الجمهوريـة الجزائـريـة الديمقراطيـة الشعبيـة

People's Democratic Republic of Algeria

وزارة التعليــم العالـي والبحـث العلمـي

N Réf : ……………

Centre Universitaire de MILA

University Center Abdelhafid Boussouf - Mila

Institute of Mathematics and Computer Science

Dissertation prepared with a view to obtaining a Master’s degree in computer science

Specialty: Artificial intelligence and its applications (I2A)

Heuristic Algorithms for The Hub Location Routing Problem

Prepared by:

Bouchelaghem Aicha

Benameur Aicha

Jury:

Supervised by:	Dr. GUEMRI Oualid	MCA	C.U. Mila
President:	Dr. MERABET Adil	MCB	C.U. Mila
Examiner:	Dr. BENHAMMADA Sadek	MCA	C.U. Mila

Academic year : 2024 - 2025

Acknowledgements

First and foremost, we praise and thank Allah for granting us the strength to complete this thesis.

We would like to express our deepest gratitude to our supervisor, **Dr. GUEMRI Oualid**, for his invaluable advice, insightful guidance, constructive feedback, patience, dedication and continuous support, all of which had a profound impact on the completion of this work.

Special thanks go to the board of examiners: **Dr. MERABET Adil** and **Dr. BENHAMMADA Sadek** for accepting to critically read and sincerely evaluate our work.

Our heartfelt thanks also go to all our esteemed professors, our parents, families, friends, and colleagues for their unwavering support and encouragement. we are truly grateful to everyone who contributed in any way to the successful completion of this thesis.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

HLRP: Hub Location Routing Problems.

HLP: Hub Location Problems.

SAHLP: Single-Allocation Hub Location Problem.

MAp-HMLP: Multiple Allocation p-Hub Median Location Problem.

p-HCLP: p-Hub Center Location Problem.

HLP-FC: Hub Location Problem with fixed Costs.

HCLP: Hub Covering Location Problem.

TSP: Traveling Salesman Problem.

VRP: Vehicle Routing Problem.

B&B: Branch and Bound.

GRASP: Greedy Randomized Adaptive Search Procedure.

GA: Genetic Algorithm.

LS: Local Search.

SA: Simulated Annealing.

TS: Tabu search.

VNS: Variable Neighborhood Search.

ILS: Iterated Local Search.

PSO: Particle swarm optimization.

ACO: Ant Colony Optimization.

ABC: Artificial Bee Colony.

CS: Cuckoo Search.

VND: Variable Neighborhood Descent.

BRKGA: Biased Random-Key Genetic Algorithm.

GVNS: General Variable Neighborhood Search.

SLS: Simple local Search.

NSRS: Neighborhood search with Random Selection.

Abstract

In this thesis, we propose three algorithms to solve the Hub Location Routing Problem: Simple Local Search, Tabu Search and Neighborhood search with Random Selection. In this problem, the goal is to select a subset of hubs from a set of candidate ones, to construct local routes that start and end at the same hub to serve not-hub nodes and to create an inter-hub route. The objective is to minimize the total transportation cost, including transportation cost of the local routes and the transportation cost of the inter-hubs route. Mainly, the proposed algorithms start with an initial random solution and keep improve it by exploring neighborhood solutions until the stopping criterion is met. The experiments showed that the three proposed methods obtained promising and competitive results comparing to the literature.

Key words: Hub Location Routing Problem, Local Search, Tabu Search, Neighborhood Search, metaheuristic, Hub location problem.

.

ملخص

في هذه المذكرة، نقترح ثلاث خوارزميات لحل مشكلة تحديد مواقع المراكز و توجيه المسارا ت: البحث المحلي البسيط، والبحث المحظور، والبحث عن الجوار مع الاختيار العشوائي. الهدف في هذه المشكلة هو اختيار مجموعة فرعية من المراكز من مجموعة من المراكز المرشحة، وإنشاء مسارات محلية تبدأ وتنتهي في نفس المركز لخدمة العقد غير المركزية، وإنشاء مسار بين المراكز. الهدف هو تقليل تكلفة النقل الإجمالية، بما في ذلك تكلفة نقل المسارات المحلية وتكلفة نقل مسار بين المراكز. بشكل أساسي، تبدأ الخوارزميات المقترحة بحل عشوائي أولي وتواصل تحسينه من خلال استكشاف حلول الجوار حتى يتم استيفاء معيار التوقف. أظهرت التجارب أن الطرق الثلاثة المقترحة قد حققت نتائج واعدة وتنافسية مقارنةً بالأدبيات.

الكلمات المفتاحية: مشكلة تحديد مواقع المراكز و توجيه المسارا ت, البحث المحلي ، البحث المحظور, البحث في الأحياء مع الإختيار العشوائي، الطرق التقريبية، مشكلة تحديد مواقع المراكز.

General introduction

Location and routing problems occupy a central place in the field of combinatorial optimization due to their significant practical relevance in various sectors, including: logistics, telecommunication, and transportation systems, etc. The optimal selection of facility location (such as hubs, distribution centres, or logistics platforms) and the design of efficient routing plans have a direct impact on the operational and economic performance of the considered networks. Poor hub placement or suboptimal routing may result in high transportation costs, low customer satisfaction, and inefficient use of resources, etc.

The Hub Location Routing Problem (HLRP) is one of the most complex variants of the Hub location problem. It involves partitioning the nodes of a graph into local routes, where each route is associated with a single hub. Each local route starts and ends at its associated hub, while a separate circular route connects all the selected hubs. This problem requires two levels of decision-making: (1) selecting the most appropriate set of hubs, which serve as consolidation and redistribution points for flows between customers, and (2) organizing efficient local routes around each hub to serve the assigned non-hub nodes. The objective is to minimize the overall cost of the system, including inter-hub transportation costs, and local routing costs.

This thesis focuses on the study of HLRP and we will propose new efficient methods to solve it. The approaches we propose include Simple Local Search (SLS), Tabu Search (TS), and Neighborhood Search with Random Selection (NSRS).

The following of the thesis is structured as: In the first chapter, we present an overview on the literature of the Hub location problems and the Hub location routing problems. In the second chapter, we will present famous and most relevant methods and algorithms that are widely used to solve combinatorial optimization problems. In the third chapter, we will present the algorithms we propose to solve the HLRP. In the fourth chapter, we will present and analyse the results obtained by applying our algorithms on a benchmark instances from the literature, and we will compare the obtained results to the literature ones. Finally, we terminate with a general conclusion.

Chapter 01: Hub location problems and Hub location routing problem

1.1 Introduction

Hub location problems are NP-hard combinatorial optimization problems [1] which have several applications notably in transportation and logistics. Mainly, the objective in these problems is to reduce costs of transportation of product(s) from one site to another and consequently improve the efficiency of the developed system. These problems involve the strategic placement of hubs, which act as intermediate points for consolidating and routing flows between origins and destinations.

Hub location problems aim to determine the optimal location of hub facilities and assign non-hub nodes to them, in order to consolidate flows and minimize overall transportation costs. They involve key decisions such as the number of hubs to open, their placement in the network, and how to allocate demand nodes. Different variant exist depending on allocation rules, capacity constraints, or cost structures.

This chapter presents the hub location problem and its variants, their applications, and their integration with the routing problems, leading to the Hub Location Routing Problem (HLRP) and its variants.

1.2 Basic Hub Location Problems (HLP)

The Hub Location Problems (HLP) are considered as variants of the facility location problems [1]. In HLP, the Hubs serve as intermediate facilities to transport products, providing indirect connections between the origins and the destinations of requests. In hub systems, products are transported from the origin to the hub, and then from the hub to the destination either directly or via another hub. The transportation cost between hubs is discounted with a factor α%, which reduces the total transportation cost and improves the efficiently of the transportation networks [1] [2] [3] [22].

Furthermore, hub location problems are classified regarding several characteristics, such as: the number of hubs, hub capacity, hub location costs, and other factors [1]. In following, we will present formal and mathematical definitions of basic variants of HLP. Therefore, first, we define the variables used in these definitions:

P: number of hubs.

h_{ij}: the amount of products to be transported from node *i* to node *j*

C_{ij}: the unit cost of transferring one unit of product from node *i* to node *j*.

α: discount factor for transportation between hubs, with ($0 \leq \alpha \leq 1$).

Z_{ij}^{km} : the flow of products transported from the origin node *i* to the destination node *j* via hub facilities located at nodes *k* and *m*.

C_{ij}^{km} : the unit transportation cost between the origin node *i*, the destination node *j*, and hub nodes *k* and m (in this order $i \rightarrow k \rightarrow m \rightarrow j$).

X_k: a decision variable that equals 1 if *k* is hub and 0 otherwise.

1.2.1 Single-Allocation Hub Location Problem (SAHLP)

The Single Allocation Hub Location Problem (SAHLP) is considered as the basic variant in the Hub Location Problems. In the SAHLP, each non-hub node can be assigned to only one hub. In addition, the number of hubs to be installed is not fixed and the fixed location cost of a hub is included in the overall cost of the solution.

Key decisions in SAHLP involves:

- Determining the number of hubs to be used. (P is not fixed).
- Location of hubs: Deciding where in the network the hubs should be located.
- Allocation of non-hub node to hubs: Assigning each non-hub node to a single hub.

The objective of SAHLP is to minimize the total cost including location cost of hubs and transportation cost [23].

In figure 1, we show an example of the Single-Allocation Hub Location Problem.

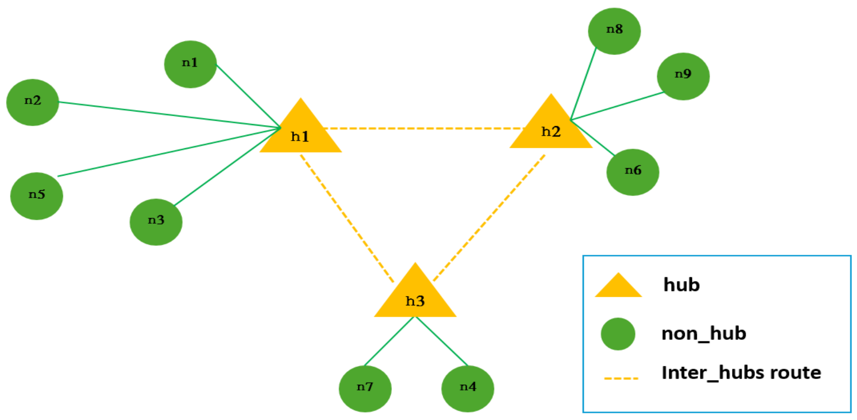


Figure 1:Single-Allocation Hub Location Problem

1.2.2Multiple Allocation p-Hub Median Location problem (MAp-HMLP)

The Multiple Allocation p-Hub Median Location Problem (MAp-HMLP) is an NP-hard optimization problem where the goal is to determine the best locations to install $\boldsymbol{p}$ hubs in a given network in order to minimize total transportation costs. Hubs serve as intermediate points to transfer products from the origins to the destinations. Each non-hub node can be assigned to one or more hubs.[1]

The mathematical formulation of the **MAp-HMLP** is as follows [1]:

$$min \sum_{\mathrm{i}} \sum_{\mathrm{j}} \sum_{\mathrm{k}} \sum_{\mathrm{m}} C_{ij}^{km}\, \mathrm{h_{ij}}\, Z_{ij}^{km} \qquad (1a)$$

s.t

$$C_{ij}^{km} = C_{ik} + \alpha C_{km} + C_{mj} \qquad (1b)$$

$$\sum_{k} X_k = P \qquad (1c)$$

$$\sum_{k} \sum_{m} Z_{ij}^{km} = 1 \forall i,j \qquad (1d)$$

$$Z_{ij}^{km} \leq X_m \quad \forall\, i,j,k,m \qquad (1e)$$

$$Z_{ij}^{km} \leq X_k \quad \forall\, i,j,k,m \qquad (1f)$$

$$Z_{ij}^{km} \geq 0 \quad \forall\, i,j,k,m \qquad (1g)$$

$$X_k \in \{0\,,1\} \quad \forall\, k \qquad (1h)$$

In this model, the objective function (1a) minimizes the total transportation cost. The constraints (1b) is the unit transportation cost between start node i, end node j, and hub nodes k and m (note that this is in order $i \rightarrow k \rightarrow m \rightarrow j$). The constraints (1c) ensures that exactly p hubs are selected. The constraints (1d) ensures that each origin–destination pair (i, j) is allocated to one pair of hub nodes (k, m). Note that the origin–destination pair (i, j) could be allocated to a single hub facility as indices k and m could be the same. The constraints (1e) and (1f) ensure that demand from origin node i to destination node j cannot be allocated to a hub pair (k, m) unless both nodes (k, m) are selected as hub facilities. Finally, the constraints (1g) and (1h) define the decision variable types.

1.2.3 p-Hub Center Location Problem (p-HCLP)

The p-Hub Center Location Problem consists in determining the optimal locations for ***p*** hubs in a given network and allocating the non-hub nodes to these hubs to minimize the maximum travel time (or distance) between any origin-destination **(o-d)** pair. This problem is particularly important for cases where minimizing travel time is essential, such as express mail services and emergency services [24] [25]. In figure 2, we show an example of the p-Hub Center Problem.

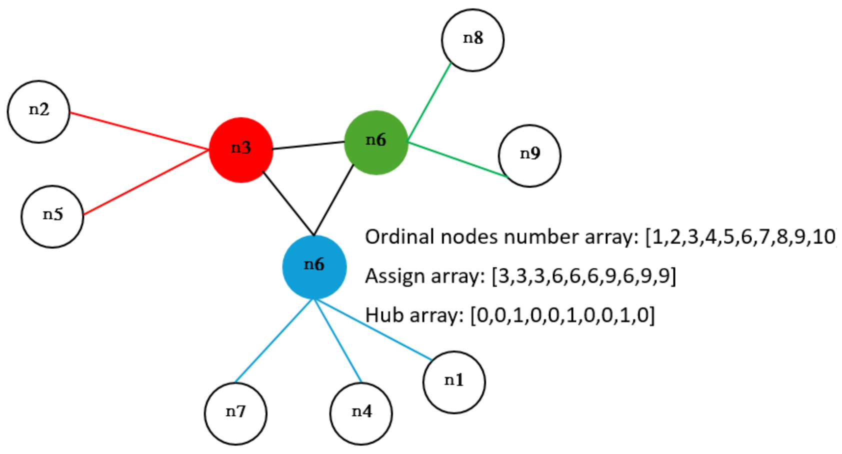

n8
n2
n3
n6
n9
n5
n6
Ordinal nodes number array: [1,2,3,4,5,6,7,8,9,10
Assign array: [3,3,3,6,6,6,9,6,9,9]
Hub array: [0,0,1,0,0,1,0,0,1,0]
n1
n7
n4

Figure 2:p-Hub Center Problem

1.2.4 Hub Location Problem with Fixed Costs (HLP-FC)

The Hub Location Problem with Fixed Costs is a variant of HLP where a **fixed cost** is imposed for assigning non-hub node to a hub. That means, when we assign a non-hub node ***i*** to a hub ***k***, a fixed cost g_{ik} will be considered in addition to the transportation costs [1].

1.2.5 Hub Covering Location Problem (HCLP)

The Hub Covering Location Problem (HCLP) is an extension of well-known Set Covering Location Problem. The goal in HCLP is to install a set of hubs so that each origin-destination pair of non-hub nodes is covered by a pair of hubs. A hub can cover a non-hub node if the distance between them is less than or equals a given maximal distance. For this coverage to be valid, the transportation cost between nodes - when routed through the selected hubs - must not exceed a maximum allowable value.[1]

1.3 Applications of hub location problems

Airlines and Airport Industries: The Hub Location Problem is one of the most important problems applied in the design of the airline networks. For the airlines companies, the goal is to reduce the number of direct flights and consequently reduce their operating costs. To do, they try to find the best intermediate points (hubs) for their passengers coming from different sites (origins). Then, the passengers are transported directly to their destinations (destination) together [9].

Emergency Services: The Hub Location Problem is effectively applied in the design of emergency service networks. Service facilities are established as central hubs, connected to nearby points (spokes) within a hub-and- spoke networks. This structure consolidates emergency resources optimizes their distribution, and enhances service coverage. Origin-Destination (O-D) flows converge at central hubs, reducing response times and costs while improving across network [10].

Telecommunication Networks: In telecommunication network design, the Hub Location Problem is applied to optimize data routing and infrastructure costs. The goal is to strategically place hubs (e.g. data centres) to efficiently route traffic-such as data or calls-from access points (origins) to end users (destinations). By minimizing direct connections between non-hub nodes and consolidating traffic while ensuring network reliability, scalability, and compliance with bandwidth or latency constraints [11][12].

Transportation Systems: Transportation systems utilize the Hub Location Problem to optimize costs and routing across multiple sectors. In freight transport, hubs streamline logistics operations and delivery routes. Public transport benefits from improved urban traffic planning through the efficient placement of hubs. In air transport, capacitated network designs enhance flight routing, while maritime transport hubs improve port management and container distribution. Overall, this network model boosts efficiency in logistics, aviation, and shipping systems [13].

1.4 Routing Problems

Routing Problems are NP-hard combinatorial optimization problems where the goal is to find the best routes for a fleet of vehicles to serve a set of customers (e.g., while respecting related constraints such as : (1) vehicle capacity, (2) time windows of the visits, (3)route durations, etc. These problems are widely used to model and solve real-life problems, such as : a school bus routing system (SBRP) involves identifying optimal routes to pick up and drop off students, while minimizing the total travel time and number of buses, taking into account constraints (1), (2) and (3) (e.g. A study applied by Hong Kong researchers to kindergartens in Hong Kong that reduced the total travel time of students by 29% compared to current practices) [35][36]. Another example of this problem is waste collection, where municipalities optimize garbage collection routes to reduce fuel costs and working hours, while respecting truck capacity and zone access restrictions (e.g. the application of this method in some regions of Eastern Finland demonstrated significant cost reductions could be achieved compared to existing practices) [37].

In the following, we will present basic variants of the routing problems: the Traveling Salesman Problem **(TSP) and** the Vehicle Routing Problem **(VRP)**. Also, we will present some variants of the VRP, and their real-world variants.

1.4.1 The Traveling Salesman Problem (TSP)

The Traveling Salesman Problem (TSP) is a well-known and widely studied optimization problem.[27]. It consists in finding the shortest path that visits all nodes of a given complete graph exactly once, and this path must begin and end in the same node. In figure 3, we show an example of the TSP.

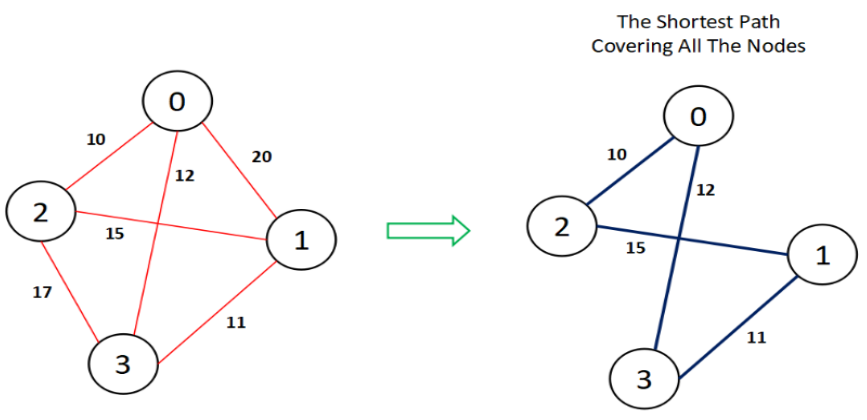


Figure 3 :The Traveling Salesman Problem

Mathematically, TSP can be modelled as [34]:

$$Min \sum_{i=0}^{N} \sum_{j\neq i, j=0}^{N} c_{ij}\, x_{ij} \qquad (2a)$$

Subject to:

$$\sum_{i=0, i\neq j}^{N} x_{ij} = 1 \qquad j = 0, \ldots, n \qquad (2b)$$

$$\sum_{j=0, j\neq i}^{N} x_{ij} = 1 \qquad i = 0, \ldots, n \qquad (2c)$$

$$\sum_{i,j\in U} x_{ij} \leq |U| - 1 \quad \forall U \subset V\,, \quad 2 \leq |U| \leq |V| - 2 \qquad (2d)$$

$$x_{ij} \in \{0,1\} \qquad i,j = 0, \ldots, n \qquad (2e)$$

Here:

N: total number of cities.

V: set of cities, labelled from 0 to *n*.

c_{ij}: distance from city *i* to city *j*

x_{ij}: is a binary decision variable that indicates whether a path exists from city *i* to city *j*. It is equal to 1 if the path goes from city *i* to city *j*, 0 otherwise.

Objective function (2a) aims to minimize the total travel cost or distance. The first constraint (2b) ensures requires that each city i be arrived at from exactly one other city j . Constraint (2c) ensures that from each city i, exactly one departure is made to another city. Constraint (2d) (subtour elimination constraints) enforce that there is only a single tour covering all cities, and not two or more disjointed tours that only collectively cover all cities.

1.4.2 The Vehicle Routing Problem (VRP)

In the Vehicle Routing Problem (VRP), the goal is to construct a set of routes for a fleet of vehicles to visit and serve a set of sites or customers while minimizing an objective function mostly represents the operating cost(s) and satisfying the constraints related to the problem. In the following, we present some variants of the VRP, which are studied in the literature. In figure 4, we show an example of the VRP.

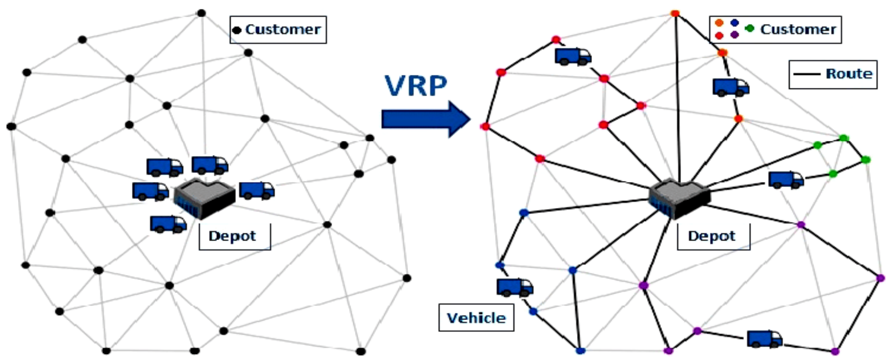


Figure 4 :The Vehicle Routing Problem

1.4.3 Variants of Vehicle Routing Problem (VRP)

In the literature, several constraints and assumptions are considered when solving the VRP, which creates numerous variants of it. In the following, we show the variant of the VRP and the specific assumptions and/or constraints considered in it.

1. **Capacitated VRP (CVRP)**
    - Each vehicle has a specific maximum capacity.
2. **VRP with Time Windows (VRPTW)**
    - Each customer must be a served in a given time window.

3. **VRP with Pickup and Delivery (VRPPD)**
    - The vehicles picks up goods/products from some locations and deliver them to other ones. The delivery visits must be planified after picking up the corresponding products.
4. **Multi-Depot VRP (MDVRP)**
    - Several depots are available and we can serve a customer by any vehicle that starts its route from any depot.
5. **Periodic VRP (PVRP)**
    - The horizon time of the service is decomposed to set of periods, where each customer might be visited one or several times.
6. **VRP with Backhauls (VRPB)**
    - Similar to The VRPPD described above, however in the VRPB the pickup sites must be visited before the delivery ones.
7. **Dynamic VRP (DVRP)**
    - In the DVRP, one or more element in the problem can be changed, such as : the requests of customer, the availability of vehicles, etc.

More details on variants of VRP can found in [16] [17] [18] [29] [30] [31].

1.5 Hub location routing problem (HLRP)

1.5.1 Definition of HLRP

The Hub Location Routing Problem (HLRP) is an **NP-hard** combinatorial optimization problem, which combines the Hub Location and the Vehicle Routing Problems. It consists in selecting $\boldsymbol{p}$ hubs from $\mathbf{n}$ nodes candidates, and assigning each non-hub node to exactly one hub. Then a local route is created for each hub, which starts and ends at this hub. Finally, we need to create another route (hub route) that visits all hubs. The number of hubs is represented by $\boldsymbol{p}$**,** and the maximum capacity of each local route is represented by C and it refers to the maximal number of nodes that can be inserted in a local route including the hub. The objective is to minimize the total transportation costs including transportation cost of the local routes and the cost of the hub route. [19] [20]. In figure 5, we show an example of the solution of Hub location routing problem.

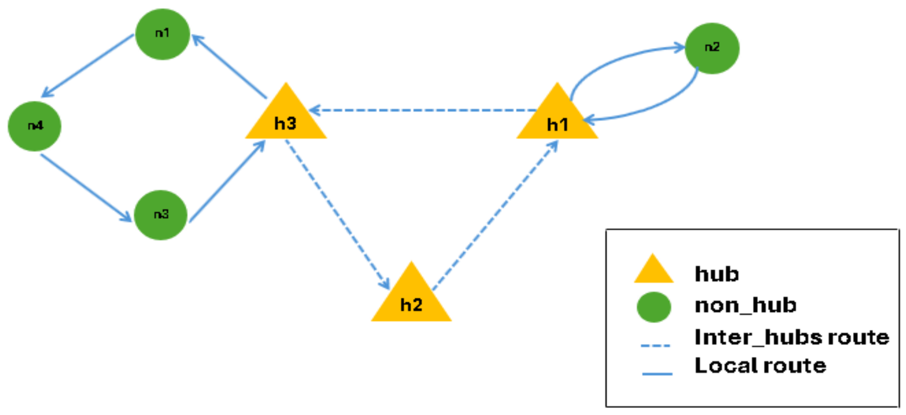


Figure 5:A solution of Hub location routing problem

1.5.2 Variants of HLRP

In the following, we cite three variant of the HLRP:

- **Capacitated Single-Allocation Hub Location Routing Problem (CSA-HLRP)**

The Capacitated Single-Allocation Hub Location Routing Problem (CSA-HLRP) is a variant of the Hub Location Routing Problem (HLRP) where: (1)each non-hub node is allocated to exactly one selected hub, and (2) the capacity constraint is imposed on both the hubs and the vehicles[32][33].

- **Capacitated Multi-Allocation Hub Location Routing Problem (CMA-HLRP)**

The Capacitated Multi-Allocation Hub Location Routing Problem (CMA-HLRP) is a variant of the Hub Location Routing Problem (HLRP) where: (1) each non-hub node can be connected to multiple hubs, allowing flexibility in service allocation, (2) both hubs and vehicles are subject to capacity constraints, ensuring resource limitations are respected, and (3) route length constraints are imposed to maintain service quality and efficiency [32].

- **Stochastic Single-Allocation Hub Location Routing Problem (CSAHLRPSD)**

The Stochastic Single-Allocation Hub Location Routing Problem (CSAHLRPSD) extends the HLRP to account for uncertainty in customer demand. In this problem: (1) each non-hub node is allocated to a single hub and the objective is to determine the hub locations and vehicle routes under demand variability. (2) Both hub and vehicle capacities are considered, with the goal of minimizing total operational costs while ensuring service reliability. This variant is particularly relevant for logistics and express delivery services operating in dynamic and uncertain environments [21].

1.6 Conclusion

In this chapter, we first highlighted the hub location problem and we showed various of its basic variants including: Single-Allocation Hub Location Problem, p-hub median Location Problem, p-hub Center Location Problem, Hub location Problem with Fixed Costs and Hub Covering Location Problem. After that, we explored some routing problems, including the Traveling Salesman Problem (TSP), the Vehicle Routing Problem (VRP), and seven variants of the VRP. Finally, we presented and defined the Hub Location Routing Problem (HLRP) and three of its variants. In the next chapter, we will present the methods and the algorithms used to solve NP-hard optimization problems including the problems discussed in this chapter.

Chapter 2: Methods and algorithms to solve optimization problems

2.1 Introduction:

Combinatorial optimization problems aim to find the best solution from a finite set of solutions while satisfying the defined constraints. To do, two main categories of methods are used: exact methods, which guarantee an optimal solution but they are useful only for small problems due to their computational time, and approximation methods, which provide high-quality solutions in a reasonable CPU time, making them suitable for large-scale problems.

In this chapter, first, we present a general define of a combinatorial optimization problem and, then we will present a various algorithms and methods that can be used to deal with the optimization problems.

2.2 Combinatorial optimization problems

The goal in a combinatorial optimization problem is to search for the best solution which minimizes or maximizes a given objective function from the set of possible solutions S. The challenge is that the S contains a huge number of solutions and examine all of them is not an easy task. Here is a simple formulation of an optimization problem [82]: $P = (S, f, \Omega)$

- P : represents the optimization problem.
- S : symbolizes the search space of the problem domain.
- f : represents the objective function .
- Ω : corresponds to the set of problem's constraints.

2.3 Exact Methods

The objective of the exact methods is to search for the optimal solution to any problem within the entire search space, which encompasses all possible solutions. These methods use techniques to exclude solution subsets that cannot be optimal or don't have promising solutions, thereby reducing the search time. However, they are used for small-sized instances, for large-sized instances these methods are impractical due to the excessive computational time required to find the optimal solution [48]. Among the methods, we can cite: (1) Backtracking method, (2) dynamic programming, (3) column generation, (4) the A* algorithm, (5) branch & bound, (6) (6) branch & cut, (7) branch & price, etc. [48].

2.3.1 Backtracking algorithm

The backtracking algorithm is a classical recursive method used for combinatorial and constraint satisfaction problems. It works by trying out different paths, and if one doesn't lead a solution, it backtracks and tries another until it finds the correct one.

The algorithm explores the search space in a depth-first manner attempting to construct a complete solution by extending partial solutions step by step. If at any step a constraint is violated, the algorithm revert to a previous state and tries a different option. This pruning reduces the number of configurations to explore, which is important for solving NP-complete problems efficiently. Backtracking guarantees finding an optimal or feasible solution if one exists. However, its worst-case time complexity remains exponential which making it more suitable for small sized instances [39,40,41].

2.3.2 Branch and bound

The branch and bound algorithm (B&B) first appeared in in the 60s [42] and is used to solve NP-hard problems. This algorithm is based on intelligently exploring all possible solutions by constructing a search tree using a partitioning technique. More, it prunes branches that do not lead to the optimal solution, thereby reducing computational time [43].

To create a branch and bound (B&B) method, we have to develop the following techniques:

The separation technique (branching): this involves dividing the search space into several parts such that their union covers all possible solutions.

The evaluation technique (bounding): this requires calculating the upper and lower bounds for each branch to determine if there might be better solutions than those already explored.

The exploration technique: used to determine the order in which branch are visited. There are several way to explore branches such as: depth-first, breadth-first, best-first, etc.

***Algorithm 1**: Branch and Bound for minimization*

```
1:  $T_{root} \leftarrow Create\ the\ root\ of\ the\ search\ tree\ according\ to\ the\ separation\ technique$ ;
2:  $U_{bound} \leftarrow +\infty; L \leftarrow T_{root}$ ;
4:  while (L≠∅)
5:     $S_C \leftarrow$ Explorer(L);
6:     If (Evaluation ($S_C$) $\leq U_{bound}$)
7:        $L' \leftarrow$ All partial solutions $S'$ that can be obtained from Sc;
8:        For (each S' in L' do)
9:           If (S' is a complete solution)
10:             update $U_{bound}$; update $S_{best}$;
12:          Else
13:             add $S'$ to L ;
14:          End
15:       End
16:    Else
17:       delete $S_C$ from L ;
18:    End
19: End
20: Return $S_{best}$;
```

2.4 Approximation Methods:

Approximate methods are commonly used to solve combinatorial optimization problems. especially when the problem involves complex constraints or a large solution space. In contrast to exact methods, which can take an impractically long time sometimes even years to find the optimal solution, approximate methods focus on finding good quality solutions within a reasonable computational time.

Approximate methods are be divided into several classes including: heuristics and metaheuristics.

2.4.1 Heuristics

Several definitions of a heuristic have been proposed by various researchers in the literature. Here, we present some of the most notable ones:

Definition 1.1 *«A heuristic (heuristic rule, heuristic method) is a rule of thumb, strategy, trick, simplification, or any other kind of device which drastically limits search for solutions in large problem spaces. Heuristics do not guarantee optimal solutions; in fact, they do not guarantee any solution at all; all that can be said for a useful heuristic is that it offers solutions which are good enough most of the time.* » Feigenbaum and Feldman (1963) [44]

Definition 1.2 *«A heuristic method (or simply a heuristic) is a method which helps in discovering a problem's solution by making plausible but fallible guesses as to what is the best thing to do next. » Feigenbaum and Feldman (1963) [44]*

Definition 1.3 *«A heuristic is a rule of thumb, strategy, method, or trick used to improve the efficiency of a system which tries to discover the solutions of complex problems. » Slagle (1971) [45]*

Definition 1.4 *«Heuristics are criteria, methods, or principles for deciding which among several alternative courses of action promises to be the most effective in order to achieve some goal. » Pearl (1984) [46]*

Moreover, in the field of combinatorial optimization, a heuristic is an approximate method designed to solve specific problem by relying on a deep understanding of it. Its primary goal is to provide high-quality solutions, although they may not necessarily be optimal, while ensuring a reduced computation time [47].

1) Greedy constructive heuristics

The greedy heuristic [48] is an algorithmic model that builds the solution gradually element by element. It starts with an empty solution ***S*** and at each stage of the construction process, selects the next element ***e*** using the local optimal choice strategy (greedy choice rule), which measures the increase or decrease in the objective function when adding each element.

Algorithm 2*:Greedy algorithm for minimization*

```
1:  $S \leftarrow \emptyset$;
2:  $C \leftarrow \{e_1, e_2, ..........., e_n\}$;
3:  Evaluate the incremental cost c(e) for all $e \in C$;
4:  While ($C \neq \emptyset$)
5:      $e_b \leftarrow$ select $e \in C$ with the smallest incremental cost c(e);
6:      $S \leftarrow S \cup \{e_b\}$;
7:      $C \leftarrow C - \{e_b\}$;
8:      Revaluate the incremental cost c(e) for all $e \in C$;
9:  End
10: return S;
```

2) Randomization and Greedy Randomized heuristics

Randomization plays an important role in the optimization algorithms [48]. It is used to enhance the exploration of the search space, improve solution diversity and avoid local optima, as is the case with greedy algorithms.

The difference between the greedy randomized heuristic (GRH), and the traditional greedy heuristic (GH) is, in the GRH we select the next element to be added to the solution randomly from a restricted list containing several locally optimal elements, where the GH selects the best [48] [49]. This algorithm is widely employed when solving hard optimization problems.

Algorithm 3:*Greedy randomized algorithm for minimization*

```
 1:  S ← ∅
 2:  $C \leftarrow \{e_1, e_2, \ldots\ldots\ldots\ldots, e_n\}$;
 3:  Evaluate the incremental cost c(e) for all e ∈ C;
 4:  While (C ≠ ∅)
 5:     Build the restricted candidate list with the candidate elements having the
        smallest incremental costs;
 6:     $e_b$ ← Select random e ∈ the restricted candidate list;
 7:     S ← S ∪ {$e_b$};
 8:     C ← C − {$e_b$};
 9:     Revaluate the incremental cost c(e) for all e ∈ C;
10:  End
11:  return S;
```

3) Local Search algorithms

Local search algorithm (LS) starts from a complete initial solution and try to find a better solution in an appropriately defined neighborhood structure. The algorithm systematically neighbouring solutions to find one that improves the quality of the current solution. When a better solution is found, it replaces the current solution, and the search process continues. This cycle is repeated until no further improvements can be found, leading the algorithm to converge to a local optimum [50].

***Algorithm 4:**Local Search*

```
1: $S \leftarrow$ start solution;
2: While (S is not a local optimal)
3:     $S_n \leftarrow$ select the best solution S $\in$ N(S);
4:     If (f ($S_n$) is better than f (S))
5:         S $\leftarrow S_n$;
6:     End
7: End
8: Return S;
```

2.4.2 Meta-Heuristic

According to Osman and Laporte (1996): *"A metaheuristic is formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions."* [51]

The main components of any Metaheuristic algorithm are intensification and diversification , also called exploitation and exploration. The combination of these two mechanisms helps guide the search process and find best solutions. [52]

- **Diversification:** consists in exploring the search space on a global scale by moving solution to other regions of the space search.

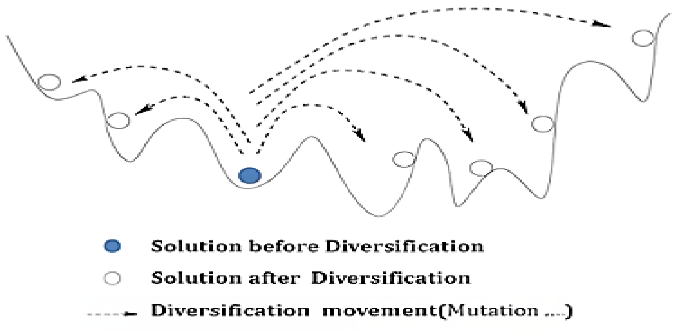

● Solution before Diversification
○ Solution after Diversification
Diversification movement(Mutation ...)

Figure 6:Diversification Process of a solution.

- **Intensification:** consists in focusing on the search in a local region by exploiting the information that a current good solution is found in this region.

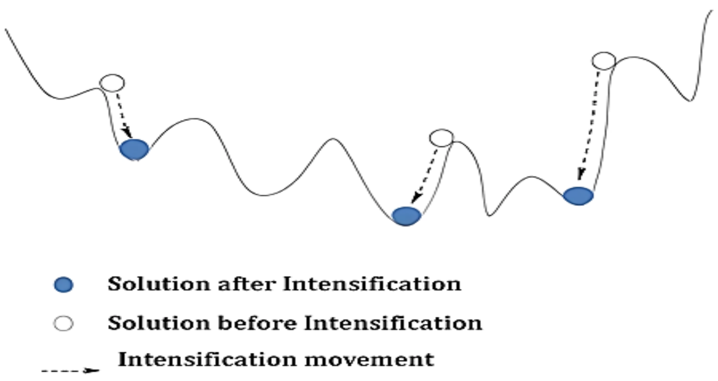

Solution after Intensification
Solution before Intensification
Intensification movement

Figure 7:Intensification Process of several solutions

a) Single-solution based metaheuristics

Single-solution-based metaheuristics work with a single solution. They start the search with an initial solution and gradually improve its quality by exploring its neighborhood structure. This improvement relies on a series of local modifications applied to the current solution to efficiently exploit the search space. Several methods of this type have been developed in the literature, including: Hill Climbing (HC), Simulated Annealing (SA), Tabu Search (TS), Variable Neighborhood Search (VNS), Iterated Local Search (ILS), Guided Local Search (GLS), and Greedy Randomized Adaptive Search Procedure (GRASP), etc.

1) Simulated Annealing

The simulated annealing algorithm (SA) is a method proposed by Kirkpatrick, Gelatt, and Vecchi [53], inspired by the annealing process used in metallurgy. This thermal procedure slowly and in a controlled manner to allow the atoms to gradually reorganize. This rearrangement promotes the formation of a more stable and homogeneous structure by reducing defects that may occur during the transition from the liquid to the solid state.

The SA algorithm starts the search with an initial solution $\boldsymbol{S}$ and initial temperature $\boldsymbol{T}$. Then, at each iteration, SA randomly selects a neighboring solution $\boldsymbol{S_n}$ of current solution. if $\boldsymbol{S_n}$ is better than $\boldsymbol{S}$, then $\boldsymbol{S}$ is replaced by $\boldsymbol{S_n}$. otherwise, $\boldsymbol{S_n}$ is accepted with a probability equal to $\boldsymbol{e^{-(\Delta/T)}}$ (**where** $\Delta = f(\boldsymbol{S_n}) - f(\boldsymbol{S})$ in minimization problems and $\Delta = f(\boldsymbol{S}) - f(\boldsymbol{S_n})$ in maximisation problems**).** Then, the temperature and the best solution found are updated. The algorithm terminates when the stopping criterion is met.

Algorithm 5:*Simulated annealing*

```
1:  $S \leftarrow$ initial solution;
2:  T ← initial temperature;
3:  while (the stop criterion is not met) do
4:      randomly choose $S_n \in N(S)$
5:      r ← a random number between 0 and 1.
6:      calculate Δ;
7:      If ($S_n$ is better than S Or $r < e^{-(\Delta/T)}$)
8:          S ← $S_n$;
9:          if ($S$ is better than $S_b$)
10:             $S_b \leftarrow$ S;
11:         End
12:     End
13:     update T;
14: End
15: return $S_b$;
```

2) Tabu Search

Tabu search (TS) was proposed in 1986 by Glover [54]. It is an optimization algorithm that extends local neighbourhood search. In addition, it uses mechanism to prevent getting trapped in local optima and to explore search space more efficiently. This mechanism is the tabu list, which is used to record the solutions discovered after each iteration to prevent them from being accepted in the future [55].

The first step in the tabu search algorithm is to create an initial solution ***S*** with empty list tabu **L**. after that, the algorithm enters a loop in which the neighborhood structure **N(S)** of the current solution S is created. The best solution $\boldsymbol{S_n}$ from **N(S)**that is not present in **L** is then selected. Next, ***S*** replaced by $\boldsymbol{S_n}$. ***S*** is then added to **L** and the oldest solution is removed (if **L** is full). If ***S*** is better than $\boldsymbol{S_b}$ it replaces it. The loop continues until the stopping condition is satisfied. Finally, the algorithm returns $\boldsymbol{S_{best}}$.

Algorithm 6:*Tabu Search*

```
1:  $S \leftarrow$ initial solution;
    L $\leftarrow \emptyset$;
2:  *while* the stop criterion is not met do
3:      Generate N(S);
4:      find the best solution $S_n$ , { $S_n \in$ N(s) and $S_n \notin$ L} ;
5:      Update L;
6:      $S \leftarrow S_n$ ;
7:      *If* (S is better than $S_b$) Then
8:          $S_b \leftarrow$ S;
9:      *End*
10: *End*
11: return $S_b$;
```

3) Greedy Randomized Adaptive Search Procedures

Greedy Randomized Adaptive Search Procedures (GRASP) is a multi-start metaheuristic developed in the 1980s by Feo and Resende [56] [57] to solve combinatorial optimization problems. Each iteration consists of two phases [57] [58]:

Construction phase: in this phase, a feasible solution is iteratively constructed using a greedy randomized function, encouraging diversity and exploration.

Local search phase: in this phase, the neighborhood of the constructed solution is explored to improve it until a local optimum is reached.

GRASP works by independently sampling the solution spaces at each iteration, retaining only the best solution found as the final result.

Algorithm 7:*Greedy Randomized Adaptive Search Procedures for Minimization*

```
1:  $f_i \leftarrow \infty$;
2:  *While* (the stop criterion is not met)
3:      $S \leftarrow$ Greedy Randomized Algorithm ( );
4:      *if* (S is not feasible)
5:          $S \leftarrow$ Repair Solution(S) ;
6:      *End*
7:      $S \leftarrow$ Local Search(S) ;
8:      *if* ( $f(S) < f_i$)
9:          $S_b \leftarrow S$ ;
10:         $f_i \leftarrow f(S)$ ;
11:     *End*
12: *End*
13: return $S_b$;
```

4) Variable Neighborhood Search

Variable Neighborhood Search (VNS) was proposed in1997 by Hansen and Mladenović [59]. VNS is an optimisation algorithm that explores different neighborhood structures to improve a given solution systematically. It relies on three iterative phases. First ,the shaking phase helps escape from local optima .Second, an improvement phase applies local search to enhance the solution. Finally, the neighborhood changes phase guides the algorithm while exploring the search space [60] [61].

The algorithm starts with an initial solution $\boldsymbol{S}$, and a set of neighborhood structures $\boldsymbol{N_L}$ is defined where **L =1, 2,…, Lmax.** . At each iteration, the initial value of **L** is set to **1**.

Subsequently, the algorithm iteratively explores the neighborhoods until the maximum neighborhood index is reached i.e. L= L_{max} .

During each iteration, a random solution $\boldsymbol{S_x}$ (shaken solution) is generated from the $\boldsymbol{L^{th}}$ neighborhood $\boldsymbol{N_L(S)}$ of current solution $\boldsymbol{S}$. A local search procedure applied to $\boldsymbol{S_x}$ to generate an improved solution $\boldsymbol{S_y}$. If $\boldsymbol{S_y}$ is better than the best-known solution $\boldsymbol{S_b}$, both $\boldsymbol{S_b}$ and $\boldsymbol{S}$ are updated with $\boldsymbol{S_y}$, and the search continues in the first neighborhood $\boldsymbol{N_1}$ .if no improvement is found, the search moves to the next neighborhood structure $\boldsymbol{N_{L+1}}$.

These operations are repeated until a termination criterion is satisfied. Finally, the algorithm returns the best solution found.

Algorithm 8:*Variable Neighborhood Search*

```
1:  $S \leftarrow$ initial solution;
2:  $N_L$, L = 1.2 .............. $L_{max}$ ;
3:  While the stop criterion is not met do
4:      $L \leftarrow 1$;
5:      While ($L < L_{max}$)
6:          $S_x \leftarrow$ Shaking($S, N_L$) ;
7:          $S_y \leftarrow$ Local Search($S_x$);
8:          if ( $f(S_y) < f(S_b)$)
9:              $S \leftarrow S_y$ ;
10:             $L \leftarrow 1$;
11:             $S_b \leftarrow S$ ;
12:         End
13:             $L \leftarrow L + 1$;
14:         End
15:     End
16: End
17: return $S_{best}$ ;
```

5) Iterated Local Search

Iterated Local Search (ILS) is a multi-start metaheuristic designed to overcome limitation of Random Restart approaches by incorporating advanced procedures [63]. Its main goal is to escape local optima by systematically modifying solutions and exploring different regions of the search space. This is achieved through strategic perturbations that guide the algorithm from one local optimum to another [62]. It operates in four phases [63]:

- First, an initial solution is constructed (**Generate an Initial Solution**).
- Next, a **Local Search** method is applied to improve the solution and find a local optimum.
- Then, a random **Perturbation** phase is applied to modify the current solution by altering some of its components.
- After that, the Local Search method is reapplied to the perturbed solution. If the resulting solution passes the **Acceptance Criterion**, it becomes the new current solution; otherwise, the algorithm reverts to the previous solution.

Algorithm 9:*Iterated Local Search*

```
1:   $S_n$ ← Generate Initial Solution ;
2:   $S^*$ ← Local Search ($S_n$) ;
3:   Repeat
4:       $S'$ ← Perturbation ($S^*$, history );
5:       $S''$ ← LocalSearch ($S'$);
6:       $S^*$ ← AcceptanceCriterion ($S^*$, $S''$, history);
7:  Until termination condition is met;
8:  return $S^*$ ;
```

b) Population-based metaheuristics

Population-based optimization methods are techniques that work on a population of solutions and are generally inspired by nature. They start with an initial population and, at each iteration, attempt to construct a new better population based on the previous one to converge toward good solution(s).

As examples of these methods, we can cite genetic algorithms, particle swarm optimization, ant colony algorithm, artificial bee colony algorithm, etc.

1) Genetic algorithm

Genetic algorithm (GA) is a method proposed by Holland in 1975[64]. The GA is stochastic algorithm inspired by the biological evolution theory, founded on genetics and natural selection (diversity, adaptation, inheritance). Each solution is represented as a chromosome composed of genes (solution elements). The algorithm uses selection to choose the solutions that will survive to the next generation, crossover to combine solutions to create Childs (new solutions) and mutation to introduce diversity. Over successive generations, these operations/processes work together to improve the overall quality of the solutions [65] [66].

The first step in GA is to Create an initial population P, typically consisting in randomly generated chromosomes .At each iteration, a set of | P | solutions is selected and saved in a new population Pn, called parents. The Crossover process is then applied to Pn to produce the set of solutions E (called Childs), which is then modified using mutation operator to introduce small random changes. Finally, the population P is updated by choosing the best solutions from both E and P for use in the following iteration.

Algorithm 10:*Genetic algorithm*

```
1:  $P \leftarrow$ Create an initial population ();
2:  While (the stopping criterion is met)
3:      $P_n \leftarrow$ Selection(P);
4:      E $\leftarrow$ Crossover($P_n$);
5:      E $\leftarrow$ Mutation(E);
6:      P $\leftarrow$ Replacement (E, P);
7:  End
8:  return the best solution found
```

i. Selection:

The Selection is the process of choosing parents solutions from current population P. each solution is selected based on its quality and can be chosen zero, one or multiple times. Here we present some well-known selection techniques:

- **Roulette selection:**

This method consists in randomly selecting a solution based on a probability proportional to the value of its objectives function, the selection probability $\boldsymbol{P_r}$**(s)** for a solution **S** is calculated as follows: $\boldsymbol{P_r(S) = \frac{f(s)}{\sum_1^{|p|} f(s\prime)}}$, $\boldsymbol{s'} \in$ P

If we are dealing with a minimization problem, then the selection probabilities must be transformed using the following equation: $\boldsymbol{P_r(S) = \frac{1 - P_r(S)}{|P|-1}}$

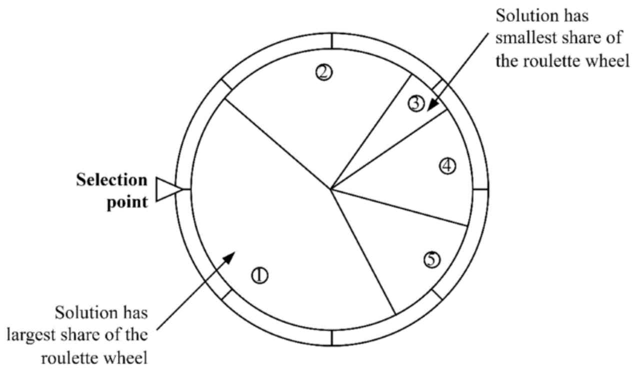

Solution has smallest share of the roulette wheel
Selection point
Solution has largest share of the roulette wheel
②
③
④
⑤
①

Figure 8:Roulette selection.

- **Rank selection:**

Rank selection is the modified form of Roulette wheel selection. It utilizes the ranks instead of fitness value. It is used when the value of solutions converges to similar values ,making it difficult to differentiate between them.

The first step is to rank the solutions based on their fitness, assigning (rank = 1) to the worst solution until **(rank = |P|)** to the best solution in the population. Then, the selection probability of each solution is calculated using the following formula: $\boldsymbol{P_r(S) = \frac{rank(S)}{\sum_1^{|p|} rank(s\prime)}}$, $\boldsymbol{s'} \in \boldsymbol{P}$

- **Tournament selection:**

This method consists in randomly selecting k solutions from population P where (K< $|\boldsymbol{P}|$). Then, the best of these solutions is selected.

- **Uniform selection:**

This method involves randomly selecting, giving all solutions an equal probability of being selected, which is :**1 / |P|** .

ii. Crossover:

The Crossover is a process applied to each pair of parents, and it consists in combining their original genes to produces new solutions, called children or offspring. It is applied on the parent population with a random probability P_c between 0.65 (65%) and 0.90 (90%). Here we present some of the well-known crossovers used in the literature :

- **Single-point crossover:** A random point is chosen to split each parent into two parts. The parts of the parents are swapped to produce two children as shown in the example.

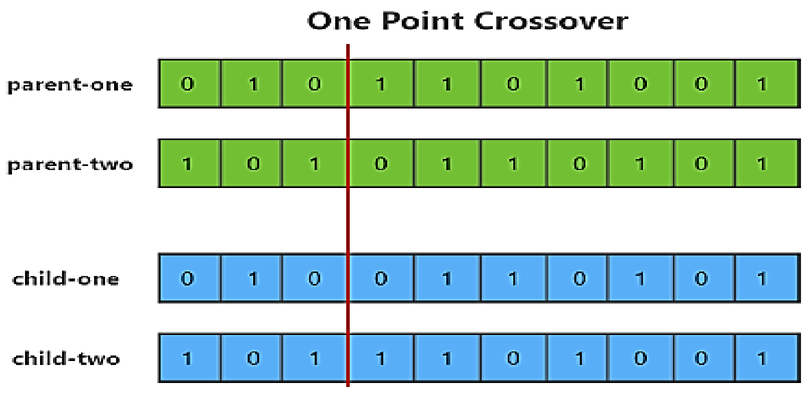


Figure 9:Single point crossover.

- **Two-point and K-point crossover:** randomly choose two or more crossover points to split the parents and then the parts of parents to produce children.

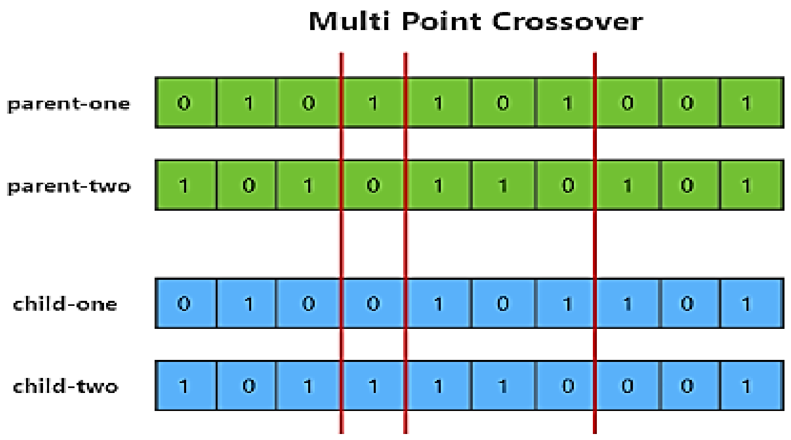


Figure 10:Three-point crossover.

- **Uniform crossover:** in this method, the parent is not split into parts. Instead, each gene is considered individually. We randomly decide if each gene is swapped with the corresponding gene in the other parent or not.

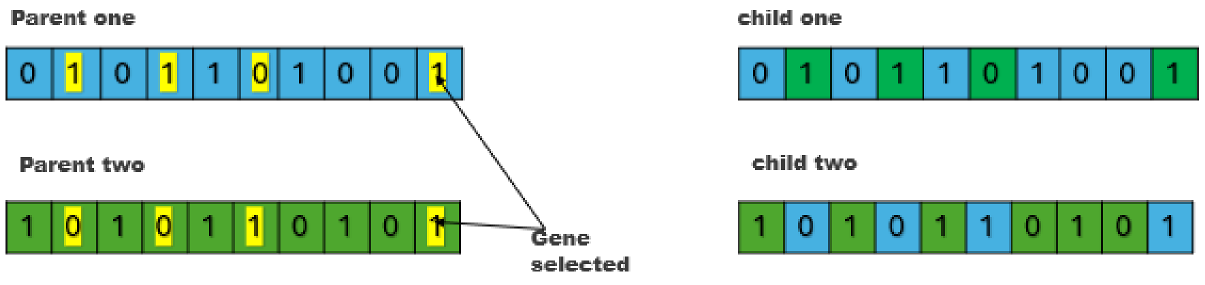


Figure 11:Uniform crossover.

iii. mutation:

The Mutation is a component of the genetic algorithm that introduces very small random changes to a solution. It ensures effective exploration of the search space and is applied to the population **E** with a probability$\mathbf{P_m}$ between **1%** and **5%.**

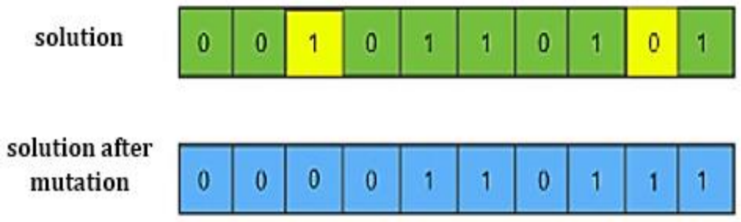


Figure 12:Simple mutation.

In the literature [66], several operators of mutation are used, such as :

- **Displacement mutation:** displaces a substring (part of the solution) of a solution to another random position within same solution. there are two variants of this mutation:
  - **Exchange mutation:** swap two randomly selected substring of the solution.

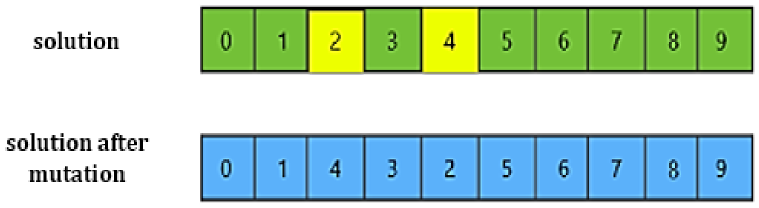


Figure 13:Exchange mutation.

- **Insertion mutation:** selected a random substring in the chromosome, remove it and re-insert it into a different position.

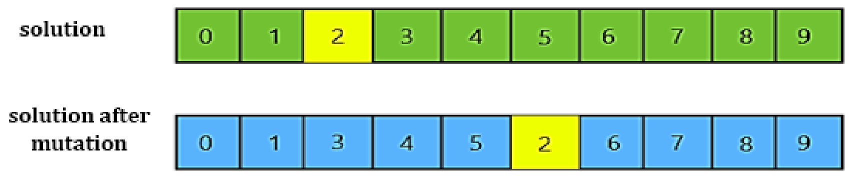


Figure 14:Insertion mutation.

iv. Replacement:

Replacement is the process of selecting the new population of the new iteration from the old population and the created children. Several methods can be used, in the following we cite three methods:

- **Complete replacement:** In this method, the children **E** is selected as the new population.
- **Combination with selection:** In this method the old population and the children (**E** and **P**) are combined. Then one of the selection methods is used to choose the new population.
- **Elitist method:** in this method the best solutions from **E** and **P** are selected.
-

2) Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a metaheuristic based on swarm intelligence, proposed in 1995 by Kennedy and Eberhart [67]. It is inspired by the collective behaviour of birds when moving in groups.

The algorithm starts with a population of solutions, called particles, where each solution is defined by a position and velocity in the search space. During the search process, each particle adjusts its position according to its current speed, its current position the best solution found in the previous iteration, and the best position identified by the set of particles. This update, carried out at each iteration, enables the particles to explore the search space and progress towards an optimal solution [68].

Algorithm 11: *Particle Swarm Optimization*

```
1:  Initialize_Speed (particles);
2:  Initialize_Position (particles);
3:  While (the stop criterion is not met)
4:      For (each particle p) do
5:          evaluate_fitness (p);
6:          update_Speed (p);
7:          update_Position (p);
8:          update_best (p);
9:      End
10:     update_best (particles);
11: End
12: Return the best solution found ;
```

3) Ant Colony Optimization

The Ant Colony Optimization (ACO) algorithm is a metaheuristic inspired by the cooperative behaviour of ants, introduced in the 1990s by Colorni, Dorigo, and Maniezzo. It is based on the ability of ants to explore their environment and communicate indirectly through pheromones to find the shortest path to a food source [69].

In ACO, first, a population of agents (the ants) explore the search space randomly. Then, when a promising solution is found, a quantity of pheromones is deposited on the corresponding path, reinforcing its attractiveness to other agents. The shorter and more efficient path is, the faster it accumulates pheromones, which accelerates convergence towards an optimal solution. In contrast, less efficient paths are gradually abandoned due to pheromone evaporation [68].

Algorithm 12:*Ant Colony Optimization*

```
1:  randomly initialize pheromone values;
2:  While (the stopping criterion is met)
3:      For (each Ant)
4:          construct a solution;
5:          update local pheromone values;
6:      End
7:  End
8:  Return the best solution found  ;
```

4) Artificial Bee Colony

The Artificial Bee Colony (ABC) algorithm is a metaheuristic introduced by Karaboga in 2005 [70], inspired by the foraging behaviour of honeybees. It simulates the bee's efficient process of finding food through cooperative communication, using a ″ waggle dance″ to share information about the direction, distance, and quality of food sources [68].

The algorithm models food sources as a potential solution and the bee colony as three types of agents [68] [71]:

Employed bees: Exploit known food sources and search for better alternatives nearby.

Onlooker bees: Observe employed bees' information and select food sources based on their quality.

Scout bees: Perform random searches for new food sources when the current ones are depleted.

The algorithm operates through an iterative cycle with three main phases:

Exploitation (Employed Bees Phase): Employed bees assess the fitness of their assigned food sources and share their findings.

Selection & Local Search (Onlooker Bees Phase): Onlookers probabilistically choose food sources based on shared nectar quality and refine existing solutions.

Exploration (Scout Bees Phase): Scouts randomly explore the search space for new solutions, replacing exhausted or poor-quality ones.

By balancing exploitation (improving known solutions) and exploration (discovering new ones), the ABC algorithm efficiently solves complex optimization problems.

Algorithm 13:*Artificial Bee Colony*

```
1: Initialize the bees;
2: Memorize the current best solution;
3: While (the stopping criterion is not met) do
4:     Employed bees phase;
5:     Onlooker Bees Phase;
6:     Scout Bees Phase;
7:     $S_b \leftarrow$ the best solution found so far;
8: End
9: Return $S_b$ ;
```

5) Cuckoo Search

Cuckoo Search (CS) is a metaheuristic, developed in 2009 by Yang and Deb, inspired by the parasitic reproductive behaviour of cuckoos. These birds lay their eggs in the nests of other species, entrusting the incubation and rearing of their chicks to host birds [72].

In the CS algorithm, a solution is represented by an egg in a nest, while a cuckoo egg symbolizes a new potential solution. The goal is to replace lower-quality solutions (eggs) with better alternatives (cuckoo eggs). This principle can be extended to more complex cases where each nest contains multiple eggs representing a set of solutions.

Standard Cuckoo Search follows three based rules:

- Each cuckoo lays one egg at a time and places it in a randomly chosen nest.
- The best nests, containing high-quality eggs, are preserved for future generations.
- The number of available host nests is fixed, and a cuckoo's egg is discovered by the host bird with a probability $p_a \in (0,1)$. if detected, the host bird either discards the egg or abandons the nest to build a new one.

To enhance the search efficiency, the algorithm incorporates Lévy flight (described below), which optimally explores the search space by combining large random jumps and small local variations. This ensures a balance between exploration and exploitation, making Cuckoo Search a good tool for solving complex optimization problems.

Lévy flight are a type of random walk which the step lengths follow a heavy-tailed probability distribution, typically a power law. They combine frequent short steps with rare but extremely long jumps. This behaviour allows for efficient exploration of large or complex search spaces. The structure of Lévy flight helps balance local exploitation and global exploitation, which is essential when navigating sparse or uncertain environments [73][74]

Algorithm 14:*Cuckoo Search*

```
1:  Define the objective function f (x);
2:  Generate initial population of n host nests $x_i$ ($i = 1 \dots n$);
3:  While (the stopping criterion is not met) do
4:      Generate a new solution (a cuckoo) using Lévy flight;
5:      Evaluate its fitness $f_i$ ;
6:      Randomly select a nest j from the population;
7:      If ($f_i > f_j$) then
8:          Replace $j$ with the new solution;
9:      End
10:     Abandon a fraction ($p_a$) of the worset nests and generate new ones;
11:     Retain the best solutions (nests with high-quality solutions);
12:     Rank the solutions and identify the current best;
13: End
14: Report the best solution found;
```

2.4.3 Hybridization

Hybridization involves combining multiple optimization methods to develop more efficient approaches. Hybrid metaheuristics are particularly suited for complex and real-world problems [75]. This subsection explores hybridization strategies, including the combination of metaheuristics with exact methods and the combination of between metaheuristics to enhance the performance of the developed algorithm.

1) Coupling metaheuristics with exact methods

Originally, the hybridization focuses on the collaboration between the metaheuristics [76]. However, several studied have shown that exact methods and metaheuristics can complement each other effectively [83]. Exact methods are well suited for solving small-scale problems and find optimal solutions to them, but they become impractical for large-scale ones due to their high computational cost. Therefore, hybrid approaches leverage the strengths of both: an exact method can be used to solve a sub-problem within a heuristic framework, while a heuristic can provide lower and upper bounds to guide an exact method [38].

2) Coupling metaheuristics with other metaheuristics

Combining metaheuristics is the most common form of hybridization in the literature [76]. Such synergy enhances the overall performance in solving optimization problems. Metaheuristics can be integrated in various ways, including the following widely used strategies [78], [79], [80]:

a) Parallel hybrids

Parallel hybrids involve multiple metaheuristics operating simultaneously. Each metaheuristic independently explores the search space and generates solutions, which are later compared to select the best. Parallelization improves search speed and solution quality for large-scale problems [78].

The authors in [77] highlight that parallelization strategies differ based on whether metaheuristics rely on single solutions or populations. For single solution based metaheuristics, three models are commonly used: **«Parallel Moves Model »** [79], **«Parallel Multi start Model»** [80], **«*Move Acceleration Model* »** [81].

Additionally, [83] categorizes parallel hybrids into two types:

- **Synchronous Parallel Hybrids:** one algorithm replaces a specific operator (e.g., substituting tabu search for mutation in a genetic algorithm).
- **Asynchronous Parallel Hybrids:** multiple algorithms exchange information dynamically during execution.

b) Sequential hybrids:

In Sequential hybrids, two methods execute consecutively, where the results of the first method serve as initial solutions for the second. This approach leverages the strengths of each method at different optimization stages.

2.5 Conclusion:

In this chapter, we explored various combinatorial optimization methods and algorithms. We began with exact methods, highlighting approaches such as branch and bound and branch and cut, which guarantee optimal solutions but remain computationally expensive for large-scale problems. Then, we introduced heuristic methods, including greedy algorithms and local search methods, which provide efficient solutions within reasonable time constraints.

In addition, we examined the metaheuristics, which offer powerful optimization frameworks for complex problems. Among the widely used metaheuristics, we presented simulated annealing (SA), tabu search (TS), variable neighborhood search (VNS), GRASP, genetic algorithms (GA), particle swarm optimization (PSO), ant colony optimization (ACO), etc. Finally, we discussed hybridization, which integrates different approaches enhance the performance and the adaptability of the developed algorithms to solve optimization problems.

Chapter 03: Heuristic algorithms for the Hub Location Routing Problem

3.1 Introduction

In this chapter, we present our proposed algorithms to solve hub location routing problem. The algorithms we propose are: (1) Simple Local Search, Tabu Search and Neighborhood Search with Random Selection (NSRS). First, we will present the definition of the HLRP as shown in [19]. Then, we will provide a review of the most related works that have addressed the HLRP. After that, we will present the solution methodologies we proposed to solve the problem, and we finish with a conclusion.

3.2 Problem definition

In our project, we deal with the variant of HLRP as defined by Lopes et Al. (2016).

- The hub location routing problem is an NP-hard problem that can be defined on an undirected complete simple graph G= (V, A), where V is the set of nodes V = {1, 2..., n} with |V|= n, and A is a set of edges such A= {(i, j) / i, j ∈ V, i ≠ j}. Each e ∈ A has a cost $C_e \in R+$.
- The objective is to select a subset of p nodes from V to be the hubs and we create local routes **R**= {$\boldsymbol{r1}$, $\boldsymbol{r2}$, $\boldsymbol{r3}$..., $\boldsymbol{rp}$} for these hubs. Each local route rk (for k = {1,2, 3…, P}) starts and ends at the hub k and visits a subset of nodes (all not-hub nodes must be visited by a local route of a hub). In addition, an Inter-hubs route is established to connect the hubs where each hub is visited one time.
- The length of a local route (including the non-hub) cannot exceed capacity C. More, a discount factor α is applied to the cost of arcs of the Inter-hubs route, where α ∈ [0,1].
- A feasible solution **s** is represented as a table of P vectors. Each vector of the table represents a local rout r_k where the first element is a hub. The total cost of the solution **s** is calculated using this formulation:

$$f(s) = \alpha\left[\left(\sum_{k=1}^{P-1} c_{r_k(1),r_{k+1}(1)}\right) + c_{r_P(1),r_1(1)}\right] + \sum_{k=1}^{P-1}\left[\left(\sum_{h=1}^{|r_k|-1} c_{r_k(h),r_k(h+1)}\right) + c_{r_k(|r_k|),r_k(1)}\right]$$

3.3 Most related works

In this subsection, we provide a review on the methods proposed in [19] [20] to deal with the HLRP:

In 2016 Lopes et Al. [19], several heuristic methods were introduced to address the problem:

1) **Multi-start Variable Neighborhood Descent (VND):** This approach iteratively applies a Variable Neighborhood Descent on a randomly generated initial solution. Two versions of VND are proposed, both used similar neighborhood structures, such as transferring non-hub nodes between local routes, swapping two non-hub nodes, and change hub node by non-hub nodes in same local route. Both variants employ the Lin-Kernighan heuristic for tour optimization and use random improvement moves rather than first or best improvement strategies. The difference between two approach is in the organization of neighborhood exploration:
   - **M-VND**: utilizes a nested Variable Neighborhood Descent.
   - **M-CNS**: Employs Consecutive Nested Neighborhood Search
2) **Biased Random-key Genetic Algorithm (BRKGA)**: A genetic algorithm where solutions are encoded as chromosomes using random keys in [0,1]. It uses a decoder to extract a solution from the chromosomes and attempts to improve the solution using a local search procedure. It implements a local search based on the Lin-Kernighan heuristic for TSP. This is a well-defined evolutionary process that uses parameterized uniform crossover and replaces the mutation operator applied to existing chromosomes with newly introduced mutants for exploration.
3) **Commercial Solver (Local Solver version 6.0)**: consists of a collection of techniques using a hybrid neighborhood search approach. Local Solver combines local search techniques, constraint propagation and inference methods, linear and mixed-integer programming techniques, as well as nonlinear programming techniques.

In 2022, Ratli et Al. [20], the authors used the **General Variable Neighborhood Search (GVNS)** algorithm, an advanced variant of the Variable Neighborhood Search (VNS), to deal with HLRP. The GVNS utilized seven neighborhood structures including: swapping non-hub nodes between local routes, swapping two non-hub nodes from two different local routes, replacing a hub node with non-hub node in the same local route, reversing a part of route, reversing a part of route consisting of two consecutive nodes (a simplified case of reversing a part of route), and inserting a node in either a forward or backward position in the same route .

The GVNS systematically alternates between two complementary phases:

- An **intensification phase**, which focuses on improving the current solution using the variable Neighborhood Descent (VND).
- A **diversification phase**, which introduces a shaking procedure to explore new regions of the search space.

These two phases are iteratively executed until the predefined stopping condition (T_{max}) is met.

Differences between the VNS algorithms used in [19] and [20]:

The main differences between GVNS, M-VND, M-CNS are as follows:

- **Number and organization of neighborhood structures**: GVNS explores seven neighborhood structures sequentially, while M-VND and M-CNS use only three neighborhoods.
- **Shaking phase**: GVNS employs a shaking procedure to escape from the local optima, while the M-VND generate a new random initial solution at the beginning of each iteration.
- **Improvement phase**: GVNS uses the Basic sequential Variable Neighborhood descent (VND). In contrast, M-VND employs a nested VND, which incorporates the Lin-Kernighan heuristic while M-CNS also applies the Lin-Kernighan heuristic.

3.4 Proposed heuristic algorithms

In this subsection, we present the three heuristic algorithms we propose to solve the HLRP. First, we present the components of these methods, and then we present how these components are employed in the three algorithms.

3.4.1 Components of solution Methodology

1) **Initial solution procedure**

The initial solution procedure starts with an empty solution ***s*** (table of **p** vectors) and creates a complete feasible solution at random. First, it selects randomly from **p** hubs the set of nodes

V, and creates a local route for each one. Then, it keeps randomly selects one node from **V** and add it in a random position into a random local route until all nodes in **V** are assigned. The initial solution procedure respects the local routes capacity constraint, therefore it doesn't assigns a new node to a local route only if its capacity allows that.

Algorithm 15:*initial_random_solution (p, C, V)*

```
1:  Create empty solution: table S [] of p vectors
2:  for i = 1 to p do
3:      select randomly node v ∈ V ;
4:      S [i].$r(1)$ ← v ;/* insert v as a hub in the first position of the local route S [i].$r$ */
5:      V ← V − {v} ; /*remove v from V*/
6:  End
7:  While (V is not empty)
8:      select randomly a node v ∈ V ;
9:      i ← random(1, P);
10:     If (|S [i].$r$| < C) then
11:         S [$i$].$r$(|S [i].$r$| + 1) ← v; /*add the non-hub v to local routeS [i].$r$ */
12:         V ← V − {v} ; /*remove v from V*/
13:     End
14: End
15: Return S;
```

2) Neighborhood structures

In our methods, we consider eight neighborhood structures where each neighborhood structure contains a distinct set of neighboring solutions generated based on a specific move. The neighborhood structures are named N1, N2, ... N8 and they are created based on the moves **intra-route Swap two non-hub, swap two hubs, intra-route Swap hub by non-hub, remove and add (shift) node, extra-route Swap two non-hub, extra-route Swap hub by non-hub, swap two local routes and remove and add (shift) local route** respectively. In the following, we will describe these moves:

i. **Intra-route Swap two non-hub (N1):** This movement consists of exchanging two nodes non-hubs in the same local route. This move is applied on all local routes of the solution. In algorithm 16, we depict, as an example, how the neighbors solutions related to this move are generated.

Algorithm 16: *N1(S)*

```
1:  N1 ← ∅ ;
2:  for i = 1 to p do
3:      for j = 2 to |S [i].$r$| do
4:          for k = j+1 to |S [i].$r$| do
5:              S’ ← swap$(S , S [i].r(j), S [i].r(k))$;
6:              N1 ← N1 ∪ { S’ };
7:          End
8:      End
9:  End
10: Return N1 ;
```

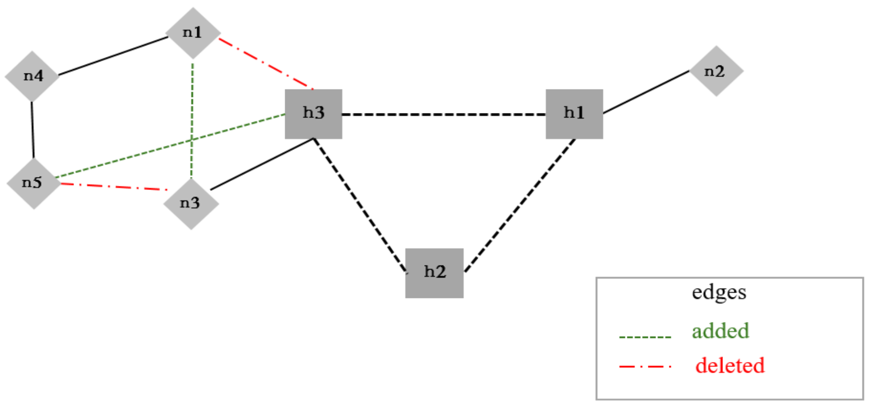


Figure 15:Swap two non-hubs (intra-route) of N1 (n1, n5).

ii. **Swap two hubs (N2):** This movement consists of exchanging two hubs in the solution.

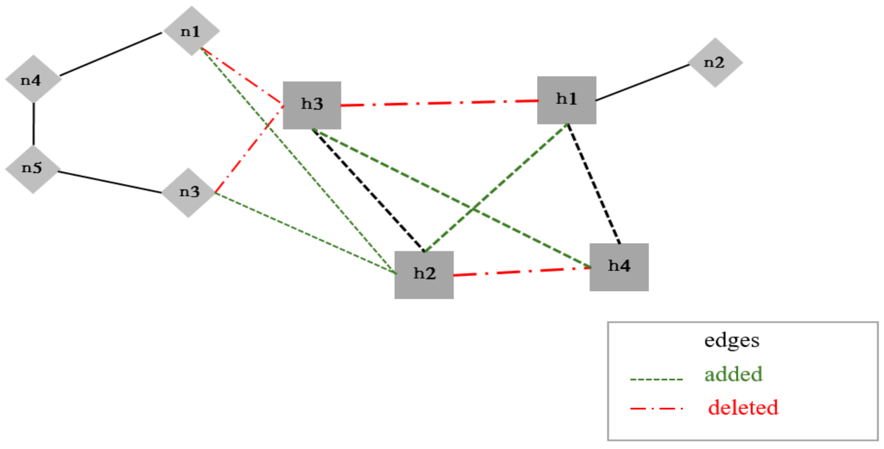


Figure 16:Swap two hubs of N2 (h2, h3).

iii. **Intra-route Swap hub by non-hub (N3):** This movement consists of exchanging the hub node with a non-hub node in the same local route. This move is applied on all local routes of the solution.

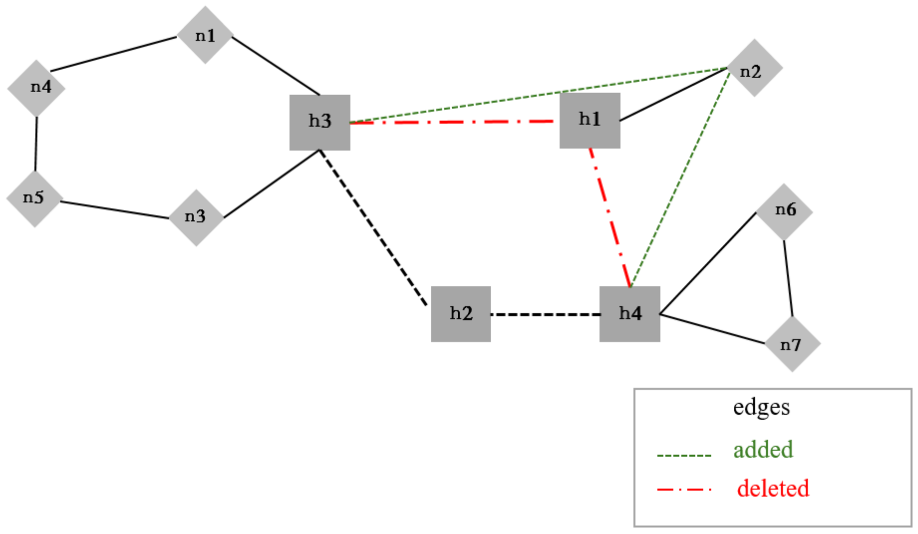


Figure 17:Swap hub by non-hub (intra-route) of N3 (h1 ,n2).

iv. **Remove and add (shift) node (N4):** this movement consists of removing a node (hub or non-hub) from its position and re-inserting it in other position in solution. This move is applied on all local routes of the solution.

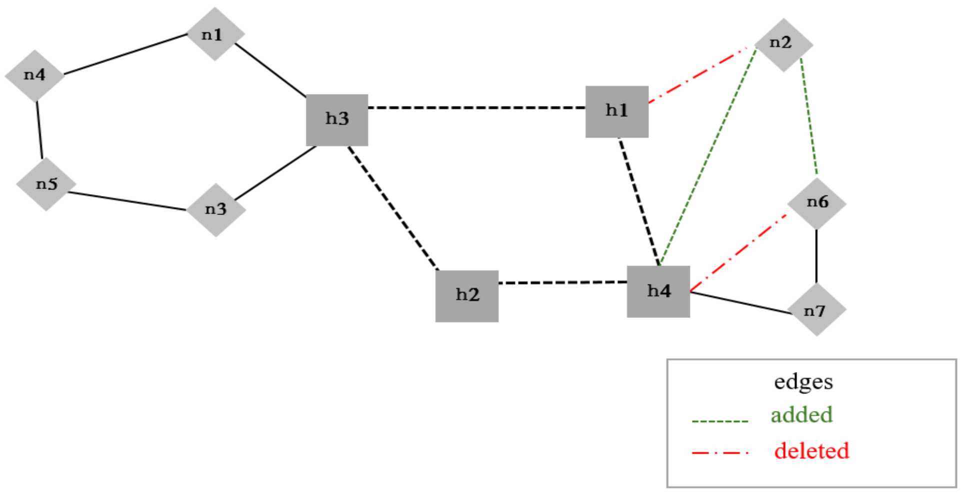


Figure 18:Remove and add (shift) node of N4 (n2).

v. **Extra-route Swap two non-hub (N5):** This movement consists of exchanging two node non-hub in the different local route. This move is applied on all combinations of two local routes in the solution.

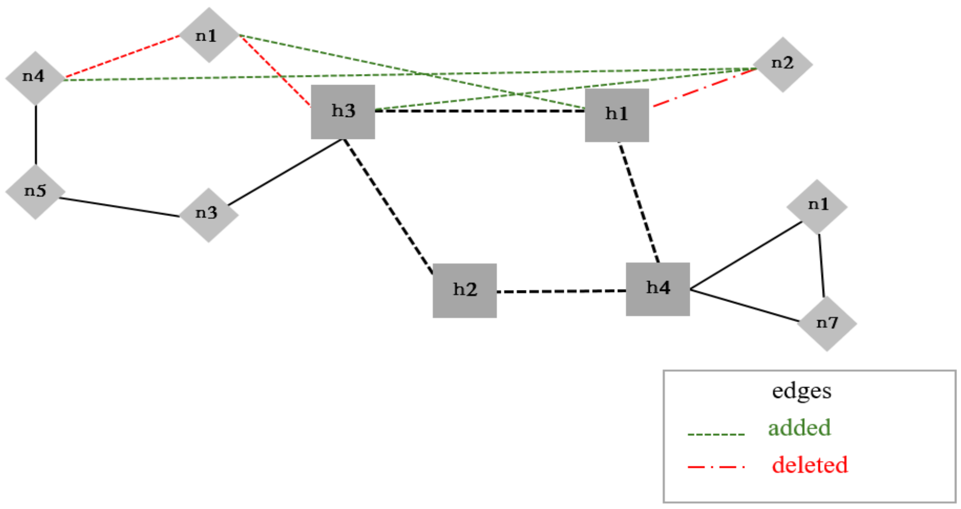


Figure 19:Swap two non-hub (extra-route) of N5 (n1, n2).

vi. **Extra-route Swap hub by non-hub (N6):** This movement consists of exchanging hub node by non-hub node in the different local route. This move is applied on all combinations of two local routes in the solution.

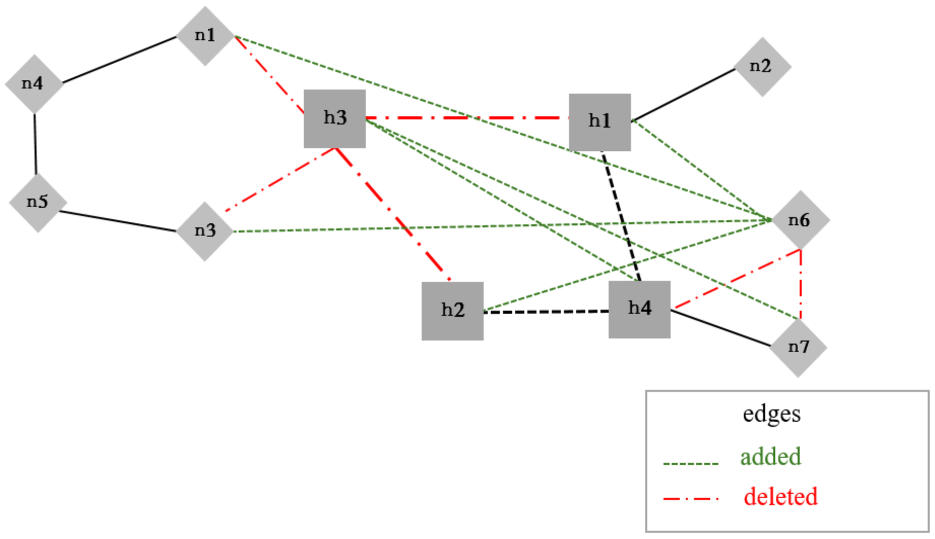


Figure 20:Swap hub by non-hub (extra-route) of N6 (h3 , n6).

vii. **Swap two local routes (N7):** This movement consists of exchanging two local routes in solution. This move is applied on all combinations of two local routes in the solution.

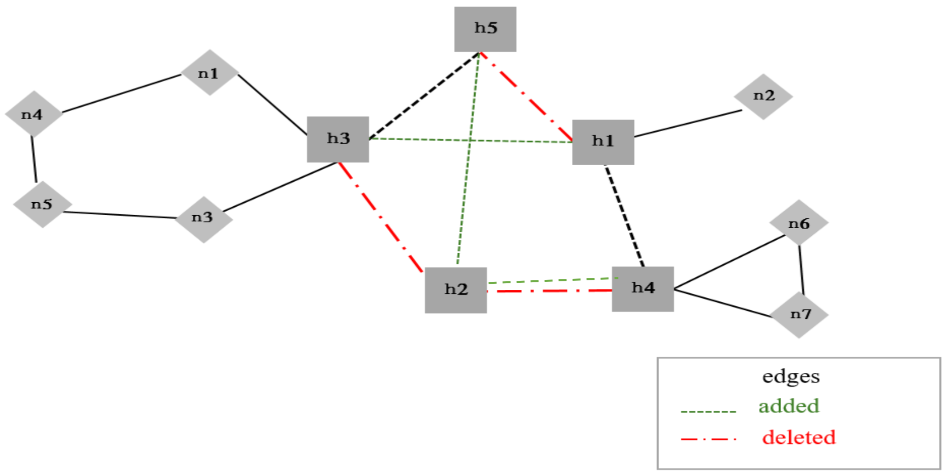


Figure 21:Swap two local routes of N7.

viii. **Remove and add (shift) local route (N8):** this movement consists of removing a local route from its position and inserting in other position. This move is applied on all local routes of the solution.

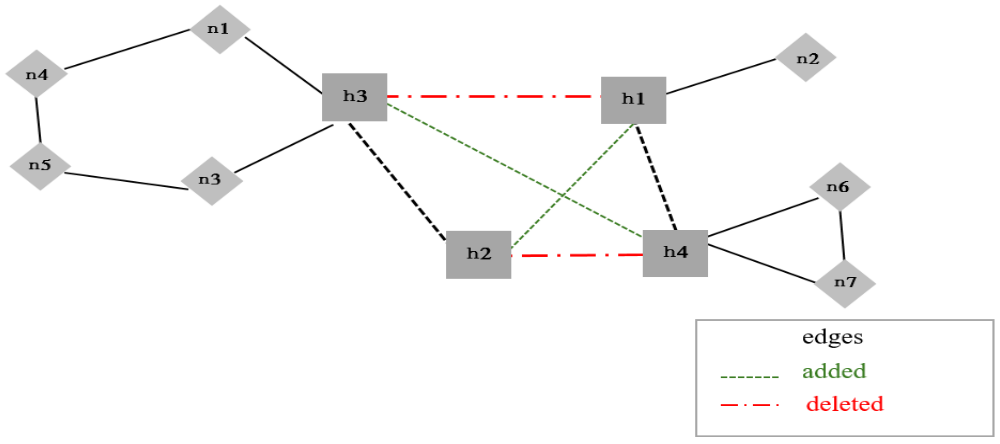


Figure 22:Remove and add (shift) local route of N8 .

3.4.2 Proposed heuristic algorithms

1) Simple Local Search

The first method we propose is a Simple Local Search (SLS) method. The SLS starts with an initial solution ***S*** generated at random using the procedure described above "**initial_random_solution**". Then, it iteratively improves this solution until reaching a local optimum. In each iteration, it creates the neighborhood structure ***N***(***S***) of the solution **S**, where N(S) = N1(S) U N2(S) …N8(S). After that, the SLS selects the best solution ***S***' In ***N***(***S***). If ***S***' is better than ***S***, then it replaces it otherwise, the procedure terminates (this case is when the best solution in the current neighborhood is the current solution, that means $S = S_{best}$ **).** The SLS is presented in Algorithm 17.

Algorithm 17:*Simple Local Search (S)*

```
1:  stop ← false ;
2:  While (not stop) do
3:      $N \leftarrow \{ N_1(S) \cup N_2(S) \cup N_3(S) \cup N_4(S) \cup N_5(S) \cup N_6(S) \cup N_7(S) \cup N_8(S) \}$
4:      S' ← choose the best solution in set $N(S)$;
5:      If ($f$(S') < $f(S_{best})$) then
6:          $S_{best}$ ← S';
7:      End
8:      If ( $S_{best} = S$ ) then
9:          Stop ← true ;
10:     End
11:     $S$ ← S';
12: End
13: Return $S_{best}$;
```

2) Tabu Search

The tabu search algorithm starts with an initial solution ***S*** generated at random using the procedure described above "**initial_random_solution**" and an empty tabu list **L**. at each iteration, it generates a set of neighbors solution ***N(S).*** After that, the TS selects the best solution ***S′ in N(s)*** where ***S'*** must be not-tabu solution, that means must not be in ***L***. Then, ***S'*** becomes the new current solution and we add it to the tabu list **L**. if ***S′*** is better than S_{best} it replaces it. The algorithm terminates after a fixed number of iterations.

Algorithm 18: *Tabu Search(S)*

```
1:  L ← ∅; $S_{best}$ ← $S$;
2:  While (iteration_number > 0) do
3:    $N$ ←{ $N_1(S) \cup N_2(S) \cup N_3(S) \cup N_4(S) \cup N_5(S) \cup N_6(S) \cup N_7(S) \cup N_8(S)$ }
4:    $S'$ ← choose the best solution in set N, { $S' \in$ N and $S' \notin$ L};
5:    Update tabu list L;/*add $S'$ into tabu list L*/
6:    If ($f$(S') < $f(S_{best})$) then
7:      $S_{best}$ ← S;
8:    End
9:    $S$ ← S';
10:   iteration_number ← iteration_number - 1;
11: End
12: Return $S_{best}$;
```

3) Neighborhood Search with Random Selection

The Neighborhood Search with Random Selection (NSRS) iteratively improves the initial solution S , (generated at random using the procedure described above "**initial_random_solution**") by exploring neighborhood structures through the "**Apply_All_Neighborhood**" function below. The **Apply_All_Neighborhood** function, at each iteration, randomly selects a neighborhood structure k from the set of all neighborhood structures L . Then, it generates the set of possible solutions for the selected $\mathbf{N_k(S)}$chooses the best solution from this set and replaces the current solution S found by this best solution. This process continues until all eight neighborhood structures have been explored and, at the end, it returns the best solution found. Finally, the NSRS terminates after a fixed number of iterations.

Algorithm 19:*Neighborhood Search with Random Selection (S)*

```
1:  $S_{best}$ ← $S$;
2:  While (iteration_number > 0 ) do
3:    $S$ ← Apply_All_Neighborhood (S);
4:    If ($f(S)$ < $f(S_{best})$) then
5:      $S_{best}$ ← S;
6:    End
7:    iteration_number ← iteration_number - 1;
8:  End
9:  Return $S_{best}$;
```

Algorithm 20:*Apply_All_Neighborhood (S)*

```
1: L ← { $N_1$ , $N_2$ , $N_3$, $N_4$, $N_5$, $N_6$, $N_7$, $N_8$ }; $N$ ←$\emptyset$ ;
2: While (L ≠ $\emptyset$) do
3:     Choose randomly $N_k$ from list L;
4:     N ← $N_k(S)$;
5:     $S$ ← choose the best solution in set N(S);
6:     L ← L - { $N_k$ };
7: End
8: Return $S$;
```

3.5 Conclusion

In this chapter, we have presented the heuristic algorithms we proposed to solve the Hub Location Routing Problem (HLRP). The proposed methods include Simple Local Search, Tabu Search, and Neighborhood Search with random Selection. Starting with the definition of the HLRP, we provided a detailed overview of the problem's formulation including the objective function, the constraints and the decision variables. After that, we reviewed the most relevant works and existing methodologies addressing the HLRP. Then, we introduced the proposed heuristic approaches, starting by the presentation of their components such as the initial solution procedure, neighborhood structures. After the presentation of the components, we presented each method and how it employed these components. In the next chapter, the proposed algorithms will be tested using benchmark data set from the literature and the obtained results will be compared with those of the most related works.

Chapter 04: Tests and Comparison with the literature

4.1 Introduction

In this chapter, we will evaluate the effectiveness of the algorithms we proposed through a series of experiments. We will begin with a detailed description of the dataset provided by [19], along with the method used for its generation. Next, we will present an example from the set of instances. Finally, we will report the results obtained by our developed algorithms and compare them with those achieved by state-of-the-art methods from the literature.

4.2 Description of benchmark

In [19], the authors derived a dataset of instances for HLRP using a set of 77 instances from TSPLIB as follows:

First, let define the following parameters:

n: Total number of nodes in the network.

p: Number of hubs.

C: Maximum capacity of each local route.

α: Discount factor applied to inter-hub transportation costs.

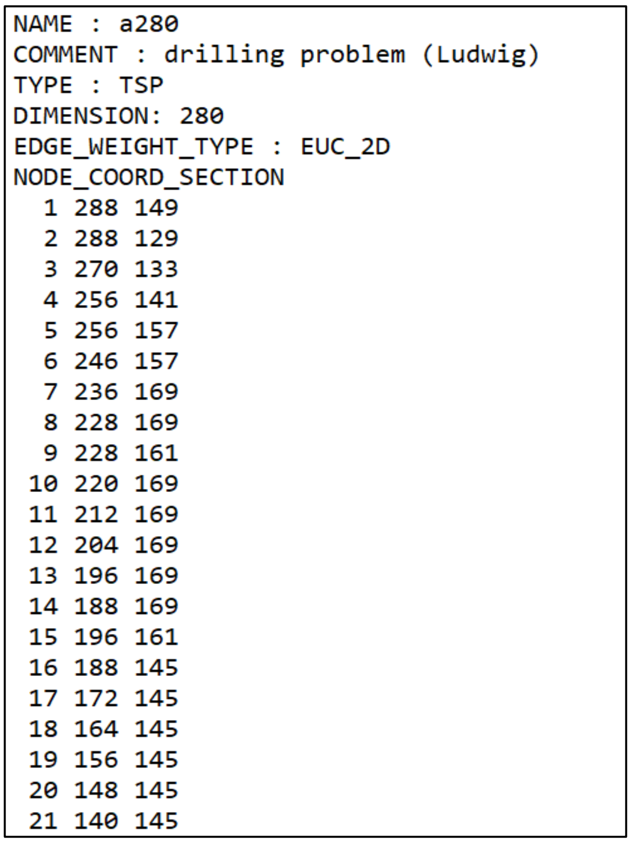
To diversify the experimental conditions, three scenarios were used to define ***p*** and ***C:***

- Scenario *ST:* $\mathrm{p} = \lceil 0.2\mathrm{n} \rceil$ and $C = \left\lceil \frac{n}{p} \right\rceil$: this scenario enforces tight local routes, ensuring that every hub at least one associated local route.
- Scenario *SL*: p as in *ST* and $C = \left\lfloor 1.8 \left\lceil \frac{n}{p} \right\rceil \right\rfloor$: this scenario allows large local routes, meaning some hubs may not have an associated local route.
- Scenario *SQ*: $p = C = \lceil \sqrt{\mathrm{n}} \rceil$: this scenario allows large local routes and hubs may have no local route associated with them.

Then, for each scenario, we set ***α*** equals to {0.2, 0.4, 0.6, 0.8}.

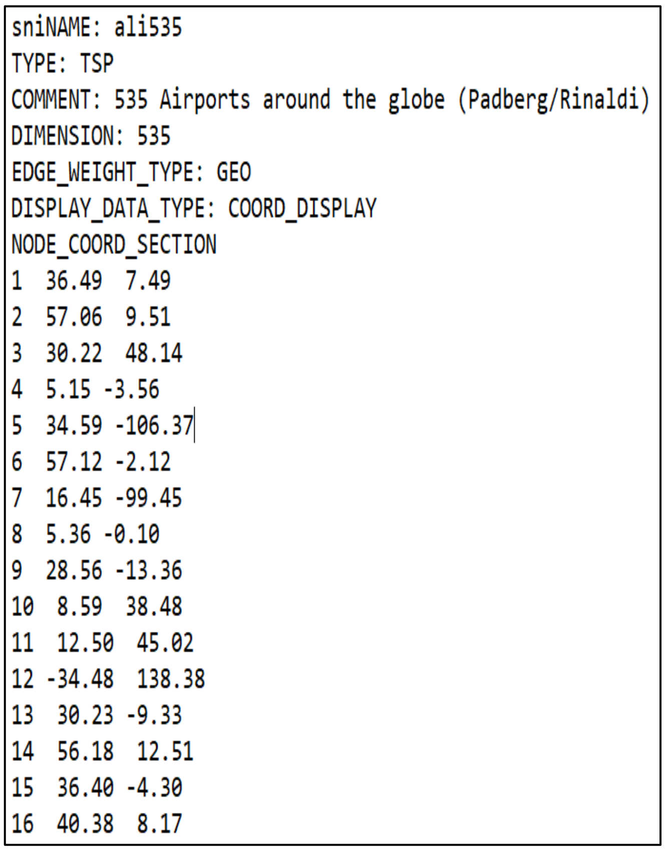
Based on the number of nodes n in each instance, they classified the dataset into two categories: *small instances* consist of 28 instances with less than 100 nodes, and *large instances* consists of 49 instances containing between 100 and 1000 nodes. All instances are available at the following address: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.

The following figures show examples of instances with varying distances and formats:



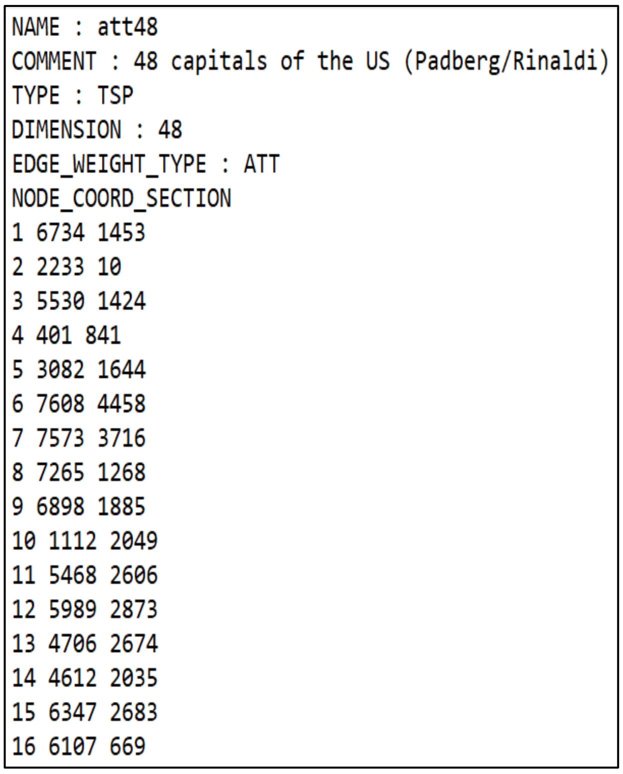
```
NAME : a280
COMMENT : drilling problem (Ludwig)
TYPE : TSP
DIMENSION: 280
EDGE_WEIGHT_TYPE : EUC_2D
NODE_COORD_SECTION
  1 288 149
  2 288 129
  3 270 133
  4 256 141
  5 256 157
  6 246 157
  7 236 169
  8 228 169
  9 228 161
 10 220 169
 11 212 169
 12 204 169
 13 196 169
 14 188 169
 15 196 161
 16 188 145
 17 172 145
 18 164 145
 19 156 145
 20 148 145
 21 140 145
```

Figure 23:Example of an instance using Euclidean distance.



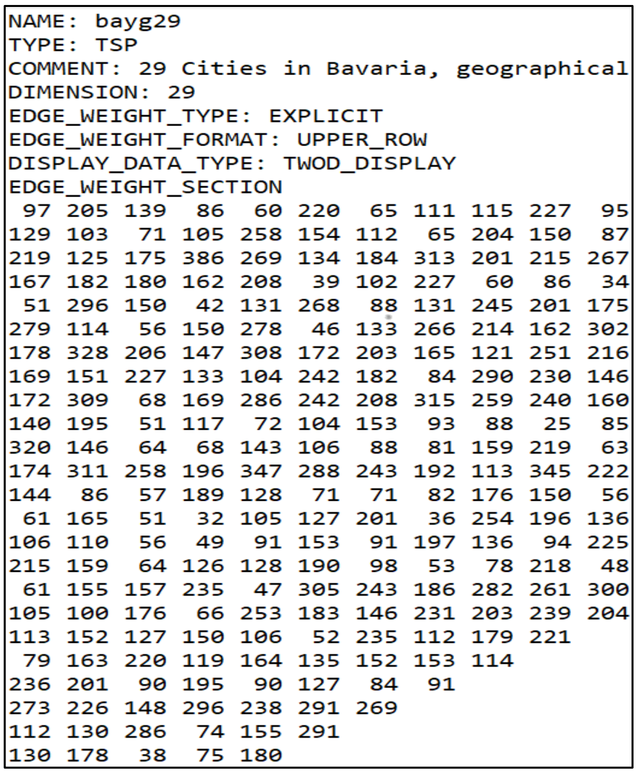
```
sniNAME: ali535
TYPE: TSP
COMMENT: 535 Airports around the globe (Padberg/Rinaldi)
DIMENSION: 535
EDGE_WEIGHT_TYPE: GEO
DISPLAY_DATA_TYPE: COORD_DISPLAY
NODE_COORD_SECTION
1  36.49  7.49
2  57.06  9.51
3  30.22  48.14
4  5.15 -3.56
5  34.59 -106.37
6  57.12 -2.12
7  16.45 -99.45
8  5.36 -0.10
9  28.56 -13.36
10  8.59  38.48
11  12.50  45.02
12 -34.48  138.38
13  30.23 -9.33
14  56.18  12.51
15  36.40 -4.30
16  40.38  8.17
```

Figure 24:Example of an instance using Geographical distance.



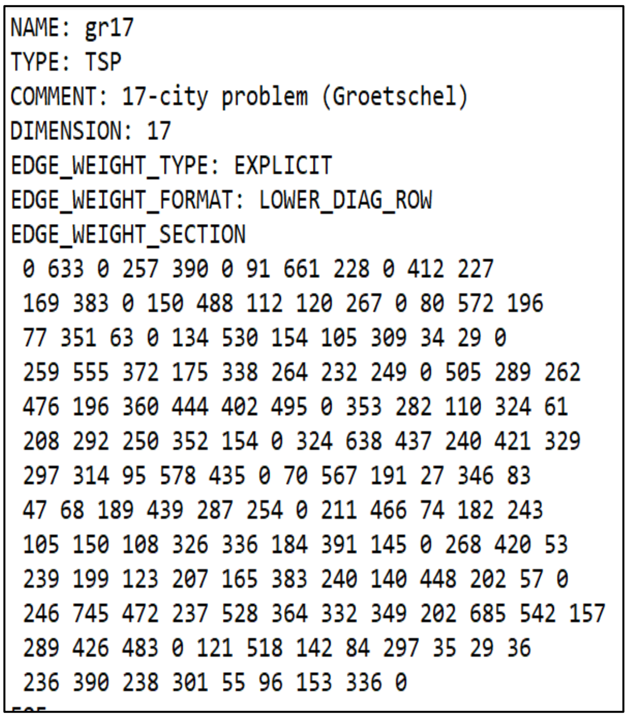
```
NAME : att48
COMMENT : 48 capitals of the US (Padberg/Rinaldi)
TYPE : TSP
DIMENSION : 48
EDGE_WEIGHT_TYPE : ATT
NODE_COORD_SECTION
1 6734 1453
2 2233 10
3 5530 1424
4 401 841
5 3082 1644
6 7608 4458
7 7573 3716
8 7265 1268
9 6898 1885
10 1112 2049
11 5468 2606
12 5989 2873
13 4706 2674
14 4612 2035
15 6347 2683
16 6107 669
```

Figure 25:Example of an instance using ATT distance.



```
NAME: bayg29
TYPE: TSP
COMMENT: 29 Cities in Bavaria, geographical
DIMENSION: 29
EDGE_WEIGHT_TYPE: EXPLICIT
EDGE_WEIGHT_FORMAT: UPPER_ROW
DISPLAY_DATA_TYPE: TWOD_DISPLAY
EDGE_WEIGHT_SECTION
 97 205 139  86  60 220  65 111 115 227  95
129 103  71 105 258 154 112  65 204 150  87
219 125 175 386 269 134 184 313 201 215 267
167 182 180 162 208  39 102 227  60  86  34
 51 296 150  42 131 268  88 131 245 201 175
279 114  56 150 278  46 133 266 214 162 302
178 328 206 147 308 172 203 165 121 251 216
169 151 227 133 104 242 182  84 290 230 146
172 309  68 169 286 242 208 315 259 240 160
140 195  51 117  72 104 153  93  88  25  85
320 146  64  68 143 106  88  81 159 219  63
174 311 258 196 347 288 243 192 113 345 222
144  86  57 189 128  71  71  82 176 150  56
 61 165  51  32 105 127 201  36 254 196 136
106 110  56  49  91 153  91 197 136  94 225
215 159  64 126 128 190  98  53  78 218  48
 61 155 157 235  47 305 243 186 282 261 300
105 100 176  66 253 183 146 231 203 239 204
113 152 127 150 106  52 235 112 179 221
 79 163 220 119 164 135 152 153 114
236 201  90 195  90 127  84  91
273 226 148 296 238 291 269
112 130 286  74 155 291
130 178  38  75 180
```

Figure 26:Example of an instance with UPPER ROW format.



```
NAME: gr17
TYPE: TSP
COMMENT: 17-city problem (Groetschel)
DIMENSION: 17
EDGE_WEIGHT_TYPE: EXPLICIT
EDGE_WEIGHT_FORMAT: LOWER_DIAG_ROW
EDGE_WEIGHT_SECTION
 0 633 0 257 390 0 91 661 228 0 412 227
 169 383 0 150 488 112 120 267 0 80 572 196
 77 351 63 0 134 530 154 105 309 34 29 0
 259 555 372 175 338 264 232 249 0 505 289 262
 476 196 360 444 402 495 0 353 282 110 324 61
 208 292 250 352 154 0 324 638 437 240 421 329
 297 314 95 578 435 0 70 567 191 27 346 83
 47 68 189 439 287 254 0 211 466 74 182 243
 105 150 108 326 336 184 391 145 0 268 420 53
 239 199 123 207 165 383 240 140 448 202 57 0
 246 745 472 237 528 364 332 349 202 685 542 157
 289 426 483 0 121 518 142 84 297 35 29 36
 236 390 238 301 55 96 153 336 0
```

Figure 27:Example of an instance with LOWER DIAG ROW format.

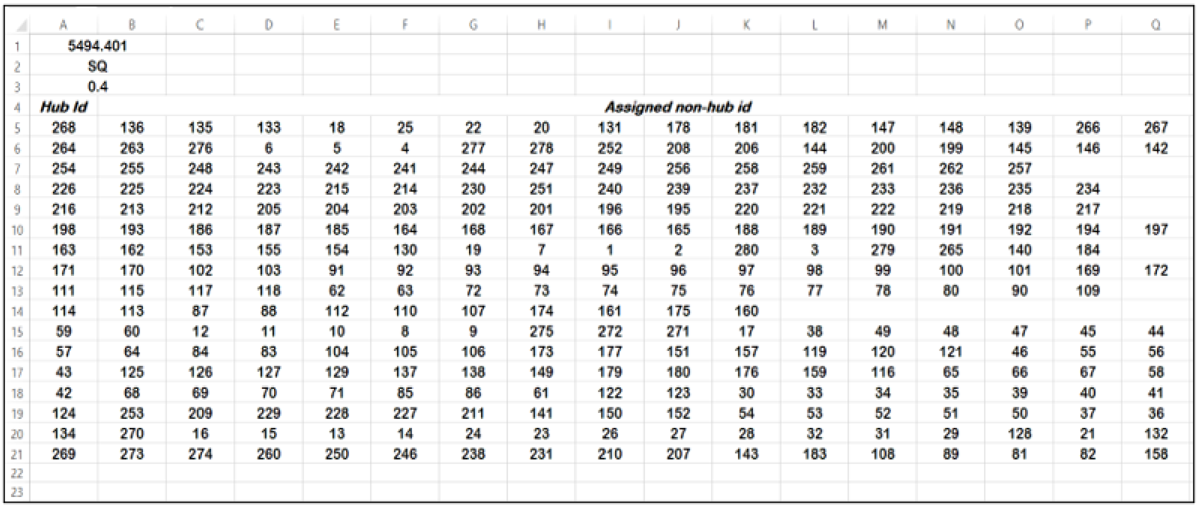
4.3 Generated solution structure

The generated solution contains the ids of the selected hubs, along with the ids of the non-hubs assigned to each hub. In the next table we present the structure of generated solution.

Fitness							
Scenario							
alpha							
Capacity local route							
1 2 3………………………….... ….C							
Nbr hub	hub Id	Assigned non-hub id					
1	5	8	55	19	25	.....	30
2	6	77	60	3	…..	18	/
3	20	4	88	65	67	.....	45
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
p	70	/	/	/	/	/	/

Table 1:Generated solution structure

We present the complete solution obtained by the Tabu Search algorithm for the a280 instance under scenario SQ with α = 0.4 in the following figure:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	5494.401																
2	SQ																
3	0.4																
4	***Hub Id***								***Assigned non-hub id***								
5	268	136	135	133	18	25	22	20	131	178	181	182	147	148	139	266	267
6	264	263	276	6	5	4	277	278	252	208	206	144	200	199	145	146	142
7	254	255	248	243	242	241	244	247	249	256	258	259	261	262	257		
8	226	225	224	223	215	214	230	251	240	239	237	232	233	236	235	234	
9	216	213	212	205	204	203	202	201	196	195	220	221	222	219	218	217	
10	198	193	186	187	185	164	168	167	166	165	188	189	190	191	192	194	197
11	163	162	153	155	154	130	19	7	1	2	280	3	279	265	140	184	
12	171	170	102	103	91	92	93	94	95	96	97	98	99	100	101	169	172
13	111	115	117	118	62	63	72	73	74	75	76	77	78	80	90	109	
14	114	113	87	88	112	110	107	174	161	175	160						
15	59	60	12	11	10	8	9	275	272	271	17	38	49	48	47	45	44
16	57	64	84	83	104	105	106	173	177	151	157	119	120	121	46	55	56
17	43	125	126	127	129	137	138	149	179	180	176	159	116	65	66	67	58
18	42	68	69	70	71	85	86	61	122	123	30	33	34	35	39	40	41
19	124	253	209	229	228	227	211	141	150	152	54	53	52	51	50	37	36
20	134	270	16	15	13	14	24	23	26	27	28	32	31	29	128	21	132
21	269	273	274	260	250	246	238	231	210	207	143	183	108	89	81	82	158
22																	
23																	

Figure 28:example of check by hand of a generated solution

4.4 Experiments results

In this section, we present the results obtained by running three algorithms (Simple Local Search, Tabu search and Neighborhood Search with Random Selection) on a selected subset of instances with different topologies. This subset consists of the following eight instances: burma14, gr14, berlin52, pr76, kroB100, ch130, a280, u574. Each instance is evaluated under three scenarios (*ST*, *SL*, and *SQ*) and with four values of α: {0.2, 0.4, 0.6, 0.8}.

4.4.1 Parameters and implementation details

All algorithms were implemented in java, using the java SE-17 (JDK 17) compiler. Our algorithms was executed on a computer equipped with an Intel(R) Core (TM) i5-8365U running at 1.60 GHz (with a maximum turbo frequency of 1.90 GHz) and with 16 GB of RAM.

In Tabu Search algorithm, we used the following parameters: we set the maximal number of iterations equals to 1000 iterations and the tabu list size equals to 20, whereas in Neighborhood Search with Random Selection we set the maximal number of iterations equals to 1000.

4.4.2 The obtained results

To evaluate the proposed algorithms, we run each one 10 times on each instance and we report: the value of the best solution found (Best), the average value (AVG) of the obtained solutions and the average of the running time (Time), aver all the 10 runs.

Instance		Simple Local Search			Tabu Search			NSRS		
		Best	AVG	Time	Best	AVG	Time	Best	AVG	Time
Small	burma14	3737.54	3778.76	0.03	3737.54	3772.61	0.03	3737.54	3772.61	1.22
	gr24	1482.8	1614.44	0.07	1469.6	1541.76	2.82	1469.6	1475.28	2.38
	berlin52	8907.91	9418.30	0.71	8836.67	9370.44	10.19	8570.24	8915.55	9.77
	pr76	138209.35	144315.34	2.11	138209.35	144093.12	21.55	123226.15	128342.53	21.14
	kroB100	32869.23	34137.14	4.26	32866.73	34109.90	26.96	28620.32	31278.00	26.69
Large	ch130	8618.74	9106.94	10.60	8702.55	9069.97	47.85	7796.88	8377.33	47.20
	a280	4058.1	4373.72	84.28	4058.1	4372.08	203.97	3665.51	3918.07	199.99
	u574	58173.4	62927.48	884.01	58310.76	63135.07	914.95	56908.06	61149.25	898.17
Average		32007.13	33709.02	123.26	32023.91	33683.12	153.54	29249.28	30903.58	150.82

Table 2:Obtained results for the ST scenario and α = 0.2.

Instance		Simple Local Search			Tabu Search			NSRS		
		Best	AVG	Time	Best	AVG	Time	Best	AVG	Time
Small	burma14	3943.82	4001.09	0.03	3943.82	3975.28	1.18	3943.82	3943.82	1.22
	gr24	1542.2	1666.86	0.07	1542.2	1637.26	2.48	1542.2	1556.82	2.49
	berlin52	9562.55	9737.12	0.63	9475.29	9710.73	9.99	9174.38	9534.81	10.12
	pr76	141780.8	154209.37	1.98	141780.83	153748.41	21.20	131225.87	137088.06	20.54
	kroB100	34918.94	36746.54	3.90	34440.65	36627.55	26.54	30944.71	34045.28	26.03
Large	ch130	9328.29	10065.93	8.70	9328.29	10065.79	43.71	8463.7	9493.31	43.59
	a280	4467.77	4888.99	88.40	4467.77	4876.80	663.90	3986.63	4312.86	212.51
	u574	63157.42	66939.28	951.93	63409.82	67033.13	931.53	62237.14	66330.58	821.48
Average		33587.72	36031.90	131.96	33548.58	35959.37	212.57	31439.80	33288.19	141.12

Table 3:Obtained results for the ST scenario and α = 0.4.

Instance		Simple Local Search			Tabu Search			NSRS		
		Best	AVG	Time	Best	AVG	Time	Best	AVG	Time
Small	burma14	4150.10	4191.1	0.042	4150.10	4150.63	1.55	4150.10	4150.10	1.420
	gr24	1665.6	1755.86	0.08	1635.6	1681.86	3.92	1614.8	1627.17	3.88
	berlin52	10238.33	10706.56	0.98	9935.26	10567.04	15.14	9889.63	10152.87	16.048
	pr76	154708.87	173030.49	2.70	154708.87	171481.92	29.00	141620.29	148466.57	29.30
	kroB100	35088.31	38318.33	5.14	34805.87	37853.46	35.09	32410.04	35399.54	33.62
Large	ch130	9695.53	10908.81	13.80	9695.53	10834.75	68.67	9626.22	10313.34	62.49
	a280	4393.77	5081.30	193.75	4388.06	5072.86	275.38	4312.54	4653.69	264.97
	u574	68456.06	73517.76	1302.86	68966.37	74013.43	1354.50	64743.83	70520.35	1481.84
Average		36049.57	39688.77	189.91	36035.70	39456.99	222.90	33545.93	35660.45	236.69

Table 4:Obtained results for the ST scenario and α = 0.6.

Instance		Simple Local Search			Tabu Search			NSRS		
		Best	AVG	Time	Best	AVG	Time	Best	AVG	Time
Small	burma14	4290.65	4399.46	0.038	4290.65	4295.79	1.95	4290.65	4290.65	1.85
	gr24	1702.2	1796.52	0.110	1676.6	1718.08	4.803	1676.6	1676.6	4.76
	berlin52	10606.21	11462.54	0.90	10606.21	11259.09	15.02	10328.04	10711.30	15.00
	pr76	168243.67	180005.78	2.24	167528.74	179488.18	25.00	148849.26	156477.86	25.84
	kroB100	37844.56	41225.56	5.49	37844.56	41113.76	35.01	35510.81	38935.05	34.05
Large	ch130	10620.66	11123.96	10.90	10535.32	11076.55	56.62	9650.92	10857.51	55.47
	a280	5127.74	5317.27	175.40	5033.11	5252.16	270.59	4554.66	4803.08	261.69
	u574	74218.14	79964.76	1305.38	74218.14	79964.66	1310.25	68688.43	75998.76	1448.79
Average		39081.72	41911.98	187.55	38966.66	41771.03	214.90	35443.67	37968.85	230.93

Table 5:Obtained results for the ST scenario and α = 0.8.

Instance		Simple Local Search			Tabu Search			NSRS		
		Best	AVG	Time	Best	AVG	Time	Best	AVG	Time
Small	burma14	2917.46	3215.3	0.03	2917.46	3215.3	1.63	2917.46	2917.46	1.35
	gr24	1321.2	1415.96	0.11	1223.2	1355.58	3.28	1192	1209.4	3.44
	berlin52	7430.3	8451.91	1.12	7417.51	8396.11	15.23	6823.71	7615.89	16.02
	pr76	114354.58	129496.82	3.55	114338.97	129092.52	32.43	105505.15	115073.53	32.38
	kroB100	25714.14	28984.00	3250.44	25656.22	28905.97	54.25	22864.09	25494.91	57.33
Large	ch130	7539.99	8125.27	19.20	7386.75	8080.77	91.05	6479.99	7066.76	104.40
	a280	3385.18	3925.86	229.76	3365.14	3915.04	429.49	3505.92	3889.82	399.60
	u574	50680.81	57806.73	2304.04	51202.02	59285.39	2173.12	50305.99	59253.18	1846.08
Average		26667.95	30177.73	726.03	26688.40	30280.83	350.06	24949.28	27815.12	307.57

Table 6:Obtained results for the SL scenario and α = 0.2.

Instance		Simple Local Search			Tabu Search			NSRS		
		Best	AVG	Time	Best	AVG	Time	Best	AVG	Time
Small	burma14	3240.65	3586.59	0.03	3199.23	3403	1.33	3199.23	3199.23	1.32
	gr24	1385.2	1483.64	0.11	1348.8	1441.5	3.42	1287	1308.56	3.49
	berlin52	3199.23	9074.19	1.11	8163.66	8869.50	16.02	7596.96	8124.41	15.80
	pr76	128209.56	141807.62	3.55	128209.56	140887.07	32.47	118403.42	129977.49	31.95
	kroB100	28427.45	30834.64	8.32	28413.75	30692.12	54.25	25117.7	27947.15	54.98
Large	ch130	7887.88	8591.53	19.83	7887.69	8586.79	104.48	6966.53	7753.95	97.31
	a280	4031.26	4396.67	230.86	4031.26	4391.00	443.05	3586.95	3390.95	427.27
	u574	55569.24	62359.33	2352.87	56557.6	64729.32	2151.35	55569.24	62956.00	1837.54
Average		28993.80	32766.77	327.08	29726.44	32875.04	350.79	27715.87	30582.22	308.71

Table 7:Obtained results for the SL scenario and α = 0.4.

Instance		Simple Local Search			Tabu Search			NSRS		
		Best	AVG	Time	Best	AVG	Time	Best	AVG	Time
Small	burma14	3422.18	3706.68	0.034	3422.18	3583.39	1.98	3422.18	3422.73	1.89
	gr24	1526.4	1598.08	0.63	1429.6	1512.80	5.77	1375.0	1394.0	7.15
	berlin52	8997.58	10279.99	1.73	8703.78	10136.31	22.76	8153.27	8727.42	23.96
	pr76	135880.57	154672.92	4.53	134555.88	150206.56	44.42	128004.44	135461.46	44.76
	kroB100	30006.92	34377.05	14.42	29912.70	33480.32	91.68	27124.35	31273.28	81.89
Large	ch130	8606.98	9255.82	25.81	8606.98	9258.11	131.37	8255.19	9152.87	123.15
	a280	3992.80	4460.77	359.16	3990.00	4414.92	634.10	3907.38	4442.41	567.91
	u574	61703.38	66467.76	2459.69	64334.75	69023.15	9101.52	63666.32	68857.05	1902.89
Average		31767.10	35602.38	358.25	31869.48	35201.94	1254.2	30488.51	32841.40	344.2

Table 8: Obtained results for the SL scenario and α = 0.6.

Instance		Simple Local Search			Tabu Search			NSRS		
		Best	AVG	Time	Best	AVG	Time	Best	AVG	Time
Small	burma14	3522.45	3819.53	0.031	3522.45	3590.30	1.92	3522.45	3522.45	1.84
	gr24	1516.8	1614.69	0.18	1464.2	1525.46	7.16	1452.4	1462.84	7.46
	berlin52	9685.86	10538.14	1.40	9658.85	10315.08	19.87	8828.43	9642.35	21.09
	pr76	147901.68	166813.99	4.51	147901.68	164017.24	40.53	135312.50	146436.27	41.16
	kroB100	31403.37	34123.11	11.97	31108.09	33867.94	85.82	29894.98	34033.76	82.88
Large	ch130	9013.38	10333.39	25.82	9013.38	10223.45	124.05	8462.70	9923.82	119.07
	a280	4301.46	4734.76	322.59	4232.38	4674.60	620.89	4365.87	4655.54	556.34
	u574	69125.64	73793.59	2522.82	73106.82	77905.92	2225.65	69042.3	75923.32	1855.22
Average		34558.83	38221.40	361.16	35000.98	38264.99	390.73	32610.20	35700.04	335.63

Table 9:Obtained results for the SL scenario and α = 0.8.

Instance		Simple Local Search			Tabu Search			NSRS		
		Best	AVG	Time	Best	AVG	Time	Best	AVG	Time
Small	burma14	3527.47	3625.07	0.02	3527.47	3607.38	1.15	3527.47	3527.47	1.09
	gr24	1505.8	1593.98	0.07	1469.6	1563.8	2.36	1469.6	1472.1	2.3267
	berlin52	8914.31	9847.79	0.86	8914.31	9760.20	12.06	8265.47	8419.37	12.7749
	pr76	139511.24	165427.28	2.4368	139452.73	164367.95	20.31	127465.15	132961.69	21.42
	kroB100	33366.75	42722.15	4.403	33366.75	42281.59	26.48	30790.97	35638.47	25.87
Large	ch130	9534.59	13242.91	1213.33	9534.59	11206.32	58.80	7243.15	9120.03	59.53
	a280	5245.01	6051.29	129.80	5209.87	5938.97	244.64	4342.99	5183.33	235.53
	u574	79208.72	98980.19	1225.02	102933.65	99334.24	863.73	83330.89	90340.10	824.24
Average		35101.73	42686.33	321.99	38051.12	42257.56	153.69	33304.46	35832.82	147.85

Table 10:Obtained results for the SQ scenario and α = 0.2.

Instance		Simple Local Search			Tabu Search			NSRS		
		Best	AVG	Time	Best	AVG	Time	Best	AVG	Time
Small	burma14	3573.87	3851.72	0.02	3573.87	3851.72	1.12	3527.47	3727.18	1.11
	gr24	1561	1649.76	0.07	1542.2	1618.6	2.40	1542.2	1557.04	2.43
	berlin52	9418.44	10598.49	0.83	9334.81	10354.40	11.01	8621.69	9153.18	12.57
	pr76	151865.95	174004.59	2.21	144409.6	165058.88	20.11	128385.64	137694.95	21.06
	kroB100	36376.67	43551.21	3.97	34127.29	41431.53	25.75	29568.58	35312.46	25.71
Large	ch130	9618.51	10787.56	12.69	9590.7	10762.68	58.60	7747.34	9843.57	59.58
	a280	5156.04	6193.513	123.03	5150.89	6115.86	238.32	4463.39	5369.33	231.68
	u574	90602.73	101973.23	1214.41	106121.1	113622.03	888.40	84512.8	88741.23	843.76
Average		38521.65	44076.26	169.65	39231.30	44101.96	155.71	33546.03	36424.87	149.74

Table 11:Obtained results for the SQ scenario and α = 0.4.

Instance		Simple Local Search			Tabu Search			NSRS		
		Best	AVG	Time	Best	AVG	Time	Best	AVG	Time
Small	burma14	3971.29	4102.08	0.019	3971.29	4094.06	1.52	3971.29	3971.29	1.52
	gr24	1636.2	1732.84	0.078	1614.8	1665.14	3.12	1614.8	1626.43	3.01
	berlin52	9417.57	10573.45	1.08	9321.80	10278.55	16.62	8843.88	9317.89	16.81
	pr76	152594.51	177713.26	2.60	152594.51	173164.98	26.18	138038.27	150099.70	29.38
	kroB100	40282.16	47467.99	5.72	39917.69	46866.45	37.48	32472.20	44606.40	36.47
Large	ch130	10056.42	11646.47	18.11	9134.04	11042.01	90.94	7522.39	9273.20	93.88
	a280	4766.32	6151.76	171.06	4725.97	6019.96	338.34	4112.49	5948.61	301.57
	u574	103085.74	112848.79	1365.88	103085.74	112848.79	1336.46	94888.29	100004.43	1317,50
Average		40726.27	46529.58	195.56	40545.73	45747.49	231.33	36432.95	40605.99	225.01

Table 12:Obtained results for the SQ scenario and α = 0.6.

Instance		Simple Local Search			Tabu Search			NSRS		
		Best	AVG	Time	Best	AVG	Time	Best	AVG	Time
Small	burma14	4193.19	4339.30	0.026	4193.19	4232.11	1.96	4193.19	4193.19	1.83
	gr24	1676.6	1783.1	0.47	1678.8	1730.41	4.02	1676.6	1676.6	4.10
	berlin52	10139.75	10798.92	1.06	10079.23	10580.86	15.20	9267.26	10123.98	17.22
	pr76	171213.55	186514.57	2.95	158178.48	182695.14	28.80	141103.89	156130.45	30.93
	kroB100	42665.31	49545.13	5.69	42474.31	48431.94	42.13	39813.15	47201.63	39.96
Large	ch130	10040.12	11722.95	17.65	9736.86	11347.85	88.26	8664.12	10632.60	77.65
	a280	5217.23	6419.08	181.96	5098.16	6316.01	327.20	4722.19	5651.11	326.25
	u574	107438.78	114459.41	1321.62	107438.78	114459.41	13294.4	88147.05	99217.15	1322.83
Average		44073.06	48197.80	191.42	42359.72	47474.21	229.62	37198.43	41853.33	227.59

Table 13:Obtained results for the SQ scenario and α = 0.8.

Tables from 02 to 13 present the results obtained from executing our algorithms on the selected set of benchmark instances. When applying the Simple Local Search algorithm to instances such as burma14, gr14, berlin52, and pr76, we observe promising results. In contract, in the instances kroB100, ch130, a280, and u574, the SLS produces lower-quality solutions compared to TS and NSRS. The Tabu Search algorithm obtained results that were generally comparable to, or slightly better than, those obtained with SLS. For Neighborhood Search with Random Selection, we observe that it consistently produces better results than both SLS and TS on most instances, with the exception of burma14, where all three algorithms obtain the same solution.

Regarding the running time, the SLS proved to be the fastest in scenarios ST and SL, and SQ for all tested instances whereas TS and NSRS required more time to converge. However, for the u574 instance, both TS and NSRS were faster than Local Search in specific cases-namely scenario ST with α = 0.6, scenario SL with α = 0.2, 0.4, 0.8, and scenario SQ with α = 0.4, 0.6.

To sum up, in general, from the obtained results we can see that the SLS was the faster method and it returns good quality solutions. In addition, we can see that the TS and the NSRS obtain better results than the SLS however they need more running times. Comparing NSRS to TS, we can see that NSRS obtains better results than TS without a considerable difference in the running times. From the obtained results, we can assume that the results of NSRS are better than those obtained by TS and SLS, and therefore we will use its results in our comparison with the literature.

4.4.3 Comparison with literature

In this subsection, we compare the results obtained by the NSRS algorithm against several state-of-the-art heuristics presented in [19], namely: BRKGA, M-VND, M-CNS, and Local Solver. It worth mentioning that we compare our results to only methods from [19], because unfortunately the detailed results of the methods proposed in [20] were not available. In tables from 14 to 25, the columns are defined as follows:

- ***Best-known***: reports the value of the best known solution found in the literature for each instance.
- ***%***: indicates the percentage gap between the solution obtained by the algorithm and the Best-known value.
- ***Time***: represent the average computational time (in seconds) requires to reach the reported solution(s).
- ***AVG***: shows the average value of the solutions obtained over multiple runs.
- ***Best***: provides the best solution found by the NSRS for each instance.

In addition, the star symbol (*) indicates that the algorithm successfully found the Best-known solution. Also, For each method, the percentage gap is calculated using the following formula:

$$Gap\ (\%) = x = \frac{value\ of\ solution - value\ of\ Best\_known}{value\ of\ Best\_known} \times 100$$

Instance		Best-known	BRKGA		M-VND		M-CNS		LS		NSRS			
			%	Time	%	Time	%	Time	%	Time	Best	AVG	%	Time
Small	burma14	3680.20	*	0	*	0	*	0	*	11	3737.54	3772.61	1.55	1.22
	gr24	1469.60	*	1	*	13	*	9	*	11	1469.6	1475.28	*	2.38
	berlin52	8564.80	11.26	1014	*	114	*	334	*	11	8570.24	8915.55	4.09	9.77
	pr76	121663.80	23.04	2438	2.09	51	4.18	148	*	1735	123226.15	128342.53	1.28	21.14
	kroB100	27320.00	22.60	3016	5.09	103	6.67	367	*	3932	28620.32	31278	4.75	26.69
Large	ch130	7592.80	152.58	2	5.95	919	*	1758	*	1758	7796.88	8377.33	2.68	47.2
	a280	3456.60	212.09	4	14.85	306	*	3662	*	3662	3665.51	3918.07	6.04	199.99
	u574	51889.80	323.94	11	17.73	42	*	3797	*	3797	56908.06	61149.25	9.67	898.17
Average		28204.7	93.18	810.75	5.71	193.5	1.35	1259.37	0	1864.6	29249.28	30905.57	3.75	150.82

Table 14:Comparison Obtained results for the ST scenario and α = 0.2.

Instance		Best-known	BRKGA		M-VND		M-CNS		LS		NSRS			
			%	Time	%	Time	%	Time	%	Time	Best	AVG	%	Time
Small	burma14	3891.40	*	0	*	0	*	0	*	11	3943.82	3943.82	1.34	1.22
	gr24	1542.20	*	24	*	75	*	14	*	11	1542.2	1556.82	*	2.49
	berlin52	9154.60	4.99	513	*	88	0.08	197	*	236	9174.38	9534.812	0.21	10.12
	pr76	130948.00	18.38	3139	0.11	476	2.27	298	*	1600	131225.87	137088.06	0.21	20.54
	kroB100	29295.80	16.16	2726	5.04	239	5.04	199	*	518	30944.71	34045.28	5.62	26.03
Large	ch130	8138.60	143.02	2	8.66	513	*	2085	*	2085	8463.7	9493.31	3.99	43.59
	a280	3770.20	192.45	5	13.67	2329	*	2332	*	2332	3986.63	4312.86	5.74	212.51
	u574	54591.40	309.45	10	15.61	84	*	3842	*	3842	62237.14	66330.58	14.0	821.48
Average		30166.52	85.55	802.3	5.38	475.5	0.92	1120.87	0	1329.37	31439.80	33288.19	3.88	141.12

Table 15:Comparison Obtained results for the ST scenario and α = 0.4.

Instance		Best-known	BRKGA		M-VND		M-CNS		LS		NSRS			
			%	Time	%	Time	%	Time	%	Time	Best	AVG	%	Time
Small	burma14	4102.60	*	0	*	0	*	0	*	11	4150.10	4150.10	1.15	1.42
	gr24	1614.80	*	67	*	106	*	13	*	22	1614.8	1627.17	*	3.88
	berlin52	9744.40	6.69	1537	*	3	0.17	91	*	416	9889.63	10152.87	1.49	16.048
	pr76	139383.00	14.93	3243	1.19	274	3.01	164	*	2174	141620.29	148466.57	1.60	29.30
	kroB100	31161.20	18.15	2420	3.75	498	2.95	735	*	2061	32410.04	35399.54	4.00	33.62
Large	ch130	8637.20	135.94	2	6.28	20	*	3865	*	3865	9626.22	10313.34	11.45	62.49
	a280	4027.00	179.71	4	10.22	2638	*	3752	*	3752	4312.54	4653.69	7.09	264.97
	u574	60074.40	278.04	11	9.60	1488	*	3358	*	3358	64743.83	70520.35	7.77	1481.84
Average		32343.07	79.18	910.5	3.88	628.37	2.04	1497.2	0	1957.37	33545.93	35660.45	4.31	236.69

Table 16:Comparison Obtained results for the ST scenario and α = 0.6.

Instance		Best -known	BRKGA		M-VND		M-CNS		LS		NSRS			
			%	Time	%	Time	%	Time	%	Time	Best	AVG	%	Time
Small	burma14	4270.80	*	0	*	0	*	0	*	11	4290.65	4290.65	0.46	1.85
	gr24	1676.60	*	2	*	53	*	23	*	11	1676.6	1676.6	*	4.76
	berlin52	10321.40	4.79	859	0.12	4	0.02	182	*	2704	10328.04	10711.30	0.06	15.00
	pr76	148434.8	10.71	3287	1.57	50	0.79	105	*	2907	148849.26	156477.86	0.27	25.84
	kroB100	33131.00	10.31	2672	4.42	209	3.57	218	*	1025	35510.81	38935.05	7.18	34.05
Large	ch130	9162.40	69.41	3594	5.50	339	*	3730	*	3730	9650.92	10857.51	5.33	55.47
	a280	/	/	/	/	/	/	/	/	/	4554.66	4803.08	*	261.69
	u574	64653.00	256.64	11	7.99	1084	*	2051	*	2051	68688.43	75998.76	6.24	1448.79
Average		33956.25	43.98	1489.2	2.45	248.42	0.54	901.28	0	1777	35443.67	37968.85	2.44	230.93

Table 17:Comparison Obtained results for the ST scenario and α = 0.8.

Instance		Best -known	BRKGA		M-VND		M-CNS		LS		NSRS			
			%	Time	%	Time	%	Time	%	Time	Best	AVG	%	Time
Small	burma14	2832.00	*	0	*	0	*	0	*	11	2917.46	2917.46	3.01	1.35
	gr24	1163.60	*	6	*	40	*	18	*	169	1192.00	1209.4	2.44	3.44
	berlin52	6700.80	6.95	572	1.86	93	6.36	74	*	3346	6823.71	7615.89	1.83	16.02
	pr76	100117.60	18.08	994	1.34	823	12.76	42	*	597	105505.15	115073.53	5.38	32.38
	kroB100	21969.80	23.33	0	4.44	1970	4.30	305	*	405	22864.09	25494.91	4.18	57.33
Large	ch130	6101.20	6.62	1	6.53	772	*	3256	*	3256	6479.99	7066.76	6.20	104.40
	a280	2923.80	14.48	4	7.00	2941	*	1927	*	1927	3505.92	3889.82	19.90	399.60
	u574	43541.60	40.02	10	2.64	3243	*	3380	*	3380	50305.99	59253.18	15.53	1846.08
Average		23168.80	13.68	198.3	2.97	1235.25	2.92	1125.25	0	1636.37	24949.28	27815.11	7.30	307.57

Table 18:Comparison Obtained results for the SL scenario and α = 0.2.

Instance		Best-known	BRKGA		M-VND		M-CNS		LS		NSRS			
			%	Time	%	Time	%	Time	%	Time	Best	AVG	%	Time
Small	burma14	3136.40	*	0	*	0	*	0	*	11	3199.23	3199.23	2.00	1.32
	gr24	1256.00	4.95	34	*	39	*	30	*	135	1287	1308.56	2.46	3.49
	berlin52	7340.40	6.92	1115	2.24	843	4.64	275	*	3470	7596.96	8124.41	10.68	15.80
	pr76	113871.4	11.50	1448	4.02	558	6.79	503	*	3447	118403.4	129977.49	3.97	31.95
	kroB100	24093.40	20.36	0	3.34	1448	3.38	357	*	2873	25117.7	27947.15	4.25	54.98
Large	ch130	6738.40	5.26	1	3.74	2184	*	2163	*	2163	6966.53	7753.95	3.38	97.31
	a280	3284.60	9.07	3	1.78	3529	*	3628	*	3628	3586.95	3390.95	9.20	427.27
	u574	47989.60	34.33	9	*	376	3.65	2231	3.65	2231	55569.24	62956.00	15.76	1837.54
Average		25963.77	11.54	326.25	1.89	1122.12	2.30	1148.37	0.45	2244.7	27715.87	30582.21	6.46	308.70

Table 19:Comparison Obtained results for the SL scenario and α = 0.4.

Instance		Best-known	BRKGA		M-VND		M-CNS		LS		NSRS			
			%	Time	%	Time	%	Time	%	Time	Best	AVG	%	Time
Small	burma14	3370.20	*	0	*	0	*	0	*	11	3422.18	3422.73	1.54	1.89
	gr24	1334.20	2.59	63	*	39	*	19	*	56	1375.0	1394.0	3.05	7.15
	berlin52	7979.00	6.10	894	2.08	155	2.83	183	*	1476	8153.27	8727.42	2.18	23.96
	pr76	124700.0	8.40	1180	2.44	158	1.87	213	*	11	128004.4	135461.46	2.64	44.76
	kroB100	25839.00	19.56	436	4.67	2220	4.02	1080	*	811	27124.35	31273.28	4.97	81.89
Large	ch130	7209.60	6.54	1	5.01	930	*	2569	*	2569	8255.19	9152.87	14.50	123.15
	a280	3513.60	8.77	3	0.82	2206	*	3876	*	3876	3907.38	4442.41	11.20	567.91
	u574	51394.80	32.22	10	*	2528	5.90	1408	5.90	1408	63666.32	68857.05	23.87	1902.8
Average		28167.55	10.52	323.37	1.87	1029.5	1.82	1168.51	0.73	1277.25	30488.51	32841.40	7.99	344.18

Table 20:Comparison Obtained results for the SL scenario and α = 0.6.

Instance		Best-known	BRKGA		M-VND		M-CNS		LS		NSRS			
			%	Time	%	Time	%	Time	%	Time	Best	AVG	%	Time
Small	burma14	3468.00	*	0	*	0	*	0	*	11	3522.45	3522.45	1.57	1.84
	gr24	1398.00	2.60	1	*	47	*	19	*	56	1452.4	1462.84	3.89	7.46
	berlin52	8526.20	0.55	2100	0.58	169	2.50	171	*	2147	8828.43	9642.35	3.54	21.09
	pr76	129003.0	17.32	0	4.69	1058	4.22	264	*	3064	135312.5	146436.27	4.89	41.16
	kroB100	27375.6	19.83	0	6.02	910	3.99	712	*	2963	29894.98	34033.76	9.20	82.88
Large	ch130	7806.60	5.92	1	1.80	2247	*	4000	*	4000	8462.70	9923.82	8.40	119.07
	a280	3758.80	7.84	4	0.87	3612	7.00	1566	7.00	1566	4365.87	4655.54	16.15	556.34
	u574	62743.6	19.74	12	*	3116	9.75	2175	9.75	2175	69042.3	75923.32	34.33	1855.22
Average		30509.97	9.22	264.75	1.74	1394.87	3.43	1113.37	2.09	1997.75	32610.20	35700.04	9.3	335.63

Table 21:Comparison Obtained results for the SL scenario and α = 0.8.

Instance		Best-known	BRKGA		M-VND		M-CNS		LS		NSRS			
			%	Time	%	Time	%	Time	%	Time	Best	AVG	%	Time
Small	burma14	3480.00	*	0	*	0	*	0	*	11	3527.47	3527.47	1.36	1.09
	gr24	1469.60	*	1	*	13	*	10	*	11	1469.6	1472.1	*	2.32
	berlin52	7956.20	7.90	583	0.17	235	1.08	50	*	1960	8265.47	8419.37	3.88	12.77
	pr76	122930.2	11.24	1395	1.56	170	3.69	351	*	1938	127465.15	132961.69	3.68	21.42
	kroB100	26494.20	14.07	3600	5.71	1038	4.13	1238	*	1487	30790.97	35638.47	16.21	25.87
Large	ch130	6982.20	59.55	1	3.92	114	*	3876	*	3876	7243.15	9120.03	3.74	59.53
	a280	3143.20	75.69	3	17.27	2785	*	1825	*	1825	4342.99	5183.33	38.17	235.53
	u574	48388.60	114.65	7	32.42	550	*	2479	*	2479	83330.89	90340.10	72.21	824.24
Average		27605.52	35.38	698.75	7.63	613.12	1.11	1228.62	0	1698.37	33304.46	35832.82	19.89	147.84

Table 22:Comparison Obtained results for the SQ scenario and α = 0.2.

Instance		Best-known	BRKGA		M-VND		M-CNS		LS		NSRS			
			%	Time	%	Time	%	Time	%	Time	Best	AVG	%	Time
Small	burma14	3690.00	*	0	*	0	*	0	*	11	3527.47	3727.18	-4.40	1.11
	gr24	1542.20	*	3	*	13	*	9	*	11	1542.2	1557.04	*	2.43
	berlin52	8419.80	9.92	571	0.25	9	0.64	42	*	45	8621.69	9153.18	2.39	12.57
	pr76	128720.0	9.52	2322	*	129	3.05	35	0.09	2850	128385.64	137694.95	-0.25	21.05
	kroB100	27893.40	7.39	2048	4.79	159	6.25	1977	*	1352	29568.58	35312.46	6	25.71
Large	ch130	7490.60	53.60	1	2.06	1596	*	766	*	766	7747.34	9843.57	3.42	59.58
	a280	3512.00	60.83	3	6.96	1344	*	2637	*	2637	4463.39	5369.33	27.08	231.68
	u574	54956.00	91.28	8	16.43	1080	*	1217	*	1217	84512.8	88741.23	53.78	843.76
Average		29528	29.06	619.5	3.81	541.25	1.24	835.37	0.01	1111.12	33546.13	36424.86	11.0	149.73

Table 23:Comparison Obtained results for the SQ scenario and α = 0.4.

Instance		Best-known	BRKGA		M-VND		M-CNS		LS		NSRS			
			%	Time	%	Time	%	Time	%	Time	Best	AVG	%	Time
Small	burma14	3900.00	*	0	*	0	*	0	*	11	3971.29	3971.29	1.8	1.52
	gr24	1614.80	*	6	*	13	*	10	*	22	1614.8	1626.43	*	3.01
	berlin52	8843.20	7.92	591	0.22	200	0.89	214	*	552	8843.88	9317.89	0.007	16.81
	pr76	133609.40	7.17	1282	0.37	266	2.28	38	*	822	138038.27	150099.70	3.31	29.38
	kroB100	29291.80	8.82	3390	4.23	261	3.44	628	*	3797	32472.20	44606.40	10.8	36.47
Large	ch130	7709.20	53.98	1	1.34	1294	*	124	*	124	7522.39	9273.20	-2.42	93.88
	a280	3706.40	55.80	3	3.76	2912	*	3076	*	3076	4112.49	5948.61	10.95	301.57
	u574	56628.80	88.12	7	15.29	668	*	4045	*	4045	94888.29	100004.43	67.56	1317,5
Average		30662.95	27.72	660	3.15	701.75	0.82	1016.87	0	1556.12	36432.95	40605.99	11.50	225.01

Table 24:Comparison Obtained results for the SQ scenario and α = 0.6.

Instance		Best-known	BRKGA		M-VND		M-CNS		LS		NSRS			
			%	Time	%	Time	%	Time	%	Time	Best	AVG	%	Time
Small	burma14	4110.00	*	0	*	0	*	0	*	11	4193.19	4193.19	2.02	1.83
	gr24	1676.60	*	2	*	16	*	10	*	11	1676.6	1676.6	*	4.10
	berlin52	9266.60	8.92	640	0.21	24	0.21	48	*	112	9267.26	10123.98	0.007	17.22
	pr76	139551.80	8.35	1458	0.94	133	2.90	128	*	1949	141103.89	156130.45	1.1	30.93
	kroB100	30956.80	4.49	3480	4.25	1568	3.93	1639	*	416	39813.15	47201.63	28.60	39.96
Large	ch130	7974.00	48.75	820	2.15	1223	*	3482	*	3482	8664.12	10632.60	8.65	77.65
	a280	3713.40	58.91	3	8.00	822	*	4000	*	4000	4722.19	5651.11	27.16	326.25
	u574	62333.20	73.29	8	5.51	1629	*	1927	*	1927	88147.05	99217.15	41.41	1322.83
Average		32447.8	25.33	801.37	2.63	676.87	0.88	1404.25	0	1488.5	37198.43	41853.33	13.61	227.59

Table 25:Comparison Obtained results for the SQ scenario and α = 0.8.

In Tables from 14 to 25, we can observe that the solutions obtained by NSRS are very competitive compared to those reported in the literature. If we examine the results by scenario and by α, and by calculating the difference between the average GAP values achieved by NSRS and those achieved by BRKGA, M-VND, M-CNS and LS respectively, we can see that :

- In the ST scenario with α = 0.2, NSRS achieved an average GAP of 3.75%, which is close to both M-CNS (1.35 %) and LS (0%). It performed better than M-VND (by 5.71%) and outperformed BRKGA (93.18 %).
- In the ST scenario with α = 0.4, NSRS achieved an average GAP of 3.88%, which is close to both M-CNS (0.92%) and LS (0%). It performed better than M-VND (5.38%) and outperformed BRKGA (85.55%).
- In the ST scenario with α = 0.6, NSRS achieved an average GAP of 4.31%, which is close to both M-CNS (2.04%), LS (0%), and M-VND (3.88%). Additionally, it outperformed BRKGA (79.18%).
- In the ST scenario with α = 0.8, NSRS achieved an average GAP of 2.44%, which is close to both M-CNS (0.54%) and LS (0%). It performed better than M-VND (2.45%) and outperformed BRKGA (43.98%).
- In the SL scenario with α = 0.2, NSRS achieved an average GAP of 7.30%, which is close to both M-CNS (2.92%), LS (0%), and M-VND (2.97%). Additionally, it outperformed BRKGA (13.68%).
- In the SL scenario with α = 0.4, NSRS achieved an average GAP of 6.46%, which is close to both M-CNS (2.30%), LS (0.45%), and M-VND (1.89%). Additionally, it outperformed BRKGA (11.54%).
- In the SL scenario with α = 0.6, NSRS achieved an average GAP of 7.99%, which is close to both M-CNS (1.82%), LS (0.73%), and M-VND (1.87%). Additionally, it outperformed BRKGA (10.52%).
- In the SL scenario with α = 0.8, NSRS achieved an average GAP of 9.3%, which is close to both M-CNS (3.43%), LS (2.09%), and M-VND (1.74%). Additionally, it outperformed BRKGA (9.22%).
- In the SQ scenario with α = 0.2, NSRS achieved an average GAP of 19.89%, which is significantly different from M-CNS (1.11%) and LS (0%), but close to M-VND (7.63%). Additionally, it outperformed BRKGA (35.38%).
- In the SQ scenario with α = 0.4, NSRS achieved an average GAP of 11.0%, which is close to both M-CNS (1.24%), LS (0.01%), and M-VND (3.81%). Additionally, it outperformed BRKGA (29.06%).
- In the SQ scenario with α = 0.6, NSRS achieved an average GAP of 11.50%, which is close to both M-CNS (0.82%), LS (0%), and M-VND (3.15%). Additionally, it outperformed BRKGA (27.72%).

- In the SQ scenario with α = 0.8, NSRS achieved an average GAP of 13.61%, which is close to both M-CNS (0.88%), LS (0%), and M-VND (2.63%). Additionally, it outperformed BRKGA (25.33%).

We present comparative diagram illustrating the performance of our best-performing algorithm, NSRS, in comparison with results reported in the literature. The evaluation is based on the average gap between solutions.

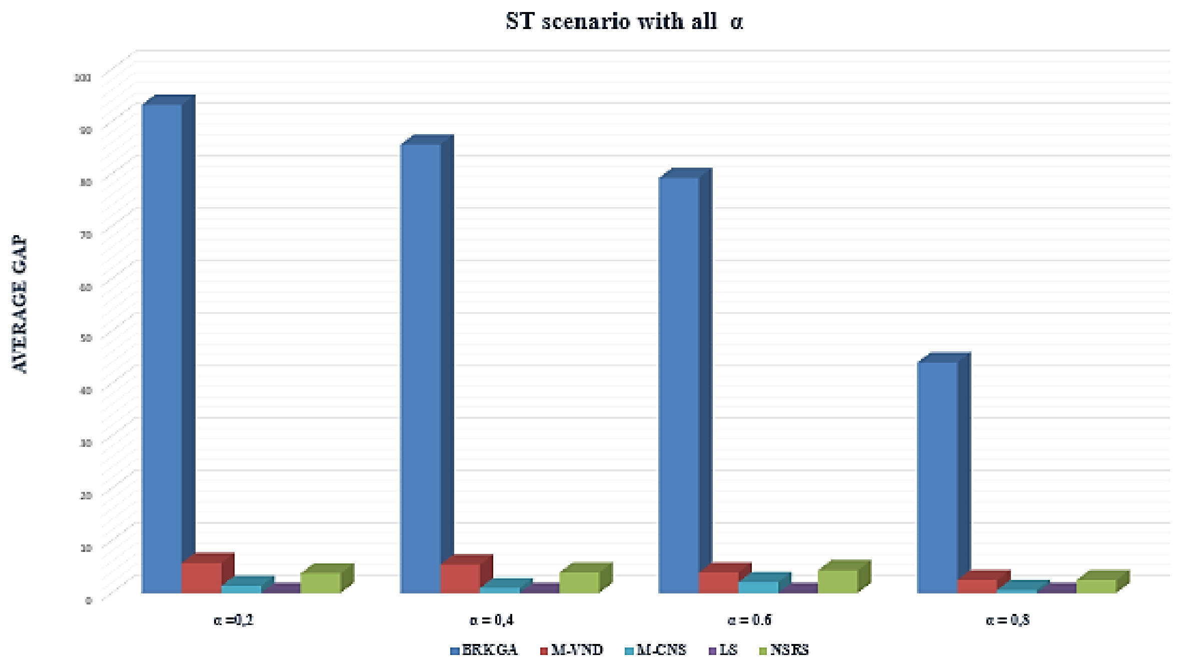


Figure 29: Comparison with the literature for the ST scenario across all α values.

These plots show that the results of the NSRS algorithm are more competitive than those of M-VND, M-CNS, and LS, and it widely outperforms the results of BRKGA.

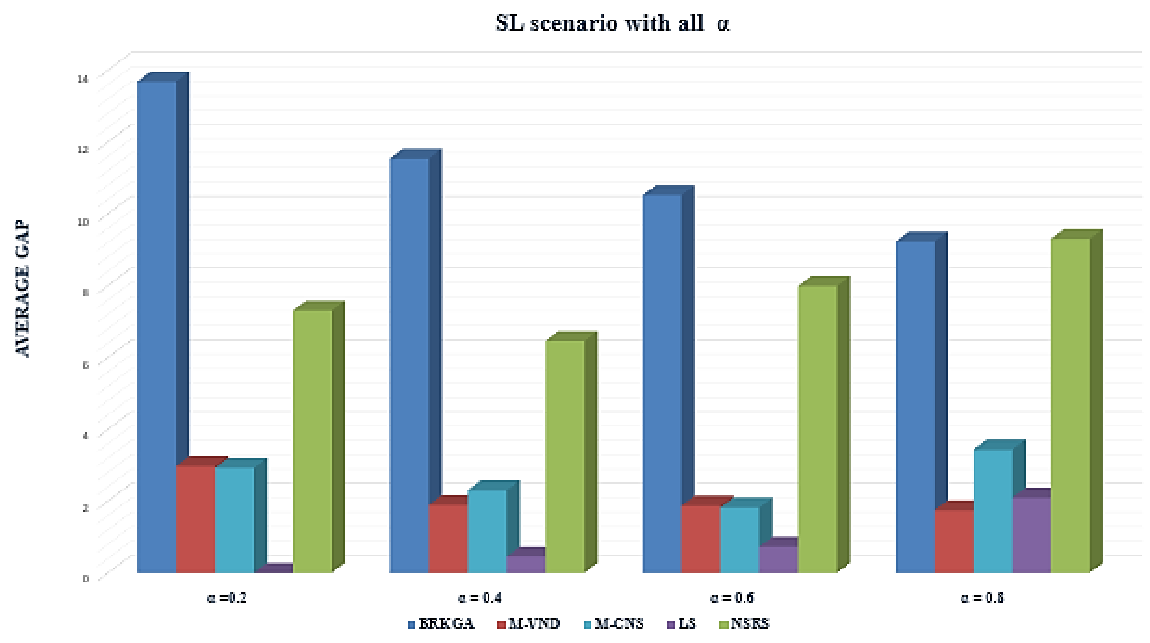


Figure 30:Comparison with the literature for the SL scenario across all α values

.

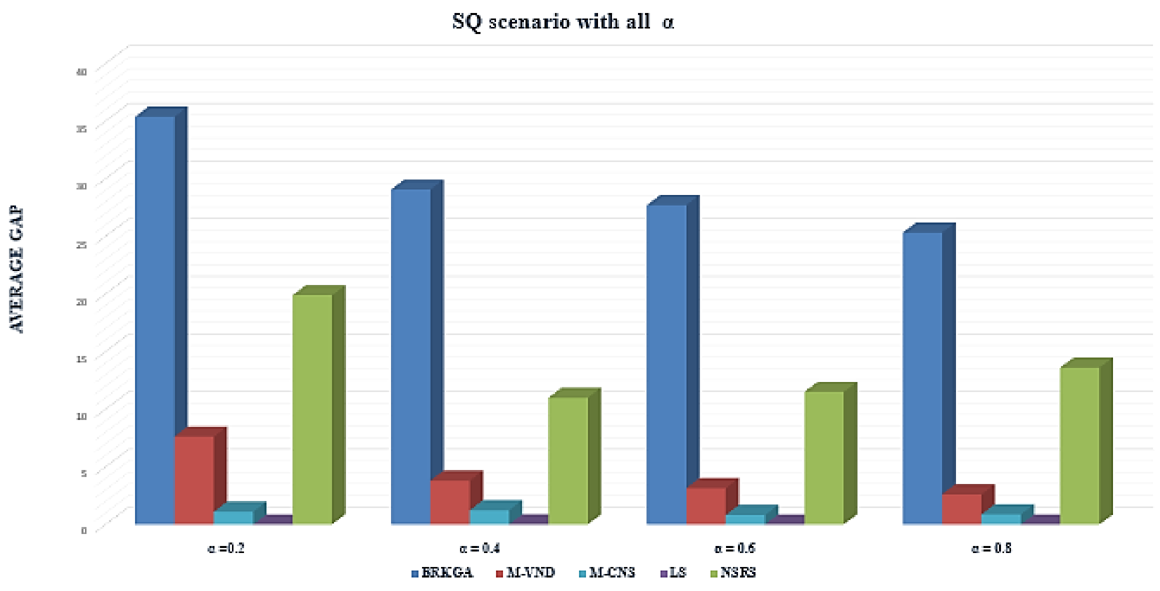


Figure 31:Comparison with the literature for the SQ scenario across all α values.

In scenarios SL and SQ, the quality of the NSRS results slightly decreases; however, it still outperforms BRKGA.

Regarding execution time, we can observed from Tables (14-25) that the average execution time of the NSRS algorithm is lower than average running times of BRKGA, M-VND, M-CNS and LS in all scenarios and **α.**

Instance	**Scenario**	**α**	**Best-know**	**BRKGA**	**M-VND**	**M-CNS**	**LS**	**NSRS**	
				%	%	%	%	Best	%
burma14	SQ	0.4	3690.00	*	*	*	*	3527.47	-4.40
gr24	ST	0.2	1469.6	*	*	*	*	1469.6	*
	ST	0.4	1542.20	*	*	*	*	1542.2	*
	ST	0.6	1614.80	*	*	*	*	1614.80	*
	ST	0.8	1676.60	*	*	*	*	1676.60	*
	SQ	0.2	1496.60	*	*	*	*	1496.60	*
	SQ	0.4	1542.20	*	*	*	*	1542.20	*
	SQ	0.6	1614.80	*	*	*	*	1614.80	*
	SQ	0.8	1676.60	*	*	*	*	1676.60	*
pr76	SQ	0.4	128720.0	9.52	*	3.05	0.09	128385.64	-0.25
ch130	SQ	0.6	7709.20	53.98	1.34	*	*	7522.39	-2.42
a280	ST	0.8	/	/	/	/	/	4554.66	*

Table 26:Comparison of the best obtained results with those from the literature.

In table 26, we present the instances and the values of obtained solutions where our NSRS finds the best-known solutions of the literature or reports new best-known solutions. So, in the instances burma14 and pr76 in the SQ scenario with α = 0.4, ch130 in the SQ scenario with α = 0.6, and a280 in the ST scenario with α = 0.8 yielded solutions that outperformed those obtained by BRKGA, M-VND, M-CNS, and LS .

For example, in the case of burma14 under the SQ scenario with α = 0.4, our approach consistently outperformed the algorithms BRKGA, M-VND, M-CNS, and LS.

Furthermore, in some cases, our method matched the best-known results for the gr24 instance, where the solutions obtained in both ST and SQ scenarios for all values of α, were identical to those produced by BRKGA, M-VND, M-CNS, and LS.

4.5 Conclusion

In this chapter, we presented the results of the experiments of our proposed methods using a benchmark dataset from the literature. First, we explained the process by which the instances of problem were generated. Then, we presented few representative examples of these instances. Next, we provided the parameters and technical details of the implementation we used. Also, we described the structure of a solution as defined in our approach. In our experiments, first, we compared the results of the proposed methods, and the NSRS reported better results than the SLS and the TS. After that, we compared the obtained results of NSRS with those reported by the state-of-the-art methods. In the comparison with the literature, we found that the NSRS was very competitive comparing to the four methods of the literature. In addition, it was able to reach 8 best known solutions and to find new best-known solutions for 4 instances. Therefore, we assume that the NSRS found very promising results.

General conclusion

In this thesis, we proposed algorithms to solve the Hub Location Routing Problem (HLRP). The main objective was to determine an optimal set of hubs to be installed, organize local routes between these hubs and non-hubs, and design an efficient inter-hub route. This should be done while minimizing the total cost, including local transportation, and inter-hub transport costs.

In the first chapter, we presented the hub location problem and its main variants, including the Single-Allocation Hub Location Problem, the Multiple Allocation p-Hub Median Location problem, and p-Hub Center Location Problem. Also, we highlighted their real-world applications. In addition, we introduced routing problems such Traveling Salesman Problem (TSP) and the Vehicle Routing Problem (VRP), along with their main variants. we then provided description of the Hub Location Routing Problem and some of its key variants.

In the second chapter, we provided a general overview of combinatorial optimization problems. We introduced exact methods, including the backtracking algorithm and the branch and bound method. We also described commonly used heuristic and metaheuristic approaches, and discussed several hybridization techniques.

In the third chapter, we defined the specific HLRP variant addressed in this work and reviewed the most related studies in the literature that dealt with it. Then, we described the algorithms we developed to solve the problem, namely: a Local Search algorithm, a Tabu Search algorithm, and Neighborhood Search with Random Selection.

In the fourth chapter, we assessed the performance of our algorithms using benchmark instances from the literature. The results obtained were promising and demonstrated competitive performance compared to existing approaches in the literature.

Finally, we are looking forward to:

- Study the application of population-based metaheuristics on the HLRP, such as Cuckoo search method.
- Study the hybridization of the proposed methods in this thesis with other optimization methods.
- Extend the methods proposed to solve more hard and realistic variants of the HLRP.

Bibliography

[1] Farahani, R. Z., Hekmatfar, M., Arabani, A. B., & Nikbakhsh, E. (2013). Hub location problems: A review of models, classification, solution techniques, and applications. Computers & industrial engineering, 64(4), 1096-1109.

[2] Klincewicz, J. G. (1991). Heuristics for the p-hub location problem. *European Journal of Operational Research*, *53*(1), 25-37.

[3] Wandelt, S., Wang, S., & Sun, X. (2024). A literature review on hub location-routing models and their solution techniques. *Computers & Operations Research*, 106861.

[4] Campbell, J. F., & O'Kelly, M. E. (2012). Twenty-five years of hub location research. *Transportation Science*, *46*(2), 153-169.

[5] Contreras, I., & O’Kelly, M. (2019). Hub location problems. *Location science*, 327-363.

[6] Campbell, J. F. (1996). Hub location and the p-hub median problem. *Operations research*, *44*(6), 923-935.

[7] Yang, K., Liu, Y. K., & Yang, G. Q. (2013). Solving fuzzy p-hub center problem by genetic algorithm incorporating local search. *Applied soft computing*, *13*(5), 2624-2632.

[8] Karimi, H., & Bashiri, M. (2011). Hub covering location problems with different coverage types. *Scientia iranica*, *18*(6), 1571-1578.

[9] Jaillet, P., Song, G., & Yu, G. (1996). Airline network design and hub location problems. *Location science*, *4*(3), 195-212.

[10] Guan, X., Ge, C., Wang, X., & Wang, G. (2013). Models and algorithms for hub and spoke locations for emergency service facilities in response to serious emergency incidents. *Human and Ecological Risk Assessment: An International Journal*, *19*(2), 553-565.

[11] Kim, H., & O'Kelly, M. E. (2009). Reliable p-hub location problems in telecommunication networks. *Geographical Analysis*, *41*(3), 283-306.

[12] Carello, G., Della Croce, F., Ghirardi, M., & Tadei, R. (2004). Solving the hub location problem in telecommunication network design: A local search approach. *Networks: An International Journal*, *44*(2), 94-105.

[13] Gelareh, S., & Nickel, S. (2011). Hub location problems in transportation networks. *Transportation Research Part E: Logistics and Transportation Review*, *47*(6), 1092-1111.

[14] Reinelt, G. (1995). Tsplib95. *Interdisziplinäres Zentrum für Wissenschaftliches Rechnen (IWR), Heidelberg*, *338*, 1-16.

[15] Braekers, K., Ramaekers, K., & Van Nieuwenhuyse, I. (2016). The vehicle routing problem: State of the art classification and review. *Computers & industrial engineering*, *99*, 300-313.

[16] Laporte, G., & Osman, I. H. (1995). Routing problems: A bibliography. *Annals of operations research*, *61*, 227-262.

[17] Toth, P., & Vigo, D. (1998). Routing Problems.

[18] Wahyuningsih, S., Satyananda, D., & Hasanah, D. (2016). Implementations of TSP-VRP variants for distribution problem. *Global Journal of Pure and Applied Mathematics*, *12*(1), 723-732.

[19] Lopes, M.C., de Andrade, C.E., de Queiroz, T.A., Resende, M.G., Miyazawa, F.K.: Heuristics for a hub location-routing problem. Networks 68(1), 54–90 (2016).

[20] Ratli, M., Urošević, D., El Cadi, A. A., Brimberg, J.,Mladenović, N., & Todosijević, R. (2022). An efficient heuristic for a hub location routing problem. Optimization Letters, 1-20.

[21] Wu, Y., Fang, H., Qureshi, A. G., & Yamada, T. (2024). Stochastic Single-Allocation Hub Location Routing Problem for the Design of Intra-City Express Systems. In *ICORES* (pp. 82-91).

[22] Khaleghi, A., & Eydi, A. (2022). Multi-period hub location problem: a review. *RAIRO-operations Research*, *56*(4), 2751-2765.

[23] Naeem, M. (2009). Using genetic algorithms for the single allocation hub location problem (Master's thesis), Faculty of Computer Science, Brock University St. Catherines, Ontario.

[24] Campbell, A. M., Lowe, T. J., & Zhang, L. (2005). The p-hub center allocation problem. *European journal of operational research*, *176*(2), 819-835.

[25] Campbell, J. F. (1994). Integer programming formulations of discrete hub location problems. *European journal of operational research*, *72*(2), 387-405.

[26] Ono, D., Rincón, J., & Thomas, A. (2020). A Criteria-Based Approach to the Traveling Salesman Problem (TSP). *Western Decision Science Journal*.

[27] Jünger, M., Reinelt, G., & Rinaldi, G. (1995). The traveling salesman problem. *Handbooks in operations research and management science*, *7*, 225-330.

[28] Golden, B. L., Raghavan, S., & Wasil, E. A. (Eds.). (2008). *The vehicle routing problem: latest advances and new challenges* (Vol. 43). Springer Science & Business Media.

[29] Injac, Z., & Drašković, D. (2024). Classification of vehicle routing problem. *JOURNAL TTTP-TRAFFIC AND TRANSPORT THEORY AND PRACTICE*, *12*(1), 36-41.

[30] Wen, M. (2010). Rich Vehicle Routing Problems and Applications (Doctoral dissertation). DTU Management Engineering.

[31] Ombuki, B., Ross, B. J., & Hanshar, F. (2006). Multi-objective genetic algorithms for vehicle routing problem with time windows. *Applied Intelligence*, *24*, 17-30.

[32] Yuehui, W. (2022). *Hub location routing problem for the design of intra-city express systems* (Doctoral dissertation, 京都大学).

[33] Wu, Y., Fang, H., Qureshi, A. G., & Yamada, T. (2024, January). Stochastic Single-Allocation Hub Location Routing Problem for the Design of Intra-City Express Systems. In *ICORES* (pp. 82-91).

[34] Kumar, R., Wei, X., & Singh, A. (2014). *Different approaches to solve Traveling Salesman Problem* (Doctoral dissertation, Georgia Institute of Technology, USA).

[35] Li, L. Y., & Fu, Z. (2002). The school bus routing problem: a case study. *Journal of the Operational Research Society*, *53*(5), 552-558.

[36] Ellegood, W. A., Solomon, S., North, J., & Campbell, J. F. (2019). School bus routing problem: Contemporary trends and research directions. *Omega*, *95*, 102056.

[37] Nuortio, T., Kytöjoki, J., Niska, H., & Bräysy, O. (2006). Improved route planning and scheduling of waste collection and transport. *Expert systems with applications*, *30*(2), 223-232.

[38] Guemri, O. (2017). *Proposition de solutions pour l'optimisation des chaînes logistiques* (Doctoral dissertation). Laboratoire d'Informatique d'Oran (LIO).

[39] Kondrak, G., & Van Beek, P. (1997). A theoretical evaluation of selected backtracking algorithms. *Artificial Intelligence*, *89*(1-2), 365-387.

[40] Bacchus, F. (2000). A uniform view of backtracking. *Unpublished manuscript*.

[41] Russell, S. J., & Norvig, P. (2021). *Artificial intelligence: a modern approach (4th ed .).* Pearson Education.

[42] Bellemalem, CH., & Boudouda, Z. T. (2024). *Une méthode de recherche tabou probabiliste pour le problème de localisation à deux niveaux avec contrainte de capacité* (Master's thesis, university center of abdalhafid boussouf-MILA).

[43] Lawler, E. L., & Wood, D. E. (1966). Branch-and-bound methods: A survey. Operations research, 14(4), 699-719.

[44] Feigenbaum, E. A., & Feldman, J. (1963). Computers and thought. McGraw-Hill.

[45] Slagle, J. R. (1971). Artificial intelligence: The heuristic programming approach. McGraw-Hill.

[46] Pearl, J. (1984). *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc..

[47] Gao, W., Friedrich, T., Neumann, F., & Hercher, C. (2018, July). Randomized greedy algorithms for covering problems. In Proceedings of the Genetic and Evolutionary Computation Conference (pp. 309-315).

[48] M. G. C. Resende et C. C. Ribeiro, «Greedy Randomized Adaptive Search Procedures: Advances, Hybridizations, and Applications, » chez Handbook of Metaheuristics, 2010, pp. 283-319.

[49] Dorigo, M., & Stützle, T. (2019). Ant colony optimization: overview and recent advances. *Handbook of metaheuristics*, pp. 311-351.

[50] I.H. Osman and G. Laporte, Metaheuristics : a bibliography. Annals of Operations Research 63, 513-623, 1996.

[51] Yang, X. S., Deb, S., & Fong, S. (2014). Metaheuristic algorithms: optimal balance of intensification and diversification. Applied Mathematics & Information Sciences, 8(3), 977.

[52] Kirkpatrick, S., Gelatt Jr, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, *220*(4598), 671-680.

[53] Glover F., 1986. Future paths for integer programming and links to artificial intelligence. Computers and Operations Research, 13, p. 533-549.

[54] Glover, F., & Laguna, M. (1997). Tabu search kluwer academic. Boston, Texas, Dordrecht.

[55] Laguna, M., Martí, R., Martínez-Gavara, A., Pérez-Peló, S., & Resende, M. G. (2025). Greedy Randomized Adaptive Search Procedures with Path Relinking. An analytical review of designs and implementations. Titre*European Journal of Operational Research*.

[56] Resende, M.G.C., Ribeiro, C.C. (2003). Greedy Randomized Adaptive Search Procedures. In: Glover, F., Kochenberger, G.A. (eds) Handbook of Metaheuristics, vol.57, p. 219-249.Springer. https://doi.org/10.1007/0-306-48056-5_8

[57] Feo, T. A., & Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of global optimization*, *6*, 109-133.

[58] N Mladenovi´c et P. Hansen, «Variable neighborhood search,» Computers & Operations Research, vol. 24, p. 1097–1100, 1997.

[59] Hansen, P., & Mladenovic, N. (2007). Variable neighborhood search methods. Groupe d'études et de recherche en analyse des décisions.

[60] Hansen, P., Mladenović, N., Todosijević, R., & Hanafi, S. (2017). Variable neighborhood search: basics and variants. EURO Journal on Computational Optimization, 5(3), 423-454.

[61] Maniezzo, V., Stützle, T., & Voß, S. (2021). *Matheuristics*. New York, NY, USA: Springer International Publishing. https://doi.org/10.1007/978-3-030-70277-9

[62] Ramalhinho, H. (2019). Iterated local search: Applications and extensions. SCITEPRESS–Science and Technology Publications. pp. 7-15.

[63] Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press: Ann Arbor, 1975.

[64] Mirjalili, S. (2019). Genetic algorithm. *Evolutionary algorithms and neural networks: Theory and applications*. Springer, pp. 43-55. https://doi.org/10.1007/978-3-319-93025-1_4

[65] Katoch, S., Chauhan, S. S., & Kumar, V. (2021). A review on genetic algorithm: past, present, and future. *Multimedia tools and applications*, *80*, 8091-8126. https://doi.org/10.1007/s11042-020-10139-6

[66] Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks* (Vol. 4, pp. 1942-1948). IEEE.

[67] Gherboudj, A. (2013). Méthodes de résolution de problèmes difficiles académiques. *Université de Constantine2*.

[68] Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE transactions on systems, man, and cybernetics, part b (cybernetics)*, *26*(1), 29-41.

[69] Kumar, A., Kumar, D., & Jarial, S. K. (2017). A review on artificial bee colony algorithms and their applications to data clustering. *Cybernetics and Information Technologies*, *17*(3), 3-28.

[70] Karaboga, D., & Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *Journal of global optimization*, *39*, 459-471.

[71] Yang, X. S., & Deb, S. (2009, December). Cuckoo search via Lévy flights. In *2009 World congress on nature & biologically inspired computing (NaBIC)* (pp. 210-214). IEEE.

[72] Dubkov, A. A., Spagnolo, B., & Uchaikin, V. V. (2008). Lévy flight superdiffusion: an introduction. *International Journal of Bifurcation and Chaos*, *18*(09), 2649-2672.

[73] Brown, C. T., Liebovitch, L. S., & Glendon, R. (2007). Lévy flights in Dobe Ju/'hoansi foraging patterns. *Human Ecology*, *35*, 129-138.

[74] Talbi, E. G. (2016). Combining metaheuristics with mathematical programming, constraint programming and machine learning. *Annals of Operations Research*, *240*(1), 171-215.

[75] El Samrout, A. (2018). *Hybridization of multicriteria metaheuristic optimization methods for mechanical problems* (Doctoral dissertation, Université de Technologie de Troyes). https://theses.hal.science/tel-03608517v1

[76] Alba, E., Luque, G., & Nesmachnow, S. (2013). Parallel metaheuristics: recent advances and new trends. *International Transactions in Operational Research*, *20*(1), 1-48.

[77] Talbi, E. G. (2009). *Metaheuristics: from design to implementation*. John Wiley & Sons.

[78] Alba, E., Luque, G., & Nesmachnow, S. (2013). Parallel metaheuristics: recent advances and new trends. *International Transactions in Operational Research*, *20*(1), 1-48.

[79] Bożejko, W., Pempera, J., & Smutnicki, C. (2013). Parallel tabu search algorithm for the hybrid flow shop problem. *Computers & Industrial Engineering*, *65*(3), 466-474.

[80] Hijaze, M., & Corne, D. (2009). An investigation of topologies and migration schemes for asynchronous distributed evolutionary algorithms. In *2009 World Congress on Nature & Biologically Inspired Computing (NaBIC)* (pp. 636-641). IEEE.

[81] Chang, Y. L., Chen, K. S., Huang, B., Chang, W. Y., Benediktsson, J. A., & Chang, L. (2011). A parallel simulated annealing approach to band selection for high-dimensional remote sensing images. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, *4*(3), 579-590.

[82] Peres, F., & Castelli, M. (2021). Combinatorial optimization problems and metaheuristics: Review, challenges, design, and development. *Applied Sciences*, *11*(14), 6449.

[83] Bachelet, V., Hafidi, Z., Preux, P., & Talbi, E. G. (1998). Vers la coopération des métaheuristiques. Calculateurs parallèles, réseaux et systèmes répartis, 9(2), 211-223.