# A hyper-heuristic algorithm for the Traveling Salesman Problem

Oussama Djelloul, Elissa Tagzirt, Chakib Islam Kedadsa, Lina Sadi, Tihami Mohammed Hicham, Cylia Messar, Nada Bousba
Ecole Nationale Supérieure d'Informatique, Algiers, Algeria

June 13, 2024

**Abstract**

The Traveling Salesman Problem (TSP) is a well-known combinatorial optimization challenge that seeks the most cost-efficient route. In this paper, we introduce a hybrid methodology employing a hyper-heuristic algorithm that integrates machine learning with Ant Colony Optimization (ACO) and an additional solution utilizing Thompson Sampling Iterated Local Search. This hybrid approach aims to enhance the efficiency of discovering the optimal route for the salesman and to produce high-quality solutions.

hyper-heuristic — Metaheuristic — Hybridization — Ant Colony Optimization — machine learning — Thompson Sampling Iterated Local Search

## 1 Introduction

### 1.1 The Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is a classic problem in optimization and computational mathematics. It involves a salesman who must visit a list of cities, each exactly once, and return to the starting city. The challenge is to determine the shortest possible route that accomplishes this task. Due to its complexity, the TSP is classified as an NP-hard problem, implying that there is no known efficient way to find the optimal solution for large instances.

### 1.2 A Review of Existing Heuristics for Solving the Traveling Salesman Problem (TSP)

Solving the Traveling Salesman Problem (TSP) requires effective approaches to manage its complexity. Heuristics and metaheuristics stand out for their ability to provide high-quality solutions. Here is an overview of the main heuristics and metaheuristics used:

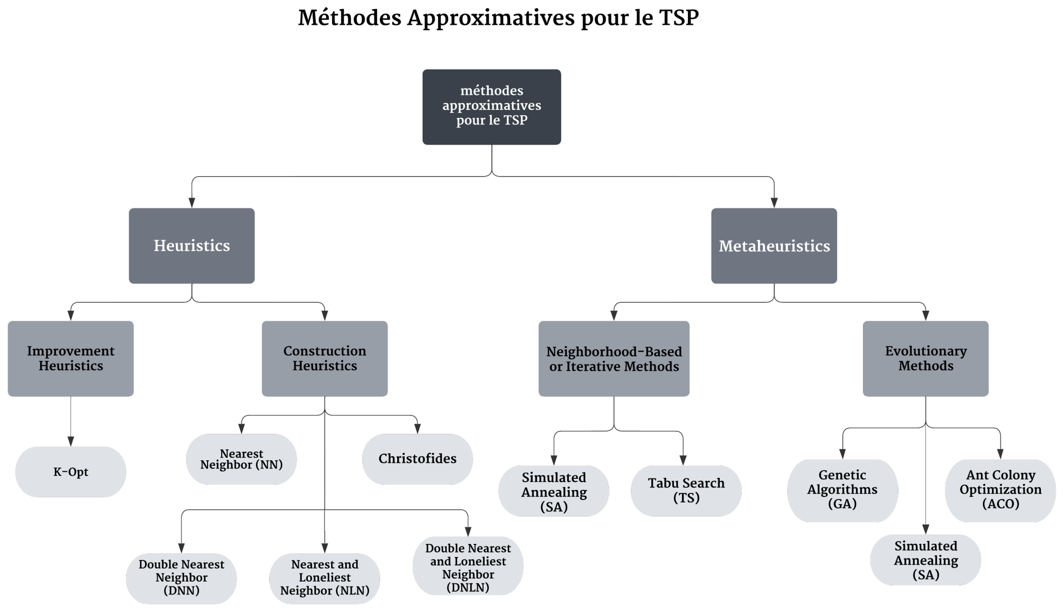

Méthodes Approximatives pour le TSP
méthodes approximatives pour le TSP
Heuristics
Metaheuristics
Improvement Heuristics
Construction Heuristics
Neighborhood-Based or Iterative Methods
Evolutionary Methods
K-Opt
Nearest Neighbor (NN)
Christofides
Double Nearest Neighbor (DNN)
Nearest and Loneliest Neighbor (NLN)
Double Nearest and Loneliest Neighbor (DNLN)
Simulated Annealing (SA)
Tabu Search (TS)
Genetic Algorithms (GA)
Ant Colony Optimization (ACO)
Simulated Annealing (SA)

Figure 1: Méthodes Approximatives pour le TSP

### 1.2.1 Heuristics

Heuristics provide quick and approximate solutions to complex problems and are divided into construction and improvement and here is an overview of the well-known and implemented heuristics.

**Construction Heuristics** These heuristics progressively build a solution for the TSP by adding elements one by one:

- **Nearest Neighbor (NN)**: Starts at an arbitrary city and always visits the nearest unvisited city until all cities are visited.
- **Double Nearest Neighbor (DNN)**: Simultaneously constructs the route from both ends adding the nearest city to each end.
- **Nearest and Loneliest Neighbor (NLN)**: Selects the most isolated city among the closest neighbors favoring less connected cities.
- **Double Nearest and Loneliest Neighbor (DNLN)**: Combines DNN and NLN to choose the most solitary city among the route ends.
- **Christofides**: Uses minimum spanning trees and perfect matchings to guarantee an approximate solution with a ratio of 1.5 of the optimal solution.

**Improvement Heuristics** These heuristics start with an initial solution and apply modifications to improve it:

- **K-Opt**: Exchanges $k$ edges to optimize the route. Common variants include 2-Opt which exchanges two edges to reduce the total tour length and 3-Opt which exchanges three edges for finer optimization.

### 1.2.2 Metaheuristics

Metaheuristics are general optimization methods designed to solve complex problems often too difficult for exact methods. They are divided into two main categories and the following are the most known and implemented during the semester:

**Neighborhood-Based or Iterative Methods** These methods manipulate a single solution and improve it iteratively:

- **Simulated Annealing (SA)**: Inspired by the cooling process of metals, simulated annealing allows temporary moves to less optimal solutions to escape local minima. The probability of accepting a less optimal solution decreases over time simulating a gradual cooling process.
- **Tabu Search (TS)**: Uses a tabu list to avoid revisiting the same solutions, allowing for more diverse exploration of the search space. Forbidden moves are remembered to escape local minima and encourage the search for new solutions.

**Evolutionary Methods** These methods manage a population of solutions and evolve them:

- **Genetic Algorithms (GA)**: Inspired by natural selection, these algorithms use selection, crossover, and mutation operations to evolve towards better solutions. Selection favors the best solutions, crossover combines solutions to create offspring, and mutation introduces diversity.
- **Ant Colony Optimization (ACO)**: Mimics the behavior of ant colonies. Ants deposit pheromones on the paths they take and the probability of another ant taking the same path increases with the concentration of pheromones, thus favoring the shortest and most efficient paths.

## 1.3 Limitations of Heuristics and Metaheuristics and Introduction to Hyper-Heuristics

Although heuristics and metaheuristics are effective in finding approximate solutions to complex problems like the TSP, they have certain limitations:

#### Limitations of Heuristics

- **Problem Specificity**: Heuristics are often designed for specific types of problems making them difficult to generalize to other contexts.
- **Solution Quality**: Heuristics can quickly converge to locally optimal solutions but do not guarantee a globally optimal solution.

#### Limitations of Metaheuristics

- **Parameter Tuning**: Metaheuristics often require fine-tuning of parameters which can be complex and problem-dependent.
- **Computation Time**: While more flexible, metaheuristics can require significant computation time especially for large-scale problems.

To overcome these limitations, researchers have developed hyper-heuristics. These high-level approaches generalize problem-solving strategies and enable the dynamic selection and combination of various low-level heuristics. They reduce dependency on experts, improve solution quality, and optimize computation time. Hyper-heuristics represent a major advancement in combinatorial optimization, addressing the shortcomings of traditional heuristics and metaheuristics.

## 1.4 Motivation for Hyper-Heuristics

**Why Hyper-Heuristics?** Hyper-heuristics have been developed to overcome the limitations of traditional heuristics and metaheuristics. They aim to provide a more general and adaptable method capable of dynamically selecting and combining various low-level heuristics.

**Advantages of Hyper-Heuristics**

- **Generalization**: Designed to be applicable to a variety of problems regardless of their specific characteristics.
- **Flexibility and Adaptability**: Capable of dynamically selecting and combining different heuristics based on the problem's needs.
- **Reduced Dependency on Experts**: Less reliance on experts for parameter tuning and heuristic design.
- **Improved Solution Quality**: Optimizes computation time by choosing the most appropriate heuristics, thus consistently improving solution quality.

## 1.5 Concept and Operation of Hyper-Heuristics

Hyper-heuristics are high-level search methodologies that choose and combine heuristics (or components of heuristics) from the low level. Their primary goal is to automate the process of selecting or generating heuristics to effectively solve complex optimization problems.

## 1.6 Architecture and Operation of Hyper-Heuristics

The architecture of hyper-heuristics is divided into three main parts: the high-level strategy, the low-level heuristics, and the domain barrier.

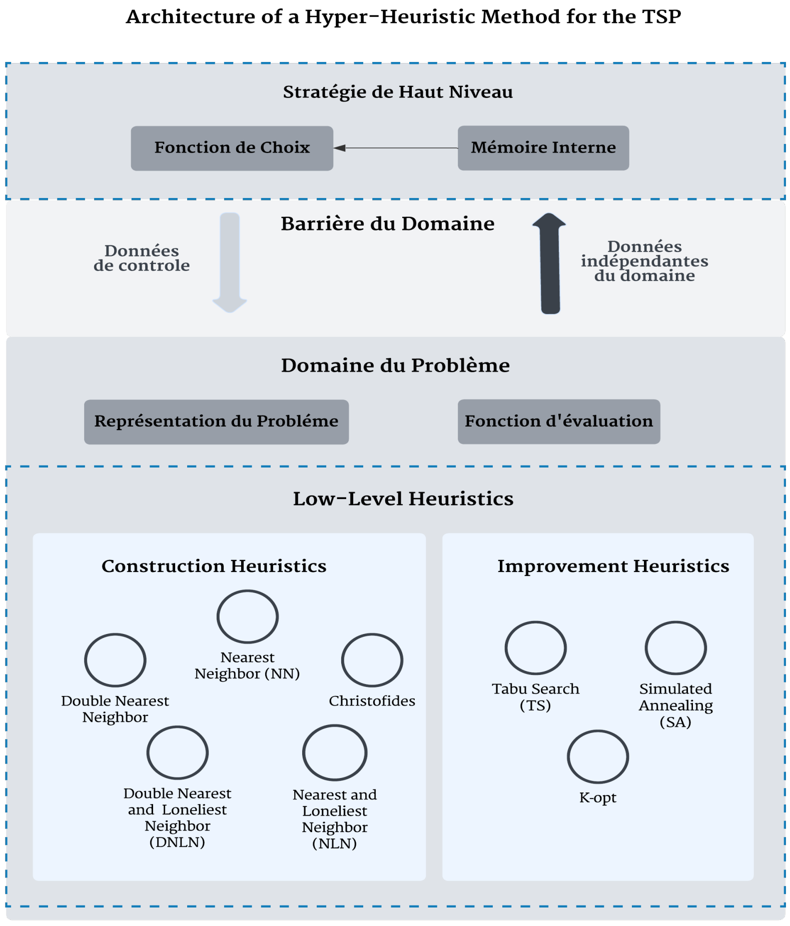


Figure 2: Architecture and Operation of Hyper-Heuristics

## High Level

- **High-Level Strategy**: This involves a choice function that determines which low-level heuristic to apply at each step of the resolution process. This strategy uses internal memory to store information about the past performance of heuristics, including effective combinations and state-heuristic associations. The feedback mechanism uses insights from the search process to inform future decisions, allowing the hyper-heuristic to adapt to the problem dynamics.

## Low Level

- **Low-Level Heuristics**: These are the specific heuristics and heuristic components applied to problem solutions. They include techniques such as Nearest Neighbor (NN), Double Nearest Neighbor (DNN), and 2-Opt.

## Domain Barrier (Filter)

- **Standard Interface**: This interface facilitates the transmission of control data between the high-level strategy and the low-level heuristics. It also ensures the representation of the problem, the evaluation function, and the initial solution.

## 1.7 Different Approaches to Hyper-Heuristics

**Selection-Based Hyper-Heuristics** Select the best low-level heuristic to apply at each step based on fixed rules, learning algorithms, or probabilistic techniques.

**Generation-Based Hyper-Heuristics** Generate new heuristics by combining components of existing heuristics, often using evolutionary techniques or genetic algorithms to explore new combinations.

## 1.8 Applications of Hyper-Heuristics

Hyper-heuristics are used in various fields due to their flexibility and adaptability to different types of problems. Here are some domains where they have been successfully applied:

**Scheduling and Planning** Burke et al. used selection-based hyper-heuristics combining construction and improvement heuristics such as Nearest Neighbor (NN), Double Nearest Neighbor (DNN), and 2-Opt to optimize work schedules and production plans, effectively solving TSP instances in complex production environments.

**Logistics and Transportation** Bai and Kendall applied generation-based hyper-heuristics using genetic algorithms to combine heuristics like Ant Colony Optimization (ACO) and Simulated Annealing (SA). This approach helped solve large-scale TSP instances, optimizing delivery routes and fleet management.

**Bioinformatics** Ochoa et al. used selection-based hyper-heuristics integrating local search heuristics such as Tabu Search (TS) and genetic algorithms (GA) to address TSP-related problems in genetic sequence alignment and protein folding, improving the precision of these complex tasks.

**Network Design** Özcan et al. applied generation-based hyper-heuristics using evolutionary techniques and local search heuristics like Variable Neighborhood Search (VNS) and Greedy Randomized Adaptive Search Procedure (GRASP) to tackle the TSP in the design and optimization of communication networks, reducing costs and increasing reliability.

## 1.9 The proposed approaches

### 1.9.1 Solution based on ACO

The first proposed approach uses an adapted Ant Colony Optimization (ACO) meta-heuristic for solving complex problems efficiently. The originality lies in:

Pheromone Updates: Pheromone levels are updated based on heuristic improvements the heuristics that give the best cost add more pheromones to their path so that we give more advantage to well-performing heuristics.

Visibility Values: Heuristics that require less CPU time are favored, optimizing execution time.

Hybrid Low-Level Heuristics: A combination of constructive and improvement heuristics leverages their strengths to enhance solution quality. This integrated strategy uniquely balances execution time and solution quality, providing robust and effective problem-solving capabilities.

Offline learning: a solution using machine learning to well initialize the parameters of the ACO heuristic.

### 1.9.2 Solution inspired by TS-ILS

One innovative solution implemented is inspired by the study of a hyperheuristic described in a paper published in 2023. This approach exploits the effectiveness history of heuristics used at the basic level, thus adapting heuristic selection to each problem instance in real-time. This method makes the approach intelligent and reactive at critical moments, as it ensures a balance between diversification and intensification. It loads the top level with intensification to improve results, while the bottom level focuses on diversification to explore the search space.

Our originality lies in the addition of a modification to the basic logic: when the high level detects stagnation, it takes charge of diversifying, while the low level intensifies, while retaining the idea of history to evaluate the effectiveness of each LLH.

## 1.10 Organization of the Rest of the Paper

The content of the paper is structured as follows:

### 1.10.1 Formulation of the Problem

This section presents a formal definition of the Traveling Salesman Problem (TSP) and its mathematical formulation. The TSP is described as the problem of finding the shortest route that visits each city exactly once and returns to the starting city. The necessary variables and constraints are defined, such as the number of cities ($n$), the distances between cities ($d_{ij}$), and the binary decision variables ($x_{ij}$).

### 1.10.2 Description of the Solution

This section details the approach taken to solve the TSP using a hyper-heuristic.

**Introduction** Presentation of the high-level strategy and the criteria for selecting low-level heuristics.

**High-Level Strategy** Use of the Ant Colony Optimization (ACO) meta-heuristic, adapted for the TSP. Updates of pheromone levels and visibility values to guide heuristic selection.

**Low-Level Heuristics** Description of construction heuristics (e.g., Nearest Neighbor, Double Nearest Neighbor) and improvement heuristics (e.g., 2-Opt), and their hybrid combination.

**Data Structures** Presentation of key data structures: distance matrix, list of visited cities, pheromone matrix, and visibility vector.

### 1.10.3 Knowledge Base Generation and Learning

This section covers the creation of a knowledge base to enhance the hyper-heuristic.

**Knowledge Base Generation** Data collection from the execution of the hyper-heuristic on different benchmarks.

**Preprocessing** Filtering the data to retain only those meeting performance criteria.

**Learning** Use of a Random Forest model to train the knowledge base with data divided into training and test sets.

### 1.10.4 Tests and Results

This section presents the tests conducted to evaluate the effectiveness of the proposed solution.

**Comparative Analysis** Presentation of the tests and performance metrics used to evaluate the solution, with radar charts illustrating relative percentage deviations (RPD).

**Comparison of Hyper-Heuristics** Analysis of the performances of the TS-ILS hyper-heuristics, ACO with 2-OPT, and ACO without 2-OPT across different benchmarks. Discussion of the advantages and disadvantages of each approach.

#### 1.10.5 Conclusion

The conclusion summarizes the main results obtained and discusses the contributions of the study. It also proposes directions for future research aimed at further improving hyper-heuristic methods for solving the TSP.

## 2 Formulation of the Problem

**Definition**

The Traveling Salesman Problem (TSP) is a classic problem in combinatorial optimization. It can be stated as follows:

Given a set of cities and the distances between each pair of cities, the objective is to find the shortest possible route that visits each city exactly once and returns to the original city.

### Objective Function

The objective is to minimize the total travel distance:

$$\text{Minimize:} \quad \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} d_{ij} \cdot x_{ij}$$

where:

- $d_{ij}$ represents the distance between city $i$ and city $j$.
- $x_{ij}$ is a binary variable that equals 1 if the tour goes directly from city $i$ to city $j$, and 0 otherwise.

### Constraints

The constraints ensure that the solution forms a valid tour that visits each city exactly once.

**1. Each city must be left exactly once**

$$\sum_{j=1, j\neq i}^{n} x_{ij} = 1, \quad \text{for } i = 1, 2, \ldots, n$$

This constraint guarantees that from each city $i$, there is exactly one outgoing edge.

**2. Each city must be entered exactly once**

$$\sum_{i=1, i\neq j}^{n} x_{ij} = 1, \quad \text{for } j = 1, 2, \ldots, n$$

This constraint ensures that each city $j$ has exactly one incoming edge.

**3. Subtour elimination constraints**

$$\sum_{i \in S} \sum_{j \notin S} x_{ij} \geq 2, \quad \text{for all non-empty proper subsets } S \subset \{1, 2, \ldots, n\}$$

These constraints prevent the formation of subtours, ensuring that the tour visits all cities in a single cycle. For any subset $S$ of cities, there must be at least two edges leaving $S$.

**4. Binary decision variables**

$$x_{ij} \in \{0, 1\}, \quad \text{for } i, j = 1, 2, \ldots, n$$

This constraint specifies that the decision variables $x_{ij}$ are binary, indicating whether the edge between city $i$ and city $j$ is part of the tour.

# 3 Description of the Solution based on ACO

## 3.1 Introduction

As previously mentioned, our initial approach focused on selecting an effective high-level strategy, which includes defining a robust selection strategy and an acceptance strategy, as well as making a judicious choice of low-level heuristics to be used. Here, we aim to find a solution that strikes a good balance between execution time and solution quality. For the high-level strategy, we chose the one that provided the best results in terms of these criteria. Similarly, we selected the low-level heuristics that satisfied the criteria in terms of cost and time.

### 3.1.1 High-Level Strategy

We chose the ACO meta-heuristic as it performed best in terms of execution time and solution quality among all the solutions implemented this semester. However, it needed to be adapted to our problem

In the Ant System, the key properties are pheromone updating activities and visibility values. In this work, the pheromone value is updated based on improvements made by the heuristics. Improvement is measured as the difference between the best quality found during the current iteration and the best quality found during the previous iteration. A higher pheromone level indicates a higher confidence level for other ants to apply the same heuristics. After completing a tour the ant puts a pheromone value on the path, and if it finds a better solution it puts more pheromones, giving more advantage to well-performing heuristics. This approach ensures that we favor heuristics that provide better costs. To prevent the pheromone value from becoming unbounded, an evaporation process is applied where edges corresponding to poorly performing heuristics are penalized by not receiving any pheromone.

The visibility value corresponds to the amount of CPU time a heuristic took to complete its task. Heuristics that require less CPU time are preferred. In ACO hyper-heuristics, there are two procedures: local and global updates. The local update is that each ant keeps track of its path in a local matrice where it adds the pheromones and the global update uses the best solution found in the current iteration to update the pheromone trail, rewarding the path that produces the best solution globally so that every ant can view the changes.

### 3.1.2 Low-Level Heuristics

We selected two types of hyper-heuristics and built a hybrid approach, which is a combination of the other two:

- **Constructive Heuristics**: These heuristics build a solution from scratch. The construction process is guided by the heuristic information and pheromone values to incrementally build a high-quality solution.

- **Improvement Heuristics**: These heuristics start with an existing solution and iteratively improve it. The improvement process is guided by the same heuristic information and pheromone values to refine the solution and enhance its quality.
- **Constructive and improvement heuristics**: Finally, we proposed a combination of the two low-levels mentioned above. This proved to be highly effective based on the low-level heuristics we chose. It leverages the strengths of both construction and improvement heuristics to provide superior results.

## 3.2 Data Structure

The main data structures used in the constructive approach include (let $N$ be the number of cities and $K$ the number of low-level heuristics):

- **Distance Matrix**: A matrix representing the distances between cities of size $N \times N$.
- **Tour**: A list representing the sequence of cities visited of size $N$.
- **Heuristic Tour**: A list representing the sequence of heuristics chosen.
  - **Constructive**: of size $N$.
  - **Improvement**: of size min(maximum number of iterations, number of iterations without stagnation).
  - **Hybrid**: of size $N+$ number of heuristic applications during improvement.
- **Pheromones**: A matrix representing the desirability of transitions between heuristics of size $K \times K$.
- **Visibility**: A vector representing the CPU time required by each heuristic of size $K$.

## 3.3 Detailed Description

The Ant Colony Optimization (ACO) algorithm with a hyperheuristic approach is designed to solve optimization problems by simulating the foraging behavior of ants. This algorithm can be broken down into the following detailed steps:

### 3.3.1 High Level Heuristic

1. Initialization of Parameters : In the initialization step, we set the initial values for the pheromone levels, visibility, and other input parameters which are the number of ants (*num_ants*), the number of iterations (*num_iterations*), the evaporation rate ($\rho$), and the parameters $\alpha$, $\beta$, and $Q$ which influence the pheromone update rules.

$$\tau_{ij} = \text{initial pheromone level for all paths } (i, j) \tag{1}$$

$$\eta_i = \text{initial visibility for all heuristics } i \tag{2}$$

2. Iteration Check : The algorithm checks if the maximum number of iterations has been reached. If so, it terminates and returns the best solution found. Otherwise, it proceeds to the next step.

3. Initialization of Local Pheromones : For each iteration, a local pheromone matrix $\tau_{ij}^{\text{local}}$ is initialized to zero for all heuristic pairs $(i, j)$.

$$\tau_{ij}^{\text{local}} = 0 \text{ for all } i, j \tag{3}$$

4. Ants Generation and Solution Construction : For each ant, a solution is constructed by sequentially applying the selected low-level heuristics. The probability of selecting a heuristic is influenced by the global pheromone levels and visibility, which is updated based on the CPU time required by each heuristic.

The probability $P_{ij}$ of choosing heuristic $j$ after heuristic $i$ is given by:

$$P_{ij} = \frac{(\tau_{ij})^{\alpha} (\eta_j)^{\beta}}{\sum_{k \in \text{allowed}} (\tau_{ik})^{\alpha} (\eta_k)^{\beta}} \tag{4}$$

where: $\tau_{ij}$ is the pheromone level for the path from heuristic $i$ to heuristic $j$, $\eta_j$ is the visibility (inverse of CPU time) of heuristic $j$, $\alpha$ and $\beta$ are parameters that control the influence of pheromone and visibility.

The chosen heuristic extends the current tour and updates the visibility to favor heuristics with lower CPU times.

5. Updating Best Solution : After constructing a solution, the algorithm updates the best solution found if the current solution has a lower cost.

6. Pheromone Update : Local pheromones are updated based on the quality of the solutions generated by the ants. The update rule for local pheromones is given by:

$$\tau_{ij}^{\text{local}} = \tau_{ij}^{\text{local}} + \frac{Q}{\text{cost}} \tag{5}$$

where: $Q$ is a constant, cost is the cost of the solution.

7. Global Pheromone Update : After all ants have constructed their solutions, the global pheromone matrix is updated using the local pheromone updates and the evaporation rate $\rho$. The update rule for global pheromones is given by:

$$\tau_{ij} = (1 - \rho)\tau_{ij} + \tau_{ij}^{\text{local}} \tag{6}$$

where: $\rho$ is the evaporation rate.

**Algorithm 1** Ant Colony Optimization Hyper-Heuristic

```
1: Initialize Parameters: pheromones $\tau_{ij}$, visibility $\eta_i$, number of ants $num\_ants$, number of
   iterations $num\_iterations$, evaporation rate $\rho$, $\alpha$, $\beta$, $Q$
2: Set $best\_solution \leftarrow$ None
3: Set $best\_distance \leftarrow \infty$
4: Set $best\_tour \leftarrow []$
5: Set $best\_time_s \leftarrow 0$
6: for $iter = 1$ to $num\_iterations$ do
7:    Initialize $\tau_{ij}^{\text{local}} \leftarrow 0$ for all heuristic pairs $(i, j)$
8:    for $ant = 1$ to $num\_ants$ do
9:       Initialize $tour \leftarrow []$, $visited \leftarrow []$, $time_s \leftarrow 0$
10:      Randomly select an initial heuristic $current\_heuristic$
11:      Set $solution \leftarrow [current\_heuristic]$
12:      while len($tour$) < NumberofCities + 1 do
13:         Calculate probabilities $P_{ij}$ for each heuristic $j$ in $low\_l\_heuristics$:
               $$P_{ij} = \frac{(\tau_{ij})^{\alpha}(\eta_j)^{\beta}}{\sum_{k \in H}(\tau_{ik})^{\alpha}(\eta_k)^{\beta}}$$
14:         Select next heuristic $selected\_heuristic$ based on $P_{ij}$
15:         if $selected\_heuristic == 0$ then
16:            Apply Heuristic H1
17:         else if $selected\_heuristic == 1$ then
18:            Apply Heuristic H2
19:         else if $selected\_heuristic == 2$ then
20:            Apply Heuristic H3
21:         else if $selected\_heuristic == 3$ then
22:            Apply Heuristic H4
23:         else if $selected\_heuristic == 4$ then
24:            Apply Heuristic H5
25:         else
26:            Print "Unknown Heuristic"
27:         end if
28:         Update $tour$, $visited$, $cost$, $elapsed\_time$
29:         $time_s \leftarrow time_s + elapsed\_time$
30:         Append $selected\_heuristic$ to $solution$
31:         $current\_heuristic \leftarrow selected\_heuristic$
32:         Update $\eta_{current\_heuristic} \leftarrow \eta_{current\_heuristic} + elapsed\_time$
33:      end while
34:      if $cost < best\_distance$ then
35:         $best\_solution \leftarrow solution$
36:         $best\_distance \leftarrow cost$
37:         $best\_tour \leftarrow tour$
38:         $best\_time_s \leftarrow time_s$
39:      end if
40:      for $i = 0$ to len($solution$) $- 2$ do
41:         $\tau_{solution[i]solution[i+1]}^{\text{local}} \leftarrow \tau_{solution[i]solution[i+1]}^{\text{local}} + \frac{Q}{cost}$
42:      end for
43:   end for
44:   for each pair $(i, j)$ do
45:      $\tau_{ij} \leftarrow (1 - \rho)\tau_{ij} + \tau_{ij}^{\text{local}}$
46:   end for
47: end for
48: return $best\_solution$, $best\_distance$, $best\_tour$, $best\_time_s$
```

### 3.3.2 Low Level Heuristics

1. **Constructive low-level**

The constructive approach involves building a solution step-by-step by sequentially applying selected low-level heuristics. This method aims to generate an initial feasible solution that can then be refined by other techniques. For this purpose, we utilized the heuristics developed earlier this semester, as described in the introduction section. Our selection criteria focused on heuristics with low execution time, optimal cost, and minimal complexity. We modified these heuristics to accept the current tour as input and return the next city to visit, the cost of the updated tour, and the execution time. This heuristics are :

- **Nearest Neighbour**: This heuristic selects the closest unvisited city to the last city in the current tour.
- **Double-Ended Nearest Neighbour**: This heuristic selects the closest unvisited city to either end of the current tour, thereby balancing the tour construction process.
- **Nearest and Loneliest Neighbour**: This heuristic prioritizes selecting the closest unvisited city to the last city in the tour, with an additional consideration for most isolated city .
- **Double-Ended Nearest and Loneliest Neighbour**: This heuristic combines the principles of the Double-Ended Nearest Neighbour and Nearest and Loneliest Neighbour approaches.It selects the loneliest unvisited city from either end of the current tour, thus aiming to diversify the search and avoid local optima.

**Example** We start with a scenario involving four cities: A, B, C, and D. Each ant iteratively constructs a tour using selected heuristics and updates local pheromone levels.

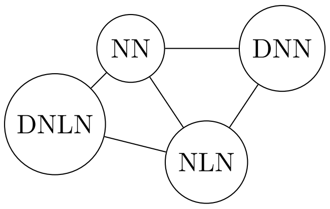
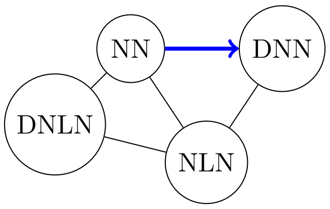
| It | Path State | Variables |
|---|---|---|
| 1 |  | **Array of Visibility:** $[1, 1, 1, 1]$ **Matrix of Local Pheromones:** $\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$ |
| 2 |  | **Array of Visibility:** $[1 + T(NN), 1 + T(DNN), 1, 1]$ **Matrix of Local Pheromones:** $\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$ |

Table 1: Two first iterations of an ant to select the next heuristic using visibility (T respresents the CPU time of an heuristic)

At the end of one ant iteration we will have

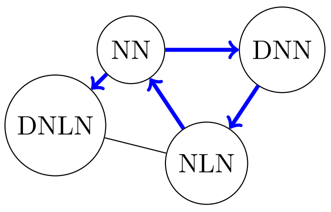
| **Path State** | **Variables** |
|---|---|
|  | **Array of Visibility:** $[1+2*T(NN), 1+T(DNN), 1+T(NLN), 1+T(DNLN)]$ **Matrix of Local Pheromones:** $\begin{bmatrix} 0 & \tau_{12} & 0 & \tau_{14} \\ 0 & 0 & \tau_{23} & 0 \\ \tau_{31} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$ |

Table 2: Final iteration of an ant updating the local pheromones and the visibility

$$\tau_{ij}^{\text{local}} = \tau_{ij}^{\text{local}} + \frac{Q}{\text{cost}} \tag{7}$$

At the end of each iteration, the best solution among all ants is identified, and the global pheromone levels are updated using the formula:

$$\tau_{ij}^{\text{global}} = (1-\rho) \cdot \tau_{ij}^{\text{global}} + \sum_k \Delta\tau_{ij}^{(k)} \tag{8}$$

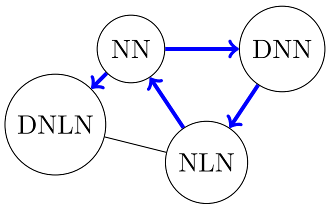
| **Path State** | **Variables** |
|---|---|
|  | **Tour solution:** $[A, C, D, B]$ **Matrix of global Pheromones:** $\begin{bmatrix} 0 & \tau_{12} & 0 & \tau_{14} \\ 0 & 0 & \tau_{23} & 0 \\ \tau_{31} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$ |

Table 3: The maximum number of ants is reached and the global pheromones are updated

**Final Solution Details**:

- **Path Heuristics**: NN → DNN → NLN → NN → DNLN
- **Optimal Tour**: A → C → D → B
- **Optimal Cost**: 80 units
- **Execution Time**: 0.3 seconds

2. **Improvement low-level**

The improvement approach focuses on iteratively refining an initial solution to improve its quality using advanced heuristic techniques. This method builds upon the initial feasible solution generated randomly, aiming to achieve near-optimal solutions through systematic adjustments. We selected these heuristics based on their efficiency, cost-effectiveness, and computational simplicity, adapting them to enhance the current tour by reducing its cost and refining its structure over successive iterations.We used in the enhancement approach these heuristics :

- **2-opt Heuristic**: Improve the tour by iteratively reversing segments of the tour to reduce its length.

- **Simulated Annealing**: This heuristic explores the solution space by probabilistically accepting worse solutions to escape local optima, with the probability of acceptance decreasing over time.
- **Tabu Search**: This heuristic uses memory structures to avoid cycles and guides the search process, prohibiting or penalizing moves that lead to recently visited solutions, thereby encouraging exploration of new areas in the solution space.

**Example**

We use the previous scenario involving four cities: A, B, C, and D. Each ant iteratively refines an initial solution using selected heuristics and updates local pheromone levels.

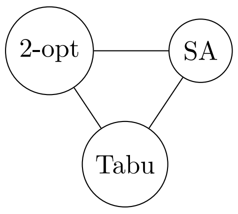
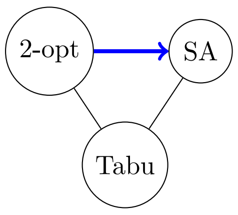
| It | Path State | Variables |
|---|---|---|
| 1 |  | **Array of Visibility:** $[1, 1, 1]$ **Matrix of Local Pheromones:** $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ |
| 2 |  | **Array of Visibility:** $[1 + T(2 - opt), 1 + T(SA), 1]$ **Matrix of Local Pheromones:** $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ |

Table 4: Two first iterations of an ant to select the next heuristic using visibility (T respresents the CPU time of an heuristic)

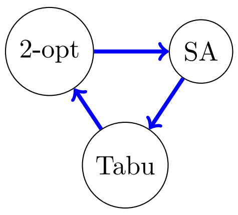
| Path State | Variables |
|---|---|
|  | **Array of Visibility:** $[1+2*T(2-opt), 1+T(SA), 1+T(Tabu)]$ **Matrix of Local Pheromones:** $\begin{bmatrix} 0 & \tau_{12} & 0 \\ 0 & 0 & \tau_{23} \\ \tau_{31} & 0 & 0 \end{bmatrix}$ |

Table 5: Final iteration of an ant updating the local pheromones and the visibility

$$\tau_{ij}^{\text{local}} = \tau_{ij}^{\text{local}} + \frac{Q}{\text{cost}} \tag{9}$$

At the end of each iteration, the best solution among all ants is identified, and the global pheromone levels are updated using the formula:

$$\tau_{ij}^{\text{global}} = (1 - \rho) \cdot \tau_{ij}^{\text{global}} + \sum_{k} \Delta\tau_{ij}^{(k)} \tag{10}$$

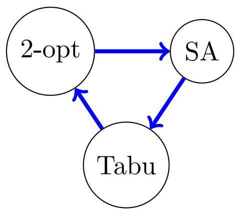
| Path State | Variables |
|---|---|
|  | **Tour solution:** $[A, B, D, C]$ **Matrix of global Pheromones:** $\begin{bmatrix} 0 & \tau_{12} & 0 \\ 0 & 0 & \tau_{23} \\ \tau_{31} & 0 & 0 \end{bmatrix}$ |

Table 6: The maximum number of ants is reached and the global pheromones are updated

**Final Solution Details**:

- **Path Heuristics**: 2-opt → SA → Tabu
- **Optimal Tour**: A → B → D → C
- **Optimal Cost**: 75 units
- **Execution Time**: 2 seconds

After testing both approaches, we observed distinct advantages and limitations for each:

- **Constructive Approach**: This method proved to be simpler and faster. It quickly generated an initial feasible solution using low-level heuristics, making it suitable for scenarios where time efficiency is crucial.
- **improvement Approach**: Although this method took more time, it provided a more optimal solution by iteratively refining the initial tour. This approach is beneficial when the quality of the solution is the priority.

Given these observations, we decided to develop a hybrid approach that leverages the strengths of both methods.

3. **Hybridation Approach**

The hybrid approach combines the simplicity and speed of the constructive approach with the quality improvement capabilities of the improvement approach. In this approach, we use the previously described constructive heuristics and a modified 2-opt heuristic for improvement . The 2-opt heuristic is selected because it provides significant improvements in tour cost without the excessive execution time required by Simulated Annealing (SA) and Tabu Search (TS). We modified the 2-opt heuristic to take the current tour as input and returns an improved tour, its cost, and execution time. So the heuristics used are:

- Nearest Neighbour (NN)
- Double-Ended Nearest Neighbour (DNN)
- Nearest and Loneliest Neighbour (NLN)
- Double-Ended Nearest and Loneliest Neighbour (DENLN)
- 2-opt

Initially, we use the constructive methods to sequentially build the tour. During this process, if the algorithm selects the 2-opt heuristic, it applies it once to enhance the current tour whenever the tour length exceeds two cities. We continue constructing the tour until all cities are visited. The path of heuristics reflects the tour's length and the number of 2-opt improvements applied.

**Example**

We continue with the scenario involving four cities: A, B, C, and D. Each ant iteratively constructs and improve a tour using selected heuristics and updates local pheromone levels.

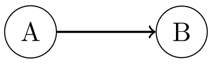
- **Initialization**:
  - Array of visibility initialized to 1 (e.g., $[1, 1, 1, 1]$).
  - Matrix of pheromones initialized to a starting value (e.g., $[\tau_{ij}]$).
  - Array of path heuristics initialized as empty (e.g., [ | | | ]).

Let's represent the steps graphically to visualize the tour construction:

- **Step 1: Double-Ended Nearest Neighbour (DNN)**

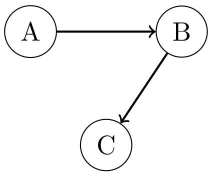
  **Tour:** A

- **Step 2: Nearest Neighbour (NN)**



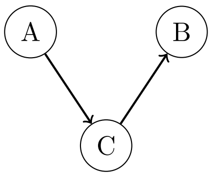
  **Tour:** A → B

- **Step 3: Double-Ended Nearest Neighbour (DNN)**



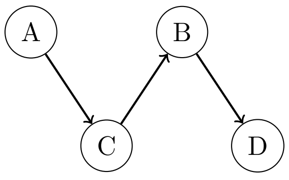
  **Tour:** A → B → C

- **Step 4: Apply 2-opt**



  **Tour:** A → C → B

- **Step 5: Nearest and Loneliest Neighbour (NLN)**



  **Tour:** A → C → B → D

| Heuristic | Tour |
|---|---|
| DNN | A |
| NN | A → B |
| DNN | A → B → C |
| 2-opt | A → C → B |
| NLN | A → C → B → D |

Table 7: Step-by-step application of heuristics and resulting tour

## Final Solution Details

- **Path Heuristics**: DNN → NN → DNN → (2-opt applied) → NLN
- **Optimal Tour**: A → C → B → D
- **Optimal Cost**: 75 (units, for example)
- **Execution Time**: 0.4 seconds

## 3.4 Learning

Now that we have constructed a hyper-heuristic, it was essential to ensure it fully meets the objectives of hyper-heuristics mentioned in 1.4 section. This is because ou solution (ACO) still depends on input parameters. And to address this issue, we employed an offline learning technique, commonly used in selection hyper-heuristics and is the primary approach of Case-Based Reasoning (CBR) so that we can initialize parameters automatically for starting the resolution.

Before constructing the knowledge base, we needed to define a case representation consisting of features, each associated with weights. We defined the following features:

- The entire distance matrix of our problem instance.
- The heuristics used.
- The number of cities.
- The number of heuristics.

*Note*: While more refined features could be included, we limited ourselves to these due to time constraints. Nonetheless, these features provided satisfactory results.

### 3.4.1 Generation of the Knowledge Base

After defining the features, it was time to construct our knowledge base, which was a challenging task. To achieve this, we needed to execute our hyper-heuristic multiple times on different benchmarks, varying the parameters. We proceeded as follows:

- **Step 1:** Improve the execution time by enhancing the hyper-heuristic.
- **Step 2:** Generate CSV files containing all executions.

For the first step, we modified the algorithm to integrate threads, so each ant had its own execution thread. For the second step, we looped through the parameters mentioned in the previous section randomly across 3 benchmarks (Att48, Eil51, Berlin52, Eil76), recording all the following information: input instance, number of cities, heuristics, number of heuristics, number of ants, number of iterations, alpha, beta, initial pheromone, Q, evaporation rate, best solution, best distance, best tour, best time, best path, and the benchmark name.

#### 3.4.2 Preprocessing

Afterward, we began the preprocessing stage, which involved filtering our data to retain only those that meet the following two conditions:

$$\text{Best cost} < \text{average(Best cost)}$$

$$\text{Best time} < \text{average(Best time)}$$

#### 3.4.3 Training

Next, we divided 70% of the data for training and the remaining 30% for testing, and injected this into a Random Forest model.

The Random Forest algorithm was chosen because of its robustness and effectiveness in handling large datasets with many features. It works by creating multiple decision trees during training time and outputting the mode of the classes (classification) or mean prediction (regression) of the individual trees.

#### 3.4.4 Testing

The testing section will be detailed in the section *Tests and Results.*

## 4 Solution inspired by TS-ILS Hyper-Heuristic:

The TS-ILS Hyper-Heuristic is one of the effective methods for solving many optimization problems such as TSP, BP, SAT, etc.

- TS: Thompson Sampling.
- ILS: Iterated Local Search.

This hyper-heuristic idea was published in 2023 in the International Journal of Advanced Computer Science and Applications, belonging to The Science and Information Organisation.

### 4.1 Principle of TS-ILS:

The Thompson Sampling Iterated Local Search (TS-ILS) hyper-heuristic is an approach that selects Low-Level Heuristics (LLH) to introduce perturbations or mutations to a current solution. It then subjects the perturbed solution to local search for improvements, and the resulting solution is considered for acceptance. This method offers six possible configurations, but we illustrate only one here. The TS-ILS logic aims to distribute the workload between the high and low levels as follows:

- **High Level:** is tasked with selecting a set of effective LLHs. After applying an LLH, the process involves a local search to enhance the returned solution. LLHs are chosen based on their historical effectiveness using parameters alpha and beta: a high alpha indicates efficiency, while a high beta indicates the opposite. The utility of an LLH is determined by the difference alpha - beta; a positive result indicates utility. This approach enables real-time adaptation of LLH selection to various problem instances. The main objective of High Level is to intensify the solution after perturbing it with low-level heuristics.

- **Low Level:** Diversification through perturbation or mutation.

This hyper-heuristic, as studied and tested, follows these key steps:

#### 4.1.1 Initialisation:

1. **Set of perturbation (or mutation) heuristics** E=H1,H2,H3,... where Cardinal(E)=n.
2. **Perturbation Strength:** Higher perturbation strength leads to larger perturbations, allowing the algorithm to explore a wider search space. Conversely, lower perturbation strength leads to lighter perturbations, allowing for more localized exploration.
3. **Alpha Vector (size n):** The more successfully a heuristic is chosen, the higher its associated alpha parameter.
4. **Beta Vector (size n):** The more a heuristic is chosen but does not improve the solution, the higher its associated beta parameter.
5. **Generate an initial solution S0:** randomly.
6. **Stopping Condition:** Until maxiter iterations.

#### 4.1.2 Main Loop:

1. **Select LLH:** Calculate the utility of heuristics and select a better configuration, i.e., a more effective heuristic Hi that satisfies: Alpha(i) - Bêta(i) = max ( Alpha(j) - Béta(j),for j from 1 to n.
2. **Apply Hi on the current S.**
3. **Local search:** Further improve the solution through local search.
4. **Acceptance test** of the solution based on a certain probability.
5. **If the solution is improved**, the selected heuristic for this improvement receives a positive point in the alpha vector; otherwise, it receives a positive point in the beta vector (this update of alpha and beta vectors measures the effectiveness of different heuristics over time).
6. **Adjust the perturbation strength** according to the defined logic at each iteration.

#### 4.1.3 END:

The process repeats until the maximum number of iterations is reached. At the end of all iterations, the best solution and its associated cost are returned.

### 4.2 Advantages:

- Provides solutions close to exact solutions.
- High-level strategy based on history (Alpha and Beta parameters) allows the algorithm to adapt to different problem instances with a high level of abstraction.
- Encourages intensification through local search.
- Encourages diversification through perturbation (perturbation heuristics).

### 4.3 Disadvantages:

May take time if the number of iterations is high and/or the number of cities (or variables) in the problem is high.

**Algorithm 2** TS-ILS Hyper-heuristic

```
1: $M \leftarrow \{m_1, m_2, \ldots, m_n\}$
2: $\alpha \leftarrow \{0, \ldots, 0\}$
3: $\beta \leftarrow \{0, \ldots, 0\}$
4: $S_0 \leftarrow$ generateInitialSolution()
5: $S_b \leftarrow S_0$
6: while $\neg$stopping_condition do
7:     $\Phi \leftarrow \{\phi_1, \ldots, \phi_n\} \leftarrow$ generateUtilityValues()
8:     select $i$ from $[1..n]$: $\phi_i$ maximizes $\Phi$
9:     $S' \leftarrow$ perturb($S_0, i$)
10:    $S'' \leftarrow$ localSearch()
11:    if ($S''$ is accepted) then
12:        $S_0 \leftarrow S''$
13:    end if
14:    if ($S'' < S_b$) then
15:        $\alpha_i \leftarrow \alpha_i + 1$
16:        updateLS()
17:        updateParam()
18:    else
19:        $\beta_i \leftarrow \beta_i + 1$
20:    end if
21:    updateLLH()
22: end while
23: return $S_b$
```

## 4.4 Proposed Solution:

Since TS-ILS aims to distribute the load between the low level (ensuring diversification) and the high level (ensuring intensification), we exploit this logic but invert the roles as follows:

- **High Level:** Diversification by the 'Ruin and Recreate' method, which randomly removes a certain number of cities from the route, then reinserts them at random positions to generate a new solution.
- **Low Level:** Intensification through improvement heuristics (typically based on local search).

## 4.5 LLHs used for testing TS-ILS:

1. **PermuteTwoCities:** In each iteration, two cities are randomly chosen and swapped.
2. **Shuffle:** A subsequence of cities is chosen and randomly reordered.
3. **DoubleBridgeMove:** The path is divided into four sequences which are then permuted.

The perturbed solution obtained is returned without checking if it is better than the entered solution.

## 4.6 The low-level heuristics (LLH) used in the proposed solution:

1. **2-OPT:** In each iteration, the algorithm examines all pairs of non-adjacent points in the route and evaluates the cost of reversing the section between these points. The best solution obtained after these reversals is retained.
2. **DoubleBridgeMove:** The path is divided into four sequences which are then permuted. The best solution obtained after these changes is retained.

**Algorithm 3** Proposed TS-ILS Hyper-heuristic Solution

```
1: Initialize $M \leftarrow \{m_1, m_2, \ldots, m_n\}$
2: Initialize $\alpha \leftarrow \{0, \ldots, 0\}$
3: Initialize $\beta \leftarrow \{0, \ldots, 0\}$
4: $S_0 \leftarrow$ generateInitialSolution()
5: $S_b \leftarrow S_0$
6: while $\neg$stopping_condition do
7:     $\Phi \leftarrow \{\phi_1, \ldots, \phi_n\} \leftarrow$ generateUtilityValues()
8:     Select $i$ from $[1..n]$ such that $\phi_i$ maximizes $\Phi$
9:     $S' \leftarrow M_i()$
10:    $S'' \leftarrow S'$
11:    if $S''$ is accepted then
12:        $S_0 \leftarrow S''$
13:    end if
14:    if $S'' < S_b$ then
15:        $\alpha_i \leftarrow \alpha_i + 1$
16:        $S_b \leftarrow S''$
17:    else
18:        $\beta_i \leftarrow \beta_i + 1$
19:    end if
20:    if destruction_condition then
21:        $S'' \leftarrow$ ruin_recreate$(S', i)$
22:        updateParam()
23:    end if
24:    updateLLH()
25: end while
26: return $S_b$
```

3. **PermuteTwoCities:** In each iteration, two cities are randomly chosen and swapped. The best solution obtained after these permutations is retained.
4. **Shuffle:** A subsequence of cities is chosen and shuffled randomly. The best solution obtained after this shuffling is retained.

## 5 Tests and Results

### 5.1 Introduction

This section presents the results of our testing scenario to evaluate the effectiveness of various heuristics and hyper-heuristics applied to the Traveling Salesman Problem (TSP). The tests were conducted using Google Colab, a cloud-based platform that provides a Python 3 environment with a Google Compute Engine backend. The machine configuration includes 12.67 GB of RAM and 107.72 GB of disk space, of which 0.95 GB of RAM and 37.72 GB of disk space were utilized. These resources facilitated the execution of our algorithms on three different benchmarks from the TSPLIB database: PR76, EIL101, and GIL262. Each benchmark varies in size and complexity, offering a robust evaluation framework for our approaches.

### 5.2 Explanation of Charts and RPD Calculation

#### 5.2.1 Charts explanation

To visualize and compare the performance of different heuristics and hyper-heuristics, we use radar and scatter plots:

- **Radar Charts**: These charts depict the Relative Percentage Deviation (RPD) values of each heuristic/hyper-heuristic across different benchmarks. A lower RPD value indicates a closer approximation to the optimal solution.
- **Scatter Plots**: These plots illustrate the relationship between execution time and RPD for each heuristic/hyper-heuristic. This helps to analyze the trade-off between solution quality and computational efficiency.

#### 5.2.2 Relative Percentage Deviation calculation

**(RPD)** is calculated as follows:

$$\text{RPD} = \frac{(C - C_{opt})}{C_{opt}} \times 100 \tag{11}$$

where $C$ is the cost of the solution provided by the heuristic/hyper-heuristic, and $C_{opt}$ is the known optimal cost of the TSP instance. RPD measures the deviation of the heuristic solution from the optimal solution as a percentage of the optimal solution, allowing for a standardized comparison across different instances.

### 5.3 Characteristics of the Instances

The selected benchmarks represent a range of problem sizes and complexities commonly used to test TSP algorithms. Table 8 summarizes the key characteristics of these instances.

| Benchmark | Size | Optimal Cost | Type |
|---|---|---|---|
| PR76 | 76 | 108159 | Euclidean |
| EIL101 | 101 | 629 | Euclidean |
| GIL262 | 262 | 2378 | Euclidean |

Table 8: Characteristics of Benchmarks Used

## 5.4 Hyperparameter Calibration

Proper parameter calibration is crucial for the effective performance of hyper-heuristics.

### 5.4.1 ACO-Hyperheuristic

For our tests, parameters were calibrated through the machine learning model which conducted preliminary runs:

- **Number of Ants:** This parameter specifies the number of ants (agents) that will be used in each iteration of the algorithm.
- **Number of Iterations:** This is the total number of iterations the algorithm will perform.
- **Pheromones:** Pheromones represent the accumulated knowledge guiding the search process. They are initialized with a small positive value and updated based on the solutions found by the ants.
- **Visibility:** Visibility represents the desirability of selecting specific heuristics. Initially equal for all heuristics, visibility is updated based on their performance to guide ants towards more effective heuristics.
- **Alpha:** Alpha is the pheromone importance parameter. It determines the relative influence of pheromone trails on the decision-making process of the ants.
- **Beta:** Beta is the visibility importance parameter. It determines the relative influence of heuristic information on the decision-making process of the ants.
- **Q:** Q is the pheromone deposit factor. It determines the amount of pheromone each ant deposits after completing its solution, which is inversely proportional to the cost of the solution.
- **Evaporation Rate:** This parameter represents the rate at which pheromones evaporate over time. It helps to avoid premature convergence by reducing the influence of older pheromone trails.

Figure 8 and 10 demonstarte the imapct of machine learning hybridization for hyperparameter tuning on ACO-2OPT performance.

### 5.4.2 TS-ILS-Hyperheuristic

TS-ILS is tailored for real-time operations, where initial parameter settings have negligible impact on final outcomes due to continuous adjustment of perturbation strength during execution. If initial perturbations fail to improve results, their intensity is dynamically augmented to enhance performance. Consequently, tests conducted on these parameters revealed minimal impact on outcomes, although increasing parameters like maximum iterations and perturbation strength extended execution durations.

- **Original TS-ILS-Hyperheuristic:**
  In this study, the TS-ILS hyper-heuristic parameters were adopted from the original article.
  - maximum_iteration=10
  - perturbation_strength = 2

- **Adjusted TS-ILS-Hyperheurisitic:**
  Given the dynamic nature of TS-ILS, our experimental approach aimed to streamline execution time without extensive parameter tuning.

  While sophisticated parameter testing was not conducted, we strategically selected values that prioritize efficient execution:

  - Max Iterations: 50
  - Perturbation Strength: 5
  - Destruction Factor: 25
  - Destruction Strength: 3

## 5.5 Performance Study and Results Analysis

We evaluated the performance of our proposed approaches by analyzing their behavior, evolution based on parameter values, and input instance characteristics. The analysis includes both construction and improvement heuristics, followed by hyper-heuristics, with detailed comparisons.

### 5.5.1 Simple Heuristics

Simple heuristics serve as a baseline for comparing the effectiveness of advanced approaches. We evaluated both construction and improvement heuristics.

**Construction Heuristics** The performance of construction heuristics was assessed using Relative Percentage Deviation (RPD) and execution time. The following heuristics were tested:

- **Christofides heuristic**
- **Double-nearest neighbor**
- **Nearest and loneliest neighbor**
- **Double-ended nearest and loneliest neighbor**

**Radar Chart for Relative Percentage Deviation (RPD) for Construction Heuristics**

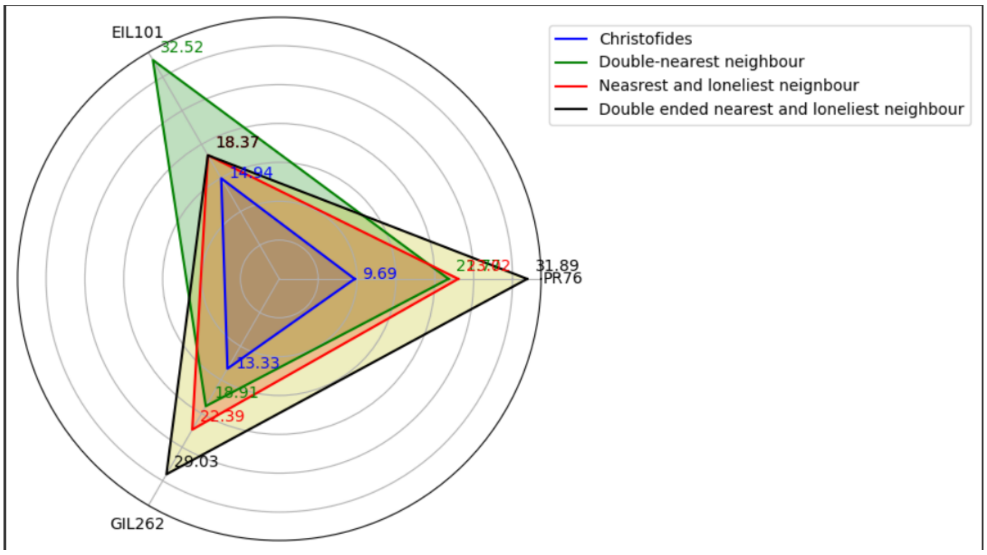
[H] 

Figure 3: Radar Chart for RPD for Construction Heuristics

The radar chart in Figure 3 shows the RPD values for each construction heuristic across the benchmarks. Christofides heuristic (blue) consistently achieves lower RPD values, indicating closer approximations to optimal solutions. Double-ended nearest and loneliest neighbor heuristic (yellow) displays the highest RPD values, especially for PR76 and GIL262, suggesting less effectiveness.

**Scatter Plot for RPD and Execution Time for Construction Heuristics**

[H]

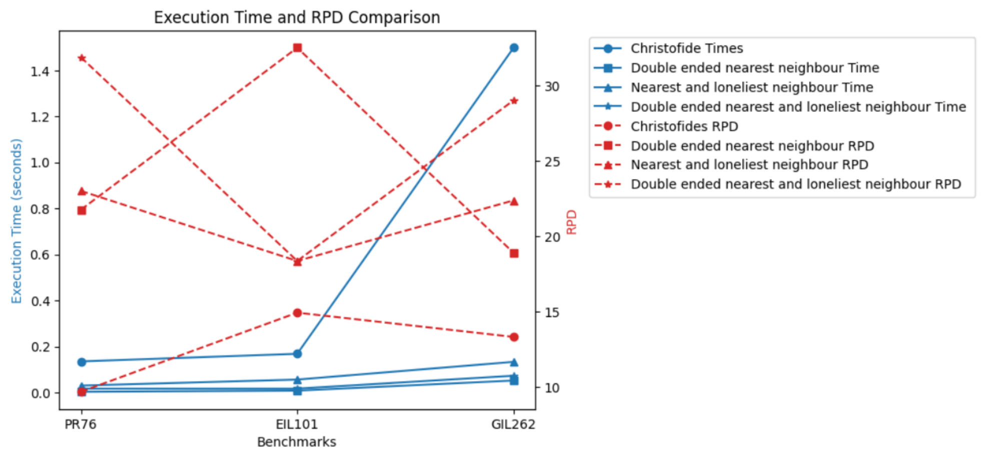
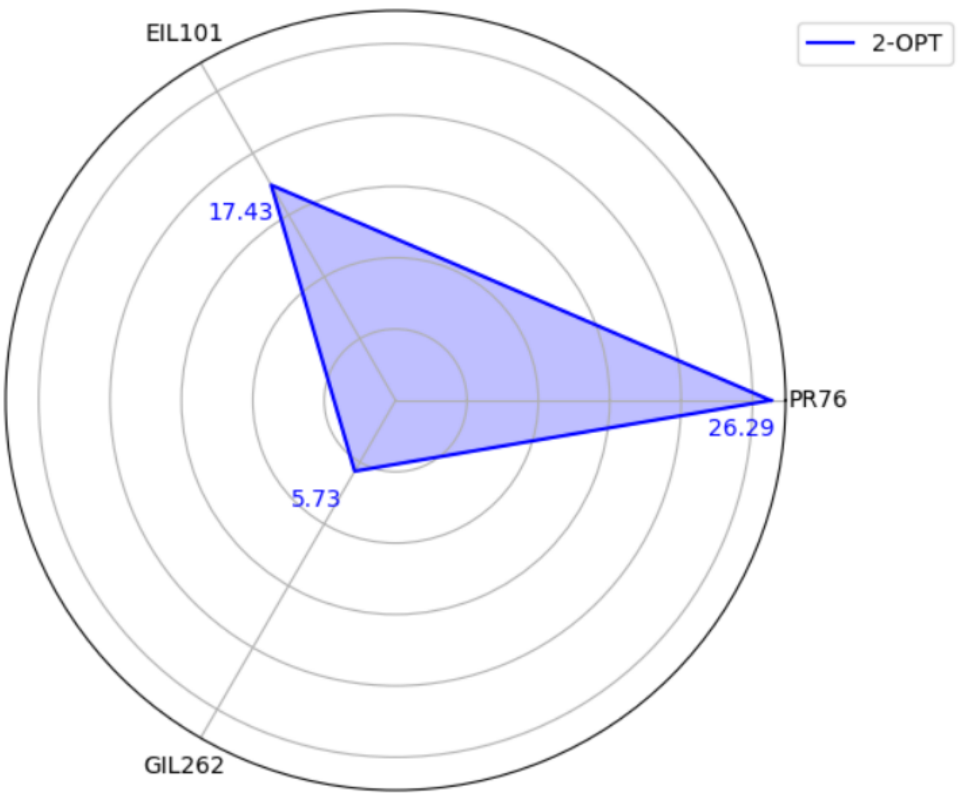


Figure 4: Scatter Plot for RPD and Execution Time for Construction Heuristics

The scatter plot in Figure 4 compares RPD values with execution times. Christofides heuristic, though computationally more intensive, provides better solution quality. Double-nearest neighbor heuristic is faster but less accurate, especially on benchmarks like EIL101.

**Improvement Heuristics** Improvement heuristics aim to refine initial solutions. We analyzed the performance of the 2-OPT heuristic.

**Radar Chart for Relative Percentage Deviation (RPD) for Improvement Heuristic**

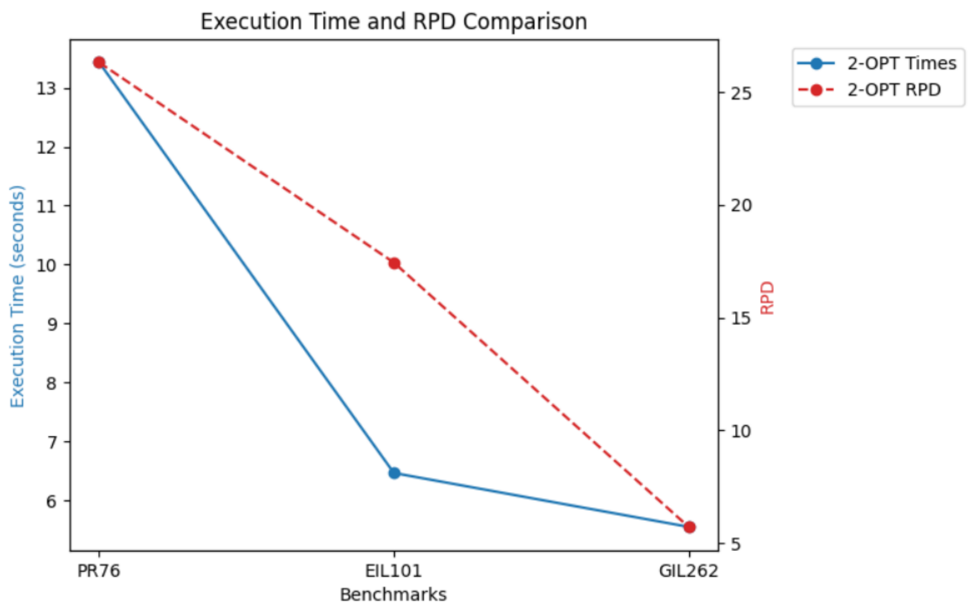


[H]

Figure 5: Radar Chart for RPD for Improvement Heuristic

The radar chart in Figure 5 highlights the RPD for the 2-OPT heuristic across benchmarks. While it performs well on larger datasets like GIL262 with the lowest RPD, its effectiveness decreases for smaller datasets such as PR76, as evidenced by a higher RPD.

**Scatter Plot for RPD and Execution Time for Improvement Heuristic**



[H]

Figure 6: Scatter Plot for RPD and Execution Time for Improvement Heuristic

The scatter plot in Figure 6 compares RPD values and execution times for the 2-OPT heuristic. Execution time is higher for PR76 but the heuristic achieves lower RPD for GIL262, indicating more efficient performance on larger datasets.

## 5.6 Comparative Study

We compared the proposed hyper-heuristic approach with simple heuristics and literature methods in terms of execution time and solution quality.

### 5.6.1 Hyper-Heuristics

The performance of hyper-heuristics was evaluated using RPD and execution times across the benchmarks.

**Radar Chart for Relative Percentage Deviation (RPD)**

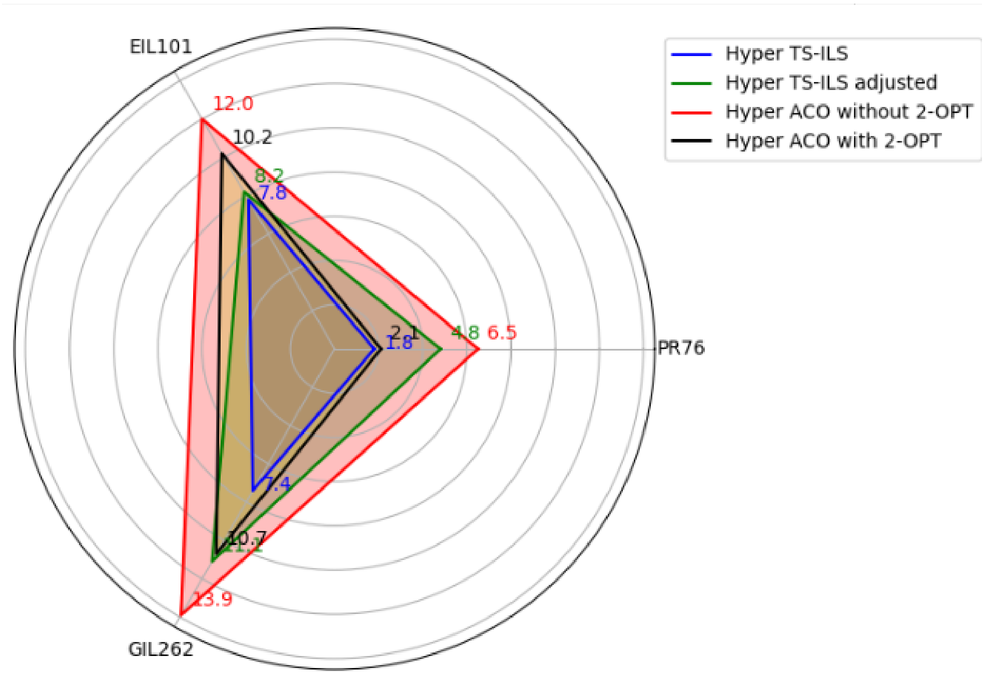


Figure 7: RPD Comparison of Hyper-Heuristics

Figure 7 suggests :

- **Hyper TS-ILS (Blue):** This is the best-performing hyperheuristic, with the lowest RPDs across all benchmarks, indicating solutions close to optimal.
- **Modified Hyper TS-ILS (Green):** It performs slightly worse than TS-ILS but remains competitive with ACO with 2-OPT, especially on the EIL101 and GIL262 benchmarks.
- **Hyper ACO with 2-OPT (Black):** Integrating 2-OPT optimization improves performance over ACO without 2-OPT, but this method does not always outperform TS-ILS.
- **Hyper ACO without 2-OPT (Red):** This hyperheuristic tends to have higher RPDs across all datasets compared to the other methods, indicating overall inferior performance.

**Impact of 2-OPT on ACO:** The integration of 2-OPT optimization into ACO (black curve) improves performance compared to ACO without 2-OPT (red curve), as evidenced by lower RPDs. This improvement is due to the diversification provided by 2-OPT.

**ACO-2OPT Hyper-Heuristic with and without Machine Learning Hybridization**

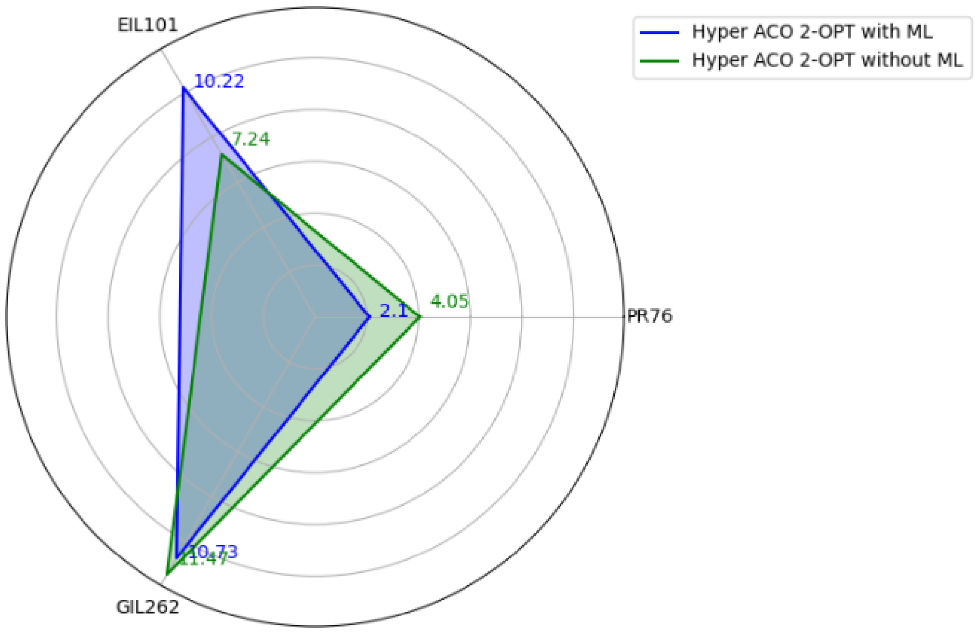


Figure 8: ACO-2OPT with/without ML Hybridization

- **Hyper ACO 2-OPT with ML (Blue):** This hyperheuristic shows lower RPDs on most benchmarks (PR76 and EIL101), indicating better performance and solutions closer to optimality compared to the hyperheuristic without ML.
- **Hyper ACO 2-OPT without ML (Green):** While competitive, this method shows slightly higher RPDs, particularly on PR76 and GIL262, suggesting less optimal solutions compared to its ML counterpart.

Figure 8 Integrating machine learning (ML) techniques into the ACO parameter selection process significantly improves hyperheuristic performance, resulting in lower RPDs, especially on PR76 and EIL101. This suggests that hybridization with ML techniques helps refine ACO parameters, leading to higher-quality solutions.

**Scatter Plot for RPD and Execution Time for Hyper-Heuristics:**

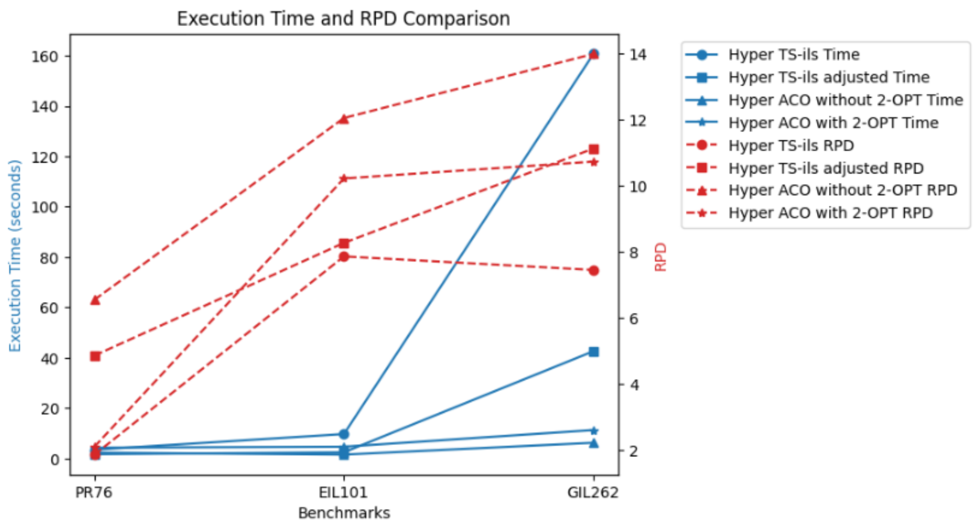


Figure 9: RPD and Execution Time for Hyper-Heuristics

The scatter plot in Figure 10 compares execution times and RPD values for different hyper-heuristics. Hyper TS-ILS (blue) demonstrates a good balance between execution time and RPD, particularly effective on larger benchmarks like GIL262.

- **Execution Times:** For all datasets, Hyper ACO without 2-OPT has the shortest execution times among the four methods. However, Hyper TS-ILS and adjusted Hyper TS-ILS have similar but slightly longer execution times than Hyper ACO without 2-OPT. Hyper ACO with 2-OPT has longer execution times on PR76 and EIL101 but remains competitive on GIL262.

- **RPD:** The RPD results show a different trend. Hyper ACO without 2-OPT generally has higher RPDs, indicating greater deviations from the best-known solution. Hyper TS-ILS has lower RPDs, suggesting relatively better performance in terms of solution quality. Notably, Hyper ACO with 2-OPT and Modified Hyper TS-ILS show intermediate RPDs and are competitive with each other but remain better than ACO without 2-OPT.

**Scatter Plot for ACO-2opt with and without ML hybridation**

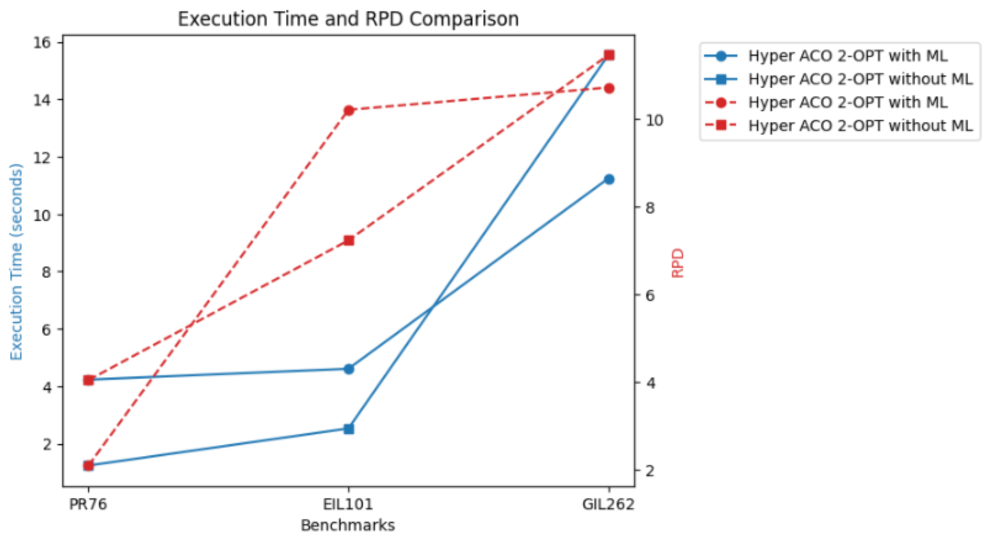


Figure 10: RPD and Execution Time for Hyper-Heuristics

This figure 10 allows for a comparison of the performance of the two versions of the ACO 2-OPT hyperheuristic in terms of execution time and solution quality (measured by RPD). Hyper ACO 2-OPT with ML appears to offer better solution quality (lower RPD) on PR76 and EIL101 at the cost of longer execution times. Hyper ACO 2-OPT without ML is faster on PR76 and EIL101 but provides inferior solution quality (higher RPD). Overall, incorporating machine learning techniques into ACO parameter selection improves solution quality for certain datasets at the expense of longer execution times, offering an interesting trade-off between performance and solution quality for the traveling salesman problem.

## 6 Synthesis about hyperheuristic and heuristic performance

### 6.1 Performance comparaison:

- **Solution Quality:** Hyper-heuristics generally outperform simple heuristics like construction and improvement heuristics, as seen from consistently lower RPD values.
- **Execution Efficiency:** Simple heuristics tend to be faster, but they sacrifice solution quality compared to hyper-heuristics, which achieve more accurate results despite longer execution times in some cases.

### 6.2 Trade-offs and Considerations:

- **Complexity, accuracy and efficiency:** the choice between simple heuristics and hyper-heuristics for the TSP depends on balancing the need for solution quality, computational efficiency, and adaptability to problem complexities. simple heuristics excel in speed and ease of implementation but may sacrifice optimal solutions in intricate TSP scenarios. Hyper-heuristics offer advanced capabilities to navigate complex solution spaces but come with higher computational demands and implementation complexity.

## 7 Conclusion

Hyperheuristics represent a powerful and flexible approach in the quest for optimal solutions to complex combinatorial problems like the Traveling Salesman Problem (TSP). Their charm lies in their ability to adapt and learn, dynamically selecting and combining simpler heuristics to navigate the intricate landscape of potential solutions. This adaptability makes them particularly appealing in varied and uncertain problem environments, where traditional heuristics might falter. However, the ongoing evolution and refinement of these techniques hold the key to unlocking even greater efficiencies and breakthroughs. By embracing these challenges and advancing the frontiers of knowledge in hyperheuristic design, we pave the way for transformative advancements in computational optimization and beyond. The future of hyperheuristics is not just about finding solutions; it's about reshaping how we approach and solve some of the world's most pressing computational challenges with innovation, rigor, and ethical integrity.

## 8 Bibliography

1. Mrs. Bessedik's Courses on combinatorial optimisation.
2. "Experimentation on Iterated Local Search Hyper-heuristics for Combinatorial Optimization Problems", Article, IJACSA, 2023