Routing algorithms for optimisation of transportation systems

Yannis Ancele

A thesis submitted in partial fulfilment of the requirements of Liverpool John Moores University for the degree of Doctor of Philosophy

December 2019

Declaration

The work presented in this thesis was carried out at the Liverpool Logistics, Offshore and Marine Research Institute, Liverpool John Moores University. Unless otherwise stated, it is the original work of the author.
While registered as a candidate for the degree of Doctor of Philosophy, for which submission is now made, the author has not been registered as a candidate for any other award. This thesis has not been submitted in whole, or in part, for any other degree.

Yannis Ancele
Liverpool Logistics, Offshore and Marine Research Institute
Faculty of Engineering and Technology
Liverpool John Moores University
Byrom Street Campus
Liverpool
L3 3AF
UK

March 2020

Abstract

As efficiency and sustainability become more important, researchers are working on new concepts to improve the way logistics are handled in the current Supply Chain Management (SCM).

One concept gaining popularity is the delivery of products in urban areas using bicycles. More companies started using bicycles as an alternative transportation mode and face challenges to efficiently satisfy their customers and employees needs. Large cities with uphill roads require transportation systems to take into account the energy needed by cyclists to move. The load carried on push-bikes has to be kept under a certain threshold for cyclists to be able to pedal on ascending roads. Therefore, cyclists have to choose their route differently to vehicle drivers, hence the need for optimised routing.

Another concept gaining popularity is the collaboration in logistics between several companies. Collaboration is thought to be an enabler for a sustainable and efficient SCM. One important improvement of modern logistics is the frequent exchange of containers via multiple cross-docks which requires spatial and time synchronisation between different types of vehicles. Since each logistics network has its specificities and requirements, new solutions such as the Physical Internet (PI) arise with standardised PI-Containers and protocols to face this challenge. By connecting several transportation networks through collaboration, the PI is expected to considerably improve the way logistics are handled in the current SCM.

Connecting several distribution networks will produce continually varying network conditions arising from traffic growth. Some regions of the network would suffer from bottlenecks that could lead to an increase in transportation costs. One reason is that traditional routing protocols in logistics do not learn from their previous experiences of network problems such as congestion. Therefore, an intelligent network traffic control method is essential to avoid this problem.

As routing problems are at centre stage in transportation sciences, they represent a critical research scope to study in order to improve logistics. As a consequence, in this thesis, three research directions involving routing problems are identified and solved to provide more flexible and extended models to the aforementioned research gaps.

First, a new problem is introduced to tackle constraints arising for bicycle deliveries. A novel Mixed-Integer Linear Programming (MILP) model and an Evolutionary Local Search algorithm are developed to efficiently solve the problem. Experimental results show the accuracy and stability of the proposed algorithm compared to the CPLEX solver from IBM on a wide range of generated instances. A real-world scenario is also studied to demonstrate the relevance of this method.

Second, this research focuses on the way the VRP can be solved while considering an important number of attributes for real-life applications. A rich vehicle routing problem with pickup and delivery including several attributes for the PI is studied. A mathematical formulation is proposed and implemented in CPLEX to solve the problem. The model is then extended to handle multi-objective, uncertainty and dynamism. Multi-threaded meta-heuristics based on Simulated Annealing and Genetic Algorithm are developed with a set of new operators to handle the problem specificities. Computational results on a generated data-set showed that the proposed meta-heuristics are superior to CPLEX in terms of solvability and computational time. A classical benchmark on pickup and delivery problems was also used to validate the proposed method against state-of-the-art methods. The algorithms are integrated into a framework and used to provide solutions for a local company.

Third, the paradigm between the digital and physical internet is explored to propose a new routing approach based on packet routing. While deep learning has been demonstrated to be a promising method for solving numerous optimisation problems efficiently, its application on packet routing is relatively new and rare. This research provides a proof of concept and sheds light on new opportunities to design efficient routing protocols for logistics with deep learning. Simulation results demonstrate that the proposed method is able to not only learn the shortest path for deliveries but also to take into account the truck fulfilment rates to improve global efficiency.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my supervisors Dr. Trung Thanh Nguyen and Dr. Dante Ben Matellini for the continuous support of my Ph.D. study and related research, for their patience, motivation, and immense knowledge. Besides my supervisors, I would like to thank Dr. Minh Hoàng Hà for his insightful comments and encouragement. Last but not the least, I would like to thank my girlfriend, family and friends for supporting me spiritually throughout writing this thesis.

This work was supported by an LJMU PhD Scholarship, a NRCP grant no. NRCP1617-6-125 delivered by the Royal Academy of Engineering, and an RSSB project no COF-INP-05.

YANNIS ANCELE　　MARCH 2020

Declaration of Authorship

I, Yannis Ancele, declare that this thesis titled, 'Routing algorithms for optimisation of transportation systems' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all the main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ______________________________

Date: ______________________________

"There have always been ideas worth fighting for."

People's History Museum

Contents

List of Figures

List of Tables

Abbreviations

A3C	**A**synchronous **A**dvantage **A**ctor-**C**ritic
ANN	**A**rtificial **N**eural **N**etwork
BRP	**B**ike **R**outing **P**roblem
CL	**C**ity **L**ogistics
DI	**D**igital **I**nternet
DNN	**D**eep **N**eural **N**etwork
DRL	**D**eep **R**einforcement **L**earning
EA	**E**volutionary **A**lgorithm
ELS	**E**volutionary **L**ocal **S**earch
GA	**G**enetic **A**lgorithm
IoT	**I**nternet **o**f **T**hings
MILP	**M**ixed-**I**nteger **L**inear **P**rogramming
ML	**M**achine **L**earning
OPL	**O**ptimisation **P**rogramming **L**anguage
OSPF	**O**pen **S**hortest **P**ath **F**irst
PDP	**P**ickup and **D**elivery **P**roblem
PI	**P**hysical **I**nternet
RL	**R**einforcement **L**earning
SA	**S**imulated **A**nnealing
SCM	**S**upply **C**hain **M**anagement
TSAM	multi-**T**hreaded **S**imulated **A**nnealing with **M**emory
VCRP	**V**ehicle and **C**ontainer **R**outing **P**roblem
VRP	**V**ehicle **R**outing **P**roblem

To whoever will read me . . .

Chapter 1

Introduction

Routing refers to the selection of paths for traffic in one or several connected networks. Routing can be used in a broad range of problems dealing with the transfer of objects from sources to destinations. As companies seek efficiency and sustainability, optimising the Supply Chain Management (SCM) becomes challenging. During the last decades, many researchers have been developing optimisation and approximation algorithms for routing problems. The Vehicle Routing Problem (VRP), introduced by Dantzig and Ramser (1959), is a combinatorial optimisation problem that was proposed in the late 1950s and it is still one of the most studied problems in the field of operations research. The great interest in the VRP is due to its practical importance, as well as the difficulty of solving it. The objective of the classical VRP is to satisfy a set of customers with known demands within a time horizon by providing minimum-cost routes for vehicles to schedule deliveries originating and terminating at the same depot. The Pickup and Delivery Problem (PDP) is a generalisation of the VRP which is about finding optimal routes to satisfy transportation requests. Each request requires both pickup and delivery with precedence constraints. As studies are carried out, emerging methodologies are being proven to be more suitable for an efficient and sustainable SCM. As an example, the VRP with cross-dock is a variant of the classical VRP which contains spatial and load synchronisation constraints. These cross-dock facilities allow products to be transferred and processed. A cross-dock can be considered as a consolidation facility which has short-term storage. The principle of such a system is to unload and sort incoming containers, then to load the outgoing ones on vehicles. This approach differs from the direct-shipping of products in which intermediate trans-shipment points are not solicited.

The main purpose of this thesis is to investigate some operational research problems for routing to propose new solution methods to fill the gaps created by emerging methodologies.

1.1 Scope of the thesis

Because the range of routing problems for transportation is wide and diverse, it is impossible to cover all the topics. As a consequence, this thesis will focus on routing of vehicles for goods transportation with cross-docks. The similarities between containers routing and packet routing protocols will be investigated. Therefore, some packet routing protocols will be studied to be adapted to the physical world. In order to provide solutions, a range of exact solvers was compared before CPLEX was eventually selected. Meta-heuristic algorithms will be designed to handle large instances by providing near-optimal solutions. As a result, in a supply chain management, this thesis covers the logistics aspect which defines the flow of goods between the point of origin and the point of consumption to satisfy customers.

1.2 Research questions

The approach adopted in this thesis is to ask a general question to get an overview of the important gaps in the field of transportation systems:

- *What are the current methodologies for vehicle routing in emerging transportation problems?*

This question allows a review of the state-of-the-art vehicle routing methods for the aforementioned emerging areas, which then brings up several questions as follows:

- *Are there any gaps related to emerging concepts?*
- *How can new models be designed to fill these gaps while also considering sustainability?*
- *How can they be applied to the Physical Internet?*
- *How can these new problems' specificities be effectively handled and solved?*
- *Can methodologies from other domains be used and applied to them?*

This set of questions are used as a building block to direct the research throughout the thesis.

1.3 Contribution of the thesis

Contributions in this thesis can be classified into three main groups:

In the first group, the following contribution is made: (1) A new problem of practical importance is introduced. (2) An MILP model is designed with a linearised constraint to use the CPLEX solver. (3) An Evolutionary Local Search algorithm including split and local search procedures that can check a solution's feasibility in $\mathcal{O}(1)$.

The second group includes the following: (1) A new model gathering several VRP attributes which are most important as logistics evolves. The model handles multiple cross-docks in a new and more flexible way while the other attributes have been gathered and combined from the literature. (2) An algorithm with new operators which include generic features as well as problem specificities. (3) An adapted version of the previous model is provided to match the needs of a company. (4) The functional framework is upgraded with realistic constraints including dynamism, multi-objective with a new lexicographic method and node prediction feature. (5) A new Evolutionary Algorithm (EA) is developed with a new crossover tailored to this problem.

The third group covers the following: (1) A preliminary work aiming at providing a proof of concept of how machine learning can be beneficial for a vehicle routing problem. (2) This research also reveals new perspectives that can be considered using state-of-the-art learning methods to solve the problem. (3) Details of further investigations are provided to continue in that direction.

1.4 Outline of the thesis

This thesis aims to answer the research questions presented above. It is organised as follows:

Chapter 2 reviews existing research related to the proposed approaches in this thesis.

In Chapter 3, a Bike Routing Problem (BRP) with a mathematical model is presented. An evolutionary algorithm is designed to handle large instances. Computational results and analyses of the algorithm are also provided to demonstrate its efficiency.

Chapter 4, presents a rich VRP with several attributes. A mathematical formulation of the problem is given with a meta-heuristic algorithm. A comparative analysis of the meta-heuristic and CPLEX results is shown. The efficiency of the algorithm on a classical benchmark is also discussed.

In Chapter 5, an extension of the work from Chapter 4 is presented. A mathematical formulation of the model including multi-objective, dynamism and uncertainty is proposed. A new and simpler meta-heuristic for the extended problem is given. Computational results and analyses of the algorithm compared to the previous one is discussed. The performances of a predictive function are also presented.

In Chapter 6, a machine learning method is presented for the container routing problem. A state-of-the-art algorithm is used to efficiently solve the problem. While the relevance of this method is discussed, suggestions for a future extension are detailed.

Finally, Chapter 7 concludes the work in the thesis. Contributions of the research are summarised and future research directions are also suggested.

Chapter 2

Literature review

2.1 Delivery using bikes

To tackle the pollution problem, many researchers are trying to solve the green vehicle routing problem (Lin et al., 2014; Erdoğan and Miller-Hooks, 2012). Green Logistics is a concept aiming at delivering efficient supply chain management while considering the environment. The traditional objective of distribution management was upgraded to minimising system-wide costs related to economic and environmental issues. In urban areas, trucks, on top of the pollution they cause, have difficulty achieving door-to-door deliveries. In densely populated urban areas, bike-couriers or bike-messengers offer delivery advantages. Although bikes are seen as a good start towards green transportation (Tipagornwong and Figliozzi, 2014), very few papers in the literature deal with the delivery of goods using push-bikes with cargo. Instead, when it comes to optimisation of models for bikes, researchers mainly focused on e-bikes, the itinerary for cycling or the bike-sharing re-balancing problem. Bike-sharing systems offer a mobility service whereby public bicycles, located at different stations across an urban area, are available for shared use. These systems contribute towards obtaining more sustainable mobility and decreasing traffic and pollution caused by car transportation (Dell'Amico et al., 2014).

An important number of papers dealt with the route choice model for cyclists. For example, Ehrgott et al. (2012) studied such a problem by considering a bi-objective variant. It is said and acknowledged that cyclists choose their route differently to drivers of private vehicles. They optimised routes for bikes by calculating a suitability score considering safety and comfort such as motor traffic volume, motor traffic speed, road lane width, presence of on-street parking, road gradient, percentage of heavy commercial vehicles, presence of cycle facilities (cycle lanes or shared bus/cycle lanes), pavement condition, etc. This score is then evaluated and considered alongside the route distance. Hrncir et al. (2015) and Song et al. (2014) also considered the bicycle routing problem with road characteristics. It is also argued that in contrast to car drivers, cyclists consider a significantly broader range of factors while deciding on their routes. They emphasise on the importance of slope, turn frequency, junction control, noise, pollution, scenery, and traffic volumes in addition to travel time and distance for cyclists. The paper from Silbernagl et al. (2016) is another example of cycling routing where elevation is considered. However, just like the previously mentioned papers, their work does not consider delivery purposes and therefore does not compute the energy required.

The problem of delivering goods using bikes was studied by Ghiani et al. (2009). They solved a dynamic vehicle dispatching problem with pickups and deliveries. Their model considers the use of bicycles for same-day courier services. Although they studied a dispatching problem instead of VRP, their paper is closer to our problem compared with the aforementioned ones. However, they did not take into account any cycling-related constraints. Another paper dealing with a delivery problem using bicycles is from Serna et al. (2010). Their problem is actually based on the VRP but they did not consider the energy required according to the weight and road slope. The paper from Ćirović et al. (2014) also solved the VRP while considering the environment. They provided solutions for routing of light delivery vehicles in urban areas but their model only accounts for motorcycles and therefore does not consider the road gradient.

Bike-messengers (Maes and Vanelslander, 2012) and bike-couriers (Lee et al., 2019) problems are also well studied in the literature. However, they are mainly using electrical bikes for deliveries and can neglect uphill constraints, and therefore do not consider the energy required according to the load carried. Using push bicycles to carry goods requires a non-negligible amount of energy power. Di Prampero et al. (1979) were certainly among the first ones providing an equation of motion for cyclists. Their equation considers several metrics such as air resistance/temperature, altitude, body size etc. to provide the power required to move the bicycle. Their aim was to predict the speed attained under a given set of conditions, provided the metabolic power of the subject is known. Since this paper, lots of studies were conducted to improve models including power (Barth and Boriboonsomsin, 2009; Ross, 1997) for motor vehicles via a similar formula. Martin et al. (1998) derived a mathematical model of cycling power from the same formula and provided values for the model parameters. A bicycle-mounted power measurement system was validated by comparison with a laboratory ergo-meter. They provided an analysis of the parameter effect by altering their values. Their results demonstrated the accuracy of the formula which emphasises that cycling power can be predicted by a mathematical model. Kara et al. (2007) introduced the so-called Energy-Minimising Vehicle Routing Problem. This problem is an extension of the VRP where a weighted load function (load multiplied by distance), rather than just the distance, is minimised. Bektaş and Laporte (2011) introduced the Pollution-Routing Problem (PRP). The PRP is an extension of the classical VRP with a broader and more comprehensive objective function. On top of the travel distance, few other constraints such as the amount of greenhouse emissions, fuel, travel times and their costs are considered. Moreover, the total load carried is considered and influences the energy required by a vehicle which makes the problem more challenging. The main difference with our work is that we consider bikes with no emission. Therefore, the calculation of the CO_2 is not necessary but the load weight is of greater importance in slopes. Also, Bektaş and Laporte (2011) calculated the CO_2 generated by vehicles in the objective while our method requires the computation of the energy for a constraint. Another difference is that they linearised their model through discretisation of the speed variables while we linearised ours to handle the accumulated load carried by each bike. They concluded that a weighted load-minimising solution can have counter-intuitive results on energy consumption due to a possible increase of distance travelled. Our model, however, only provides distance-minimising solutions where the energy is checked in a constraint. Therefore, such results are not obtained by our proposed approach.

2.2 Physical Internet

Sustainability in the current SCM is receiving growing attention (Montreuil, 2012). As a consequence, new systems have to be built to handle physical objects throughout the world in a manner that is economically, environmentally and socially efficient and sustainable. As logistics organisation is one of the major problems, designing a new logistics organisation using connected systems can represent a step toward sustainable solutions. To respond to this situation, it was suggested that it could be possible to mimic the Digital Internet (DI) into the physical one. The idea is to use the DI as a metaphor for the physical world (Sarraj et al., 2012). The Physical Internet (PI-π) implements this idea by interconnecting logistics systems and networks. A set of world-standard modular containers, interfaces and protocols (Montreuil et al., 2012) are being designed to improve the worldwide efficiency and sustainability of logistics (Montreuil, 2012). The main concept is that logistics have to evolve and stop using specific material transportation means in order to focus on PI-container transportation means instead. The goods carried or products would have to fit and to be consolidated into containers to minimise the waste of space.

Just like encapsulation protocols in the DI, conveyors would only deal with PI-containers without knowing the content. These modularised black-boxes are much easier to be transported through networks and facilities than loads of non-standardised cases and pallets. A PI-container can be made up of smaller PI-containers. On top of that, by transmitting information, these smart objects would minimise their movements. Basically, these PI-containers are routed through Physical Internet facilities called PI-hubs (which are cross-docks adapted to the PI). By using such facilities, a container delivery process will dramatically evolve. The container delivery process will be similar to a parcel one. Each truck will exchange their loads at transit PI-hubs every hundred miles, so that, another truck could rapidly pick up the loads to exchange it further in its itinerary until the destination is reached. PI-hubs have some differences with classical cross-docking hubs as described in Meller et al. (2012) but the concept of trans-shipment stays the same. However, while cross-dock hubs are restricted to specific users like suppliers, retailers, clients, etc., PI-hubs can be used by any contracted users around the world. This is facilitated by the use of only modular and standard PI-containers with fixed sizes.

Just like the PI, City Logistics (CL) is a research area which is focused on the transportation improvement. CL and PI have similarities which allow them to work alongside each other. One of the first works about their compatibility was done in Crainic and Montreuil (2016). In their paper, it is explained that the CL provides the final building blocks for the PI to be complete with regards to transportation in cities. This concept is not limited to trucks but can also include trains, planes, bicycles, cars etc. The Physical Internet has already been applied to a few real cases (Meller et al., 2012). Also, technical papers which specify various PI-object aspects can be found in the literature (Montreuil, 2011). In the end, this concept could also be applied to human mobility (Crainic and Montreuil, 2016).

With emerging concepts such as PI, the implementation of new methodologies becomes possible. As detailed in Meller et al. (2012), there are similarities between PI-hubs and cross-docks. This could allow a better transition and modelling by using current cross-docking methods. By exploiting the similarities with the DI, one could use packet routing protocols to route containers. PI-containers with their standard interface could be encapsulated like in packet switching (Montreuil et al., 2012).

2.3 Practical attributes for the VRP

A fundamental building block to optimise transportation logistics, such as PI, is the PDP. In operational research, it is a central problem for logistics as it gives a solution to the transportation process. To help grasp the relevant models and methodologies, numerous reviews can be found such as the work of Silva and Zuluaga (2016). They not only presented a classification of the different attributes of the problem but also provided insights on modelling and solution techniques. When a combination of multiple constraints is being solved, the problem can be categorised as a rich VRP. Caceres-Cruz et al. (2014) gave a survey on the rich VRP by summarising problem combinations, constraint definitions and different approaches. Lahyani et al. (2015) also worked on the rich VRP by providing a comprehensive and relevant taxonomy alongside its definition. As real-world applications require an increasing number of attributes to be considered simultaneously, researchers tend to design general-purpose solvers. Vidal et al. (2014) developed a unified solution framework to tackle a multi-attribute VRP. They demonstrated that such a general method can be efficient for this class of problem. Another attempt to solve a rich VRP with a unified heuristic can be found in Pisinger and Ropke (2007). A pickup and delivery model was provided with a robust and self-calibrating framework to solve it.

Different classes of problems arise for the PDP, two of them are simultaneous and mixed PDP. In the simultaneous PDP, pickups and deliveries can be made at the same time. Originally, this variant considers a homogeneous fleet of vehicles to satisfy the customer demands. However, nowadays, there are different types of vehicles available to be used. Wang et al. (2015) addressed the VRP in which customers require simultaneous pickup and delivery of goods during specific time windows. They used a parallel Simulated Annealing algorithm to efficiently solve this variant. This problem was also solved in Zachariadis et al. (2010) with an adaptive memory framework that generates high-quality solutions by collecting and combining promising solution features. In the mixed PDP, pickups and deliveries can occur in any order on a vehicle route. Wassan and Nagy (2014) presented a taxonomy of different problem versions including mixed pickup and delivery. They focused on the back-hauling aspect of the PDP while providing a review of solution methodologies and highlighting issues in the literature. Rich solution frameworks like in Vidal et al. (2014) also consider this attribute. However, this variant seems to suffer from a lack of published papers even though it has immense practical applicability within logistics.

A commonly used attribute for the VRP is time windows, as shown in Solomon (1985), it requires that the delivery is made within a specific time window given by the customers. Two categories can be defined: soft time windows which can be violated while inducing a penalty cost and hard time windows which cannot. Cordeau et al. (2000) wrote a survey in which they presented approximation methods and optimal approaches to tackle this variant.

Compatibility or site-dependency constraints can also be added on a VRP model. As shown in Pellegrini et al. (2007), this refers to the situation in which a customer must be served from a specific depot, by a specific vehicle or a specific driver. For instance, goods demanded by a customer might require vehicles with special equipment for loading and unloading. Access restrictions regarding the vehicle type can apply in a given area or city. The work of Desrochers et al. (1990) also covered this attribute.

Companies that have several depots can solve the multi-depot VRP to satisfy all its customers. In this variant, vehicles can be located at different depots while customers can be assigned to different depots. Montoya-Torres et al. (2015) reviewed the state-of-the-art on this variant by considering relevant papers since 1988. They studied several variants and provided a classification for solution approaches dealing with single or multiple objectives. Nagy and Salhi (2005) also studied this variant while avoiding the common assumption that pickups and deliveries must be completed in two different stages.

It has been demonstrated that cross-docking strategy plays an important role in goods distribution. In their PDP and VRP with cross-dock models, Nikolopoulou et al. (2017) gave a comparison of direct-shipping and cross-docking strategies. They developed a local-search optimisation framework and tested on existing and new benchmark data-sets. It was concluded that depending on the environment and constraints, a cross-docking strategy can outperform the direct-shipping. However, in some cases direct-shipping can still be relevant, this could mean that hybrid methods are necessary. As cross-docking strategies become common, scheduling and door assignment are being solved simultaneously with cross-dock constraints. Enderer et al. (2017) proposed two formulations of the problem and compared computational results. They showed that integrated solution approaches can lead to significant savings in costs. Cross-docking might create situations in which vehicles have to wait for empty doors or even products to arrive. Dondo and Cerdá (2014) considered this constraint by introducing a mixed-integer linear programming formulation. They also used an integrated solution approach in which the dock door assignment and the truck scheduling at the cross-dock are simultaneously decided. Miao et al. (2012) studied the multiple cross-docks where penalty values are added when the time windows are not met. They proved the NP-hard difficulty of the multiple cross-docks problems and therefore designed a hybrid method to solve the problem compared with CPLEX. Although the interest for cross-docking is increasing, only very few papers considered multiple cross-docks constraints. The paper of Maknoon and Laporte (2017) is one of them, they proposed a mathematical formulation of the problem in which requests have to pass through at least one cross-dock. The efficiency of their proposed adaptive large neighbourhood search heuristic was demonstrated with a comparison with CPLEX. However, their model using heterogeneous vehicles which must start and end their routes at the same assigned cross-dock is not entirely compatible with PI constraints. Their transportation process is divided into two separate shifts (pickup and delivery) which cannot model a mixed pickup and delivery.

In some situations, once the last customer on the route has been visited, the driver does not have to return to the depot. The driver could terminate his route at another depot or even at home. This problem was introduced by Schrage in 1981 and is called open VRP. A few papers have been devoted to this problem. Alinaghian et al. (2016) dealt not only with cross-docking but also with open VRP. On top of the capacity of vehicles that is not completely used, some companies use rental vehicles due to the high cost of purchasing vehicles with high capacity. Therefore, the authors proposed a cross-docking and open-close VRP problem solved by a simulated annealing algorithm. Russell et al. (2008) showed relevant applications of the open VRP like newspaper logistics. It was explained that the independent outsourcing characteristic of these processes as independent contractors requires this type of modelling. Yu et al. (2016) studied and solved a capacitated homogeneous cross-docking and open VRP with a simulated annealing algorithm. They showed that their proposed method can outperform CPLEX. Atefi et al. (2018) also solved this variant while considering a decoupling points strategy which generalises the open VRP as multiple trucks delivery. The first truck performs part of the deliveries, then drops off the load while the second one and others continue from that point onwards. Their decoupling points strategy is similar to our model handling consolidations in a PI environment. On top of open routes, our model also allows vehicles to start and end their routes to different depots.

Although the above VRP attributes have been studied with different combinations, to the best of our knowledge, none has modelled this problem considering all the above constraints simultaneously. Only a few papers proposed to solve the VRP with a single cross-dock and very few solved this problem with multiple cross-docking facilities. Papers like Maknoon and Laporte (2017), Miao et al. (2012), Chen et al. (2015), Wang et al. (2017) and Ahkamiraad and Wang (2018) dealt with multiple cross-docks but modelled the problem differently compared to this thesis. When considering multiple cross-docks, pickups and deliveries were often handled in a less flexible way. They were satisfied in separate stages/vehicles or could not be consolidated within several successive cross-docks before delivery. However, there are real-world problems with all the aforementioned constraints.

2.4 Dynamic, multi-objective and uncertainty constraints for the VRP

As the interest for connected technologies in logistics increases, real-time systems become even more important nowadays. The most common source of dynamism in VRP is the arrival of customer requests during the operation. In this type of dynamism, some requests are available initially, while customers can place orders at any time in the day. However, dynamism can also be on travel time, service time or even on vehicle availability when the breakdown of vehicles is considered.

Wilson and Colvin (1977) pioneered the work of Dynamic Vehicle Routing Problem (DVRP). They studied a single-vehicle dial-a-ride problem in which customer requests appear dynamically. Since that paper, this research area has gained interest in the literature. Abbatecola et al. (2016) reviewed the state-of-art of DVRP and emphasised the importance of the information and communications technologies for real-time decision making. Psaraftis et al. (2016) proposed a taxonomy of DVRP papers according to several criteria. They identified important gaps in the literature and discussed future research incorporating probabilistic information. In their survey, Pillac et al. (2013) classified VRPs from the perspective of information quality and evolution. They introduced and investigated the degrees of dynamism which represents the frequency at which new data becomes available. They also presented a comprehensive review of applications and solution methods for DVRP while mentioning the lack of reference benchmarks for this class of problems. Ghiani et al. (2009) studied a real-time vehicle dispatching problem in which an anticipatory algorithm is developed. They developed an algorithm which anticipates future demands through a sampling procedure and demonstrated the relevance of such methods. In the applications considered in this thesis, the source of dynamism is on the customers themselves. To solve a DVRP, a possible approach is to change solutions at any point in time, depending on events such as the arrival of new requests or the service of a customer. In this work, however, we opted for a different approach in which changes to the solution may only be made at pre-determined intervals.

In a VRP, when one or several components of the problem are random, the problem is called the Stochastic Vehicle Routing Problem (SVRP). There are three common kinds of SVRP, (1) stochastic customers, (2) stochastic demands and (3) stochastic times. When some data are random, it is no longer possible to guarantee that all constraints will be satisfied for all realisations of the random variables. Therefore, two approaches can be considered, (1) the model includes corrective actions to be taken when a constraint is violated or (2) the decision-maker requires the satisfaction of some constraints with a given probability. In SVRP, two stages are usually made to solve the problem. In the first stage, a solution is determined before knowing the realisations of the random variables. Then, in the second stage, the recourse or corrective action can be applied if necessary when the values of the random variables are known.

The survey of Ritzinger et al. (2016) summarised the recent literature in this area. Besides the traditional classification according to the available stochastic information, they introduced a new classification. Their classification is based on when computational effort for determining decisions or decision policies arises. Furthermore, they analysed the solution quality between approaches which consider only dynamic or stochastic problems with those which consider both. Hence, the strength of the approaches incorporating dynamic and stochastic information was demonstrated. Oyola et al. (2016) Oyola et al. (2017) also reviewed the literature in this area by providing two different types of models: chance-constrained program and stochastic programming with recourse. Albareda-Sambola et al. (2014) introduced the dynamic multi-period VRP with Probabilistic Information. Their work is similar to ours in the sense that at each time period, the set of customers requiring a service in later time periods is unknown, but its probability distribution is available. As usual, requests must be satisfied within a given time window in the time horizon. Moreover, to reduce distribution costs, they proposed an adaptive service policy that aims at estimating the best time period to satisfy each request. Through computational experiments, they showed the effectiveness of their policy method compared with alternative methods. Bruni et al. (2014) studied an innovative concept to tackle the SVRP. A robust demand-responsive transportation (DRT) system where vehicles may deviate from the planned route to accept late requests, is introduced. They proposed a new formulation of the problem as a stochastic mixed-integer program and provided an efficient heuristic procedure that merges different scenario solutions. They also provided computational results which demonstrated the validity of the heuristic and provided useful insights into DRT systems. Their work is also contributing to promoting the incorporation of uncertainty into the planning process. In our work, customer requests are predicted with a probability and added into the model to be solved. As the time horizon passes, dynamic requests are ordered in and replace predicted requests. The uncertainty characteristic of the problem implies that not all requests are correctly predicted, hence the pseudo optimal nature of the solutions created.

Due to the constraints and structure of the VRP, the optimisation of one objective may lead to the deterioration of other important characteristics. Multi-objective optimisation focuses on solving optimisation problems with several objectives simultaneously. Optimisation methods for this type of problem can be divided into four classes that can be differentiated by the decision-maker's involvement. (1) In “no preference” methods, there is no decision-maker, but a trade-off solution is identified without preference information. (2) In “a priori” methods, preferences are asked from the decision-maker in order to provide a solution that best suits its preferences. (3) In "a posteriori" methods, a set of Pareto optimal solutions is first found and provided to the decision-maker which then has to choose a single solution. (4) In "interactive" methods, the decision-maker is involved in the search to direct the process and get the most preferred solution.

Our work considers several objectives for a generalised version of the VRP - the pickup and delivery problem. Many other papers tackled the problem from a multi-objective perspective. García-Nájera and López-Jaimes (2018) studied a similar problem as they solved a PDP with multi-objective. They analysed the problem as the objectives change via different perspectives - problem difficulty, problem properties, objective conflicts and scalability. Four reference algorithms were used for comparison purposes with an EA tested on benchmark instances. They provided comprehensive insights as to how the benchmark instances were generated. Núñez et al. (2014) also developed a predictive control approach to solve the multi-objective PDP. However, they considered a dynamic version of the dial-a-ride service. They considered the practicability of their tool for a dispatcher to make decisions. They demonstrated its potential benefits in terms of the operator cost and quality of service perceived by the users. Finally, Yang et al. (2017) solved a dynamic multi-objective PDP with time windows. On top of that, they considered three-dimensional request predictions to enhance the effectiveness of their method. Similarly to our work, the statistical distribution of historical data is used to make predictions about future requests. The predictive routes are created and tuned subsequently when the dynamic requests appear. Their research is close to ours as they use request prediction but only considered one vehicle and no cross-docks which makes the problem simpler compared to ours. They showed the efficiency of the proposed algorithm by comparing it with two other popular algorithms.

2.5 Machine learning applications for routing problems

Viewing the PI as a DI allows us to use networking protocols as described in Montreuil et al. (2012). The term "ad hoc networking" typically refers to a system of network elements that combine to form a network. In other words, a network is ad hoc if it does not rely on a pre-existing infrastructure, such as routers in wired networks or access points in managed wireless networks. Instead, each node participates in routing by forwarding data for other nodes, so the determination of which nodes forward data is made dynamically based on network connectivity and the routing algorithm in use. A Mobile Ad hoc NETwork (MANET), also known as wireless ad hoc network or ad hoc wireless network, is a continuously self-configuring, infrastructure-less network of mobile devices connected wirelessly (Zanjireh and Larijani, 2015; Toh, 2001). Each device in a

MANET is free to move independently in any direction, and will therefore, change its links to other devices frequently. Each must forward traffic unrelated to its use, and therefore be a router. Such networks may operate by themselves or may be connected to the larger Internet. This results in a highly dynamic, autonomous topology (Zanjireh et al., 2013). In this analogy, PI-hubs are the routers of the DI while links are the vehicle routes. Vehicular Ad hoc NETworks (VANETs) is another concept that could be exploited if we consider different PI-hubs to be used over time. VANETs are created by applying the principles of MANETs where networks can be formed and information can be relayed among cars.

To solve these problems, traditional methods like RIP (Hedrick, 1988) or OSPF (Moy, 1997) can be used. However, with the recent breakthrough in ML, numerous research studies showed that ML could be applied to a broader range of applications. ML is composed of four main areas - supervised, unsupervised, semi-supervised and reinforcement learning. (1) In supervised learning, there's an external "supervisor", which has knowledge of the environment and who shares it with the agent to complete the task. The task is to learn a function that maps an input X to an output Y based on example input-output pairs - the data. The end result is to be able to predict the output variables Y for the given input. Supervised learning can be further grouped into regression and classification problems. (2) Unsupervised learning is where you only have input data X and no corresponding output variables. In unsupervised learning, rather than considering a mapping, the task is to find the underlying patterns in order to learn more about the data. Unsupervised learning can be further grouped into clustering and association problems. (3) Semi-supervised learning is another area that makes use of both labelled and unlabelled data for training. Usually, a small amount of labelled data is used alongside a large amount of unlabelled data. Therefore, semi-supervised learning can be classified between unsupervised learning and supervised learning. (4) Reinforcement Learning (RL) is concerned with how software agents ought to take actions in an environment to maximise a cumulative reward. It is about mapping situations to actions in order to learn what to do. Usually, the agent is not told which action to take but instead must discover which action will yield the maximum reward.

An Artificial Neural Network (ANN) is a computational model based on the structure and functions of biological neural networks. Neural networks are widely used in supervised learning and reinforcement learning problems. The main purpose of a neural network is to receive a set of inputs, perform progressively complex calculations on them, and give output to solve real-world problems like classification. Information that flows through networks affects their structure because of changes during the training phase. It is said that the network learns to produce an output based on the input. These networks are based on a set of layers connected to each other. An ANN can have one or multiple hidden layers in which the weights and biases are modified during the training. When more than one hidden layer is used, the network is called a deep neural network (DNN). Similar to shallow ANNs, DNNs can model complex non-linear relationships. In deep learning, each level learns to transform its input data into a slightly more abstract and composite representation. DNN models can produce better results than normal ML networks. Deep Reinforcement Learning (DRL) is another method which combines the strength of ANN and RL. The goal of DRL is to create artificial agents that can achieve a high level of performance and generality. The performance at scale of this method was demonstrated in Mnih et al. (2015) for the first time. Many papers on DNN and DRL can be found in the literature (Lillicrap et al., 2015; Mnih et al., 2013; Silver et al., 2016).

In the area of ML, RL techniques have already been used in the context of routing optimisation, which was pioneered in Boyan and Littman (1994). Several reviews can be found in the literature. Fadlullah et al. (2017) did a comprehensive survey on the state-of-the-art on deep learning architectures and algorithms. A proof-of-concept was introduced for applying the deep learning technique to perform intelligent traffic control in future networks. They emphasised the challenges due to the continually varying network conditions arising from the tremendous traffic growth. It is argued that deep learning is a viable approach to configure and manage networks more intelligently and autonomously. However, they stated that DNN for routing received little attention compared to other applications. Alsheikh et al. (2014) also recommended deep learning as a more efficient method for the Wireless Sensor Networks (WSN). Readers can also refer to the survey of Mao et al. (2018) on deep learning for intelligent wireless networks. More recently, Wang et al. (2018) promoted the use of deep reinforcement learning for networking problems. They provided a selective survey of the latest representative advances with explanations of their design principles and benefits.

ML techniques can be used for routing in different ways. One of them is delay prediction in which an ML algorithm is plugged into a network to gather information to assist routing decisions. Wang et al. (2007) developed such a method to analyse data and extract useful features and correlations in WSNs for Routing protocols. To address this problem, they used supervised learning techniques to make informed decisions. They not only showed that offline learning was able to improve the data delivery rate over traditional methods but also showed that an online learning algorithm can achieve similar accuracy. Guo et al. (2010) also proposed a method to predict delays with neural networks. Their system is devised as a distributed, independent, and continuous neural network training and prediction process conducted on individual nodes. As in Dudukovich et al. (2017), by integrating the delay prediction mechanism with another protocol, they were able to improve the packet deliveries.

Another way of using ML algorithms is by directly controlling routing decisions. Stampa et al. (2017) proposed such a method in which a DRL approach is used to optimise routing and reduce delays. Similarly, You et al. (2019) introduced an approach in which two multi-agent DRL routing algorithms are integrated. The first one replaces Q-table by a deep neural network, while the second employs information including the past actions and the destinations of packets. Tang et al. (2018) proposed an online algorithm to be integrated into any routing protocol to improve or provide new routing strategies. They used an OSPF to train a DNN based on deep convolutional neural networks. Massively parallel computing is also discussed and argued to be an enabler for DNN to improve network traffic control. Mao et al. (2017) also emphasised the current parallel computing capacities which can enable DNN applications on routing. They explored new opportunities in packet processing to shift from rule-based to DNN based route computation for high-throughput packet processing. Like previous researchers, they also emphasised the lack of DNN based route computation in the literature.

2.6 Conclusion

This chapter reviewed some fields in optimisation algorithms applied to routing.

The literature related to bike usage is mainly about one-way itinerary planning, bike-sharing re-balancing problem or routing delivery. However, a few researchers considered delivery problems using push-bikes but did not include important bike-related constraints such as uphill and energy. Hence the need for filling this gap by providing a new BRP.

A rich VRP can arise in many practical applications and therefore contributes toward a more efficient SCM. The PI is one of such examples which uses several types of vehicle and allows load exchanges at multiple facilities to satisfy customers within time windows. Collaboration using external transportation means is another feature of the PI which must be considered. Vehicles can belong to external companies or even private owners and are therefore available for a given period but are not forced to return to the same depot. One issue cross-docking is tackling is the capacity of vehicles not being entirely used during the deliveries. Having mixed pickups and deliveries is another characteristic of a model that can improve the overall efficiency by introducing some degree of flexibility. The vehicle fulfilment could also be improved by using consolidations during deliveries. Real-world problems usually have to deal with dynamic and multi-objective constraints. Several methods exist to tackle these constraints, for the dynamism, it is common that changes to the solution can occur over time to account for new requests. To handle multiple objectives, the literature contains numerous methods involving more or less the dispatcher to make decisions. It was also demonstrated that anticipating future demands can improve the resulting solutions. Developing models and algorithms that can deal with all these constraints is of high practical value. Therefore, these requirements represent another research direction to contribute.

As stated by several researchers, ML and RL methods were successfully adapted to fit in the DI for packet routing. It was demonstrated that those methods can improve the overall efficiency of the routing decisions. However, there is a gap in the literature regarding this promising research area. On top of that, very few conducted such research applied to a logistics transportation problem. Since this problem is tackled from a PI perspective, this thesis takes advantage of the PI and DI similarities and solves the proposed VCRP with this approach.

Chapter 3

A bike routing problem with energy constraints

3.1 Introduction

Around 80 per cent of European citizens live in urban areas (Allen et al., 2010). Because of the high density of population in those areas, large quantities of goods have to be transported for commercial and domestic purposes. However, transportation has several impacts on the environment, such as resource consumption, noise, and greenhouse gas emissions (Bektaş and Laporte, 2011). Among these, gas emissions, in particular CO_2 emissions, are of great concern as they have consequences on human health and contribute towards climate change. In several countries, transportation is one of the biggest cause of CO_2 emissions. Thus, decision-makers are trying to reduce pollution in their countries or cities (Koning and Conway, 2016). Bike use is promoted and encouraged in countries like the Netherlands, Denmark and Germany. While bicycles have even become the standard over vehicles in cities like Amsterdam, some other cities like London voted laws to get entry fees for vehicles. As a result, Supply Chain Management businesses became more concerned about environmental issues and try to tackle those issues. Those growing concerns force to revise planning approaches for road transportation by for example changing last miles deliveries. One adopted solution is to focus on environment-friendly means such as bikes. Cycling is said to be one of the most energy efficient and healthy transport modes (Ehrgott et al., 2012). Moreover, cargo-based push-bikes as shown in Figure 3.1 can transport much heavier loads than traditional ones. However, the speed can be dramatically reduced with a heavy load due to the effort demanded of the cyclist. A cyclist could even struggle to move a fully loaded bike in a too steep road.

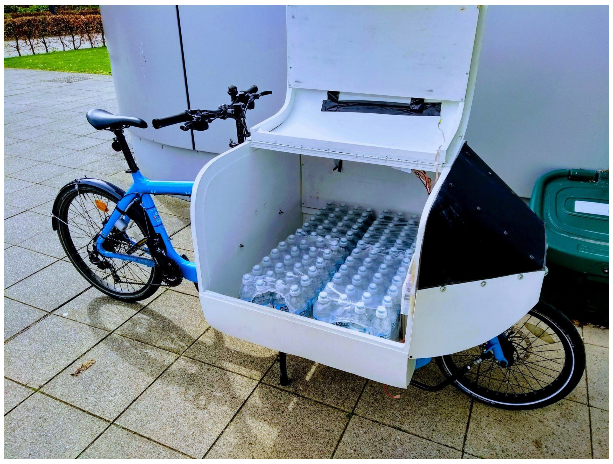


FIGURE 3.1: The cargo-based push-bike of the industry partner

Therefore, this chapter introduces a new vehicle routing variant called the Bike Routing Problem (BRP), that takes effort into account. The energy required to move push-bikes with cargo is calculated and checked so that it does not exceed a cyclist's power.

3.2 Problem description and formulation

3.2.1 Context

Formally, the BRP is defined on a complete graph $G = (N, A)$ with $N = \{0, 1, 2, ..., n\}$ as the set of nodes and A as the set of arcs defined between each pair of nodes. Node 0 is the depot where a homogeneous set of bikes $K = \{1, 2, ..., m\}$ is located. Each bike has a capacity of Q. The remaining nodes represent a set of customers $N_0 = N \setminus \{0\}$. Each customer $i \in N_0$ has a positive demand q_i representing the amount of product that needs to be picked up and transport to the depot. The distance from node i to node j is denoted by d_{ij}. Each cyclist k needs a certain amount of energy e_{ij}^k to travel over an arc (i, j). This amount is dependent on a number of factors, such as load weight and slope. The BRP consists in findings at most m bike routes such that the total travel distance is minimised and

- Each bike begins and ends at the depot;
- Each customer node is visited exactly once;
- The total load on each bike is less than its capacity Q;
- The total energy spent by each cyclist is less than a given value E.

The power requirements of a vehicle to move along a road segment depend on not only the dynamic parameters (e.g., speed and acceleration), but also the static ones as vehicle weight, aerodynamic drag, rolling resistance, and road grade, etc. We use the formula proposed in Martin et al. (1998) to compute the energy $\overline{e}_{ij}^k$ (in *joules/seconds*) generated by a biker k to go from node i to node j without any load. In their paper Martin et al. (1998), Martin et al. showed that some terms of the formula contribute far more than others that are thus negligible. By omitting these terms (Frictional loss in the drive chain, frictional loss in wheel bearings etc.) we get the same formula as in (Barth and Boriboonsomsin, 2009):

$$\overline{e}_{ij}^k = \frac{d_{ij}}{v}\left[w_k a v + w_k g v \sin(\theta_{ij}) + 0.5 c_w f \rho v^3 + w_k g c_r \cos(\theta_{ij}) v\right] \ \forall k \in K \ \forall i, j \in N \quad (3.1)$$

The parameters used in this formula and their default values are provided as follows:

- c_r: rolling resistance coefficient = 0.005 (for a typical bitumen road on clinchers)
- c_w: wind resistance coefficient = 0.5 (no headwind nor tailwind)
- ρ: air density = 1.226 kg/m^3 (at sea level)
- g: gravitational constant = 9.8 m/s^2
- w_k: the total weight of bike and biker = 100 kg
- a: the acceleration 0.5 m/s^2
- v: the speed of the bike on the road = 20 km/h (or 5.56 m/s)
- f: the frontal area of the bike and rider = 1.5 m^2
- θ_{ij}: the slope of the road node i to node j.

3.2.2 Mixed-integer linear programming formulation

We now formulate the problem as a Mixed Integer Linear Programming (MILP). Our formulation allows to define mathematically the problem and can be used to solve small-size instances to optimality with MILP solvers. There are three types of variables as follows:

- x_{ij}^k: binary variables equal to 1 if bike k travels from i to j

- l_i^k: real variables representing the load weight on bike k when it leaves node i
- e_{ij}^k: real variables representing the energy spent on arc (i, j) by bike k.

As shown in Bektaş and Laporte (2011), minimising the energy can increase the distance travelled by vehicles with detours (e.g. to avoid uphills). Since we consider cyclists, the distance remains more important than the energy that is thus handled as a constraint. Therefore, the objective function is to minimise the total distance travelled by bikes:

$$\textbf{Minimise} \sum_{k \in K} \sum_{i,j \in N} d_{ij} x_{ij}^k \tag{3.2}$$

subject to:

Each customer is visited by exactly one bike:

$$\sum_{k \in K} \sum_{j \in N} x_{ij}^k = 1 \; \forall i \in N_0 \tag{3.3}$$

Flow conservation constraint. Each bike must continue its tour until the depot:

$$\sum_{i \in N} x_{ij}^k = \sum_{i \in N} x_{ji}^k \; \forall k \in K \; \forall j \in N \tag{3.4}$$

Constraint for the depot. All bike tours must pass by the depot:

$$|N| \sum_{i \in N} x_{0i}^k \geq \sum_{i,j \in N} x_{ij}^k \; \forall k \in K \tag{3.5}$$

Capacity constraint. Each bike load must respect the bike's capacity

$$l_i^k \leq Q \; \forall i \in N_0 \; \forall k \in K \tag{3.6}$$

Relationship between l_i^k and x_{ij}^k: This constraint also removes sub-tours:

$$l_j^k \geq l_i^k + q_j - M_1 * (1 - x_{ij}^k) \; \forall k \in K \; \forall i \in N \; \forall j \in N_0 \tag{3.7}$$

$$l_j^k \leq l_i^k + q_j + M_1 * (1 - x_{ij}^k) \; \forall k \in K \; \forall i \in N \; \forall j \in N_0 \tag{3.8}$$

where M_1 is a large number and can be estimated by Q.

Load of bike k at depot is null:

$$l_0^k = 0 \; \forall k \in K \tag{3.9}$$

Energy constraint: The energy generated by a biker is limited to a given value:

$$\sum_{i,j \in N} \bar{e}_{ij}^k \leq E \quad \forall k \in K \tag{3.10}$$

where e_{ij}^k is the amount of energy that biker k generates to move from i to j and its computation is based on formula (3.1). By taking into account the time a bike needs to travel between i and j and its load weight when leaving node i, we get:

$$\begin{aligned} e_{ij}^k = \frac{x_{ij}^k d_{ij}}{v} \big[&(l_i^k + w_k) a v + (l_i^k + w_k) g v \sin(\theta_{ij}) + 0.5 c_w f \rho v^3 + \\ &(l_i^k + w_k) g c_r \cos(\theta_{ij}) v \big] \; \forall k \in K \; \forall i, j \in N \end{aligned} \tag{3.11}$$

These constraints are non-linear and can be linearised by using an additional variable $y_{ij}^k = x_{ij}^k l_i^k$. It equals 0 if $x_{ij}^k = 0$ and equals l_i^k if $x_{ij}^k = 1$. We then can replace (3.11) by the following constraints:

$$\begin{aligned} e_{ij}^k = &\big[a + g \sin(\theta_{ij}) + g c_r \cos(\theta_{ij}) \big] y_{ij}^k d_{ij} + \\ &x_{ij}^k d_{ij} \big[w_k a v + w_k g \sin(\theta_{ij}) + 0.5 c_w f \rho v^2 + w_k g c_r \cos(\theta_{ij}) \big] \; \forall k \in K \; \forall i, j \in N \end{aligned} \tag{3.12}$$

$$y_{ij}^k \leq M_2 x_{ij}^k \quad \forall k \in K \; \forall i, j \in N \tag{3.13}$$

$$y_{ij}^k \leq l_i^k \quad \forall k \in K \; \forall i, j \in N \tag{3.14}$$

$$y_{ij}^k \geq l_i^k + M_3(x_{ij}^k - 1) \quad \forall k \in K \; \forall i, j \in N \tag{3.15}$$

Constraints (3.12) express the relationship between variables e_{ij}^k and y_{ij}^k. Constraints (3.13) ensure that if $x_{ij}^k = 0$, the variable y_{ij}^k must be set to zero. The combination of constraints (3.14) and (3.15) requires that if $x_{ij}^k = 1$, the variable y_{ij}^k is set to l_i^k. Here, M_2 and M_3 are sufficiently large numbers and can be set to Q.

3.3 Meta-heuristic

The algorithm proposed in this chapter is an Evolutionary Local Search (ELS) derived from Prins (2009). We decide to select this metaheuristic because it is simple, fast and has been used widely in the literature to successfully solve a variety of VRP variants (e.g., Hà et al. (2013, 2014)). In the ELS method, a single solution is mutated to obtain several children that are then improved by local search operators. The next generation is the best solution among the parent and its children.

Our version of the algorithm includes the following: (1) a modified `split()` function which handles the energy and is bounded, (2) local search procedures which check the energy constraints (using the distance, load and slope) in $\mathcal{O}(1)$, (3) a mutation operator which swaps two nodes to improve the efficiency of the search for small instances. Algorithm 1 describes the overall ELS algorithm which handles giant tours and BRP solutions with `split()` and `concat()` functions described below. ni is the number of iterations (number of attempts to produce better local optima). nc is the number of children solutions generated in each iteration. Hence, the total number of calls to the local search is $ni * nc$. Function `cost()` is used to get the distance cost. The current best solution S_b is updated only when it is outperformed by the best child S.

The well-known savings heuristic from Clarke and Wright (1964) is used to initialise ELS. Their heuristic starts from a trivial solution with one dedicated trip per customer. Each iteration evaluates all capacity-feasible mergers (concatenations) of two trips and executes the one with the largest positive saving. The process stops when no such merger can be found. The resulting VRP solution must be converted into a giant tour to start the alternating cycle, using function `concat()`. Their algorithm was adapted to fit into our model by not only evaluating the capacity of bikes but also their energy.

Algorithm 1: ELS algorithm

```
Input: problem data;
Output: best solution $S_b$ found;
$S_b \leftarrow$ init_solution();
$\overline{T} \leftarrow$ concat($S_b$);
for $i$ from 1 to $ni$ do
    $\overline{f} \leftarrow cost(S_b)$;
    for $j$ from 1 to $nc$ do
        $T \leftarrow$ mutate($\overline{T}$);
        $S \leftarrow$ split($T$);
        $S \leftarrow$ local_search($S$);
        if $cost(S) < \overline{f}$ then
            $S_c \leftarrow S$;
            $\overline{f} \leftarrow cost(S)$
    if $cost(S_c) < cost(S_b)$ then
        $S_b \leftarrow S_c$;
        $\overline{T} \leftarrow$ concat($S_b$);
```

Algorithm 2: Bounded Split

```
//Input: a max number $K$ of bikes, an order of $n$ customer nodes;
//Output: an optimal solution regarding the given order of customers;
for $k$ from 0 to $K+1$ do
    $p[k][0] \leftarrow 0$;
    for $i$ from 1 to $n$ do
        $p[k][i] = \infty$;
for $i$ from 0 to $n-1$ do
    $load \leftarrow 0$;
    $j \leftarrow i+1$;
    while $j \leq n$ do
        if $j = i+1$ then
            $distance \leftarrow d_{0,j}$;
            $energy \leftarrow e_{0,j}$ //computed by Eq. (3.16) with current load;
        else
            $distance \leftarrow distance + d_{j-1,j}$;
            $energy \leftarrow energy + e_{j-1,j}$;
        $load \leftarrow load + q_j$;
        if $load \leq Q$ and $energy + e_{j,0} \leq E$ then
            for $k$ from 0 to $K$ do
                if $p[k][i] \neq \infty$ and $p[k][i] + distance + d_{j0} < p[k+1][j]$ then
                    $p[k+1][j] \leftarrow p[k][i] + distance + d_{j0}$;
                    $pred[k+1][j] = i$;
            $j \leftarrow j+1$;
```

As described in Prins (2009), our algorithm alternates between solutions encoded as TSP tours, called giant tours, and genuine BRP solutions. A split algorithm is used to get a BRP solution from a giant tour. The function adds all the depots to create a complete solution which satisfies the capacity and energy constraints. Like most papers on VRP meta-heuristics, the number of vehicles used is variable but bounded. The overall process is defined as follows. Given a giant tour of n customers such as $T = (T_1, T_2, ..., T_n)$, the function builds a weighted directed graph $H = (X, A, D)$. This idea was proposed by Beasley (1983) as a route-first cluster-second heuristic. The nodes in X are indexed from 0 to n: 0 is a dummy node while each node $i \neq 0$ represents a customer T_i. Each sub-sequence of customers $(T_i, T_{i+1}, ..., T_j)$ of the giant tour is evaluated to see if the trip $(0, T_i, T_{i+1}, ..., T_j, 0)$ is feasible. This evaluation not only includes the bike capacity, but also the total energy required for the trip. If the trip is feasible, it is modelled in the arc-set A by one arc $(i-1, j)$, with a weight equal to the trip cost. An optimal splitting of T into feasible trips corresponds to a min-cost path from node 0 to node n in the created graph. The Bellman algorithm is then used to find the shortest path problem which indicates how to separate the giant tour. Each selected arc represents a bike trip which is included in the returned solution.

$$e_{i,j} = \big[(load + w)a + (load + w)g\sin(\theta_{i,j}) + 0.5C_w f \rho v^2 + (load + w)gC_r\cos(\theta_{i,j})\big]d_{ij} \qquad (3.16)$$

Algorithm 2 shows the different steps of the split procedure. The main difference with Prins (2009) is that checking if a trip is feasible in $\mathcal{O}(1)$ also requires us to check the total energy using the formula in Eq. (3.16). At the end of the procedure, array *pred* contains several paths to reach the last node T_n in the graph H. Given a max bike number k and a node i, value $pred[k][i]$ represents the precedent node in the graph path. Therefore, reversely looping on *pred* with a decreasing k gives us the last customer node of each bike trip bounded by k trips. Since graph H is directed acyclic by construction and presents other characteristics described in Vidal (2016), other faster algorithms could be used.

After the local search, function `concat()` in Algorithm 3 is called to convert a BRP solution into a giant tour T by concatenating the sequences of customers of its different trips. Each copy of the depot node used as trip delimiters is then removed from the solution. Function `insert()` is used to insert a node at a given position in a solution route.

Algorithm 3: concat function

```
1 //Input: a solution $S$, a request ID, the depot node;
2 //Output: solution $S'$ found;
3 $S' \leftarrow \varnothing$;
4 foreach $node$ in $S$ do
5     if $node \neq depot$ then
6         insert($S', node$);
7 return $S'$;
```

Algorithm 4: Local search function

```
1 //Input: a solution $S$, string length $\lambda$, improvement policy $bi$;
2 //Output: a better solution $S$;
3 while ($i_1$ or $i_2$ or $i_3$ or $i_4$ or $i_5$) = true do
4     Classical2Opt($S, bi, i_1$);
5     CrossoverMove($S, bi, i_2$);
6     SwapTwoNodes($S, bi, i_3$);
7     OrOptMove($S, \lambda, bi, i_4$);
8     StringExchange($S, \lambda, bi, i_5$);
9 return $S$;
```

The Local Search (LS) described in Algorithm 4 includes the following procedures: Function `Classical2Opt()` takes a sub-sequence of customers in a trip and inverts its order; Function `Crossover()` replaces two edges by two other edges from two different trips; Function `SwapTwoNodes()` swaps two nodes from two different trips; Function `OrOptMove()` changes the position of a sub-sequence of customers from two different trips; Function `StringExchange()`, which exchanges two sub-sequences of customers from two different trips. To limit the running time of the procedure, the length of a string is restricted to $\lambda \in \{1, 2, 3\}$. Our local search operators are executed in a best-improvement fashion. After a best-improvement move is found, the procedure executes it, which modifies S, and sets a Boolean flag (last argument) to true. Otherwise, the flag is set to false. All neighbourhoods are searched again if at least one of them brings an improvement. Every procedure is given a solution S encoded in a list including several depots. The first and last nodes are depots while the others delimit the different trips. The local search cannot alter any depot position. Therefore, the LS procedures can only modify edges in one or two trips but cannot modify the trip length.

The main difference between our local search and Prins (2009) is that checking if a trip is feasible in $\mathcal{O}(1)$ is more complicated. This is due to the energy formula which includes the bike load at any given position in the trip. Hence, changing a node position at the beginning of the trip would affect the energy required at the end of the trip because the load would not be the same anymore. Therefore, the procedure has to check if the rest of the trip from the changed edge is still feasible. A straightforward approach would be to parse the new solution from the index of the changed node until a depot is reached. The function can therefore, return false if the constraints are violated and true otherwise. The best case scenario is when the modified node is the last one in a bike trip. The parsing will then calculate the energy only for one node. However, if the solution only contains a single trip and the changed node is the first one, the worst case scenario is that the parsing will have to go through the entire solution. In order to improve that and check each move in $\mathcal{O}(1)$, we use the following mechanism.

The formulation to compute the energy spent on arc (i, j):

$$e_{ij} = \frac{d_{ij}}{v}\big[(l_i^k + w_k)av + (l_i^k + w_k)gv\sin(\theta_{ij}) + \\ 0.5C_w f\rho v^3 + (l_i^k + w_k)gC_r\cos(\theta_{ij})v\big] \quad (3.17)$$

This can be rewritten as follows:

$$e_{ij} = d_{ij}l_i^k[a + g\sin(\theta_{ij}) + gC_r\cos(\theta_{ij})] + d_{ij}\big[w_k a + w_k g\sin(\theta_{ij}) + \\ 0.5C_w f\rho v^2 + w_k gC_r\cos(\theta_{ij})\big] \quad (3.18)$$

Only the first term depends on the bike load. The remaining terms depend on the arc only and do not depend on the load. We then can simplify the equation as follows:

$$e_{ij} = l_i^k A_{ij} + B_{ij} \quad (3.19)$$

Let S be a solution such as $S = (0, 1, 2, 3, 4, 0, 5, 6, 7, 0)$ where nodes 0 are the depots. When only considering the term depending on load, the energy of this solution's first trip is computed as follows:

$$e = A_{01}l_0 + A_{12}l_1 + A_{23}l_2 + A_{34}l_3 + A_{40}l_4 \quad (3.20)$$

As the load weight l_i^k represents the accumulated demands from the depot to node i, the equation can be rewritten as follows:

$$e = A_{12}q_1 + A_{23}(q_1 + q_2) + A_{34}(q_1 + q_2 + q_3) + A_{40}(q_1 + q_2 + q_3 + q_4) \quad (3.21)$$

where q_i is the demand of customer i. Finally the equation can be simplified as:

$$e = q_1(A_{12} + A_{23} + A_{34} + A_{40}) + q_2(A_{23} + A_{34} + A_{40}) + q_3(A_{34} + A_{40}) + q_4 A_{40} \quad (3.22)$$

As a result, an array A_i can be calculated to save the summation of A from node i to the depot. An array L_i can also be calculated to save the total load of bikes at node i. Therefore, if node 2 and node 6 are swapped, the energy in the first trip will change by an amount calculated as follows:

$$(q_6 - q_2)A_3 - e_{12} - e_{23} + e_{16} + e_{63} \tag{3.23}$$

This energy delta is then used to update the current energy value of the first trip. In this example, the energy for the second trip is then calculated in the same way and checked if it violates the constraint. It should be noted that L_i is used to compute $e_{16} + e_{63} - e_{12} - e_{23}$ in $\mathcal{O}(1)$.

Algorithm 5 depicts a template search function used in all the procedures to check if a move is feasible in $\mathcal{O}(1)$ with the mechanism described above. The procedure iteratively checks each possible position and moves the string only if the constraints are respected. The move is applied only if the indices i and j belong to two different trips and if the string does not contain any depot. This simplifies the function as otherwise, in the case of `SwapTwoNodes`(), the array A should be updated because of the other node change. The constraints for the first trip are not checked as we remove nodes from it, therefore it is necessarily feasible. Function `cost`() calculates the distance cost of a trip or solution. Function `demand`() returns the demand of a given node. Functions `energy`() and `load`() return the total energy and load of the trip containing the given node. Functions `energies`() and `loads`() initialise the arrays with a given solution. Function `move`() applies the move on a solution. For the procedure `OrOptMove`(), the move consists in relocating the string of customers from index i to index j.

The objective of the mutation procedure is to diversify the search. Here, we do not operate this procedure directly on a complete BRP solution, but indirectly on a giant tour. Thus, neither capacity nor energy constraint needs to be considered. We use two operators detailed in Algorithms 6 and 7 which are randomly selected at each iteration. The first one relocates a random node while the second one consists of swapping two different nodes randomly selected in the giant tour. After an operator is selected, p successive moves are executed. The value of p is modified during the search process. It is set to a minimum value p_{min} at the beginning and each time a new best solution is found. It is incremented whenever the mutation followed by local search returns a degraded solution, but never exceeding a maximum value p_{max}. Function `random`() is used to return a random number given a bound. Function `remove`() does this opposite of `insert`(), it removes a node from a given position.

Algorithm 5: Template for local search procedure based on `OrOptMove`()

```
//Input: a solution $S$, improvement policy $bi$, [string length $\lambda$];
//Output: a better solution $S$;
$d_{best} \leftarrow \infty$;
while $bFlag$ do
    $A \leftarrow$ energies($S$);
    $L \leftarrow$ loads($S$);
    $d \leftarrow$ cost($S$);
    $bFlag \leftarrow false$;
    for $i$ from $1$ to $|S| - \lambda - 1$ do
        for $j$ from $i + \lambda$ to $|S| - 1$ do
            $a \leftarrow j - \lambda + 1$;
            $b \leftarrow j$;
            $d_\Delta \leftarrow d_{S[j],S[i]} - d_{S[i-1],S[i]}$;
            $d_\Delta \leftarrow d_\Delta + d_{S[i+\lambda-1],S[j+1]} - d_{S[i+\lambda-1],S[i+\lambda]}$;
            $d_\Delta \leftarrow d_\Delta + d_{S[i-1],S[i+\lambda]} - d_{S[j],S[j+1]}$;
            $d_{new} \leftarrow d + d_\Delta$;
            $e \leftarrow$ energy($a$) //get the total energy required for trip containing node $a$;
            $l \leftarrow$ load($a$) //get the total load for trip containing node $a$;
            $l_\Delta \leftarrow 0$;
            for $z$ from $0$ to $\lambda$ do
                $c \leftarrow z + a$;
                $e_\Delta \leftarrow e_\Delta + e_{S[c-1],S[c]}$;
                $l_\Delta \leftarrow l_\Delta + q_c$;
            $e_\Delta \leftarrow e_\Delta + e_{S[b],S[b+1]}$;
            $e_\Delta \leftarrow e_\Delta - e_{S[a-1],S[b+1]}$;
            $e_{new} \leftarrow e + e_\Delta + l_\Delta * A[b+1]$;
            $l_{new} \leftarrow l + l_\Delta$;
            if $d_\Delta < 0$ and $d_{new} < d_{best}$ and $e_{new} < E$ and $l_{new} < Q$ then
                $d_{best} \leftarrow d_{new}$;
                $i_{best} \leftarrow i$;
                $j_{best} \leftarrow j$;
                $bFlag \leftarrow true$;
    if $bFlag = true$ then
        $S \leftarrow$ move($i_{best}, j_{best}, S$);
        $bFlag \leftarrow bi$;
return $S$
```

3.4 Computational results

All the computational results were obtained with a computer that has the following specifications: a CPU 'Intel(R) Core(TM) i9-7900X CPU @3.30GHz' and 32 GB of RAM. The proposed algorithm is developed in a framework using Java 1.8. The model is implemented using the CPLEX OPL library and included in the framework. The

Algorithm 6: Move operator

```
//Input: a solution $S$;
//Output: solution $S'$;
$S' \leftarrow S$;
$index1 \leftarrow$ random($|S'|$);
$index2 \leftarrow$ random($|S'|$);
$node \leftarrow$ remove($S', index1$);
insert($S', node, index2$);
return $S'$;
```

Algorithm 7: Exchange operator

```
//Input: a solution $S$;
//Output: solution $S'$;
$S' \leftarrow S$;
$index1 \leftarrow$ random($|S'|$);
$index2 \leftarrow$ random($|S'|$);
$node1 \leftarrow$ remove($S', index1$);
$node2 \leftarrow$ remove($S', index2$);
insert($S', node1, index2$);
insert($S', node2, index1$);
return $S'$;
```

version 12.7 of CPLEX is used with its default configuration and therefore allows parallel computing.

3.4.1 Data

Generator parameters	Values
customer number	{10, 25, 50, 100}
bike number	{1, 5, 10}
bike capacity	100 kg
bike speed	5.56 m/s
request load	1-10 kg
surface	1.5 m^2
acceleration	0.5 m/s^2
cyclist + bike weight	100 kg
3D coordinates (x,y,z)	0-100

TABLE 3.1: Generator parameters

Request loads and customer locations were generated randomly with a uniform distribution. x and y represent the latitude and longitude respectively and are bounded by

0-100. z represents the altitude and is bounded by 0-10. Parameter E was firstly set as a big value to get feasible solutions. Then for each solution, the energy calculated was selected and used to modify the instances. The max energy parameters were therefore set with values slightly lower than the energy calculated to constrain the algorithm to find other solutions.

3.4.2 Parameter tuning

Description	Parameters modified	Energy values
	none	5314062
speed	v=1	205206
speed	v=10	23555000
bike and load weight	w+l=200	10436562
acceleration	a=1	2814062
slope	θ=0.349	13686169
slope	θ=0.698	21033700
surface	A=2	5505625

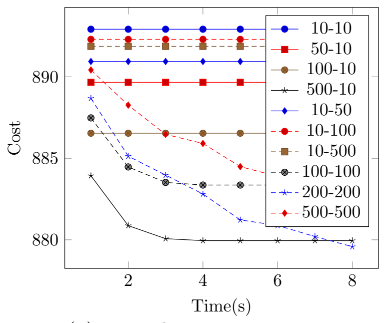
TABLE 3.2: Energy formula sensitivity

Table 3.2 shows different values of the equation (3.17). Each value was obtained by modifying a single parameter from these default values - v=5, w+l=100, a=2, θ=0, A=1. It can be seen that the speed is the variable that affects the most the energy required while the surface is the least. All the other variables play an equally important role in the energy required.

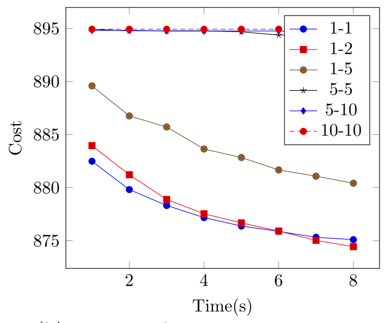
ELS parameters	Values
ni	200
nc	200
bi	true
λ	2
p_{min}	1
p_{max}	2

TABLE 3.3: ELS parameters

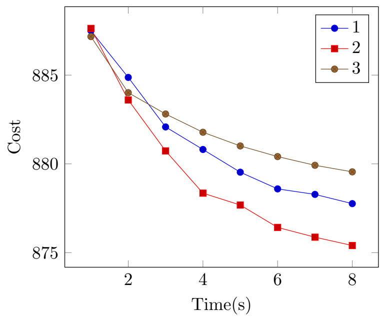
Figures from 3.2(a) to 3.2(d) depict the convergence of ELS and were used to select the default parameters values in Table 3.3. The plots show the average cost at each time-stamp from 1 to 8 seconds for 30 runs.



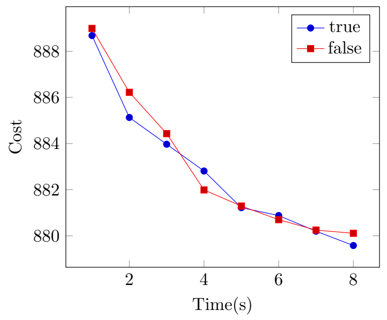
(a) ni and nc parameters



(b) pmin and pmax parameters



(c) string length λ parameter



(d) bi parameter

FIGURE 3.2: Convergence of the algorithm with different parameter values

3.4.3 CPLEX and ELS comparison

	CPLEX				ELS			
Instance	Objective	Lower Bound	Gap	Time (s)	Avg objective	Avg time (s)	Best objective	Best time (s)
10c1b	269.16	269.16	optimal	0	269.16	0	269.16	0
10c1b	236.85	236.85	optimal	0	236.85	0	236.85	0
10c1b	183.89	183.89	optimal	0	183.89	0	183.89	0
25c5b	387.41	223.33	42.35	OOM	374.48	19	374.48	2
25c5b	234.06	152.19	34.98	OOM	227.17	17	227.17	2
25c5b	731.08	110.26	84.92	OOM	481.28	38	476.95	14
50c5b		199.21		>10800	432.5	365	431.67	833
50c5b		154.56		>10800	472.17	52	470.28	24
50c5b		173.5		>10800	492.41	39	492.34	101
100c10b		224.2		>10800	740.74	7592	732.98	10040
100c10b		229.67		OOM	889.88	7366	1063.51	9911
100c10b		235.31		OOM	1081.17	4558	1063.51	4107

TABLE 3.4: CPLEX and ELS performances on clustered instances

	CPLEX				ELS			
Instance	Objective	Lower Bound	Gap	Time (s)	Avg objective	Avg time (s)	Best objective	Best time (s)
10c1b	253.89	253.89	optimal	0	253.89	0	253.89	0
10c1b	333.04	333.04	optimal	0	333.04	0	333.04	0
10c1b	303.31	303.31	optimal	0	303.31	0	303.31	0
25c5b	641.28	386.14	39.79	OOM	574	11	574	6
25c5b	741.12	412.51	44.34	>10800	723.79	34	722.48	43
25c5b	688.1	351.58	48.91	OOM	493.47	18	493.47	43
50c5b		494.8		>10800	702.54	111	701.42	397
50c5b		548.95		OOM	820.32	372	779.54	112
50c5b		519.35		>10800	701.56	125	700.44	27
100c10b		612.2		OOM	1361.91	1747	1324.31	2672
100c10b		656.5		>10800	1453.71	5823	1416.05	9204
100c10b		642.75		OOM	1362.53	3629	1342.56	6093

Table 3.5: CPLEX and ELS performances on random instances

	CPLEX				ELS			
Instance	Objective	Lower Bound	Gap	Time (s)	Avg objective	Avg time (s)	Best objective	Best time (s)
10c1b	290.75	290.75	optimal	0	290.75	0	290.75	0
10c1b	305.41	305.41	optimal	0	305.41	0	305.41	0
10c1b	328.01	328.01	optimal	0	328.01	0	328.01	0
25c5b		349.68		>10800	732.58	17	715.92	27
25c5b	712.04	431.05	39.46	OOM	677.43	13	674.41	12
25c5b	594.36	407.51	31.44	OOM	551.53	48	548.16	22
50c5b		476.1		>10800	839.83	490	775.8	968
50c5b		546.89		>10800	809.72	118	783.32	146
50c5b		487.35		>10800	863.23	591	852.63	284
100c10b		640.39		>10800	1349.05	4651	1320.75	6247
100c10b		653.6		>10800	1434.31	5412	1394.61	5071
100c10b		598.69		>10800	1240.62	6133	1225.48	5079

Table 3.6: CPLEX and ELS performances on clustered-random instances

CPLEX was run only once with a time limit of 3 hours. ELS was run 30 times with the parameters described in Table 3.3. However, for the instances containing a single bike, parameter p_{max} was set to 5 as the algorithm mostly relies on the mutation operators to find solutions. This is due to the local search procedures requiring two distinct trips to operate. Tables 3.4 to 3.6 show the comparison of CPLEX and ELS. The instance names give some details about their characteristics, for example, "10c1b" means that the instance includes 10 customers and 1 bike. While column "Best objective" gives the objective value of the best solution found among the 30 runs, column "Best time (s)" provide the time needed to find this same solution. Column "Time (s)" can sometimes contain "OOM" which stands for "Out Of Memory" and indicates that CPLEX was not able to continue the search as too much memory was being needed. As a consequence, the process was automatically stopped. Three sets of instances were generated by varying the customer locations - random, clustered and semi-clustered. To generate clustered instances, 5 points were randomly selected as centres of the clusters with 10 as radius. Then for each customer, its location is randomly generated within the boundaries of a random cluster. As expected, although CPLEX struggles to scale up for instances containing 50 or more customers, ELS does not seem to have difficulties and can find better solutions. Moreover, ELS is able to find all the optimal solutions found by CPLEX in less computational time.

3.4.4 Real-world case

The need for such a model is reinforced by our collaboration with Peloton, a company which delivers goods using bikes in Liverpool. As their customer identities and locations are private, 50 restaurants in Liverpool were randomly selected to create instances. The depot is Asda Breck Road Superstore. The bike number is bounded to 5, the request load is bounded by 1 and 5 and the bike capacity is 50 kg. All the remaining properties of the instance are set as described in Table 3.1. An OpenStreetMap API was used to get the coordinates (latitude, longitude, altitude). The real distances were also got from this API by requesting the shortest path for each pair of customers.

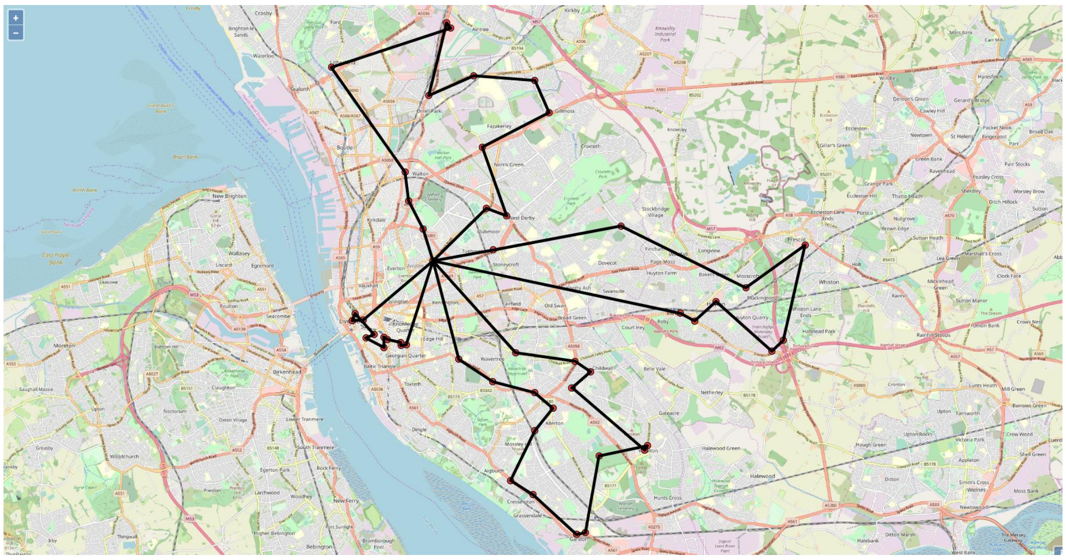


FIGURE 3.3: BRP solution in Liverpool

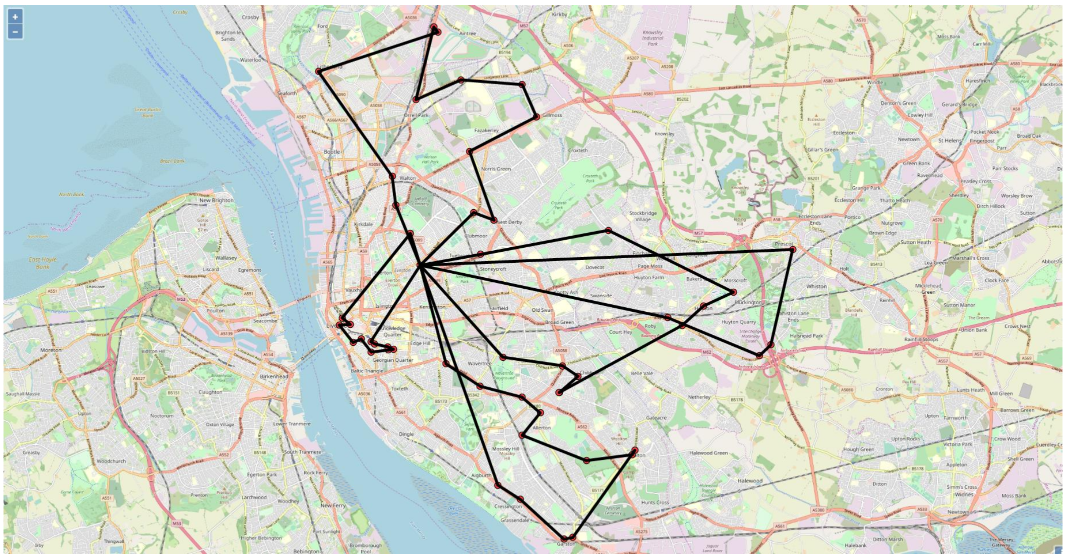


FIGURE 3.4: BRP solution in Liverpool with tighter energy constraint

The first instance is created with a large value for the energy constraint. Figure 3.3 shows the solution route for such a delivery. Figure 3.4 shows the solution for a second instance based on the previous one but with a tighter energy constraint.

	Bike 1	Bike 2	Bike 3	Bike 4	Bike 5	Totals
Instance 1						
distance	29058	12362	29385	33233	0	104038
energy	34727182	15120361	37648382	39928471	0	127424396
load	28	32	45	30	0	135
uphill	21.97	18.83	28.25	0.29	0	69.27
Instance 2						
distance	30271	27840	27803	28294	14575	128783
energy	33795350	33214404	33235254	33081808	17537016	150863832
load	12	31	32	25	35	135
uphill	7.84	17.26	20.4	20.4	0.37	66.27

TABLE 3.7: Details of the bike trips with and without a tight energy constraint

Table 3.7 presents the results of the algorithm in both instances. Due to their capacities of 50 kg, 4 bikes are needed to satisfy all the customers in the first instance. In both figures we can notice some small triangles made by the itineraries of the bikes. Although this might not seem as the shortest path for distances as the crow flies, it is however the shortest path when considering real distances. In some places, this is due to the network of real roads e.g. one way roads or detours.

Adding an energy limit on bikes forces the algorithm to use one more bike in the second instance compared to first one. As a result, we can see that the bikes' loads are more balanced. Moreover, the total distance and energy required have increased. This can be explained by the fact that one more bike has to travel to and from the depot to satisfy customers that could have been handled by another bike already in the area.

$$\frac{\sum_{i,j \in N_0} \max(x_{ij}^k \theta_{ij}, 0)}{|trip|} \tag{3.24}$$

We compute a metric representing the uphills a bike has to go through with Eq. (3.24) where $|trip|$ is the number of arcs in its trip. This metric measures how steep and therefore difficult a bike trip is. From the rows "uphill" in Table 3.7, it can be seen that the difficulty of the trips is more balanced, this is also supported by the slightly decreased value of the total uphill compared with the instance 1. The algorithm avoided steep roads.

Overall, the two solutions are not dramatically different because the objective is still to minimise the distance. However, we suspect that the distances in an instance can affect such a change. When distances are too large, changing the customer order to avoid uphills would not necessarily decrease the energy as the detours induced could require more energy. Moreover, changing the sequence of customers would also modify the load carried and therefore impact the energy. In order to witness a bigger change, one would need instances with customers closer to each other with good elevation gradients. Alternatively, one could include another constraint limiting the instantaneous energy required to move a bike with its load on each arc. As shown in Table 3.2, slopes play an important role in the energy, therefore the objective could also be changed to include the energy so that the algorithm would better avoid uphills. Otherwise, limiting the energy per bike trip would mainly result in re-balancing the load between bikes as in Figure 3.4. Finally, in our model, the altitude gradient between 2 customers is used to calculate the elevation profile of each arc i, j. However, 2 customers at the same altitude could be linked by a road that could have a series of uphills and downhills which should also be considered.

3.5 Conclusion

This chapter introduced a bike routing problem and made several contributions. First, a new model was proposed to tackle deliveries by push-bikes while taking into account the road slopes. It considers the energy required to move the bike as the load accumulates during the trip. A linearisation was applied to simplify the model and avoid using non-linear solvers. Second, a modified Evolutionary Local Search was introduced to solve large instances in a reasonable amount of time. A new split function was designed to handle energy constraints. Third, a sensitivity analysis was conducted to show how each parameter influences the performance. This gave a set of default values used for a comparative analysis of ELS and the proposed mathematical model solved by CPLEX. The computational results demonstrated the efficiency of our method compared to CPLEX. Moreover, an instance based on real-world customers was created to better demonstrate the relevance of the method.

Further research directions can be investigated. Grappe et al. (1997) studied the power required from a cyclist to move given a certain position. They concluded that an important part of the energy is due to the aerodynamic drag of air. Due to the cargo-based bikes that are being used, the frontal surface area is not negligible when facing significant winds. Therefore, studying the BRP while considering the wind direction could lead to better itineraries. Another improvement could be to consider real characteristics of the roads such as the elevation profile or alternative paths. Two customers could be linked by several paths of different elevation profiles which could require different energy from the cyclist. The energy constraint could also be modified to limit the instantaneous energy on a given road section instead of considering the accumulated energy for a trip. Finally, the model could be extended to an arc routing problem for another type of delivery.

Chapter 4

A rich multi-cross-docking VRP with pickup and delivery

4.1 Introduction

Commonly, the VRP assumes a homogeneous fleet of vehicles to serve a set of customers requiring delivery and pickup services with time windows. Sometimes different attributes are considered alongside another one or two. However, in many practical situations, companies need to consider numerous attributes of the VRP at once. The large numbers of transactions implying goods to be delivered to customers all over the world in an efficient way is a challenge for the logistics companies. To face this challenge, many research efforts have been made with a strong focus on VRP attributes.

Many VRP attributes have been mostly studied independently, however, emerging concepts like the PI emphasises the need to consider multiple attributes simultaneously. Few papers like Vidal et al. (2014) attempt to unify and solve numerous problems with a single general solution. This kind of unification eases the choice of a solution algorithm from the literature given a particular problem. As problems with cross-docking and the PI are given more interest, companies will need to face optimisation problems with such constraints. Therefore, this chapter aims to solve a rich VRP with pickup and delivery including several attributes at once to fill the gap in the literature. To tackle this problem, a linear programming model is first introduced and solved with CPLEX for comparison purposes on small instances. Then a multi-threaded SA is proposed to handle real-world size instances.

4.2 Problem description and formulation

4.2.1 Context

In this work, we aim at scheduling routes of a set of vehicles to satisfy all the customer requests. The following list of realistic properties are handled: capacitated, heterogeneous, mixed pickup and delivery, multiple depots, open route, different start/end depots, multiple cross-docks, customer time windows, site-dependent. Although this work is not concerned about the scheduling at cross-dock doors, it differs from most of the work studying cross-docking as it gathers numerous VRP attributes and models them with new constraints. As stated by Sarraj et al. (2013), a perceived possible drawback of the PI comes from the shift from direct-shipping to distributed transport involving containers being consolidated in several cross-docks with a possible costs increase. However, they showed that PI scenarios can result in significantly lower overall costs hence making it a profitable alternative. It is also demonstrated that by using consolidations, PI can induce a significant gain regarding transportation fulfilment rate. Many other researchers mentioned the benefit of using cross-docking. Yu et al. (2016) studied cross-docking while considering the cost of hiring vehicles. In Yu et al. (2015), they discussed the importance of cross-docking for efficient distribution networks as the costs of holding and handling can be lowered in warehouses. Similarly, Chen et al. (2016) stated that cross-docking can be an extremely efficient strategy when the inventory costs are high. While showing that cross-docking can increase the cost of vehicle used, Ahmadizar et al. (2015) also emphasised the efficiency of consolidation to minimise the total overall costs. Other research proved the reduction of costs due to cross-docking strategies in the SCM (Kreng and Chen, 2008). Real-world cases also support those findings (Wang et al., 2017). In some scenarios, a direct-shipping strategy can still incur lower costs (Nikolopoulou et al., 2017). However, when P&D pairs are remotely positioned and clustered, cross-docking can reduce the cost compared to the direct-shipping. This is also the case when facing a densely connected network with a many-to-many relation between suppliers and customers. Hence the need for a flexible model capable of handling both strategies which could resemble a many-to-many problem because of standardised PI-containers and consolidation. Several simulations already demonstrated the benefit of the PI when simultaneously considering different cost related to vehicles, truck fulfilment, distance travelled, handling, storage etc. Moreover, as the PI is about collaboration by sharing resources, even if the cost of a company's fleet could increase, it has been shown that the global cost for all the companies collaborating would certainly decrease as they would share the fleet (Montreuil, 2011; Ballot et al., 2012). As a consequence, our model designs a set of routes for vehicles while only considering the cost related to driving time.

In our VRP with cross-docks, a logistics organisation operates with multiple cross-dock facilities O to satisfy a set of customers C calling for transportation requests R. The problem is defined on a graph including a set of nodes $N = C \cup O$ in which for each pair of nodes i and j, there exists an arc (i, j) of driving time d_{ij}. Each customer requests is characterised by a container, an origin and a destination given by two customers $c_1, c_2 \in C$ where the first denotes the pickup point, and the second one denotes the delivery point. A customer request is satisfied when a vehicle picks up the container at the origin and delivers it at the destination. Cross-docks and customers require a service time s_i to handle requests. All requests are associated with their demands/loads $q_r > 0$ and must be satisfied between a time window $[A_i, B_i]$. For a given request r, its pickup and delivery locations are represented by the set $\{H_r^p, H_r^d\}$. Each request has the flexibility to pass through one or multiple cross-docks to be consolidated with others but is not forced to visit any cross-dock if a direct shipping strategy is more efficient. Moreover, requests are not necessarily expected to arrive at the cross-dock simultaneously if an exchange is happening. The organisation uses a set of vehicles K in which each vehicle $k \in K$ has a capacity Q_k, a type w_k and a set of assigned start and end cross-docks $O^k = \{o_s^k, o_e^k\}$. A single route can contain pickup as well as delivery locations but cannot include the same cross-dock more than once, except for the cross-dock depot. Each vehicle k is available within a time window $[E^k, F^k]$ and can only visit nodes for which its type is allowed with $z_i^{w_k}$ being equal to 1. Finally, to handle open routes, a dummy node o_d representing a cross-dock depot is introduced in the graph. It is assumed that each arc connecting the dummy cross-dock to another node of the graph has zero driving time. As a result, any vehicle k with an open route can be addressed by the model with $o_s^k = o_e^k = o_d$.

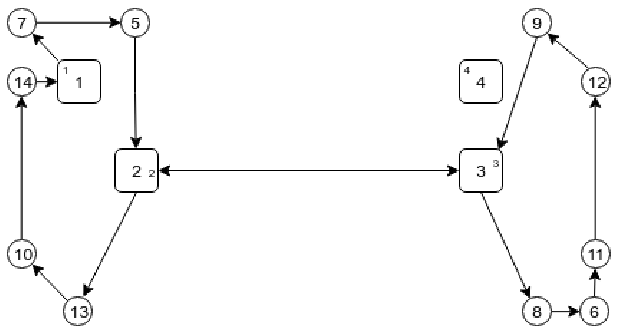


Figure 4.1: VRP solution with load exchanges

Figure 4.1 shows an example of a VRP solution which includes 4 cross-docks (squares), 10 customer locations (circles) and 4 vehicles represented with 4 small numbers in the cross-docks. Five requests call for transportation of containers from nodes 5, 7, 9, 11 and 13 to nodes 6, 8, 10, 12 and 14 respectively. In this scenario, pickup and delivery nodes are located in different cities and therefore must be done by different vehicles. Moreover, some constraints force vehicles to return to the depot within time windows. As a result, vehicles are required to transit via the cross-dock facilities to enable consolidation to share the transport work. Vehicle 1 first picks up the containers from nodes 7 and 5 to drop them to cross-dock 2. Then vehicle 2 carries these containers to cross-dock 3 where vehicle 3 picks them to deliver customers 8, 6. Customers 11 and 12 are also satisfied by vehicle 3 during its journey in which the container from customer 9 is brought to the cross-dock 3. After a wait, vehicle 2 can come back to cross-dock 2 with the container to be delivered by vehicle 1 which was also waiting before visiting customers 13, 10 and 14. The model gives vehicles the flexibility to mix pickups and deliveries through several cross-docks as for customers 5, 7, 6 and 8 or to directly deliver containers as for customers 13, 14, 11 and 12. Cross-docks can be left unused if necessary as for cross-dock 4 and an alternative scenario could allow vehicle 1 to return to another depot or none after visiting customer 14.

4.2.2 Mixed-integer linear programming formulation

The decision variable y for request transportation and the Big-M formulation/method are inspired by Maknoon and Laporte (2017) as this work also considers multiple cross-docks. With M as a large constant slightly greater than the highest value of variables B_i, the model is defined as follows:

$$\textbf{Minimise} \sum_{k\in K}\sum_{i\in N}\sum_{j\in N} x_{ij}^k * d_{ij} \tag{4.1}$$

subject to:

$$u_j^k \leq B_j \ \forall k \in K,\ \forall j \in C \tag{4.2}$$

$$v_i^k + M\left(1 - \sum_{j\in N\setminus i} x_{ij}^k\right) \geq A_i + s_i \ \forall k \in K,\ \forall i \in C, \tag{4.3}$$

$$v_o^k + M\left(1 - \sum_{j\in N\setminus o} x_{oj}^k\right) \geq E^k \ \forall k \in K, \ | \ o = o_s^k \tag{4.4}$$

$$u_o^k - M\left(1 - \sum_{j\in N\setminus o} x_{jo}^k\right) \leq F^k \ \forall k \in K, \ | \ o = o_e^k \tag{4.5}$$

$$\sum_{k\in K}\sum_{j\in N\setminus i} x_{ij}^k = 1 \ \forall i \in C \tag{4.6}$$

$$\sum_{j\in N\setminus i} x_{ij}^k \leq z_i^{w_k} \ \forall k \in K, \ \forall i \in N \tag{4.7}$$

$$\sum_{i\in N} x_{ij}^k = \sum_{i\in N} x_{ji}^k \ \forall k \in K, \ \forall j \in N\setminus O^k \tag{4.8}$$

$$\sum_{i\in N} x_{ij}^k \leq 1 \ \forall k \in K, \ \forall j \in O \tag{4.9}$$

$$v_i^k + d_{ij} \leq u_j^k + M\left(1 - x_{ij}^k\right) \ \forall k \in K, \ \forall i,j \in N \tag{4.10}$$

$$v_i^k + d_{ij} \geq u_j^k - M\left(1 - x_{ij}^k\right) \ \forall k \in K, \ \forall i,j \in N \tag{4.11}$$

$$u_j^k + s_j * \sum_{i\in N\setminus j} x_{ij}^k \leq v_j^k \ \forall k \in K, \ \forall j \in N\setminus\{o_s^k, o_e^k\} \tag{4.12}$$

$$\sum_{r \in R} q_r y_{rij}^k \leq Q_k \ \forall i, j \in N,\ \forall k \in K \tag{4.13}$$

$$x_{ij}^k \geq y_{rij}^k \ \forall k \in K,\ \forall r \in R,\ \forall i, j \in N \tag{4.14}$$

$$\sum_{k \in K} \sum_{i \in N} y_{rio}^k \leq 1 \ \forall r \in R,\ \forall o \in O \tag{4.15}$$

$$\begin{cases} \textbf{if } o_s^k = o_e^k \\ \sum_{i \in N} x_{ai}^k + \sum_{i \in N} x_{ib}^k = 2 \mid a = o_s^k, b = o_e^k \\ \textbf{otherwise} \\ \sum_{i \in N} x_{ai}^k = 1 \mid a = o_s^k \\ \sum_{i \in N} x_{ia}^k - x_{aa}^k = 0 \mid a = o_s^k \\ \sum_{i \in N} x_{bi}^k = 0 \mid b = o_e^k \\ \sum_{i \in N} x_{ib}^k + x_{aa}^k = 1 \mid a = o_s^k,\ b = o_e^k \end{cases} \quad \forall k \in K \tag{4.16}$$

$$\sum_{k \in K} \sum_{j \in N} y_{raj}^k = 1 \ \forall r \in R \mid a = H_r^p \tag{4.17}$$

$$\sum_{k \in K} \sum_{j \in N} y_{rja}^k = 0 \ \forall r \in R \mid a = H_r^p \tag{4.18}$$

$$\sum_{k \in K} \sum_{j \in N} y_{rja}^k = 1 \ \forall r \in R \mid a = H_r^d \tag{4.19}$$

$$\sum_{k \in K} \sum_{j \in N} y_{raj}^k = 0 \ \forall r \in R \mid a = H_r^d \tag{4.20}$$

$$v_a^k \leq u_b^k \; \forall k \in K, \; \forall r \in R \mid a = H_r^p, \; b = H_r^d \tag{4.21}$$

$$\sum_{i \in N} y_{ria}^k - \sum_{j \in N} y_{raj}^k = 0 \; \forall r \in R, \; \forall k \in K, \; \forall a \in C \backslash H_r \tag{4.22}$$

$$\sum_{k \in K} \sum_{i \in N} y_{rio}^k = \sum_{k \in K} \sum_{i \in N} y_{roi}^k \; \forall r \in R, \; \forall o \in O \backslash H_r \tag{4.23}$$

$$u_o^{k'} - M \left(1 - \sum_{i \in N} y_{rio}^{k'} \right) \leq v_o^k + M \left(1 - \sum_{i \in N} y_{roi}^k \right) \forall r \in R, \; \forall o \in O, \; \forall k, k' \in K \tag{4.24}$$

$$x_{ij}^k \in \{0, 1\} \; \forall i, j \in N, \; \forall k \in K \tag{4.25}$$

$$y_{rij}^k \in \{0, 1\} \; \forall r \in R, \; \forall i, j \in N, \; \forall k \in K \tag{4.26}$$

$$u_i^k \in \mathbb{R} \; \forall i \in N, \; \forall k \in K \tag{4.27}$$

$$v_i^k \in \mathbb{R} \; \forall i \in N, \; \forall k \in K \tag{4.28}$$

Eq. (4.1) is the objective considered in this chapter, it is the minimisation of the total driving time of all the vehicles (without waiting times). The objective is subject to constraints 4.2 - 4.28. Constraints (4.2) and (4.3) enforce that each customer j is available for pickup or delivery between times A_j and B_j. Similarly, constraints (4.4) and (4.5) enforce that each vehicle is only available between times E^k and F^k. Constraint (4.6) imposes that each customer location is served by exactly one vehicle. Constraint

(4.7) ensures that pickups and deliveries are made by allowed vehicles. Constraint (4.8) means that each vehicle k can only start and end its route at its assigned cross-docks o_s^k and o_e^k, respectively. Constraint (4.9) forbids vehicles to visit the same cross-dock more than once. Constraints (4.10) and (4.11) computes the arrival and leaving times at node j which in turn depends on the arrival and leaving times of node i and the driving time between the two nodes. Given an arrival time, the service time at node j is considered by constraint (4.12) to calculate the leaving time of a vehicle. Constraint (4.13) tracks the vehicle load that must respect its capacity. For each request, constraint (4.14) links its transfer decision to a vehicle route. Constraint (4.15) prevents requests from visiting a same cross-dock more than once. Constraint (4.16) forces each vehicle to depart and arrive at its assigned cross-dock depots, but it must not visit the depots more than once ($x_{aa}^k = 1$ if vehicle k is not used). Constraints (4.17) and (4.20) allow request pickup H_r^p or delivery H_r^d to be handled by a vehicle. The precedence constraints of pickups and deliveries are checked by constraint (4.21) as they must be consistent. Constraint (4.22) forbids a vehicle to drop any request load before reaching the delivery location. Constraint (4.23) controls the flow of requests entering a cross-dock o which has to leave this cross-dock. Constraint (4.24) synchronises each vehicle k leaving a cross-dock with another vehicle v carrying its loads to be consolidated. Constraints (4.25)-(4.28) define the domain of the decision variables as follows. Variable x_{ij}^k in (4.25) is the route decision, it is equal to 1 if and only if vehicle k travels from node i to node j, otherwise, x_{ij}^k equals 0. Variable y_{rij}^k in (4.26) is the transportation decision, it is equal to 1 if customer request r is transported using vehicle k on its route from node i to node j, otherwise, y_{rij}^k equals 0. The variables u_i^k in (4.27) and v_i^k in (4.28) represent respectively the arrival and leaving time of vehicle k at node i.

4.3 Meta-heuristic

As shown in Section 4.4, the mathematical model in Section 4.2 cannot be solved exactly for large instances within a reasonable time. Therefore a meta-heuristic approach is considered - a multi-Threaded Simulated Annealing with Memory (TSAM). Simulated Annealing (SA) algorithms have been successfully used to solve combinatorial problems such as the VRP. Moreover, as shown in the result section, SA tested on the selected reference benchmark in Li and Lim (2001) yielded good performance for instances up to 200 customers. Consequently, proposing a new SA to solve this problem and comparing it with CPLEX and best-known results on large instances from SINTEF is the adopted methodology in this chapter. Researchers, however, can easily integrate any other global search algorithms of their choice with the features described in this work to handle the proposed model or a variation of it.

4.3.1 Architecture

The proposed algorithm has been inspired by Li and Lim (2001) and extended to tackle the new model. The differences between the algorithm from Li and Lim (2001) and TSAM are as follows. First, functions `overall`(), `restart`(), `neighbour_search`(), and `simulated_annealing`() in Algorithms 8, 9, 24, 26 were modified and new functions were introduced. Second, the algorithm was parallelised to better cope with different instance characteristics simultaneously. The parallelisation allows the algorithm to apply traditional operators on a solution while trying to add consolidations without slowing down the overall search. The tabu list was removed and a memory mechanism was added for the threads to communicate and exchange their last created solutions. Third, operator `PD_rearrange`() was removed and new ones were added to handle the model constraints. For example, operator `swap`() in Algorithm 29 is introduced to better handle the "different start/end depots" attribute of the model while operator `consolidation`() in Algorithm 31 handles attribute "multiple cross-docks".

Figure 4.2 shows the architecture of the algorithm. Three different threads are launched on procedure `overall`(). Basically, procedure `overall`() handles the current solution which will be altered by the other functions. Procedure `restart`() is used to explore several times a different neighbourhood of the current solution. Compared to procedure `random`() in Algorithm 25, procedure `neighbour_search`() only returns a solution that is better than the current one. However, they both use a random operator from a list *op_list*. By calling procedure `random`(), procedure `simulated_annealing`() is used to allow the exploration of worse neighbourhoods in order to escape local minima.

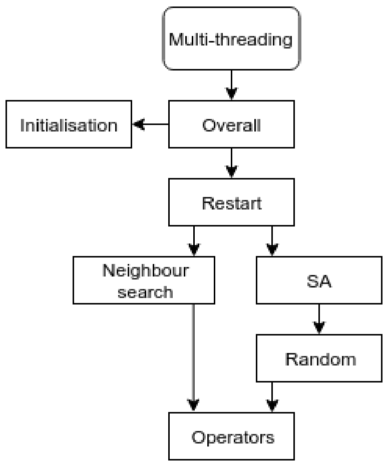


FIGURE 4.2: TSAM function architecture

The algorithm uses hierarchical clustering to extract clusters from the instance at hand. The heights of the clusters are used to identify the group of requests that must be consolidated together. If two requests share the same pickup and delivery node clusters (two distinct clusters), then they belong to the same group. Otherwise, they belong to separate groups. The group list *req_groups* is then used in function `PD_consolidate`() to ensure that the operators modify the requests (from the same group) in the same way.

Each solution and its vehicle routes have a flexible size which depends on the number of visited customer locations/nodes. In Figure 4.3, the vehicle tours/routes are delimited by the departure and arrival cross-docks/depots which are the numbers without any subscript. The ones with subscripts can be the customer or cross-dock nodes. These subscripts are the links between the nodes which represent the request travel paths. In Figure 4.3, such links are shown with the subscript numbers. Each node has a list of request path IDs which links them to other nodes. Such links are necessary to specify that a certain node must always be in the same vehicle route as another one. Moreover, a link is used as a position constraint. Therefore, a link contains an ID, is associated with two nodes and determines the positioning constraints of these two nodes. In Figure 4.3, node 1 and node 2 share a link of ID 1. The presence of this link, plus the respective positions of these nodes in the solution representation mean that node 1 must be in the same vehicle route as node 2, which must be positioned somewhere after node 1. This is because the vehicle must pick up the request at node 1 before delivering it to node 2.

7	1_1	2_1	5_3	3_2	4_2	6_3	7	9	9	10	10	8	8
7	5_3	6_3	7	9	3_2	1_1	4_2	2_1	9	10	10	8	8

FIGURE 4.3: Solution representation

The TSAM algorithm uses several parameters as follows: $PSMS$ defines the size of the memory for the previous solutions. $RPLI$ defines the number of iterations for operator `consolidation`() in Algorithm 31 to change the request path length. $RPLR$ defines the probability for operator `consolidation`() to choose between operators `PD_stretch`() and `PD_shrink`() in Algorithms 10 and 11. MIT defines the number of iterations before using a previous solution in function `overall`() defined in Algorithm 8. RIT defines the maximum number of restarts for procedure `restart`() in Algorithm 9. $T0$ defines the initial temperature for procedure `simulated_annealing`() in Algorithm 26. δ defines the size of the temperature step for procedure `simulated_annealing`().

Here we describe some sub-functions used by the algorithm. Function `cost`() computes the objective value of a solution as defined in Section 4.2. Function `random_double`() returns a random double value between 0 and 1. Function `random`() picks a random item from the given list. Function `shuffle`() randomly permutes the given list. Function `is_delivery`() returns *true* if the given node is a delivery node. Function `is_hub`()

returns *true* if the given node is a cross-dock. Function `remove()` removes the given request from the given node. If the node has no request left, it will be removed from the given solution route. Function `insert()` inserts the given request path ID in the given node if it is not present. Then insert the node in the specified solution route. However, if the node already exists in the route, the function only adds the request path ID to the node. Function `get_best_indices()` returns all the possible indices from the solution route at which the given node can be inserted. The indices are sorted from best to worst giving the resulting route distance. Function `get_best_solution()` returns the best solution found from all the threads. Function `random_previous_solution()` returns randomly a previously found solution. Function `is_valid()` checks if the given solution is valid according to all the attributes defined in Section 4.2. Function `PD_arrange()` creates a solution where all the P&D pairs are positioned in vehicles of which the start depots are located in the same clusters.

4.3.2 Algorithm functions

This section describes all the functions of the algorithm and gives some pseudo-codes. Readers can refer to the appendix to get more details for the rest of the pseudo-codes.

Algorithm 8: Overall algorithm

```
1  //Input: problem instance;
2  //Output: the best solution $S_b$ found;
3  $S_b \leftarrow$ init_solution();
4  $no\_progress \leftarrow 0$;
5  while Termination criterion not reached do
6      $S \leftarrow$ get_best_solution() //from shared memory;
7      if $no\_progress \,\%\, MIT = 0$ then
8          $S \leftarrow$ random_previous_solution() //from shared memory;
9      $S \leftarrow$ restart($S$);
10     if cost($S$) < cost($S_b$) then
11         $S_b \leftarrow S$;
12         $no\_progress \leftarrow 0$;
13     else
14         $no\_progress \leftarrow no\_progress + 1$;
15 return $S_b$;
```

Algorithm 8 includes the main steps of the meta-heuristic. It is launched in parallel by three different threads which contain a different operator list *op_list*. Thread 1 has an operator list of `PD_interchange()` and `PD_move()`. Thread 2 has an operator list of `PD_consolidate()`. Thread 3 has an operator list of `PD_swap()` and `PD_exchange()`. Each thread memorises all the solutions found so that they can be re-used if there is no improvement for a long time. This memory is shared between all the threads so that a solution can be modified by all the operators. If the memory size is greater than $PSMS$, a random solution is then removed.

In Algorithm 8, function `restart()` in step 9 is iteratively launched with the best solution or a random previous solution until the termination criterion is reached. While step 6 is used to retrieve the best solution found among all the threads, step 8 get a random solution found by all the threads. Algorithm `init_solution()` in step 3 is used to generate an initial solution based on the insertion heuristic of Solomon. The initialisation function does not use the consolidation operators `PD_stretch()` and `PD_shrink()`. Only customer nodes are handled at this stage as solutions involving load exchange with hubs cannot be found at this stage. As a consequence, the feasible region of instances must include at least one solution without consolidation. To start, one first pair of P&D customers is inserted then the insertion positions of each unrouted pair of nodes are considered to minimise the additional distance induced by their insertion in the partially created route. The function continues inserting P&D pairs in the current route until a constraint is violated, in that case, the insertion is tried in the next route. As P&D nodes must be kept together, in case a delivery node could not be inserted, the function first removes the pickup node and then tries another solution route with both nodes.

Algorithm 9: Restart function

```
1  //Input: a current solution $S_c$;
2  //Output: the best solution $S_b$ found;
3  $S \leftarrow$ neighbor_search($S_c$);
4  $S_b \leftarrow S$;
5  while $no\_progress < RIT$ do
6      $S \leftarrow$ simulated_annealing($S$);
7      $S \leftarrow$ reorder_routes($S$) //re-order routes modified by PD_exchange() and
       PD_move();
8      $S \leftarrow$ neighbor_search($S$);
9      if $cost(S) < cost(S_b)$ then
10         $S_b \leftarrow S$;
11         $no\_progress \leftarrow 0$;
12     else
13         $no\_progress \leftarrow no\_progress + 1$;
14 return $S_b$;
```

Algorithm 9 is used to explore the neighbourhood of a solution several times. Procedure `simulated_annealing()` in step 6, is used to escape local minima by potentially accepting worse solutions. At each iteration, the temperature is updated to give a probability to accept a solution created by procedure `random_solution()`. This algorithm allows the global search to make random moves by returning a random solution in the neighbourhood of a given solution. At each iteration, a random operator in *op_list* is applied to the given solution until any solution is found or until the iteration count is greater than *RIT*. To better explore the neighbour solutions, the proposed approach includes procedure `neighbor_search()` which returns a better solution in step 8. At each iteration, a random operator in *op_list* is applied to the current solution until no improvement is made. Function `reorder_routes()` in step 7 is used to re-order the routes modified by operators `PD_exchange()` and `PD_move()`. Each pair of nodes is re-inserted into its route using Solomon's insertion procedure.

4.3.3 Algorithm operators

This section describes all the operators implemented in the algorithm and provides pseudo-codes. As before, readers can refer to the appendix to get more details for the rest of the pseudo-codes. Each operator iteratively performs a move and returns a new solution if it is valid. Otherwise, if the iteration count is greater than the number of customer $|N|$ or $RPLI$ for Algorithm 31, the operator stops and returns no solution. Algorithm 27 alters a node position in a vehicle route by removing and inserting it somewhere else in the same vehicle route. Algorithm 28 moves a pair of P&D nodes from a vehicle route to another one. It tries to select vehicles that are already used. Algorithm 29 replaces an entire vehicle route by another one. All the nodes from two routes are exchanged while checking if the new routes do not contain the depots as intermediate cross-docks. This can be useful when the vehicles have different depots. Algorithm 30 selects four P&D nodes from two different vehicle routes and exchanges them. Algorithm 31 changes a request path length by adding or removing an intermediate cross-dock. At each iteration, a request group is randomly selected. Function `PD_stretch()` or `PD_shrink()` is randomly selected (given the probability $RPLR$) to be applied to the request path ID in the group. If the vehicle or the cross-dock count is equal to 1, there are not enough resources to have load exchange, therefore the operator is not used.

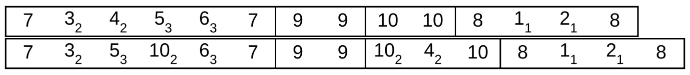


FIGURE 4.4: Stretch operator

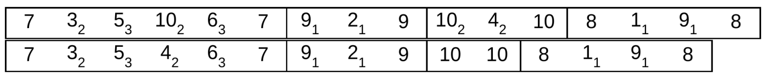


FIGURE 4.5: Shrink operator

Algorithm 10: PD Stretch operator

```
1  //Input: a current solution $S_c$, a request $r$, a set $K$ of vehicles;
2  //Output: a solution $S$ found;
3  $valid \leftarrow false$;
4  while not $valid$ do
5      $S \leftarrow S_c$;
6      $k \leftarrow$ random($K$) //get a random vehicle route where $r$ is present;
7      $v \leftarrow$ random($r, K/k$) //$v$ must be different from $k$ and must not contain $r$;
8      $pickup \leftarrow$ get_pickup($r, S^k$) //get pickup node of request $r$ from route $k$ of
         solution $S$;
9      $delivery \leftarrow$ get_delivery($r, S^k$);
10     $hub_1 \leftarrow$ get_hub($S^k, S^v$) //random cross-dock which is not a depot in the given
         routes;
11     $hub_2 \leftarrow$ clone($hub_1$);
12     remove($delivery, r, S^k$);
13     insert($hub_1, r, S^k$);
14     insert($hub_2, r, S^v$);
15     insert($delivery, r, S^v$);
16     $valid \leftarrow$ is_valid($S$);
17 return $S$;
```

Algorithm 11: PD Shrink operator

```
1  //Input: a solution $S_c$, a request $r$, a set $K$ of vehicles;
2  //Output: a solution $S$ found;
3  $valid \leftarrow false$;
4  while not $valid$ do
5      $S \leftarrow S_c$;
6      $k \leftarrow$ random($K$) //get a random vehicle route where $r$ is present;
7      $pickup \leftarrow$ get_pickup($r, S^k$);
8      $hub_1 \leftarrow$ get_delivery($r, S^k$);
9      if is_hub($hub_1$) $= false$ then
10         continue;
11     $v \leftarrow$ get_route($r, k$) //get the vehicle route linked to route $k$ via request $r$;
12     $hub_2 \leftarrow$ get_pickup($r, S^v$);
13     $delivery \leftarrow$ get_delivery($r, S^v$);
14     remove($hub_1, r, S^k$);
15     remove($hub_2, r, S^v$);
16     remove($delivery, r, S^v$);
17     insert($delivery, r, S^k$);
18     $valid \leftarrow$ is_valid($S$);
19 return $S$;
```

Algorithm 10 inserts intermediate nodes in a request path. Instead of using the direct shipping strategy, a trans-shipment strategy is applied. Therefore the requested container will be exchanged at a cross-dock. When selecting v and $hub1$, vehicles and cross-docks from the same cluster as the pickup and delivery nodes are favoured. The function memorises and reuses the selected vehicle route and intermediate cross-dock so that each request path from a group gets the same route and cross-dock. In Figure 4.4, the solution at the top has been altered and resulted in the solution at the bottom. The request path 2 has been changed and now includes the cross-dock node 10 as an intermediate. This means that vehicle 1 will pick up the container from node 3 to drop it at node 10 so that vehicle 3 could pick it up from its departure node 10 and deliver it at node 4. Algorithm 11, represented in Figure 4.5, does exactly the opposite to Algorithm 10.

4.4 Computational results

4.4.1 Solver configuration

Generator parameters	Values
cross-dock number	{1, 2, 5}
request number	{10, 15, 30, 50}
vehicle number	{2, 4, 5}
vehicle capacity	{50, 100}
request load	5
node time window	0-1000
vehicle time window	0-1000
node service time	1-10
2D coordinates (x,y)	0-100

TABLE 4.1: Generator parameters

As this VRP with P&D and multiple cross-docks is a new problem, there is no data-set available. Therefore, data-sets must be generated randomly. Table 4.1 shows an example of parameters that can be used to generate instances. Node locations and other instance characteristics are randomly generated with an uniform distribution. The time windows are generated as follows. They were first generated as loose constraints and then iteratively tightened until the model became infeasible. In the end, values from the last feasible iteration were saved and used. The ranges for the other parameters are selected in a similar way. There is no predefined unit for the time and the coordinates. Details about the instances are given in Table 4.3.

Algorithm parameters	Values	References
$PSMS$	100	8
$RPLI$	2	31
$RPLR$	0.50	31
RIT	20	9
MIT	5	8
$T0$	50	26
δ	0.75	26

Table 4.2: Solver parameters

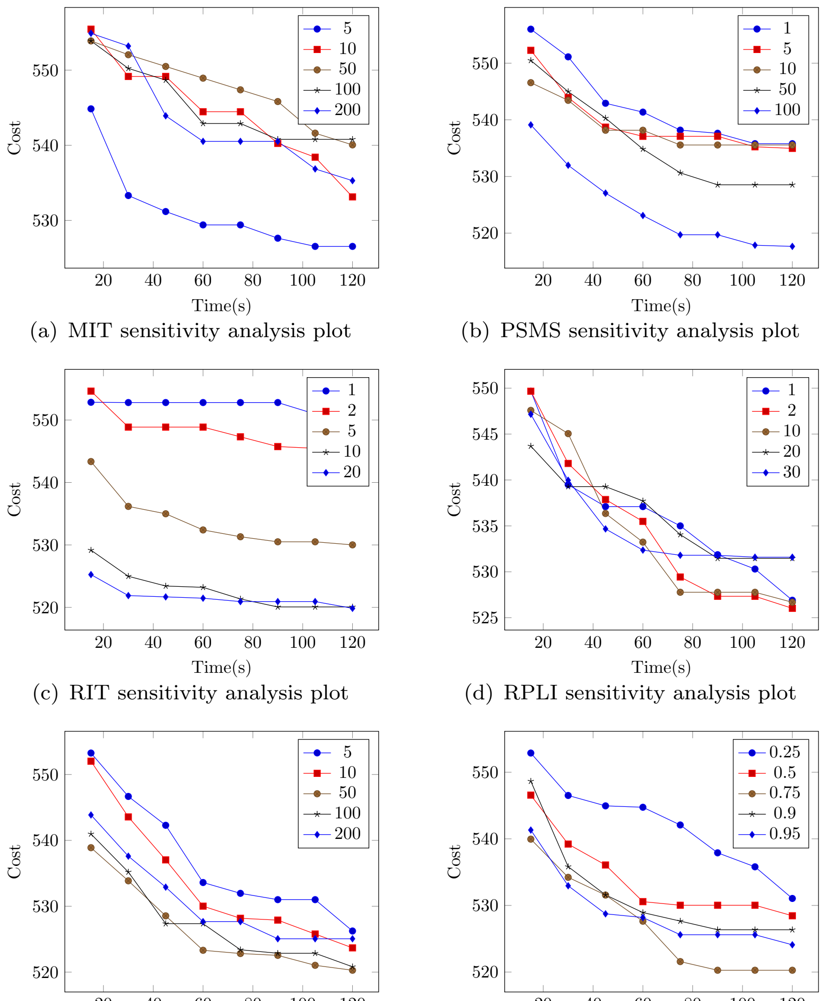
Table 4.2 presents the parameter values used for the experiments. These parameters values have been identified after a sensitivity analysis to allow the algorithm to provide the best performances. Column "references" indicates where the parameters are mentioned in the chapter.

Just like several papers in the literature, a comparative analysis of the proposed algorithm and the proposed mathematical model solved by CPLEX is presented. The model of the CPU used is 'Intel(R) Core(TM) i9-7900X CPU @3.30GHz'. The model is implemented using the CPLEX OPL library and included in a Java framework. The version 12.7 of CPLEX is used with its default configuration and therefore allows parallel computing. Therefore each instance was solved only once by CPLEX. The meta-heuristic algorithm was also implemented and included in a Java framework. The algorithm was launched 30 times for each instance, then the averages were reported.

4.4.2 Parameter tuning

Figure 4.6 shows the convergence of the algorithm with different parameters on the instance $d5q50k2c1r10$. Compared to the one used in Table 4.3, this instance is more challenging as the algorithm tends to get trapped in local minima more easily with bad parameter values. Those results have been used to set the default values of the parameters in Table 4.2. In order to analyse the parameter sensitivity, the following reference values have been used: $PSMS = 20$, $MIT = 5$, $T0 = 40$, $delta = 0.95$, $RIT = 5$, $RPLI = 10$, $RPLR = 0.50$. For each figure, a single parameter is changed to 5 different values and the convergence of the algorithm is reported. The termination criterion is set to 2 minutes of running time.

Overall, parameter values which give the best convergences also give the best results at the end of the search, therefore those values are selected as default. $RPLR$ is set



(a) MIT sensitivity analysis plot

(b) PSMS sensitivity analysis plot

(c) RIT sensitivity analysis plot

(d) RPLI sensitivity analysis plot

(e) T0 sensitivity analysis plot

(f) δ sensitivity analysis plot

FIGURE 4.6: Parameters sensitivity analysis

at 0.50 as this ratio provides a good trade-off to create solutions with consolidation by adding or removing intermediate cross-docks. A bigger ratio would provide solutions with detours while a smaller ratio would prevent consolidations. It should be noted that those convergences might vary from instance to instance as their characteristics can be different. $RPLI$ could be set to a bigger value to allow a longer search in an infeasible region while using consolidations. $PSMS$ and MIT can be useful to escape local minima as they allow the algorithm to research in previous solution neighbourhoods. Besides, $T0$ and δ can be adjusted to better explore those regions.

4.4.3 CPLEX and meta-heuristic performances

	CPLEX				TSAM			
Instance	Objective	Lower bound	Gap	Time (s)	Avg objective	Avg time (s)	Best objective	Best time (s)
d5q50k2c1r10	441.98	441.98	Optimal	107	441.98	0	441.98	0
d5q50k2c1r15		378.46		>10800	558.14	63	557.56	77
d5q50k4c1r30		603.18		>10800	1078.34	101	1010.07	49
d5q50k5c1r50					1535.28	209	1444.22	232
d5q50k2c2r10	589.65	589.65	Optimal	65	589.65	37	589.65	18
d5q50k2c2r15		440.24		>10800	681.04	29	649.43	77
d5q50k4c2r30		801.32		>10800	1224.89	116	1185.26	56
d5q50k5c2r50		973.09		>10800	1739.4	172	1571.92	113
d5q50k5c5r10	447.12	336.28	24.79	10343	407.88	60	398.22	232
d5q50k5c5r15		486.52		>10800	731.77	64	622.19	20
d5q50k5c5r30		592.49		>10800	1022.47	118	900.65	254
d5q50k5c5r50				>10800	1510.3	199	1371.46	237
d5q100k2c1r10	459.73	433.78	5.64	4358	459.73	6	459.73	2
d5q100k2c1r15		441.81		>10800	550.62	22	550.54	50
d5q100k4c1r30		614.89		>10800	1131.31	122	1025.49	79
d5q100k5c1r50				>10800	1622.08	213	1465.62	253
d5q100k2c2r10	605.43	605.43	Optimal	0.5	605.43	0	605.43	0
d5q100k2c2r15	769.08	769.08	Optimal	7	805.68	24	769.09	32
d5q100k4c2r30		745.17		>10800	1066.58	81	1056.99	94
d5q100k5c2r50		1002.97		>10800	1598.83	210	1509.94	278
d5q100k5c5r10	447.12	336.28	24.79	10326	409.8	39	398.22	138
d5q100k5c5r15		485.88		>10800	735.4	67	622.19	111
d5q100k5c5r30		588.83		>10800	1008.49	116	870.56	100
d5q100k5c5r50				>10800	1445.76	220	1325.15	271

Table 4.3: CPLEX and TSAM comparison

Table 4.3 shows the results of CPLEX and the meta-heuristic algorithm for several instances of different request quantities. The instance names give the request demand d, the capacity of the vehicle q, the vehicle number k, the cross-dock number c and the request number r. On top of that, there are vehicles with different start and end depots, requests with special vehicle type requirements and vehicle time windows. As CPLEX is an exact solver and gives optimal solutions when feasible, it needs significantly more time to finish the search. Therefore, the termination criteria for the meta-heuristic and CPLEX are set at 5 min and 3 hours, respectively. Columns *Lower bound* and *Gap* give the last infeasible lower bound and Gap found by CPLEX when it ended. If CPLEX was able to find the optimal solution before the end, columns *Lower bound* would give this optimal solution and column *Gap* would contain the word *optimal*. Column *Time* (s) indicates when the solver found the last feasible solution. If the value is > 10800, it indicates that the search needed more than 3 hours to converge. Columns *Avg objective* and *Avg time* (s) give the average of the results over 30 runs while columns *Best objective* and *Best time* (s) give the details of the best solution.

From Table 4.3 it can be seen that the proposed algorithm outperforms CPLEX. On the one hand, on instances containing more than 30 customers, CPLEX couldn't provide any feasible solutions but gave the lower bound. However, for some instances containing 50 requests and therefore 100 customers, CPLEX could not provide any lower bound therefore, those rows are left blank. It can be noticed that the difficulty can also vary between two instances containing similar characteristics. This is due to the node locations. On the other hand, the proposed algorithm is not only able to match the results of CPLEX for small instances but is also able to find solutions for large instances. As expected, on average, when the number of requests, vehicles, cross-docks increases or the vehicle capacity decreases, the instance difficulty increases.

4.4.4 Consolidation performances

	With				Without			
Instance	Distance	Vehicle	Consolidation	Time (s)	Distance	Vehicle	Consolidation	Time (s)
d1q2-5k5c6r2	253.14	3	4	0	284.7	1	0	0
d1q5k5c6r5	267.76	3	10	1	291.71	1	0	0
d1q10k6c4r10	280.74	6	12	3	292.99	1	0	1

TABLE 4.4: Consolidation feature results

To study the difference brought by using consolidation centres, we tested TSAM on some special instances with and without the consolidation feature. Table 4.4 shows the best results of the meta-heuristic algorithm on several instances with different numbers of requests. The instances are clustered so that they represent cities where the deliveries have to be made by special vehicles. Moreover, those cities are linked by highways that can only be used by a specific type of vehicle. Vehicles of this type are not forced to return to the start depot. As a result, using this vehicle would improve the solution cost but require consolidations with the other vehicles. Columns *With* and *Without* give the results of the algorithm with and without the consolidation function, respectively. The distance, the number of vehicles used and the number of consolidations are reported. From Table 4.4 it can be seen that the consolidation feature is of great importance to find better solutions when cross-docking is involved. Without the consolidation, only one vehicle is used to pick up and deliver all the requests, then it returns to the start depot. However, with the consolidation, several vehicles are used and exchange their loads. This decreases the total distance driven as a specific vehicle is used on the highway and does not need to return to the start depot.

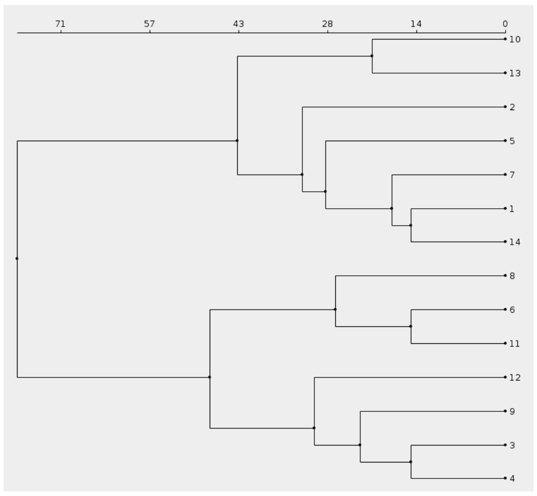


FIGURE 4.7: Dendrogram of an instance with clustered nodes

Figure 4.7 represents the hierarchical clustering of the instance shown in Figure 4.1. In this dendrogram using an average linkage strategy, the heights reflect the distance between the clusters. In this case, the dendrogram shows that there are two main clusters grouping customers. This clustering method helps the algorithm to group requests that can be consolidated together for better solutions cost. In this case, since pickup nodes 5 and 7 belong to a different cluster from delivery nodes 6 and 8, the algorithm will try to consolidate their containers.

4.4.5 Benchmark performances

	Best known		TSAM					
Instance	Distance	Vehicle	Avg distance	Avg time (s)	Avg iteration	Best distance	Best vehicle	Best time (s)
lc101s	828.94	10	828.94	0	0	828.94	10	0
lr101s	1650.8	19	1650.8	66	493	1650.8	19	17
lrc101s	1708.8	14	1713.15	114	849	1703.21	15	252
lc1_2_1s	2704.57	20	2704.57	25	46	2704.57	20	18
lr1_2_1s	4819.12	20	5106.47	845	2129	4873.54	21	884
lrc1_2_1	3606.06	19	3765.98	859	3188	3606.06	19	872
lc109q	1000.60	9	839.25	56	695	827.82	10	88
lr201	1253.23	4	1339.09	320	5218	1286.83	5	512
lrc_1_2_5s	3715.81	16	4313.25	1043	4950	4044.81	20	1309

TABLE 4.5: Li&Lim benchmark

Table 4.5 shows the results of the proposed algorithm on some instances of Li&Lim's benchmark SINTEF to evaluate the performance of TSAM on the general PDP. It should be noted that the results in SINTEF are continuously updated with the best-known results from the state-of-the-art algorithms in the literature. To test all the different configurations, instances from each group (clustered, random, and random-clustered nodes) for 100 customers and 200 customers were selected. Columns *Best known* give the best-known solution details (taken from SINTEF) on the selected instances while columns *TSAM* give the results of the proposed algorithm. Columns *Avg distance* and *Avg time* (*s*) give the average of the results over 30 runs while columns *Best distance*, *Best vehicle* and *Best time* (*s*) give the details of the best solution. Those results include the distance driven by all the vehicles, the number of vehicles used and the time at which the solution was found. The termination criterion for the meta-heuristic is set at 25 min.

From Table 4.5 it can be seen that the proposed algorithm is able to find some of the best-known solutions in a reasonable amount of time. On top of that, the interesting result of instance *lc*109 shows that TSAM can provide solutions which improve the distance compared to the best-known ones while using more vehicles. However, there are instances for which TSAM could only find near-optimal solutions. This can be explained by the objective being different, this type of instance is better solved by approaches which focus on reducing the number of vehicles used. Our method, on the other hand, needs to try several vehicles to find consolidation opportunities as shown in Table 4.4 while also trying different combinations of vehicles due to the multi-depot constraint.

4.5 Conclusion

Cross-docking is known to be one of the most effective strategies in logistics systems to improve the flow of products in supply chains. Therefore, in this chapter, a rich VRP with pickup and delivery and multiple cross-docks was considered. This problem tackles several attributes of the problem to cope with realistic constraints - capacitated, heterogeneous, mixed pickup and delivery, multiple depots, open route, different start/end depots, time windows and site-dependent. An MILP model is designed and solved by CPLEX. Given the high complexity of the considered problem, especially in large scale, a meta-heuristic is also developed. Experiments are conducted using a wide range of generated data-sets that reflect different real-life constraints. Those constraints can force requests to be consolidated in cross-docks during deliveries. It is shown that CPLEX cannot find good solutions in a reasonable amount of time for the biggest instances. However, the proposed algorithm not only outperforms CPLEX in all the benchmark instances but is also able to scale up. Moreover, the proposed algorithm can also match some of the best-known results by state-of-the-art methods on the benchmark of Li&Lim on large instances.

The main contribution is twofold. Firstly, we presented a new rich VRP that has not been addressed before. An MILP model is proposed to tackle this problem. Secondly, a multi-threaded simulated annealing algorithm with memory including new operators is introduced to handle real-world size instances.

There are a few directions for further research. Since the problem is new, some standard benchmarks could be created for future comparison purposes. Also, the proposed algorithm could be integrated into a unified solution framework for multi-attribute. Finally, one could extend the problem at hand by considering other constraints such as working hours, rest times for drivers or even multi-objective and dynamism.

Chapter 5

A rich multi-objective and dynamic VRP with uncertainty

5.1 Introduction

Because of recent advances in information and communication technologies such as GPS IoT, and traffic flow sensors, vehicle fleets can now be monitored in real-time. As a result, interest from researchers increases to address the dynamic or stochastic version of the problem. In this context, the dynamism characteristic implies that the information needed to solve the problem is being revealed while searching a solution. This real-time constraint forces the solver to re-optimise the problem solutions several times. Whereas in a stochastic problem, some elements are known only by their probability distribution and become certain when the vehicle visits the customers. Moreover, modern logistics receives increasing attention for planning and scheduling operations of transport systems in a more efficient and environmentally sustainable way. To respond to those new concerns, the new concept of PI emerged to interconnect different transportation networks while allowing collaborations to deliver containers. By doing so, transportation loads are better shared, and therefore, cost and waste are reduced.

As an extension of the VRP, the pickup and delivery problem is also an important combinatorial optimisation problem in the transportation industry. Therefore, this chapter considers the previous work from Chapter 4 in which we studied a rich VRP with pickup and delivery while incorporating constraints for the PI. We extend the problem with realistic constraints such as dynamism, multi-objective and uncertainty. Our motivation in this context is the collaboration with a local company which handles a fleet of bikes to deliver goods across the city. The company provides us with historical data which are then used to anticipate future requests. Those request predictions are given probabilities and included in the model as stochastic information. Delivery via cross-docking is also of great interest, therefore, this research aims at solving a rich VRP with pickup and delivery including multi-objective, dynamism and uncertainty. The extension is also intended to illustrate that the proposed approach can be applied to a wide range of real-world problems.

5.2 Problem description and formulation

5.2.1 Context

This section reintroduces the problem described in Chapter 4 - a rich VRP while proposing an extension of the model. The following list of properties are handled: capacitated, heterogeneous, mixed pickup and delivery, different request types, open route, different start/end depots, multiple cross-docks, multiple depots, customer time windows, vehicle time windows (vehicle availability), site-dependent, dynamism, multi-objective, uncertainty. The problem at hand is based on a case study from a distribution company in Liverpool. One of the services offered by this company is to pick up goods from customer locations and deliver them to other customer locations using bikes. As cycling uphill with a cargo-bike might represent a considerable effort, altitude gradients have to be considered. The company can use intermediate facilities such as cross-docks to allow load exchanges between bikes. Bikes are initially placed at depots, are homogeneous, and have a given capacity which cannot be exceeded. The logistics company operates with multiple cross-dock facilities O and uses a set of bikes K where each bike k has a capacity Q_k. Each bike must start and end its route at its assigned cross-docks O_s^k and O_e^k. The company is responsible for satisfying a set of customer requests R. Each request is related to two nodes - customers or cross-docks ($n_1, n_2 \in N$); the first denotes the pickup point, and the second one denotes the delivery point. All the requests are associated with their demands $q_r > 0$. For a given request r, its pickup and delivery locations are represented by the set $H_r = \{H_r^p, H_r^d\}$. Each request can pass through multiple different cross-docks to be consolidated with others. Customers are expected to be served the same day and can have time windows restricting the service. Customers are allowed to call in orders at any time during the day. Therefore, a rolling-horizon procedure is followed to handle the dynamic characteristic of the system. This approach consists in splitting the planning horizon in a sequence of periods of equal length. The dynamic problem is therefore modelled as a sequence of static sub-problems aiming at optimising the problem using only the information known in the current period. For each period, a set of new customers C^+ is revealed and incorporated into the model with previous customers C^- such as the set of current customers is C. The problem is defined on a graph including a set of nodes N such as $N = C \cup O = C^- + C^+ \cup O$. For each pair of nodes i and j, there exists an arc (i, j) of driving time d_{ij}. The aim is to design a set of routes with the lowest cost where each route is a solution for a certain bike.

5.2.2 Mixed-integer linear programming formulation

One of the objectives of the PI is to be more flexible as to where requests can originate and terminate. Therefore, constraint (5.1) allows requests to not only be from customers to customers (customer requests) but also from customers to cross-docks (mixed requests), cross-docks to customers (mixed requests) or cross-docks to cross-docks (cross-dock requests). A request entering a cross-dock o still has to leave this cross-dock if the delivery location is not this cross-dock. Therefore just like customers, cross-docks are now also given time windows. Constraints (5.2) and (5.3) enforces that each node j is available for pickup or delivery between times A_j and B_j. Variable x_{ij}^k is equal to 1 if and only if bike k travels from node i to node j. Variable y_{rij}^k is equal to 1 if customer request r is transported using bike k on its route from node i to node j, otherwise, y_{rij}^k equals 0.

$$\sum_{k \in K} \sum_{i \in N} y_{rio}^k = \sum_{k \in K} \sum_{i \in N} y_{roi}^k \ \forall r \in R,\ \forall o \in O \backslash H_r \tag{5.1}$$

$$u_j^k \leq B_j \ \forall k \in K,\ \forall j \in N \tag{5.2}$$

$$v_i^k + M \left(1 - \sum_{j \in N \backslash i} x_{ij}^k \right) \geq A_i + s_i \ \forall k \in K,\ \forall i \in N \tag{5.3}$$

In this thesis, the DVRP is tackled by re-optimising solution routes at pre-determined time intervals throughout the day. This allows a formulation as a p-interval problem, where p is the number of periods where re-planning is allowed. By splitting the problem into periods, we allow the algorithm to accumulate events before responding. As opposed to the static routing problem, in the dynamic case, the time dimension is essential. The dispatcher must know the position of all bikes at any given point in time and particularly between each period. This allows the decision making to change a previous solution according to the nodes already visited by the bikes. Therefore, constraints (5.4) and (5.5) forbid any decision taken before time t to be changed unless the bike has not yet visited the node related to the decision. Given $o = O_s^k$, if x_{oo}^k equals to 1 then the bike k stays in its start cross-dock and is not used for at least the current period. Variable x_{ij}^{k-} is equal to 1 if bike $k \in K$ was supposed to travel from location i to j according to the solution of the previous period.

$$v_i^{k-} < t \wedge x_{oo}^{k-} = 0 \rightarrow x_{ij}^k = x_{ij}^{k-} \ \forall i, j \in N^{k-}, \ \forall k \in K \mid o = O_s^k \tag{5.4}$$

$$v_i^{k-} < t \wedge x_{oo}^{k-} = 0 \rightarrow y_{rij}^k = y_{rij}^{k-} \ \forall r \in R^-, \ \forall i, j \in N^-, \ \forall k \in K \mid o = O_s^k \tag{5.5}$$

A request prediction based on historical data is introduced to handle dynamic customer requests. Since customers are mostly charities, restaurants, or shops, the requests are regular enough to anticipate requests given a weekday. It is then reasonable to predict the new dynamic requests according to the historical data. With accurate prediction, the proposed algorithm is able to plan a good predictive route that can be easily tuned to fit the real requests over time. Future consolidations can be anticipated by sending a bike in a certain area to wait for requests. Each request prediction is given a probability to appear in a certain period. The apparition probability is noted $\mathcal{P}(r)$. Then the system filters the predictions with probabilities lower than a threshold and returns the requests. The number of requests for each period is limited.

For the problem considered in this chapter, the uncertainty is defined by the likelihood that the request would appear or not. All the other request attributes such as pickup and delivery locations, demands, TWs are deterministic. Given the uncertainty at hand, recourse actions can be performed as bikes might not be able to satisfy all the customers within their TWs. In this case, recourses are handled by allowing TWs to be violated as a soft constraint. Therefore, delays and early arrivals are now an objective to be minimised by considering equations (5.6) and (5.7) for the delay and early time of bike k at the node i.

$$l_i^k = \begin{cases} 0, & if\ u_i^k \leq B_i \\ u_i^k - B_i & otherwise \end{cases} \ \forall i \in N, \ \forall k \in K \tag{5.6}$$

$$e_i^k = \begin{cases} 0, & if\ u_i^k \geq A_i \\ A_i - u_i^k & otherwise \end{cases} \quad \forall i \in N,\ \forall k \in K \tag{5.7}$$

The objectives considered in this chapter can be summarised as the minimisation of the vector function F defined in (5.12). F is subject to the constraints in Chapter 4 and the modified ones in this section.

Equation (5.8) computes the total travel distance (f_1). Equation (5.9) computes the number of bikes used (f_2). Equation (5.10) computes the total delays and early times of all the bikes (f_3). Equation (5.11) computes the elevation difficulty for the bike tours (f_4) with θ_{ij} as the slope of arc (ij).

All these objectives have to be considered simultaneously to satisfy the company's needs. As shown in Chapter 3, distance minimising might increase the cyclist effort due to slopes. Moreover, minimising the distance or the travel time does not necessarily assure that all TWs will be met as customers might have TWs scattered throughout the day. Finally, as discussed in Section 4.2, due to the cross-docking nature of this problem, fewer vehicles does not always guarantee a better solution.

$$f_1 = \sum_{k \in K} \sum_{i \in N} \sum_{j \in N} x_{ij}^k * d_{ij} \tag{5.8}$$

$$f_2 = \sum_{k \in K} \left(1 - x_{ii}^k\right) \mid i = O^k \tag{5.9}$$

$$f_3 = \sum_{k \in K} \left(\sum_{i \in N} l_i^k + \sum_{i \in N} e_i^k \right) \tag{5.10}$$

$$f_4 = \sum_{k \in K} \frac{\sum_{i,j \in N} \max(x_{ij}^k \theta_{ij}, 0)}{\sum_{i,j \in N} x_{ij}^k} \tag{5.11}$$

$$F = \{f_1, f_2, f_3, f_4\} \tag{5.12}$$

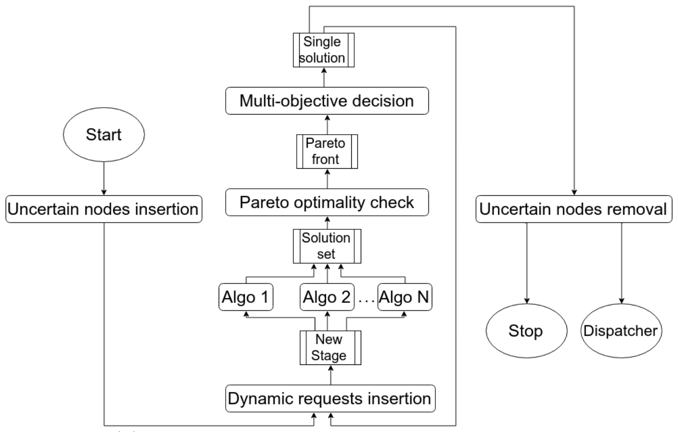
5.3 Meta-heuristics

The mathematical model of Section 5.2 cannot be handled by traditional solvers, therefore a meta-heuristic approach is considered - a Genetic Algorithm (GA). In a GA, a population of individuals evolves generations after generations toward better solutions which are assessed for their fitness. For the VRP, each candidate solution is represented by an individual chromosome which can be mutated and altered. GAs have been largely used to solve VRPs, for example, papers by Prins (2004), Kuo and Zulvia (2017), Wang and Wu (2017) showed the quality of GAs applied to the VRP. When it comes to adaptability and flexibility to handle a maximum of constraints as in Vidal et al. (2014), GAs are often used to tackle those problems.

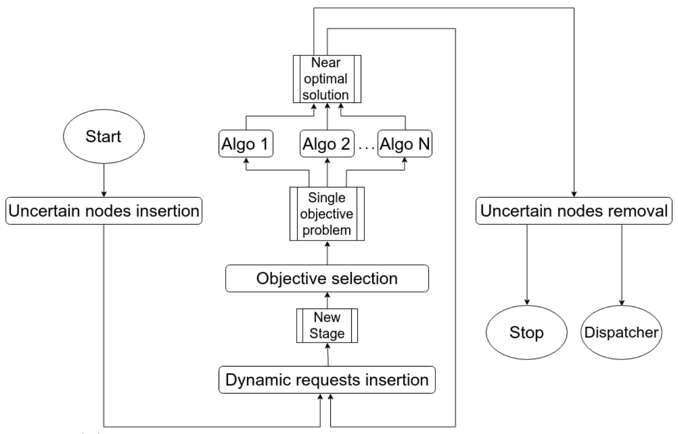
Moreover, only a few papers proposed to solve the VRP with a single cross-dock using GAs, for instance, Touihri et al. (2017). The others used other methods such as Tabu Search, Swarm intelligence etc. But very few tried to solve this problem with multiple cross-docking facilities (Maknoon and Laporte, 2017), and even fewer solved it with many additional attributes (Dondo et al., 2011). Evolving a population of solutions including multiple cross-docks is a challenging task. Spatial and time synchronisations of the vehicles have to be constantly checked in order to keep a population of only feasible solutions. This thesis fills this gap by solving this problem with a GA.

In a GA, a population of solutions called chromosomes is evolving through several generations. This evolution process includes parents mating and mutations to beget offsprings. The fitness of each offspring is computed to control the evolution of the population and discard the worst chromosomes. The idea is to steer the population to search for the optimal solutions to a given problem instance. As generations occur, offsprings are filtered while the best ones get stored. Classical GA crossovers are not applicable to our problem due to the multiple cross-docks constraint. For instance, Shi et al. (2009) proposed a genetic algorithm and implemented a pheromone-based crossover operator for the VRP with pickup and delivery. It differs from the general VRP with pickup and delivery as the commodity provided by pickup customers can be trans-shipped to any delivery customers. Their pheromone-based crossover creates offsprings by selecting neighbouring customers from the parents. However, in a cross-docking problem, their crossover selection feature could not work due to cross-dock nodes appearing multiple times in a single chromosome. Hence the need for a new crossover that can handle multiple cross-docks.

5.3.1 Architecture



(a) Architecture considering a Pareto method



(b) Architecture considering a Lexicographic method

FIGURE 5.1: Examples of framework architectures to handle the problem

Our framework can work in two different modes, the Pareto mode and the Lexicographic mode. Its architecture is illustrated in Figure 5.1. For both modes, one main algorithm is selected from N algorithms to solve the instance. In this chapter, we developed and selected algorithm `genetic_algorithm()` defined in Algorithm 12 to be launched by three different threads. The process is defined as follows: The predicted nodes with the static nodes are given to the framework as input to provide a solution. Before sharing with the dispatcher, predicted request nodes that have not yet occurred are removed from the solution. Figure 5.1(a), represents the Pareto method in which a Pareto front of trade-off optimal solutions is found at each interval. The dispatcher must then choose a single solution to schedule vehicles. Three procedures are used to handle the multi-objective feature in this mode. Procedure `pareto_dominance()` in Algorithm

32 compares two solutions to know if the first one C_1 dominates or not the second one C_2. Procedure `pareto_front()` in Algorithm 33 returns a set of solutions which are not dominated by any other solutions - the Pareto front. Procedure `pareto_ranks()` in Algorithm 34 ranks the solutions in a given set by iteratively removing Pareto fronts. Figure 5.1(b), represents the Lexicographic method used as default in our framework. In the classical lexicographic method, objectives are ranked in the order of importance. A sequence of single-objective linear problems is solved according to the objective ranks. Once a solution to the problem has been found, a new constraint is added to the next subsequent linear problem to limit the value of the current objective. The next linear problem is then solved considering the next objective in the rank. We propose a new version of the lexicographic method which differs from the above as the objectives are given probabilities instead of ranks. Single-objective linear problems are still solved sequentially but no constraint is added. Only one linear problem is solved during each period with a selected objective according to its probability. For instance, given the objective priorities as follows: TW delays, distance, elevation and vehicle. A possible probability distribution for the objectives could be 0.4 0.3 0.2 0.1. Therefore, at each interval during the time horizon, a different objective is selected in order to get a single solution from the Pareto set that best minimises the selected objective. As new requests are ordered in, predicted nodes are being replaced accordingly if they match. However, all the predicted nodes are removed from the solution given to the dispatcher. The choice of such a method to handle dynamism is to try and test other features instead of well-known scalarization methods. The idea is motivated by the hypothesis that the crossover and local search operators can better focus on a single objective at a time. As the time horizon is highly dynamic, the selected objective would regularly change to allow those operators to produce solutions of different characteristics.

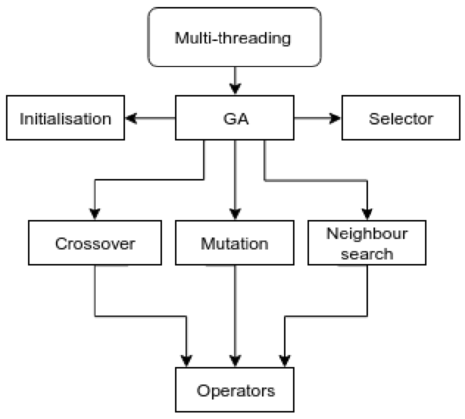


FIGURE 5.2: Algorithm methods

The main features of the algorithm are separated from the problem specificities. Operators are generic ones, this gives to the framework the flexibility to use other meta-heuristics. Moreover, VRP attributes are handled by components which can be easily added or removed to modify the problem. As shown in Figure 5.2, just like TSAM, it is launched in parallel by three different threads which contain different operator lists *op_list*. Thread 1 has an operator list of `PD_interchange`() and `PD_move`(). Thread 2 has an operator list of `PD_consolidate`(). Thread 3 has an operator list of `PD_swap`() and `PD_exchange`(). Each thread memorises all the solutions found so that they can be re-used at any time. This memory is shared between all the threads so that a solution can be modified by all the operators.

7	1_1	2_1	5_3	3_2	4_2	6_3	7	9	9	10	10	8	8
7	5_3	6_3	7	9	3_2	1_1	4_2	2_1	9	10	10	8	8

FIGURE 5.3: Chromosome representation

Each solution and its vehicle routes have a flexible size which depends on the number of visited nodes. In Figure 5.3 the vehicle tours/routes are delimited by the departure and arrival cross-docks/depots which are the numbers without any subscript. The ones with subscripts can be the customer or cross-dock nodes. These subscripts are the links between the nodes which represent the request travel paths. In Figure 5.3 such links are shown with the subscript numbers. Each node has a list of request path IDs which links them to other nodes. Such links are necessary to specify that a certain node must always be in the same vehicle route as another one. Moreover, a link is used as a position constraint. Therefore a link contains an ID, is associated to two nodes and determines the positioning constraints of these two nodes. In Figure 5.3, node 1 and node 2 share a link of ID 1. The presence of this link, plus the respective positions of these nodes in the solution representation mean that node 1 must be in the same vehicle route as node 2, which must be positioned somewhere after node 1. This is because the vehicle must pick up the request at node 1 before delivering it to node 2.

5.3.2 GA overall

The main steps of the proposed GA are defined in Algorithm 12. To initialise the procedure, in step 3, function `init_solution`() defined in Algorithm 23 creates a population of chromosomes of which the size is given by parameter *rps*. Each P&D pair is sequentially inserted into the solution routes. The given nodes to be inserted are sorted by function `sort_nodes`() to prioritise urgent requests over the others using their time windows. In case a delivery node could not be inserted, the function first removes the pickup node and then tries another solution route with both P&D nodes. Procedure

Algorithm 12: Genetic Algorithm

```
//**Input:** set $C$ of P&D locations, set $K$ of vehicles;
//**Output:** set $S$ of best chromosomes;
$P \leftarrow$ init_solution();
while Termination criterion not reached do
    add($P, c1, c2$) //add two chromosomes from other threads' memories;
    roulette_wheel($P$) //select and keep good chromosomes;
    $P \leftarrow$ crossover($P$);
    $P \leftarrow$ mutate($P$) //select and apply one mutation operator;
    $P \leftarrow$ neighbor_search($P$);
```

`roulette_wheel_selector()` defined in Algorithm 13 in step 6 must not only keep the population size as defined in the parameters but also select good individuals for coupling based on their fitnesses and selective pressure values. As shown in Sections 2.3 and 2.4, the VRP can have multiple attributes that must be taken into consideration during the solving procedure. Attributes are handled by algorithms in two different ways. (1) Operators consider the structure of the solutions they produce. For instance, precedences between nodes are respected while performing any move. "mixed pickup and delivery", "different start/end depots", "multiple cross-docks", "multiple depots" are included in that category. (2) Operators check the validity of the solutions afterwards and re-perform their moves or discard infeasible solutions. "capacitated", "customer time windows" "vehicle time windows (vehicle availability)" "site-dependent" are included in that category.

Algorithm 13: Roulette Wheel Selector function

```
//**Input:** set $P$ of chromosomes, $rps$ running Population Size;
//**Output:** population of chromosome $S$;
filter($P$) //remove all the duplicates;
$sp \leftarrow$ get_selective_pressure($P$);
while $|P| < rps$ do
    $c1 \leftarrow$ get_chromosome($sp$)//get a chromosome given a selective pressure;
    $c2 \leftarrow$ get_chromosome($sp$);
    add($S, c1, c2$);
return $S$;
```

5.3.3 GA operators

All operators take advantage of the links present in any chromosome. Each operator must check the links' integrity to validate their moves or cancel the operation so that the resulting solutions are always feasible. Operators `PD_shrink()` and `PD_stretch()` are exceptions as they can produce infeasible solutions when used alone. However, they are

Algorithm 14: Selective pressure function

```
1  //Input: set $P$ of chromosomes;
2  //Output: selective pressure $sp$ of population;
3  $f \leftarrow$ get_fitness($P$) //according to current objective;
4  $sp \leftarrow \varnothing$;
5  foreach $c$ in $P$ do
6      $fitness \leftarrow 1/f[c]$;
7      $sum\_fitness \leftarrow sum\_fitness + fitness$;
8      add($sp, fitness$);
9  $index \leftarrow 0$;
10 $v \leftarrow \varnothing$;
11 foreach $c$ in $P$ do
12     $pressure \leftarrow pressure + sp[index]/sum\_fitness$;
13     $sp[index] \leftarrow pressure$;
14     $index \leftarrow index + 1$;
15 return $sp$;
```

only used via operator PD_consolidation() to iteratively call them multiple times. This is to better access local minima by allowing the operator to shrink and stretch solutions until a feasible one is found.

Algorithm 15: Crossover overall algorithm

```
1  //Input: set $P$ of chromosomes, crossover probability $cp$;
2  //Output: population $P'$ of offsprings;
3  $P' \leftarrow \varnothing$;
4  while $P \neq \varnothing$ do
5      $parent1 \leftarrow$ remove($P$);
6      $parent2 \leftarrow$ remove($P$);
7      if $random\_double() < cp$ then
8          $links1, links2 \leftarrow$ distribute_links($links1, links2$);
9          $offspring1 \leftarrow$ CX_mate($\{parent1, parent2\}, \{links1, links2\}$);
10         $offspring2 \leftarrow$ CX_mate($\{parent1, parent2\}, \{links2, links1\}$);
11         if $offspring1 \neq \varnothing$ then
12             add($P', offspring1$);
13         else
14             add($P', parent1$);
15         if $offspring2 \neq \varnothing$ then
16             add($P', offspring2$);
17         else
18             add($P', parent2$);
19     else
20         add($P', parent1$);
21         add($P', parent2$);
22 return $P'$;
```

The crossover is described in Algorithm 15 while its process is represented in Figures 5.4, 5.5 and 5.6. It follows a probability given by parameter *cp*. Step 11 gives a list of best indices where the node must be inserted according to the current objective. In the example of Figure 5.4, there are 4 different requests and therefore 4 different links in each chromosome. Instead of operating at the node level as traditional crossovers in the literature, this crossover operates on the links. Step 8 creates two lists of links randomly distributed from the two selected parents. With Figure 5.4 as parents, Figures 5.5 and 5.6 can be considered as their offpsrings. Offspring 1 gets request links 2 and 4 from parent 1 and gets request links 1 and 3 from parent 2. Offspring 2 gets request links 1 and 3 from parent 1 and gets request links 2 and 4 from parent 2. Table 5.1 shows the request paths in the parent chromosomes and the resulting offsprings' ones.

1 8 5 6 1	2 9 7 2	2 10 11 3	3 12 1 4
1 7 10 8 1	2 9 1 2	2 11 5 3	3 6 12 4

FIGURE 5.4: Crossover parents

1 8 10 1	2 9 1 7 2	2 11 5 3	3 12 6 1 4

FIGURE 5.5: Crossover offspring 1

1 5 7 6 8 1	2 9 2	2 10 11 3	3 12 4

FIGURE 5.6: Crossover offspring 2

Algorithm 16: Crossover mating algorithm

```
1  //Input: list $P$ of parent chromosomes, list $L$ of request links, vehicles set $K$;
2  //Output: chromosome $C$;
3  $C \leftarrow \varnothing$;
4  for $i$ from 1 to 2 do
5      $parent \leftarrow P[i]$;
6      $links \leftarrow L[i]$;
7      foreach $k$ in $K$ do
8          foreach $node$ in $parent^k$ do
9              foreach $link$ in $node$ do
10                 if $link \in links$ then
11                     $indices \leftarrow$ get_best_indices($C^k, node$);
12                     foreach $j$ in $indices$ do
13                         if insert($C^k, node, j$) $= true$ then
14                             $inserted \leftarrow true$;
15                             break;
16                     if $inserted = false$ then
17                         return $false$;
18 return $C$;
```

Algorithm 17: Mutation algorithm

```
1  //Input: population $P$, mutation probability $mp$;
2  //Output: population $P'$;
3  while $P \neq \varnothing$ do
4      $c \leftarrow$ remove($P$) //remove one chromosome from population $P$;
5      if random_double() $< mp$ then
6          $c' \leftarrow$ PD_operator($c$) //apply a random operator from op_list;
7          if $c' \neq \varnothing$ then
8              add($P', c'$);
9      add($P', c$);
10 return $P'$;
```

	Request IDs	Request paths
Parent 1	Request 1	5→6
	Request 2	7→2→3→1→8
	Request 3	9→2→10
	Request 4	11→3→12
Parent 2	Request 1	5→3→6
	Request 2	7→8
	Request 3	9→1→10
	Request 4	11→3→12
Child 1	Request 1	5→3→6
	Request 2	7→2→3→1→8
	Request 3	9→1→10
	Request 4	11→3→12
Child 2	Request 1	5→6
	Request 2	7→8
	Request 3	9→2→10
	Request 4	11→3→12

TABLE 5.1: Chromosome request paths

In addition to the crossover, mutation operators are implemented to make random moves following a probability given by parameter mp. Procedure `mutation()` in Algorithm 17 depicts the function handling the call of operators for the mutations.

5.3.4 Request anticipation and dynamism

To handle historical data, the initial idea was to record all the previous request details such as the node locations, demand, time windows etc. As the historical data are processed, each request is given a probability to appear in a certain period of the day. If the probability of a request is high enough (over a threshold), it will be considered an "anticipated request" and will be included in the problem instance. At the end of each dynamism period, the list of anticipated requests will be updated based on updated data when time goes by. Those that either did not happen or will become less likely to happen will be removed from the list. Time windows of the remaining anticipated nodes stay the same to allow vehicles to wait for probable opportunities. This provides a pseudo optimal solution, but more robust. Since the logistics company handles requests that are regular enough in a given weekday, this should method work.

However, since we are not allowed to share the real data of the case-study company, we need to generate realistic data based on the actual operational plan of the company to test this feature. When instances are generated, a "forecast validity rate" parameter is provided to create uncertain requests with a probability. Their probabilities are randomly generated given the ranges of parameters "Valid node probability" and "Invalid node probability".

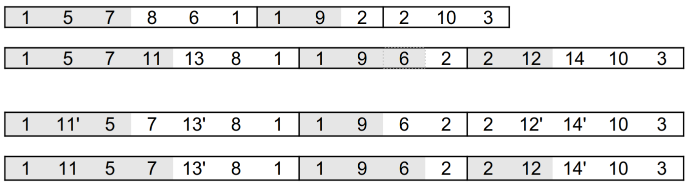


FIGURE 5.7: Best chromosomes over time

Figure 5.7 shows the dynamic process of nodes without node anticipation in the first two chromosomes and with node anticipation in the last two. During the planning horizon, 5 requests will be called in as follows: request 1 from node 5 to node 6; request 2 from node 7 to node 8; request 3 from node 9 to node 10; request 4 from node 11 to node 12; request 5 from node 13 to node 14.

The first and third chromosomes represent the solution planning at a given period while the second and fourth are for the next period. In Figure 5.7, as vehicles progress through their journey, visited nodes appear with a grey background. Therefore, the nodes with a grey background cannot be changed in the future periods. The first chromosome contains 3 requests in which the delivery process has already started as shown by the grey background. Consolidations occur during this delivery, for example, request 3 is first picked by vehicle 2 and then dropped by vehicle 3. Without the node anticipation, when additional requests are registered, they must be added to the end of the vehicle route as the grey nodes can not be changed. However, with node anticipation, future requests are processed and added since the beginning of the planning horizon, if they are well predicted. Hence, request 4 being handled at the beginning of the first trip in the third chromosome. Uncertain nodes are represented by numbers with quotes. Over time, those uncertain nodes are replaced by the new request nodes if they occur. With this feature enabled, vehicles can predict requests and wait for them in strategic locations instead of going back to the depot and not being able to take requests anymore. They could also forecast consolidation opportunities and deviate their routes to take advantage of such situations. Besides, as time passes, vehicle routes get fixed which make previous choices even more important, hence the usefulness of this feature. This is shown by the fact that handling request 4 at the beginning will save some cost compared to the solution found in the second chromosome.

5.4 Computational results

5.4.1 Data

Generator parameters	Values
cross-dock number	{1, 2, 4, 5, 8}
request number	{10, 15, 30, 50, 100}
vehicle number	{2, 4, 5}
vehicle capacity	{50, 100}
request load	5
node time window	0-10000
vehicle time window	0-10000
node service time	10
3D coordinates (x,y,z)	0-100
cluster number	4
cluster radius	10
period number	1-20
period duration	100
prediction request number	10
forecast validity rate	1
valid node probability	0.7-1
invalid node probability	0-0.4

Table 5.2: Generator parameters

As this work extends Chapter 4, finding benchmarks in the literature on this specific problem is still an issue. The VRP model with multiple cross-docks is a new problem and there is no data-set available, and we are not allowed to publish the industry data from the partner. Therefore, new data-sets must be generated. Accordingly, the dynamic requests are generated and attributed to different periods with a uniform distribution. Table 5.2 shows an example of parameters that can be used to generate instances. Distances are euclidean but there is no predefined unit for the time and coordinates.

Algorithm 18: Clustering function

```
1  //Input: cluster count $g$, set of nodes $N$, the graph coordinates $(x, y)$, the size of
     clusters $(c_x, c_y)$;
2  //Output: set of nodes $N$ clustered;
3  $P \leftarrow \varnothing$;
4  for $i$ from 1 to $g$ do
5      $p \leftarrow$ generate_point$(x_{min}, x_{max}, y_{min}, y_{max})$ //generate a random point in the
         given graph coordinates;
6      add$(P, p)$;
7  foreach $n$ in $N$ do
8      $p \leftarrow$ random$(P)$;
9      $n_x, n_y \leftarrow$ generate_point$(p_x - c_x, p_x + c_x, p_y - c_y, p_y + c_y)$ //with cluster size;
10 return $N$;
```

To create realistic instances, the generator uses Algorithm 18 to cluster nodes as we find in geographical regions. The different groups of nodes could represent cities where heavy vehicles cannot enter. In our case, a mix of different types of vehicles could be used such as cars, trucks or bikes.

5.4.2 Parameter tuning

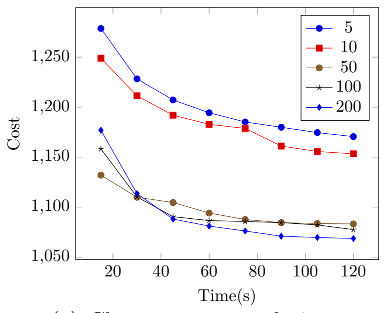
Figures from 5.8(a) to 5.8(d) show the convergence of the GA over time. These results were used to set the values of the algorithm parameters in Table 5.3. These parameters values are used as standard during all the runs to allow the algorithm to provide the best performances.

GA parameters	Values	References
running population size *rps*	200	5.3.2
mutation probability *mp*	0.9	5.3.3
crossover probability *cp*	0.9	5.3.3

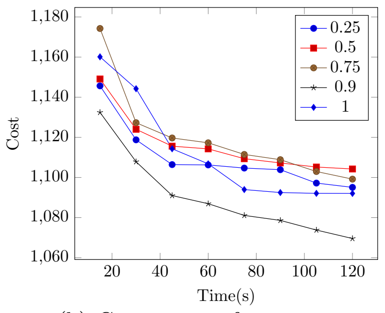
Table 5.3: GA parameters

5.4.3 Comparison on generated instances

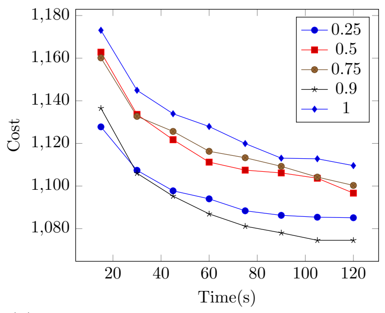
As stated in Chen et al. (2016), there is presently no reference benchmark for the specific problem presented in this chapter. Therefore, the previously tested algorithm TSAM



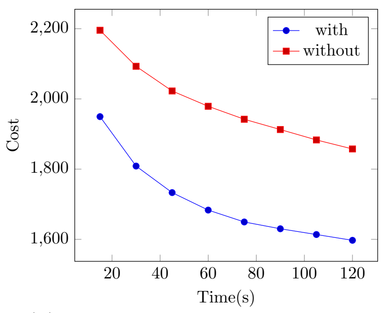
(a) Chromosome population



(b) Crossover performance



(c) Mutation operators' performance



(d) Multi-threading performance

Figure 5.8: Convergence of the proposed algorithm with different parameters values

from Chapter 4 is used as a reference and compared with the new algorithm. For a fair comparison, just like GA, we use TSAM with the procedure `init_solution`() in Algorithm 23 to initialise solutions and without function `reorder_routes`().

	GA				TSAM			
Instance	Avg objective	Avg time (s)	Best objective	Best time (s)	Avg objective	Avg time (s)	Best objective	Best time (s)
d5q50k2c1r10	441.99	0	441.99	0	448.93	48	441.99	41
d5q50k2c1r15	581.52	0	581.52	1	569.61	44	483.73	185
d5q50k4c1r30	1049.08	107	1026.73	24	1115.78	98	953.6	77
d5q50k5c1r50	1639.63	262	1466.71	204	1641.8	93	1445.21	100
d5q50k2c2r10	589.66	0	589.66	0	589.66	8	589.66	2
d5q50k2c2r15	648.68	122	604.8	78	666.35	63	628.26	2
d5q50k4c2r30	1221.52	76	1105.29	166	1262.67	73	1122.31	3
d5q50k5c2r50	1716.89	226	1598.88	294	1768.7	106	1568.26	82
d5q50k5c5r10	387.99	7	387.99	2	403.48	31	387.99	1
d5q50k5c5r15	632.6	12	622.19	5	694.33	69	622.19	123
d5q50k5c5r30	950.54	103	866.31	144	987.14	114	851.59	287
d5q50k5c5r50	1529.79	239	1412.53	229	1525.35	138	1419.44	49
d5q100k2c1r10	459.74	0	459.74	0	459.74	1	459.74	1
d5q100k2c1r15	549.18	128	533.91	52	606.25	42	533.91	76
d5q100k4c1r30	1012.37	172	940.75	221	1124.93	87	981.52	49
d5q100k5c1r50	1570.08	263	1358.34	296	1600.25	119	1445.96	108
d5q100k2c2r10	605.43	0	605.43	0	605.43	0	605.43	0
d5q100k2c2r15	769.09	1	769.09	1	769.09	5	769.09	1
d5q100k4c2r30	1164.18	27	1052.87	286	1197.9	88	1111.15	32
d5q100k5c2r50	1657.39	252	1532	277	1716.5	101	1595	97
d5q100k5c5r10	387.99	5	387.99	6	398.73	50	387.99	188
d5q100k5c5r15	631.29	11	622.19	3	704.99	78	587.66	169
d5q100k5c5r30	947.1	105	852.88	79	963.72	110	859.45	245
d5q100k5c5r50	1513.53	248	1395.13	261	1487.83	152	1376.79	209

TABLE 5.4: Results for GA and TSAM comparison

Table 5.4 shows the results of TSAM and GA for several instances of different request quantities. The instance names give the request demand d, the capacity of the vehicle q, the vehicle number k, the cross-dock number c and the request number r. On top of that, there are vehicles with different start and end depots, requests with special vehicle type requirements and vehicle time windows. The termination criteria for TSAM and GA are set at 5 min. Columns *Avg objective* and *Avg time* (s) give the average of the results over 30 runs while columns *Best objective* and *Best time* (s) give the details of the best solution. From Table 5.4 it can be inferred that on average the GA outperforms TSAM on this type of instances. This could be explained by the population-based characteristic of a GA allowing a thorough exploration of a search space containing several local minima.

5.4.4 Comparison on existing benchmark instances

	GA					TSAM				
Instance	Avg distance	Avg time (s)	Best distance	Best vehicle	Best time (s)	Avg distance	Avg time (s)	Best distance	Best vehicle	Best time (s)
lc101s	831.93	88	828.94	10	84	828.94	29	828.94	10	18
lr101s	1653.42	134	1650.8	19	141	1650.8	94	1650.8	19	21
lrc101s	1766.25	203	1714.99	15	172	1741.47	158	1703.21	15	99
lc1_2_1s	2828.65	30	2704.57	20	1019	2754.66	635	2704.57	20	753
lr1_2_1s	5319.74	1115	5015.94	23	1095	5231.25	996	5024.23	23	793
lrc1_2_1	3977.33	1121	3744.31	20	1175	3863.58	879	3724.37	20	781
lc109q	835.42	199	827.82	10	114	827.82	30	827.82	10	29
lr201	1421.07	251	1364.19	9	286	1419.98	174	1330.15	8	247
lrc_1_2_5s	4233.85	1185	4012.59	21	960	4184.54	1099	3985.84	20	1139

TABLE 5.5: Results on Li&Lim benchmark

Table 5.5 shows the results of the GA on some instances of Li&Lim's benchmark SINTEF compared to those of TSAM. To test all the different configurations, instances from each group (clustered, random, and random-clustered nodes) for 100 customers and 200 customers were selected. Columns *Avg distance* and *Avg time* (*s*) give the average of the results over 30 runs while columns *Best distance*, *Best vehicle* and *Best time* (*s*) give the details of the best solution. Those results include the distance driven by all the vehicles, the number of vehicles used and the time at which the solution was found. The termination criterion for the meta-heuristics is set at 25 min. From Table 5.5 it can be seen that the GA is able to match TSAM on some instances. However, TSAM yields on average better results than GA on other instances. As opposed to the previous subsection, these results might indicate that a single-solution algorithm can provide better results on this type of instances.

5.4.5 Comparison on clustered and non-clustered instances

	GA				TSAM			
Instance	clustered 4c	clustered 8c	random 4c	random 8c	clustered 4c	clustered 8c	random 4c	random 8c
d5q50k4c4r30	335.18	553.1	805.81	872.67	339.55	636.93	892.62	931.18
d5q50k4c4r30	510.27	296.91	774.13	860.35	633.42	393.55	887.52	985.79
d5q50k4c4r30	518.38	418.19	885.79	866.18	609.68	528.78	977.45	1014.55

TABLE 5.6: Results on clustered and non-clustered instances

Table 5.6 shows the performances of both GA and TSAM algorithms on clustered and non-clustered instances. The results are averages of 30 runs of 20 minutes on 4 different types of instances containing clustered or random nodes and 4 or 8 cross-docks represented by columns "4c" and "8c" respectively. As in the previous section, the instance details are given by their names. Each type of instances was generated 3 times. Table 5.6

shows that GA outperforms TSAM in any type of clustered or non-clustered instances. Increasing the number of cross-docks has an impact on the complexity of instances, especially if the nodes are clustered. However, the gap between both algorithms stay the same - the GA provides better results.

5.4.6 Comparison on dynamic instances

	1		2		3		4		5		6		7		8		9		10	
Instance	cost	time	cost	time	cost	time	cost	time	cost	time	cost	time	cost	time	cost	time	cost	time	cost	time
d5q50k4c4r20																				
GA	127.79	0	127.79	0	262.76	0	370.16	0	388.38	0	406.71	0	475	0	608.22	217	667.42	328	716.84	400
TSAM	127.79	0	127.79	0	262.76	0	370.16	0	388.38	0	406.71	0	475	0	629.16	79	675.18	84	730.34	170
d5q50k4c4r20																				
GA	91.27	0	91.27	0	218.87	0	218.87	0	317.75	0	343.11	0	469.21	20	561.64	12	623.93	278	685.52	307
TSAM	91.27	0	91.27	0	218.87	0	218.87	0	317.75	0	343.11	0	472.61	135	577.86	109	674.05	94	763.85	100
d5q50k4c4r20																				
GA	182.8	0	358.74	0	369.47	0	470.84	23	608.85	164	595.93	343	629.09	348	724.94	49	724.94	0	724.94	0
TSAM	182.8	0	358.74	0	369.47	0	471.15	0	589.69	119	589.11	10	635.78	134	704.6	50	704.1	25	703.96	182

Table 5.7: Results of all the intervals on dynamic instances

Table 5.7 shows the average results of 30 runs for GA and TSAM on dynamic instances. The time horizon is divided into 10 intervals of equal length. The algorithm is launched at the beginning of each interval, with new information about customer requests. The cost and time of both GA and TSAM are reported on 3 different instances series. As in the previous section, the instance details are given by their names. The results from Table 5.7 also demonstrate the effectiveness of GA in a dynamic setting. Compared to the previous section, the number of requests is decreased to 20. As shown in the third series of instances, TSAM became competitive after the fifth period. However, GA still provided better results for the two first series.

5.4.7 Comparison on dynamic multi-objective instances

	GA				TSAM			
Instance	distance	delay	vehicle	elevation	distance	delay	vehicle	elevation
d5q60k4c4r20	1819.36	5175.17	3.93	0.16	1753.14	5027.42	3.9	0.15
d5q60k4c4r20	1577.48	1158.54	3.9	0.15	1648.64	1974.91	3.87	0.15
d5q60k4c4r20	1644.95	2683.85	3.9	0.12	1525.23	2062.53	3.77	0.13

Table 5.8: Results of the last interval on dynamic multi-objective instances

Table 5.8 also shows the results of GA and TSAM on dynamic instances but with multiple objectives considered. The algorithms are run 30 times on the instances containing 10 periods. As in the previous section, the instance details are given by their names.

Details about the last periods are reported to show how these algorithms converge on such instances. The probability distribution is (0.5, 0.2, 0.1, 0.2) for distance, delays, vehicle and elevation objectives respectively. From Table 5.8, it is not clear anymore which algorithm is better than the other one. Both algorithms were able to compete with each other and provided mixed results that can be explained by the randomness included in them. Indeed, at each period, a new objective is randomly selected to be improved. Moreover, if an algorithm happens to find a bad solution during the first periods, it will carry this bad solution until the end due to the continuity of the time horizon. In other words, finding a bad solution at the beginning penalises more than finding one at the end.

5.4.8 Performance of the prediction feature using uncertainty

	With				Without			
Instance	distance	delay	vehicle	elevation	distance	delay	vehicle	elevation
d5q70k4c4r30	1839.29	15046.13	4	0.18	2231.16	3035.66	3.97	0.15
d5q70k4c4r30	1900.82	11101.11	4	0.18	2231.44	5530.19	4	0.16
d5q70k4c4r30	2011.67	12349.6	4	0.18	2183.8	6201.22	4	0.16

Table 5.9: GA results of last interval with and without the node prediction feature on dynamic multi-objective instances

The node prediction feature is tested and results are reported in Table 5.9. To better demonstrate the performances of this feature, the GA is launched 30 times on each instance over 15 periods, with and without the feature enabled. Average results of the last periods are reported to show how the algorithm converges with such settings. As in the previous section, the instance details are given by their names. Table 5.9 demonstrates the validity of the node prediction feature. The probability distribution is (0.5, 0.2, 0.1, 0.2) for distance, delays, vehicle and elevation objectives respectively. GA was able to find better solutions for all the instances with regard to the probability distribution. However, no improvement is made on objectives elevation and vehicle as they have a low probability. It must be noted that improving those objectives can be difficult because of the elevation profile of the instance and the number of requests being high compared to vehicle capacities.

5.5 Conclusion

In this chapter, we presented a new method to solve a rich VRP with pickup and delivery. The problem at hand has extended previous work to originally satisfy a company's needs.

The company is responsible for delivering goods by bikes and therefore must not only consider distances and time windows, but also altitude gradients and the number of bikes used. This company usually receives requests throughout the day and can use cross-dock facilities to exchange loads between bikes. To efficiently solve the problem, we modelled a new multi-objective, dynamic VRP dealing with uncertainty. A predictive node feature was then developed to use historical data to improve the performance of the algorithm. A genetic algorithm was introduced and compared to the previous algorithm. A new crossover was developed to handle our multiple cross-docks problem without any repair function. Computational results show that the new algorithm is highly competitive while being simpler to develop and to configure.

Three main gaps are identified as opportunities for further research. Firstly, when considering the Pareto mode, the new algorithm can be improved to better cope with multiple objectives. A mechanism could be added to direct the search toward unexplored regions of the Pareto front. Secondly, a feature which reuses the population from a previous interval with an insertion heuristic could improve the speed of the search in dynamic conditions. Thirdly, as the literature only has few benchmarks for the VRP with a single cross-dock, one could create a benchmark for the PDP with multiple cross-docks. Otherwise, a reference algorithm in multi-objective could be implemented for further comparisons.

Chapter 6

A learning algorithm for the vehicle and container routing problem

6.1 Introduction

In previous chapters, the PI was viewed from a vehicle routing perspective, algorithms were proposed to solve the VRP of different sizes. This allowed containers to be carried from customers to customers as in a pickup and delivery problem. PI requirements such as consolidations were considered and modelled using constraints like cross-docking. However, the PI can also be viewed from a container routing perspective (Sarraj et al., 2012). Readers can refer to Sarraj et al. (2013) for more information on the idea of container routing. The recent work of Gontara et al. (2018) was also about the analogy between routing PI-Containers in PI and routing data using BGP Protocols. In their network, requests can go through several cities connected by larger providers. This way of modelling the problem allows us to see it as an ad hoc network where the links represent the VRP routes which can change over time. Hence the comparison with ad hoc originally made by Sarraj (2013).

In this chapter, a new perspective is presented - the Vehicle and Container Routing Problem (VCRP). This idea accounts for several VRPs being solved in different areas which are linked and included in a container routing problem. Figure 6.1 shows a possible scenario of France with several PI-hubs linked by the road network. With this representation, a VRP area can include one or several PI-hubs. Containers are therefore routed to go through sequences of VRP areas in which vehicles carry them to the customers. This can also be viewed as a solution to connect CL and PI as described

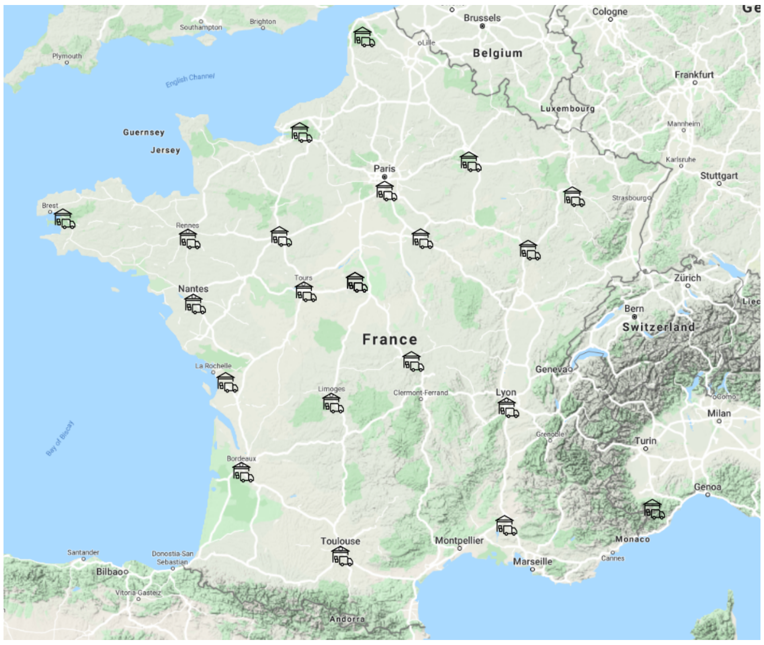


FIGURE 6.1: France with PI-hub facilities

in Crainic and Montreuil (2016). This research aims to improve the routing of containers on a higher level. VRP algorithms proposed in previous parts will be used alongside a container routing algorithm introduced in the following sections. To further exploit the PI concept, Machine Learning (ML) methods can be used and included as the container algorithm to improve driving distances and vehicles fulfilment. As a result, we design an ML algorithm based on a state-of-the-art method to solve this new logistics problem. Since greenhouse gases emission and cost are correlated to those metrics, we would not only contribute to the idea of "drivers should be able to return home at end of their shifts" (Montreuil, 2011) but also to make a more sustainable SCM.

6.2 Problem definition

The VCRP aims at finding the optimal transportation route between pickup and delivery customers. The objective considered in this model is to minimise the total distance while considering the truck fulfilment rates. In future work, researchers could easily replace these metrics in the reward function by e.g., the travel time, energy or even CO_2 emission etc. Nevertheless, it should be noted that minimising empty trips contributes toward a sustainable transportation system, which is one of the goals of the PI.

The VCRP network is modelled as a directed graph in which nodes are the PI-hubs or customers and edges are vehicle routes. This graph can be separated into several clusters of sub-graphs representing VRP areas modelling cities, regions or even countries

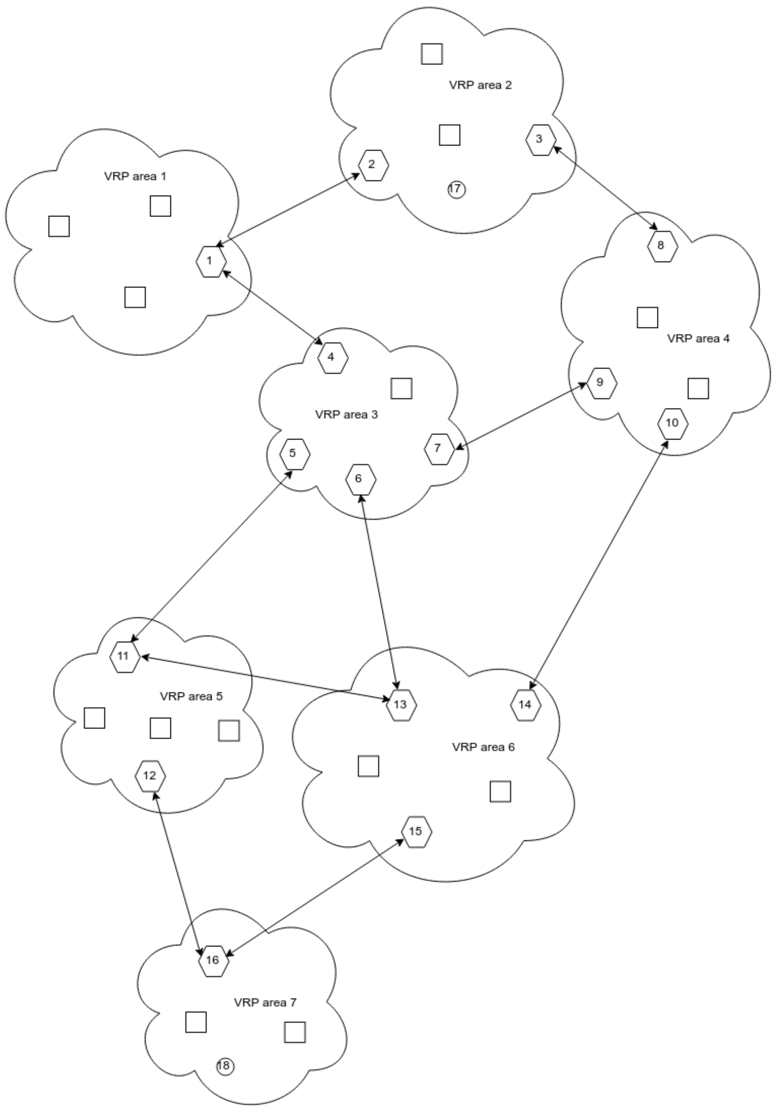


FIGURE 6.2: VRP networks

depending on the instance. Figure 6.2 shows such a topology of PI-hubs/cross-docks that can be seen as routers in the network. There are two types of PI-hubs represented by: (1) Squares for internal PI-hubs which are used for VRP consolidations only, as described in previous work. (2) Hexagons for external PI-hubs which are not only used for consolidations but also for inter-cities transportation. This difference can be viewed as the difference between Border Gateway Protocol (BGP) routers and normal routers (or switches L3). In this network, the PI-hubs' locations do not change over time as opposed to the customers and vehicle routes. Arrows in Figure 6.2 represent logistics services a container has to go through to reach another area. However, in a given area, it is assumed that any PI-hub can be linked to another one from the same area since the VRP could provide such routes, if necessary. As a result, a VRP area could be simplified as a fully connected sub-graph. Therefore, traditional routing algorithms such as OSPF, RIP or the one proposed in Gontara et al. (2018) can be used alongside the ML algorithm. Customers are distinct from the PI-hubs and are represented by circles in Figure 6.2. They are all associated with one area to be solved by one VRP heuristic.

Customer locations and characteristics are randomly generated. Then, requests between customers are handled by the VCRP model as follows: For each new request, the closest external PI-hub in the same area will be selected to handle the container. Each request path is being computed as the container progresses through the network. When a container reaches an external PI-hub, it will check the destination and provide the next-hop external PI-hub the container must be brought to. If the current and next-hop PI-hubs are in the same VRP area, the VRP heuristic would handle the transportation, otherwise, another transportation service is used. As stated in Gontara et al. (2018), this separation would allow different Logistic Service Providers (LSP) participating in PI to handle their networks as they wish. When a container reaches the destination area, it would be brought to the customer instead of being carried to another external PI-hub. It is assumed that vehicle routes can only be congested by the customer requests creating a bottleneck in one or several VRP areas. Therefore, external factors such as private vehicles on roads is not considered. For the sake of simplicity, it is also assumed that each time slot is equal to one day. Transportation services in the entire network start in the morning, therefore a container needs one working day to go through a VRP area or be carried from one area to another. If a container has to go through 3 VRP areas, the transportation will necessarily last 5 days (3 areas + 2 inter transportation). This is the case even if the actual driving time is several hours. The total distance and number of empty vehicles metrics can still be considered. The dynamic case would be considered in future work as we would need to anticipate driving transportation duration for the system to know when a container would arrive at its next hop hub. Whereas in this static case, we can easily anticipate that the container will arrive the next day.

Request path ID	Paths
1	17 2 1 4 5 11 12 16 18
2	17 2 1 4 6 13 15 16 18
3	17 3 8 10 14 15 16 18

Table 6.1: Examples of VCRP request path

In the traditional pickup and delivery problem, requests are usually from customers to customers, and could also be brought from/to a depot for some variants of the VRP. However, as modelled in Chapter 5, requests in our VRP are more flexible and can be placed from cross-docks to cross-docks, customers to customers or customers to cross-docks etc. This allows VRP heuristics to handle requests passing through VRP areas via two external PI-hubs. Table 6.1 shows examples of possible request paths for a container transportation from customer 17 to customer 18 represented in Figure 6.2. Considering request path 1, in the beginning, a call for transportation is placed from customer 17 to customer 18. The container is routed to the closest external PI-hub 2 by the VRP heuristic. The VCRP algorithm provides node 1 as the next-hop destination the container must be brought to via a special logistics vehicle operating between those areas. At node 1, node 4 is provided as the next-hop destination without the need for a VRP vehicle to operate. However, at node 4, the container has to go through the VRP network to reach node 5. Those steps are repeated until the container reaches the destination area 7 where a VRP vehicle will carry it to customer 18.

As explained above, the system infrastructures are static but the model can have dynamic properties. Requests and VRP solutions differ from day to day and create new scenarios that need to be dealt with. Traditional packet routing algorithms can deal with congestion but not under situations that have not been anticipated beforehand by developers. In other words, they do not learn from previous experiences regarding network abnormalities such as congestion etc. On top of that, they usually react only when congestion is already present. Moreover, when congestion is detected in a path link of a container, they would try to select another path even though the congestion could have been gone by the time the container reaches the link. This is because they were designed for the DI and react in milliseconds, whereas in the PI, the time is measured in hours or even days. Therefore, an intelligent network traffic control method is essential to avoid this problem. RL is typically the kind of algorithm that could anticipate that. Moreover, it is said that even if the data are not labelled correctly, ML would still be able to learn, but slowly. Therefore, in our case, even if VRP heuristics are not able to provide optimal solutions, the ML algorithm could still learn. All the VRP routing decisions of the entire VCRP are used to create a large set of data and are fed to a single RL network to learn from. Consequently, all the PI-hub uses the same RL network to get the path next-hop. It is important to note that, as stated in You et al. (2019), algorithms with a centralised learning process are not necessarily the best choices for real computer networks. This could be due to centralised learning controllers not being able to gather information about their neighbours. The bandwidth can be a constraint for a widely distributed system to communicate as protocol information packets would add up on top of data packets. However, since our model deals with the PI, in this analogy, routers are PI-facilities. Therefore, PI-hubs will still be able to communicate with each other by using an actual DI network on top of the PI one. This would avoid issues related to bandwidth congestion as we deal with physical containers. We expect the RL algorithm to learn when and where to send containers in an efficient way based on a small number of input metrics: the number of containers, their sizes/weights, their current locations and the time. The algorithm would learn how to balance the flow of containers to avoid congestion by carefully selecting intermediate PI-hubs while minimising the distance and truck unfulfilment rates.

6.3 Methodology

6.3.1 Reinforcement learning

RL algorithms work in a trial and error way through the actions of an agent. These actions are performed in an environment at discrete times indexed by t and called episodes. There are two main types of RL methods: (1) The policy-based methods such as "REINFORCE" and "Policy Gradients"; and (2) the value-based methods such as "Q-learning" and "Deep Q-learning". Policy-based methods learn a policy $\pi : s_t \rightarrow a_t$ which describes which action a_t should be taken in each state s_t. The objective is to find the optimal policy directly without the Q-value and by using the total rewards of each episode; Value-based methods learn a value function that maps each state-action pair to a value. This estimates the value of each state based on future rewards that can be obtained, starting at a given state. The objective is to find the optimal value function. The higher the value, the better the action. Equation 6.1 shows the value of a state s_t which is the expected total sum of discounted future rewards starting from this state.

$$V(s_t) = E\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\right] = E\left[R_{t+1} + \gamma V(s_{t+1})\right] \tag{6.1}$$

where $\gamma \in [0, 1]$ is the discount factor.

6.3.2 Environment

The environment is composed of several VRP areas as shown in Figure 6.2. As opposed to classical implementations where only one action is available from the environment, our implementation provides an agent with a set of functions it can perform. Function `handle_requests()` sequentially generates new requests, creates and solves VRP instances with the requests present in a given area. Fulfilment rate and distance metrics are retrieved from the VRP solutions. Function `move_requests()` transfers requests from one node to another. Function `select_requests()` handles requests that need to be routed and remove the ones that have arrived. Function `preprocess_requests()` returns the current state given a request being routed. Function `do_action()` performs the routing for the request being processed. When called iteratively, function `get_results`() returns the different states, actions and rewards for any request arrived.

An episode in the RL algorithm corresponds to a day in the transportation network. Therefore, at the end of every episode, all the present requests are already routed and transferred to another location. Each episode is composed of several steps which are defined as the action of routing transiting requests inside and outside VRP areas. The environment embeds the Dijkstra algorithm coded with a priority queue to be used in the reward function for finding the shortest paths between nodes. Since customers appear at different locations over time, the RL algorithm would struggle to learn where to route requests. Therefore, when selecting the next-hop hub during the routing phase for a given request, the inputs do not consider the departure and arrival customers but instead closest external hubs. As a consequence, a request is routed to an external hub which is in the destination area. This external hub is chosen by comparing the path distances to all external hubs in the destination area and selecting the shortest one.

6.3.3 Asynchronous Advantage Actor-Critic

We implement our RL algorithm as the Asynchronous Advantage Actor-Critic (A3C) described in Mnih et al. (2016). The most important differences with traditional RL algorithms are now explained.

The actor-critic property combines the benefits of both value-based and policy-based approaches. In the case of A3C, the network will estimate both a value function $V(s)$ which determines how good a certain state is to be in and a policy π(s) which is a set of action probability outputs. These will be separate fully-connected layers sitting at the top of the network. The policy structure is called the actor, which takes actions in states. The value structure is called the critic, which criticises the current policy being followed by the actor. The structure of the actor-critic model is illustrated in Figure 6.3. The environment presents the representation of the current state s_t to both the actor and the critic. The actor uses this input to compute the action to perform according to its current policy. The actor then selects the action which causes the agent to get into a new state s_{t+1}

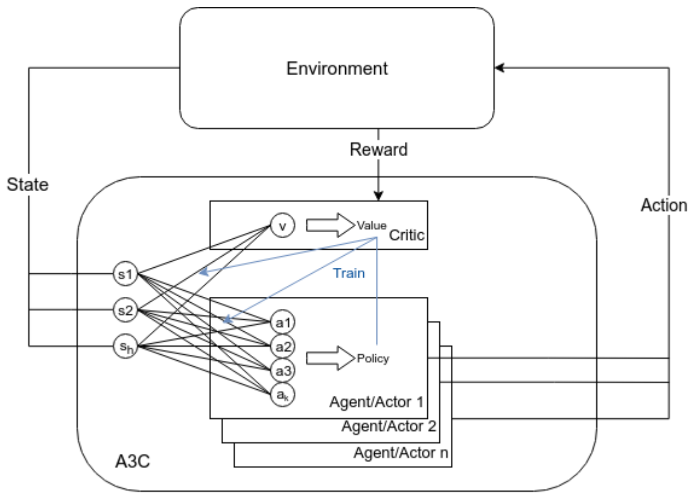


FIGURE 6.3: A3C architecture

In DRL, a single agent represented by a single neural network interacts with a single environment. Whereas, the asynchronous property of A3C allows the algorithm to utilise multiple agents to learn more efficiently from copies of the environment. In this way, all agents start at different points within the environment which produces a more diverse overall experience for training. The agents share one neural network that feeds into separate actors to perform different actions, into one critic that connects them together. The agents provide each other with knowledge of the environment through the critic. The network gets updated and then the critic shares the information to all agents. This consists of two values being back-propagated through the critic to the agents - value Loss 6.5 (related to the critic) and policy Loss 6.6 (related to the actor).

The critic knows the value of the state but it doesn't know how much better an action could be compared to the current value of state. This is what the advantage property represented in Eq. 6.2 is used for. $Q(s_t, a_t)$ represents the max value we could get from state s_t, and $V(s_t)$ the average value. Intuitively, this means how much better it is to take a specific action a_t compared to the average, at the given state. The discounted rewards in Eq. 6.3 is used to tell the agent which of its actions were "good" and which were "bad". When the environment gives a new reward based on the previous action, the critic observes a new state and computes its estimate for this new state. Based on the reward and the current value function estimation, both R_{t+1} and $\gamma V(s_{t+1})$ are now available and used in Eq. 6.4.

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t) \tag{6.2}$$

$$Q(s_t, a_t) = E\left[R_{t+1} + \gamma V(s_{t+1})\right] \tag{6.3}$$

$$A(s_t, a_t) = R_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{6.4}$$

$$L_v = \sum (R_t - V(s_t))^2 \tag{6.5}$$

where $R_t - V(s_t)$ is an estimate of $A(s_t, a_t)$

$$L_p = -\log(\pi(s_t)) * A(s_t) - \beta * H(\pi) \tag{6.6}$$

Value Loss 6.5 and Policy Loss 6.6 functions are used in order to encourage and discourage actions accordingly. The policy function incorporates the advantage $A(s)$ and an entropy $H(\pi)$ which measures the diversity of the action probabilities. As mentioned, these losses provide gradients to update the global network parameters (the critic and actor). The training of the two networks is performed separately and uses gradient ascent to find the global maximum. Each of these gradients is typically clipped to prevent overly-large parameter updates which can destabilise the policy.

6.3.4 A3C implementation

Since the RL algorithm is used to route requests through a VCRP network, we define a state input as a matrix M representing the current position and destination of a request. The matrix dimensions are $N_e * N_e$ where N_e is the number of external hubs. This input matrix is initialised with zeros and gives the request flow as described in Eq. 6.7. The output is a vector that gives probabilities to select the next-hop hub. From an RL perspective, this is the action the agent can perform. Given a request location, a certain number of hubs are not directly reachable due to the network topology. Therefore, we set the vector dimension to match the number of gates N_g an external PI-hub has. As a consequence, instead of providing a hub ID, the output gives the gate ID that will be

Algorithm 19: Reward function

```
1  //Input: agent_action, dijkstra_action, request;
2  //Output: reward;
3  $dijkstra\_ehub \leftarrow request\_current\_node\_gates[dijkstra\_action]$;
4  $agent\_ehub \leftarrow request\_current\_node\_gates[agent\_action]$;
5  if $dijkstra\_ehub = agent\_ehub$ then
6      $dreward \leftarrow 1$
7  else
8      $distances \leftarrow \varnothing$;
9      foreach $hub$ in $request\_current\_node\_gates$ do
10         $d \leftarrow$ get_dijkstra_distance($hub$, $request\_arrival\_hub$);
11         append($distances$, $d$);
12         if $hub = agent\_ehub$ then
13             $cdist \leftarrow d$;
14     $d_{max} \leftarrow$ max($distances$);
15     $d_{min} \leftarrow$ min($distances$);
16     $delta \leftarrow d_{max} - d_{min}$;
17     $dreward \leftarrow -(cdist - d_{min})/delta$;
18 if $request\_next\_node$ in $request\_path$ then
19     $loops \leftarrow loop + 1$;
20     $reward \leftarrow -1$;
21 else
22     $reward \leftarrow \alpha * dreward + \sigma * freward$ //fulfilfment $freward$ is calculated and
         provided afterward;
23 if $request\_next\_node\_area = request\_arrival\_area$ then
24     $reward \leftarrow reward + 10$;
```

used to reach the connected PI-hub. This representation prevents the algorithm from producing infeasible routes by giving non-connected hubs as the next-hops.

$$M[current\_node][destination\_node] = 1 \tag{6.7}$$

The reward function in Algorithm 19 takes into account the distances and the fulfilment rate of vehicles. This allows request paths to converge toward the shortest paths while favouring consolidations. As fulfilment and distance objectives can compete with each other, we introduced two parameters α and σ to control this trade-off in step 22. As stated in the comment, $freward$ is calculated afterwards as all the requests need to be routed to calculate the resulting fulfilment. Steps 3 and 4 retrieve the neighbour hubs from the Dijkstra and RL algorithms, respectively. Step 10 is used to identify the distance to reach the destination hub from a selected neighbour hub. The resulting distance is then used in step 17 to calculate a penalty induced by taking a certain gate.

Algorithm 20: Training algorithm

```
1  //Input: baseline_values, rewards, actions, states;
2  $value \leftarrow 0$;
3  $values \leftarrow \varnothing$;
4  foreach $reward$ in reversed($rewards$) do
5      if $reward = -1$ then
6          $value \leftarrow 0$;
7      $value \leftarrow reward + \gamma * value$;
8      add($value$, $values$);
9  reverse($values$);
10 $advantages \leftarrow values$ - $baseline\_values$;
11 train_agent($states$, $actions$, $values$, $advantages$);
```

In step 20, a penalty is added if a container returns to a PI-hub already visited - a loop is made by the container. Finally, a bonus is added in step 24 when a container reaches a PI-hub located in the same area of its destination.

Algorithm 20 depicts our training function. As implemented in steps 5 and 6, the advantage used in our algorithm is different from the standard one represented in Eq. 6.2. We set the value to 0 when a request makes a loop. As a consequence, the agent does not consider the discounted rewards from the part of the path which contains loops. Therefore, this modification prevents the agent from learning paths containing loops.

ANN properties	layer 1	layer 2	layer 3	policy output	value output
outputs number	256	512	256	N_g	1
activation	relu	relu	relu	softmax	none
weights initialiser	xavier	xavier	xavier	xavier	xavier
biases initialiser	zeros	zeros	zeros	none	none

TABLE 6.2: Neural network configuration

Our implementation of A3C uses the Adam algorithm as the optimiser with the network configuration described in Table 6.2. Readers can refer to the papers from He et al. (2015); Glorot and Bengio (2010); Bishop (2006); Kingma and Ba (2014) for more details about these properties. Our A3C algorithm launches as many agents as there are VRP areas. Algorithm 21 describes the instructions processed by the agents in different threads. As detailed in step 3, we include a series of instructions to synchronise the agents with each other. This is to avoid race conditions in the case an agent X is, for example, moving a request to another area handled by an agent Y. If agent Y has already routed its requests, the ones coming from agent X will not be routed during the current episode. Step 11 checks if there are requests in the given environment area to

Algorithm 21: Agent algorithm

```
1  //Input: environment $env$ handled by the current agent;
2  while $T < T_{max}$ do
3      wait() //wait for all the other agents to reach this step;
4      handle_requests();
5      wait();
6      move_requests();
7      wait();
8      select_requests();
9      wait();
10     $state$, $terminal$ = preprocess_requests();
11     while $terminal = false$ do
12         $policy$, $value$ ← get_policy_and_value($state$);
13         $action$ ← random_choice($N_g$, $policy$);
14         do_action($value$, $action$);
15         $state$, $terminal$ ← preprocess_requests();
16         while $True$ do
17             $values$, $states$, $actions$, $rewards$ ← get_results();
18             if $rewards \neq \varnothing$ then
19                 do_training($values$, $rewards$, $actions$, $states$);
20             else
21                 break;
```

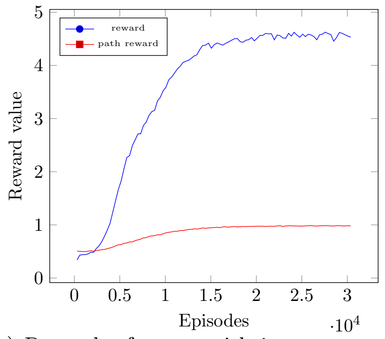
route. Step 12 retrieves the policy and value from the ANN for a given state. Step 13 select a random action given a probability distribution from the policy. Step 14 performs the action and stores the value and action in the environment alongside the state and reward for a given request. Step 17 retrieves all the variables needed to train the agent when some requests have arrived.

6.4 Computational results

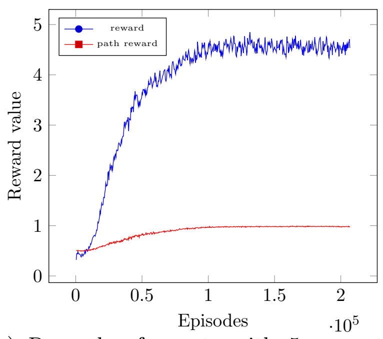
All the computational results were obtained with a computer that has the following specifications: a CPU 'Intel(R) Core(TM) i9-7900X CPU @3.30GHz' and 32 GB of RAM. The proposed algorithms are developed with python 3.7 using the following libraries: TensorFlow version 1.14.

6.4.1 Example of 7 areas for distance optimisation

The machine learning algorithm is launched on the network represented in Figure 6.2. Note that since this research is a proof of a new concept, the size of the network is kept relatively small for ease of analysis and understanding. Each area is of dimension 100*100 while the entire graph dimension is 1,000*1,000. All the external hubs have 4 gates to connect with hubs from other areas via logistic service providers. Several gates can lead to the same hub when the number of connected neighbour areas is less than 4. Requests are randomly generated between each pair of external hubs with a uniform distribution. The request load is randomly selected between 1 and 5. The vehicle capacities are set to 30 while parameters α and σ equal 1 and 0, respectively. We report the results of the RL algorithm learning on its own for 2 different examples of traffic intensity in Figure 6.4. The reward computed in Algorithm 19 is plotted alongside the path reward computed at step 17. The former represents the overall average quality of an agent's decisions while the latter represents the average quality of an action regarding the shortest path. Figure 6.5 shows the performance of the algorithm while using Dijkstra to make routing decisions until episode 200,000. After this episode, the RL algorithm learns on its own.

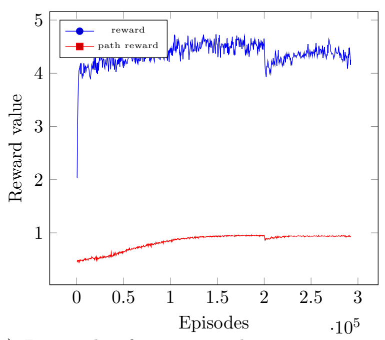


(a) Rewards of agents with 1 request generated per area per episode

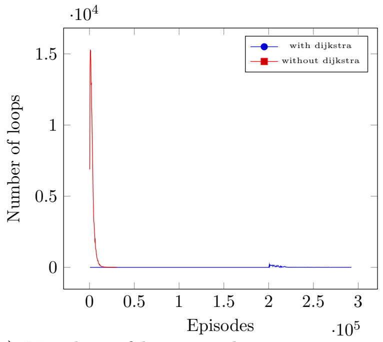


(b) Rewards of agents with 5 requests generated per area per episode

FIGURE 6.4: Convergence of the algorithm with different parameter values for the traffic



(a) Rewards of agents with 1 request generated per episode



(b) Number of loops with 1 request generated per episode

FIGURE 6.5: Convergence of the algorithm learning with Dijkstra

6.4.2 Example of 6 areas for fulfilment optimisation

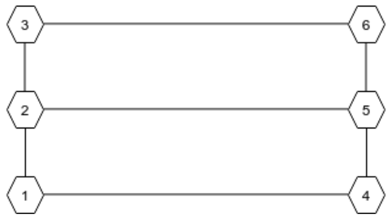
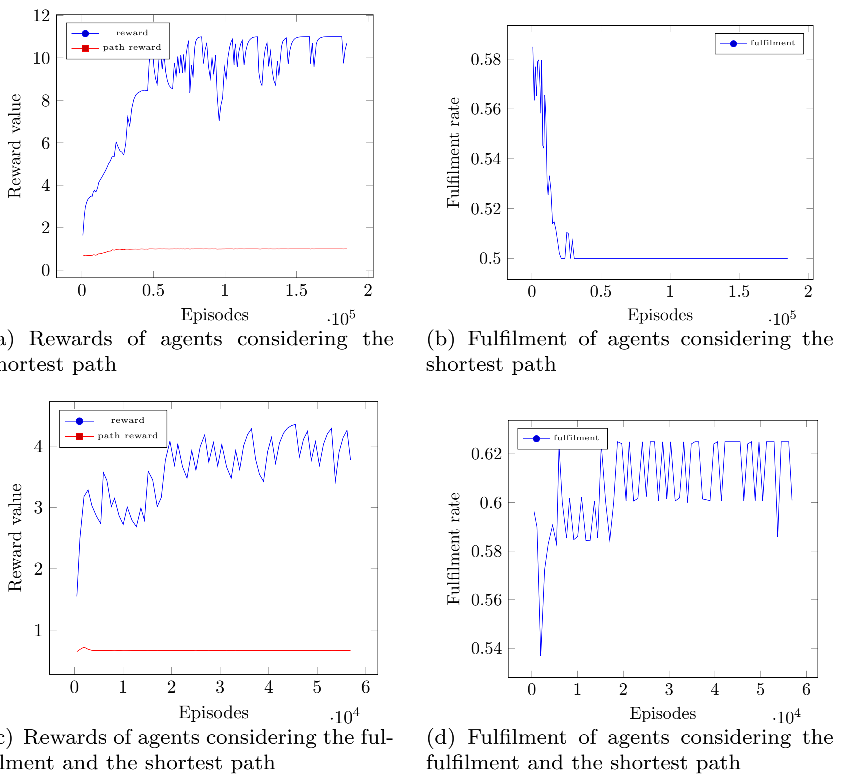


FIGURE 6.6: VRP networks including 6 external hubs in 6 different areas

In this section, we will analyse the benefit of machine learning (using fulfillment reward) in improving vehicle fulfillment and reducing travel distance. This will be done by comparing the case with shortest path rewards only against the case with both the shortest path and fulfilment rewards. The ML algorithm is launched on the network represented in Figure 6.6. The network was modelled to emphasise a specific use case in which considering the fulfilment rate could improve global efficiency. The entire graph dimension is 300*1,000. At each episode, two requests are generated at node 1 and node 3 to be delivered at node 4 and node 6, respectively. The request load is fixed at 5. The vehicle capacities are set to 10 while parameters α and σ equal to 0.1 and 1, respectively. Figure 6.7 illustrates the performance of the algorithm considering different reward functions on the network.



(a) Rewards of agents considering the shortest path

(b) Fulfilment of agents considering the shortest path

(c) Rewards of agents considering the fulfilment and the shortest path

(d) Fulfilment of agents considering the fulfilment and the shortest path

FIGURE 6.7: Convergence of the algorithm with different reward functions

6.4.3 Discussion

Our computational results show that ML techniques can be applied to routing problems to not only optimise the driving distance with the shortest path but also to improve the vehicles' fulfilment via consolidations. From Figures 6.4(a) and 6.4(b), we demonstrated that the proposed technique is entirely independent of the existing routing protocols to learn how to handle requests. The path rewards attest that the algorithm is able to route requests with near-optimal decisions after 100,000 episodes when there are 5 requests generated per area and after 15,000 episodes when there is 1 request generated.

RL for routing suffers from a drawback: during the training, containers will be sent in wrong directions causing delays. One possible solution is to run a simulation and train the network before handling real-world requests. Another solution is to route containers with a shortest path routing algorithm such as Dijkstra to avoid sending containers to wrong destinations. In the meantime, the RL algorithm could learn these routes to reduce the number of loops when it would take over. A possible outcome is that one could start learning the shortest path to learn a near-optimal policy and switch to another reward function to continue optimising a different objective. Figure 6.5(a) demonstrates that the algorithm can passively learn the Dijkstra routing decisions. As a result, when Dijkstra is switched off at episode 200,000 in Figure 6.5(b), the RL algorithm can take over while limiting the number of loops. Without Dijkstra, 179,316 loops were made whereas using RL as an online method with Dijkstra significantly reduced this number to 3,694.

Moreover, we also showed that the algorithm can be used as it is to optimise different objectives by only changing the reward function. When using a smaller graph as in Figure 6.6 and considering the shortest path in the reward function, the near optimality for routing decisions is reached faster as shown in Figure 6.7(a). We can observe an average fulfilment rate of 0.5 as requests travelled directly from node 1 to node 4 and from node 3 to node 6. However, when the fulfilment is considered in the reward function as in Figure 6.7, we observe a decrease in the average path reward. This is due to the agent routing requests to hub 2 even though it is not the shortest path, they now travel via arc 2-5. This increases the distances travelled by each request but the overall cost is reduced as the vehicles have to drive less distance, hence the average fulfilment rate increased to 0.65 on average. Readers can refer to Sarraj et al. (2014) and Section 4.2 for details about the efficiency of consolidations in PI-hubs regarding logistics cost.

From a fulfilment perspective, the predictability of the traffic flow is of great importance. In Figure 6.6, an agent can notice rapidly that a request is always sent from node 1 to node 4 and another from node 3 to node 6. Therefore, during its exploration phase, it can learn that sending both requests on the arc 2-5 would increase the reward. However, in Figure 6.2, since the number, origin and destination of requests are randomly chosen, the agent would have difficulties learning when and where to do consolidations. As a consequence, we suggest a possible research direction to overcome this difficulty.

6.4.4 Future work

To better anticipate consolidation opportunities, one could add several other matrix frames to the input. The first matrix would still be the request flow as presented in Eq. 6.7. However, a second one could represent the load of the request being processed as in Eq. 6.8. As described in Algorithm 22, a third matrix could represent the accumulated load flow of all the requests in all the areas. The matrices' dimensions would still be the same - $N_e * N_e$ where N_e is the number of external hubs. One could even add temporal frames which would capture the network flow at previous episodes. Convolution layers could be envisaged to handle the different frames and extract properties that can be used by the ANN.

The agent should learn over time where to efficiently send containers in order to minimise the distances and maximise the fulfilment rates. This would be done by mapping matrix 1, 2 and 3 with the reward indicating if the vehicles were full. This design should even work in the case where requests are scheduled to depart with delays or at a specific time.

The challenge is that when the agent tries to send more containers for consolidation, the fulfilment rate would increase until a certain threshold above which it will decrease due to the use of another vehicle. Because of this, the algorithm might not be able to converge toward an optimal policy. When too many requests must be handled, another challenge occurs due to the travelled distances. This is because if the requests load in an area is greater than the capacity of the vehicles available, they would have to make detours at some hubs to satisfy part of the requests before handling the rest. As a consequence, when the distance is being optimised, those situations would induce more distance driven and decrease the reward value. This situation can be considered as a congested area.

$$M_2[r_{current\_node}][r_{destination\_node}] = r_{load} \mid r = current\_request \tag{6.8}$$

Algorithm 22: Algorithm for the requests flow input

```
1 //Input: set of requests $R$;
2 //Output: a matrix $M_3$;
3 foreach $r$ in $R$ do
4     $M_3[r_{current\_node}][r_{destination\_node}] = M_3[r_{current\_node}][r_{destination\_node}] + r_{load}$
```

6.5 Conclusion

In this chapter, a deep reinforcement learning algorithm was designed to solve the container routing problem including several VRP networks. This problem is of great practicability as PI protocols are gaining importance across the world. Several contributions were made. First, a new model handling the transportation of containers in a PI environment was designed. Second, a deep reinforcement learning method using A3C was proposed. Third, suggestions to better anticipate load deliveries and congestion were provided. Simulation results showed the relevance of the proposed method to tackle such a problem. Once the DRL algorithm was configured, several reward functions were used to solve the problem differently. The learning mechanism was used as a black box to optimise different aspects of delivery transportation like the distance and fulfilment rate.

In addition to the suggestions in Section 6.4, further research directions are possible to extend this work. (1) Instead of iteratively calculating the next-hop for requests, one could calculate the entire path when a request is placed. This would allow the algorithm to better anticipate the traffic and avoid congestion or seize consolidation opportunities. (2) Instead of using meta-heuristics in the VRP areas to transfer containers between nodes, one could use ANNs to accelerate the learning phase. As studied in Nazari et al. (2018), ANNs can be trained to provide VRP solutions instead of running heuristics. Since the locations of the hubs are static, given a set of requests, an ANN could learn to return different metrics such as distances or fulfilment rates. As a result, a trained ANN could dramatically improve the speed of the learning phase for container routing. (3) To better demonstrate the relevance of the proposed method, a simulation can be launched to compare the results with a more traditional approach such as OSPF.

Chapter 7

Conclusion and future work

This thesis investigated routing algorithms to improve transportation systems from an efficiency and sustainability perspective. Routing models were introduced to solve a variety of problems for bicycle, vehicle and container routing related to pickups and deliveries. This type of problem is of great practicability as concepts like the PI are gaining importance across the world.

The research questions raised in Section 1.2 were all answered in different chapters. First, a literature review was conducted to identify the current methodologies used within the scope presented in Section 1.1. Then, each technical chapter has an introduction stating the different gaps to be filled. The next questions related to the design and solvability of new models was answered with the different new problems and methods created. While MILP models were implemented with CPLEX to compare the performances, meta-heuristics were also developed to handle large instances. Experimental results showed the accuracy and stability of the proposed algorithms. The resulting algorithms were integrated into a framework to be used by a local delivery company. The adaptability of the models was also addressed by selecting and handling a list of constraints that are mandatory for the PI. As demonstrated in Chapter 6, methods from other domains such as the DI can also be adapted and used to offer more flexibility.

The results of this thesis on vehicle and container routing provide a deeper understanding of methodologies to solve these problems, and suggest some promising ways to solve these challenging problems using meta-heuristics and learning methods.

7.1 Summary of major contributions

Details of the contributions can be found at the end of each chapter. The most significant contributions are listed as follows:

1. A new model was proposed to tackle deliveries by push-bikes while taking into account the road slopes. It considers the energy required to move the bike as the load accumulates during the trip. A modified evolutionary local search was introduced with a new split function handling energy constraints to solve large instances in a reasonable amount of time.
2. A new rich VRP that has not been addressed before was introduced with an MILP model to be implemented with CPLEX. A multi-threaded simulated annealing algorithm with memory including new operators is introduced to handle real-world size instances.
3. A new multi-objective, dynamic VRP dealing with uncertainty was modelled to efficiently solve the problem. A genetic algorithm was developed with a predictive node feature to use historical data to improve the performance of the algorithm. A new crossover was developed to handle a multiple cross-docks problem without any repair function.
4. A new model handling the transportation of containers in a PI environment was designed with a deep reinforcement learning method using A3C. Suggestions to better anticipate load deliveries and congestion were provided. Insights as to how one could efficiently use the learning algorithm were provided to optimise different aspects of delivery transportation like the distance and fulfilment rate.

7.2 Future work

As this thesis covered a large research area, several related research topics in the routing domain have been identified. Among these topics, some promising future research directions are listed as follows:

1. While studying the power required from a cyclist to move given a certain position, Grappe et al. (1997) concluded that an important part of the energy is due to the aerodynamic drag of air. As a consequence, due to the shape of cargo-based bikes, studying the BRP while considering not only the instantaneous energy on a given road section but also the wind direction could lead to better itineraries. The model could be extended to an arc routing problem for another type of delivery.

2. As the literature only has few benchmarks for the VRP with a single cross-dock, one could create a benchmark for the rich PDP with multiple cross-docks introduced in this thesis. Also, one could integrate the proposed algorithms into a unified solution framework for more flexibility regarding multi-attribute such as working hours and rest times etc.

3. The new GA can be improved to better cope with multiple objectives by using a mechanism to direct the search toward unexplored regions while considering the Pareto front. A feature which reuses the population from a previous interval with an insertion heuristic could improve the speed of the search in dynamic conditions. A reference algorithm in multi-objective could be implemented for further comparisons.

4. Instead of iteratively calculating the next-hop for requests, one could calculate the entire path when a request is placed. This would allow the algorithm to better anticipate the traffic to avoid congestion or seize consolidation opportunities. As studied by Nazari et al. (2018), ANNs can be trained to provide VRP solutions instead of running heuristics. Therefore, instead of using meta-heuristics in the VRP areas to transfer containers between nodes, one could use ANNs to accelerate learning phases. To better demonstrate the relevance of the proposed method, a simulation can be launched with a more traditional approach such as OSPF included.

References

Abbatecola, L., Fanti, M. P., and Ukovich, W. (2016). A review of new approaches for dynamic vehicle routing problem. In *12th Conference on Automation Science and Engineering*.

Ahkamiraad, A. and Wang, Y. (2018). Capacitated and multiple cross-docked vehicle routing problem with pickup, delivery, and time windows. *Computers & Industrial Engineering*, 119:76–84.

Ahmadizar, F., Zeynivand, M., and Arkat, J. (2015). Two-level vehicle routing with cross-docking in a three-echelon supply chain: A genetic algorithm approach. *Applied Mathematical Modelling*, 39(22):7065–7081.

Albareda-Sambola, M., Fernández, E., and Laporte, G. (2014). The dynamic multiperiod vehicle routing problem with probabilistic information. *Computers & Operations Research*, 48:31–39.

Alinaghian, M., Kalantari, M. R., Bozorgi-Amiri, A., and Raad, N. G. (2016). A novel mathematical model for cross dock open-close vehicle routing problem with splitting. *International Journal of Mathematical Sciences and Computing*, 2(3):21–31.

Allen, J., Browne, M., and Holguin-Veras, J. (2010). Sustainability strategies for city logistics. *Green logistics: Improving the environmental sustainability of logistics*, pages 282–305.

Alsheikh, M. A., Lin, S., Niyato, D., and Tan, H.-P. (2014). Machine learning in wireless sensor networks: Algorithms, strategies, and applications. *IEEE Communications Surveys & Tutorials*, 16(4):1996–2018.

Atefi, R., Salari, M., Coelho, L. C., and Renaud, J. (2018). The open vehicle routing problem with decoupling points. *European Journal of Operational Research*, 265(1):316–327.

Ballot, E., Gobet, O., and Montreuil, B. (2012). Physical internet enabled open hub network design for distributed networked operations. In *Service orientation in holonic and multi-agent manufacturing control*, pages 279–292. Springer.

Barth, M. and Boriboonsomsin, K. (2009). Energy and emissions impacts of a freeway-based dynamic eco-driving system. *Transportation Research Part D: Transport and Environment*, 14(6):400–410.

Beasley, J. E. (1983). Route first-cluster second methods for vehicle routing. *Omega*, 11(4):403–408.

Bektaş, T. and Laporte, G. (2011). The pollution-routing problem. *Transportation Research Part B: Methodological*, 45(8):1232–1250.

Bishop, C. M. (2006). *Pattern recognition and machine learning.* springer.

Boyan, J. A. and Littman, M. L. (1994). Packet routing in dynamically changing networks: A reinforcement learning approach. In *Advances in neural information processing systems*, pages 671–678.

Bruni, M., Guerriero, F., and Beraldi, P. (2014). Designing robust routes for demand-responsive transport systems. *Transportation research part E: logistics and transportation review*, 70:1–16.

Caceres-Cruz, J., Arias, P., Guimarans, D., Riera, D., and Juan, A. A. (2014). Rich vehicle routing problem: Survey. *ACM Comput. Surv.*, 47(2):32:1–32:28.

Chen, M.-C., Hsiao, Y.-H., Reddy, H., and Tiwari, M. K. (2015). A particle swarm optimization approach for route planning with cross-docking. In *7th International Conference on Emerging Trends in Engineering & Technology (ICETET)*. IEEE.

Chen, M.-C., Hsiao, Y.-H., Reddy, R. H., and Tiwari, M. K. (2016). The self-learning particle swarm optimization approach for routing pickup and delivery of multiple products with material handling in multiple cross-docks. *Transportation Research Part E: Logistics and Transportation Review*, 91:208–226.

Ćirović, G., Pamučar, D., and Božanić, D. (2014). Green logistic vehicle routing problem: Routing light delivery vehicles in urban areas using a neuro-fuzzy model. *Expert Systems with Applications*, 41(9):4245–4258.

Clarke, G. and Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581.

Cordeau, J.-F., Desaulniers, G., Desrosiers, J., Solomon, M. M., Soumis, F., and GERAD (2000). *The VRP with time windows.* Groupe d'études et de recherche en analyse des décisions Montréal.

Crainic, T. G. and Montreuil, B. (2016). Physical internet enabled hyperconnected city logistics. *Transportation Research Procedia*, 12:383–398.

Dantzig, G. B. and Ramser, J. H. (1959). The truck dispatching problem. *Management Science*, 6(1):80–91.

Dell'Amico, M., Hadjicostantinou, E., Iori, M., and Novellani, S. (2014). The bike sharing rebalancing problem: Mathematical formulations and benchmark instances. *Omega*, 45:7–19.

Desrochers, M., Lenstra, J. K., and Savelsbergh, M. W. P. (1990). A classification scheme for vehicle routing and scheduling problems. *European Journal of Operational Research*, 46(3):322–332.

Di Prampero, P., Cortili, G., Mognoni, P., and Saibene, F. (1979). Equation of motion of a cyclist. *Journal of Applied Physiology*, 47(1):201–206.

Dondo, R. and Cerdá, J. (2014). A monolithic approach to vehicle routing and operations scheduling of a cross-dock system with multiple dock doors. *Computers & Chemical Engineering*, 63:184–205.

Dondo, R., Méndez, C. A., and Cerdá, J. (2011). The multi-echelon vehicle routing problem with cross docking in supply chain management. *Computers & Chemical Engineering*, 35(12):3002–3024.

Dudukovich, R., Hylton, A., and Papachristou, C. (2017). A machine learning concept for dtn routing. In *2017 IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE)*, pages 110–115. IEEE.

Ehrgott, M., Wang, J. Y., Raith, A., and Van Houtte, C. (2012). A bi-objective cyclist route choice model. *Transportation research part A: policy and practice*, 46(4):652–663.

Enderer, F., Contardo, C., and Contreras, I. (2017). Integrating dock-door assignment and vehicle routing with cross-docking. *Computers & Operations Research*, 88(Supplement C):30–43.

Erdoğan, S. and Miller-Hooks, E. (2012). A green vehicle routing problem. *Transportation Research Part E: Logistics and Transportation Review*, 48(1):100–114.

Fadlullah, Z. M., Tang, F., Mao, B., Kato, N., Akashi, O., Inoue, T., and Mizutani, K. (2017). State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems. *IEEE Communications Surveys & Tutorials*, 19(4):2432–2455.

García-Nájera, A. and López-Jaimes, A. (2018). An investigation into many-objective optimization on combinatorial problems - analyzing the pickup and delivery problem. *Swarm and Evolutionary Computation*, 38:218–230.

Ghiani, G., Manni, E., Quaranta, A., and Triki, C. (2009). Anticipatory algorithms for same-day courier dispatching. *Transportation Research Part E: Logistics and Transportation Review*, 45(1):96–106.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256.

Gontara, S., Boufaied, A., and Korbaa, O. (2018). Routing the pi-containers in the physical internet using the pi-bgp protocol. In *2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA)*, pages 1–8. IEEE.

Grappe, F., Candau, R., Belli, A., and Rouillon, J. D. (1997). Aerodynamic drag in field cycling with special reference to the obree's position. *Ergonomics*, 40(12):1299–1311.

Guo, Z., Sheikh, S., Al-Najjar, C., Kim, H., and Malakooti, B. (2010). Mobile ad hoc network proactive routing with delay prediction using neural network. *Wireless Networks*, 16(6):1601–1620.

Hà, M. H., Bostel, N., Langevin, A., and Rousseau, L. (2013). An exact algorithm and a metaheuristic for the multi-vehicle covering tour problem with a constraint on the number of vertices. *European Journal of Operational Research*, 226(2):211–220.

Hà, M. H., Bostel, N., Langevin, A., and Rousseau, L. (2014). An exact algorithm and a metaheuristic for the generalized vehicle routing problem with flexible fleet size. *Computers & Operations Rearch*, 43:9–19.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

Hedrick, C. L. (1988). Routing information protocol. Technical report, No. RFC 1058.

Hrncir, J., Zilecky, P., Song, Q., and Jakob, M. (2015). Speedups for multi-criteria urban bicycle routing. In *OASIcs-OpenAccess Series in Informatics*, volume 48. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Kara, I., Kara, B. Y., and Yetis, M. K. (2007). Energy minimizing vehicle routing problem. In *International Conference on Combinatorial Optimization and Applications*, pages 62–71. Springer.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Koning, M. and Conway, A. (2016). The good impacts of biking for goods: Lessons from Paris city. *Case studies on transport policy*, 4(4):259–268.

Kreng, V. B. and Chen, F.-T. (2008). The benefits of a cross-docking delivery strategy: a supply chain collaboration approach. *Production Planning and Control*, 19(3):229–241.

Kuo, R. J. and Zulvia, F. E. (2017). Hybrid genetic ant colony optimization algorithm for capacitated vehicle routing problem with fuzzy demand - a case study on garbage collection system. In *4th International Conference on Industrial Engineering and Applications (ICIEA)*, page 244–248.

Lahyani, R., Khemakhem, M., and Semet, F. (2015). Rich vehicle routing problems: From a taxonomy to a definition. *European Journal of Operational Research*, 241(1):1–14.

Lee, K., Chae, J., and Kim, J. (2019). A courier service with electric bicycles in an urban area: The case in seoul. *Sustainability*, 11(5):1255.

Li, H. and Lim, A. (2001). A metaheuristic for the pickup and delivery problem with time windows. In *Proceedings 13th IEEE International Conference on Tools with Artificial Intelligence. ICTAI 2001*, page 160–167.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Lin, C., Choy, K. L., Ho, G. T., Chung, S. H., and Lam, H. (2014). Survey of green vehicle routing problem: past and future trends. *Expert systems with applications*, 41(4):1118–1138.

Maes, J. and Vanelslander, T. (2012). The use of bicycle messengers in the logistics chain, concepts further revised. *Procedia-Social and behavioral sciences*, 39:409–423.

Maknoon, Y. and Laporte, G. (2017). Vehicle routing with cross-dock selection. *Computers & Operations Research*, (77):254–266.

Mao, B., Fadlullah, Z. M., Tang, F., Kato, N., Akashi, O., Inoue, T., and Mizutani, K. (2017). Routing or computing? the paradigm shift towards intelligent computer network packet transmission based on deep learning. *IEEE Transactions on Computers*, 66(11):1946–1960.

Mao, Q., Hu, F., and Hao, Q. (2018). Deep learning for intelligent wireless networks: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 20(4):2595–2621.

Martin, J. C., Milliken, D. L., Cobb, J. E., McFadden, K. L., and Coggan, A. R. (1998). Validation of a mathematical model for road cycling power. *Journal of applied biomechanics*, 14(3):276–291.

Meller, R. D., Montreuil, B., Thivierge, C., and Montreuil, Z. (2012). Functional design of physical internet facilities: A road-based transit center. Technical report, Progress in Material Handling Research 2012.

Miao, Z., Fu, K., and Yang, F. (2012). A hybrid genetic algorithm for the multiple crossdocks problem. *Mathematical Problems in Engineering*.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.

Montoya-Torres, J. R., Franco, J. L., Isaza, S. N., Jiménez, H. F., and Herazo-Padilla, N. (2015). A literature review on the vehicle routing problem with multiple depots. *Computers & Industrial Engineering*, 79:115–129.

Montreuil, B. (2011). Towards a physical internet: Meeting the global logistics sustainability grand challenge. In *Logistics Res., vol. 3, nos. 2-3, pp. 71-87, 2011.*

Montreuil, B. (2012). Physical internet manifesto. In *Transforming the way physical objects are moved, stored, realized, supplied and used, aiming towards greater efficiency and sustainability.*

Montreuil, B., Ballot, E., and Fontane, F. (2012). An open logistics interconnection model for the physical internet. In *14th IFAC Symposium on Information Control Problems in Manufacturing.*

Moy, J. (1997). Ospf version 2. Technical report, No. RFC 2178.

Nagy, G. and Salhi, S. (2005). Heuristic algorithms for single and multiple depot vehicle routing problems with pickups and deliveries. *European Journal of Operational Research*, 162(1):126–141.

Nazari, M., Oroojlooy, A., Snyder, L., and Takác, M. (2018). Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*, pages 9839–9849.

Nikolopoulou, A. I., Repoussis, P. P., Tarantilis, C. D., and Zachariadis, E. E. (2017). Moving products between location pairs: Cross-docking versus direct-shipping. *European Journal of Operational Research*, 256(3):803–819.

Núñez, A., Cortés, C. E., Sáez, D., Schutter, B. D., and Gendreau, M. (2014). Multiobjective model predictive control for dynamic pickup and delivery problems. *Control Engineering Practice*, 32:73–86.

Oyola, J., Arntzen, H., and Woodruff, D. L. (2016). The stochastic vehicle routing problem, a literature review, part i: models. *EURO Journal on Transportation and Logistics*, page 1–29.

Oyola, J., Arntzen, H., and Woodruff, D. L. (2017). The stochastic vehicle routing problem, a literature review, part ii: solution methods. *EURO Journal on Transportation and Logistics*, 6(4):349–388.

Pellegrini, P., Favaretto, D., and Moretti, E. (2007). *Multiple Ant Colony Optimization for a Rich Vehicle Routing Problem: A Case Study*, pages 627–634. Springer Berlin Heidelberg.

Pillac, V., Gendreau, M., Guéret, C., and Medaglia, A. L. (2013). A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1):1–11.

Pisinger, D. and Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403–2435.

Prins, C. (2004). A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 31(12):1985–2002.

Prins, C. (2009). A grasp$\times$ evolutionary local search hybrid for the vehicle routing problem. In *Bio-inspired algorithms for the vehicle routing problem*, pages 35–53. Springer.

Psaraftis, H. N., Wen, M., and Kontovas, C. A. (2016). Dynamic vehicle routing problems: Three decades and counting. *Networks*, 67(1):3–31.

Ritzinger, U., Puchinger, J., and F. Hartl, R. (2016). A survey on dynamic and stochastic vehicle routing problems. *International Journal of Production Research*, 54(1):215–231.

Ross, M. (1997). Fuel efficiency and the physics of automobiles. *Contemporary Physics*, 38(6):381–394.

Russell, R., Chiang, W.-C., and Zepeda, D. (2008). Integrating multi-product production and distribution in newspaper logistics. *Computers & Operations Research*, 35(5):1576–1588. Part Special Issue: Algorithms and Computational Methods in Feasibility and Infeasibility.

Sarraj, R. (2013). *Interconnexion des reseaux logistiques : elements de definition et potentiel*. PhD thesis, Ecole Nationale Superieure des Mines de Paris.

Sarraj, R., Ballot, E., Pan, S., Hakimi, D., and Montreuil, B. (2013). Interconnected logistic networks and protocols: simulation-based efficiency assessment. *International Journal of Production Research*, 52(11):3185–3208.

Sarraj, R., Ballot, E., Pan, S., Hakimi, D., and Montreuil, B. (2014). Interconnected logistic networks and protocols: simulation-based efficiency assessment. *International Journal of Production Research*, 52(11):3185–3208.

Sarraj, R., Ballot, E., Pan, S., and Montreuil, B. (2012). Analogies between internet network and logistics service networks: challenges involved in the interconnection. *Journal of Intelligent Manufacturing*, 25(6):1207–1219.

Serna, M. D. A., Adarme-Jaimes, W., and Cortés, J. A. Z. (2010). Commodities distribution using alternative types of transport. a study in the colombian bread smes. *Dyna*, 77(163):222–233.

Shi, X., Zhao, F., and Gong, Y. (2009). Genetic algorithm for the one-commodity pickup-and-delivery vehicle routing problem. In *Intelligent Computing and Intelligent Systems, 2009. ICIS 2009. IEEE International Conference on*, volume 1, page 175–179. IEEE.

Silbernagl, D., Krismer, N., Malfertheiner, M., and Specht, G. (2016). Optimization of digital elevation models for routing. In *GvD*, pages 103–108.

Silva, P. P. B. and Zuluaga, A. E. (2016). Review of state of the art vehicle routing problem with pickup and delivery (vrppd). *Ingeniería y Desarrollo*, 34(2):463–482.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484.

SINTEF. Li&lim benchmark. `https://www.sintef.no/projectweb/top/pdptw/li-lim-benchmark/`. Accessed: 2018-09-10.

Solomon, M. M. (1985). Algorithms for the vehicle routing and scheduling problems with time window constraints. In *Operations Research*, number 254-265.

Song, Q., Zilecky, P., Jakob, M., and Hrncir, J. (2014). Exploring pareto routes in multi-criteria urban bicycle routing. In *Intelligent Transportation Systems (ITSC), 2014 IEEE 17th International Conference on*, pages 1781–1787. IEEE.

Stampa, G., Arias, M., Sanchez-Charles, D., Muntés-Mulero, V., and Cabellos, A. (2017). A deep-reinforcement learning approach for software-defined networking routing optimization. *arXiv preprint arXiv:1709.07080.*

Tang, F., Mao, B., Fadlullah, Z. M., Kato, N., Akashi, O., Inoue, T., and Mizutani, K. (2018). On removing routing protocol from future wireless networks: A real-time deep learning approach for intelligent traffic control. *IEEE Wireless Communications*, 25(1):154–160.

Tipagornwong, C. and Figliozzi, M. (2014). Analysis of competitiveness of freight tricycle delivery services in urban areas. *Transportation Research Record*, 2410(1):76–84.

Toh, C. K. (2001). *Ad hoc mobile wireless networks: protocols and systems.* Pearson Education.

Touihri, A., Dridi, O., and Krichen, S. (2017). A multi operator genetic algorithm for solving the capacitated vehicle routing problem with cross-docking problem. In *Computational Intelligence (SSCI), 2016.* IEEE.

Vidal, T. (2016). Split algorithm in O(n) for the capacitated vehicle routing problem. *Computers & Operations Research*, 69:40–47.

Vidal, T., Crainic, T. G., Gendreau, M., and Prins, C. (2014). A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research*, 234(3):658–673.

Wang, C., Mu, D., Zhao, F., and Sutherland, J. W. (2015). A parallel simulated annealing method for the vehicle routing problem with simultaneous pickup–delivery and time windows. *Computers & Industrial Engineering*, 83:111–122.

Wang, J., Jagannathan, A. K. R., Zuo, X., and Murray, C. C. (2017). Two-layer simulated annealing and tabu search heuristics for a vehicle routing problem with cross docks and split deliveries. *Computers & Industrial Engineering*, 112:84–98.

Wang, M., Cui, Y., Wang, X., Xiao, S., and Jiang, J. (2018). Machine learning for networking: Workflow, advances and opportunities. *IEEE Network*, 32(2):92–99.

Wang, S. and Wu, Y. (2017). A genetic algorithm for energy minimization vehicle routing problem. In *2017 International Conference on Service Systems and Service Management*, page 1–5.

Wang, Y., Martonosi, M., and Peh, L.-S. (2007). Predicting link quality using supervised learning in wireless sensor networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 11(3):71–83.

Wassan, N. A. and Nagy, G. (2014). Vehicle routing problem with deliveries and pickups: modelling issues and meta-heuristics solution approaches. *International Journal of Transportation*, 2(1):95–110.

Wilson, N. H. M. and Colvin, N. J. (1977). *Computer control of the Rochester dial-a-ride system*. Number 77. Massachusetts Institute of Technology, Center for Transportation Studies.

Yang, Y., Ma, X., Sun, Y., and Zhu, Z. (2017). Multi-objective memetic algorithm based on three-dimensional request prediction for dynamic pickup-and-delivery problem with time windows. In Shi, Y., Tan, K. C., Zhang, M., Tang, K., Li, X., Zhang, Q., Tan, Y., Middendorf, M., and Jin, Y., editors, *Simulated Evolution and Learning*, page 810–820, Cham. Springer International Publishing.

You, X., Li, X., Xu, Y., Feng, H., and Zhao, J. (2019). Toward packet routing with fully-distributed multi-agent deep reinforcement learning. *arXiv preprint arXiv:1905.03494*.

Yu, V. F., Jewpanya, P., and Kachitvichyanukul, V. (2015). Particle swarm optimization for the multi-period cross-docking distribution problem with time windows. *International Journal of Production Research*, 54(2):509–525.

Yu, V. F., Jewpanya, P., and Redi, A. P. (2016). Open vehicle routing problem with cross-docking. *Computers & Industrial Engineering*, 94:6–17.

Zachariadis, E., D. Tarantilis, C., and Kiranoudis, C. (2010). An adaptive memory methodology for the vehicle routing problem with simultaneous pick-ups and deliveries. *European Journal of Operational Research*, 202:401–411.

Zanjireh, M. M. and Larijani, H. (2015). A survey on centralised and distributed clustering routing algorithms for wsns. In *2015 IEEE 81st Vehicular Technology Conference (VTC Spring)*, pages 1–6. IEEE.

Zanjireh, M. M., Shahrabi, A., and Larijani, H. (2013). Anch: A new clustering algorithm for wireless sensor networks. In *2013 27th International Conference on Advanced Information Networking and Applications Workshops*, pages 450–455. IEEE.

Appendices

Appendix A

Publications resulting from this thesis

Refereed or submitted journal papers

[1] Y. Ancele, H. H. Minh, C. Lersteau, D. B. Matellini, and T. T. Nguyen. Toward a more flexible vrp with pickups and deliveries allowing consolidations. Transportation Research Part C: Emerging Technologies, 2019.

[2] Y. Ancele, H. H. Minh, D. B. Matellini, and T. T. Nguyen. Bike routing problem with energy constraints. European Journal of Operational Research, 2020.

Future journal submissions

[3] Y. Ancele, D. B. Matellini, and T. T. Nguyen. A learning algorithm for the vehicle and container routing problem. IEEE Access, 2020.

[4] Y. Ancele, D. B. Matellini, and T. T. Nguyen. A rich multi-objective and dynamic VRP with uncertainty.

The following lists materials (or part) of the publications presented in the thesis:

- Chapter 2 : publications [1,2,3,4]
- Chapter 3 : publication [2]
- Chapter 4 : publications [1]

- Chapter 5 : publication [4]
- Chapter 6 : publication [3]

Appendix B

Algorithms for functions and operators of meta-heuristics

Algorithm 23: Initialisation function

```
//**Input:** set $C$ of P&D customers, set $K$ of vehicles;
//**Output:** an initial solution $S$;
add_depots($S$) //add depot nodes for all vehicle routes;
sort_nodes($C$) //sort P&D pairs by $B_i$;
$node \leftarrow$ pop($C$) //remove first node;
$k \leftarrow$ random($K$);
while $node \neq \varnothing$ do
    $indices \leftarrow$ get_best_indices($k, S, node$);
    foreach $i$ in $indices$ do
        if insert($S^k, node, i$) then
            $inserted \leftarrow true$;
            if is_delivery($node$) then
                $pickup \leftarrow \varnothing$;
                $k \leftarrow$ random($K$);
            else
                $pickup \leftarrow node$;
            break;
    if not $inserted$ then
        add($C, node$);
        if $pickup \neq \varnothing$ then
            remove($S^k, pickup$) //remove the pickup node as the delivery one failed;
            add($C, pickup$);
            $pickup \leftarrow \varnothing$;
        $k \leftarrow$ random($K$);
    $node \leftarrow$ pop($C$);
```

Algorithm 24: Neighbour Search function

```
//Input: a solution $S$;
//Output: the best solution $S_b$ found;
$S_b \leftarrow S$;
while $true$ do
    $S' \leftarrow$ PD_operate($S_b$) //apply a random operator from op_list;
    if $cost(S') < cost(S_b)$ then
        $S_b \leftarrow S'$;
    else
        break;
return $S_b$;
```

Algorithm 25: Random Solution function

```
//Input: a solution $S$;
//Output: a solution $S'$ found;
$it \leftarrow 0$;
$S' \leftarrow \varnothing$;
while $S' = \varnothing$ and $it < RIT$ do
    $S' \leftarrow$ PD_operate($S$);
    $it \leftarrow it + 1$;
if $S' = \varnothing$ then
    $S' \leftarrow$ random_previous_solution();
return $S'$;
```

Algorithm 26: Simulated Annealing function

```
//Input: a solution $S$;
//Output: a solution $S'$ found;
$f \leftarrow false$;
$t \leftarrow T0$;
while $f = false$ do
    $S' \leftarrow$ random_solution($S$);
    $\Delta \leftarrow$ cost($S'$) $-$ cost($S$);
    if $\Delta \leq 0$ then
        $p \leftarrow 1$;
    else
        $p \leftarrow e^{-\Delta/T}$;
    $t \leftarrow \delta * T$;
    if random_double() $\leq p$ then
        $f \leftarrow true$;
    else if $t < 0.01$ then
        $f \leftarrow true$;
        $S' \leftarrow S$;
return $S'$;
```

Algorithm 27: PD Interchange operator

```
//Input: a current solution $S_c$, a set $K$ of vehicles;
//Output: a solution $S$ found;
$valid \leftarrow false$;
while $valid = false$ do
    $S \leftarrow S_c$;
    $k \leftarrow$ random($K$);
    $node \leftarrow$ remove($S^k$) //remove a random node;
    if insert($node, S^k$) then
        $valid \leftarrow$ is_valid($S$);
return $S$;
```

Algorithm 28: PD Move operator

```
//Input: a current solution $S_c$, a set $K$ of vehicles, a set $R$ of requests;
//Output: a solution $S$ found;
$valid \leftarrow false$;
shuffle($K$);
$r \leftarrow$ random($R$);
$v \leftarrow \varnothing$;
$changed \leftarrow 0$;
while $valid = false$ do
    $S \leftarrow S_c$;
    foreach $k$ in $K$ do
        foreach $node$ in $S^k$ do
            //cross-docks can contain several r;
            if contain($node, r$) then
                if $v = \varnothing$ then
                    $v \leftarrow$ random($K/k$) //new vehicle route $v$ must be different from $k$;
                remove($node, r, S^k$);
                if not insert($node, r, S^v$) then
                    break 2 loops;
                $changed \leftarrow changed + 1$;
            if $changed = 2$ then
                $valid \leftarrow$ is_valid($S$);
                break 2 loops;
return $S$;
```

Algorithm 29: PD Swap operator

```
1  //Input: a solution $S$, a set $K$ of vehicles, a set $R$ of requests;
2  //Output: a solution $S'$ found;
3  $valid \leftarrow false$;
4  while $valid = false$ do
5      $S' \leftarrow \varnothing$;
6      $v_1 \leftarrow$ random($K$);
7      $v_2 \leftarrow$ random($K/v_1$) //vehicle $v_2$ must be different from $v_1$;
8      foreach $k$ in $K$ do
9          if $k = v_1$ then
10             $S'_k \leftarrow S_{v_2}$ //the depots are those from vehicle $v_1$;
11         else if $k = v_2$ then
12             $S'_k \leftarrow S_{v_1}$ //the depots are those from vehicle $v_2$;
13         else
14             $S'_k \leftarrow S_k$;
15     $valid \leftarrow$ is_valid($S'$);
16 return $S'$;
```

Algorithm 30: PD Exchange operator

```
//**Input:** a current solution $S_c$, a set $K$ of vehicles, a set $R$ of requests;
//**Output:** a solution $S$ found;
$valid \leftarrow false$;
shuffle(K);
while $valid = false$ do
    $r_1 \leftarrow$ random($R$);
    $r_2 \leftarrow$ random($R/r_1$);
    $node_{1a} \leftarrow \varnothing$;
    $node_{1b} \leftarrow \varnothing$;
    $node_{2a} \leftarrow \varnothing$;
    $node_{2b} \leftarrow \varnothing$;
    $S \leftarrow S_c$;
    foreach $k$ in $K$ do
        foreach $node$ in $S^k$ do
            if contain($node, r_1$) and ($node_{1a} = \varnothing$ or $node_{1b} = \varnothing$) then
                $k_1 \leftarrow k$;
                if $node_{1a} = \varnothing$ then
                    $node_{1a} \leftarrow node$;
                else
                    $node_{1b} \leftarrow node$;
                remove($node, r_1, S^k$);
            if contain($node, r_2$) and ($node_{2a} = \varnothing$ or $node_{2b} = \varnothing$) then
                $k_2 \leftarrow k$;
                if $node_{2a} = \varnothing$ then
                    $node_{2a} \leftarrow node$;
                else
                    $node_{2b} \leftarrow node$;
                remove($node, r_2, S^k$);
    if contain($S^{k_1}, r_2$) or contain($S^{k_2}, r_1$) then
        continue;
    insert($node_{1a}, r_1, S^{k_2}$);
    insert($node_{1b}, r_1, S^{k_2}$);
    insert($node_{2a}, r_2, S^{k_1}$);
    insert($node_{2b}, r_2, S^{k_1}$);
    $valid \leftarrow$ is_valid($S$);
return $S$;
```

Algorithm 31: Consolidation function

```
1  //Input: a current solution $S_c$;
2  //Output: the best solution $S_b$ found;
3  $S_b \leftarrow \varnothing$;
4  $S \leftarrow S_c$;
5  if random_double() $< 0.5$ then
6      $S \leftarrow$ PD_arrange($S$);
7  group $\leftarrow$ random(req_groups);
8  while no_progress $<$ RPLI do
9      if random_double() $<$ RPLR then
10         operator $\leftarrow$ PD_stretch;
11     else
12         operator $\leftarrow$ PD_shrink;
13     cnt $\leftarrow 0$;
14     foreach $r$ in group do
15         $S \leftarrow$ operator($S, r$) //apply PD_stretch() or PD_shrink();
16     if is_valid($S$) and cost($S$) $<$ cost($S_b$) then
17         $S_b \leftarrow S$;
18         no_progress $\leftarrow 0$;
19         group $\leftarrow$ random(req_groups);
20     else
21         no_progress $\leftarrow$ no_progress $+ 1$;
22 return $S_b$;
```

Appendix C

Algorithms for Pareto optimisation

Algorithm 32: Pareto dominance function

```
//Input: chromosomes $c_1, c_2$, objective set $O$;
//Output: the dominance relation $r$;
$better \leftarrow 0$;
$worse \leftarrow 0$;
foreach $obj$ in $O$ do
    if $\texttt{cost}(obj, c_1) < \texttt{cost}(obj, c_2)$ then
        $better \leftarrow better + 1$;
    else if $\texttt{cost}(obj, c_1) > \texttt{cost}(obj, c_2)$ then
        $worse \leftarrow worse + 1$;
if $better = |O|$ then
    $r \leftarrow pareto.dominates$;
else if $worse = |O|$ then
    $r \leftarrow pareto.dominated$;
else if $better > 0$ and $worse = 0$ then
    $r \leftarrow pareto.dominates$;
else if $worse > 0$ and $better = 0$ then
    $r \leftarrow pareto.dominated$;
else if $worse = 0$ and $better = 0$ then
    $r \leftarrow pareto.equal$;
else
    $r \leftarrow pareto.equivalent$;
return $r$;
```

Algorithm 33: Pareto front function

```
1  //Input: set of chromosomes $C$;
2  //Output: a Pareto front $P$;
3  foreach $c$ in $C$ do
4      foreach $p$ in $P$ do
5          if pareto_dominance$(c, p) = pareto.better$ then
6              mark $p$ to be removed from $P$;
7              mark $c$ to be added in $P$;
8          else if pareto_dominance$(c, p) = pareto.worse$ then
9              unmark $c$;
10             break;
11         else
12             mark $c$ to be added in $P$;
13     manage marked solutions;
14 return $P$;
```

Algorithm 34: Pareto ranking function

```
1  //Input: population of chromosomes $C$;
2  //Output: the Pareto ranking $pr$ of $C$;
3  $i \leftarrow 1$;
4  while $C \neq \varnothing$ do
5      $front \leftarrow$ pareto_front$(C)$ //remove the resulting front from the given set $C$;
6      foreach $c$ in $front$ do
7          $pr[c] \leftarrow i$;
8      $i \leftarrow i + 1$;
9  return $pr$;
```