-A85i8haeQQSYkRILF0uYZOZ0Niba0zJ/view. Accessed March 3, 2024.
- Bidirectional Hash Table. Retrieved from https://stackoverflow.com/questions/3318625/how-to-implement-an-efficient-bidirectional-hash-table. Accessed March 13, 2024.
- Dynamic Tabu Tenure. Retrieved from https://www.researchgate.net/publication/220867608_Adaptive_Tabu_Tenure_Computation_in_Local_Search#full-text. Accessed March 14, 2024.

AI Tools Used

- Github Copilot
  - Help expediting code documentation
  - Help describing the menus in the README