Order No.: ....

**MOHAMMED SEDDIK BENYAHIA UNIVERSITY-JIJEL**
**FACULTY OF EXACT SCIENCES AND COMPUTER SCIENCE**

**MASTER'S THESIS**

For the degree of:

## MASTER

In **COMPUTER SCIENCE**

**Option:** Information Systems and Decision Support

By:

**-Kribaa Chahine**

**-Chennib Ramdane**

Theme

# An Adaptive Large Neighborhood Search heuristic for the Team Orienteering Problem With Time Windows

Publicly defended on September 17, 2024, before the jury composed of:

| | | | |
|---|---|---|---|
| Mrs. | Bouaziz Hamida | University of Jijel | President |
| Mr. | Kerroum Ali | University of Jijel | Supervisor |
| Mrs. | Kerada Ouidad | University of Jijel | Examiner |

**Class of 2023**

**Abstract**

The Team Orienteering Problem with Time Windows (TOPTW) is a complex extension of the Team Orienteering Problem (TOP) that involves finding optimal routes for teams or vehicles to maximize the total score from visited locations, while respecting specific time windows. This problem is highly relevant in fields such as logistics, tourism, and disaster response, where routing must account for time-sensitive constraints. Due to its computational difficulty, solving TOPTW often requires advanced metaheuristic algorithms. This thesis focuses on the adaptive large neighborhood search algorithm (ALNS), which is tailored to efficiently explore diverse solution spaces for TOPTW through adaptive mechanisms and innovative neighborhood structures. The tests carried out on benchmark instances demonstrate that ALNS performs poorly compared to other approaches such as Ant Colony Optimization (ACO), Iterated Local Search (ILS), Variable Neighborhood Search (VNS), Fast Simulated Annealing (FSA) and Slow Simulated Annealing (SSA). However, further improvements could be made by incorporating additional operators that focus on score maximization, offering pathways for future research and optimization.

**Keywords:** Team orienteering problem, TOPTW, Route optimization, Time windows, Metaheuristics, Adaptive large neighborhood search, Ant colony optimization, Simulated annealing, Iterated local search, Variable neighborhood search.

**Résumé**

Le problème d'orientation par équipe avec fenêtres de temps (Team Orienteering Problem with Time Windows,TOPTW) est une extension complexe du problème d'orientation par équipe (Team Orienteering Problem,TOP) qui consiste à trouver des itinéraires optimaux pour des équipes ou des véhicules afin de maximiser le score total des lieux visités, tout en respectant des fenêtres de temps spécifiques. Ce problème est hautement pertinent dans des domaines tels que la logistique, le tourisme et la réponse aux catastrophes, où la planification d'itinéraires doit prendre en compte des contraintes temporelles. En raison de sa difficulté computationnelle, la résolution du TOPTW nécessite souvent des algorithmes métaheuristiques avancés. Cette thèse se concentre sur l'algorithme de recherche adaptative à grands voisinages (ALNS), qui est conçu pour explorer efficacement des espaces de solutions variés pour le TOPTW grâce à des mécanismes adaptatifs et des structures de voisinage innovantes. Les tests réalisés sur des instances de référence démontrent que l'ALNS a des performances inférieures par rapport à d'autres approches telles que l'Optimisation par Colonies de Fourmis (ACO), la Recherche Locale Itérée (ILS), la Recherche à Voisinage Variable (VNS), le Recuit Simulé Rapide (FSA) et le Recuit Simulé Lent (SSA). Toutefois, des améliorations supplémentaires pourraient être apportées en intégrant des opérateurs supplémentaires axés sur la maximisation des scores, offrant des pistes pour des recherches et des optimisations futures.

**Mots-clés :** Problème d'orientation par équipe, TOPTW, Optimisation d'itinéraires, Fenêtres de temps, Métaheuristiques, Recherche adaptative à grands voisinages, Optimisation par colonies de fourmis, Recuit simulé, Recherche locale itérée, Recherche à Voisinage Variable.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF ALGORITHMS

# LIST OF ACRONYMS

| | |
|---|---|
| **2-OPT** | Two-Optimization |
| **ACO** | Ant Colony Optimization |
| **ALNS** | Adaptive Large Neighborhood Search |
| **BKS** | Best Known Solution |
| **FSA** | Fast Simulated Annealing |
| **GA** | Genetic Algorithm |
| **ILS** | Iterated Local Search |
| **NP-hard** | Non-deterministic Polynomial-time Hardness |
| **OOP** | Object-Oriented Programming |
| **OP** | Orienteering Problem |
| **OPTW** | Orienteering Problem with Time Windows |
| **PDPTW** | Pickup and Delivery Problem with Time Windows |
| **PSO** | Particle Swarm Optimization |
| **RoRo** | Roll-on/Roll-off |
| **SA** | Simulated Annealing |
| **SAILS** | Simulated Annealing with Iterated Local Search |
| **SSA** | Slow Simulated Annealing |
| **TOP** | Team Orienteering Problem |
| **TOPTW** | Team Orienteering Problem with Time Windows |
| **TSP** | Traveling Salesman Problem |
| **VNS** | Variable Neighborhood Search |
| **VRP** | Vehicle Routing Problem |
| **VRPTW** | Vehicle Routing Problem with Time Windows |
| **VS Code** | Visual Studio Code |

# GENERAL INTRODUCTION

Team orienteering problem TOP is one of the commonly faced real-life problems, and it has a wide range of applications such as logistics, transportation, tourism, and disaster relief management. The core of this problem is finding optimal routes for multiple teams while maximizing collected scores from visited locations. TOP gets even more complex as we add the time constraint where every city must be visited in a specific time frame, and it is called team orienteering problem with time windows TOPTW.

Solving the Team Orienteering Problem with Time Windows (TOPTW) is crucial for improving operational efficiency, reducing costs, and maximizing gains. Due to its NP-hard nature, heuristic algorithms are preferred for finding high-quality solutions, especially in large-scale instances where exact algorithms struggle.

The Adaptive Large Neighborhood Search (ALNS) algorithm is a metaheuristic that adjusts its search operators based on performance, efficiently exploring and exploiting the solution space. It employs adaptive strategies like destroy and repair mechanisms to guide the search process toward optimal solutions, making it particularly effective for solving complex routing problems like TOPTW.

The goal of this study is to solve the Team Orienteering Problem with Time Windows (TOPTW) using the ALNS algorithm. By leveraging the ALNS algorithm's adaptive exploration and exploitation capabilities, our objective is to develop a robust solution strategy capable of effectively navigating the complex TOPTW search space. The performance of the proposed method, along with potential improvements over existing approaches, will be evaluated by comparing it to benchmark instances that have been previously established.

In our thesis, Chapter 1 introduces and describes the Team Orienteering Problem with Time Windows (TOPTW), providing the necessary background and context. Chapter 2 explores the Adaptive Large Neighborhood Search (ALNS) algorithm, detailing its mechanisms and explaining why it is chosen as the solution approach for TOPTW. Chapter 3 focuses on presenting the results obtained from applying ALNS, with a comparative analysis against existing methods to demonstrate its effectiveness.

# CHAPTER 1

# TEAM ORIENTEERING PROBLEM WITH TIME WINDOWS (TOPTW)

## 1.1 Introduction

Within the domain of combinatorial optimization and operations research, the Team Orienteering problem with Time Windows (TOPTW) is a challenging and intriguing topic. It expands upon the traditional Orienteering Problem (OP) by adding team-based decision-making and time window constraints, which makes it extremely applicable to a wide range of real-world scenarios.

In the Orienteering Problem (OP), a single vehicle must visit a subset of sites from a list of possible sites in order to maximize the overall score accrued in a predetermined amount of time.The Orienteering Problem with Time Windows (OPTW) and Team Orienteering Problem (TOP) are two more complex variations of this fundamental problem that build upon it.

Expanding on the OP, the Team Orienteering Problem (TOP) has numerous vehicles, each of which has to maximize the overall score while staying within its own time budget. In situations where cooperative efforts are needed to maximize results, like in field services and logistics, this team-based approach is essential.

The Orienteering Problem with Time Windows (OPTW) adds another level of difficulty by imposing time limits on visiting each place. For applications like delivery services and tourism planning, where speed is crucial, the challenge becomes more realistic and complex because each site must be visited within a given time frame.

When these two variations are combined, the Team Orienteering Problem with Time Windows (TOPTW) demands the use of numerous vehicles in order to optimize the overall score while respecting the time budgets and time windows that are unique to each location. Because of this, TOPTW is among the most complex and difficult issues in its field, with important applications.

In this chapter, we first explore the orienteering problem and some of its variations. Then, we give a description and mathematical formulation of the team orienteering problem with time windows (TOPTW). We will also investigate several approaches to solving the TOPTW and talk about its real-world applications, emphasizing its significance and

adaptability in real-world contexts.

## 1.2 Orienteering Problem (OP)

The Orienteering Problem (OP) draws inspiration from the sport of orienteering [11]. Shown in the following figure 1.1. In orienteering, competitors navigate a course, visiting checkpoints starting from the depot within a time limit to maximize their score. Similarly, the OP seeks a path that visits a subset of locations (vertices) while maximizing the total score collected, all within a limited travel distance or time. Finding the optimal solution becomes computationally expensive for large problems, making it NP-hard [64]. This means that for large datasets, finding the absolute best solution can take a very long time. Despite this challenge, the OP's versatility allows it to model real-world problems in logistics, tourism, and other fields [2]. The Orienteering Problem involves finding a path that maximizes the total score collected by visiting a subset of given locations within a limited travel distance or time. Each location has a score, and the objective is to maximize the sum of the scores of the visited locations. This problem combines elements of the Knapsack Problem and the Traveling Salesman Problem (TSP). Unlike TSP, where the goal is to minimize travel distance or time by visiting all locations, the OP focuses on selecting a subset of locations that yield the highest score within the given constraints [64].

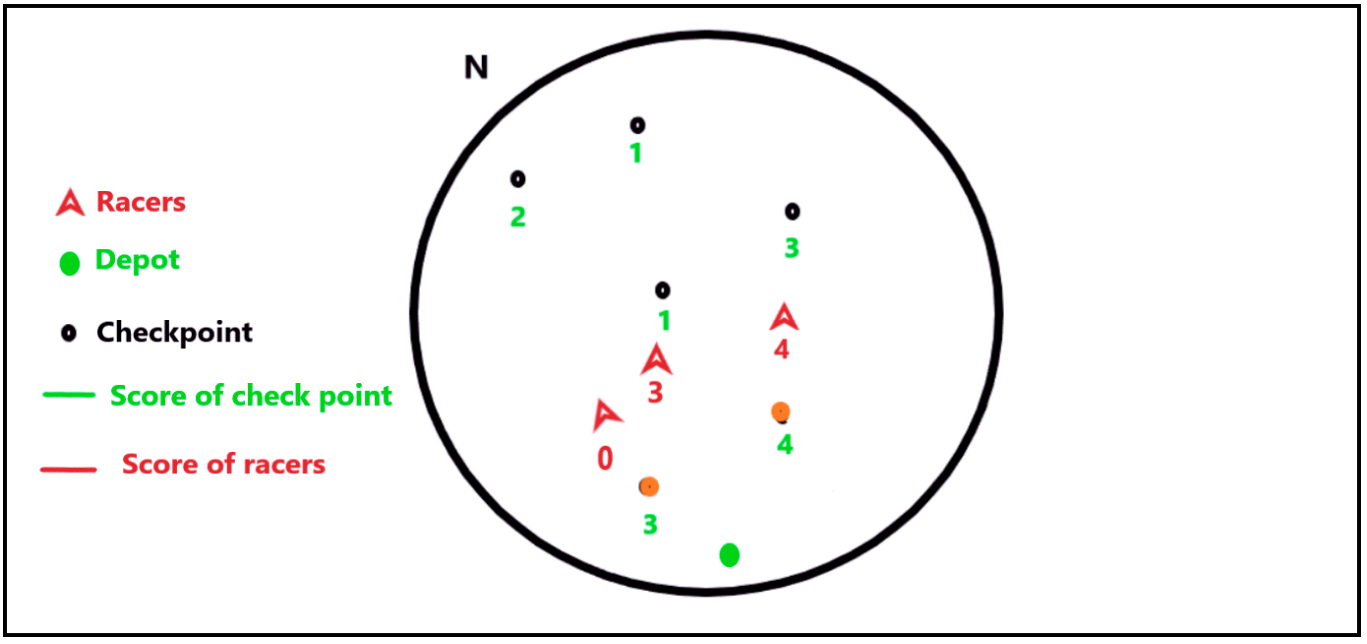


Figure 1.1: Orienteering Problem: Regions, Profits, and Contestant Movements

### 1.2.1 Team Orienteering Problem (TOP)

The Team Orienteering Problem extends the OP to multiple paths. The goal is to maximize the total score collected by a team of paths, each starting and ending at specified points, while ensuring that the combined travel distance or time of all paths does not exceed a given limit.The TOP is derived from the OP by involving multiple routes instead

of a single route. This variant is particularly useful for applications like routing multiple vehicles or teams that need to collect the highest possible score collectively [64].

### 1.2.2 Orienteering Problem with Time Windows (OPTW)

The Orienteering Problem with Time Windows introduces specific time intervals during which each location can be visited. The objective remains to maximize the collected score, but now the visits to the locations must also comply with these time constraints. This variant builds upon the OP by adding the complexity of time windows. Each location is associated with a time window, and the path must be planned such that the locations are visited within their respective time windows, in addition to adhering to the overall travel constraints [64].

### 1.2.3 Team Orienteering Problem with Time Windows (TOPTW)

The Team Orienteering Problem with Time Windows combines elements from both the TOP and OPTW. It involves planning multiple paths that collectively maximize the total score while respecting the travel limits and the time windows for each location. The TOPTW is derived from the combination of TOP and OPTW, incorporating the multi-path structure of TOP and the time window constraints of OPTW. This variant addresses scenarios where multiple teams or vehicles must operate within specific time windows to collect the maximum score [64].

## 1.3 Description of TOPTW

Let us consider a set of visiting locations $N = \{1, 2, \ldots, n\}$ plus a depot indexed by 0. For model formulation, let us define a copy of the depot indexed by $n + 1$.

The Team Orienteering Problem with Time Windows (TOPTW) is a vehicle routing problem set on a network $G = (N \cup \{0, n + 1\}, A)$ with $n + 2$ distinct vertices, where $N = \{1, 2, \ldots, n\}$ represents customer locations, and $A$ denotes arcs connecting these locations. Each arc $(i, j) \in A$ is associated with a non negative travel time $t_{ij}$.

A fixed number of vehicles $m$ are available, each starting at depot 0 and ending at depot $n + 1$ within a specified time window $[O_0, C_0] = [O_{n+1}, C_{n+1}]$. The total time for traveling and visiting time within route $m$ cannot exceed the limited time $T_{\max}$. Each location can be visited at most once.

Each customer $i \in N$ has a given score $S_i$, service time $T_i$, and a time window $[O_i, C_i]$ during which service can be provided. A successful service to a customer requires starting within this time window; arriving early requires waiting until the window opens. the total time for traveling and visiting time within route $m$ cannot exceed the limited time $T_{\max}$. Each customer location can be visited at most once. Due to limited vehicles, not all customers must be served in the solution.

The objective is to optimize vehicle routes to serve customers within their respective time windows, starting at depot 0 and ending at the dummy depot $n + 1$, to achieve the highest possible score [37].

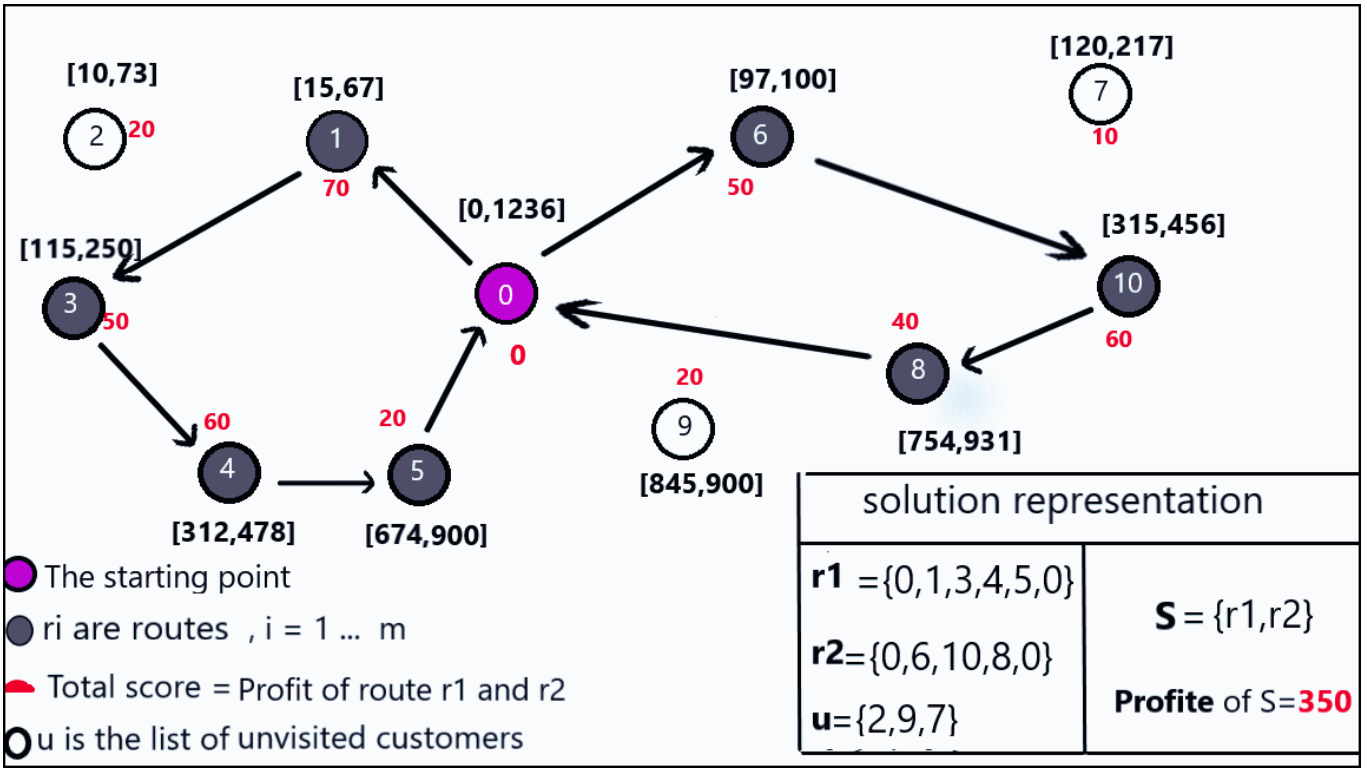


Figure 1.2: Example of representing the problem and the solution for TOPTW

In Figure 1.2, a solution $S$ to the TOPTW problem is depicted using a graphical representation to illustrate the nodes (locations) and the links between them.

**Explanation of symbols and elements in Figure 1.2:**

- Nodes: Represent the locations. The numbers inside the nodes are the node identifiers.
- Time Windows: The numbers within brackets next to the nodes indicate the time windows during which the nodes can be visited.
- Profits: The red number next to each node represents the profit (score) earned when visiting each location.
- Paths: The arrows indicate the possible paths between the nodes.

**Solution Explanation:**
Each solution $S$ consists of a set of paths (routes).

$$S = [R_1, R_2, \ldots, R_m] : \quad \text{Set of paths starting from node "0" and ending at "0"}$$

**Selected Path:** A specific path has been chosen, illustrated by the arrows connecting different nodes. The black nodes indicate the visited nodes, while the white nodes indicate the unvisited nodes.

## 1.4 Mathematical model of TOPTW

The Team Orienteering Problem with Time Windows (TOPTW) can be formulated using the following integer programming model. The binary decision variable $x_{ijd}$ is 1 if the route $d$ includes a visit to location $i$ followed by a visit to location $j$, and 0 otherwise. The binary decision variable $y_{id}$ is 1 if location $i$ is included in a route $d$ and 0 otherwise. The decision variable $s_{id}$ denotes the start time of the service at location $i$ in route $d$, and $M$ is a large constant [27].

$$\text{Max} \sum_{d=1}^{m} \sum_{i=1}^{n} S_i y_{id} \tag{1.1}$$

$$\sum_{d=1}^{m} \sum_{j=1}^{n} x_{0jd} = \sum_{d=1}^{m} \sum_{i=1}^{n} x_{i(n+1)d} = m \tag{1.2}$$

$$\sum_{i=0}^{n} x_{ikd} = \sum_{j=1}^{n+1} x_{kjd} = y_{kd} \quad (k = 1, \ldots, n; d = 1, \ldots, m) \tag{1.3}$$

$$s_{id} + T_i + t_{ij} - s_{jd} \leq M(1 - x_{ijd}) \quad (i = 0, \ldots, n+1; d = 1, \ldots, m) \tag{1.4}$$

$$\sum_{d=1}^{m} y_{kd} \leq 1 \quad (k = 1, \ldots, n) \tag{1.5}$$

$$\sum_{i=0}^{n} \left( T_i y_{id} + \sum_{j=1}^{n+1} t_{ij} x_{ijd} \right) \leq T_{\max} \quad (d = 1, \ldots, m) \tag{1.6}$$

$$O_i \leq s_{id} \quad (i = 0, \ldots, n+1; d = 1, \ldots, m) \tag{1.7}$$

$$s_{id} \leq C_i \quad (i = 0, \ldots, n+1; d = 1, \ldots, m) \tag{1.8}$$

$$x_{ijd}, y_{id} \in \{0, 1\} \quad (i, j = 0, \ldots, n+1; d = 1, \ldots, m) \tag{1.9}$$

Objective (1.1) maximizes the total score that is collected. Equation (1.2) guarantees that a route that starts at destination 0 will be ended at location $n + 1$. Equation (1.3) and (1.4) determine the relationship and timeline of each tour. Equation (1.5) assures that each destination will be visited once. Equation (1.6) explains that each route will finish in limited time. Equations (1.7) and (1.8) guarantee every visit will start during operational hours. Equation (1.9) prevents same route planning.

## 1.5 Solution methods for TOPTW

The NP-hardness of TOPTW requires the use of approximation algorithms or heuristic approaches to find good (but potentially not guaranteed optimal) solutions within a reasonable time frame.

Here are some common solution methods for TOPTW.

### 1.5.1 Exact algorithms

Exact algorithms attempt to find the optimal solution by exploring all possible combinations systematically. However, for even moderately sized TOPTW instances, exact algorithms become computationally intractable due to the exponential growth in the number of potential solutions.

**Branch and Price**:
Branch-and-Price is an advanced algorithm for solving integer programming problems. It combines the Branch-and-Bound algorithm with pricing techniques used in the simplex method. The algorithm begins by decomposing the original problem into smaller subproblems, called nodes. Each node is then solved using a pricing algorithm to optimize partial solutions. The pricing algorithm focuses on generating new columns that add partial solutions, which can improve the overall solution. This method is particularly effective for solving routing and scheduling problems that involve a large number of variables and constraints [8].

To apply the Branch-and-Price algorithm to the Team Orienteering Problem with Time Windows (TOPTW), the process involves several steps. First, a mathematical model based on set packing is formulated to represent the problem, defining binary variables to indicate whether each point is visited within the time limits. The problem is then divided into smaller subproblems through branching on these binary variables. Each subproblem is solved using a pricing algorithm, which may include bidirectional dynamic programming and state space relaxation with a two-phase dominance rule to generate new columns that improve the partial solutions. The partial solutions are further optimized by adding new columns to the original problem and using cutting planes to eliminate suboptimal solutions. The results obtained from this approach are compared with previous algorithms, showing improvements in solving a larger number of test instances to optimality and identifying new solutions for previously unsolved cases [10].

### 1.5.2 Metaheuristic algorithms

Unlike Exact algorithms Metaheuristic techniques are designed to find good solutions to complex optimization problems by exploring and exploiting the search space intelligently. They do not guarantee the optimal solution but can provide near-optimal solutions within a reasonable time frame, making them suitable for large-scale TOPTW instances

**Ant Colony Optimization (ACO)**:
The Ant Colony Optimization (ACO) system is a metaheuristic algorithm inspired by the natural behavior of ants. This algorithm comprises a group of artificial ants that mimic the behavior of real ants in searching for food sources. Each ant starts from a designated starting point and navigates through different locations, leaving pheromone trails along its path. The ant follows a specific path based on a certain probability determined by the amount of pheromone. These pheromone trails are updated based on the overall performance of the ants, with an increase in pheromone on better-performing paths and maintenance of low pheromone levels on poorer paths [53].

To apply Ant Colony Optimization (ACO) to the Team Orienteering Problem with Time Windows (TOPTW), the algorithm employs a colony of artificial ants to construct solutions iteratively. Each ant builds a feasible route by probabilistically selecting the next node based on pheromone trails and heuristic information, balancing exploration and exploitation. The pheromone levels, representing the learned desirability of choosing specific paths, are updated based on the quality of the solutions found. The algorithm also integrates local search procedures to enhance the constructed solutions by exploring their neighborhoods, further optimizing the total collected profit while adhering to the time windows and path constraints [43].

**Iterated Local Search (ILS)**:
The idea behind the Iterated Local Search (ILS) algorithm is to generate neighboring solutions to the initial solution by inserting new customers into the route or removing existing customers from the route. This process is repeated iteratively, gradually improving the solutions until an acceptable or optimal solution is reached [53].

ILS algorithm is an effective algorithm applied to TOPTW, aiming to find high-quality solutions iteratively. Initially, an initial solution is generated, often using a constructive algorithm. Neighboring solutions are then created by making small modifications, such as inserting or removing customers from the route. Each solution is evaluated based on criteria like total delay and compliance with time windows. The best solution is selected, and the process iterates until an acceptable or optimal solution is found. This iterative refinement approach enables ILS to effectively tackle the challenges posed by TOPTW, leading to improved solutions over time [27].

**Simulated Annealing (SA):**
Simulated Annealing (SA) is an optimization algorithm inspired by the annealing process in metallurgy, where materials are heated and then slowly cooled to reach a low-energy state. It works by accepting worse solutions with a probability that decreases over time, helping to escape local optima. The probability of accepting worse solutions is governed by a temperature parameter, which gradually decreases [36].

### 1.5.3 Hybrid Approach

Hybrid approaches combine elements from different methods to leverage their strengths. For example, an exact algorithm might be used to find an initial good solution, followed by a metaheuristic to further refine it.

**SAILS:**
In the SAILS algorithm, the initial solution is generated using Iterated Local Search (ILS), which improves the solution through local moves. Simulated Annealing (SA) is then integrated to avoid getting stuck in local optima by probabilistically accepting worse solutions based on a decreasing temperature. After each local improvement, the SA step ensures thorough exploration of the solution space by allowing occasional acceptance of suboptimal moves [22, 39].

SAILS algorithm applied to the Team Orienteering Problem with Time Windows (TOPTW), a combination of Iterated Local Search (ILS) and Simulated Annealing (SA) was used to effectively improve solutions. The algorithm starts by creating an initial solution using a greedy construction method, and this solution is improved using Iterated Local Search, which performs local improvement moves such as shift, insertion, and swap. Local search operations like SWAP and 2-OPT are applied to gradually improve the solutions. Simulated Annealing is integrated to prevent the algorithm from getting trapped in local optima, where after each local improvement using ILS, negative improvements are accepted with a certain probability depending on a probabilistic function that relates the change in solution value to the current temperature, which decreases gradually over time, reducing the chance of accepting worse solutions as the process progresses. The algorithm employs the Simulated Annealing strategy after each local improvement to ensure a thorough exploration of the solution space, and if no improvement is found after a certain number of iterations, the focus is shifted to the best solution found so far, using it as a new starting point. Through this combination, the algorithm effectively improved solutions for the TOPTW problem, benefiting from the strength of Local Search in improving solutions and utilizing Simulated Annealing for comprehensive search and avoidance of local optima [22].

## 1.6 Applications of TOPTW

TOPTW, serves as a versatile modeling framework for addressing various optimization Real-World Optimization Challenges.

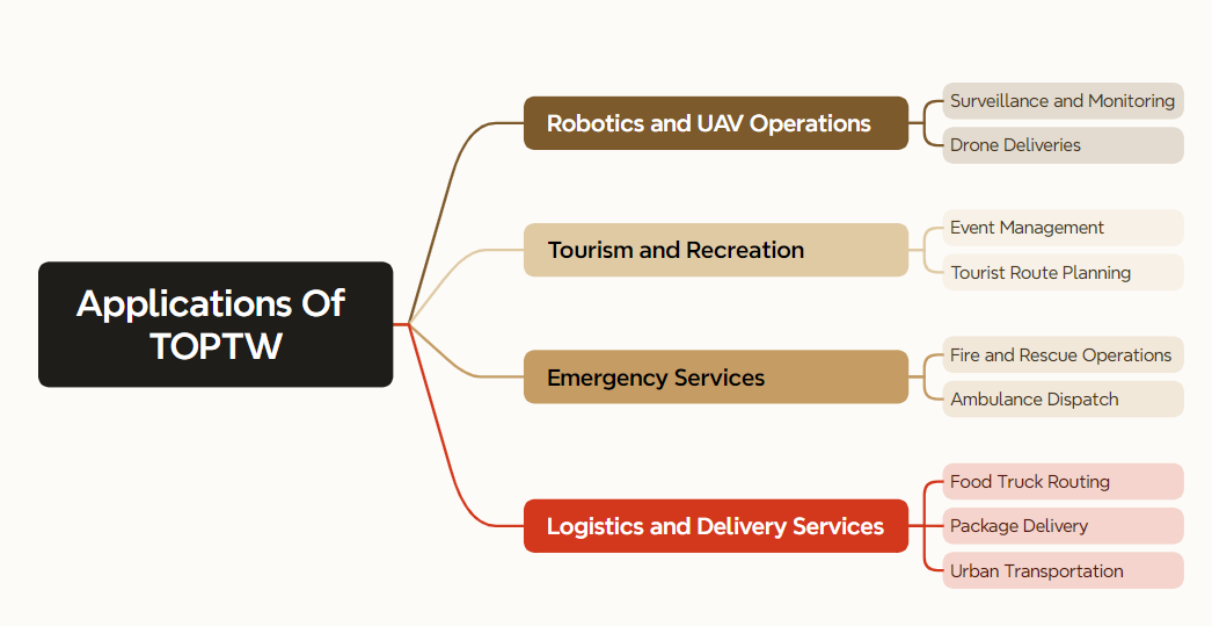


Figure 1.3: Application of TOPTW

### 1.6.1 Logistics and Delivery Services

**Food Truck Routing**: TOPTW is particularly useful in optimizing routes for food trucks, ensuring they can serve customers efficiently while maximizing profit within time constraints. This application helps in minimizing travel time and maximizing customer satisfaction by ensuring timely service [69].
**Package Delivery:** In logistics, companies utilize TOPTW to enhance delivery efficiency. By optimizing routes for delivery vehicles, businesses can ensure that packages are delivered within specified time windows, thus improving service reliability and customer satisfaction [30].

### 1.6.2 Emergency Services

**Ambulance Dispatch**: The principles of TOPTW can be applied to emergency response scenarios, where ambulances must reach patients within critical time frames. Optimizing routes can significantly reduce response times, potentially saving lives [45].
**Fire and Rescue Operations:** Similar to ambulance services, fire departments can use TOPTW to determine the most efficient routes to reach emergency sites, ensuring timely intervention during critical situations [45].

### 1.6.3 Tourism and Recreation

**Tourist Route Planning:** TOPTW can be applied in the tourism sector to design optimal routes for guided tours. By considering time windows at various attractions, tour operators can enhance visitor experiences by ensuring that groups visit popular sites efficiently [69].

### 1.6.4 Robotics

**Drone Deliveries:** With the rise of drone technology, TOPTW is increasingly relevant in optimizing drone delivery routes. Drones can be programmed to visit multiple locations while adhering to time constraints, enhancing delivery speed and efficiency [29].

TOPTW's applications reach far beyond conventional trip planning, permeating essential sectors such as logistics, humanitarian endeavors, and maintenance operations. These applications offer invaluable solutions to intricate real-world optimization challenges, revolutionizing how we approach resource allocation, efficiency enhancement, and problem-solving across diverse domains [25, 65].

## 1.7 Conclusion

In this chapter, we explored the fundamentals of the Team Orienteering Problem with Time Windows (TOPTW), beginning with its roots in the Orienteering Problem (OP) and its variations. We examined the mathematical modeling of TOPTW and reviewed different solution methods, including exact algorithms, metaheuristics, and hybrid approaches. Additionally, we discussed the wide-ranging applications of TOPTW in areas such as

logistics, emergency services, tourism, and robotics, emphasizing its importance in solving real-world problems. In the next chapter, we will focus on solving TOPTW using the Adaptive Large Neighborhood Search (ALNS) algorithm.

# CHAPTER 2

# SOLVING TOPTW USING ADAPTIVE LARGE NEIGHBORHOOD SEARCH (ALNS)

## 2.1 Introduction

Metaheuristic optimization techniques have become highly popular over the past two decades. Among these techniques, algorithms such as Genetic Algorithm (GA), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), and local search have become well-known not only among computer scientists but also among researchers from various fields. These techniques have been applied in diverse areas of study, raising the question of why metaheuristics are so widespread. The answer to this question can be summarized in four main reasons: simplicity, flexibility, a derivation-free mechanism, and avoidance of local optima [9].

In this chapter, we will discuss the approach we followed to solve the team orienteering problem with time windows and the heuristic used, the Adaptive Large Neighborhood Search (ALNS).

## 2.2 Definition

Adaptive Large Neighborhood Search (ALNS) is a powerful metaheuristic optimization method intended for use in solving challenging combinatorial optimization issues by selecting neighborhoods in an adaptable manner according to how well they performed throughout the search. ALNS employs a series of heuristics for each stage of its iterative destruction and repair process. The adaptive strategy of the ALNS algorithm and its ability to balance between exploration and exploitation boost its efficacy and efficiency and help to avoid local optima. It is very useful for complex and extensive optimization problems.

## 2.3 Key components of ALNS

ALNS begins with an initial solution and then iterates until a stopping criterion is met. In each iteration, a destroy and repair operator are selected, which transform the current

solution into a candidate solution. This candidate solution is then evaluated by an acceptance criterion, and the operator selection scheme is updated based on the evaluation outcome. Once the stopping criterion is met, ALNS returns the best solution it has found.

### 2.3.1 Initial solution

The initial solution marks the starting point of ALNS and serves as the basis from which the algorithm starts its exploration. The Initial Solution can be generated using different methods such as heuristics, randomization or even another optimization algorithm however the quality of the initial solution can play a crucial role in improving the final result of the ALNS algorithm [46].

### 2.3.2 Neighborhood structures

The core of ALNS is the use of destroy and repair operators, which define the neighborhood structures explored during the search. Destroy operators remove parts of the current solution, creating a partial solution. Repair operators then reinsert the removed elements, generating a new complete solution. The choice of effective destroy and repair operators is crucial for the performance of ALNS [51].

### 2.3.3 Adaptive mechanism

ALNS is adaptive, meaning that it learns which destroy and repair operators are more effective during the search and adjusts their selection probabilities accordingly. This is done by maintaining a score for each operator, which is updated based on the quality of the solutions produced by that operator. Operators that generate better solutions are selected more often in subsequent iterations [1].

### 2.3.4 Acceptance criteria

ALNS uses an acceptance criterion to decide whether to accept or reject a new candidate solution. Common acceptance criteria include simulated annealing, greedy acceptance, and threshold acceptance. The acceptance criterion controls the balance between exploration and exploitation during the search [1].

## 2.4 Applications of ALNS

The ALNS method has been effectively applied to a wide range of complex optimization problems across various industries. Here are some key applications:

**Transportation and Logistics**: ALNS has been widely used to solve vehicle routing problems (VRP), including the PDPTW and VRPTW. These applications aim to optimize routes for fleets of vehicles to minimize costs, time, or environmental impact [21]

**Maritime and Shipping**: In maritime logistics, ALNS has been applied to the stowage planning problem for Roll-on/Roll-off (RoRo) ships, where the goal is to efficiently load and unload vehicles on ships while minimizing the shifting costs during port calls [26]

**Manufacturing and Scheduling**: ALNS has been used to solve various scheduling problems in manufacturing, such as nurse scheduling, steel manufacturing operations, and high school timetabling. These problems involve assigning tasks or shifts to resources in a way that meets constraints and optimizes certain objectives, such as minimizing costs or maximizing efficiency [21]

**Supply Chain Management**: In supply chain optimization, ALNS has been utilized to design supply chain networks, including the placement of warehouses and distribution centers to optimize logistics costs and service levels [21].

## 2.5 ALNS Algorithm

**Algorithm 1** ALNS

```
Input: S_init : The initial solution.
D : The set of destruction operators.
R : The set of repair operators.
q : Number of locations to remove.
u : List of removed locations.
ext : List of unvisited locations.
max_iteration : The maximum number of iterations.
Segment : The number of iterations after which weights are modified.
T : Initial temperature.
Output: S_best
Begin
    Generate an initial solution S_init
    S_best ← S_init
    S ← S_init
    Initialize temperature T
    Initialize scores, weights, and probabilities for the operators in D
    Initialize scores, weights, and probabilities for the operators in R
    Nbr_iteration ← 0
    while Nbr_iteration ≤ max_iteration do
        S' ← S
        Select a random number q
        Choose a destruction operator d using the roulette wheel
        u ← d(S', q)
        u ← u ∪ ext
        Increment the usage count of operator d
        Choose a repair operator r using the roulette wheel
        Apply r to S' with u
        Increment the usage count of operator r
        if f(S') > f(S_best) then
            S_best ← S'
            S ← S'
            Update the scores of operators d and r
        else if f(S') > f(S) then
            S ← S'
            Update the scores of operators d and r
        else if S' meets acceptance criteria then
            S ← S'
            Update the scores of operators d and r
        end
    end
    Increment Nbr_iteration
    if Nbr_iteration MOD Segment = 0 then
        Update the weights and probabilities of operators d and r
        Reset the scores and usage counts of operators d and r
    end
end
    Update the temperature T
    Return S_best
End
```

In the first step, an initial feasible solution is constructed.

The ALNS method is based on two kinds operators: destruction and repair, which improve the initial solution. First, we initialize the scores, weights, and probabilities of the operators $D$ and $R$ as well as the initial temperature $T_0$, which depends on the initial solution.

In each iteration, the destruction and repair operators $d$ and $r$ are chosen using the concept of a roulette wheel based on their performance (measured by scores and weights) in previous rounds.

Three destruction operators are proposed to select the tasks to remove from the current solution: Random removal, Worst removal, and Unrandomized worst removal. The cities are first removed from $S'$ using the selected destruction operator $d$ and added to the collection of externalized cities. Then, they are reinserted into the routes using the appropriate repair operator $r$. The repair operator uses four strategies: the best insertion, the second-best insertion, random insertion at the end route and random insertion. the usage count of the operator $r$ is incremented.

If the resulting solution $S'$ is better than the best solution identified so far, it replaces $S_{best}$(line 29). If the resulting solution $S'$ is not better than $S_{best}$ but better than $S$, it replaces $S$(line 33). We can accept a solution $S'$, which is slightly worse than $S$, provided it meets certain acceptance criteria (we use simulated annealing), and it then replaces $S$ (line 36).

In all three cases, we update the scores of the operators. In the first case, we add a larger number to the score compared to the others (line 32). In the other cases, we add a smaller number to the score (lines 35 and 38).

Then, at each segment (a certain number of iterations), the scores and weights of the operators $d$ and $r$ are adjusted based on the quality of the resulting solution, and the search moves to the next iteration. This operation is repeated until a stopping condition is met (we set a specific number of iterations). We take the best result, which is the value of $S_{best}$.

## 2.6 Destruct and repair operators

Destruction and repair are the fundamental mechanisms in the ALNS algorithm since they work synergistically to explore the solution space effectively and escape local optima. The destruction phase involves partially dismantling the current solution and removing a subset of its components to create an incomplete intermediate state. This process introduces diversity into the search, enabling the algorithm to explore new regions of the solution space that might be inaccessible from the current configuration. Conversely, the repair phase focuses on reconstructing a feasible solution from this intermediate state by reintroducing the removed components in a strategic manner. This reconstruction process ensure that the rebuilt solution is not only feasible but also potentially closer to the optimal solution.

### 2.6.1 Destruction operators

In our study we used three types of destruction operators, Random removal, worst removal with random selection and worst removal.

#### 2.6.1.1 Random removal

Random removal is the simplest destruction operator in optimization algorithms. This operator randomly selects and removes locations from the solution. This operation is repeated until all q locations have been removed. This operator is crucial in the ALNS algorithm because it can remove any element of the solution regardless of the cost, creating essential diversification during the search of a large solution space. By introducing randomness, it helps prevent the algorithm from becoming stuck in local optima, allowing for broader exploration.

**Algorithm 2** Random removal

**Input:** `solution` : The current solution.
`q` : The number of locations to remove.
**Output:** The updated solution after removal (`solution`), and the removed locations (`removed_locations`).

**Begin**
Create a list of locations $L$ from the routes in the solution
Initialize `removed_locations` as an empty list
**while** *q > 0* **do**
    Select a random location from $L$
    Remove the selected location from $L$
    Add the selected location to `removed_locations`
    `q` $\leftarrow$ `q` $-1$
Update the routes by removing the removed locations
**return** `solution` and `removed_locations`

#### 2.6.1.2 Worst removal with random selection

Worst removal with random selection is a destruction operator used in optimization algorithms, it targets the removal of locations based on their performance but incorporates a random selection element. Specifically, it removes a number of locations (q) from the solution by first sorting the locations according to a specific attribute that signifies poor performance. Then, it randomly selects locations from this sorted list to remove, adding them to a list of removed locations. This process is repeated until all q locations are removed. This operator is crucial in the ALNS algorithm as it combines performance-based removal with randomness, enhancing diversification and helping the algorithm explore different parts of the solution space, thus avoiding local optima.

**Algorithm 3** Worst removal with random selection

**Input:** `solution` : The current solution.
`q` : The number of locations to remove.
`p` : Float
**Output:** `solution` : The updated solution after removal.
`removed_locations` : The removed locations.
**Begin**
Create a list of locations $L$ from the routes in the solution
Sort $L$ in ascending order by profit
Initialize `removed_locations` as an empty list
**while** *q > 0* **do**
    Generate a random number $y$ between 0 and 1
    Select a location with index $y^p \times |L|$
    Remove the selected location from the corresponding route
    Remove the selected location from $L$
    Add the selected location to `removed_locations`
    q ← q −1
**return** `solution` and `removed_locations`

#### 2.6.1.3 Worst removal

Worst Removal is a more deterministic destruction operator. This operator systematically removes the worst-performing locations from the solution. It starts by making a copy of the initial solution and extracting all locations. These locations are then sorted based on the performance attribute (s). The operator removes the top q worst-performing locations from the sorted list, updating the routes accordingly and adding these locations to a list of removed locations. By focusing on removing the worst locations, this operator intensifies the search around promising regions of the solution space, refining the overall solution quality.

**Algorithm 4** Worst removal

**Input:** `solution` : The current solution.
`q` : The number of locations to remove.
**Output:** `solution` : The updated solution after removal.
`removed_locations` : The removed locations.
**Begin**
Create a list of locations $L$ from the routes in the solution
Sort $L$ in ascending order by profit
Initialize `removed_locations` as an empty list
**while** *q > 0* **do**
    Select the first location in $L$
    Remove the selected location from $L$
    Update the routes by removing the selected location
    Add the selected location to `removed_locations`
    q ← q −1
**return** `solution` and `removed_locations`

### 2.6.2 Repair operators

We used three types of repair operators: *Greedy Repair*, *Repair End*, and *Repair Shortest Path*. These operations rely on a function called `is_valid_path` to verify the validity of a path based on time constraints.

The `is_valid_path` function checks:

- **Total travel time**: It ensures that the total time taken to complete the path is within the allowed maximum time (*tmax*), preventing the vehicle from exceeding its allotted time.
- **Vehicle arrival time**: It confirms that the vehicle arrives within the service time window or earlier, ensuring compliance with the specified schedule for each location.

Using `is_valid_path` in these three repair operations ensures that the selected paths meet the time constraints, enhancing the solution's effectiveness and adherence to time limitations.

#### 2.6.2.1 Greedy repair

This algorithm tries to repair the paths (routes) of the current solution in a greedy manner by inserting the most profitable unvisited locations into the best positions.

- Ensure each path starts and ends at the depot.
- Remove the first location from the unvisited list.
- Sort the unvisited locations by their profit in descending order.
- For each unvisited location:
  - Try to find the best position in the path to insert the location to maximize profit.
  - If a valid position is found, insert the location into the path.
- Calculate and return the total profit of all paths

**Algorithm 5** Greedy Repair

**Input:** `unvisited_list` : List of unvisited locations.
`depot` : The location of the depot.
`Tmax` : The maximum allowed time.
`paths` : List of paths.
**Output:** Updated paths and list of unvisited locations.
**Begin**
**for** *each path in* ***`paths`*** **do**
  **if** *path is empty* **then**
    Initialize with depot at both ends
  **end**
  Ensure depot is at the start and end of the path
  Remove depot from `unvisited_list` if present
**end**
Sort `unvisited_list` by profit in descending order
**for** *each location in* ***`unvisited_list`*** **do**
  **for** *each path in* ***`paths`*** **do**
    Determine the best insertion position for maximum profit
    Insert location at this position
    Remove location from `unvisited_list` Break the loop
  **end**
**end**
**return** the updated `paths` and `unvisited_list`

#### 2.6.2.2 Repair End

This algorithm tries to add unvisited locations to the end of the path randomly while ensuring the path remains valid.

- Ensure the path starts at the depot.
- Randomly select locations from the unvisited list and try to add them to the path.
- For each location in the unvisited list:
  - If adding the location keeps the path valid, remove it from the unvisited list.
  - Stop if all points have been tested without finding a valid insertion.
- Ensure the path ends at the depot.
- Calculate and return the path profit.

**Algorithm 6** Repair End

**Input:** `unvisited_list` : List of unvisited locations.
`depot` : The location of the depot.
`Tmax` : The maximum allowed time.
`path` : A single path.
**Output:** Updated path, list of unvisited locations, and profit.
**Begin**
Initialize the path ensuring it starts at the depot
Randomly select locations from `unvisited_list` and try to add them to the path
**for** *each location* `loc` *in* `unvisited_list` **do**
  **if** *adding the location keeps the path valid* **then**
    Remove it from `unvisited_list`
  **end**
  Stop if all points have been tested without finding a valid insertion
**end**
Ensure that the path ends at the depot
Calculate and return the path profit

#### 2.6.2.3 Repair shortest path

The 'Shortest Path Insertion' algorithm starts by ensuring each path in the list begins and ends at the depot, then removes the depot from the list of unvisited locations. For each location in the unvisited list, the algorithm searches for the optimal insertion point in the existing paths to minimize the increase in travel time. It evaluates each potential insertion point, selecting the one that results in the least additional travel time and maintains the validity of the path within the specified time constraints. Once the best insertion point is found, the location is permanently inserted into the path, and the process continues with the next location. Finally, the algorithm ensures all paths end at the depot and returns the updated paths along with the remaining unvisited locations.

**Algorithm 7** Shortest Path Insertion

**Input:** `unvisited_list` : List of unvisited locations.
`depot` : The location of the depot.
`Tmax` : The maximum allowed time.
`paths` : List of paths.
**Output:** Updated paths, list of unvisited locations.
**Begin**
Ensure each path in `paths` starts and ends at the depot.
Remove the depot from `unvisited_list`.
**for** *location in unvisited_list[:]* **do**
  `best_insertion_index` ← None
  `best_increase` ← $\infty$

  **for** *path in paths* **do**
    **for** *i from 1 to len(path)* **do**
      Calculate `increase` in travel time if `location` is inserted at position `i`.
      **if** *increase* < *best_increase and is_valid_path(path, depot, Tmax)* **then**
        `best_increase` ← `increase`
        `best_insertion_index` ← `i`
      **end**
    **end**
    **if** *best_insertion_index is not None* **then**
      Insert `location` at `best_insertion_index` in `path`.
      Remove `location` from `unvisited_list`.
      **break**
    **end**
  **end**
**end**
Ensure each path in `paths` ends at the depot.
**return** `paths, unvisited_list`

## 2.7 Update temperature

The "Update Temperature" algorithm is a simple yet essential component in optimization techniques like simulated annealing. It takes the current temperature, T, and a cooling coefficient, phi (defaulting to 0.99975), as inputs. The algorithm then updates the temperature by multiplying T by phi, thereby gradually reducing the temperature. This cooling process is critical for controlling the exploration and exploitation balance in optimization, allowing the system to search more broadly at higher temperatures and focus more finely as the temperature decreases. The algorithm returns the new, reduced temperature, facilitating the iterative process of finding an optimal solution by avoiding local optima and converging towards a global optimum [5] [4].

**Algorithm 8** Update temperature

**Input:** `T`: The current temperature.
`phi`: The cooling coefficient (default 0.99975).
**Output:** The new temperature T
**Begin**
`T` $\leftarrow$ `T` $\times$ `phi`
Return T

## 2.8 Selection and update of destruction and repair operators

Following the approach of Ropke and Pisinger [50], we implemented separate scores and weights for each destruction and repair operator. The probability of selecting a specific destruction operator $d$ is proportional to its associated weight $\rho_d$. Let $n_d$ represent the total number of destruction operators. The probability $\phi_d$ of selecting a particular operator $d$ is calculated as follows:

$$\phi_d = \frac{\rho_d}{\sum_{d'=1}^{n_d} \rho_{d'}} \tag{2.1}$$

In each iteration of the ALNS algorithm, one destruction operator is selected using a roulette wheel selection method.

As outlined in [50], the weights are dynamically adjusted during the search process based on the scores obtained. Initially, all operators $d$ are assigned a weight $\rho_d$ of 1, and their scores $\psi_d$ are set to zero. After each iteration, the scores of the employed operators are updated as follows:

$$\psi_d = \begin{cases} \psi_d + \sigma_1 & \text{if it improves the best solution } S_{best}, \\ \psi_d + \sigma_2 & \text{if it improves the current solution } S, \\ \psi_d + \sigma_3 & \text{if it is accepted as the new current solution } S \text{ despite being worse.} \end{cases} \tag{2.2}$$

Here, $\sigma_1$, $\sigma_2$, and $\sigma_3$ are parameters defined by the user.

Every 100 iterations, the weights $\rho_d$ are updated based on the scores accumulated over the last 100 iterations. At the start of each new time segment, the scores $\psi_d$ are reset to zero. The weights are then updated as follows:

$$\rho_d = (1 - \rho_{\text{react}})\, \rho_d + \rho_{\text{react}} \frac{\psi_d}{\max(1, \Theta_d)} \tag{2.3}$$

Where $\Theta_d$ is the number of times operator $d$ was used in the last time segment, and $\rho_{\text{react}}$ controls how quickly the weight adjustment algorithm reacts to changes in the effectiveness of destroy and repair operators.

The same method is applied to calculate the scores, weights, and probabilities for the repair operators $r$. The probability $\phi_r$ of selecting a particular repair operator $r$ is calculated similarly:

$$\phi_r = \frac{\rho_r}{\sum_{r'=1}^{n_r} \rho_{r'}} \tag{2.4}$$

Finally, the weights $\rho_r$ for each repair operator $r$ are updated as follows:

$$\rho_r = (1 - \rho_{\text{react}})\, \rho_r + \rho_{\text{react}} \frac{\psi_r}{\max(1, \Theta_r)} \tag{2.5}$$

Here, $n_r$ denotes the number of repair operators used, and $\Theta_r$ represents the number of times the repair operators were employed in the last time segment.

## 2.9 Create initial solution

**Algorithm 9** Create initial Solution

**Input:** `location_list` : List of locations to visit.
`num_paths` : Number of paths (routes) to generate.
`depot` : The location of the depot.
`Tmax` : The maximum allowed time.
`method` : Repair method ("greedy" or "insert_end").
**Output:** Generated paths, list of unvisited cities, and total profit.
**Begin**
Create a copy of the list of unvisited cities
Initialize a list of empty paths for the specified number of paths
Initialize the total profit to zero
**if** *the method is "greedy"* **then**
  Apply the greedy repair to generate the paths and calculate the profit
**else if** *the method is "insert_end"* **then**
  **for** *each path* **do**
    Apply the random repair to generate the paths and calculate the profit
  Update the list of unvisited cities
Display the total profit of the initial solution
Return the generated paths, the list of unvisited cities, and the total profit

This algorithm generates initial solutions by creating paths using either the greedy repair method or the repair end method.

- Create a copy of the list of unvisited cities.
- Initialize a list of empty paths for the specified number of paths.
- Initialize the total profit to zero.
- If the method is "greedy": Apply the greedy repair to generate the paths and calculate the profit.
- Else if the method is "insert_end":

- For each path, apply the random repair to generate the paths and calculate the profit.
- Update the list of unvisited cities.

- Display the total profit of the initial solution.
- Return the generated paths, the list of unvisited cities, and the total profit.

## 2.10 Acceptance criteria

The ALNS algorithm uses Simulated Annealing (SA) for acceptance criteria:

- **Improvement Acceptance**: Any improved solution ($f(S') > f(S)$) is accepted.
- **Worse Solution Acceptance**: Worse solutions ($f(S') < f(S)$) are accepted with a probability of $e^{(f(S')-f(S))/T}$, where $T$ is the temperature, reduced each iteration by $\phi = 0.99975$.

In the acceptance method, the probability of accepting solutions that are worse than the current one is high, but it gradually decreases. Specifically, for a given fixed cost difference $f(S') - f(S)$, the acceptance probability declines exponentially as $e^{-(f(S')-f(S))/T}$. The initial temperature is determined by the initial solution and is calculated as follows:

$$T = -\frac{\omega_T}{\ln 0.5} f_c(S)$$

$\omega_T = 0.05$ means a solution 5% less profitable than $f_p(S)$ is accepted with a probability of 0.5 [50].

**Algorithm 10** Acceptance criteria

**Input:** `S_prime`: The new solution.
`S`: The current solution.
`T`: The current temperature.
`f`: The profit function.
**Output:** `True` if the new solution is accepted, otherwise `False`.
**Begin**
**if** *f(S_prime) > f(S)* **then**
  return `True`
**end**
**else**
  `delta_f` ← `f(S_prime) − f(S)`
  `acceptance_probability` ← `math.exp(delta_f / T)`
  Generate a random number $rn$ between 0 and 1
  **if** $rn <$ *acceptance_probability* **then**
    return `True`
  **end**
**end**
return `False`

## 2.11 Conclusion

In this chapter, we presented the implementation of the adaptive large neighborhood search (ALNS) method for addressing the team orientation problem with time windows (TOPTW). ALNS is a metaheuristic method that uses a combination of destruction and repair operators to explore the solution space effectively and escape local optima. It used an adaptive mechanism that adjusts operator selection based on performance. The next chapter will focus on the results of applying ALNS and compare their performance to some proposed methods for solving TOPTW.

# CHAPTER 3

# APPLICATION AND RESULTS

## 3.1 Introduction

In the previous chapter, we discussed the theoretical approach we used to solve TOPTW using ALNS. In this chapter, we will delve into the practical implementation of the previously mentioned algorithms, shedding light on the software and environment used to develop our approach. Then, we will discuss instances and compare our final result with other the results of other methods such as Ant Colony Optimization (ACO), Iterated Local Search (ILS), Variable Neighborhood Search (VNS), Fast Simulated Annealing (FSA) and Slow Simulated Annealing (SSA).

## 3.2 Development environment and tools

### 3.2.1 Visual Studio Code

Visual Studio Code (VS Code) is a free, open-source code editor developed by Microsoft. It's widely used by developers due to its versatility, speed, and extensive feature set [3].

**Integrated Git:** VS Code has built-in Git support, allowing developers to perform version control operations such as commits, branching, and merging directly from the editor.

**Debugging:** VS Code includes powerful debugging capabilities, supporting breakpoints, call stacks, and an interactive console for various languages.

**Intuitive Interface:** The editor features a clean and user-friendly interface with a sidebar for file navigation, a main editor window for code, and an integrated terminal [42].

Figure 3.1: VScode

### 3.2.2 Python

Python is our chosen programming language for this project due to its inherent strengths that make it particularly well-suited for algorithm development and scientific computing. As a free and open-source language, Python offers accessibility to a vast community of developers and an extensive repository of libraries. Its high readability and clean visual layout contribute to ease of use, enabling developers to write code that is easy to understand and maintain.

Python's robust memory management and support for object-oriented programming (OOP) further enhance its capabilities in handling complex data structures and algorithms. Additionally, Python's portability allows it to run on various platforms without modification, making it versatile for diverse computing environments. Its high performance in scientific computations and flexibility in accommodating different programming paradigms make it a go-to choice for solving problems like the Team Orienteering Problem with Time Windows (TOPTW).

As displayed in the Figure 3.2, these features collectively underscore Python's suitability for our project.

**Readability and Ease of Use:** Python's clear and concise syntax, combined with its dynamic typing, accelerates the development process and facilitates experimentation. This proves invaluable when adapting algorithms like ALNS to a complex problem domain like TOPTW.

**Built-in Libraries:** Python's standard library offers a wealth of modules that streamline the implementation of our alns algorithm:

**Math:** Provides essential mathematical functions such as square root and trigonometric functions, which are used in fitness calculations.

**Random:** Enables the generation of random numbers, which is crucial for introducing randomness into the search process.

**Copy:** Allows for the copying of objects in Python, whether shallow or deep. Copying is important when creating independent copies of objects to avoid affecting the original copies during modifications or experimental processes.

**Pygame:** We utilized Pygame to design the user interface, enabling an interactive and visually appealing experience for users.

**Data Structures and Manipulation:** Python's built-in data structures (lists, tuples, dictionaries) and list understandings offer flexibility in representing the TOPTW problem data (locations, rewards, time windows) [47].


Figure 3.2: Features of Python [52]

## 3.3 Hardware

All our codes were created in python and executed using visual studio code in the following configuration:

**CPU** : Intel(R) Core (TM) i7-4700HQ CPU @ 2.4GHz, 4 cores, 8 logical processors.

**RAM** : 12 Gb.

**OS** : Microsoft windows 11 Pro 64 bits.

## 3.4 Graphical user interfaces

As part of our project involving the development of graphical user interfaces using the Pygame library, we have created three primary interfaces aimed at facilitating interaction with the ALNS algorithm to solve the Team Orienteering Problem with Time Windows (TOPTW). These interfaces vary in their functions and objectives, each providing a dedicated environment for a specific aspect of the project.

The first interface (Figure3.3) features the project's name at the top. Below, We provide information about the project members, including both the professor and students involved, detailing their names and emails. Additionally, there is a button labeled "simulate" that allows users to jump into the simulation environment where the ALNS algorithm is utilized.

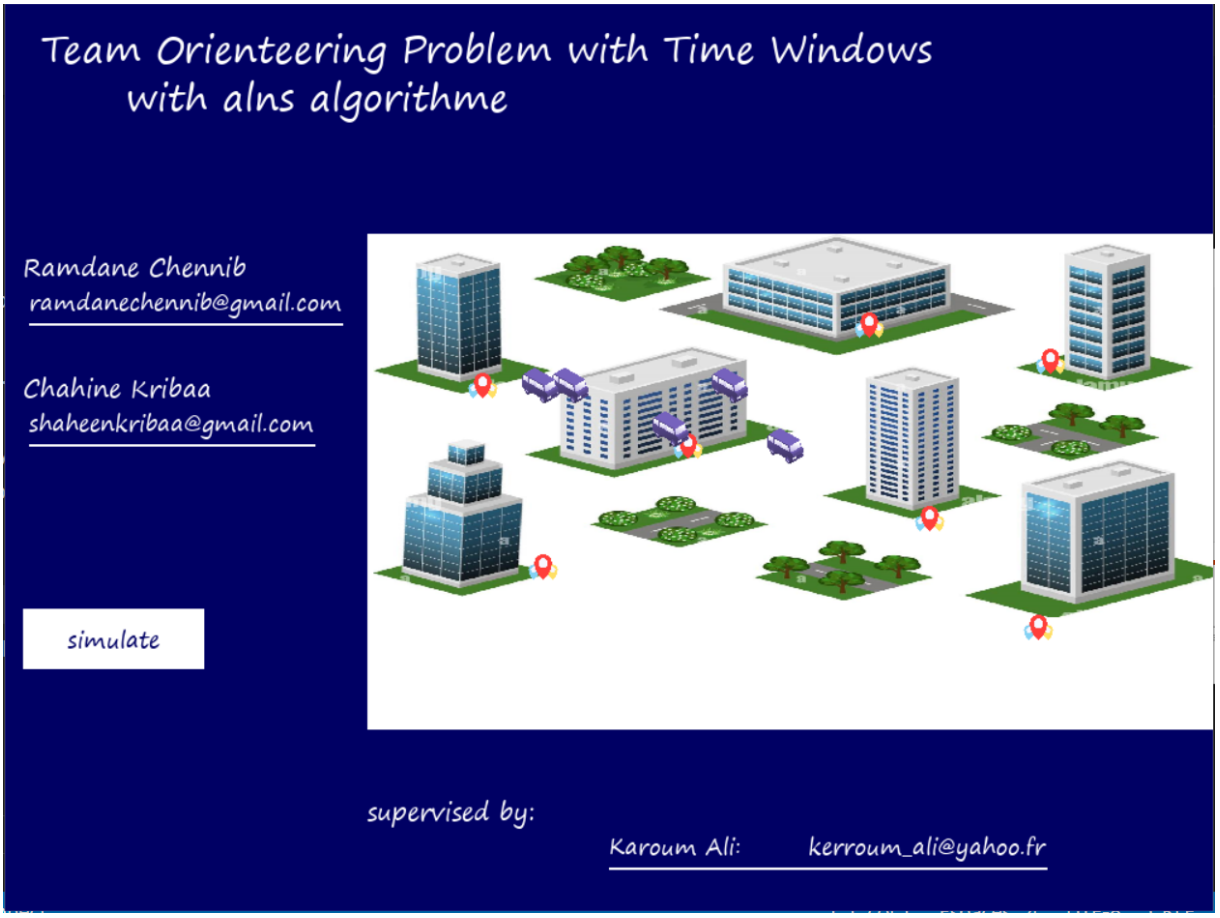


Figure 3.3: Main window

**Home page** (Figure3.4) allows the user to pick the input parameter of our algorithm.
**instances**: the icon enclosed in a red rectangle allows the user to import the instances file from the computer.
**initialization operator**: choosing which method will be used to create the initial solution.
**inputs**:

- Number of Vehicles: This field allows the user to enter the number of vehicles used in the solution, which must be less than or equal to 10.
- Number of segments: This field allows the user to enter the number of segments.
- Number of iterations: This field allows the user to enter the number of iterations, higher number of iterations means higher score but also longer execution.

**Buttons**: Only visible after importing an instance file.
**generate:** it start the operation of creating the initial solution and improving it with ALNS.


Figure 3.4: Optimization Algorithm Settings

**Show_initial/show_best:** it takes us to the interface where details about the generated solution are displayed.
The top table presents the route each vehicle took in the solution, along with the associated profit and the time needed to complete the route. Below it, the table details all the cities visited in the solution, providing information on their respective profit, arrival time, wait time, service time, and departure time. The table on the right illustrates the travel time between cities. Lastly, the "form of best solution" section displays the final solution as a vector.

path details

| vehicle | route | profit | route time |
|---|---|---|---|
| V1 | [0, 48, 2, 49, 47, 0] | 60.0 | 1201 |
| V2 | [0, 27, 88, 75, 0] | 60.0 | 1103 |

travel time

| City1 | City 2 | travel_time |
|---|---|---|
| 0 | 48 | 23 |
| 48 | 2 | 43 |
| 2 | 49 | 39 |
| 49 | 47 | 2 |
| 47 | 0 | 18 |
| 0 | 27 | 17 |
| 27 | 88 | 43 |
| 88 | 75 | 21 |
| 75 | 0 | 16 |
| 0 | 17 | 33 |
| 17 | 45 | 47 |
| 45 | 51 | 5 |
| 51 | 69 | 21 |
| 69 | 0 | 16 |

city information

| | | | | | |
|---|---|---|---|---|---|
| City 47 | 10.0 | 1093 | 0 | 90 | 1183 |
| City 0 | 0.0 | 1022 | 0 | 0 | 1022 |
| City 0 | 0.0 | 0 | 0 | 0 | 0 |
| City 27 | 10.0 | 17 | 244 | 90 | 351 |
| City 88 | 30.0 | 394 | 251 | 90 | 735 |
| City 75 | 20.0 | 756 | 241 | 90 | 1087 |
| City 0 | 0.0 | 1022 | 0 | 0 | 1022 |
| City 0 | 0.0 | 0 | 0 | 0 | 0 |
| City 17 | 20.0 | 33 | 66 | 90 | 189 |
| City 45 | 10.0 | 236 | 305 | 90 | 631 |
| City 51 | 10.0 | 636 | 89 | 90 | 815 |
| City 69 | 10.0 | 836 | 80 | 90 | 1006 |
| City 0 | 0.0 | 1022 | 0 | 0 | 1022 |

total profit

170.0

draw graphe

the form of the solution

Solution: [[0, 48, 2, 49, 47, 0], [0, 27, 88, 75, 0], [0, 17, 45, 51, 69, 0]]

Figure 3.5: Initial solution results

path details

| vehicle | route | profit | route time |
|---|---|---|---|
| V1 | [0, 57, 63, 62, 74, 16, 88, 89, 2, 1, 47, 0] | 290.0 | 1181 |
| V2 | [0, 33, 25, 27, 15, 85, 6, 34, 21, 75, 0] | 240.0 | 1133 |

travel time

| City1 | City 2 | travel_time |
|---|---|---|
| 0 | 57 | 35 |
| 57 | 63 | 27 |
| 63 | 62 | 5 |
| 62 | 74 | 3 |
| 74 | 16 | 60 |
| 16 | 88 | 51 |
| 88 | 89 | 3 |
| 89 | 2 | 22 |
| 2 | 1 | 2 |
| 1 | 47 | 36 |
| 47 | 0 | 18 |
| 0 | 33 | 34 |
| 33 | 25 | 21 |
| 25 | 27 | 2 |
| 27 | 15 | 28 |

city information

| city | city profit | arrival time | wait time | service time | deparute time |
|---|---|---|---|---|---|
| City 0 | 0.0 | 0 | 0 | 0 | 0 |
| City 57 | 40.0 | 35 | 0 | 90 | 125 |
| City 63 | 50.0 | 152 | 19 | 90 | 261 |
| City 62 | 20.0 | 266 | 0 | 90 | 356 |
| City 74 | 50.0 | 359 | 0 | 90 | 449 |
| City 16 | 40.0 | 509 | 0 | 90 | 599 |
| City 88 | 30.0 | 650 | 0 | 90 | 740 |
| City 89 | 10.0 | 743 | 0 | 90 | 833 |
| City 2 | 30.0 | 855 | 0 | 90 | 945 |
| City 1 | 10.0 | 947 | 0 | 90 | 1037 |
| City 47 | 10.0 | 1073 | 0 | 90 | 1163 |
| City 0 | 0.0 | 1168 | 0 | 0 | 1168 |
| City 0 | 0.0 | 0 | 0 | 0 | 0 |
| City 33 | 40.0 | 34 | 53 | 90 | 177 |
| City 25 | 40.0 | 198 | 0 | 90 | 288 |
| City 27 | 10.0 | 290 | 0 | 90 | 380 |

total profit

740.0

draw graphe

Solution: [[0, 57, 63, 62, 74, 16, 88, 89, 2, 1, 47, 0], [0, 33, 25, 27, 15, 85, 6, 34, 21, 75, 0], [0, 13, 17, 8, 92, 93, 97, 51, 22, 69, 49, 0]]

Figure 3.6: Best solution results

**Draw graph**: draw graph button shows the map and the location of each city and the route taken by every vehicle.

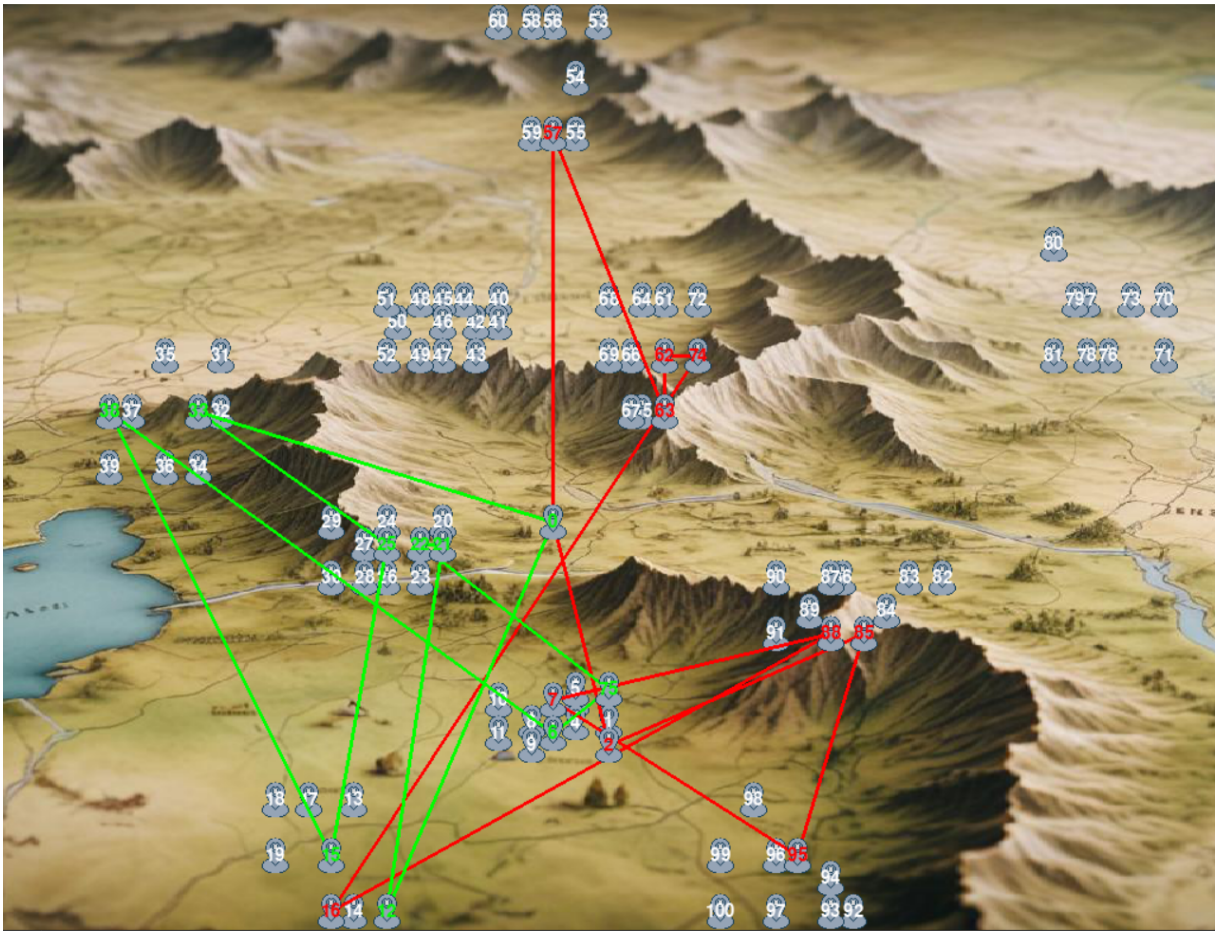

Figure 3.7: Road network in the solution and display of all locations

**compare**: Shows a comparison between the initial solution and final solution. Showing the total score of each one and the number of cities visited.

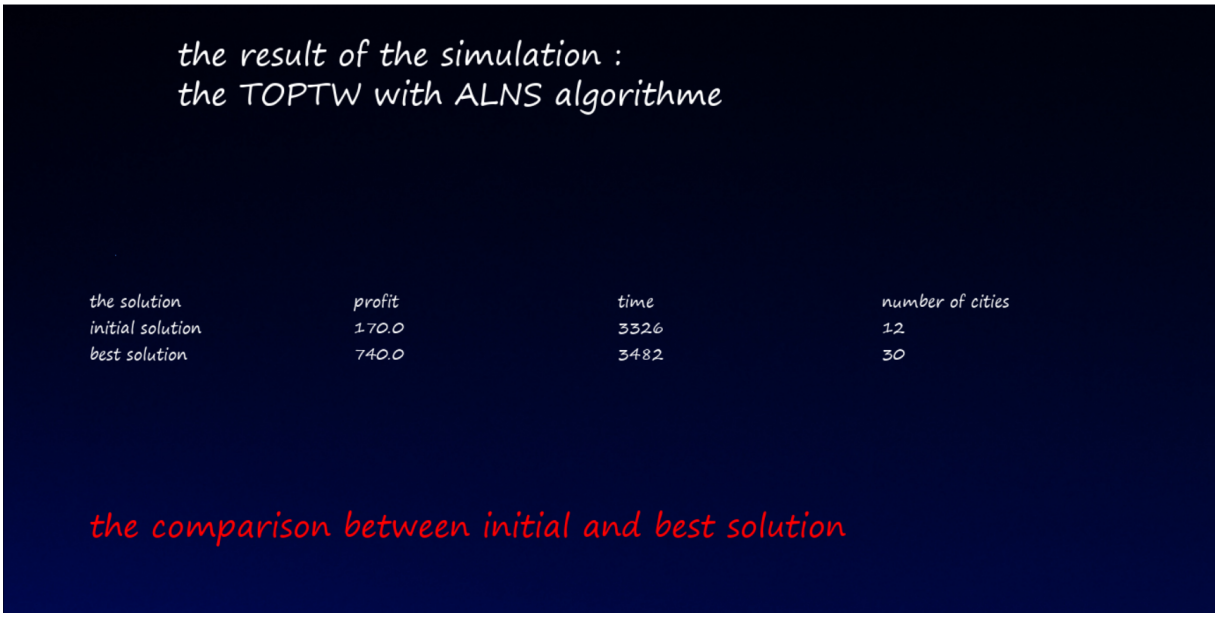


Figure 3.8: Comparison between initial solution and final solution

**all comparison**: shows all the previously tested instances including initial and final solution of every set of instance.

history of all tests

| Instance | vehicles | initial profit | final profit |
|---|---|---|---|
| c101.txt | 3 | 150.0 | 560.0 |
| c101.txt | 2 | 510.0 | 580.0 |
| c102.txt | 2 | 590.0 | 610.0 |
| c101.txt | 1 | 300.0 | 300.0 |
| c107.txt | 2 | 100.0 | 550.0 |

Figure 3.9: History of all previous tests

## 3.5 Test Instances

The experiments are conducted using instances specifically designed for the Team Orienteering Problem with Time Windows (TOPTW). These instances are tailored to address the unique challenges of the TOPTW, where the objective is to maximize the total score collected by visiting customers within given time windows.

The TOPTW instances are typically generated based on different configurations, such as the number of customers, time window settings, and geographical distributions. Commonly used instance sets for TOPTW include those adapted from the benchmark instances of Solomon , but with specific modifications to better align with the orienteering problem's objectives [57]. These adaptations often involve selecting a subset of customers with assigned scores and adjusting the time windows to create varying levels of difficulty [16]

For the TOPTW, instances are often categorized based on their geographical characteristics: random (R), clustered (C), and a mix of random and clustered (RC). These instances may be further divided into short (R1, C1, RC1) and long (R2, C2, RC2) planning horizons. The density of the time windows is also an important factor, with instances like 01 having all customers with time windows, while instances like 03 may only have time windows for a portion of the customers. [54] Additionally, TOPTW instances can vary in terms of the available resources, such as the number of vehicles (teams) and their capacities. Some instances are designed to be feasible without any external help, while others may require the use of additional resources or even outsourcing certain tasks due to limited vehicle availability.

In these instances, the cost of visiting a customer or completing a task may be modeled in various ways, often involving a combination of travel distance, time spent, and the difficulty of accessing the location within the given time window. The goal is to balance the total score collected with the associated costs, making strategic decisions about which customers to visit and when.

The Cordeau instances (pr01 to pr10) are another well-known set of benchmark instances specifically designed for the TOPTW. These instances, named pr01 to pr10, vary in size and complexity, ranging from smaller instances with fewer locations to larger, more challenging instances. Each instance in the Cordeau set is characterized by the number of locations, the associated scores, and the time windows for each location. The objective in these instances is to develop efficient routes that maximize the total collected score while respecting the time constraints [14].

These instances are widely used in academic research to benchmark the performance of various heuristic, metaheuristic, and exact algorithms developed for solving the TOPTW.

| Category | Details |
|---|---|
| **Instance Type** | Random, Clustered, Mixed (Random + Clustered) |
| **Prefix** | R1, R2, C1, C2, RC1, RC2 |
| **Planning Horizon** | Short (R1, C1, RC1), Long (R2, C2, RC2) |
| **Teams** | Variable |

Table 3.1: Structure of the datasets for the Team Orienteering Problem with Time Windows (TOPTW).

## 3.6 Parameter Adjustment

The parameters of our ALNS algorithm are set according to the values given in [7], as represented in the table 3.2.

| settings | $\rho_{\text{worst}}$ | $\sigma_1$ | $\sigma_2$ | $\sigma_3$ | $\rho_{\text{react}}$ | Segment |
|---|---|---|---|---|---|---|
| value | 3 | 33 | 9 | 13 | 0.1 | 100 |

Table 3.2: The parameters of the ALNS algorithm

The proposed ALNS algorithm is based on several parameters:

In worstRem, the parameter $\rho_{\text{worst}}$ controls the amount of randomization in the adaptive weight adjustment mechanism. The weight adjustment quantity is controlled by the parameters $\sigma_1$, $\sigma_2$, and $\sigma_3$, and the reaction factor $\rho_{\text{react}}$, which must be defined.

## 3.7 Comparison of Results

| Instance | BKS | ACO | T(s) | ILS | T(s) | VNS | T(s) | FSA | T(s) | SSA | T(s) | ALNS | T(s) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **C101** | 320 | 320 | 0.5 | 320 | 0.4 | 320 | 72.2 | 320 | 0.4 | 320 | 20.4 | 320 | 0.5 |
| **C102** | 360 | 360 | 0.7 | 360 | 0.3 | 360 | 102.2 | 360 | 0.3 | 360 | 20.7 | 340 | 0.4 |
| **C103** | 400 | 400 | 16.9 | 390 | 0.5 | 398 | 106.2 | 390 | 0.5 | 400 | 24.2 | 390 | 1.22 |
| **C104** | 420 | 420 | 33.5 | 400 | 0.3 | 418 | 124.3 | 410 | 0.3 | 420 | 21.9 | 400 | 2.92 |
| **R101** | 198 | 198 | 0.1 | 182 | 0.1 | 198 | 49.9 | 198 | 0.1 | 198 | 19.7 | 198 | 2.07 |
| **R102** | 286 | 286 | 11.1 | 286 | 0.2 | 285.2 | 77.9 | 282 | 0.2 | 286 | 21.0 | 286 | 2.41 |
| **R103** | 293 | 292.6 | 640.6 | 286 | 0.2 | 293 | 90.2 | 293 | 0.2 | 293 | 20.3 | 288 | 2.5 |
| **R104** | 303 | 303 | 164 | 297 | 0.2 | 303 | 95.9 | 294 | 0.2 | 303 | 23.2 | 297 | 2.95 |
| **RC101** | 219 | 219 | 0.2 | 219 | 0.2 | 219 | 61.3 | 216 | 0.2 | 219 | 19.8 | 216 | 2.05 |
| **RC102** | 266 | 266 | 30.9 | 259 | 0.2 | 266 | 0.353.3 | 249 | 0.2 | 266 | 20.2 | 266 | 24.37 |
| **RC103** | 266 | 266 | 57.2 | 265 | 0.3 | 266 | 62.9 | 265 | 0.3 | 266 | 20.7 | 258 | 2.52 |
| **RC104** | 301 | 301 | 29.4 | 297 | 0.3 | 301 | 58 | 263 | 0.3 | 301 | 27.5 | 301 | 2.41 |
| **C201** | 870 | 870 | 507.8 | 840 | 1.1 | 870 | 507.8 | 870 | 1.1 | 870 | 28.3 | 860 | 13.15 |

Table 3.3: Comparison between different approaches of TOPTW with 1 vehicule on solomon instances

| Instance | BKS | ACO | T(s) | ILS | T(s) | VNS | T(s) | FSA | T(s) | SSA | T(s) | ALNS | T(s) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **C101** | 590 | 588.0 | 110.8 | 590 | 1.4 | 587.0 | 83.8 | 590 | 1.4 | 590 | 26.2 | 580 | 33 |
| **C102** | 660 | 660.0 | 1427.4 | 650 | 0.9 | 653.0 | 100.0 | 660 | 0.9 | 660 | 26.5 | 650 | 38.67 |
| **C103** | 720 | 710 | 938.9 | 700 | 1.2 | 717 | 93.8 | 720 | 1.2 | 720 | 26.8 | 690 | 38.15 |
| **C104** | 760 | 754 | 1355 | 750 | 1.5 | 759 | 96.7 | 740 | 1.5 | 760 | 28 | 730 | 42.12 |
| **R101** | 349 | 349.0 | 36.5 | 330 | 0.4 | 349.0 | 35.3 | 344 | 0.4 | 349 | 25.0 | 343 | 34.15 |
| **R102** | 508 | 506.6 | 2006.6 | 508 | 0.9 | 498.5 | 64.1 | 497 | 0.9 | 508 | 43.9 | 486 | 36.64 |
| **R103** | 522 | 518.8 | 1095.1 | 513 | 0.9 | 515.4 | 51 | 505 | 0.9 | 519 | 32.3 | 491 | 4354.12 |
| **R104** | 552 | 540 | 2291.8 | 539 | 1.5 | 542.5 | 66.9 | 523 | 1.5 | 548 | 29.1 | 524 | 68.54 |
| **RC101** | 427 | 427.0 | 27.9 | 427 | 0.6 | 419.0 | 54.9 | 427 | 0.6 | 427 | 25.3 | 408 | 28.76 |
| **RC102** | 505 | 497.2 | 1496.9 | 494 | 0.8 | 504.1 | 61.1 | 480 | 0.8 | 505 | 42.4 | 492 | 38.62 |
| **RC103** | 524 | 510.2 | 1965.9 | 519 | 1.1 | 519 | 53.1 | 510 | 1.1 | 523 | 26.5 | 503 | 314.52 |
| **RC104** | 575 | 568 | 2381.3 | 565 | 0.7 | 568 | 53 | 558 | 0.7 | 575 | 63.1 | 558 | 35.7 |
| **C201** | 1460 | 1452.0 | 508.6 | 1400 | 2.7 | 1447.0 | 634.5 | 1440 | 2.7 | 1450 | 64.3 | 1360 | 418.38 |

Table 3.4: Comparison between different approaches of TOPTW with 2 vehicule on solomon instances

| Instance | BKS | ACO | T(s) | ILS | T(s) | VNS | T(s) | FSA | T(s) | SSA | T(s) | ALNS | T(s) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **C101** | 810 | 810.0 | 1515.6 | 790 | 1.1 | 810.0 | 67.0 | 810 | 1.1 | 810 | 31.7 | 770 | 2.2 |
| **C102** | 920 | 918.0 | 442.8 | 920 | 2.1 | 916.0 | 89.3 | 920 | 2.1 | 920 | 32.7 | 900 | 91.7 |
| **C103** | 990 | 972 | 1415.3 | 960 | 2.2 | 972 | 91.4 | 970 | 2.2 | 980 | 37.3 | 930 | 47 |
| **C104** | 1030 | 1004 | 681.7 | 1010 | 1.3 | 1020 | 78.9 | 1010 | 1.3 | 1010 | 33.1 | 950 | 48.91 |
| **R101** | 484 | 481.0 | 240.6 | 481 | 0.8 | 482.3 | 42.9 | 475 | 0.8 | 484 | 48.6 | 472 | 34.16 |
| **R102** | 694 | 682.0 | 2435.4 | 685 | 1.0 | 681.7 | 66.9 | 678 | 1.0 | 694 | 43.5 | 651 | 46.45 |
| **R103** | 747 | 729.8 | 2694.9 | 720 | 2 | 726.6 | 61.2 | 714 | 2 | 736 | 46.1 | 703 | 93.04 |
| **R104** | 778 | 760.6 | 2531.4 | 765 | 1.5 | 759.7 | 60.2 | 754 | 1.5 | 777 | 71.2 | 661 | 5.6 |
| **RC101** | 621 | 620.4 | 195.7 | 604 | 1.4 | 619.8 | 58.7 | 581 | 1.4 | 621 | 34.2 | 614 | 51.64 |
| **RC102** | 714 | 698.8 | 1145.5 | 698 | 1.3 | 710.3 | 60.4 | 689 | 1.3 | 710 | 41.3 | 696 | 61.97 |
| **RC103** | 764 | 741.4 | 2754 | 747 | 1.1 | 746.5 | 67.3 | 721 | 1.1 | 764 | 44.6 | 702 | 60.88 |
| **RC104** | 835 | 813.2 | 2277 | 822 | 1.3 | 824.5 | 59.8 | 795 | 1.3 | 814 | 33 | 797 | 64.45 |
| **C201** | 1810 | 1810.0 | 259.2 | 1750 | 2.2 | 1810.0 | 256.5 | 1800 | 2.2 | 1800 | 48.1 | 1750 | 106.86 |

Table 3.5: Comparison between different approaches of TOPTW with 3 vehicule on solomon instances

| Instance | BKS | ACO | T(s) | ILS | T(s) | VNS | T(s) | FSA | T(s) | SSA | T(s) | ALNS | T(s) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **C101** | 1020 | 1018.0 | 1049.1 | 1000 | 3.8 | 1013.0 | 78.5 | 1020 | 3.8 | 1020 | 37.1 | 960 | 39 |
| **C102** | 1150 | 1142.0 | 1211.3 | 1090 | 1.8 | 1139.0 | 90.1 | 1140 | 1.8 | 1150 | 40.0 | 1060 | 23.53 |
| **C103** | 1210 | 1186 | 2329.6 | 1150 | 2.5 | 1180 | 93.8 | 1180 | 2.5 | 1190 | 38.1 | 1150 | 103.53 |
| **C104** | 1260 | 1226 | 1493.9 | 1220 | 3 | 1248 | 79.7 | 1230 | 3 | 1230 | 41 | 1150 | 569.12 |
| **R101** | 611 | 608.0 | 55.1 | 601 | 1.4 | 610.2 | 40.4 | 605 | 1.4 | 611 | 33.8 | 600 | 38.64 |
| **R102** | 843 | 825.6 | 1924.5 | 807 | 1.7 | 828.4 | 59.4 | 841 | 1.7 | 843 | 45.2 | 787 | 44.97 |
| **R103** | 928 | 902.2 | 2622.2 | 878 | 2.2 | 909.9 | 68.8 | 898 | 2.2 | 926 | 97.1 | 887 | 49.74 |
| **R104** | 975 | 944.2 | 2343.9 | 941 | 3.8 | 954.8 | 62.9 | 928 | 3.8 | 964 | 84.7 | 888 | 61.32 |
| **RC101** | 811 | 805.4 | 1324.3 | 794 | 1.9 | 777.2 | 55.3 | 806 | 1.9 | 808 | 44.3 | 759 | 40.23 |
| **RC102** | 909 | 899.4 | 2218.8 | 881 | 2.3 | 893.4 | 67 | 876 | 2.3 | 902 | 126.9 | 889 | 47.7 |
| **RC103** | 975 | 941.8 | 2005.2 | 947 | 2 | 945.4 | 64.1 | 936 | 2 | 970 | 70 | 917 | 49.41 |
| **RC104** | 1065 | 1013.2 | 2139.3 | 1019 | 1.7 | 1033.5 | 64.8 | 1030 | 1.7 | 1059 | 75.3 | 997 | 51.84 |
| **C201** | 1810 | 1810.0 | 1.2 | 1810 | 1.1 | 1810.0 | 92.5 | 1810 | 1.1 | 1810 | 44.9 | 1810 | 1.5 |

Table 3.6: Comparison between different approaches of TOPTW with 4 vehicule on solomon instances

| Instance | BKS | ACO | T(s) | ILS | T(s) | VNS | T(s) | FSA | T(s) | SSA | T(s) | ALNS | T(s) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| pr01 | 308 | 308.0 | 256.2 | 304 | 0.5 | 308.0 | 81.2 | 304 | 0.5 | 305 | 8.3 | 305 | 19.28 |
| pr02 | 404 | 403.8 | 1147.8 | 385 | 0.6 | 403.9 | 251.5 | 392 | 0.6 | 404 | 29.1 | 380 | 49.4 |
| pr03 | 394 | 394.0 | 2024.3 | 384 | 1.0 | 390.5 | 419.8 | 381 | 1.0 | 394 | 59.9 | 339 | 519 |
| pr04 | 489 | 482.6 | 1404.7 | 447 | 1.9 | 488.1 | 733.2 | 470 | 1.9 | 489 | 106.7 | 451 | 625.3 |
| pr05 | 595 | 576.8 | 2075.7 | 576 | 4.6 | 586.1 | 1663.2 | 527 | 4.6 | 589 | 281.7 | 545 | 79.22 |
| pr06 | 591 | 564.6 | 2199.8 | 538 | 2.5 | 588.2 | 1628.4 | 557 | 2.5 | 575 | 253.4 | 541 | 92.58 |
| pr07 | 298 | 298.0 | 20.1 | 291 | 0.4 | 297.5 | 141.5 | 289 | 0.4 | 298 | 15 | 277 | 22.82 |
| pr08 | 463 | 462.6 | 2476.0 | 463 | 1.0 | 452.2 | 450.2 | 438 | 1.0 | 462 | 76.0 | 417 | 54.6 |

Table 3.7: Comparison between different approaches of TOPTW with 1 vehicle on cordeau instances

| Instance | BKS | ACO | T(s) | ILS | T(s) | VNS | T(s) | FSA | T(s) | SSA | T(s) | ALNS | T(s) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| pr01 | 502 | 502.0 | 602.2 | 471 | 0.5 | 487.3 | 80.1 | 492 | 0.5 | 502 | 20.2 | 474 | 100.92 |
| pr02 | 715 | 705.2 | 1017.6 | 660 | 1.2 | 696.9 | 216.0 | 686 | 1.2 | 712 | 37.3 | 684 | 31.64 |
| pr03 | 742 | 731.4 | 2083.0 | 714 | 3.3 | 715.3 | 356.0 | 710 | 3.3 | 741 | 217.7 | 680 | 53.12 |
| pr04 | 928 | 883.6 | 2439.3 | 863 | 4.1 | 906.4 | 578.5 | 889 | 4.1 | 905 | 245.0 | 794 | 72.39 |
| pr05 | 1103 | 1021.8 | 2278.5 | 1011 | 7.1 | 1064.8 | 982.7 | 986 | 7.1 | 1053 | 249.8 | 932 | 119.23 |
| pr06 | 1076 | 987.6 | 1724.1 | 997 | 9.8 | 1004.4 | 932.9 | 984 | 9.8 | 1022 | 439.3 | 923 | 155.41 |
| pr07 | 566 | 566.0 | 972.3 | 552 | 1.0 | 550.1 | 130.0 | 559 | 1.0 | 566 | 20.5 | 543 | 19.54 |
| pr08 | 834 | 813.0 | 1730.0 | 796 | 5.1 | 799.6 | 344.7 | 793 | 5.1 | 822 | 75.6 | 778 | 47.87 |

Table 3.8: Comparison between different approaches of TOPTW with 2 vehicle on cordeau instances

## 3.8 Insights and Commentary on the Results

Tables 3.3, 3.4, 3.5, 3.6, 3.7 and 3.8 provide a comprehensive comparison of various algorithms applied to solve the Team Orienteering Problem with Time Windows (TOPTW) with 1 to 4 vehicles, respectively. The algorithms compared are Ant Colony Optimization (ACO) [44], Iterated Local Search (ILS) [62], Variable Neighborhood Search (VNS) [58], Fast Simulated Annealing (FSA) [38], Slow Simulated Annealing (SSA) [38] , and Adaptive Large Neighborhood Search (ALNS). Each table lists the Best Known Solution (BKS) for each instance, along with the solution obtained by each algorithm, and the corresponding computational time (T) in seconds.

### 3.8.1 Performance Analysis of ALNS

The ALNS algorithm shows notable performance in several instances across different tables, especially considering the balance between solution quality and computational time. The following sections provide a detailed analysis of the performance of ALNS in comparison with other algorithms.

### 3.8.2 Solomon instances

#### 3.8.2.1 Comparison with 1 Vehicle (Table 3.3)

Focusing solely on the solution scores. ALNS performs well in some instances, but falls short in others.

For example, in C101, ALNS matches the best-known solution (BKS) of 320, performing equally with other algorithms. However, in C102, ALNS scores 340, which is below the BKS of 360, showing weaker performance compared to the other methods. Similarly, in C103 and C104, ALNS underperforms with scores of 390 and 400, respectively, compared to the BKS and other algorithms that meet or approach BKS.

On the R-class instances, ALNS performs closer to the BKS, achieving perfect scores on R101 (198) and R102 (286), indicating strong competitiveness. However, in R103 and R104, ALNS slightly lags behind with scores of 288 and 297, compared to the BKS of 293 and 303, showing some room for improvement. In the RC-class instances, ALNS achieves the BKS in RC102 and RC104 but falls short in RC101 and RC103 with lower scores.

Finally, on the C201 instance, ALNS scores 860, slightly below the BKS of 870, while other algorithms meet the optimal solution except for ILS.

#### 3.8.2.2 Comparison with 2 Vehicles (Table 3.4)

When the number of vehicles is increased to 2, ALNS we can observe a small fall back compared to the results of 1 vehicle.

In the **C-class instances**, algorithms like ACO, FSA, and SSA demonstrate strong performance, often meeting or closely matching the BKS. For instance, in **C101**, ACO, ILS, FSA, and SSA all achieve the BKS of 590, while ALNS trails with a score of 580. Similarly, in **C102**, ACO and FSA match the BKS of 660, but ALNS scores lower at 650. In **C103**, ACO, FSA, and SSA all meet the BKS of 720, while ALNS achieves 690, falling behind the top performers. A similar trend is seen in **C104**, where ALNS scores 730,

whereas ACO and SSA reach 754 and 759, approaching the BKS of 760. This shows that ALNS tends to underperform in C-class instances compared to the top algorithms.

The **R-class instances** further highlight this trend. In **R101**, ACO, VNS, FSA, and SSA all meet the BKS of 349, while ALNS scores 343, slightly lower but still ahead of ILS (330) and FSA (344). In **R102**, ALNS scores 486, below the BKS of 508, with ACO, FSA, and SSA performing significantly better by closely approaching the BKS. Similarly, in **R103**, ACO achieves the BKS of 522, while ALNS reaches 491, underperforming compared to SSA (519) and ACO. In **R104**, ALNS scores 524, falling behind ACO and SSA, which come closer to the BKS of 552. However, in this instance, ALNS surpasses ILS (539) and FSA (523), showing a modest improvement.

For the **RC-class instances**, ALNS continues to face challenges. In **RC101**, ACO, ILS, and SSA all match the BKS of 427, while ALNS scores 408, falling noticeably behind. In **RC102**, ACO and SSA come close to the BKS of 505, whereas ALNS achieves 492, lagging behind. Similarly, in **RC103**, ACO and SSA both approach the BKS of 524, while ALNS scores 503, indicating room for improvement. In **RC104**, ALNS manages to score 558, outperforming ILS (565) and VNS (568), but still falls short of the top performers like ACO and SSA, which achieve higher scores.

In the **C201** instance, ALNS faces its most significant gap, with a score of 1360 compared to the BKS of 1460. ACO, SSA, and VNS perform notably better, with ACO scoring 1452 and SSA reaching 1450.

#### 3.8.2.3 Comparison with 3 and 4 Vehicles (Table 3.5, 3.6)

As the complexity increases with 3 and 4 vehicles, we notice ALNS's performance going down and that is due to the lack of destruction and repair operators.

we can notice that Alns even though it achieves near-optimal solution it still underperforme other older algorithms.

For the instances with three vehicles, ALNS occasionally lags behind the BKS, as seen in instances like C101 and C102. However, it performs competitively in others, such as R102 and RC102, where it shows decent performance, although still below some of the more robust algorithms like ACO and VNS in many cases. In instances like C104, R104, and RC104, ALNS trails behind ACO, VNS, and ILS.

When using four vehicles, the pattern holds, with ALNS consistently offering solutions that are slightly suboptimal compared to BKS, especially in instances such as C101 and RC103. However, in some cases like R101 and RC104, it shows competitive performance. Despite the differences, ALNS has shown potential for certain instances but does not always outperform algorithms like ACO or SSA, which achieve solutions closer to the BKS in many cases.

### 3.8.3 Cordeau instances

Wether with 1 vehicle or multiple vehicles we can notice that ALNS is performing worst compared than its results in solomon instances and that goes to the high complexity of the cordeau instances and the lack of necessary methods and operators that can treat such difficult instances effectivly

#### 3.8.3.1 Instance `pr01` with 1 Vehicle:

ALNS achieves a solution of 305. This is quite close to the best known solution (BKS) of 308 achieved by ACO and VNS.

#### 3.8.3.2 Instance `pr05` with 1 Vehicle:

ALNS produces a solution of 545 . This score is better than FSA (527 ) and worst than SSA (589).

#### 3.8.3.3 Instance `pr01` with 2 Vehicles:

ALNS delivers a solution of 474. This is quite close to the BKS performing better than ILS and lower than other algorithms.

#### 3.8.3.4 Instance `pr04` with 2 Vehicles:

ALNS provides a solution of 794, which, is lower than the BKS (928), by a significant amout.

### 3.8.4 Overall Observations

Across all the tables, ALNS consistently demonstrates its ability to provide near-optimal solutions within a reasonable computational time with solomon instances. It often performs equal to or close to the best-known solutions; meanwhile, we can obviously notice that ALNS fall short in term of results on cordeau instances compared to older methods. The comparison with the most recent BKS shows that the current version of Alns is underperforming. But this does not reflect its true potential. As a new method, **ALNS** requires further refinement, especially in terms of adding more diverse operators for destruction and repair, which should lead to better exploration of the solution space and improved results in future implementations.

## 3.9 Conclusion

In this chapter, we presented the hardware and software tools used in the development of the proposed method, along with the graphical interfaces. We also discussed the results obtained from applying the ALNS algorithm to the Team Orienteering Problem with Time Windows (TOPTW). Finally, we compared our results with those achieved by other algorithms such as ACO, ILS, VNS, FSA, and SSA. Our algorithm although it may be weaker than other algorithms at its current form, its ability to adapt and improve by adding more diverse destruction and repair operators give it the potential to achieve further improvements if adjusted correctly.

# GENERAL CONCLUSION

In this thesis, we have explored the intricacies of the Team Orienteering Problem with Time Windows (TOPTW), a complex variant of the classic Orienteering Problem (OP) that combines the challenges of both time management and optimal path finding in team settings. The significance of TOPTW in various real-world applications, ranging from logistics and delivery services to emergency response and tourism, underscores its importance in operations research and optimization.

Through a comprehensive study, we first delved into the fundamental concepts of OP, TOP, and OPTW to build a solid foundation for understanding TOPTW. This was followed by a detailed mathematical formulation of the problem, which highlighted the various constraints and objectives that must be balanced to achieve optimal solutions. The literature review provided insights into some existing approaches for solving TOPTW, including exact algorithms, metaheuristics, and hybrid methods.

A key focus of this work was the development and application of the Adaptive Large Neighborhood Search (ALNS) algorithm to solve TOPTW. ALNS, known for its flexibility and efficiency in handling large and complex problem instances, was meticulously tailored to address the specific challenges of TOPTW. The algorithm's key components, including the initial solution generation, neighborhood structures, and adaptive mechanisms, were thoroughly discussed and implemented. The innovative use of destruction and repair operators within ALNS demonstrated its capability to explore diverse solution spaces while maintaining computational efficiency.

The practical implementation of ALNS was conducted in a controlled development environment, leveraging powerful tools such as Visual Studio Code and Python. Rigorous testing was performed on benchmark instances to evaluate the performance of ALNS in comparison to other established algorithms.

In conclusion, this thesis contributes to the field of optimization by advancing the understanding and application of ALNS in solving the TOPTW. The results not only validate the effectiveness of ALNS but also provide valuable insights for future research and potential applications in related fields. However, it is important to note that the results presented in this thesis are not final and leave room for significant improvements. The current implementation lacks operators that specifically focus on score maximization, which is a critical component for enhancing the overall solution quality in TOPTW. By incorporating additional operators designed to strategically increase scores, future

iterations of the algorithm could further refine its performance and effectiveness. Such enhancements would undoubtedly strengthen the proposed solution and pave the way for even more robust applications.

As TOPTW continues to present new challenges and opportunities, the findings of this thesis lay a solid foundation for further exploration and innovation in this dynamic area of study. Future research could explore not only the integration of more specialized operators but also the potential combination of ALNS with other advanced optimization techniques, thereby pushing the boundaries of what can be achieved in solving complex routing problems.

# BIBLIOGRAPHY


[1] . Alns. https://alns.readthedocs.io/en/latest/index.html.

[2] . Orienteering problem. https://topics-beta.apps.semanticscholar.org/topic/3407530548?corpusId=20373170. Accessed on June 4, 2024.

[3] . Vscode. https://code.visualstudio.com/docs,.

[4] Baeldung. Simulated annealing. https://www.baeldung.com/cs/simulated-annealing. Accessed: 2024-07-20.

[5] Rafael E. Banchs. Simulated annealing. https://rbanchs.com/documents/THFEL_PR15.pdf„ 2015.

[6] N. Bansal, A. Blum, S. Chawla, and A. Meyerson. Approximation algorithms for deadline-tsp and vehicle routing with time-windows. In *Proceedings of the thirty-sixth annual ACM symposium on theory of computing - STOC'04*. ACM, 2004. https://doi.org/10.1145/1007352.1007385.

[7] R. Bechkit and H. Boutana. Heuristic solution of the technician routing and scheduling problem (trsp). Master's thesis, University of Jijel, Jijel, Algeria, September 2022.

[8] N. Bianchessi, R. Mansini, and M.G. Speranza. A branch-and-cut algorithm for the team orienteering problem. *International Transactions in Operational Research*, 25(3):627–635, 2018.

[9] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 35(3):268–308, 2003.

[10] S. Boussier, D. Feillet, and M. Gendreau. An exact algorithm for team orienteering problems. *4OR*, 5(3):211–230, 2007. Received: 1 November 2005 / Revised: 1 May 2006 / Published online: 18 July 2006.

[11] I. Chao, R. Eggleston, and J.E. Beasley. Benchmarking optimization algorithms for the orienteering problem. *OR Spektrum*, 18(4):479–490, 1996.

[12] C. Chekuri and A. Kumar. Maximum coverage problem with group budget constraints and applications. In *Lecture Notes in Computer Science*, pages 72–83, 2004. https://doi.org/10.1007/978-3-540-27821-4_7.

[13] C. Chekuri and M. Pal. A recursive greedy algorithm for walks in directed graphs. In *46th annual IEEE symposium on foundations of computer science (FOCS'05)*, 2005. https://doi.org/10.1109/sfcs.2005.9.

[14] J.F. Cordeau, G. Laporte, and A. Mercier. A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational Research Society*, 52(8):928–936, 2001.

[15] T Cura. An artificial bee colony algorithm approach for the team orienteering problem with time windows. *Computers & Industrial Engineering*, 74:270–290, 2014. https://doi.org/10.1016/j.cie.2014.06.004.

[16] Division CIB, KU Leuven. The Orienteering Problem: Test Instances. https://www.mech.kuleuven.be/en/cib/op#autotoc-item-autotoc-6. [Accessed: 20-Sep-2024].

[17] C. Doğan. Balina optimizasyon algoritması ve gri kurt optimizasyonu algoritmaları kullanılarak yeni hibrit optimizasyon algoritmalarının geliştirilmesi. Master's thesis, T.C. Erciyes University, Kayseri, Turkey, 2019.

[18] L. Doğan. Robot yol planlaması için gri kurt optimizasyon algoritması. Master's thesis, Bilecik Şeyh Edebali University, Bilecik, Turkey, 2018.

[19] M. Gendreau, G. Laporte, and F. Semet. A branch-and-cut algorithm for the undirected selective traveling salesman problem. *Networks*, 32:263–273, 1998. https://doi.org/10.1002/(sici)1097-0037(199812)32:4<263::aid-net3>3.0.co;2-q.

[20] M. Gendreau, G. Laporte, and F. Semet. A tabu search heuristic for the undirected selective travelling salesman problem. *European Journal of Operational Research*, 106(2–3):539–545, 1998. https://doi.org/10.1016/s0377-2217(97)00289-0.

[21] M. Gendreau and J.Y. Potvin. Large neighborhood search. In *Handbook of Metaheuristics*, volume 272 of *International Series in Operations Research & Management Science*, pages 99–127. Springer, 2019.

[22] A. Gunawan, H. Chuin, and L. Kun. Sails: hybrid algorithm for the team orienteering problem with time windows. In *Proceedings 7th multidisciplinary international scheduling conference*, pages 276–295, 2015.

[23] A. Gunawan, H C. Lau, and K. Lu. Well-tuned ils for extended team orienteering problem with time windows. Technical report, LARC Technical Report Series, 2015. https://research.larc.smu.edu.sg/larcweb/larc/publications/technicalreports/Well-Tuned-ILS-for-Extended-Team-Orienteering-Problem-with-Time-WindowsTR-01-15.pdf.

[24] A. Gunawan, H C. Lau, and P. Vansteenwegen. Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research*, 255(2):315–332, 2016.

[25] A. Gunawan, H C. Lau, and P. Vansteenwegen. The team orienteering problem with time windows: An overview of metaheuristics. *Journal of the Operational Research Society*, 67(3):318–339, 2016.

[26] J.R. Hansen, K. Fagerholt, M. Stålhane, and J.G. Rakke. An adaptive large neighborhood search heuristic for the planar storage location assignment problem: application to stowage planning for roll-on roll-off ships. *Journal of Heuristics*, 2020.

[27] I. Hapsari, I. Surjandari, and K. Komarudin. Solving multi-objective team orienteering problem with time windows using adjustment iterated local search. *Journal of Industrial Engineering International*, 15:679–693, 2019.

[28] I. Hapsari, I. Surjandari, and K. Komarudin. Solving multi-objective team orienteering problem with time windows using adjustment iterated local search. *European Journal of Industrial Engineering*, 13(3):335–361, 2019. `https://core.ac.uk/download/483648950.pdf`.

[29] E.M. Herrera, J. Panadero, P. Carracedo, A.A. Juan, and E. Perez-Bernabeu. Determining reliable solutions for the team orienteering problem with probabilistic delays. *Mathematics*, 10(20), 2022.

[30] Mobisoft Infotech. Logistics app benefits for businesses. `https://mobisoftinfotech.com/resources/blog/logistics-app-benefits-for-businesses/`, 2024. Accessed: [Date of Access].

[31] M. Kantor and M. Rosenwein. The orienteering problem with time windows. *Journal of the Operational Research Society*, 43(6):629–635, 1992. `https://doi.org/10.1057/jors.1992.88`.

[32] M.A. KIMOUCHE and M. BENZID. Les méthodes métaheuristiques pour l'optimisation en génie électrique. Mémoire, Département D'Électrotechnique, Ministère de l'Enseignement Supérieur et de la Recherche Scientifique, PB 98, Ouled Aissa. 18000. Algérie, 2019. Master en Electromécanique, Option : Electromécanique.

[33] N. Labadie, R. Mansini, J. Melechovsk, and R. Wolfler Calvo. The team orienteering problem with time windows: An lp-based granular variable neighborhood search. *European Journal of Operational Research*, 220(1):15–27, 2012.

[34] N. Labadie, J Melechovský, and R Wolfler Calvo. Hybridized evolutionary local search algorithm for the team orienteering problem with time windows. *Journal of Heuristics*, 17(6):729–753, 2010. `https://doi.org/10.1007/s10732-010-9153-z`.

[35] J. Li and P. Fu. A label correcting algorithm for dynamic tourist trip planning. *Journal of Software*, 7(12):2899–2905, 2012. `https://www.jsoftware.us/vol7/jsw0712-32.pdf`.

[36] S.W. Lin and F.Y. Yu. A simulated annealing heuristic for the team orienteering problem with time windows. *European Journal of Operational Research*, 217(1):94–107, 2012.

[37] S.W. Lin and V.F. Yu. A simulated annealing heuristic for the team orienteering problem with time windows. *Production, Manufacturing & Logistics*, 2012.

[38] S.W. Lin and Vincent F. Yu. A simulated annealing heuristic for the team orienteering problem with time windows. *Computers & Operations Research*, 39(2):94–107, 2012.

[39] L Liu and A Medaglia. Towards an efficient hybrid algorithm for the team orienteering problem with time windows. *Computers & Operations Research*, 125:105112, 2021.

[40] Helena R. Lourenço, Olivier C. Martin, and Thomas Stützle. Iterated local search. In Frederick S. Hillier and Charles C. Price, editors, *International Series in Operations Research and Management Science*, pages 320–353. Springer, 2013. `https://doi.org/10.1007/0-306-48056-5_11`.

[41] R. Mansini, M. Pelizzari, and R. Wolfer. A granular variable neighbourhood search heuristic for the tour orienteering problem with time windows. Technical Report R.T 2006-02-52, University of Brescia, Italy, 2006.

[42] Microsoft. Visual studio code documentation, 2024.

[43] R. Montemanni and L M. Gambardella. An ant colony system for team orienteering problems with time windows. *Foundations of Computing and Decision Sciences*, 30(2):143–156, 2005.

[44] R. Montemanni and LM. Gambardella. An ant colony system for team orienteering problems with time windows. *Foundations of Computing and Decision Sciences*, 34(4):287, 2009.

[45] M. Peyman, X.A. Martin, J. Panadero, and A.A. Juan. A sim-learnheuristic for the team orienteering problem: Applications to unmanned aerial vehicles. *Algorithms*, 17(5), 2024.

[46] D. Pisinger and S. Ropke. Large neighborhood search. In M. Gendreau and J.Y. Potvin, editors, *Handbook of Metaheuristics*, pages 399–419. Springer US, Boston, MA, 2010.

[47] Python Software Foundation. Python documentation, 2024.

[48] H. Reddad, M. Zemzami, N. El Hami, and N. Hmina. Metaheuristic optimization and mechatronic application. *Laboratoire LGS, Université Moulay Sultan Slimane, Béni Mellal, Maroc*, 2022.

[49] G. Righini and M. Salani. Decremental state space relaxation strategies and initialization heuristics for solving the orienteering problem with time windows with dynamic programming. *Computers & Operations Research*, 36(4):1191–1203, 2009. `https://doi.org/10.1016/j.cor.2008.01.003`.

[50] S. Ropke and D. Pisinger. A unified heuristic for a large class of vehicle routing problems with backhauls. *European Journal of Operational Research*, 171(3):750–775, 2006.

[51] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In Michael Maher and Jean-Francois Puget, editors, *Principles and Practice of Constraint Programming — CP98*, pages 417–431, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[52] Nagendra Singh. Python programming language for beginners, 2021. Accessed: 2024-09-19.

[53] N Sluijk. Implementation of the iterative three-component heuristic for the team orienteering problem with time windows. Bachelor's thesis, Erasmus School of Economics, Erasmus University Rotterdam, Rotterdam, The Netherlands, May 2024. Econometrics and Operations Research.

[54] M.M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.

[55] Bhaskar Suman and P Kumar. A survey of simulated annealing as a tool for single and multiobjective optimization. *Journal of the Operational Research Society*, 57(10):1143–1160, 2006.

[56] F. Tricoire, M. Romauch, K F. Doerner, and R F. Hartl. Heuristics for the multi-period orienteering problem with multiple time windows. *Computers & Operations Research*, 37(2):351–367, 2010. https://doi.org/10.1016/j.cor.2009.05.012.

[57] Fabien Tricoire, Markus Romauch, Karl F Doerner, and Richard F Hartl. Heuristics for the multi-period orienteering problem with multiple time windows. *Computers & Operations Research*, 37(2):351–367, 2010.

[58] Fabien Tricoire, Martin Romauch, Karl F. Doerner, and Richard F. Hartl. Heuristics for the multi-period orienteering problem with multiple time windows. *Computers Operations Research*, 37(2):351–367, 2010.

[59] J. Tsitsiklis. Special cases of traveling salesman and repairman problems with time windows. *Networks*, 22(3):263–282, 1992. https://doi.org/10.1002/net.3230220305.

[60] P. Vansteenwegen. Planning in tourism and public transportation. *4OR*, 7(3):293–296, 2008. https://doi.org/10.1007/s10288-008-0086-4.

[61] P. Vansteenwegen, W. Souffriau, GV. Berghe, and DV. Oudheusden. A guided local search metaheuristic for the team orienteering problem. *European Journal of Operational Research*, 196(1):118–127, 2009. https://doi.org/10.1016/j.ejor.2008.02.037.

[62] P. Vansteenwegen, W. Souffriau, GV. Berghe, and DV Oudheusden. Iterated local search for the team orienteering problem with time windows. *Computers & Operations Research*, 36(12):3281–3290, 2009. https://doi.org/10.1016/j.cor.2009.03.008.

[63] P. Vansteenwegen, W. Souffriau, and K. Sörensen. The mobile mapping van problem: a matheuristic for capacitated arc routing with soft time windows and depot selection. In *Proceedings 13th information control problem in manufacturing*, pages 1119–1124, 2009.

[64] P. Vansteenwegen, W. Souffriau, and D. Van Oudheusden. The orienteering problem: A survey. *European Journal of Operational Research*, 209(1):1–10, 2011. Archived version: Author manuscript; The content is identical to the content of the published paper, but without the final typesetting by the publisher. Published version: http://dx.doi.org/10.1016/j.ejor.2010.03.045.

[65] P. Vansteenwegen, W. Soufriau, and D. Oudheusden. The orienteering problem: a survey. *European Journal of Operational Research*, 209(1):1–10, 2011. https://doi.org/10.1016/j.ejor.2010.03.045.

[66] P. Vansteenwegen and D. Van Oudheusden. The mobile tourist guide: An or opportunity. *OR Insights*, 20(3):21–27, 2007.

[67] Pieter Vansteenwegen, Wouter Souffriau, Greet Vanden Berghe, and Dirk Van Oudheusden. The city trip planner: An expert system for tourists. *Expert Systems with Applications*, 38(6):6540–6546, 2011.

[68] Vendasta. Local search definition. https://www.vendasta.com/glossary/local-search/, Accessed 2024.

[69] F.Y. Yu, P. Jewpanya, S.W. Lin, and R Perwira. Team orienteering problem with time windows and time-dependent scores. *Computers & Industrial Engineering*, 127:213–224, 2019.

[70] K. Zhou, C. Tan, Y. Zhao, J. Yu, Z. Zhang, and Y. Wu. Research on solving flexible job shop scheduling problem based on improved gwo algorithm ss-gwo. *Neural Processing Letters*, 56(1):26, 2024. https://doi.org/10.1007/s11063-024-11488-1.

[71] Yalan Zhou, Chen Li, and Yanyue Li. A simulated annealing for multi-modal team orienteering problem with time windows. https://link.springer.com/chapter/10.1007/978-981-13-2829-9_3. Accessed April 30, 2024.