Master's Thesis

**Master's Degree in Industrial Engineering**

# Metaheuristic algorithm for pickup and delivery vehicle route problems with time windows


**Author:** Pau Daniel Fuentes
**Supervisor:** Dr. Kazushi Sano
**Summon:** July 2020

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

ETSEIB UPC

Nagaoka University of Technology

長岡技術科学大学

# Summary

This project tackles the study, the design, the implementation, and the analysis of the results of a metaheuristic algorithm for a vehicle route problem, in particular, for the pickup and delivery problem with time windows. This problem concerns the transportation of goods between paired pickup and delivery locations where the pickup locations must be visited before the corresponding delivery locations by the same vehicle.

The project is divided into two parts. The first part is a theoretical part in which the Vehicle Route Problem (VRP) is introduced, and its different versions are explained. Moreover, the mathematical model for the Pickup and Delivery Problem with Time Windows (PDPTW) is formulated, and its functioning is analysed.

The second part of the project is practical. It includes the design of a metaheuristic algorithm based on a tabu neighbourhood search. Later, this algorithm has been tested aiming the minimum travel distance.

Finally, the quality of the results obtained have been analysed. Using the public data provided by Li & Lim Benchmark, the algorithm has been validated, and the results obtained have been compared with the ones provided by one of Europe's largest independent research organizations known as SINTEF.

# Contents

# 1. Introduction

## 1.1. Motivation

Transportation is the movement of humans, animals, and goods from one place to another. It enables trade between people, which is essential for development in every stage of civilization. Recently, due to the current state of the world economy caused by the COVID-19, it has been proved that transportation has become the cornerstone of how smoothly the economy and society function.

Logistics management requires considerable attention to reduce transportation costs. Advancements in logistics planning systems encourage industrial players to reduce a vast amount of money spent on distribution and transportation. They attempt to create a competitive advantage over competitors regarding cost leadership. Among IT tools, optimization tools are one of the most potent equipment for logistics planning.

Due to the recent emergence of e-commerce, Third-party Logistics Providers (3PLs) and Logistics Service Providers (LSPs) market have become highly competitive at the regional, national, and international levels. Moreover, in the last few decades, operating costs have increased dramatically. To gain a competitive advantage, a large number of 3PLs and LSPs have sought to lower the cost of their operations, which has led to rising interest in the applications of Operations Research (OR). Due to advances in computer technology, OR techniques have become more powerful, capable of efficiently solving optimization problems in a reasonable timeframe. With the development of the Geographic Information System (GIS) and the Global Positioning System (GPS), OR techniques turn transportation planning into reality.

## 1.2. Goals and scope of project

This project's primary goal is to study and develop a metaheuristic algorithm to solve the pickup and delivery vehicle route problem with time windows. The idea is to propose a method in which its strength remains on its simplicity. The heuristic relies on simple principles to construct a solution, to improve it, to update the solution pool, and to generate paths based on initial and guiding solutions.

Another goal is to tackle real-life routing problems arising in logistics businesses by efficiently implementing the proposed algorithm developed in the thesis. This algorithm has been tested with Li & Lim Benchmark data provided by SINTEF[1] to validate the quality of the results and its utility.

[1] SINTEF: https://www.sintef.no/projectweb/top/pdptw/li-lim-benchmark/

The scope of this thesis is structured as follows:

- Understanding and studying different versions of Vehicle Route Problem (VRP).
- Researching of the mathematical model, already designed by Ropke and Pisinger[2], of the Pickup and Delivery Vehicle Route Problem with Time Windows (PDPTW) to use its formulation for the algorithm development.
- Developing an algorithm based on a tabu neighbourhood search to solve the problem using heuristic procedures and metaheuristic optimization aiming to get as close as possible to the optimal solution.
- Testing the algorithm's utility with Li & Lim Benchmark data, analysing the quality of the results, and comparing them with those provided by SINTEF[1].

## 1.3. Related Work

The PDPTW was originally formulated by Dumas, Desrosiers, and Soumis[3] in 1991. Since then, the problem and its variants have been studied extensively in the literature, and several attempts have been made using exact methods to solve the problem. Although many solution methods are proposed, exact methods are only useful in a reduced dimension of the problem. As the problem is a Non-deterministic Polynomial-time hardness (NP-hard), which means that the computational time required to solve it increases exponentially is the size of the problem expands, the focus has been mostly on heuristics algorithms that have only been widely proposed for PDPTW over the last decade. As developing heuristics algorithm is our focus in this study, only the advances of the heuristic methods for PDPTW are reviewed.

In the early stage, metaheuristics were usually built around available local search operators that have been worked well on VRP. One of the very first heuristics for solving PDPTW was introduced by Nanry and Barnes[4]. The authors presented a reactive tabu search formalized by Fred W. Glover in 1986; this method approaches using three distinct move neighbourhoods, which are hierarchically performed after the initial solution is constructed. The process can obtain optimal or near-optimal solutions with relatively small computational effort compared to exact methods back then, which can only solve relatively minor problems. Its efficiency is another substantial evidence for researchers interested in heuristic methods and wants to solve PDPTW.

[2] Ropke & Pisinger: An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows.
[3] Dumas, Desrosiers, and Soumis: The pickup and delivery problem with time windows.
[4] Narny and Barnes: Solving the pickup and delivery problem with time windows using reactive tabu search.

Soon later, Bent and Van Hentenryck[5] proposed a two-stage hybrid heuristic for the PDPTW based on Large Neighbourhood Search (LNS). The algorithm was tested on the problem instances submitted by Li and Lim[6]. The results showed that their LNS outperformed previous methods.

A few years later, an essential extension for LNS, Adaptive Large Neighbourhood Search (ALNS), was developed in the work of Ropke and Pisinger[2]. This adaptive feature significantly influences the quality of solutions and has made it probably the most effective metaheuristic for PDPTW so far, with results reported for up to 1000 locations.

Over the last few years, hybrid metaheuristics have attracted more attention from researchers, thanks to its immense success in solving various kinds of combinatorial optimization problems. Many authors encourage the use of population-based approaches hybridized with single-solution procedures. Hashimoto and Yagiura[7] suggested the hybridization of the variants of population-based approaches with local search or other metaheuristics for solving VRPs and generated very competent results.

[5] Bent and Van Hentenryck: A two-stage hybrid algorithm for pickup and delivery vehicle route problems with time windows.

[6] Li and Lim: A metaheuristic for the pickup and delivery problem with time windows.

[7] Hashimoto and Yagiura: A path relinking approach with an adaptive mechanism to control parameters for the vehicle routing problem with time windows.

# 2. Vehicle Route Problems

Nowadays, one of the most common problems with more applications in the real world is the optimization of vehicle routes. It has been proved that using an effective distribution route; shipping cost can reduce by 10% or even 20%.

Route problems can be classified into two main types; the first one is known as Nodes Route Problems (NRP), while the second one is named Arcs Route Problems (ARP). In this thesis, only the latter type is considered because this is the one used for distribution and package delivery between depots and customers. Some real examples of this kind of transportation are food delivery, disabled people transportation…

## 2.1. Main factors

The distribution of goods involves many factors such as service, distance, time, and a set of customers and vehicles. These goods are in depots and are transported by drivers to their destinations, taking the road network on which vehicles circulate as another factor to be considered.

One of the most effective ways to draw up this kind of problem is by using graph theory, where a graph is composed of a group of nodes and their connections. The nodes symbolize both types of customers, pickup in depots and delivery to destinations, while their links represent the road network between these customers. These connections could be one-way directions (edges) or two-ways directions (arcs), depending on what kind of network they represent. Each link has a cost associated, usually it is related to the amount of time or the travelled distance that is needed to go through it. Moreover, this cost also depends on the type of vehicle used and the time of the day that it is used. For example, driving through the main road at peak hour is not the same as driving it at night.

Regarding what the customers represent in the apexes of the graph, each of them has a cost associated with the product's quantity that needs to be picked up and delivered in each node. Furthermore, each customer could be visited only during a specific time window reducing its availability to a short period. Another factor to consider to adjust the graph representation as much as possible to reality is the well-known service time referred to the time spent loading and unloading the vehicle for each customer. The service time is also affected by the features of the vehicle used.

The capacity of the vehicles is limited by weight, volume, and the number of pallets that can be loaded. In some cases, depending on the load, a specific vehicle might be needed to deliver the shipment, which adds a limitation on the number of customers that each vehicle can visit. Sometimes, the vehicles are also limited by the type of road where they can or not drive through.

As an example, in some cities, the use of big vehicles is restricted only to industrial areas, and they cannot get into the heart of the town.

Finally, each vehicle is associated with a driver that is also limited by its company that works for as well as the country where the delivery is taking place. The laws and the rights of the workers change, affecting the maximum hours that a driver can work, the break time, the non-stop time on the road, etc. Generally, all drivers' limitations are considered as vehicle limitations.

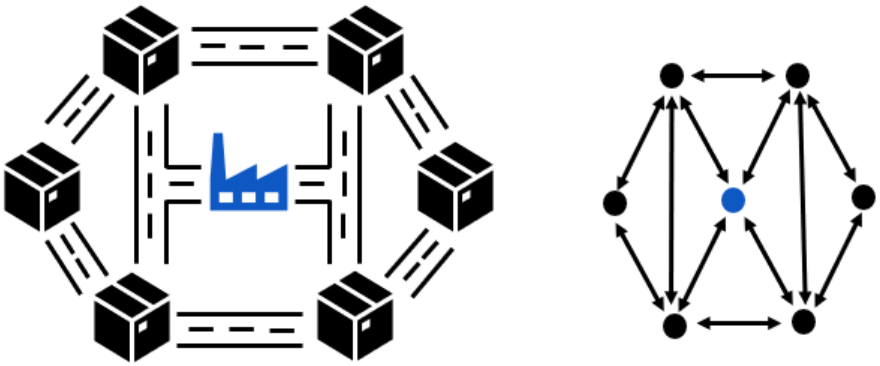

*Figure 1: Representation of a basic vehicle route problem using graph theory.*

## 2.2. Objectives

The vehicle route problem arises as follows. Owning a fleet of vehicles, located in one or more warehouses, a set of geographically distributed customers must be served. Therefore, the most common objective is to satisfy the demand for this group of customers at the lowest cost, finding the optimal route that starts and ends in the warehouse.

In general, there are multiple objectives in vehicle route problems, and some of them are:

- Minimize the transportation cost, taking into consideration distances and time.
- Minimize the number of vehicles or drivers used to attend the customers.
- Minimize the waiting time for each vehicle when the problem presents time windows.
- Minimize the number of customers that haven't been served.

These objectives have to be achieved without violating any constraint. The solution must be feasible by respecting the vehicle capacity, the customers' priority, the existing road network, etc. In case that the problem includes time windows, they must be added as a constraint depending on the type of problem to be solved. If it is a hard-time-window problem, delays are not allowed under any circumstances, however, if the problem is a soft-time-window, customers can be served late and the time windows can be violated, including a penalty cost as results of the possible delays.

## 2.3. Versions of vehicle route problem

There are many variants of the vehicle route problem (VRP); the most prominent are described below:

- **Capacity Vehicle Route Problem**

  The Capacity Vehicle Route Problem (CVRP) is the most basic version of the variants that are going to be described in this chapter. The following versions are based on this one but presenting more complexity. In the CVRP, the load is collectively picked up or delivered to all the customers, and the load's quantity is previously known.

  The vehicles are all identical and they operate from the same central warehouse. The primary constraints are that the maximum capacity of each vehicle cannot be exceeded and that the customers can be visited only once, which means the load cannot be split. The objective of this problem is minimizing the cost associated with the entire service.

  This problem presents different versions of itself, one of them is when the number of vehicles is variable, and each vehicle has its own cost associated. The other is when the vehicle's capacity is not uniformed, etc.

- **Vehicle Route Problem with Time Windows**

  The Vehicle Route Problem with Time Windows (VRPTW) is an extension of the CVRP. Its main characteristic is that each customer adds a time window with it. This time window is a period composed of an initial time and a final time that represents the customer's availability.

  A variable related to time needs to be defined: it starts when the vehicles leave the warehouse and end when all of them have come back after having served all the customers. This variable must include the travel time spent by the driver inside the vehicle, the serving time associated with the amount of time needed to load or unload the vehicle, and also the waiting time filled when the customer is visited before its initial time window. If a vehicle visits a customer after its final time window, the route is no longer valid and cannot be included in the problem's solution.

  The principal constraints of this type of problem are: each vehicle departs and comes back only once from the warehouse; each customer can be visited once and only by a vehicle, the capacity of the vehicle cannot be overpassed; and the last one, time windows must be respected.

- **Vehicle Route Problem with Backhauls**

  This version of the problem is also an extension of the CVRP, and its main difference is that customers can demand or provide goods to the vehicle. As a consequence, customers can be divided into two main groups. The first one includes all the customers that require or receive goods; they are known as Linehaul customers or Deliveries; the second group is composed of those customers who provide products and they are named as Backhaul customers or Picked up. All the vehicles start in a central warehouse where the delivery load is stocked and at the end of the route and the goods picked from the customers are stocked in the same warehouse.

  The principal constraint of the Vehicle Route Problems with Backhauls (VRPB) is that for each route, firstly linehaul customers must be served and afterwards backhauls customers can be visited. Also, as an extension of the CVRP, the maximum capacity of the vehicle must be respected all the time, during the delivery and also on the pickup.

- **Pickup & Delivery Vehicle Route Problem**

  The Pickup and Delivery Vehicle Route Problem (PDVRP) is close to real cases, which explains why it is the most analysed version. This problem is also based on the CVRP; however, it is profoundly different from the ones previously described. In this version, some customers provide goods that must be delivered to another customer, which allows classifying all customers into pickup or delivery destinations.

  The primary constraint added to this problem is the fact that the load needs to be picked up before it can be delivered. Delivery destinations can be visited if their pickup destination has been visited earlier by the same vehicle. In this version, loads' transshipment is not allowed. Moreover, the constraint related to the vehicle capacity has to be constantly considered.

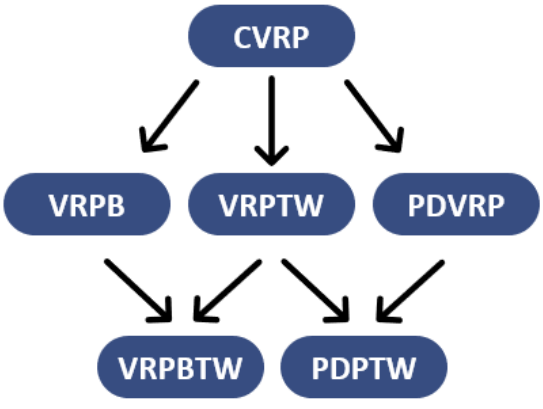


*Figure 2: Versions of the vehicle route problem*

# 3. Pick up & Delivery Problem with Time Windows

## 3.1. Problem description

The PDPTW being considered in this study is a generalization of the VRPTW in which each customer has a given origin and a given destination. With that being said, moving goods from depot to customers is no longer the case; instead, a customer requires products to be transported from a pickup location to a delivery location.

Each demand for moving shipment within the pickup-delivery pair is called a request. Each request is associated with two-time windows: a pickup time window at origin indicating when the load can be picked up, and a delivery time window at the destination for when the shipment can be dropped off. Moreover, service at a location (pickup/drop-off time, dispatch time, etc.) requires a determined amount of time to be done, denoted by service time.

On the one hand, a vehicle is allowed to arrive at a location even before the start of the time window, but it must wait until the time window opens available for service, creating what is known as a "waiting time". It is reasonable that a vehicle has to wait if its untimely arrival occurs. On the other hand, tardiness is not allowed in any case. That said, all time window constraints must be satisfied, vehicles should arrive either before or within the time window of a location. This is called hard-time-windows, unlike its opposite variant, soft-time-windows, which allow violating timely matters.

Each request is assigned to a set of vehicles, each of which has a limited capacity. That means vehicles are not allowed to carry overweight. In the beginning, a vehicle leaves depot to start its tasks and return there after all its assigned customers have been served. Pick up, carry, then drop off a shipment is carried out by the same vehicle. In other words, there are no transhipments among vehicles during daily tasks. A variant of this problem with transhipment points has already been studied by Liana Van Der Hagen (2017)[8].

A limited number of vehicles sometimes might generate situations where some requests cannot be assigned to any vehicle without violating the constraints mentioned above. Then a request bank is created to store such requests. However, in this thesis, having demands in the request bank is not allowed for the final solution, but in a heuristic transition stage, the request bank may be used.

The task in PDPTW is to construct a solution comprised of a set of feasible routes for vehicles to serve all requests. A path is defined as possible if time windows and vehicle capacity constraints are

[8] Liana Van Der Hagen: The Pickup and Delivery Problem with Time Windows: An Adaptive Large Neighborhood Search heuristic.

satisfied along the route. A shipment is picked up before being dropped off at the corresponding delivery point by the same vehicle. Most importantly, the solution should be able to minimize the cost function in an attempt to optimize the operational cost. The objective of the problem is to reduce a weighted sum comprising of the following three components: the sum of distance travelled by the vehicles, the amount of time spent by each vehicle, the number of requests in the request bank. These three components are weighted by the coefficients $\alpha, \beta,$ and $\gamma$, respectively.

Even though it is possible to place a request into request bank as a result of trying to cut losses by denying demands not worth serving, it should not be the case when there are still enough vehicles available to be in service. A request should only place in the request bank if it is impossible to serve with given strict constraints (too tight time windows between pickup and delivery locations, overweight loads, etc.). Trying to deliver as many requests as possible is translated as assigning a high value to alpha.

## 3.2. Mathematical model

Ropke and Pisinger[2] gave a mathematical model of the PDPTW. Since then, many authors have used it for their researches as Nagata and Kobayashi[9], and Nguyen[10].

The PDPTW involves the design of optimal vehicle routes given a number of customers requests, $r_i = (i^+, i^-)$, consisting of a pickup node $i^+$ and a delivery node $i^-$. Each node $i$ is assigned a time window $[a_i , b_i]$, which means that no service can be provided at the location before the start of the time window nor after the end of the time window. A vehicle is allowed to arrive at the customer location before the beginning of the time window, but has to wait to start its service.

Let $N = P \cup D$ be the set of all customer nodes, with $P$ the set of pickup nodes and $D$ the set of delivery nodes, where $|P| = n$. Let $K$ be the set of all available vehicles, with $|K| = m$. Each request $r_i$ is assigned a subset of vehicles $Kr_i$, which can serve this request. Analogous, $P_k$ and $D_k$ contain all pickup and delivery nodes, respectively, that can be served by vehicle $k$ and $N_k = P_k \cup D_k$. Requests that cannot be served by every vehicle are called special requests.

Each vehicle $k$ has a limited capacity $C_k$ and a given start terminal, $\tau_k$, and an end terminal, $\tau'_k$. These terminals also have a specified time window, vehicles have to leave the start terminal at $a\tau_k$ and must return to the end terminal before $b\tau'_k$. The graph $G = (V, A)$ consists of all customer nodes, start terminals and end terminals and arcs $A = V \times V$. The graph $G_k = (V_k, A_k)$ is the subgraph

[9] Nagata & Kobayashi: Guided Ejection Search for the Pickup and Delivery Problem with Time Windows.

[10] Anh Minh Nguyen: A Hybrid Adaptive Large Neighborhood Search for Pickup and Delivery Problem with Time Windows

for vehicle $k$, where $V_k$ consists of the nodes that may be visited by vehicle $k$, that is $V_k = N_k \cup \{\tau_k\} \cup \{\tau'_k\}$ and $A_k = V_k \times V_k$.

The distance and travel time from node $i$ to node $j$ are denoted by $d_{ij}$ and $t_{ij}$, respectively. The travelled time and the distances are assumed to satisfy the triangle inequality. Furthermore, $s_i$ indicates the duration of service at node $i$ and $l_i$ the load that should be carried from node $i$, if node $i$ is a pickup node, or the load that should be transported to that node if $i$ is a delivery node. When constructing the routes, it should be considered that the pickup location and the delivery location are on the same route, and the pickup location must be visited before the delivery location.

There are four main types of decision variables in this mathematical model. The first one is $x_{ijk}$, a binary variable, equal to 1 if the edge between node $i$ and node $j$ is used by vehicle $k$ and equal to 0 alternatively. It might be that the set of available vehicles is not able to serve all requests. These requests are placed in a so-called request bank. In this thesis, it is not allowed for the final solutions to have requests in the request bank, but in a transition stage of the heuristic, the request bank may be used. The second variable is $z_i$ and is also binary, it is set to 1 if the request corresponding to pick up node $i$ is placed in the request bank, otherwise it is set to 0. The other decision variables are $S_{ik}$ and $L_{ik}$. They denote the start time of service by vehicle $k$ at node $i$ and an upper bound on the number of goods in vehicle $k$ after visiting node $i$, respectively. $T_i = \sum_{k \in K} \sum_{j \in Vk} S_{ik} x_{ijk}$ indicates the time at which service starts at node $i$.

As a summary of all the above, the mathematical formulation of the problem is as follows.

- **Sets:**

  $N$ : set of nodes, $N = \{0, \ldots, 2n+1\}$
  $P$ : set of pickup nodes, $P = \{1, \ldots, n\}$
  $D$ : set of delivery nodes, $D = \{n+1, \ldots, 2n\}$
  $K$ : set of vehicles, $|K| = m$
  $K_i$ : set of vehicles that can serve request $i$, $K_i \subseteq K$
  $P_k$ : set of pickups that can be distributed by vehicle $k$, $P_k \subseteq P$
  $D_k$ : set of deliveries that can be served by vehicle $k$, $D_k \subseteq D$

- **Parameters:**

  $q_i$ : demand/supply at node i ($i \in V$)
  $C_k$ : the capacity of the vehicle $k$ ($k \in K$)
  $a_i$ : earliest time to begin service at node $i$ ($i \in V$)

$b_i$ : latest time to begin service at node $i$ ($i \in V$)

$s_i$ : service duration at node $i$ ($i \in V$)

$d_{ij}$ : distance of arc ($i$, j) ($i, j \in A$)

$t_{ij}$ : travel time of arc ($i$, j) ($i, j \in A$)

- **Decision variables:**

$x_{ijk}$ : Value = 1 if arc ($i$, j) is travelled by vehicle $k$; Value = 0 otherwise

$z_i$ : Value = 1 if request $i$ is placed in the request bank; Value = 0 otherwise

$S_{ik}$ : Non-negative number that indicates when the vehicle $k$ starts the service at location $i$

$L_{ik}$ : Non-negative number that indicates the amount of goods in the vehicle $k$ after visiting node $i$

- **Objective function:**

$$\text{Min } \alpha \sum_{k \in K} \sum_{(i,j) \in A} d_{ij} x_{ijk} + \beta \sum_{k \in K} (S_{\tau'_k,k} - a_{\tau_k}) + \gamma \sum_{i \in P} z_i$$

The objective function minimizes a weighted sum of the total distance travelled ($\alpha$), the time spent by all vehicles ($\beta$), and the number of requests that are placed in the request bank ($\gamma$).

- **Constraints:**

(1) $\sum_{k \in K} \sum_{j \in N_k} x_{ijk} + z_i = 1$ $\quad \forall i \in P$

(2) $\sum_{j \in V_k} x_{ijk} - \sum_{j \in V_k} x_{i,n+i,k} = 0$ $\quad \forall k \in K, \forall i \in P_k$

(3) $\sum_{j \in P_k \cup \tau'_k} x_{\tau_k,j,i} = 1$ $\quad \forall k \in K$

(4) $\sum_{i \in D_k \cup \tau_k} x_{i,\tau'_k,k} = 1$ $\quad \forall k \in K$

(5) $\sum_{i \in V_k} x_{ijk} - \sum_{i \in V_k} x_{jik} = 0$ $\quad \forall k \in K, \forall j \in N_k$

(6) $x_{ijk} = 1 \Rightarrow S_{ik} + s_i + t_{ij} \le S_{jk}$ $\quad \forall k \in K, \forall (i, j) \in A_k$

(7) $a_i \le S_{ik} \le b_i$ $\quad \forall k \in K, \forall i \in V_k$

(8) $S_{ik} \le S_{n+1,k}$ $\quad \forall k \in K, \forall i \in P_k$

(9) $x_{ijk} = 1 \Rightarrow L_{ik} + l_j \leq L_{jk}$ $\forall k \in K, \forall (i, j) \in A_k$

(10) $L_{ik} \leq C_k$ $\forall k \in K, \forall i \in V_k$

(11) $L_{\tau_k k} = L_{\tau'_k k} = 0$ $\forall k \in K$

(12) $x_{ijk} \in \mathrm{B}$ $\forall k \in K, \forall (i, j) \in A_k$

(13) $z_i \in \mathrm{B}$ $\forall i \in P$

(14) $S_{ik} \geq 0$ $\forall k \in K, \forall i \in V_k$

(15) $L_{ik} \geq 0$ $\forall k \in K, \forall i \in V_k$

Constraint (1) specifies that either a pickup node is in one route or the corresponding request is placed in the request bank.

Constraint (2) ensures that the pickup location and the delivery location are visited by the same vehicle and check that the delivery node is visited if and only if the pickup node is visited.

Constraints (3), (4), and (5) specify that each vehicle has to leave its start terminal, arrive at its end terminal and when visiting a customer node, it must arrive at that node and leave that node, respectively.

Constraints (6) and (7) ensure that service cannot start at a location before the vehicle is able to arrive at that location. It cannot start before the beginning of the time window or after the end of the time window at that location, respectively.

Constraint (8) is included to ensure that each delivery location is visited later than its corresponding pickup location.

Constraints (9), (10), and (11) ensure that the loads on the vehicles are set correctly and satisfy the capacity constraints.

Finally, constraints (12), (13), (14), and (15) specify the domain of the variables, in other words, if they are binaries, non-negatives, etc.

# 4. Heuristic algorithm design

The Pickup and Delivery Problem with Time Windows (PDPTW) is well known for being a Non-deterministic Polynomial-time hardness (NP-hard) problem, which means that the computational time required to solve it increases exponentially as the size of the problem expands. Two specific types of algorithms can be used to solve this kind of problem.

The first type, named exact algorithm, is based on mathematical models to provide the optimal solution and guarantee it in any case. However, its main drawback is the computational time required to solve the problem, especially if it is a NP-hard, and that's the reason for the existence of a second type. It is known as a series of heuristic algorithms; they use sophisticated mathematical optimization methods as branch-and-cut and neighbourhood-search. As a result, the execution time is shorter in heuristics algorithms than in exact algorithms. Although its process timing is fast, the optimal solution is not guaranteed, and the results could not be good enough for its utility.

The two main objectives of this thesis are to design an algorithm that minimizes the vehicle travelled distance for a PDPTW, and validate the quality of the developed algorithm in real situations. Due to the lack of real data, the algorithm has been proved with the Li & Lim Benchmark data that involve unreal instances between 100 and 1000 customers. The design of a heuristic algorithm has been considered more appropriate for this situation due to the dimension of these instances.

The heuristic procedures that create an algorithm can be divided broadly into three different categories: construction heuristics, local search heuristics, and metaheuristics. All these categories have been used in the algorithm designed, and they are thereupon explained.

## 4.1. Construction heuristics

Laporte and Semet[11] defined construction heuristics as follows:

> *Constructive heuristics gradually build a feasible solution while keeping an eye on solution cost, but they do not contain an improvement phase per se.*

The main goal of a construction heuristic is to provide a feasible solution with several iterative and straightforward methods. The acquired solution might be far from the optimal result, but at least it is a valid solution achieved in a short time. A fast and useful constructive heuristic is vital from a practical point of view as many real-world applications of heuristics require an instant response. In

[11] Laporte and Semet: A Guide to Vehicle Routing Heuristics

a vehicle routing application, one needs to quickly reconstruct a part of the solution if an incident happens while carrying out the plan or if a customer calls in with a new transportation task and wants to know if the task can be carried out. Fast construction algorithms are often the preferable algorithm for such situations and substantial problems containing thousands of customers.

In the algorithm design of this thesis, the constructive heuristic is based on a dynamic greedy algorithm. According to Paul E. Black[12], a greedy algorithm follows the problem-solving heuristic of making the locally optimal choice at each stage with the intent of finding a global optimum.

### 4.1.1. Greedy heuristic

The greedy heuristic proposed in this project to solve the PDPWT works as follows. The heuristic procedure function receives as an input the *Number of Vehicles* that the initial solution is required to have and the *Area Radius*. It starts setting up all the *Global parameters* to their initial states like the *Total number of Locations* that still need to be visited (*TL*), the *Global Time*, *Global Distance, Global Service Time*, and *Global Waiting Time*. After setting these variables, the first iteration process starts, and it runs until all the vehicles available have been used. Once inside the first iteration, for each vehicle, the *Vehicle parameters* (*Total Time, Travel Time, Service Time, and Waiting Time*) are set up to their initial states, which include the locations that have already been visited by this vehicle, and also its initial location and load. The heuristic procedure continues selecting all the *Pickup Locations* (*PiL*) from the *TL*, and gets into the second iterative process. This step lasts until the current vehicle runs out of locations to be visited or time to go back to warehouse (*Return Time*).

Inside the second iteration process, the search for the next optimal location starts. This optimal location will be saved as the *Vehicle's Next Location* (*NL*). First of all, the procedure selects the locations where the *Vehicle Load* (*VL*) doesn't exceed the *Maximum Vehicle Load* (*MVL*) in order to avoid overloading. These candidates are placed in a list known as *Possible Locations* (*PoL*). Then, from these *PoL*, the open ones, are classified as *Open Possible Locations (OPL)*. This means that the current *Total Time* + *Travel Time* is between the *Initial Time Window* (*TWI*) and the *Final Time Window* (*TWF*). In case that the *OPL* list is empty, the *NL* to be visited is going to be the one that presents less *Travel Time* + *Waiting Time* from the current *Vehicle Position* (*VP*) of the *PoL* list. If this list is empty too, the second iteration process will be broken.

After obtaining the *OPL* list, the next step is identifying the locations that remain inside the *Area Radius* previously defined and add them into a new list named *Area Locations* (*AL*). The *NL* is filled with the earliest location to be closed of *AL*. However, if the *AL* list shows up as empty, the *NL* variable is completed with the closest location of the *OPL*. Once the *NL* has been saved, a check

[12] Paul E. Black: Dictionary of Algorithms and Data Structures

with the *Priority Destinations* (*PD*) must be done. In case that one of the *PD* becomes an unavailable destination due to the current assigned *NL*, the *NL* must be switched into this specific *PD*.

Finally, when the *NL* has been established, the *Vehicle parameters* are updated, and the current *NL* is removed from the *TL* and *PiL*. If the *NL* is one of the *PiL*, its destination is added to the *PD* list. The second iteration continues until it runs out of possible locations. At this point, a new vehicle is set up and the first iteration process proceeds until all the locations have been visited or all the vehicles have been used. When it happens, the *Global parameters* are updated, and the final solution with the *Global* and each *Vehicle parameters* is returned by the function.

**HEURISTIC PROCEDURE**

**Function** ConstructiveHeuristic (*Number of Vehicles*, *Area radius*):
  *Global parameters* → Initial states
  **Iteration 1:** while *Vehicle* ≤ *Number of Vehicles*
    *Vehicle parameters* → Initial states
    *PiL* = Pickup locations of *TL*
    **Iteration 2:** while *Total Time* + *Return Time* ≤ $TWF_W$
      *PoL* = Locations of *PiL* with *VL* ≤ *MVL*
      *OPL* = Locations of *PoL* with *TWI* ≤ *Total Time* + *Travel Time* ≤ *TWF*
      **If** *OPL* is empty:
        **If** *PoL* is not empty:
          *NL* = Minimum (*Travel Time* + *Waiting Time*) of *PoL*
        **If** *PoL* is empty:
          Break iteration
      **If** *OPL* is not empty:
        *AL* = Locations of *OPL* with *Distance* ≤ *Area radius*
        **If** *AL* is not empty:
          *NL* = Minimum (*TWF*) of *AL*
        **If** *AL* is empty:
          *NL* = Minimum (*Distance*) of *OPL*
      **If** *PD* is not empty:
        *ND* = *PD* locations with $TWF_{PD}$ < *Total Time* + $Travel\ Time_{NL}$ + $Service\ Time_{NL}$ + $Travel\ Time_{PD}$
        *NL* = Minimum (*Distance*) of *ND*
      *Vehicle parameters* → Update to *NL*
    Return the vehicle to the warehouse and *Vehicle parameters* → Update to warehouse
    *Vehicle* = +1
  *Global parameters* → Update to all vehicles
  **Return** *Global parameters* and *Vehicle parameters* of each vehicle

The strength of this heuristic procedure is its flexibility to modify the criterion used to select the next optimal location in each stage. A simple modification on its input makes this alteration on the *Area Radius*.

On the one hand, the criterion can be focused on *Distance* rather than *Time* by setting up the *Area Radius* into 0, which makes it unable for locations to be inside the area range. In this situation, the time windows are not considered as a priority as the optimal candidate is selected for being the closest valid location to the current *Vehicle Location*.

On the other hand, the heuristic criterion can be focused on *Time* instead of *Distance* by establishing an *Area Radius* wide enough to cover all the available locations. When this situation takes place, the optimal location is chosen for being the earliest to be closed, which means less *TWF* related to the current *Total Time*.

However, if *Area Radius* is defined in a medium-range, the criterion priority is hybrid, taking into account both *Time* and *Distance* taking into account. Of course, the wider it gets, the more critical the *Time* becomes and vice versa.

### 4.1.2. Initial solution

The heuristic procedure generates an initial solution depending on the *Number of Vehicles* and the *Area Radius* given as an input; this solution is composed of two main parts. The first one, the *Global parameters* of the solution, which provide information about the general settings of the result, taking into consideration all the vehicles that have been used. One of the parameters that are shown is the list of locations that haven't been served, which indicates if the solution is complete or not. As it has been said in the mathematical model, the initial solution can include a request bank, but it must be empty in the final solution. The second part of the initial solution is named *Vehicle parameters*. It is formed by all the individual features of each vehicle used, like the route of locations that it follows, *Total Distance*, and *Total Time*. In the figure 3, a small example of an initial solution can be appreciated.

```
GLOBAL PARAMETERS
Locations not visited: [1, 2, 4, 6, 9, 11, 26, 29, 34, 29]
Global Distance: 985.05
Global Time 9449.11

VEHICLE PARAMETERS
Vehicle: 1 - Route: [0, 5, 3, 7, 8, 10, 30, 28, 36, 105, 22, 21, 75, 0] - Total Distance: 128.46 - Total Time: 1129.66
Vehicle: 2 - Route: [0, 20, 24, 25, 27, 38, 64, 102, 51, 101, 50, 52, 49, 47, 0] - Total Distance: 145.4 - Total Time: 1223.43
Vehicle: 3 - Route: [0, 67, 65, 63, 62, 74, 72, 61, 68, 66, 69, 0] - Total Distance: 59.41 - Total Time: 1027.2
Vehicle: 4 - Route: [0, 43, 42, 41, 40, 44, 46, 45, 48, 0] - Total Distance: 56.7 - Total Time: 776.7
Vehicle: 5 - Route: [0, 90, 87, 86, 83, 82, 84, 85, 88, 89, 91, 0] - Total Distance: 76.08 - Total Time: 976.08
Vehicle: 6 - Route: [0, 13, 17, 18, 19, 15, 16, 14, 12, 0] - Total Distance: 95.89 - Total Time: 815.89
Vehicle: 7 - Route: [0, 98, 96, 95, 94, 92, 93, 97, 106, 100, 99, 0] - Total Distance: 95.96 - Total Time: 905.96
Vehicle: 8 - Route: [0, 32, 33, 31, 35, 37, 39, 23, 103, 0] - Total Distance: 97.96 - Total Time: 835.0
Vehicle: 9 - Route: [0, 57, 55, 54, 53, 56, 58, 60, 59, 0] - Total Distance: 101.89 - Total Time: 821.89
Vehicle: 10 - Route: [0, 81, 78, 104, 76, 71, 70, 73, 77, 79, 80, 0] - Total Distance: 127.3 - Total Time: 937.3
```

*Figure 3: Initial solution generated by the heuristic procedure*

## 4.2. Local search heuristics

Local search heuristics are heuristics that take a solution as an input, modify this solution by performing a sequence of operations on the solution, and produce a new, hopefully, improved solution. At all times, the heuristic has a current solution, and it modifies this result by evaluating the effect of changing the solution systematically. If any of these changes leads to an improved solution, the current solution is replaced by the new improved result, and the process is repeated until no more upgrades are found.

Local search heuristics are often built on neighbourhood moves that make small changes to the current solution, such as moving a request from one route to another or exchanging two requests as Nanry and Barnes[4] and Li and Lim[6] proposed.

Laporte and Semet[11] defined the term improvement heuristic, and it can be used to describe a local search heuristic that only performs operations that improve the objective of the solution. In this thesis, the algorithm sometimes performs changes that lead to a solution that is worse than the current. This is done as one can hope to find an even better solution after a few more changes. Because of this, the optimization process has been considered as a local search heuristic instead of an improvement heuristic. The designed algorithm of this project is composed of three types of local search heuristics that are detailed in the sections below.

### 4.2.1. Improvement heuristics

As it has been mentioned above, an improvement heuristic aims for a better result for the solution given as an input. Two main functions have been designed with this goal to fulfil the proposed algorithm.

The first improvement heuristic is named *Improve Route*, as its name reveals it has the target to improve an existing route of the initial solution. This function only receives one input, and it is a single route, which means that this procedure is not going to add or remove locations to any other place. Moreover, it is not able to mix locations with other routes or even the request bank.

The function starts saving the *Initial Route* and its state as the initial *Best Route* and *Best State*. The state of the route refers to its current situation, which comprises the *Total Distance,* the existence of delays, overload, or illogical order in the solution. The procedure continues moving each *Location* of the *Best Route* to each possible *Position* inside the route, creating several *New Routes*. If any of these *New Routes* presents a better state that the current *Best State,* this particular route and its state become the new *Best Route* and *Best State*. Finally, the function returns the *Best Route*.

**IMPROVEMENT HEURISTIC 1**

```
1  Function ImproveRoute (Initial Route):
2     Best Route & Best State = Initial Route & State of the Initial Route
3     For each Location in the Best Route
4        For each Position in the Best Route
5           Move the actual Location to the current Position
6           New State = State of the New Route
7           If the New State is better that the Best State
8              Best Route & Best State = New Route & New State
9     Return Best Route
```

The other improvement heuristic that has been developed has the objective of finding a better result for a whole solution. It has been named *Improve Solution* because it involves all the existing routes of a solution. It mixes the routes to achieve a better global solution even if sometimes it makes them individually worse. The request bank is not used in this procedure, so no new locations are added or removed from the solution.

This improvement procedure receives an *Initial Solution* as an input, which is kept along with its state as the *Best Solution* and *Best State*. The state of the solution has the same meaning as the state of the route previously explained. However, this one refers to the global solution of all the routes. The function proceeds to move each *Location* of each *Route* of the *Best Solution* to all the possible *Positions* in every *Route*. As a result, multiple *New Solutions* are generated, in case that any of them shows a better state than the actual *Best State,* this *New Solution* and its state are saved as the new *Best Solution* and *Best State,* respectively. In the end, the procedure returns the *Best Solution*.

**IMPROVEMENT HEURISTIC 2**

```
1  Function ImproveSolution (Initial Solution):
2     Best Solution & Best State = Initial Solution & State of the Initial Solution
3     For each Route in the Best Solution
4        For each Location in the Route
5           For each Position of each Route
6           Move the actual Location to the current Position
7           New State = State of the New Solution
8           If the New State is better that the Best State
9              Best Solution & Best State = New Solution & New State
10    Return Best Solution
```

The only difference between these two improvement heuristics is that one improves a single route while the other one upgrades the whole solution. The first procedure might seem useless because it is included inside the second procedure. Despite that, the utility of the first one is still highly important because it is used before the initial solution is generated within the constructive heuristic, unlike the second, which is implemented after the initial result has been created.

### 4.2.2. Insertion heuristics

Once the initial solution has been improved, it's time to search for new solutions using the remaining locations of the request bank. This bank is provided as part of the *Initial Solution* generated by the constructive heuristic inside the *Global parameters*. The procedures that modify a solution by adding elements to it are known as insertion heuristics. In order to consider a solution as the *Final Solution,* all the locations must be placed into a feasible route creating a correct solution. As a consequence, the request bank needs to be empty, which means that there are no remaining locations to visit.

An insertion heuristic procedure has been developed to satisfy this need. It is named *Reduce Request Bank,* and it gets as input the solution previously improved. The objective of this function is to place every remaining location of the request bank into the route that would increase less its distance due to the insertion. The procedure returns a *Solution* with its empty *Request Bank*; however, the result might present delays, so the *Solution* could not be considered as the *Final Solution.*

**INSERTION HEURISTIC**

```
1  Function ReduceRequestBank (Solution):
2    For each Location in the Request Bank
3      For each Route in the Solution
4        For Position in the Route
5          Insert the Location into the Position
6          Save the distance increase compared to the distance before the insertion
7        Save the shortest distance increase of the Route and its Position
8      Select the shortest distance increase of the Solution and its Position
9      Update the Solution inserting the Location into the Position selected
10   Return Solution
```

If the *Solution* obtained has delays, the first step that needs to be executed to solve this problem is to use the previous improvement heuristic known as *Improve Solution.* After this procedure has been applied, if the *Solution* is still presenting delays, it is time to use a removal heuristic to make the *Solution* feasible again.

### 4.2.3. Removal heuristics

This kind of heuristic aims to extract parts of the solution that are believed not to work correctly. It consists of modifying the result by removing out locations of a *Solution,* provides as an input. A removal procedure has been developed, as its name indicates (*Remove Delays*), its target is to extract all the delays of the *Solution* and requests responsible for the highest increase on the *Distance.* All the locations removed are put back in the *Request Bank,* generating a solution without latenesses. If the input provided to this function presents delays, the *Solution* generated by this procedure will never be considered as the *Final Solution* due to the missing *Locations*.

**REMOVAL HEURISTIC**

```
1  Function RemoveDelays (Solution):
2      For each Route in the Solution
3          Remove to the Request Bank the request responsible for the highest increase on the Distance
4          If the Route presents delays
5              For each Location that is served late
6                  Remove the Location and place it in the Request Bank
7      Update the Solution considering the extractions
8      Return Solution
```

After executing this removal heuristic, the output is reviewed by the *Improve Solution* procedure to search for a better result. Finally, as a way to achieve the best feasible solution and consider it as the *Final Solution,* the insertion heuristic is applied until its output does not show any tardiness.

## 4.3. Metaheuristics

According to Gendreau and Potvin[13], a metaheuristic procedure is a type of heuristic that is high level designed to find, generate, or select a heuristic that may provide a sufficiently right solution to an optimization problem.

Compared to optimization algorithms and iterative methods, metaheuristics do not guarantee that a globally optimal solution will be found in some kinds of problems. Many metaheuristics implement some form of stochastic optimization so that the solution found is dependent on the set of random variables generated. In combinatorial optimization, by searching over a broad set of feasible solutions, metaheuristics can often find reasonable solutions with less computational effort than optimization algorithms, iterative methods, or simple heuristics.

[13] Gendreau and Potvin: Metaheuristics in Combinatorial Optimization

Heuristic algorithms face many problems, though one of the most significant is to avoid that solutions end in a local optimum. A local optimum of an optimization problem is a solution that is optimal within a neighbouring set of candidate solutions. This is in contrast to a global optimum, which is the optimal solution among all possible solutions, not just those in a particular neighbourhood of values. When a solution ends in a local optimum, most of the local search heuristics become useless and unable to get the solution out of there. Preventing being stuck in a local optimum is one of the main reasons why metaheuristic procedures are applied in algorithms.

Two different categories of metaheuristics were defined by many authors like Bräysy, Hasle, & Løkketangen[14]. Both of them have been implemented in this thesis: single-solution metaheuristics, where only one solution is considered at a time; and population metaheuristics, where several solutions evolve concurrently.

### 4.3.1. Tabu Search

The single-solution metaheuristic implemented in the algorithm design is a Tabu Search. It is a local search-based metaheuristic where, at each iteration, the solution with the shortest distance in the neighbourhood of the current result is selected as the new current solution, even if it leads to an increase of delays in the solution. Through this mechanism, the method can thus escape from bad local optimums. A short-term memory, known as the tabu list (that gave its name to the method), stores recently visited solutions to avoid short-term cycling. The search stops after a fixed number of iterations or after several consecutive iterations have been performed without any improvement to the best-known solution.

The Tabu Search has been applied as an upgrade for the previous local search heuristics. New functions have been designed based on the improvement procedures, and they have been named as *Tabu Improve Route* and *Tabu Improve Solution*. This update supposes a new level of neighbourhood search, allowing to explore solutions that would have been considered infeasible otherwise. By keeping a register of the last valid solution, the procedure can search for neighbours around wrong results, and in case that any of them turn out to be feasible and better than the one that has been kept, it becomes the best solution.

One of the main disadvantages of this metaheuristic procedure is the amount of time required. When the range of the neighbourhoods is expanded, thousands of new solutions need to be analysed, increasing the processing time. In order to keep it under control, only a new level of neighbours has been added. Consequently, the Tabu Search is not as effective as it could be, but it still been satisfactory useful.

[14] Bräysy, Hasle, and Løkketangen: Metaheuristics for the Vehicle Routing Problem and Its Extensions

### 4.3.2. Multiboot

The other type of metaheuristic works with multiple solutions and it is known as population metaheuristic. The procedure implemented is a multiboot heuristic that generates several initial solutions using the constructive heuristic previously mentioned. The function has been named *Multiboot,* and it takes advantage of the flexibility the greedy heuristic provides. The multiboot procedure creates numerous *Initial Solutions* by modifying the *Area Radius*. The range of solutions varies from the ones that have the *Time* as a priority to those which present the *Distance*.

When an *Initial Solution* is generated, a local search starts around its neighbours, aiming the best result as possible until it finds an optimum. However, this optimum could be local and not be the best solution to the problem. Using the *Multiboot* metaheuristic method, the probabilities of having a solution stuck in a local optimum are reduced.

In *Figure 4*, a hypothetical representation of the utility of a multiboot method can be appreciated. As in the designed algorithm, the goal is to minimise the distance of the solution. On the vertical axis, the *Global Distance* of the solutions is plotted, whereas, on the horizontal axis, the range of all the feasible solutions is represented. In this particular case, the *Initial Solution* 3 is the one that shows the minimum initial *Global Distance*. Despite that, after the optimization process, it ends in a local optimum, providing a worse *Final Solution* than the *Initial Solutions* 1 and 2, even presenting a higher distance in their initial results.

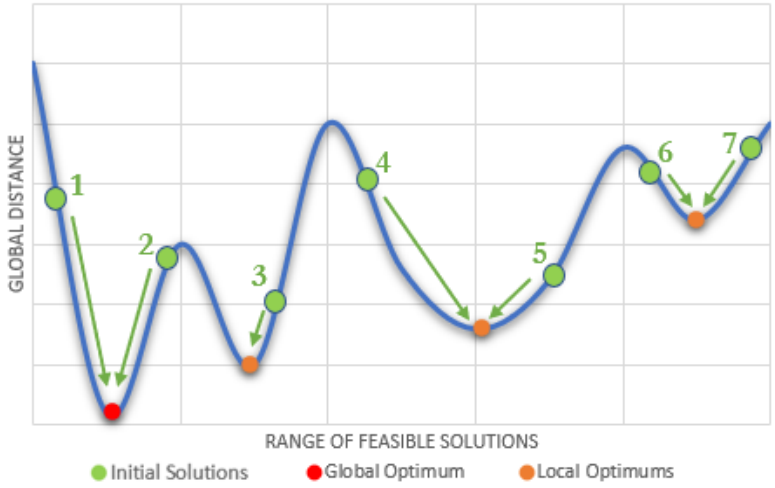


*Figure 4: Hypothetical graphical representation of the utility of a multiboot method*

As discussed in the other metaheuristic procedure, a considerable handicap needs to be faced, and its once again the processing time. The more *Initial Solutions* used, the more probabilities will have the algorithm to avoid local optimums. However, the time required to analyse these solutions until the end increases with every new solution. For this reason, the number of *Initial Solutions* has been limited in the proposed algorithm.

# 5. Computational experiments

This section provides the computational experiments made to test the performance of the heuristics. The algorithm has been developed using the programming language Python, and it has been created in the interface known as Google Collaboratory.

The main idea was to tackle the algorithm with real data and experiment on how close it is from being a real case problem-solver. Due to the inexistence of previous real data and the situation the world is living at the time this project is being developed, because of the actual crisis, it has been impossible to obtain real data.

## 5.1. Li & Lim Benchmark

In order to seek available benchmarks, one set of test instances proposed by Li & Lim[6] has been spotted to be quite close to the model considered in this thesis. The benchmark was generated based on Solomon's Benchmark[15] for the VRPTW. It presents several instances with different features as the organization of the locations (clustered or spread), the number of customers to visit (from 100 to 1000 customers), the width of the time windows (short or long), etc. The data and the best-known results can be downloaded from SINTEF[1] web page.

By using this set of instances, some constraints must be taken into account. The first one consists of a single warehouse, located in the geographical centre of all the customers. Another consideration comprises that all the requests must be served, a solution with unvisited locations will never be a final solution, it will be taken as an unfeasible result. Moreover, there are no special requests, all vehicles are assumed to be able to serve all the customers.

For the purpose of testing the algorithm, some instances have been selected. They present different characteristics like data with geographical diversity (clustered and spread) and also data with various time windows widths. The groups of data used to check the algorithm are formed by the instances classified as LC100 and LC200.

These groups of instances are based on two different distribution models that are being used by logistic companies nowadays. The ones known as LC100 are related to an intercity distribution model where a company must serve several cities in a short period. In this model, each cluster represents a city and usually, all the locations of this city are served by the same vehicle. As a result, the solution to this kind of distribution problem includes a higher number of vehicles. However,

[15] Marius M Solomon: Algorithms for the vehicle routing and scheduling problems with time window constraints.

in the other group of instances named LC200, an urban distribution model is simulated. In these instances, the locations are spread, representing multiple customers inside a single city. As in real life, customers can decide any working hours to get served, but in this case, it is the logistics company, the one that fixes the width of time windows. By creating a long-time window, the solution is usually formed by fewer vehicles than in short time windows.

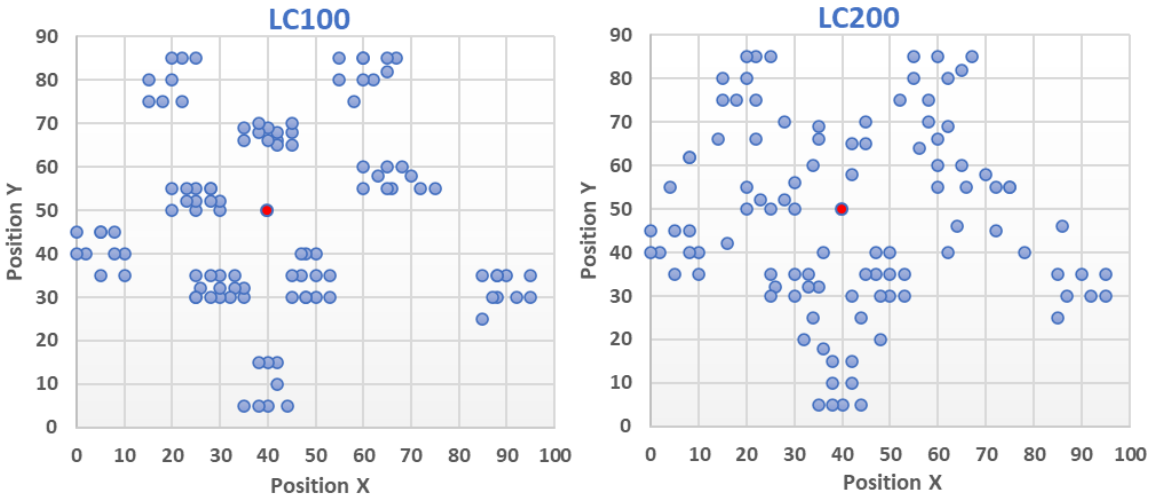


*Figure 5: Geographical distributions of the locations of the customers in blue and the warehouse in red.*

The instances used to validate the algorithm have been briefly modified from the originals. None of the changes applied to the data affect the results or procedures. They have been implemented only to make it easier to program the algorithm. The main alteration has been implemented to the data columns that indicate the destination or the providence of the load. Both columns have been merged into a single one in order to reduce the dimension of the data. Furthermore, the data has been transposed to facilitate its reading for the program.

The objective of the Li & Lim model is hierarchical. Its primary objective is to minimize the number of vehicles used, while its secondary goal is to minimise the travel distance given several vehicles found by the former objective. As a consequence, a solution with a higher number of vehicles is always considered worse than those using fewer vehicles, regardless of how much the total distance might be reduced. This implies that the number of vehicles prevails upon the total travelled distance.

However, by minimizing first, the number of vehicles rather than the travel distance, the algorithm sometimes can lead to impractical solutions in situations where using more vehicles could have been more beneficial. So, in this project, it has been considered more practical to switch the hierarchical order, and the designed metaheuristic algorithm has as a primary objective to minimize the travel distance and then minimize the number of vehicles used.

## 5.2. Numerical results

In this section, the numerical results generated by the designed algorithm are presented in *Figure 6*. They are compared with the best-known results provided by SINTEF[1], where many researchers published their results. In the first column of the table, the instance references are placed with their original name. In the next columns, the minimum number of vehicles and the minim distance are shown of the Li & Lim Benchmark and the design algorithm, respectively. Finally, in the last column, the distance percentage differences between both results are presented.

| | Li & Lim Benchmark | | Designed Algorithm | | |
|---|---|---|---|---|---|
| **Instance** | **Vehicles** | **Distance** | **Vehicles** | **Distance** | **Difference** |
| **LC101** | **10** | **828.94** | **10** | **828.94** | **0.00%** |
| **LC102** | **10** | **828.94** | **10** | **828.94** | **0.00%** |
| **LC103** | **9** | **1035.35** | **10*** | **837.72*** | **-23.59%** |
| **LC104** | **9** | **860.01** | **10*** | **823.37*** | **-4.45%** |
| **LC105** | **10** | **828.94** | **10** | **828.94** | **0.00%** |
| **LC106** | **10** | **828.94** | **10** | **828.94** | **0.00%** |
| **LC107** | **10** | **828.94** | **10** | **828.94** | **0.00%** |
| **LC108** | **10** | **826.44** | **10** | **826.44** | **0.00%** |
| **LC109** | **9** | **1000.60** | **10*** | **827.88*** | **-20.86%** |
| **LC201** | **3** | **591.56** | **3** | **591.56** | **0.00%** |
| **LC202** | **3** | **591.56** | **3** | **591.56** | **0.00%** |
| **LC203** | **3** | **591.17** | **3** | **591.17** | **0.00%** |
| **LC204** | **3** | **590.60** | **3** | **590.60** | **0.00%** |
| **LC205** | **3** | **588.88** | **3** | **588.88** | **0.00%** |
| **LC206** | **3** | **588.49** | **3** | **588.49** | **0.00%** |
| **LC207** | **3** | **588.29** | **3** | **588.29** | **0.00%** |
| **LC208** | **3** | **588.32** | **3** | **588.32** | **0.00%** |

*Figure 6: Comparison between the results obtained by the Li & Lim Benchmark and the designed algorithm*

Most of the results generated by the algorithm coincide with the ones provided by Li & Lim. They present the same number of vehicles and distance, making each of them, the possible optimal solution for its instance. Despite that, there are few marked solutions (*) that are significantly different. As it has been mentioned formerly, the objective function of the algorithm has been altered from the one proposed by Li & Lim. The main priority of this algorithm is to achieve the minimum distance rather than the minimum number of vehicles. Due to this modification, as it can be appreciated in *Figure 6,* the results of instances LC103, LC104 and LC109 are notably varied

from the originals presenting a solution composed by 10 vehicles instead of 9 vehicles, but with a lower distance. Thanks to this change, the algorithm can sometimes achieve more practical results. This has been the case of instances LC103 and LC109, where the difference between the original distance and the one generated differ 23,59% and 20,86%, respectively. Such significant decreases have to be taken into account to decide if it is worth it or not to use fewer vehicles.

However, the difference is not always relevant enough to make a clear decision. In the instance LC104, the solution generated by the proposed algorithm presents a higher number of vehicles, but only a reduction of 4,45% on the distance. This difference could be thousands of kilometres or just a few of them. Using more vehicles to reduce the distance depends on the used unit's dimension, but unfortunately, these instances don't present units.

The Li & Lim instances are dimensionless, which means that the time and the distances considered here don't have units such as minutes, hours, kilometres, etc. This lack of units makes it impossible to associate an operating cost with the result and parallelly complicates the selection of the most practical solution.

The designed algorithm seems to generate high-quality results that match with the ones provided by SINTEF[1]. If any reader decides to use it, the algorithm, the instances, and the detailed results have been published in the following link: *https://drive.google.com/drive/folders/1G6FjBXBJigyGfOs-VVJD-ik2BbQXb3QC?usp=sharing*. Inside the previous link, it can be found two folders named *Instances LC100 & LC200* and *Results LC100 & LC200,* the main file with the name *Metaheuristic Algorithm,* and a *Readme* file. To execute the algorithm, the main file must be open with the application known as Google Collaboratory, and the user needs to follow the instructions posted in the application. If any problem appears, a *Readme* file has also been uploaded to help any user.

The results that are shown in the folders are slightly different from the one provides by SINTEF[1]. This variation is centesimal, and it is caused by the double-precision used to calculate the results, which are rounded to two decimals. However, even having meaningless changes, the vehicle routes of the solutions are the same as the ones from the website.

# 6. Conclusions

This section concludes the main goals of this thesis into three main parts: the understanding of the Pickup & Delivery Problem with Time Windows (PDPTW), the design of a metaheuristic algorithm to solve the problem, and the experimentation of the proposed algorithm to test its efficacy.

The idea behind the first part of this project is to research the NP-Hard problem known as Vehicle Route Problem (VRP). As it has been explained in sections 2 and 3, there are many variants of the original problem and even several objective functions in a single version. However, most of these variations seem to show common roots, making it possible to treat them as the initial problem with additional constraints. Another primary aspect of these sections is to become familiar with the mathematical model of the PDPTW and its notation. After acquiring enough comprehension of both of them, a deep understanding of the real handicaps that a problem-solver method needs to face has been achieved. This priceless knowledge provided by the mathematical model has been used to develop the algorithm proposed in this thesis. A conclusion of this first part would be that successful research has been carried out to gain useful information for the following part of the project.

This second part is based on the design of the metaheuristic algorithm to solve the PDPTW. To develop it, several types of heuristics have been used as constructive, improvement, insertion, removal, tabu search, and multiboot. The flexible constructive heuristic provides to the algorithm the feature to alter its criterion priority almost effortlessly. This flexibility is also used by the multiboot procedure to create different initial solutions to evade local optimums. Another metaheuristic added to the algorithm to avoid local optimums is the Tabu Search, which allows unfeasible results to get out of them. Local search heuristics like improvement, insertion, and removal are part of the algorithm too. Its main goal is to modify the solution in order to generate a better result or to become it feasible. The combination of all of these heuristics procedures results in a practical algorithm able to provide feasible solutions using quick and straightforward heuristics succeeding in the main objective of section 4.

The last part of the thesis tackles the validation of the proposed algorithm. Its main goal is to check if the solutions generated by the algorithm are good enough to consider it adequate to accomplish the established aims. The Li & Lim Benchmark has been used to test it, as a result, the efficacy of the designed algorithm has been demonstrated to be real, as it has been proved in section 5. Furthermore, the solutions generated are feasible and can be treated as possible optimal results. Despite a few differences in some instances due to the alteration in the objective function, the results obtained are as good as the ones provided by SINTEF[1], at least in the cases studied.

To sum up, the main conclusion of this thesis is that the designed metaheuristic algorithm has succeeded in solving the PDPTW, proving that well planned and simple heuristic procedures are a real solution to NP-Hard problems. Of course, the algorithm presents many aspects in which it could be improved. However, the instances tested in this project have been solved not only with valid results but with high-quality too.

# Acknowledgements

The research presented in this thesis has been carried out at the Urban Transportation Engineering & Planification Laboratory, Department of Civil and Environmental Engineering at Nagaoka University of Technology. The opportunity to research in Japan has been possible to an international learning agreement between the Nagaoka University of Technology and my current university, which is known as the Polytechnic University of Catalonia. I would like to thank both universities for providing the necessary facilities to conduct the research.

I sincerely thank my advisor, Dr. Kazushi Sano, for the valuable advice, facilities, and support that he has given me. Also, I would like to thank the other members of my dissertation committee, Associate Professor Kiichiro Hatoyama and Associate Professor Yoko Matsuda, for their helpful comments and suggestions during the weekly seminars.

A pleasant, friendly atmosphere has helped me to feel motivated and comfortable during my period at the Urban Transportation Engineering & Planification Laboratory; it has been thanks to all the laboratory team.

Finally, I would like to thank my whole family, friends, and above all, to Judith, for their unconditional support and encouragement during all these past years. I couldn't have made it this far without you.

# 謝辞

この論文で発表された研究は、長岡技術科学大学大学院　環境社会基盤専攻　都市交通研究室で行われました。日本で研究する機会は、長岡技大と私が現在在籍しているカタルーニャ工科大学の間の国際的な学術協定が可能にしました。ここに私の研究に必要な機会を与えてくれた両大学に感謝致します。

私の指導教員である佐野可寸志教授には貴重なアドバイス、設備、研究サポートをしていただき、大変感謝しております。また、毎週のゼミにおいて、鳩山紀一郎先生、松田曜子先生には、有意義なコメントと助言をしていただいたことに大変感謝しております。

快適でフレンドリーな雰囲気は、私が都市交通工学・計画研究所での私の期間中にやる気と快適さを感じるのに役立ちました。それはすべての研究所チームに感謝しています。

最後にこれまで惜しみないサポートをしてくれた家族、友達、そしてジュディスに感謝します。あなたがいなければ私はここまで成し遂げることは出来ませんでした。

# Bibliography and references


[1] SINTEF: Website - *https://www.sintef.no/projectweb/top/pdptw/li-lim-benchmark/*

[2] Ropke & Pisinger: *An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows.*

[3] Dumas, Desrosiers and Soumis: *The pickup and delivery problem with time windows.*

[4] Narny and Barnes: *Solving the pickup and delivery problem with time windows using reactive tabu search.*

[5] Bent and Van Hentenryck: *A two-stage hybrid algorithm for pickup and delivery vehicle route problems with time windows.*

[6] Li and Lim: *A metaheuristic for the pickup and delivery problem with time windows.*

[7] Hashimoto and Yagiura: *A path relinking approach with an adaptive mechanism to control parameters for the vehicle routing problem with time windows.*

[8] Liana Van Der Hagen: *The Pickup and Delivery Problem with Time Windows: An Adaptive Large Neighborhood Search heuristic.*

[9] Nagata & Kobayashi: *Guided Ejection Search for the Pickup and Delivery Problem with Time Windows.*

[10] Anh Minh Nguyen: *A Hybrid Adaptive Large Neighborhood Search for Pickup and Delivery Problem with Time Windows.*

[11] Laporte and Semet: *A Guide to Vehicle Routing Heuristics.*

[12] Paul E. Black: *Dictionary of Algorithms and Data Structures.*

[13] Gendreau and Potvin: *Metaheuristics in Combinatorial Optimization*

[14] Bräysy, Hasle, and Løkketangen: *Metaheuristics for the Vehicle Routing Problem and Its Extensions.*

[15] Marius Solomon: *Algorithms for the vehicle routing and scheduling problems with time window constraints.*