INSTITUTO
UNIVERSITÁRIO
DE LISBOA

Optimization of Public Transport Networks
Reinforcement Learning for Smart Mobility

António Luís Barros Mota

Master in Data Science,

Supervisor:
Professor Diana Elisabeta Aldea Mendes (PhD, Associate Professor)
ISCTE – University Institute of Lisboa

September, 2024

BUSINESS SCHOOL

TECNOLOGIAS E ARQUITETURA

Departamento de Métodos Quantitativos para Gestão e Economia

Departamento de Ciências e Tecnologia da Informação

Optimization of Public Transport Networks

Reinforcement Learning for Smart Mobility

António Luís Barros Mota

Master in Data Science,

Supervisor:
Professor Diana Elisabeta Aldea Mendes (PhD, Associate Professor)
ISCTE – University Institute of Lisbon

Acknowledgments

This work could not have been completed without the invaluable support of Matilde Caldas, my everyday partner. Despite being from a very different scientific field, she listened attentively, asking insightful questions that, by challenging my work, ultimately strengthened it. Her moral support and unparalleled ability to help me organize my thoughts do not surprise me, as they have always, always, been there. I am also grateful for her willingness to review the text. I want to express my heartfelt gratitude to Professor Nuno R. Faria, from the Imperial College London, for his time and support in the final stages and for helping me raise the bar for this work. My special thanks to Federico Berto, from the Korea Advanced Institute of Science & Technology (KAIST), for his generosity in assisting me with the more technical challenges I faced. Finally, I am grateful to Professor Diana Mendes for her positive and constructive guidance throughout the process and her enthusiasm during our discussions and debates, which helped guide me in the right direction.

Resumo

Desenhar uma rede pública de transportes eficiente é um problema complexo (NP-difícil) que envolve, entre outros, a escolha de paragens, dos melhores trajectos, a definição de horários e frequências, tendo em conta múltiplos factores em conflito. Dada a complexidade, a busca por soluções óptimas é preterida em favor do uso de métodos heurísticos (regras gerais de decisão) e recurso a conhecimento especializado, que permitem encontrar soluções satisfatórias. Esta dissertação foca-se na otimização da rede de transportes da Carris, em Lisboa, explorando a aplicação de mecanismos de aprendizagem reforçada (AR) para resolver uma parte deste problema: encontrar melhores trajectos entre várias paragens, servidas por um número variável de linhas – um problema de roteamento de veículos. Através do algoritmo *Multi-task Vehicle Routing Solver with Mixture-of-Experts*, treinaram-se dois modelos. Os resultados foram comparados com os da rede Carris e com os do algoritmo de economias de Clarke e Wright. A AR demonstra potencial para "aprender" boas heurísticas e encontrar melhores soluções que as atuais, tendo o modelo minimizado a distância em linha recta do menor troço da rede. No entanto, a complexidade da mobilidade urbana é ainda um desafio, tendo sido necessário efectuar simplificações para modelar este problema. Apesar das limitações, tais como os baixos recursos computacionais e a natureza estática dos dados, esta análise demonstra que através da integração da informação de trânsito e do desenvolvimento de algoritmos mais abrangentes, a AR tem o potencial de melhorar a eficiência destas redes e construir soluções que se ajustem dinamicamente aos diferentes constrangimentos diários.

Palavras-chave

Aprendizagem Reforçada; VRP; Lisboa; Carris; Mobilidade Inteligente; Redes de Transporte Público

Classificação JEL

C61; R41

Abstract

Designing an efficient public transport network is a complex problem (NP-hard) that involves, among other factors, selecting stops, determining the most optimal routes, and defining schedules and frequencies, all while considering multiple conflicting factors. Given this complexity, the pursuit of optimal solutions is often set aside in favor of heuristic methods (general decision rules) and expert knowledge, which allow for identifying satisfactory solutions. This dissertation focuses on optimizing Lisbon's Carris public transport network by exploring the application of reinforcement learning (RL) mechanisms to address part of this problem: finding more optimal routes between several stops served by a variable number of lines – a vehicle routing problem. Two models were trained using the Multi-task Vehicle Routing Solver with Mixture-of-Experts algorithm. The results were compared with the Carris network and the results given by the Clarke and Wright Savings algorithm. RL shows potential in learning good heuristics and finding better solutions than the current ones, as the model minimized the straight-line distance of the shortest segment of the network. However, the complexity of urban mobility remains a challenge, requiring simplifications to model this problem effectively. Despite limitations such as low computational resources and the static nature of the data, this analysis demonstrates that by integrating traffic information and developing more comprehensive algorithms, RL can improve the efficiency of these networks and create solutions that dynamically adjust to different daily constraints.

Keywords

Reinforcement Learning; VRP; Lisbon; Carris; Smart Mobility; Public Transit Networks

JEL Classification

C61; R41

Table of Contents

List of Figures and Tables

Glossary

Glossary of Acronyms

AGI – Artificial General Intelligence

CML – Lisbon City Hall

CNN – Convolutional Neural Network

CVRP – Capacitated Vehicle Routing Problem

DNN – Deep Neural Network

DQN – Deep Q-Network

GAT – Graph Attention Network

INE – National Statistics Institute

MDP – Markov Decision Process

MVMoE – Multi-task Vehicle Routing Solver with Mixture-of-Experts

OP – Orienteering Problem

PCTSP – Prize Collecting Travelling Salesman Problem

RL – Reinforcement Learning

RL4CO – Reinforcement Learning for Combinatorial Optimization

RMSE – Root Mean Square Error

RNN – Recurrent Neural Network

SVM – Support Vector Machines

TNDFSP – Transit Route Network Design and Frequency Setting Problem

TRNDP – Transit Route Network Design Problem

TSP – Travelling Salesman Problem

VRP – Vehicle Routing Problem

VRPPD – Vehicle Routing Problem with Pickups and Deliveries

VRPTW – Vehicle Routing Problems with Time Windows

1 Introduction

The development of artificial general intelligence (AGI)[1] represents the goal of machine learning, a topic of discussion since at least the mid-20th century[2]. Today, with the rapid advancement of technology, increased computational capacity (in terms of speed, memory, and infrastructure), and the creation of complex algorithms that mimic specific biological processes, coupled – it must be admitted – with some hype[3] surrounding technology and its potential, the argument that the human brain is nothing more than a complex machine has gained traction. In this view, emotions, consciousness (and even morality and culture) are seen as emergent properties arising from millions of interactions between neurons, which could also potentially emerge from artificial neural networks[4].

Since the 1970s, the observation that complex behaviors can emerge from simple and limited (i.e., localized) rules[5] strengthens this line of reasoning. It demonstrates how non-deterministic properties (or at least those we cannot predict) can arise from combining a few deterministic rules. On the other hand, the significant gaps in the understanding of the processes that underpin intelligence or creativity, and the fact that artificial neural networks function very differently from the human brain[6], suggest that we might be "building a plane, not a bird", implying that current technology may not be capable of producing true intelligence.

While the pursuit of AGI is the subject of intense philosophical, ethical, moral, economic, and technical debate, studies addressing portions of this problem are already well underway. These studies involve the use of statistical tools and learning methods – supervised (through examples), unsupervised (through statistical patterns in data), and reinforced (through trial and error) – to model different aspects of human intelligence and learning, in various deep neural networks[7]. In a now-classic example, deep

[1] A human-level intelligence capable of performing multiple clearly distinct tasks.

[2] Consider, for instance, the famous article "Computing Machinery and Intelligence" by Alan Turing, published in 1950. In it, the author develops the "imitation game" to determine whether a machine can think or not.

[3] (Eric Rosenbaum, 2023)

[4] David Deutsch is one of the thinkers who supports this thesis: "[Douglas Hofstadter and Lady Lovelace] share the wrong premise that lower-level computational steps cannot in any way ascend to a higher-level 'self' that affects everything" (Deutsch, 2013, p. 208).

[5] One may note, for example, the behavior of the "agents" in John Conway's "Game of Life" or the development of fractal structures.

[6] Take note of the (much) greater efficiency of the human brain compared to the current energy expenditures required to train large language models or the fact that these models need vast amounts of data for training.

[7] In particular, several neural network architectures, such as Feedforward Neural Networks (FNN), Recurrent Neural Networks (RNN), and Convolutional Neural Networks (CNNs). More recently, the Transformer, an architecture based on attention mechanisms, has garnered media interest due to its versatility (as it is capable of capturing dependencies at various distances in sequential information) and has brought about progress and some unification among different architectures.

neural networks learned to identify new, previously unseen examples of handwritten digits by using a labeled dataset of thousands of handwritten digit images. Closer to the topic of this thesis, neural networks have learned to play the game of Go – which has an enormous state-space, much larger than chess – defeating one of the top players in 2016.

The various architectures developed have impacted domains as varied as image classification, object detection, speech recognition, text translation, and speech and image generation. They have also shown promise in fields like medical diagnostics, pharmaceutical research, autonomous driving, and climate change mitigation, among others.

Initially focused on understanding images, text, and sound and processes like translation, machine learning is now shifting its focus toward handling vast amounts of data in more meaningful ways. It is moving from data description, prediction, and generation to the prescription of solutions, an area where Reinforcement Learning (RL) – the method of machine learning on which this thesis will focus – plays a crucial role.

Despite significant advances, and although some claim that some forms of learning (such as reward-based learning) can generalize and replace others[8], it could be argued that the type of learning currently being mimicked is limited and does not encompass the full complexity of human learning. Moreover, these algorithms' technical[9], social[10], and implementation challenges suggest that it may take considerable time before these tools can be widely implemented (e.g., in automating repetitive tasks, performing artistic activities, or making professional, legal, or political decisions without human intervention).

The discussion initiated by Wagstaff, (2010) is particularly relevant in this context. The mechanisms developed in RL models present additional challenges for evaluation and implementation, not only due to the technical aspects of this class of algorithms but also because of safety concerns when these algorithms interact with the real world (Whittlestone *et al*., 2021). This is because these algorithms are designed for decision-making, i.e., they are intended to choose actions from a set of possible options and act on the environment to achieve a particular goal, which may be complex. Implicitly in this design, there's the

[8] "We consider an alternative hypothesis: that the generic objective of maximizing reward is enough to drive behavior that exhibits most if not all abilities studied in natural and artificial intelligence" (Silver *et al*., 2021, p.1).

[9] Wagstaff, (2010) discusses the difficulty of implementing machine learning techniques and solutions outside academia. This article notes that much of the research remains purely academic, done in isolation (without contact with real-world problems), and often focuses only on 'theoretical' problems or uses standard datasets. A reflection of this is the frequent use of generalist metrics, such as root mean square error (RMSE), to evaluate model performance, which are not related to the specific problem being studied.

[10] "Artificial Intelligence (AI) entails a number of potential risks, such as opaque decision-making, gender-based or other kinds of discrimination, intrusion into our private lives, or being used for criminal purposes" (Nikolinakos, 2023, p.1).

assumption, for example, that a human objective can be reduced to a number – a value to be maximized (Silver *et al*., 2021). However, translating human desires or intentions into a mathematical framework is challenging. Thus, particular attention should be paid to testing and safety issues during deployment.

The technology holds immense potential, even when considered for local applications. These systems' capabilities can help humans understand many phenomena or assist in solving "smaller" or more localized problems, such as the one this dissertation aims to address.

1.1 Background and Motivation

The transportation of people and goods has become increasingly prevalent across all temporal and spatial scales (Brockmann *et al*., 2006). Over the past 20 years, the movement of people and goods has consistently grown, and mobility patterns have evolved across various scales (Gkiotsalitis, 2023).

In this context, designing more efficient transportation networks, particularly public ones, has gained increasing importance. The more efficient the network (concerning a specific factor), the more people it can serve[11] or the more goods it can distribute with fewer vehicles or shorter distances. The development of these networks thus has environmental impacts (such as reduced carbon emissions and the adoption of alternative fuels), economic impacts (such as reduced transportation costs), and social benefits (such as shorter travel times, reduced traffic, and fewer accidents), among others.

As mobility demands have grown, transport infrastructures – including roads, ports, bus stops, schedules, fleet management, and other elements – have necessarily developed alongside them. Given the economic utility and social benefits of such infrastructures (Pereira *et al*., 2022), as well as the complexity of their design[12] (Kepaptsoglou & Karlaftis, 2009), these transport networks must be carefully planned.

However, this planning is far from trivial (Wu *et al*., 2018). Not only are there conflicting interests – those who provide transportation aim to reduce costs while users desire faster services – but the problem of designing an optimal transport network is constantly and rapidly evolving. Modern societal digitization, the introduction of sensors in vehicles, autonomous vehicle development, shared mobility services, the rise of electric cars, and the necessary infrastructure to support them, as well as the integration of different public mobility modes, are all reshaping the way public transport networks are designed (Gkiotsalitis, 2023).

[11] Travel time, the number of transfers, and the lack of direct connections are some of the reasons car users do not use public transportation more often (CML, 2021).

[12] Many users, many operators, and many interconnected and contradictory objectives.

Furthermore, in designing these networks, one must consider future needs, as medium- and long-term trends also drive change. Recent studies (INE, 2023) show a shift in mobility patterns, with a greater focus on inter-municipal travel rather than intra-municipal travel, as was previously the case. In Lisbon, these studies reveal an expansion in travel patterns, with a less radial flow of people. This implies greater complexity in interurban mobility within the metropolitan area.

Buses play a pivotal role in developing these networks within the city or broader areas such as the Lisbon Metropolitan Area due to their greater flexibility compared to rail-based vehicles. This flexibility allows for adjustments to short- and long-term shifts in population movements, making buses an essential component of urban mobility planning.

1.2 Research Problem and Objective

The complexity of adapting a transportation network to meet current needs, in conjunction with the constantly evolving mobility demands[13], makes this an ideal problem for machine learning techniques. These algorithms can potentially reveal faster and more effective methods than those currently used.

This thesis will focus on applying RL to model a mobility problem known as the Vehicle Routing Problem (VRP), which is a logistics problem involving the determination of optimal routes for a fleet of vehicles to deliver goods or services while minimizing or maximizing a specific factor. The goal is to assess whether these algorithms are suitable for solving this class of problems.

Specifically, the VRP will be addressed at the municipal level[14], utilizing available data on Lisbon's public transport network, focusing on the Carris bus network. The thesis aims to evaluate the existing network and compare it with solutions generated by machine learning models.

1.3 Scope and Research Relevance

The quest for more efficient transportation networks falls under the domain of combinatorial optimization, a subfield of mathematical optimization[15] that involves finding an optimal solution from a discrete set of

[13] Many factors, such as the phenomenon of shared mobility, telecommuting, or the migration of people out of urban centers (as is happening in Lisbon, where jobs have traditionally been concentrated), have contributed and continue to contribute to changes in mobility patterns, generating new problems that must be solved daily if we are to have a more efficient transportation network (and more broadly, the distribution of goods and people).

[14] With its own characteristics compared to a VRP for problems with longer distances (for example, a VRP between cities), an idea that will be discussed later.

[15] "Mathematical optimization, also known as mathematical programming, is a field of mathematics that (...) concerns the characterization and search for minimizers or maximizers of a certain function within a certain set, generally defined by algebraic equations and inequalities" (translation from (Soares, 2005, p.1)). In the more

possible solutions. This area has seen rapid development since the 1950s, driven by the growth of public transport networks. However, traditional algorithms can only provide exact solutions for smaller problems, as the solution space grows exponentially with the number of points — making these problems NP-hard (Bengio *et al*., 2021).

This thesis hypothesizes that machine learning, particularly RL methods, can be applied to solve the problem at hand. These methods may help discover new decision heuristics for selecting optimal paths in specific instances of combinatorial optimization problems without expert intervention.

Given the growing need for broader algorithms capable of tackling diverse and evolving problems, developing more general algorithms using machine learning techniques is relevant. These algorithms can potentially solve a broader range of problems or find more efficient strategies for approaching optimization challenges.

1.4 Thesis Structure

In addition to this introduction, the thesis will present a *State-of-the-Art* section where the VRP and its solutions will be analyzed, along with the application of RL in this domain and the justification for its use over other learning methods. The *Methodology* section will discuss data collection and problem formulation, followed by the selection of specific RL algorithms for testing. In the *Experimental Setup* section, the environment and agent will be specified, along with the parameters and hyperparameters for the experiments. The *Results and Discussion* section will evaluate the trained models using performance metrics and reflect on their ability to solve a specific problem. Finally, the *Conclusion* will summarize the findings. After the Conclusion, the section *Data Availability* contains information regarding where to find other materials, considered important to the achievement of the results and conclusions presented during the work.

general case of optimization problems, the function to be optimized assumes values in the set of real numbers, and the set where the function is defined is continuous.

2 Literature Review

Given that Data Science is fundamentally a transdisciplinary field, any state-of-the-art review must focus not only on current research on the algorithms and analytical methods being employed but also on the specific domain being studied.

Data Science is a relatively recent field incorporating knowledge from disciplines like mathematics, probability, statistics, and computing – all of which intersect and influence each other[16]. So, it necessitates a solid foundation in basic concepts, which ensures that more complex explorations can be pursued without getting lost in the sea of information.

In this sense, a careful and thorough reading of seminal books on various topics was essential to establish a firm grounding in these foundational concepts.

Key works in this regard include "*Introdução à Probabilidade e à Estatística*" (Pestana & Velosa, 2008) and "The Elements of Statistical Learning" (Hastie *et al*., 2009), which were critical for a more technical review of the statistical principles underpinning all of Data Science. In particular, statistical methods such as regression, classification, or Markov decision processes, as well as statistical techniques like statistical regression, support vector machines (SVMs), and principal component analysis, were explored and expanded upon in these texts.

Additionally, books like "Reinforcement Learning: An Introduction" (Sutton & Barto, 2015), "Deep Reinforcement Learning" (Plaat, 2022), and "Algorithms for Reinforcement Learning" (Szepesvári, 2010) were crucial for consolidating the mathematical formalism behind RL algorithms. These texts provided a rigorous definition of key concepts, such as policy, reward signal, value function, and environment, among others. They also contributed to developing the intuition needed to understand how these algorithms operate, particularly regarding their mathematical conceptualization as Markov decision processes and their resolution via Bellman's equation.

Finally, the article "Human-level control through deep reinforcement learning" (Mnih *et al*., 2015) – a milestone in RL research[17] – offered a first glimpse into the operationalization of deep reinforcement learning approaches. This work highlighted how it is possible to combine various techniques within different architectures to achieve varied results. It also introduced the technical challenges involved in the

[16] Often, the problems identified in one field of study and the questions raised there determine the research direction in another field (Kuhn & Hacking, 2012).

[17] For instance, the creation of the first deep Q-networks, which combine deep neural networks and RL, allowed for the first time the same algorithm (i.e., identical architecture and hyperparameters) to perform multiple distinct tasks, such as playing various Atari games (Mnih *et al*., 2015).

development and practical application of these algorithms, showcasing an area ripe with potential and challenges.

The initial foray into the extensive bibliography on these topics revealed that, despite being a relatively young field, RL has attracted considerable attention in recent years. A preliminary analysis of published papers showed that around 79.000 RL-related papers have been published in Scopus since 2014. A search using the terms "reinforcement learning" and "combinatorial optimization" returned 538 results. The focus was therefore placed on finding more specific articles. A search with the terms "transit network" (related to the problem being studied) and "reinforcement learning" identified 75 papers since 2014; of these, 11 were selected and analyzed as they were closer to the initial research theme – the design of public transport networks. Additionally, the 24 articles identified by searching for "transit route network design problem" were pondered upon.

It should be noted that more than an exhaustive search and random selection of papers – and considering the scarcity of literature on the specific problem –, the focus was placed on a deep reading of a few critical articles on combinatorial optimization and VRPs, reinforcement learning, and graph neural networks. These articles were chosen based on recommendations from experts in the field, citation counts in Scopus, references in official European Union documents, and, most importantly, references found in the bibliographies of the articles read, which is, one could say, a more vertical than horizontal reading selection approach.

2.1 The Vehicle Routing Problem

Both combinatorial optimization and machine learning are vast areas of study with many open problems. This fact represents a challenge. On the one hand, many problems addressed by combinatorial optimization are complex, NP-hard problems for which exact algebraic solutions are generally considered impossible, as they often involve enormous solution spaces where a tree search is not achievable. On the other hand, the development of machine learning methods for solving such problems is still in its infancy, with the first developments of neural networks with specific architectures for working with graphs appearing in 2009, and only in 2017 the development of Graph Attention Networks (GAT) that improve previous models (Veličković *et al*., 2017)[18].

[18] It is also worth noting that, compared to the broader field of machine learning, RL is a relatively new topic that has only begun to gain significant traction in recent years.

Combinatorial optimization studies problems with some form of combinatorial structure, with graph structures[19] being one of the archetypal mathematical representations of this type of problem. Many combinatorial optimization problems deal with concepts typically associated with graphs, such as path, flow, direction, and cut, among others. As a result, problems across many scientific fields[20] – as well as many "mundane" problems encountered in daily life[21] – are combinatorial optimization problems and can be modeled by graphs. These graphs, with vertices and edges enriched with attributes or dimensions, effectively generalize their structure: "Instances of the same type of problem are solved again and again on a regular basis, maintaining the same combinatorial structure, but differing mainly in their data" (Dai *et al*., 2017, p.1). Studying this structure may, therefore, be fruitful in various domains, including the one addressed in this thesis.

Building an efficient and effective public transportation infrastructure capable of transporting people between locations in the shortest possible time while using the minimum number of resources (specifically, with the fewest vehicles in operation) is a combinatorial optimization problem. More specifically, it is a routing problem[22] and a complex topic under debate since the 1960s (Darwish *et al*., 2020). The design of a transport network that serves the entire population involves balancing the needs of multiple stakeholders with different interests and priorities. The algorithms developed for designing such networks must be such that they balance these competing goals[23]. Furthermore, these objectives often conflict with one another (Darwish *et al*., 2020); for example, operators aim to reduce operational costs (by minimizing the number of vehicles in operation at any given time), which inevitably increases user travel times.

[19] A graph is a mathematical structure composed of vertices (nodes) and edges (links) that connect them.

[20] Examples of application areas include chemistry (design of new molecules), sociology (study of networks), telecommunications or transport network management, scheduling and timetabling, modeling passenger movement, among others.

[21] Questions such as finding the best route, the shortest path, scheduling tasks, or distributing workloads are central to this field. This area of knowledge seeks to address practical issues that arise in day-to-day life and are intrinsically related to society's economic organization. The problems posed within this discipline are often straightforward to describe, such as: "What is the shortest path?", "How can we allocate people to available schedules?", "What load can I carry without exceeding my vehicle's capacity?", "What routes should mail carriers take?", or "How should bus stops be distributed across a city?"

[22] Eldrandaly *et al*. (2008) state that "routing problems can be classified into two main categories based on their objectives and complexity. The first category, path finding problems, includes a set of problems whose main objective is to get the shortest path. These problems are actually simple compared to other types of routing problems. The second category, tour construction problems, includes a set of problems that aim to build a complete tour in a given network" (p. 51).

[23] The different objectives considered in the design of these networks serve as benchmarks for optimizing the algorithms developed, since the algorithms must minimize or maximize values for each objective or a metric that encompasses them all.

The broader problem to be addressed in this thesis – the design of such networks – is referred to in the literature as the "Transit Route Network Design Problem" (TRNDP), "Network Design and Frequency Setting Problem" or a combination of these two names like "Transit Network Design and Frequency Setting Problem" (TNDFSP). To introduce this problem, the articles "Transit Route Network Design Problem: Review" (Kepaptsoglou & Karlaftis, 2009) and "Optimizing Public Bus Transit Networks Using Deep Reinforcement Learning" (Darwish *et al*., 2020) were fundamental, alongside other complementary works.

Kepaptsoglou & Karlaftis, (2009) provided a first introduction to the problem, characterized it, and distinguished and classified the different solutions found to date. The TNDFSP is generally divided into five stages:

- Route design
- Frequency setting for each route
- Timetable development
- Vehicle Scheduling
- Driver scheduling

Many studies consider each stage in isolation, focusing on developing solutions to specific parts of the more general problem. This is because each stage corresponds to a particular combinatorial optimization problem of NP-hard complexity[24] (Darwish *et al*., 2020). As Kepaptsoglou & Karlaftis, (2009) explain:

> The selection of an optimal route structure for a transit network of realistic size is a combinatorial problem of astronomical proportions. (…) while the selection of optimal frequencies in an existing network is, in general, a convex optimization problem[25], (…) the choice of routes is generally a nonconvex (even concave) optimization problem for which no simple procedure exists short of direct comparison of various local minima (p. 3)[26].

VRPs fall within the scope of the TNDFSP and represent a particular class of combinatorial optimization problems. They involve designing a set of routes between stops to serve a certain number of customers while optimizing a specific objective. This typically means minimizing or maximizing the cost of a particular variable to which the system is optimized.

[24] This implies that there is likely no procedure to precisely find the optimal routes (Sutton & Barto, 2015, p.295).

[25] Special class of mathematical optimization problems where the objective function is convex, and the feasible region is a convex set. Convexity ensures that any local minimum is also a global minimum, which makes these problems easier to solve compared to non-convex optimization problems.

[26] Quoting G. F. Newell: Institute of Transportation Studies, University of California.

Kepaptsoglou & Karlaftis, (2009) also conducted a meta-analysis of studies on the VRP, systematizing the different types of possible analyses. They propose classifying the analyses based on three dimensions: objectives, parameters, and methodology. Like all combinatorial optimization problems, VRPs consist of the following elements or dimensions: the objective, which is the variable to be optimized (it could be distance traveled, the maximum number of stops, the maximum distance between stops, operator profit, etc.); the parameter dimension, which divides studies based on the variables the algorithm aims to compute (such as route design or determining the frequency of each mode of transport) and the system constraints (e.g., a bus serving passengers has a maximum capacity that cannot be exceeded along its route, or one might require all routes between A and B to pass through C)[27]; and the methodology dimension, which categorizes studies by the type of algorithms and tools used to solve the specific problem.

Any VRP also involves defining an objective function, which is the formula that calculates the objective's value for each possible solution (e.g., the sum of each partial trip along a more extensive route).

Based on these concepts, a feasible solution satisfies all problem constraints, while an optimal solution is a feasible solution where the objective function value is the best possible – a global maximum or minimum (Google OR-Tools, 2024).

2.2 Approaches to the Vehicle Routing Problem

Various approaches to this problem have been developed over the years. Dai *et al*. (2017) note that there are generally three paradigms or approaches to solving the problems we aim to address: exact algorithms, approximate algorithms, and heuristics.

Several algorithms (e.g., Branch and Bound, Branch and Cut, Branch and Price) have been developed to find exact solutions to this problem. However, these algorithms are only practical for tiny instances or straightforward cases of the problem. Approximate algorithms guarantee that the problem can be solved in polynomial time[28]. Still, these are not always applicable to the situation and often do not guarantee "desired optimality" (weak optimality guarantees). Heuristic approaches have been developed for cases where the solution space is too ample to find an exact solution. While these approaches do not guarantee

[27] These operational and environmental variables affect and constrain the network and are often predefined from the outset. Common constraints in these problems include the number of routes, the maximum length of each route, the number of available vehicles, their capacity, and the permissible frequencies or schedules. Due to their significance, some of these constraints have led to the development of separate problems, such as the Capacitated Vehicle Routing Problem (CVRP) or the Vehicle Routing Problems with Time Windows (VRPTW).

[28] An algorithm is considered polynomial if the time it takes to find a solution for an input of size n is less than a polynomial of degree n (i.e., less than Cn^k, where C and k are constants).

that the solution found will be optimal, they allow for satisfactory solutions to be found quickly, which is often sufficient for practical applications. Heuristic methods are based on rules for choosing one path over another (e.g., always selecting the nearest point), reducing the space of viable solutions, and enabling tree searches in smaller spaces[29]. Meta-heuristic methods, including constructing and improving a possible solution through local searches (specifically avoiding or escaping non-global minima), such as genetic algorithms or particle swarm optimization, have also been developed (Nazari *et al*., 2018).

Regarding methodology, many approaches still involve heuristic decision-making and rules based on expert knowledge, requiring a deep understanding of the specific transportation system being optimized (Darwish *et al*., 2020). Consequently, they are often not generalizable or scalable, as they require significant human intervention and expertise about the particular problem at hand[30] and are still insufficiently robust (i.e., the models may be affected by small changes).

More recently, methods based on machine learning have been developed, where the algorithm "learns" better decision rules. Attempts to use supervised learning to solve this problem can be found in the literature. However, this approach faces challenges: on the one hand, it requires previously solved VRP problems (which can be expensive), and on the other, it seems contradictory, as solving the problem is precisely what is being attempted. Berto *et al*., (2023) highlight these same ideas: "Although supervised learning methods are shown to be effective in [Neural Combinatorial Optimization], they require the availability of high-quality solutions, which is unrealistic for large instances or theoretically hard problems" (p.2).

Therefore, an RL approach is more promising. Interpreted as decision policies, the rules for choosing the best path can be learned by a deep neural network using RL mechanisms. The idea that these rules can be learned without human intervention was previously defended by Kool *et al.* (2018):

> A decade ago, computer vision algorithms used hand-crafted features but today they are learned end-to-end by Deep Neural Networks (DNNs) (…). Heuristics are typically expressed in the form of rules, which can be interpreted as policies to make decisions. We believe that these policies can be parameterized using DNNs, and be trained to obtain new and stronger algorithms for many different combinatorial optimization problems, similar to the way DNNs have boosted performance in the applications mentioned before (p. 1).

[29] One of the first to develop heuristic methods ["handcrafted" by experts] was Christoph Mandel, who applied them to a real-world scenario in Switzerland. This Switzerland network is considered a benchmark for evaluating VRP algorithms (Darwish *et al*., 2020).

[30] This is indeed a limitation, as the number of problem instances (the number of possible configurations of vertices and edges in a graph) is vast.

2.3 Introduction to Reinforcement Learning

As previously mentioned, reinforcement learning is a relatively new topic whose practical applications have only gained traction in recent years. Figure 1 shows that the number of articles on this topic has increased over the last few years.

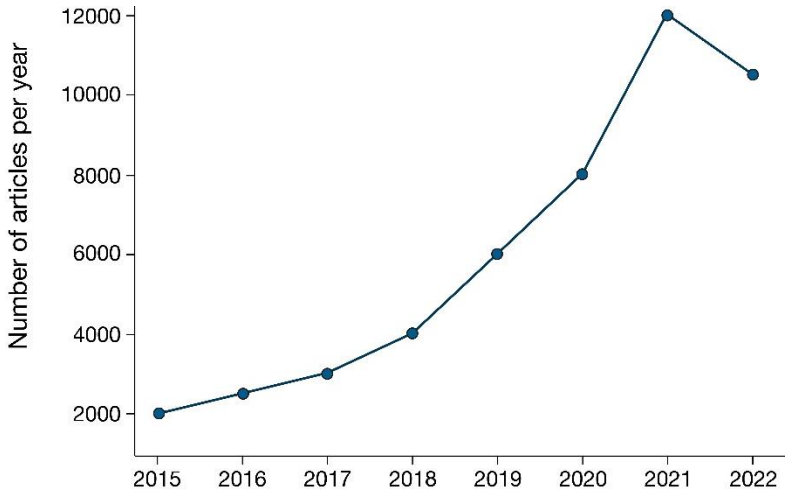

Figure 1. Number of articles in Scopus with the term "reinforcement learning". As of September 2022.

Although the first mathematical ideas and statistical concepts – such as Bellman's equation and Markov decision processes – that enabled the development of RL emerged in the mid-20th century, the development of specific RL algorithms only began at the end of the twentieth century[31], and it was only more recently that some of these concepts and algorithms were successfully applied[32].

These developments – in RL techniques, methods, and algorithms – have opened new possibilities for applying machine learning to other problems, especially those where it is not possible to program for an "infinity" of potential cases (e.g., robotic motion) or where it is impossible to "show" enough significant examples (i.e., where supervised learning techniques cannot be used).

[31] Examples of such algorithms include Q-learning (1989) and REINFORCE (1992). It is also worth noting that Sutton & Barto (2015) is considered one of the seminal books on RL, with the first edition dating back to 1998.

[32] Applications of RL have been demonstrated in helicopter control, data center cooling optimization, and the victory in the game of Go. More recently, applications have expanded into diverse areas such as logistics and inventory management for perishable goods (Selukar *et al.*, 2022), economics and price formation, electric grid management, optimal control problems in greenhouses or supply chains, the search for faster algorithms for matrix multiplication, drug design, and many other examples (Fawzi *et al.*, 2022; Ha & Jeong, 2022; Nakabi & Toivanen, 2021; Popova *et al.*, 2018; Zhang *et al.*, 2022; Q. Zhou *et al.*, 2022).

Reinforcement learning[33] is a computational approach to the problem of decision-making in uncertain environments, where algorithms are expected to learn how to make decisions and act independently without human supervision or prior knowledge. RL algorithms aim to mimic certain facets of human learning – such as the ability to modify actions based on past mistakes or repeat actions that have proven beneficial in the past – by translating this behavior into algorithms[34]. This approach is highly interdisciplinary, incorporating knowledge from fields like mathematics, computing, and engineering, as well as sciences that study various aspects of human behavior, such as psychology, sociology, economics, and neuroscience (Silver, 2016).

Many aspects distinguish reinforcement learning from supervised and unsupervised learning. In RL, learning occurs through a reward signal (feedback) from the environment in which the algorithm operates (as opposed to pre-labeled data). This signal is used to compute decisions about the following action[35] via dynamic mechanisms of trial and error, balancing exploration and exploitation phases[36] (Plaat, 2022).

Since RL algorithms act on the environment, they are called "agents". As Whittlestone *et al*. (2021) explain:

> RL systems are often referred to as 'agents', because they act autonomously in their environment. This does not mean that an RL agent is responsible or aware of their actions, just that the human designer is one step removed from the action selection process. (p. 1006)

Thus, while supervised and unsupervised learning typically involves a set of input data (either categorized or not) and produces classifications, regressions, or descriptions of associations between that data, RL techniques do not necessarily require an input dataset. Instead, they only need a clearly defined task, and their output is a decision embodied in an action. The algorithm has access to an environment with which it interacts, collecting the necessary information and building knowledge from it. These algorithms entail specific technical and computational challenges, as their construction must account for the fact that data arrives in sequential order, the observations made by the agent are often highly

[33] More specifically, Deep Reinforcement Learning.

[34] It has even been speculated that positive or negative rewards are sufficient to summarize all forms of learning and that maximizing a reward could be all that is needed to develop seemingly intelligent behaviors. Attend to the hypothesis raised by Silver *et al*. (2021): “that the generic objective of maximizing reward is enough to drive behavior that exhibits most, if not all, abilities that are studied in natural and artificial intelligence” (p. 1)

[35] The set of possible actions may not be discrete (Lillicrap *et al*., 2015).

[36] Exploration refers to any action taken to discover something unfamiliar (even if it results in a lower reward). In contrast, exploitation refers to actions where the algorithm chooses a known path because it expects a good return based on experience.

correlated[37], the agent influences its surrounding environment and, consequently, what it "sees" and finally, that the reward of an action may not be – and often is not – immediate.

Fundamentally, RL algorithms are used to solve problems with sequential characteristics. The algorithm "learns" by interacting with its surrounding environment, receiving rewards for its actions to accomplish predefined tasks. This sequential nature makes RL ideal for modeling "real-world" situations, where constant, incremental decision improvements are needed (based on rewards or signals from the environment) to achieve a task by acting on the environment and continuously planning for the future.

Thus, while supervised and unsupervised learning are primarily used for static problems, reinforcement learning addresses dynamic problems where the algorithm's actions interact with the environment, changing it and altering the information available for future collection and analysis. The problem is dynamic because the algorithm must adapt to a different situation at every new time step.

It should also be noted that RL algorithms are particularly suitable for dealing with problems involving multiple actors, each potentially with different motivations, and situations where it is necessary to balance multiple, possibly contradictory objectives (Darwish *et al*., 2020).

As previously mentioned, the article "Human-level control through deep reinforcement learning" (Mnih *et al*., 2015) marked a significant development in RL. This advancement corresponds to the creation of the first deep Q-networks (DQN), which combine deep neural networks[38] and reinforcement learning, establishing the field of deep reinforcement learning. This innovation expanded the possible applications of RL:

> Because neural networks can approximate any function, they can be used to encode the policy of an agent. This allows RL to extend to problems with state or action spaces that are too large for the tabular approach – which includes most interesting modern applications (Whittlestone *et al*., 2021, p. 1007).

DQN allowed, for the first time, the same algorithm – the same agent with the same architecture and hyperparameters – to perform several distinct tasks, such as playing various Atari games. In the case of DQN, deep neural networks were used to approximate the action-value function (Lillicrap *et al*., 2015). The combination of RL with deep neural networks extended the reach of these algorithms, as this new approach overcame previous limitations related to their ability to function only in low-resolution environments with discrete action spaces. According to Mnih *et al*. (2015), "This work bridges the divide

[37] Unlike many traditional problems, where information is (or is considered to be) identically distributed.
[38] Specifically, CNNs.

between high-dimensional sensory inputs and actions, resulting in the first artificial agent that is capable of learning to excel at a diverse array of challenging tasks" (p. 529). However, DQN was still limited, as it could handle large observation spaces but required small, discrete action spaces (Lillicrap *et al*., 2015).

Deep reinforcement learning refers to using neural networks to approximate any of the functions in RL: the policy, the reward function, the environment model, and any of the value functions (Selukar *et al*., 2022).

In summary, RL algorithms are designed to select and execute an action at each step, then evaluate the new state of the environment resulting from their action and receive a reward from the environment (see Figure 2). This reward serves as feedback, indicating whether the action taken brought the algorithm closer to its goal[39]. As Matsuo *et al*. (2022) puts it "reinforcement learning is a learning framework that improves a policy in terms of a given objective through interaction with an environment where an agent perceives the state of that environment" (p. 270).

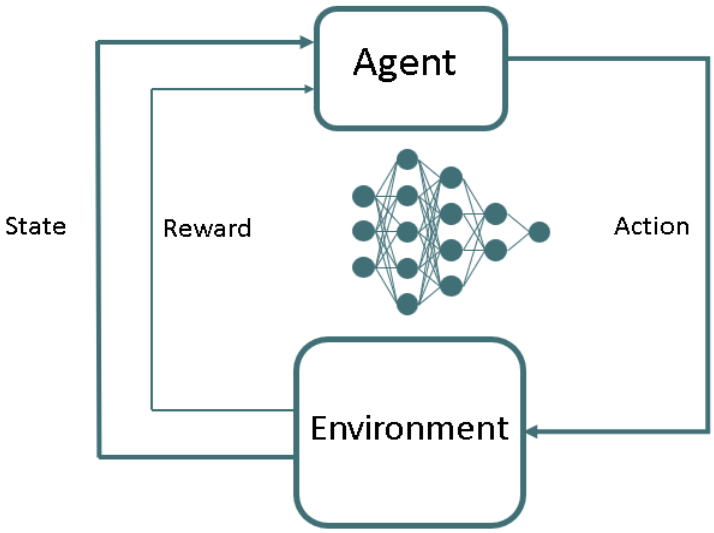

Figure 2. Relationship between Agent and Environment[40].

[39] This notion of reward is more complex than it might seem at first glance; not only can the reward for a given action be realized many time steps after the action (i.e., only understood retrospectively), but RL algorithms can also "plan" by accepting a negative reward in the present for a particular action to achieve a greater reward in the future.

[40] Image inspired by Bengio *et al*. (2021).

The following section, where some fundamental concepts for understanding this class of algorithms will be defined, is primarily supported by (Sutton & Barto, 2015), (Mazyavkina *et al*., 2020), (Kirk *et al*., 2021), and (Plaat, 2022):

Core Concepts of Reinforcement Learning

a. **State (S)**: Information used by the agent to determine the following action; it is an internal representation the agent makes of the history (i.e., what it "remembers" or not), representing the current situation of the environment at a given moment; it is a subset of the history.
b. **Action (A)**: A decision or movement that affects the state of the environment.
c. **Reward (R)**: A scalar value; it is the immediate feedback provided to the agent, dependent on the state of the environment and the action taken: $\mathbf{R_S = \mathcal{E}[R_{T+1} \mid S_T = S, A_T = a]}$
d. **History (H)**: The sequence of observations, actions, and rewards.
e. **Agent**: A function that maps the observed history to the following action. The agent observes the states, decides on actions, executes them, and collects rewards. The goal of the agent is to maximize the sum of all rewards (the return).

Interaction Between Agent and Environment

f. **Environment State**: The private information of the environment, which determines, from the environment's perspective, what happens after a particular action; this information is often not accessible to the agent.
g. **Return (G)**: The total discounted reward from time *t*: $\mathbf{G_T = R_{T+1} + \gamma R_{T+2} + \gamma^2 R_{T+3} + \gamma^3 R_{T+4} + \ldots}$ where $\boldsymbol{\gamma} \in [0,1]$ is a discount factor that implies, as a general rule, that present rewards are valued more highly than future rewards.
h. **Policy (π)**: Defines the agent's behavior, i.e., how the agent chooses actions; it is a function that maps each state to an action. It can be deterministic or stochastic, in which case it represents a distribution of actions given a specific state: $\boldsymbol{\pi(a \mid s) = \mathcal{P}[A_T = a \mid S_T = s]}$

Value Functions and Models

i. **State Value Function (V)**: A function that predicts the future rewards of a given state (i.e., how "good" or "bad" a state is) by following a specified policy **π**; it can be estimated from samples: $\mathbf{V_\pi(S) = E_\pi[R_T + \gamma R_{T+1} + \gamma^2 R_{T+2} + \ldots \mid S_T = s] = E_\pi[G_T \mid S_T = s]}$ where $\boldsymbol{\gamma} \in [0,1]$ is the previously mentioned discount factor.

j. **Action Value Function (Q)**: The expected return from state ***s***, following action ***a***, and then policy **π**: $\mathbf{Q_\pi(s,a) = E_\pi[G_T \mid S_T = s , A_T = a]}$

k. **Model**: A model predicts what the environment will do next. Regarding RL, the model of the environment can be:

   1. **Transactional Model**: Predicts the next state of the environment (i.e., the dynamics of the environment): $\mathbf{P_{SS'} = P[S' = s' \mid S = s , A=a]}$
   2. **Reward Model**: Predicts the immediate reward: $\mathbf{R_S = E[R \mid S = s , A = a]}$

Markov Framework and Mathematical Foundations

l. **Markov Process**: A "memoryless" random process, i.e., a sequence of states with the Markov property: the probability of a future event depends only on the present state, not on the past states: $\mathbf{P[S_{T+1} \mid S_T] = P[S_{T+1} \mid S_1,\dots,S_T]}$

m. **State Transition Probability Matrix (P)**: A matrix that defines the transition probabilities between each pair of states for all possible states of the environment.

n. **Markov Decision Process (MDP)**: "Markov Decision Processes are a tool for modeling sequential decision-making problems where a decision-maker interacts with a system in a sequential fashion" (Szepesvári, 2010). MDPs are an extension of Markov processes, incorporating the possibility of different actions (to maximize the return). Mathematically, they correspond to a vector – **(S,A,P,R,γ)** – composed of the following elements:

   1. **S** is a (finite) set of states
   2. **A** is a (finite) set of actions
   3. **P** is the state transition matrix
   4. **R** is the reward function
   5. **γ** is the discount factor

o. **Bellman Equation**: The Bellman equation provides a way to find the value functions of a system. This equation can decompose both the state value and the action value functions. The value function can be decomposed into immediate and discounted rewards in future states. This decomposition allows for a recursive relationship between $\mathbf{V_\pi(S)}$ and $\mathbf{Q_\pi(s,a)}$. Thus:

$$\mathbf{V(S) = E[G_T \mid S_T = S]}$$
$$\mathbf{= E[R_{T+1} + \gamma R_{T+2} + \gamma^2 R_{T+3} + \dots \mid S_T = S]}$$
$$\mathbf{= E[R_{T+1} + \gamma(R_{T+2} + \gamma R_{T+3} + \dots) \mid S_T = S]}$$

$$= E[R_{T+1} + \gamma G_{T+1} \mid S_T = S]$$

$$= E[R_{T+1} + \gamma V(S_{T+1}) \mid S_T = S]$$

$\therefore V(S) = E[R_{T+1} + \gamma V(S_{T+1}) \mid S_T = S]$ is the Bellman equation **(1)**

Solving equation (1) algebraically:

$$V(S) = E[R_{T+1} \mid S_T = S] + E[\gamma V(S_{T+1}) \mid S_T = S]$$

$$= R_S + \gamma E[V(S_{T+1}) \mid S_T = S]$$

$$= R_S + \gamma \sum P_{SS'} V(S'),$$

where $P_{SS'}$ is the transition probability from **S** to **S'**.

Simplifying the notation:

$$V = R + \gamma PV \quad \textbf{(2)}$$

where **P** is the State Transition Probability Matrix (2)

From (2), we can conclude that:

$$V - \gamma PV = R \Rightarrow$$

$$(1 - \gamma P)V = R \Rightarrow$$

$$V = (1 - \gamma P)^{-1} R \quad \textbf{(3)}$$

From (3), it follows that one must invert the state transition probability matrix to solve the Bellman equation.

p. Note that inverting matrix P in (3) is computationally complex, so solving the Bellman equation directly is only possible in more straightforward cases. Generally, different techniques are required, such as:

1. Dynamic Programming
2. Monte Carlo Simulation
3. Temporal-Difference Learning

These techniques help characterize the type of RL algorithm being used. For example, Dynamic Programming requires complete knowledge of the system (i.e., knowledge of the state transition probability matrix and reward function); Monte Carlo Evaluation estimates the state value function based on the average return of sampled episodes and requires that episodes terminate; Temporal-Difference Learning combines both techniques, learning (i.e., approximating the actual state value function) before the end of each episode (and does not require episodes to terminate).

Based on the definitions provided, RL algorithms can be classified by their learning strategy and representation into:

a. *Value-based algorithms*: when the agent has a value function, which may be based on the state value (state value function, **V(S)**) or on the action value (action value function, $\mathbf{Q_\pi(s,a)}$);
b. *Policy-based algorithms*: when the agent has an explicit representation of the policy to follow;
c. *Actor-Critic algorithms*: in these cases, the agent has both a defined policy and the value associated with each state.

They can also be classified by the origin of the data the agent uses to learn:

a. *On-policy methods*: the agent learns based on the policy it is currently using, i.e., the policy used to determine the agent's behavior is the same as the one being updated;
b. *Off-policy methods*: the agent can use information generated by a policy that is different from its current policy.

2.4 Applications of Reinforcement Learning in Combinatorial Optimization Problems

The hypothesis that RL can be one of the possible approaches to solving combinatorial optimization problems is supported by Bengio *et al*. (2021) and Berto *et al*. (2023). However, considering that the application of RL to the Transit Network Design and Frequency Setting Problem is still in its early stages[41] and that this is a highly complex problem[42], studying procedures and solutions for more specific instances of the TNDFSP, such as the Travelling Salesman Problem (TSP), the VRP, or the Orienteering Problem (OP)[43], among others, could provide insights into solving the TNDFSP.

Bello *et al*. (2017) previously made a significant effort in this direction by tackling the resolution of the TSP without relying on prior knowledge[44]. Their results, while not yet entirely satisfactory, demonstrate possible paths for applying deep RL to optimization problems. In this article, the authors use the concept of the "negative of the tour distance" as a reward signal for the agent; the problem is mathematically

[41] The first article dedicated to solving the TNDFSP using RL appears to be (Darwish *et al*., 2020).
[42] "Sources of complexity for the TRNDP: its combinatorial and multi-objective nature, the difficulties in formulating the problem and formally defining acceptable spatial route layouts, and the nonlinearities and non-convexities characterizing the problem" (Baaj and Mahmassani, 1991, as cited in Kepaptsoglou & Karlaftis, (2009)).
[43] The TSP is a classic combinatorial optimization problem. Given a list of cities and the distance between each pair of cities, the goal is to find the shortest path that returns to the origin after visiting each city exactly once. The OP involves choosing a subset of locations constrained by specific factors to maximize the value collected from them.
[44] The problem addressed here is the TSP in a 2D Euclidean space (Euclidean plane).

formulated[45], and the solution is redefined as an attempt to find a permutation of the graph's vertices that satisfies the desired conditions (namely, a permutation that forms a complete cycle through the graph's points). The authors utilize a pointer network trained differently from the one originally developed by (Vinyals *et al*., 2015)[46]. Instead of being trained using supervised learning[47] and a loss function based on an entropy measure[48] – as the original authors did – Bello *et al*. (2017) modified the pointer network to be trained with a policy-based approach (specifically, actor-critic) using the REINFORCE algorithm (Williams, 1992, as cited in Bello *et al*. (2017)).

Kool *et al*. (2018) propose a network architecture of the Encoder-Decoder[49] type and a model based on multiple layers of attention mechanism[50] and RL, initially used to find solutions for the TSP and VRP variations[51]. The encoder they used is called Transformer, a network architecture developed initially by Vaswani *et al*. (2017) and successfully tested in translation tasks[52]. It does not use RNNs in the encoder-decoder structure but attention mechanisms, similar to pointer networks. Kool *et al*. (2018) emphasize that models based solely on attention mechanisms have advantages over the architecture proposed by Vinyals *et al*. (2015) (where training is done using supervised learning).

Regarding the works of Vinyals *et al*. (2015) and Bello *et al*. (2017), which are considered seminal, Dai *et al*. (2017) note that although these approaches can handle graphs of different sizes – a desirable property for generalizing the algorithms – they only do so through ad hoc engineering (for example, by

---

[45] Defining the distance between two points and the objective as minimizing this distance, defining the path as a permutation of points, and factoring the total probability of a path into conditional probabilities (chain rule).

[46] Pointer Networks: Neural networks specifically designed to predict a permutation of elements, which are discrete tokens at a specific position in the input sequence. The use of RNNs for solving combinatorial optimization problems has limitations since RNNs generally require the input and output data to be the same size, which is different for many such problems. Pointer networks represent an attempt to overcome this limitation (Vinyals *et al*., 2015). They also represent a supervised approach to combinatorial optimization problems (Berto *et al*., 2023).

[47] This means they are based on annotated data: to generate annotated data, Vinyals *et al*. (2015) used various algorithms (of a heuristic nature) to solve the TSP sub-optimally.

[48] "Entropy is a measure of disorder that can be applied to a set (…). Disorder corresponds to how mixed (impure) the segment is with respect to these properties of interest (…) we can define information gain to measure how much an attribute improves (decreases) entropy over the whole segmentation it creates" (Provost & Fawcett, 2013, pp. 51-52).

[49] In these types of networks, the encoder initially encodes the data into high-dimensional vectors (in this particular article, spatial points in 3D are encoded into 128-dimensional vectors).

[50] "Attention mechanisms have become an integral part of compelling sequence modeling and transduction models in various tasks, allowing modeling of dependencies without regard to their distance in the input or output sequences" (Vaswani *et al*., 2017, p.2).

[51] Such as the OP and the Prize Collecting TSP (PCTSP).

[52] Here, the definition of success is more technical, as the authors aim to demonstrate that simpler networks based on attention mechanisms (and not RNNs, typically used for sequential problems) are more parallelizable and, consequently, faster. This network architecture gave birth to the first large language model known to the public – the GPT.

padding zeros to adjacency matrices[53] to handle graphs of different sizes). Moreover, they point out that these approaches adopt strategies that do not consider the graph's structure, being "graph-agnostic sequence-to-sequence mapping" (Dai *et al*., 2017, p. 2).

As previously mentioned, the structure that best describes the TNDFSP is a graph, directed or undirected, where both the vertices and edges can have various attributes. The adjacency matrix is one of the primary ways of representing graphs in tabular format.

Combinatorial optimization problems on graphs have gained considerable attention in recent years, as these complex structures can generalize many aspects of human reality, aspects represented by different data but maintaining the same combinatorial structure (Dai *et al*., 2017). Thus, the attempt to automatically learn general heuristics that work across many instances of the same problem seems promising. Dai *et al*. (2017) apply this idea to combinatorial optimization problems such as the Minimum Vertex Cover, Maximum Cut, and TSP by combining RL with a graph embedding method called "Structure2Vec."

Veličković *et al*. (2017) argue that there are limitations to using convolutional neural networks (CNNs)[54] to solve combinatorial optimization problems:

> CNNs have been successfully applied to tackle problems such as image classification (...), semantic segmentation (...) or machine translation (...), where the underlying data representation has a grid-like structure (...). However, many interesting tasks involve data that cannot be represented in a grid-like structure and instead lie in an irregular domain. This is the case of 3D meshes, social networks, telecommunication networks, biological networks or brain connectomes. Such data can usually be represented in the form of graphs. (p. 1)

There have also been arguments against using RNNs for combinatorial optimization problems. Indeed, Vinyals *et al*. (2015) note that:

[53] "The adjacency matrix, sometimes also called the connection matrix, of a simple labeled graph is a matrix with rows and columns labeled by graph vertices, with a 1 or 0 in (…) according to whether (…) [the respective nodes] are adjacent or not. For a simple graph with no self-loops, the adjacency matrix must have 0s on the diagonal. For an undirected graph, the adjacency matrix is symmetric" (Wolfram, 2023).

[54] It is worth noting that an image is a very particular case of a graph where each node has always the same number of connections. Traditional convolution techniques (such as those used in DQNs) allow only for inputs of regular size because each pixel in the image always has the same number of neighbors.

> [RNN-based] methods still require the size of the output dictionary to be fixed a priori. Because of this constraint, we cannot directly apply this framework [RNN] to combinatorial problems where the size of the output dictionary depends on the length of the input sequence (p. 1).

Considering these arguments, Veličković *et al*. (2017) developed a network architecture called Graph Attention Networks. This architecture improves previous implementations of graph modeling through deep learning mechanisms by contributing to the generalization of convolution processes to the domain of graphs, particularly by using attention mechanisms (mentioned earlier). The article details how these attention mechanisms work, noting several advantages over previous implementations:

- More efficient operations because they are parallelizable
- Can be applied to all vertices of a graph, regardless of their degree[55]
- Greater generalization capacity for unseen graphs (out-of-sample data)

2.5 Relevant Studies on RL-Based VRP Solutions

The use of tools developed for problems other than the TNDFSP is also the approach adopted by Darwish *et al*. (2020), where the authors use the architecture created by Kool *et al*. (2018) (which was designed and applied to more straightforward problems, such as the TSP, the OP, and the Prize Collecting Travelling Salesman Problem (PCTSP)) to attempt a solution for the TNDFSP itself. Building on the fact that Kool *et al*. (2018) used attention mechanisms with promising results and noted that their architecture could be applied to other problems by adjusting how visited points are masked (or not) by the algorithm, Darwish *et al*. (2020) achieved interesting results, particularly in modeling specific aspects of a public transportation network, such as line changes, and even a method for accounting for wait times at stops. These aspects have not been seen in any other work. Yoo *et al*. (2023) also claim to model more complex situations than those presented in other studied papers. In their case, the authors developed an architecture that simultaneously models (and optimizes) the number of routes, their design, and the transport frequency. However, their paper is quite vague regarding the type of algorithms used. Unfortunately, since the code used in these papers is not available online, it was impossible to reproduce the results.

Nazari *et al*. (2018) developed a model for solving the Capacitated Vehicle Routing Problem (CVRP), which can be applied to both "static" instances, where all conditions are known in advance, and "dynamic" situations where external disruptions (such as the appearance of new customers mid-process that change

[55] The degree of a vertex in an undirected graph corresponds to its number of edges.

the requirements of a given location) mean that the Markov transition function is unknown. They developed a model applicable to stochastic situations. In their model, they replaced the RNN encoder used by Bello *et al*. (2017), arguing that, unlike translation tasks where the input sequence provides relevant information, RNNs are unnecessary in combinatorial optimization problems where the input order to the model has no significance. In their article, they compare their results with those of other solvers, achieving very good results, particularly for larger instances of the problem (50 points and more). They use two decoding techniques: greedy, where the highest probability point is chosen at each step, and beam search, which selects the path corresponding to the shortest route among the highest probability points. Although they obtained better results with the second decoder, the difference between results diminishes as the problem size increases.

Peng *et al*. (2020) built on top of the model developed by Kool *et al*. (2018), modifying it so that each vertex's characteristics are updated incrementally as the agent constructs the paths. They aimed to overcome what they considered a limitation in Kool *et al*. (2018), where the characteristics of each vertex are defined at the beginning of the process and do not change afterward[56].

More recently, different approaches have been developed with different rationales (i.e., not only aiming to solve a VRP). Considering the ability of these algorithms to generalize not only to unseen instances (classic generalization) but also to solve multiple distinct VRP variants with different constraints, and given that existing approaches focus mainly on solving a single variant, Zhou *et al*. (2024) developed an algorithm called the Multi-task Vehicle Routing Solver with Mixture-of-Experts (MVMoE), capable of handling several VRP variants simultaneously. This neural network is trained on a different problem instance in each training epoch. The MoE mechanism consists of a set of "experts", which are feed-forward networks with independent and trainable parameters, and a gating network, which determines which input aspects are distributed to each "expert". Similar approaches are followed by Liu *et al*. (2024) and Berto *et al*. (2024), who propose a model that solves a VRP with combinations of constraints different from those seen by the model during training. They do so by redefining the VRP itself, generalizing it as a combination of different attributes, where different mixes create a distinct variation.

[56] This should not be the case, at least in theory, since a node's characteristics depend on those of its neighbors and the neighbors of its neighbors (in a chain as deep as desired), and if those change, the initial characteristics should also change.

3 Methodology

3.1 Formulation of the VRP in Lisbon

This thesis aims to deepen the understanding of RL and its potential applications in solving a combinatorial optimization problem. Through a case study, one of the previously described RL algorithms will be applied to a real-world scenario. The motivation is twofold: on the one hand, it seeks to evaluate the performance of these algorithms when applied to real data – rather than just benchmark data, as is common in many studies. On the other hand, it is relevant to explore the application of these algorithms to socially significant problems, such as improving a public transportation network, which can have various positive impacts on a city. As part of these motivations, the discussion and evaluation of the algorithms will incorporate some form of metric regarding the impact of the solution found[57].

The chosen case study focuses on the city of Lisbon, specifically the public bus network operated by *Carris*. This network represents a good starting point because, in addition to not needing to define bus stops[58] (the graph's vertices), bus networks (as opposed to metro networks) are more flexible. In practice, it would be easier to alter bus stops, the connecting lines, and the bus frequencies if necessary.

Regarding the Carris network, Lisbon is divided into five geographic zones (Center, West, Northwest, North, and East). There is also a functional zone (called Circular) that does not correspond to any specific geographic zone but covers routes that traverse all or part of the other zones, facilitating connections between non-adjacent zones (see Figure 3).

Figure 3. Stylized Diagram of the Carris Network[59]**.** *Ocidental*=West*, Noroeste*=Northwest*, Norte*=North*, Oriental*=East*, Centro*=Center*, Circulares*=Circular.

[57] The efficiency of a transportation network can be measured using various indicators, such as the maximum time it takes for each transport vehicle to cover a certain route, the number of stops it makes, the total distance traveled, the number of passengers it can carry, the maximum distance between stops, the cost of maintaining the existing fleet, the number of drivers required, the average travel time for each passenger, etc.

[58] As previously mentioned, in most cases the selection of bus stops is treated as a separate problem. Here, it is assumed that this choice is optimal, considering a certain level of rationality by *Carris*.

[59] Image taken from *Carris* website: https://carris.pt/viaje/mapas/.

The four outer zones (except the Center and Circular zones) consist primarily of lines with a more radial movement, converging toward the city center. This is likely because the most common movement of the population involves commuting to the city center for work in the morning and returning home in the evening. The Center zone comprises lines without a defined direction and serves more complex objectives. These include accommodating various transportation needs beyond the home-to-work commute, such as facilitating connections with other modes of transport like the metro and trains.

Generally, each bus line belongs to only one zone, except for the lines from the Circular zone. These zones are not entirely isolated; each is connected through its permeable borders to adjacent zones. Some lines in each zone extend slightly into neighboring zones to allow communication between zones at their borders.

It is also noteworthy that each zone has two types of lines: main lines, which traverse the zone with buses running in both directions, and smaller lines (operated by smaller buses, also known as neighborhood lines), which form directional loops. These neighborhood lines start and end at the same point, with buses running in a continuous circular direction (either clockwise or counterclockwise).

The Carris network consists of 1.701 bus stops. However, the total number of stops (counting all stops on every line) amounts to 2.935. To increase the complexity and realism of the analysis, it would be necessary to understand this network not only as a set of points connected by various lines but also in terms of the connections between them. This would involve assigning three dimensions to each point – two geographic dimensions and a third representing the number of lines passing through that point – and considering that a line transfer corresponds to part of the journey where time (waiting time) is consumed but distance is not.

Given the size and complexity of the network, the exploratory phase focused on two stops for each line: the first and the last. The corresponding graph was then drawn, producing a very rough representation of the current network, totaling 218 points connected by 109 lines[60] (Appendix A). Even at this early stage, some of the city's characteristics and, consequently, those of the Carris network are visible: empty spaces or areas without lines (corresponding to places like *Monsanto* or Lisbon Airport, where there are few roads or less need for transportation) and a transportation structure that becomes denser (with more stops) and more connected (with more lines) the closer one gets to the city center. Additionally, points of greater connectivity, such as *Marquês de Pombal*, *Sete Rios*, and *Cais do Sodré*, can be identified. These are locations where many lines converge or where connections to other forms of

[60] The quantitative data referenced throughout this work relates to information collected about the network between June and July 2023.

transportation exist. The goal was to progressively refine this construction by building graphs with increasingly detailed points, bringing the representation closer to the real network with all its connections, intersections, and circular routes (Figure 4), thus gaining more insight into the network.

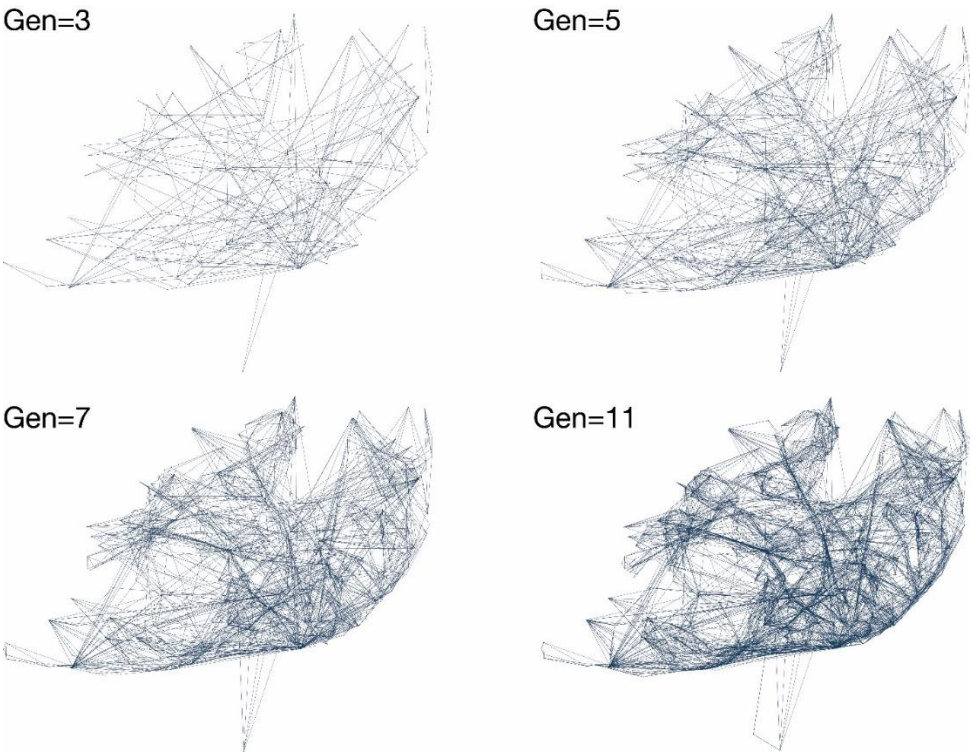

Figure 4. Fractal Evolution of the Carris Network. Four generations, each with more stops.

Initially, the idea was to analyze – and find a solution for – the entire set of bus stops in Lisbon, representing 1.701 stops distributed across 109 lines (see Figure 5).

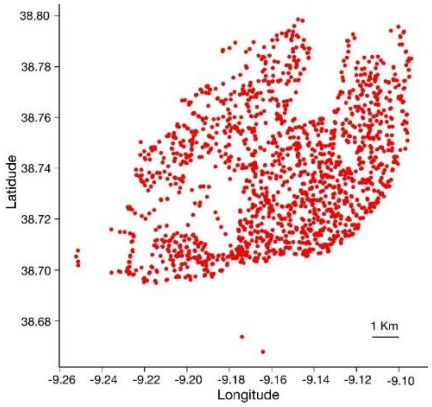

Figure 5. Location of Bus Stops – Carris Network. As of June 2023.

It was found, however, that the scale of the proposed analysis – to use the available tools to examine a graph representing the city of Lisbon – was unrealistic, even with all possible simplifications[61]. Therefore, considering that *Carris* divides the city into different zones, it was decided to focus the analysis on just one of these zones. Refer to Table 1, which presents the number of bus lines and (non-unique) bus stops per zone:

Table 1. Number of Lines and Stops per Zone

Zone	Color of Zone	# lines	# stops
Neighborhood	Yellow	28	586
Northwest	Blue	14	382
Nocturnal	Dark blue	6	276
Circular	Grey	11	423
Center	Orange	19	310
West	Pink	9	252
North	Green	7	176
East	Red	15	457

The Nocturnal line and Neighborhood line were not considered in the selection of the zone to analyze: the former because it is a temporal category rather than a geographic one, and the latter because it consists of circular lines, which posed problems in both the processing ("what is the distance between two points on a circular route?") and the analysis of the data (as cycles add complexity to the modeling[62]). Considering these factors, the North zone was selected for analysis. This zone has fewer lines and stops – only seven lines and 176 stops, corresponding to 144 unique bus stops. The set of analyzed points can be seen in Appendix B.

[61] To give an idea of the difficulty: the best model trained by one of the tested RL algorithms – (Peng *et al*., 2020) – whose authors claim it can compete with Google's solver (OR Tools, which does not incorporate artificial intelligence in its design), was trained to handle problems with a maximum of 50 vertices. Some simple exploratory tests with OR Tools also revealed weaknesses when the number of vertices exceeded 100 – for example, in a test with 150 vertices and 15 lines to approximate the numbers of the modeled zone, the solver returned solutions where many transport vehicles did not connect to any vertices.

[62] The existence of cycles can imply a less efficient system because they represent redundancy: in a cycle, all points are connected by two different paths. Many algorithms do not allow for the existence of "cycles" – during training, the algorithm hides ("masks") all points already visited by the agent.

3.2 Data Extraction and Processing

Automated and high-quality data extraction is crucial to conducting a meaningful and statistically significant analysis. Data was collected primarily from GoogleMaps[63] and Carris[64] websites during June 2023, using scraping techniques (via the *BeautifulSoup* and *Selenium* libraries) to automate data collection. The Google API was also used to provide essential information such as the "distance matrix" and "duration matrix"[65] which are crucial to conducting the analysis.

The data collected for each bus line in the city included bus stops, geographic location (latitude and longitude), a unique *ID* provided by Google, the associated URL, the bus stop name (each stop has a name on the Carris website, though it is not unique), the type of stop, its status (Google Maps provides information on the kind of stop – station, terminal, garage, etc. – and whether it is active or inactive), and the bus schedules for each stop. The distance in a straight line, the distance by road, and the travel duration between each pair of stops were also calculated, thus constructing a "complete graph" (Kool *et al*., 2018).

After collecting the data, it was compiled into several Excel files for easier observation and cleaning. Scraping often generates errors, usually due to missing information on a given page or difficulties accessing information during the collection process (for example, changes in the HTML structure when navigating between different pages on the same site). Data cleaning involved removing duplicates and correcting some of the bus stop *ID* references, which showed inconsistencies (some *IDs* returned by the Google API were clearly non-random, so it was necessary to recollect this information).

For this particular study, data cleaning also required eliminating bus stops that were very close to each other (these usually represent locations with multiple stops, such as significant hubs like *Sapadores*, *Cais do Sodré*, *Marquês de Pombal*, and *Estação do Oriente*, or stations on the same street in opposite directions). Although it is not entirely precise, the straight-line distance was used to eliminate stops that were too close to each other, and only one stop was selected among those less than 50 meters apart.

After data cleaning, several metrics regarding the network were calculated, such as the number of lines, the number of stops per line, and the number of stops per zone[66]. These metrics provided a general

[63] https://www.google.com/maps

[64] https://carris.pt/viaje/carreiras/ and https://carris.pt/viaje/mapas/

[65] Both matrices aim to describe a graph. In this case, the distance matrix provides information on the distance between any two points (depending on the problem, either a straight line or following existing roads). The duration matrix provides information on the time it takes to travel from one point to another (this information considers traffic at the time the data was collected).

[66] One relevant information considered at this stage was the number of buses running each line at a particular time of day. Unfortunately, these values could not be calculated due to the technical complexities of analyzing online data.

overview of the network's structure and informed subsequent decisions about which zone to choose for the subsequent analysis.

3.3 Libraries Used

Several practical implementations from the previously mentioned papers (available online) were considered for use. The algorithms (and corresponding code) developed by Kool *et al*. (2018), Nazari *et al*. (2018), Peng *et al*. (2020), and Berto *et al*. (2023) were explored to determine whether code designed for modeling a different but related problem could be adapted and serve our purpose[67].

The models developed by Kool *et al*. (2018) and Nazari *et al*. 2018), available on *GitHub*[68], are currently outdated. In the first case, many of the files have not been updated in five years, while in the second case, the models depend on outdated Python 3.6 and Pytorch 0.4.1 versions.

As for Peng *et al*. (2020) who built on the work of Kool *et al*. (2018) by making the encoding of locations dynamic, it was impossible to train new models with the code provided by the authors[69]. Several experiments were conducted with their pre-trained models, but two issues arose: first, the pre-trained models were designed for environments with a maximum of 50 points. Although the authors suggested the models could work with more points, the results for around 150 points (a number closer to the actual scenario being modeled) were disappointing. It was hypothesized (but not tested) that these pre-trained models performed well for uniform point distributions (i.e., environments similar to the training data) but may need to generalize better to situations where the distribution is non-uniform, such as the distribution of bus stops in Lisbon. As a result, this approach was abandoned.

Considering all these factors, the architecture developed by Berto *et al*. (2023) was ultimately selected – a library called RL4CO ("Reinforcement Learning for Combinatorial Optimization"), available on *GitHub*[70]. This architecture was chosen not only because it is readily available, up-to-date, and the most recent implementation with active user support but also because, unlike the other approaches, it represents an attempt to build an open-source framework that can be used and developed for various distinct problems with as general an application as possible. Its open-source nature allowed understanding and manipulating its source code to correct and adapt when facing challenges or errors (see Appendix E).

[67] Besides the implementations mentioned here, others were intended for exploration because they presented more promising results or were trained in more complex environments, making them more interesting. However, some authors, such as Darwish *et al*. (2020), do not provide the code associated with their articles online.
[68] https://github.com/wouterkool/attention-learn-to-route; https://github.com/OptMLGroup/VRP-RL
[69] https://github.com/d-eremeev/ADM-VRP
[70] https://github.com/ai4co/rl4co/tree/main

Unlike other implementations, whose code was developed to address a specific problem (such as VRP) and could not be successfully adapted to this thesis, the RL4CO library allowed for code to be built from scratch and enabled relatively easy experimentation[71] with different models and hyperparameters, which could then be compared. This library, which acts as a wrapper around PyTorch, facilitates the creation of different environments and the control of various aspects of the training and testing process. In the words of the authors:

> RL4CO [is] a unified and extensive benchmark with in-depth library coverage of 23 state-of-the-art methods and more than 20 combinatorial optimization problems. (…) RL4CO allows researchers to seamlessly navigate existing successes and develop their unique designs, facilitating the entire research process by decoupling science from heavy engineering. (Berto *et al*., 2023, p.1)

Another advantage of this library is that it can be parallelized. It is based on TorchRL, which allows for running environments and algorithms using GPUs (in contrast to environments based on OpenAI Gym).

3.4 RL Algorithms

Given the significance of the experiment from Kool *et al*. (2018), which is one of the most cited studies and was among the first to use attention mechanisms to solve this type of problem, it was decided to use an architecture similar to the one constructed in their paper (encoder-decoder with attention mechanisms), but with a different baseline[72]. In particular, the used architecture is known as MVMoE – Multi-task Vehicle Routing Solver with Mixture-of-Experts and was proposed by Zhou *et al*. (2024). This architecture incorporates a mixture of experts into the attention mechanisms, improving the model's efficiency without increasing computational capacity. Greedy decoding will be used to build the solution, following the experiments by Nazari *et al*. (2018), which did not reveal significant differences between greedy decoding and beam search when the number of points in the problem increased.

Thus, two experiments were conducted with MVMoE, a policy-based algorithm (as it directly optimizes a policy rather than a value function), on-policy (using baselines, the executed policy is the

[71] The ease of use is indeed relative: as Frederico Berto stated during an online conversation held in June 2024, this framework has a steep learning curve, but it quickly becomes more manageable; this was also the experience with this work.

[72] Baseline: A reference value used to stabilize the learning process, reducing variance in policy gradient estimation and improving convergence to the optimal policy. The baseline used in Kool *et al*. (2018), which varies over time, is more complex than the one used in the experiments described in the following chapters – which corresponds to a shared baseline.

evaluated one), and using Monte Carlo evaluation (where the episodes are expected to finish before evaluation).

4 Experimental Setup

This chapter will detail the experimental setup used to validate the proposed methods and models. It will also discuss the simulation environment, including a description of the environment generated for agent training and the evaluation metrics chosen for comparing the models. Additionally, the configuration parameters of the algorithms, along with the tools and technologies used for developing and conducting the experiments, will be described. The proper design of these components is essential as it directly impacts the algorithm's effectiveness and efficiency throughout the learning process.

The models were trained on the *Google Colab* platform. Several exploratory tests were conducted to decide which GPU to use, with no significant difference in training time when varying the GPU. Therefore, all experiments were performed using an NVIDIA T4 Tensor Core GPU hardware accelerator. In terms of software, all code was written in Python, version 3.10; the experiments were logged using the *Weights & Biases* platform and the *TensorBoard* tool.

Two experiments were conducted where the number of training epochs varied — four in the first experiment and ten in the second. The configuration of the different states that the environment could assume, the various actions the agent could execute, and the calculation of the reward for each epoch did not vary between experiments. In both experiments, 10.000 training instances (randomly generated VRP problems), 10.000 test instances, and 1.000 validation instances were used, providing a solid foundation to evaluate the proposed algorithm's performance.

4.1 Decision Dynamics

Solving a VRP applied to the Lisbon public transportation network involves modeling various factors, such as passenger pickup and delivery, location definition, and vehicle capacity, among other parameters. For this study, the environment provided by the RL4CO library was adapted to the passenger transportation problem. Below, the main components and configuration of the simulation environment are described.

Simulation Environment States

The chosen environment for the experiments was the VRP with pickups and deliveries (Vehicle Routing Problem with Pickups and Deliveries – VRPPD), as this was the most suitable environment available in the library for the goal of this study – modeling passenger transportation in an urban

environment. It allows the definition of constraints such as vehicle capacity[73], where routes can be either "closed" or "open" (i.e., returning to the starting point or not), and the inclusion of both pickups and deliveries in any order (respecting the rule that passengers must be picked up before being delivered) and the definition of time windows (this study did not use time windows for reasons that will be briefly explained). These specifications closely mirror the real-world scenario: Carris buses must pick up passengers at various locations and drop them off at multiple other locations, buses have limited capacity, and vehicles must adhere to a daily schedule for each location.

The experiments were configured with the following definitions[74]:

- *Number of locations*: This determines how many points the vehicles must visit. To simplify the application of algorithms to real data[75], the experiments were conducted for 144 locations, corresponding to the number of unique locations in Lisbon's North Zone.
- *Location of points*: This information is randomly generated at the start of training; for efficiency and consistency during training, all points are uniformly distributed in the unit square $[0, 1]^2$.
- *Vehicle position*: Dynamic information updated as the agent interacts with the environment, i.e., as the agent decides where to go next.
- *Vehicle capacity*: Statically configured, with a maximum capacity of 100 passengers, intended to reflect the public transport scenario in Lisbon.
- *Remaining vehicle capacity*: Dynamic information updated along the route, indicating the number of passengers the vehicles can still accommodate.
- *Service demand*: The number of passengers the vehicle must pick up at each location and their destination is randomly generated for each point, with limits between a predefined minimum (set to 1) and a maximum (set to 20).
- *Distance Matrix* or *Duration Matrix*: In this case, these matrices were not part of the environment definition, but defining one of these matrices is common in such problems. Many algorithms do not include the geographical locations (latitude and longitude) but only the distances between points.

[73] They can include weight limits, volume limits, restrictions on the order in which goods are loaded or unloaded, and product compatibility, among others.

[74] At the code level, the state is encoded in a TensorDict, generally called "td".

[75] The application of these algorithms to real-world everyday situations necessarily implies that they must be usable in scenarios with a variable number of locations and, consequently, in situations with a different number of points than the number of points they were trained for. However, addressing this issue poses technical challenges, the resolution of which is beyond the scope of this work.

- *Visited points*: Dynamic information that is updated throughout the training process; the algorithm masks visited points to prevent them from being selected again.
- *Time windows*: Constraints related to the time each location must be visited. Some exploratory experiments with time windows were conducted to mimic reality, where vehicles arrive at each location at a specific time. However, it was found that it was not possible to establish these schedules without significantly influencing the agent's choices during training[76], so time windows were not defined in the experiments conducted.
- *Vehicle speed*: For simplicity, speed was defined as "1" for all vehicles.

The locations of the points, the depot location, the number of passengers at each location, and their destination were randomly generated to create a diverse set of scenarios.

Action Space

The action space refers to the possible choices the agent can make at each step. In this case, the action space is relatively small, reducing the problem's computational complexity. The agent must select one of the unvisited locations (excluding the depot). In environments with time windows, the agent could also choose to wait at a specific location.

Reward Function

Defining an objective is central to learning. Regarding the reward function, it is necessary to determine what constitutes "the best route". If the algorithm's goal was simply to find the route with the shortest total distance, without any other constraints, the optimal solution would involve using only one vehicle to connect all locations in a single route, reducing the problem to a TSP. A common way to define the objective in the case of VRP is for the agent to minimize the length of the longest route among all the routes while meeting the given constraints. This was the objective function defined, with the reward considered as the negative of the distance, which penalizes longer routes and incentivizes a more balanced distribution among vehicles.

[76] The tendency is to define very tight time windows, thus artificially constraining the agent's choices. For example, if the vehicle must be at point A at 17:00 and at point B at 17:03, the algorithm will naturally follow that path, even if it is not the most efficient route, which is what we are trying to discover here.

Parameter and Hyperparameter Selection

It was found that when training a model with a number of locations in the range of hundreds (e.g., 150) and a larger batch size (e.g., 128 or 256), the GPU could not handle the required number of calculations. Successive experiments were conducted, gradually lowering this hyperparameter until the training process could be completed. Ultimately, the batch size[77] was significantly reduced, stabilizing at eight instances, which is considerably lower than what was observed in other experiments. Reducing this hyperparameter also required reducing the number of training epochs[78], as the smaller the batch size, the longer the training took. Thus, two models were trained, with a variable number of epochs: the first model was trained for four epochs and took about four hours, while the second was trained for ten epochs, taking about nine hours. The number of epochs can be considered low, given other examples observed and the size of the solution space. The Adam optimizer[79] was used in both experiments, with a learning rate[80] of 0.0001 and a weight decay[81] of 1e-06.

4.2 Computational Infrastructure and Training

The model consists of the environment (as described previously), which provides the interface through which the agent interacts via its policy – the structure responsible for deciding which actions to take based on the state of the environment.

The policy in this experiment is a neural network composed of two parts: an encoder and a decoder. The encoder produces embeddings (representation vectors) for each point, and the decoder takes the encoder's output as input and is responsible for producing the solution in the form of a sequence of points.

[77] In our context, batch size corresponds to the number of episodes stored before updating the policy or value function. An episode is a complete sequence of agent-environment interactions (i.e., a sequence of steps where each step involves selecting a point, moving to that point, receiving a reward, and transitioning to a new state), from the environment's initial state (no points visited, vehicle at the depot) to its final state (all points visited, vehicle at the depot or last point visited). It should be noted that in many algorithms, batch size refers to the number of samples (state-action-reward vectors) used in gradient descent during the optimization process of the policy or value function.

[78] An epoch is the concept used to structure training. Each epoch is a cycle of training by the agent; it generally corresponds to a set of episodes always performed with the same policy. After completing an epoch, the accumulated experience is used to update the model's policy (or, in other cases, its value function) (Sutton & Barto, 2015).

[79] An algorithm for first-order gradient-based optimization of stochastic objective functions. The Adam optimizer is used in machine learning and deep learning, particularly in training neural networks (Kingma & Ba, 2014).

[80] The learning rate is a hyperparameter used in the training process to determine the step size of the input value adjustment to optimize the loss function in methods using gradient descent; larger learning rates imply faster weight adaptations, which can cause instability in training (Goodfellow *et al*., 2016).

[81] The weight decay is a hyperparameter used in the training process to control a model's tendency to overfit or underfit (Goodfellow *et al*., 2016).

Appendix C provides an exhaustive description of this neural network; a summary of the most significant aspects can be read below.

4.2.1 Encoder

The encoder consists of two layers. The initial fully connected layer encodes the locations into a higher-dimensional space; each location (initially with seven dimensions) is transformed into a 128-dimensional representation vector (corresponding to 7*128 + 128 = 1054 parameters). A separate layer also transforms the depot (with only two spatial dimensions) into a 128-dimensional representation vector (corresponding to 2*128 + 128 = 384 parameters). Thus, the encoder calculates a much larger representation from a reduced-dimensional input through a learned linear projection. The second layer, which is a sequential layer, is a Graph Attention Network, which processes and updates the previous layer's output using five multi-head attention layers. Each of these attention layers consists of a sequence of four sub-layers: SkipConnection -> Normalization -> SkipConnection -> Normalization. The first SkipConnection implements the multi-head attention module, while the second SkipConnection implements the Mixture of Experts module[82], which increases the model's capacity without increasing computation levels. This module replaces the more traditional final linear layer of attention mechanisms (Berto *et al*., 2023, p.35). In addition to this general view of the model, other secondary layers and sub-layers, such as normalization, activation functions, softmax, and soft plus, can be referred to.

4.2.2 Decoder

The decoder has an initial linear layer that encodes the environment context information into a 128-dimensional representation vector. A second layer encodes dynamic characteristics (i.e., those updated during training), such as the remaining vehicle capacity or information about already visited points, into a total of five dimensions. Each of these 133 dimensions is subsequently transformed into 128-dimensional vectors (corresponding to 133*128 = 17,024 parameters).

Additionally, the decoder implements a pointer mechanism, which is important for sequential generation tasks (such as selecting the next point in the route). The decoder also includes another Mixture of Experts module with four linear layers that do not change the size dimension of the input, as they map 128-dimensional vectors into 128-dimensional vectors. This module has two functions: an activation

[82] This module is incorporated only in this specific model.

function (softplus) and another called softmax. The softmax function applied here ensures the output is a probability distribution over the set of following points to choose from.

In summary, the experimental model consists of the following components:

- *Encoder*: Approximately 3.000.000 parameters
  - Uses a Graph Attention Network with multiple Multi-Head Attention Layers.
  - Each layer uses skip connections, normalization (InstanceNorm1d), and a mixture of experts (MoE) mechanism.
- *Decoder*: Approximately 150.000 parameters
  - The context representation and pointer attention use MoE to select the expert network for decoding.
- *Baseline*:
  - A shared baseline was used, meaning the value did not change throughout the training process.

4.3 Evaluation Metrics and Reference Models for Comparison

To evaluate the solutions and provide a measure of comparison, the possibility of applying other methods to the problem was explored, specifically the use of open-source implementations such as OR Tools and PyVRP. However, it was considered that using this software was beyond the scope of this thesis, so a more straightforward approach was chosen.

The *Clarke and Wright Savings Algorithm*[83] (Savings algorithm) was used as a baseline for comparison to evaluate the solutions produced by the trained RL models. The Savings algorithm was implemented from Peng *et al*. (2020), whose code was made available by the authors on GitHub. This algorithm does not incorporate any machine learning mechanism. It is a heuristic approach to solving the VRP in cases where the number of vehicles is not fixed, making it a different method from the one tested in this dissertation.

Darwish *et al*., (2020) define several metrics that offer the advantage of evaluating models against multiple, often conflicting, interests. These metrics provide transparency and the ability to weigh the most important factors in constructing any model. While calculating these metrics for the models trained here is beyond the scope of the thesis (due to the more straightforward reward function used in this study compared to the one the authors employed), they are worth mentioning for their relevance:

[83] Originally developed by Clarke & Wright, (1964).

- Average travel time for all satisfied passengers (including a penalty for each vehicle change);
- Percentage of trips completed with a maximum of two vehicle changes;
- Fleet size required for the network.

In trying to provide a more concrete evaluation and finding metrics more directly related to the problem being studied, the different models were compared in terms of the number of lines (routes) found in the solution, the total distance of the network, the distance of the longest segment, and the network's total journey time.

5 Results and Discussion

In addition to the simplifications made explicit in previous chapters regarding the model's construction, further simplifications were necessary during the model's application: to achieve a final result and enable a comparison between the reality of the Carris network and the solutions found, it was also necessary to define the starting point – the depot – for the vehicles. This simplification does not pertain to defining a starting point, as it exists in reality (vehicles return to a location at night from where they depart again the following day). Although it was impossible to confirm this information, it is reasonable to assume that there is a designated depot in each city zone where vehicles are stored overnight. The assumption was that the depot is centrally located relative to all points in the zone, meaning it is positioned at the average latitude and longitude of all points in the zone. Therefore, the distances between the depot and other points were not considered when comparing the results with those from the Savings algorithm or the Carris network. In other words, when calculating the length of each line in the Carris network, the distance between the depot and the first and last stops of each line was excluded, and this segment was also omitted in the models' calculations and figures.

It was also impossible to accurately model the supply and demand at each location (i.e., how many passengers were waiting at each stop and their destination). For simplicity, it was assumed that half the locations were origin points, and the remaining half were destinations. This assumption also implies that if passengers travel from stop A to stop B, no passengers travel from B to A. Naturally, these simplifications diverge from reality and make the models less capable of solving real-world problems. While it is easy to think of a solution to this issue (for instance, assuming – as is the case in reality – that there are bus stops on opposite sides of the street that different lines should link), it is not as straightforward to determine how to mask points located across the street. It would always be necessary to mask such locations to apply these algorithms to real situations, as the algorithm would naturally consider connecting them with the same line due to their proximity, which clearly does not make sense. For example, it would be necessary to mask already-visited points and those extremely close (e.g., less than 100 meters) to the vehicle's current location.

The experiments yielded the following results:

- *Experiment 1 (Model 1, 4 epochs)*: The average reward during training was -16.42 per episode, and the average reward during testing was -15.36 per episode.
- *Experiment 2 (Model 2, 10 epochs)*: The average reward during training was -14.70 per episode, and the average reward during testing was -14.35 per episode.

The fact that both models had very similar results in testing and training indicates no overfitting, signaling that the models are likely to generalize well to new problems. Additionally, Model 2, which was trained for more epochs, achieved a better reward, meaning it was able to find shorter routes on average.

Regarding the comparison between the results from these experiments and the solution produced using the Savings algorithm (implemented based on the code from Peng *et al*. (2020), with the network diagram available in Appendix D), both trained models significantly outperformed this algorithm when evaluated according to the previously defined metrics. Concerning the total straight-line distance of the network, the Savings algorithm returned a solution with approximately 240 km, which is much greater than the solution from both models, which was around 60 km. Furthermore, the longest straight-line segment produced by the Savings solution was around nine km, whereas the trained models' solutions produced segments of around two km. Thus, the Savings algorithm had an underwhelming performance with far worse results (i.e., higher values) than the Carris network and the solutions from the trained models.

Although the cause of such discrepant results was not analyzed, the poor results achieved with this algorithm may be related to the relatively large problem being solved, the fact that the distribution of points is not uniform (since the solution quality for this algorithm depends somewhat on the initial conditions) or the fact the merging routes are fixed based on the savings list, which can be restrictive (Laporte & Semet, 2002).

Despite this, the Savings algorithm presented one advantage over the experiments: it found a solution with the same number of lines as the Carris network. Unfortunately, defining the number of lines proved challenging during the experiments due to the lack of environments where this parameter could be easily predefined. With the Savings algorithm, it was possible to specify the desired number of lines. Table 2 below presents these results (page 44).

5.1 Application of the Model to the Carris Network

Many algorithms identified in the research were tested under "ideal conditions", such as Mandl's network[84], which serves as a benchmark for many models. In this work – following some of the reviewed literature – it was considered more valuable to apply the model and attempt to find good solutions for real-world problems than to achieve good results in idealized situations. Thus, the following presents the

[84] A dataset for 20 cities in Switzerland.

results obtained from applying the model to a real-world scenario that approximates the Carris network in Lisbon and an evaluation of those results based on the specific metrics defined earlier.

Figure 6 shows the set of routes defined by Carris and the stop locations for Lisbon's North Zone.

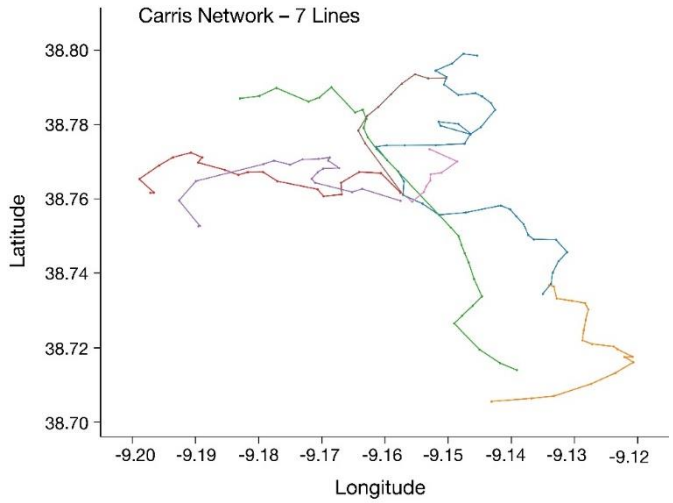

Figure 6. Carris Lines – North Zone. Each color corresponds to a different line.

In Figure 7, the 11 lines generated by Model 1 can be seen. Note the contrast with Figure 6: the routes generated by Model 1 are highly fragmented, and some of the routes are very short, which is directly related to the fact that the model generated more lines than those in the Carris network. Additionally, it is easy to observe areas where the algorithm could have selected shorter paths.

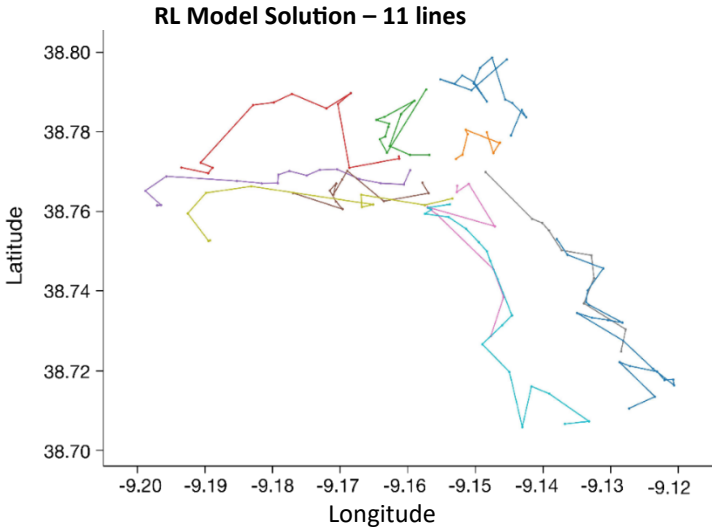

Figure 7. Lines generated by Model 1. Each color corresponds to a different line.

The results were significantly better in Experiment 2: in this case, the solution found had only eight lines, and Model 2 created a network whose design (Figure 8) is closer to that of the Carris network, with longer and more continuous routes, although the length of the shortest route remains small.

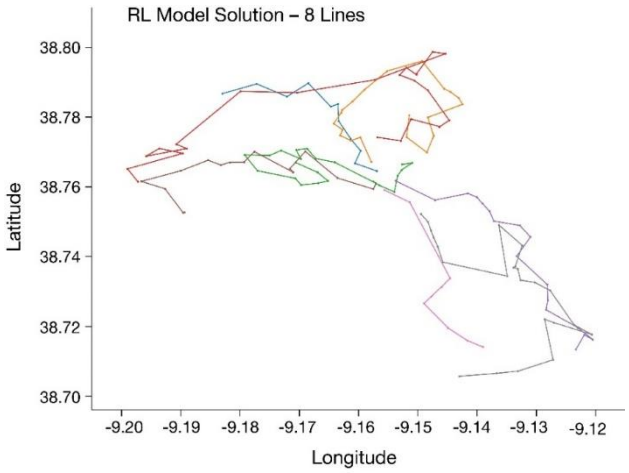

Figure 8. Lines generated by Model 2. Each color corresponds to a different line.

See Table 2, which summarizes the main results and the values of some metrics for the Carris network.

Table 2. Achieved results – RL models and Savings Algorithm

Metric	Carris Network	Modelo 1 (4 epochs)	Model 2 (10 epochs)	Savings Solution
Number of Lines	7	11	8	7
Total Network Distance straight line - Km	47.41	66.86	58.51	237.82
Longest Segment Distance straight line - Km	2.54	1.84	2.49	8.56
Total Network Distance by road - Km	60.88	147.48	125.41	406.67
Network Route Duration by road - min	207.48	452.45	377.77	952.62

The trained models were able to minimize the maximum distance between two points in the network. In the Carris network, the maximum distance is 2.54 km, while in the networks generated by the models, the maximum distances were 1.84 km (Model 1) and 2.49 km (Model 2). This can be considered a good

result, as minimizing the maximum distance between two points in the network was the objective the algorithm sought to achieve (rather than, for example, minimizing the total network distance).

However, the total distances found by models 1 and 2 were significantly worse than those of the current Carris network: the Carris network has a total distance of 47 km, while the network solution generated by Model 2 reached 58 km, making it 19% longer than the Carris network. The solutions found would have benefited from subsequent optimization methods, which operate locally on the network, such as the Lin-Kernighan heuristic[85]. In Model 1, it can be easily observed that the algorithm selected less optimal routes than were possible, as evidenced by the corresponding figure.

Finally, it is worth noting that the road distance found by the model diverged significantly from the current road distance (61 km in the Carris network versus 125 km with Model 2). This discrepancy is due to the fact that the models were trained using geographic coordinates and straight-line distances between them rather than road distances or travel durations between locations.

This was one of the limitations of these algorithms: the environment only allowed the definition of points based on their coordinates without the ability to include road distances between them. Moreover, the way most models handle data – scaling location coordinates to the unit square to standardize and facilitate training – ignores the scale of these problems during training. So, it implicitly assumes, in effect, that solving the problem between cities is identical to solving it within a city or that solving it for a car is the same as solving it for walking or bus travel[86].

However, the difference between routes between cities and those within a city is not just of scale – one is not simply a larger version of the other; they are qualitatively different. Intercity routes tend to resemble a “straight line” where the road distance between two points is approximately equal to the straight-line distance. Using location coordinates or a distance matrix as input is interchangeable in these cases. However, city routes are often winding, and fractal-like, where the straight-line distance between two points rarely corresponds to the actual road distance, and more importantly, the “distance” between two points depends not only on Euclidean distance but also on the urban context of those coordinates. Moreover, the relationship between road distance and travel time between two points is nonlinear.

[85] This method is based on a generalization of the "interchange transformation" method. The central idea of this method consists of exchanging a non-fixed number of edges from a previously found solution, which are then rearranged to create a new solution; it is a local search algorithm aimed at progressively improving an initial solution while avoiding local minima. (Lin & Kernighan, 1973)

[86] In addition to this issue, there are still uncertainties regarding the normalization process of the locations, as certain normalization methods appear to modify the relative distances between points.

In the case of Lisbon, routes can be extremely winding. Although outside the scope of this thesis, each route could be modeled as a random walk, forming a fractal pattern whose dimension – representing the network's irregularities – could be analyzed and calculated, contributing to the study of the network as a whole and potentially serving as a hyperparameter for later model training.

In addition to these considerations, it is essential to note that applying these algorithms to model urban traffic (rather than intercity traffic) adds extra complexity. The VRP in a city context has two characteristics that make it substantially more complex: scheduling, as buses must visit stops multiple times, and there are many interconnections (which are less relevant in intercity applications); and the varied motivations for travel, resulting in more complex movement patterns, with far more stops than in most problems. Moreover, even when close together, these stops do not necessarily need to be connected (passengers may switch transport modes on foot). Finally, factors such as traffic and one-way streets can cause the route from A to B to differ significantly from the route from B to A, causing distance and duration matrices to be highly asymmetrical, contrary to assumptions made by several studies (as in, for example, Darwish *et al*. (2020)).

In conclusion, current tools, particularly the algorithms available in the RL4CO library, still only allow the creation of highly simplified environments compared to reality.

Despite all the simplifications, the results were only slightly better than those of the Carris network regarding the longest segment distance, and neither of the models found shorter routes (in straight-line distance) than those of Carris.

Nevertheless, the results were achieved without requiring prior knowledge of the city, traffic, or routing problems, which can be considered an advantage of the approach explored in this thesis. Additionally, the results suggest that algorithms trained for a specific number of points can be generalized to scenarios with fewer points, indicating that the model can be easily generalized to other situations.

6 Conclusion

6.1 Summary of Findings

This dissertation explored the application of reinforcement learning mechanisms to solve a vehicle routing problem focused on optimizing the public transportation network in Lisbon, specifically part of the Carris bus network.

Using the architecture called Multi-task Vehicle Routing Solver with Mixture-of-Experts, it was possible to obtain satisfactory results in efficiently discovering shorter routes than those in the Carris network. The trained model successfully minimized the length of the longest segment without incorporating any prior knowledge of mathematical optimization or specialized knowledge of Lisbon's complex urban environment. The model achieved this within an acceptable timeframe, especially considering this was a relatively large routing problem. Therefore, this approach could contribute to alternatives to traditional methods, such as exact mathematical or metaheuristic methods.

The research indicates that reinforcement learning can improve the efficiency of public transportation networks by enabling their dynamic optimization. Although the training time may be lengthy, once trained, the model can return solutions to various cases, solving dynamic situations and adapting to new challenges, such as temporary road closures or traffic constraints. Further research into reinforcement learning models could help reduce routes while maintaining service levels, impacting both operators (through cost reduction) and users (through reduced travel time and improved service reliability).

6.2 Challenges and Limitations

Despite the relative success of applying RL methods to the VRP, several challenges were encountered. The complexity of the Carris network and the large number of stops posed computational difficulties, requiring the simplification of the network to a subset of stops. Additionally, while RL models are effective, they depend on extensive computational resources, particularly for large-scale problems. Moreover, in this dissertation’s case, the static nature of the dataset used may limit the model’s ability to handle real-time dynamic changes, such as traffic conditions or shifting travel patterns.

Regarding the broader RL algorithms used to solve this class of problems, the research concludes that much development is still needed for these algorithms to deliver more satisfactory results and be applied to large-scale, real-world cases. As Kevin Tierney from the University of Bielefeld notes, machine learning approaches to mathematical optimization problems are still in their infancy, often not providing the best

solution and only solving straightforward problems[87]. They are also not easily scalable or generalizable, meaning that a new model must be trained if the problem constraints change slightly. In the same vein, Berto *et al*. (2024) state that "despite the recent progress made in learning to solve individual VRP variants, there is a lack of a unified approach that can effectively tackle a wide range of tasks, which is crucial for real-world impact" (p. 1).

These observations align with the results obtained in this dissertation, where, despite simplifications and the successful minimization of the longest segment, the total route distances found were far from ideal and not shorter than those in the Carris network.

6.3 Practical Recommendations for Carris

To improve Carris's operational efficiency, it is recommended that the company consider implementing RL-based optimization in pilot programs. The proposed model could be used to reassess existing bus routes and suggest improvements for underperforming lines based on identified demand patterns. Moreover, RL algorithms could dynamically adjust bus frequencies, ensuring a more adaptable service that aligns with passenger demand in real-time. These actions would likely reduce operational costs and improve passenger satisfaction by minimizing waiting times and optimizing bus allocation.

For buses to be efficiently allocated and routes better defined, it is also necessary to study passenger travel patterns. This study is required because *Carris* currently does not have precise data on where it transports passengers. While the network can track where passengers board, it does not know where they disembark, as passengers validate their tickets upon entry but not upon exit. Furthermore, it is unclear whether *Carris* has data on the number of passengers unable to board buses due to overcrowding. While mobility studies conducted by the National Statistics Institute (*Statistics Portugal* – INE) are important, *Carris* would benefit from more granular, neighborhood-level data.

Recently, it was reported that Carris buses have never been as slow as in 2024, averaging 14 km/h[88]. While much of this is due to external factors (e.g., the increase in cars in the city or major construction projects in Lisbon), improving the routes and adapting them based on predictable situations is a factor to

[87] The full lecture by Kevin Tierney, "Search Heuristics for Solving Routing Problems with Deep Reinforcement Learning", where the author discusses various limitations of the RL approach to optimization problems, from which this statement is taken, can be viewed at https://www.youtube.com/watch?v=nqAubq2K_Ug. (accessed on December 2023)

[88] See the article from Público, "*Em mais de 15 anos, autocarros de Lisboa e Porto nunca estiveram tão lentos*" at https://www.publico.pt/2024/06/20/local/noticia/15-anos-autocarros-lisboa-porto-tao-lentos-2094651. (accessed on June 2024)

consider in enhancing Carris's services. Additionally, studies conducted by INE[89] indicate that one reason the public does not use buses as much is due to the unpredictability of the service and waiting times, factors that could be improved with RL methods. Thus, further exploring the approaches examined in this thesis could provide significant benefits in such situations.

6.4 Future Research

Throughout this dissertation, several aspects were identified for further exploration. From a general perspective, developing the models will necessarily involve expanding the RL model to incorporate more dynamic variables, such as real-time traffic data or fluctuating passenger densities, and integrating additional layers of complexity, such as multimodal transport options (metro, trains, and bike-sharing systems). These developments could provide a more holistic solution to urban mobility challenges.

Another area for development, from a technical perspective, involves scaling the proposed algorithms to more locations[90], which could further validate the model's effectiveness.

Regarding the algorithms used, specifically the RL4CO library, it would be interesting to modify the function that masks locations, allowing specific locations to be visited multiple times (corresponding to points where multiple routes pass, making it possible to change vehicles).

Traffic data was not incorporated into any of the experiments, although including this information in the models was considered. Understanding how to integrate such data into models that do not allow for distance matrices (or an equivalent time matrix with information on travel time between locations, which naturally incorporates traffic information) will be essential. Even if this information cannot be directly included in models (such as with the RL4CO library), it could be applied when implementing the model with real-world data by assigning weights to each edge (i.e., the paths between two points). When normalizing points to the unit square, all locations could be rescaled with a direction parallel to the location-depot vector and magnitude proportional to the ratio between distance (location to depot) and duration (location to depot). This would effectively incorporate the notion of traffic into the final result.

It has already been noted that the models experimented in this thesis currently have several limitations. One necessary development is to address more complex problems – not just solving VRPs with constraints but tackling the entire issue of designing such networks. Constructing the "best" possible

[89] (INE - Instituto Nacional de Estatística, 2017).

[90] During the literature review, some methods currently used to solve much larger instances of the VRP were identified, such as strategies that use "divide and conquer" approaches, where a large problem is divided into several smaller ones (possibly with the help of RL) and then solved with heuristics that work very well for smaller problems.

routes is a much broader problem than simply solving a VRP. In practice, before finding these routes, it is also necessary to define the stop locations (at least for city routes). And after finding the routes, it is necessary to determine, for instance, schedules and timetables. This problem set is called the Transit Network Design and Frequency Setting Problem (TNDFSP). Today, each of these problems is still treated separately (Darwish *et al*. (2020), relying on different data, techniques, and algorithms. For example, the choice of stop locations may depend on theoretical paradigms related to urban planning (what one wants when thinking of a city) and may be made using supervised methods. Finding the "best" routes may mean finding faster or more connected routes, which can be achieved through RL methods. Finally, the scheduling and frequency-setting process may (and should) depend on user travel needs and interconnections with other transport modes (fundamental when designing a public transport network) and may be determined by expert experience. A future research direction would be to develop algorithms capable of optimizing routes across multiple parameters while also defining the stops and setting the timetables and frequencies for the transportation system.

In conclusion, despite the challenges mentioned above, this work clearly shows that reinforcement learning offers a promising avenue to optimize public transport networks in the context of bus routes in Lisbon, through the exploration of higher spatiotemporal resolutions of larger-scale problems.

7 Data Availability

Throughout this dissertation, several materials were produced, including:

- Written dissertation in Portuguese
- Translation of the dissertation into English
- Source code adapted to the problem in question (RL4CO library)
- Various Jupyter notebooks with Python 3.10 code used to extract all the information necessary for the dissertation
- Files containing the model through which the experiments can be repeated
- Excel files with various information about Lisbon bus stops (including latitude and longitude)

Both Portuguese and English texts, the code, the data, the model file, the Jupyter notebooks, and other information can be found in the GitHub repository:

https://github.com/datatad/Optimization-of-Public-Transport-Networks

8 Bibliographical References

Bello, I., Pham, H., Le, Q. V, Norouzi, M., Bengio, S., & Brain, G. (2017). *Workshop track-ICLR 2017 Neural Combinatorial Optimization With Reinforcement Learning*.

Bengio, Y., Lodi, A., & Prouvost, A. (2021). Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, *290*(2), 405–421. https://doi.org/10.1016/j.ejor.2020.07.063

Berto, F., Hua, C., Park, J., Luttmann, L., Ma, Y., Bu, F., Wang, J., Ye, H., Kim, M., Choi, S., Zepeda, N. G., Hottung, A., Zhou, J., Bi, J., Hu, Y., Liu, F., Kim, H., Son, J., Kim, H., … Park, J. (2023). *RL4CO: an Extensive Reinforcement Learning for Combinatorial Optimization Benchmark*.

Berto, F., Hua, C., Zepeda, N. G., Hottung, A., Wouda, N., Lan, L., Tierney, K., & Park, J. (2024). *RouteFinder: Towards Foundation Models for Vehicle Routing Problems*.

Brockmann, D., Hufnagel, L., & Geisel, T. (2006). The scaling laws of human travel. *Nature*, *439*(7075), 462–465. https://doi.org/10.1038/nature04292

Clarke, G., & Wright, J. W. (1964). Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. *Operations Research*, *12*(4), 568–581. https://doi.org/10.1287/opre.12.4.568

CML. (2021). *Inquérito à Moblidade Lisboa*.

Dai, H., Khalil, E. B., Zhang, Y., Dilkina, B., & Song, L. (2017). *Learning Combinatorial Optimization Algorithms over Graphs*.

Darwish, A., Khalil, M., & Badawi, K. (2020). Optimising Public Bus Transit Networks Using Deep Reinforcement Learning. *2020 IEEE 23rd International Conference on Intelligent Transportation Systems, ITSC 2020*. https://doi.org/10.1109/ITSC45102.2020.9294710

Deutsch, D. (2013). *O Início do Infinito - Explicações Que Transformam o Mundo* (1st ed.).

Eldrandaly, K. A., Ahmed A. H. N., & AbdAll A. F. (2008). The 43rd Annual Conference on Statistics, Computer Sciences and Operations Research 22-25 Dec 2008. In Institute of Statistical Studies and Research - Cairo University (Ed.), *Routing Problems: A Survey* (pp. 51–70).

Eric Rosenbaum. (2023, February 11). The ChatGPT AI hype cycle is peaking, but even tech skeptics don't expect a bust. *CNBC - Technology Executive Coucil*. https://www.cnbc.com/2023/02/11/chatgpt-ai-hype-cycle-is-peaking-but-even-tech-skeptics-doubt-a-bust.html

Fawzi, A., Balog, M., Huang, A., Hubert, T., Romera-Paredes, B., Barekatain, M., Novikov, A., R. Ruiz, F. J., Schrittwieser, J., Swirszcz, G., Silver, D., Hassabis, D., & Kohli, P. (2022). Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, *610*(7930), 47–53. https://doi.org/10.1038/s41586-022-05172-4

Gkiotsalitis, K. (2023). Public Transport Optimization. In *Public Transport Optimization*. Springer International Publishing. https://doi.org/10.1007/978-3-031-12444-0

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*.

Google OR-Tools. (2024, March 11). *What is an optimization problem?* Google OR-Tools. https://developers.google.com/optimization/introduction/python#optimization

Ha, S., & Jeong, H. (2022). *Social learning spontaneously emerges by searching optimal heuristics with deep reinforcement learning*.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning*. Springer New York. https://doi.org/10.1007/978-0-387-84858-7

INE. (2023). *O que nos dizem os Censos sobre dinâmicas territoriais*. https://www.ine.pt/xportal/xmain?xpid=INE&xpgid=ine_publicacoes&PUBLICACOESpub_boui=66320870&PUBLICACOESmodo=2

INE - Instituto Nacional de Estatística. (2017). *IMOB - Inquérito à Mobilidade nas Áreas Metropolitanas de Lisboa e do Porto*. https://www.ine.pt/xportal/xmain?xpid=INE&xpgid=ine_destaques&DESTAQUESdest_boui=334619442&DESTAQUESmodo=2&xlang=pt

Kepaptsoglou, K., & Karlaftis, M. (2009). Transit Route Network Design Problem: Review. *Journal of Transportation Engineering*, *135*(8), 491–505. https://doi.org/10.1061/(ASCE)0733-947X(2009)135:8(491)

Kingma, D. P., & Ba, J. (2014). *Adam: A Method for Stochastic Optimization*.

Kirk, R., Zhang, A., Grefenstette, E., & Rocktäschel, T. (2021). *A Survey of Generalisation in Deep Reinforcement Learning*. https://doi.org/10.1613/jair.1.14174

Kool, W., van Hoof, H., & Welling, M. (2018). *Attention, Learn to Solve Routing Problems!*

Kuhn, T. S., & Hacking, I. (2012). *The Structure of Scientific Revolutions*. University of Chicago Press. https://doi.org/10.7208/chicago/9780226458144.001.0001

Laporte, G., & Semet, F. (2002). *The Vehicle Routing Problem* (P. Toth & D. Vigo, Eds.). Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9780898718515

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

Lin, S., & Kernighan, B. W. (1973). An Effective Heuristic Algorithm for the Traveling-Salesman Problem. *Operations Research*, *21*(2), 498–516. https://doi.org/10.1287/opre.21.2.498

Liu, F., Lin, X., Wang, Z., Zhang, Q., Tong, X., & Yuan, M. (2024). *Multi-Task Learning for Routing Problem with Cross-Problem Zero-Shot Generalization*.

Matsuo, Y., LeCun, Y., Sahani, M., Precup, D., Silver, D., Sugiyama, M., Uchibe, E., & Morimoto, J. (2022). Deep learning, reinforcement learning, and world models. *Neural Networks*, *152*, 267–275. https://doi.org/10.1016/j.neunet.2022.03.037

Mazyavkina, N., Sviridov, S., Ivanov, S., & Burnaev, E. (2020). *Reinforcement Learning for Combinatorial Optimization: A Survey*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. https://doi.org/10.1038/nature14236

Nakabi, T. A., & Toivanen, P. (2021). Deep reinforcement learning for energy management in a microgrid with flexible demand. *Sustainable Energy, Grids and Networks*, *25*, 100413. https://doi.org/10.1016/j.segan.2020.100413

Nazari, M., Oroojlooy, A., Snyder, L. V., & Takáč, M. (2018). *Reinforcement Learning for Solving the Vehicle Routing Problem*. http://arxiv.org/abs/1802.04240

Nikolinakos, N. Th. (2023). *A European Approach to Excellence and Trust: The 2020 White Paper on Artificial Intelligence* (pp. 211–280). https://doi.org/10.1007/978-3-031-27953-9_5

Peng, B., Wang, J., & Zhang, Z. (2020). *A Deep Reinforcement Learning Algorithm Using Dynamic Attention Model for Vehicle Routing Problems*.

Pereira, R. H. M., Andrade, P. R., & Vieira, J. P. B. (2022). Exploring the time geography of public transport networks with the gtfs2gps package. *Journal of Geographical Systems*. https://doi.org/10.1007/s10109-022-00400-x

Pestana, D., & Velosa, S. (2008). *Introdução à Probabilidade e à Estatística* (Fundação Calouste Gulbenkian, Ed.; 3rd ed.).

Plaat, A. (2022). *Deep Reinforcement Learning*. Springer Nature Singapore. https://doi.org/10.1007/978-981-19-0638-1

Popova, M., Isayev, O., & Tropsha, A. (2018). Deep reinforcement learning for de novo drug design. *Science Advances*, *4*(7). https://doi.org/10.1126/sciadv.aap7885

Provost, F., & Fawcett, T. (2013). *Data Science for Business* (O'Reilly, Ed.; 1st ed.).

Selukar, M., Jain, P., & Kumar, T. (2022). Inventory control of multiple perishable goods using deep reinforcement learning for sustainable environment. *Sustainable Energy Technologies and Assessments*, *52*, 102038. https://doi.org/10.1016/j.seta.2022.102038

Silver, D. (2016). *RL Course by David Silver - Lecture 1: Introduction to Reinforcement Learning [Video]. YouTube.* https://www.youtube.com/watch?v=2pWv7GOvuf0&list=PLzuuYNsE1EZAXYR4FJ75jcJseBmo4KQ9-

Silver, D., Singh, S., Precup, D., & Sutton, R. S. (2021). Reward is enough. *Artificial Intelligence*, *299*, 103535. https://doi.org/10.1016/j.artint.2021.103535

Soares, J. L. (2005). *Optimização Matemática*. http://www.mat.uc.pt/˜jsoares/.1

Sutton, R. S., & Barto, A. G. (2015). *Reinforcement Learning: An Introduction* (2º ed (in progress)). https://doi.org/10.1109/TNN.1998.712192

Szepesvári, C. (2010). *Algorithms for Reinforcement Learning*. Springer International Publishing. https://doi.org/10.1007/978-3-031-01551-9

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). *Attention Is All You Need*.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2017). *Graph Attention Networks*.

Vinyals, O., Fortunato, M., & Jaitly, N. (2015). *Pointer Networks*.

Wagstaff, K. L. (2010). *Machine Learning that Matters*.

Whittlestone, J., Arulkumaran, K., & Crosby, M. (2021). The Societal Implications of Deep Reinforcement Learning. *Journal of Artificial Intelligence Research*, *70*, 1003–1030. https://doi.org/10.1613/jair.1.12360

Wolfram, S. (2023). *Wolfram - Computation meets Knowledge*. WolframMathWorld. https://mathworld.wolfram.com/AdjacencyMatrix.html

Wu, G., Li, Y., Bao, J., Zheng, Y., Ye, J., & Luo, J. (2018). Human-Centric Urban Transit Evaluation and Planning. *2018 IEEE International Conference on Data Mining (ICDM)*, 547–556. https://doi.org/10.1109/ICDM.2018.00070

Yoo, S., Lee, J. B., & Han, H. (2023). A Reinforcement Learning approach for bus network design and frequency setting optimisation. *Public Transport*, *15*(2), 503–534. https://doi.org/10.1007/s12469-022-00319-y

Zhang, J. E., Wu, D., & Boulet, B. (2022). Time Series Anomaly Detection via Reinforcement Learning-Based Model Selection. *2022 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, 193–199. https://doi.org/10.1109/CCECE49351.2022.9918216

Zhou, J., Cao, Z., Wu, Y., Song, W., Ma, Y., Zhang, J., & Xu, C. (2024). *MVMoE: Multi-Task Vehicle Routing Solver with Mixture-of-Experts*.

Zhou, Q., Yang, Y., & Fu, S. (2022). Deep reinforcement learning approach for solving joint pricing and inventory problem with reference price effects. *Expert Systems with Applications*, *195*, 116564. https://doi.org/10.1016/j.eswa.2022.116564

Appendix A

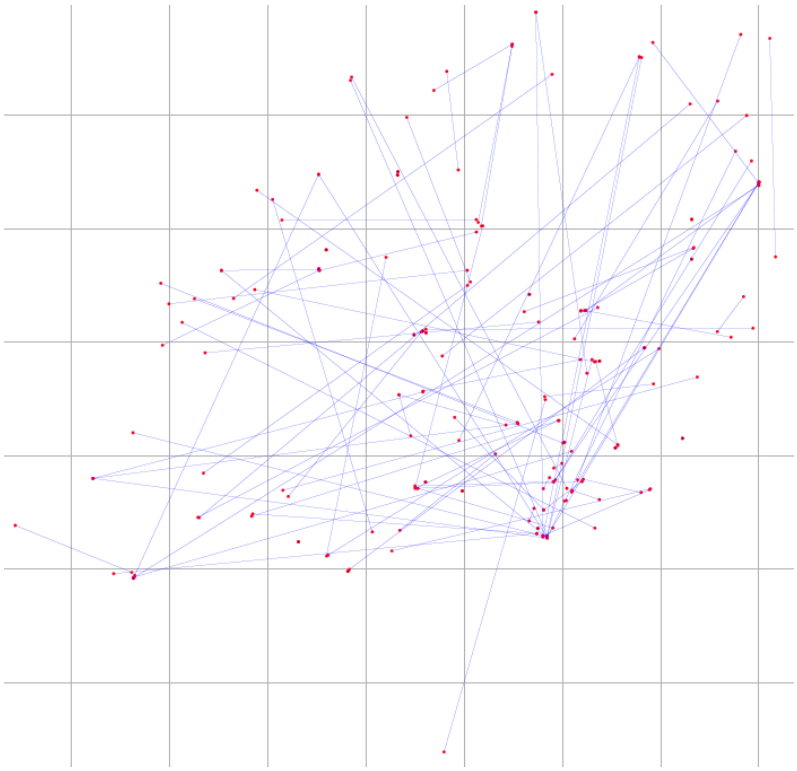

Figure S1. Rough Image of the Carris Network. First and last stop of each line, as of June 2023.

Appendix B

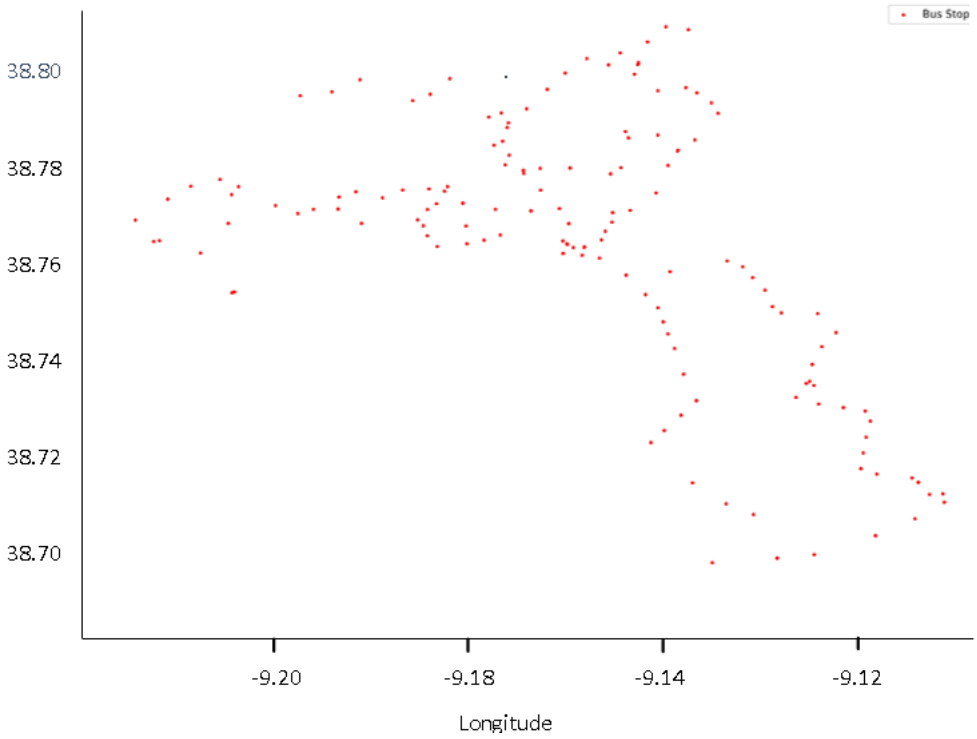

Figure S2. Set of Points in the North Zone. 144 Stops.

Appendix C

Computational Architecture of the Agent's Policy

```
MVMoE_POMO(
  (env): MTVRPEnv()
  (policy): AttentionModelPolicy(
    (encoder): AttentionModelEncoder(
      (init_embedding): MTVRPInitEmbedding(
        (init_embed): Linear(in_features=7, out_features=128, bias=True)
        (init_embed_depot): Linear(in_features=2, out_features=128, bias=True)
      )
      (net): GraphAttentionNetwork(
        (layers): Sequential(
          (0): MultiHeadAttentionLayer(
            (0): SkipConnection(
              (module): MultiHeadAttention(
                (Wqkv): Linear(in_features=128, out_features=384, bias=True)
                (out_proj): Linear(in_features=128, out_features=128, bias=True)
              )
            )
            (1): Normalization(
              (normalizer): InstanceNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=False)
            )
            (2): SkipConnection(
              (module): MoE(
                (experts): ModuleList(
                  (0-3): 4 x MLP(
                    (hidden_act): ReLU()
                    (out_act): Identity()
                    (lins): ModuleList(
                      (0): Linear(in_features=128, out_features=512, bias=True)
                      (1): Linear(in_features=512, out_features=128, bias=True)
                    )
                    (input_norm): Identity()
                    (output_norm): Identity()
                  )
                )
                (softplus): Softplus(beta=1.0, threshold=20.0)
                (softmax): Softmax(dim=-1)
              )
            )
            (3): Normalization(
              (normalizer): InstanceNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=False)
```

```
      )
    )
    (1): MultiHeadAttentionLayer(
      (0): SkipConnection(
        (module): MultiHeadAttention(
          (Wqkv): Linear(in_features=128, out_features=384, bias=True)
          (out_proj): Linear(in_features=128, out_features=128, bias=True)
        )
      )
      (1): Normalization(
        (normalizer): InstanceNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=False)
      )
      (2): SkipConnection(
        (module): MoE(
          (experts): ModuleList(
            (0-3): 4 x MLP(
              (hidden_act): ReLU()
              (out_act): Identity()
              (lins): ModuleList(
                (0): Linear(in_features=128, out_features=512, bias=True)
                (1): Linear(in_features=512, out_features=128, bias=True)
              )
              (input_norm): Identity()
              (output_norm): Identity()
            )
          )
          (softplus): Softplus(beta=1.0, threshold=20.0)
          (softmax): Softmax(dim=-1)
        )
      )
      (3): Normalization(
        (normalizer): InstanceNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=False)
      )
    )
    (2): MultiHeadAttentionLayer(
      (0): SkipConnection(
        (module): MultiHeadAttention(
          (Wqkv): Linear(in_features=128, out_features=384, bias=True)
          (out_proj): Linear(in_features=128, out_features=128, bias=True)
        )
      )
      (1): Normalization(
        (normalizer): InstanceNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=False)
      )
      (2): SkipConnection(
```

```
          (module): MoE(
            (experts): ModuleList(
              (0-3): 4 x MLP(
                (hidden_act): ReLU()
                (out_act): Identity()
                (lins): ModuleList(
                  (0): Linear(in_features=128, out_features=512, bias=True)
                  (1): Linear(in_features=512, out_features=128, bias=True)
                )
                (input_norm): Identity()
                (output_norm): Identity()
              )
            )
            (softplus): Softplus(beta=1.0, threshold=20.0)
            (softmax): Softmax(dim=-1)
          )
        )
        (3): Normalization(
          (normalizer): InstanceNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=False)
        )
      )
      (3): MultiHeadAttentionLayer(
        (0): SkipConnection(
          (module): MultiHeadAttention(
            (Wqkv): Linear(in_features=128, out_features=384, bias=True)
            (out_proj): Linear(in_features=128, out_features=128, bias=True)
          )
        )
        (1): Normalization(
          (normalizer): InstanceNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=False)
        )
        (2): SkipConnection(
          (module): MoE(
            (experts): ModuleList(
              (0-3): 4 x MLP(
                (hidden_act): ReLU()
                (out_act): Identity()
                (lins): ModuleList(
                  (0): Linear(in_features=128, out_features=512, bias=True)
                  (1): Linear(in_features=512, out_features=128, bias=True)
                )
                (input_norm): Identity()
                (output_norm): Identity()
              )
            )
            (softplus): Softplus(beta=1.0, threshold=20.0)
```

```
          (softmax): Softmax(dim=-1)
        )
      )
      (3): Normalization(
        (normalizer): InstanceNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=False)
      )
    )
    (4): MultiHeadAttentionLayer(
      (0): SkipConnection(
        (module): MultiHeadAttention(
          (Wqkv): Linear(in_features=128, out_features=384, bias=True)
          (out_proj): Linear(in_features=128, out_features=128, bias=True)
        )
      )
      (1): Normalization(
        (normalizer): InstanceNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=False)
      )
      (2): SkipConnection(
        (module): MoE(
          (experts): ModuleList(
            (0-3): 4 x MLP(
              (hidden_act): ReLU()
              (out_act): Identity()
              (lins): ModuleList(
                (0): Linear(in_features=128, out_features=512, bias=True)
                (1): Linear(in_features=512, out_features=128, bias=True)
              )
              (input_norm): Identity()
              (output_norm): Identity()
            )
          )
          (softplus): Softplus(beta=1.0, threshold=20.0)
          (softmax): Softmax(dim=-1)
        )
      )
      (3): Normalization(
        (normalizer): InstanceNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=False)
      )
    )
    (5): MultiHeadAttentionLayer(
      (0): SkipConnection(
        (module): MultiHeadAttention(
          (Wqkv): Linear(in_features=128, out_features=384, bias=True)
          (out_proj): Linear(in_features=128, out_features=128, bias=True)
        )
```

```
        )
        (1): Normalization(
          (normalizer): InstanceNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=False)
        )
        (2): SkipConnection(
          (module): MoE(
            (experts): ModuleList(
              (0-3): 4 x MLP(
                (hidden_act): ReLU()
                (out_act): Identity()
                (lins): ModuleList(
                  (0): Linear(in_features=128, out_features=512, bias=True)
                  (1): Linear(in_features=512, out_features=128, bias=True)
                )
                (input_norm): Identity()
                (output_norm): Identity()
              )
            )
            (softplus): Softplus(beta=1.0, threshold=20.0)
            (softmax): Softmax(dim=-1)
          )
        )
        (3): Normalization(
          (normalizer): InstanceNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=False)
        )
      )
    )
  )
  (decoder): AttentionModelDecoder(
    (context_embedding): MTVRPContext(
      (project_context): Linear(in_features=133, out_features=128, bias=False)
    )
    (dynamic_embedding): StaticEmbedding()
    (pointer): PointerAttnMoE(
      (project_out): None
      (project_out_moe): MoE(
        (experts): ModuleList(
          (0-3): 4 x Linear(in_features=128, out_features=128, bias=False)
        )
        (softplus): Softplus(beta=1.0, threshold=20.0)
        (softmax): Softmax(dim=-1)
      )
    )
    (project_node_embeddings): Linear(in_features=128, out_features=384, bias=False)
    (project_fixed_context): Linear(in_features=128, out_features=128, bias=False)
```

```
    )
  )
  (baseline): SharedBaseline()
)
```

Appendix D

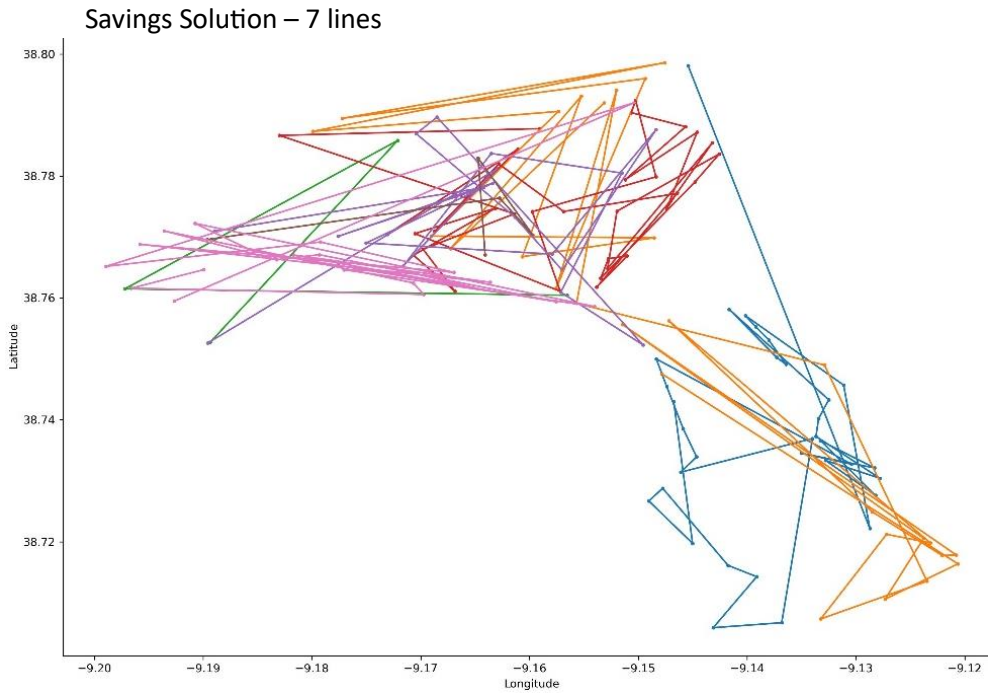

Figure S3. Lines Generated by the Savings Algorithm – North Zone (Savings Solution). Colors correspond to different lines.

Appendix E

Source Code Modifications

Some of the modifications made to the source code are listed here, specifically those related to the RL4CO library. These changes may result in improvements to the source code available on the *GitHub* platform.

I. After properly installing the PyVRP library (a solver that allows comparison of results obtained by RL models), the following line was executed:

```
pyvrp_actions, pyvrp_costs = env.solve(instances = td, max_runtime = 5, num_procs = 10, solver="pyvrp")
```

which resulted in the error:

```
ImportError: PyVRP is not installed. Please install it using `pip install -e .[solvers]`.
```

This error was due to an incorrect import in the source code, as it incorrectly indicated the location of a function. To resolve this, the following file was modified:

```
./rl4co/envs/routing/mtvrp/baselines/solve.py
```

where it previously read:

```
import routefinder.baselines.pyvrp as pyvrp
```

was changed to:

```
import pyvrp as pyvrp
```

II. I removed all instances of max_runtime from the following files:

```
/usr/local/lib/python3.10/dist-packages/rl4co/envs/routing/mtvrp/env.py
/usr/lib/python3.10/multiprocessing/pool.py
/usr/local/lib/python3.10/dist-packages/rl4co/envs/routing/mtvrp/baselines/solve.py
/usr/local/lib/python3.10/dist-packages/rl4co/envs/routing/mtvrp/baselines/pyvrp.py
```

III. In the file

```
/usr/local/lib/python3.10/dist-packages/rl4co/envs/routing/mtvrp/baselines/solve.py
```

where it previously read:

```
_solve = solvers[solver]
    func = partial(_solve, max_runtime = max_runtime, **kwargs)
```

was changed to:

```
_solve = solvers[solver]
    func = partial(_solve, stop = 10, **kwargs)
```

IV. Still related to the lines from point I:

```
pyvrp_actions, pyvrp_costs = env.solve(instances=td, num_procs=10,
solver="pyvrp")
```

a new error occurred:

```
AttributeError: 'TensorDict' object has no attribute 'num_locations'
```

Therefore, it was necessary to manually add this information to the original environment tensor (which did not contain direct information about the number of stops).

V. When running the following line (in the file TNDFSP_aplicação_V2):

```
with torch.no_grad():
  out = policy(td.clone(), decode_type='greedy', return_actions=True)
```

it resulted in the error:

```
AssertionError: Cannot use subsample if variant_preset is not specified.
```

Thus, it was necessary to modify the source code, particularly the file specified in the error:

From `variant_preset=None`, changed to `variant_preset="cvrp"` and from `subsample=True` changed to `subsample=False`