Online Vehicle Routing Under Stochastic Demands Using Approximate Dynamic Programming

Louis Edouard DOUGE

Supervisor:

Associate Professor Chew Ek Peng

Examiners:

Associate Professor Ng Kien Ming

and

Associate Professor Adam Ng Tsan Sheng

National University of Singapore

2018

Online Vehicle Routing Under Stochastic Demands Using Approximate Dynamic Programming

Louis Edouard DOUGE

A thesis submitted
for the degree of Master of Engineering

Department of Industrial and Systems
Engineering and Management
National University of Singapore
2018

Declaration

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

[illegible]

Louis Edouard DOUGE

May, 24th 2018

To my father

Acknowledgments

I would like to thank Prof. Lee Loo Hay and Prof. Chew Ek Peng, my supervisors, for their regular feedbacks. They pushed me to constantly improve my work by adopting a critical viewpoint on my ideas.

I would also like to thank Dr. Chenhao Zhou for his steady support throughout this thesis. His guidance and advice have made this thesis possible.

I must also express my gratitude to all my friends for their presence and encouragements.

Finally, I would like to warmly thank my family for their support that has also made this work possible.

Contents

Abstract

This thesis studies an instance of a Dynamic Vehicle Routing Problem (DVRP) under Stochastic Demands, drawn from a real-world situation. Specifically, a single courier must accomplish two kinds of tasks: deliveries that are known from the beginning of operations, and pickups that appear after the beginning of operations, at random times. The objective is to maximize the rewards obtained from serving both types of customers, during a limited period of time. Our contribution lies in the use of historical couriers' decisions to learn a base heuristic. The Reinforcement Learning framework is then used to make the base heuristic explore new scenarios through simulations. The best trajectories from this exploration process allow the generation of new data to further train the base policy. The enhanced policy thus obtained is finally used in the real world, in combination with the simulator to estimate the downstream value of available actions. The most rewarding one is finally chosen. We show that under some conditions, our approach allows the serving of on average 4.4% more customers than the Nearest-Neighbor policy.

List of Tables

List of Figures

List of Symbols

L	total number of locations
$\{1, \ldots, L\}$	all possible indexed locations
k	current step
S	state space
A	action space
Σ	scenario space
$S_k = (l_k, t_k, \mathcal{D}_k, \mathcal{P}_k)$	state of the environment at step k
l_k	current location at step k
t_k	current time at step k
$\mathcal{D}_k = \{d_k^{(1)}, d_k^{(2)}, \ldots\}$	set of remaining deliveries at step k
$d_k^{(i)}$	remaining delivery at location i at step k
$\mathcal{P}_k = \{p_k^{(1)}, p_k^{(2)}, \ldots\}$	set of remaining pickups at step k
$p_k^{(i)}$	remaining pickup at location i at step k
$A_k = \mathcal{D}_k \cup \mathcal{P}_k$	space of available actions at step k
x_k or a_k	action taken at step k
μ_k	mean of the travel time between l_k and x_k
W_{k+1}	exogenous information between steps k and $k+1$
σ_k	standard deviation of the travel time between l_k and x_k
$S^M : S \times A \to \mathcal{S}$	transition function
$g : S \times A \to R$	single stage reward function
β_d	reward for serving one delivery
β_p	reward for serving one pickup
$\pi_0 : S \to A$	policy
$\sim \pi_0 : S \to A$	policy in exploratory mode

T	maximum number of steps
$\mathcal{T} : S \times A \to R$	traveling time function
N_{monday}	number of customers on a specific day (here Monday)
$\mathcal{S}$	scenario
$\mathcal{N}_d$	number of deliveries during the day
$\mathcal{N}_p(t)$	number of pickups appearing after time t
$\mathcal{G}_d$ and $\mathcal{G}_p$	geographical distribution of resp. deliveries and pickups
$\rho : R \times R \to [0, 1]$	spatial probability distribution
K	Gaussian Kernel
h	Bandwidth of the Gaussian Kernel
$\mathcal{N}_d^{(i)}(t)$ and $\mathcal{N}_p^{(i)}(t)$	number of customers at location i ($\in \{1, \dots, L\}$)
$\mathcal{R}_i$	distribution giving pickup's ready time at location i ($\in \{1, \dots, L\}$)

Chapter 1

Introduction

1.1 Last-mile delivery: background

The last-mile delivery industry has experienced tremendous growth for the past few years. In 2015, the Courier-Express-Parcel (CEP) sector grew by 7 to 10% in large economies like Germany or the United-States. As reported in a McKinsey report (Joerss, Schroder, Neuhaus, Klink, & F., 2016), this is largely due to the growth in the e-commerce sector. The trend is set to accelerate in the future with an important fact to notice: Business-to-Consumer (B2C) is growing at a faster pace than Business-to-Business (B2B). Known to be already highly competitive, the CEP sector has seen the emergence of new business models to face this trend, like Deliveroo, a British online food delivery company using self-employed bicycles, motorcycles or car couriers. In this context, any improvement of the margins by a few percentage points can provide a significant advantage.

However, large logistics providers like FedEx or DHL are best placed to face the global surge in last-mile deliveries. The need for investments to meet worldwide demand is extremely large and it is not surprising to see that leaders in the market are those who invest the most. As reported in an Accenture report (*Adding Value to Parcel Delivery*, 2015), Amazon has spent US$3 billion since 2010 in warehousing and fulfillment to support its business. Alibaba has been spending US$1.5 billion over the past three years to expand the logistics networks in China. Numerous mergers and acquisitions these years like the US$4.4 billion bid for TNT, underline the need to build scale in order to make profits from a low-margin activity. However, investments, as well as mergers and acquisitions, are not the only way to gain the upper hand in this competitive sector.

Improving operational efficiency is determinant too and is an opportunity for smaller companies to compete, at least locally, against large logistics CEP multinationals. Asset-light companies, most often sharing economy startups such as Deliveroo decrease operating costs by bringing advanced technologies like routing algorithms or online information to their couriers, thanks to connected smartphone applications.
Though these companies' focus is mainly on B2C deliveries for a specific sector, there are examples of delivery companies tackling both B2B and B2C. eCourier Ltd is an early example of a small company succeeding locally on both. The company developed a fleet management system as described in an academic paper (Attanasio, Bregman, Ghiani, & Manni, 2007). The margin increase brought by the technology has allowed the company to succeed in the London area and to remain as of today, an important actor in London's last-mile delivery sector.

In Singapore, densely populated areas make it especially suitable for delivery companies to work on both B2B and B2C too. Regular commercial deliveries may occur near residential areas where the probability of an individual requesting to receive or send a parcel is high. It is tempting to assign couriers to serve both tasks and gain in efficiency. But how should these new requests be treated? Should the courier have less commercial deliveries to free some time to potential new customer requests? What should this amount of time be? This amount of time would certainly depend on the time of the year: for example the days before Christmas would be a period of intense home deliveries. Is this strategy even efficient? What about scheduling ahead of time a longer route serving both commercial areas and regions where new requests are highly probable at the time of the courier's passage? This study explores some methods to answer these challenging questions.

1.2 A real case

We examine now the case of a large logistics company in Singapore, facing both commercial (B2B) and individuals (B2C) deliveries as introduced in the previous

section. The objective is to see how the company answers the previous questions, and what are the associated limitations.
The company receives parcels in a depot on the outskirts of Singapore from potentially anywhere in the world. The last-mile delivery process starts with clusters allocation to each courier as shown in Fig.1.1. Couriers should stay as much as possible in their assigned sectors, for both commercial or individual deliveries. These clusters are assigned based on past experience and updated only occasionally. Thus, a parcel yet to be delivered falls automatically into one of these clusters and its associated courier. This can result in unbalanced workloads from one courier to another. Therefore, it is not uncommon that couriers negotiate amongst themselves before leaving the depot, to smooth the workload differences. However, this results in an increase in the number of jobs for some couriers.

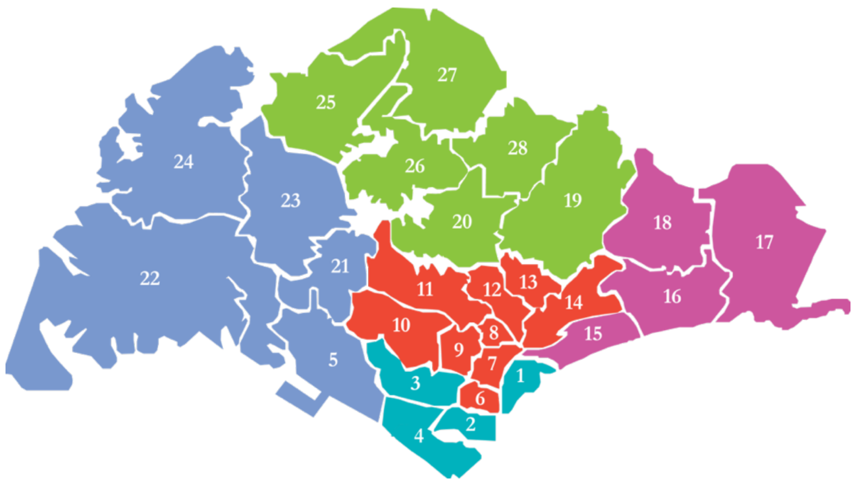


Figure 1.1: Allocation by clusters based on postal codes

In the particular settings of our company, couriers also need to pick up parcels with some constraints. These pickups, usually from commercial customers occur throughout the day and are not known in advance. They also need to be satisfied before a deadline, which is usually the closing time of the customer's business.

The current strategy adopted by the company consists in giving an important autonomy to the couriers and the way they handle known and unknown customers in their cluster. However, an emphasis is put on deliveries (known customers) in the morning, in order to free time for pickups (unknown customers) in the afternoon. This approach seems reasonable as serving the deliveries first, gives more time for the pickups to appear. In turn, this allows the courier to plan its route at the beginning of the afternoon with more certainty than in the morning, as less and less new customers are likely to appear.

However, this methodology does not make use of potential patterns in the way pickups appear through time. What if we could exploit known pickup patterns? It should be possible to modify the courier's route ahead of time in order to increase the chance to be close to new pickups while serving deliveries. It would be also interesting to combine these patterns with the knowledge of traffic conditions.

1.3 Focus and Assumptions

In this study, we intend to improve the delivery process efficiency at the courier level by providing the sequence of locations to visit. We precisely consider the patterns of the way pickups appear, along with the traveling times and courier's experience. We start by learning a base policy from thousands of historical decisions made by couriers in many different settings. This policy informs the courier which location to visit next. Then, we introduce a method where the policy gets to explore a simulated environment that reproduces patterns previously observed in the real world. We are finally able to take a decision from this policy-simulator interaction to effectively perform an action in the real world. This is repeated once the state of the environment has been observed. Fig.1.2 provides a summary of this sequential procedure.

Our work will focus on the morning operations only. This period is critical as most of the deliveries must be satisfied while a large part of the daily pickups' requests also appears. An efficient delivery procedure during this period allows relieving couriers' workload in the afternoon.

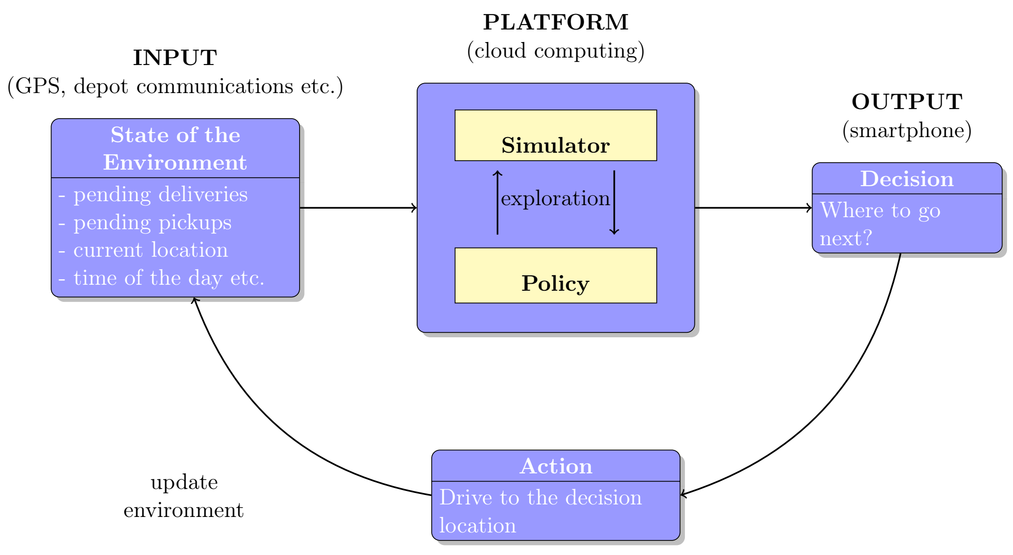


Figure 1.2: Sequential Procedure to route the delivery van

We assume the following points throughout our study:

- the delivery van's capacity is not a binding constraint (as it is the case in reality),
- a finite number of locations to visit is considered (postal codes will be used in the numerical experiments),
- couriers only receive rewards when serving a customer, and there is no penalty for not serving known or previously unknown customers. However, rewards are higher for deliveries than they are for pickups,
- No holding costs and no finite capacity are considered for customers

1.4 Contribution of the thesis

Our contribution can be summed up as follows: we built a suboptimal policy to a DVRP with Stochastic Demands (DVRPSD) that is faced by a real-world logistics company in Singapore. It combines a Policy Approximation method using Supervised Learning methods on historical decisions. It then improves itself with Monte

Carlo simulations to include complex environment's patterns represented as distributions learned from real-world data. When deployed, the policy is eventually used in a Sample Average Approximation (SAA) algorithm to evaluate the downstream value of taking actions. Our implementation has taken care of allowing an important degree of modularity: modules such as travel time models or demands models can easily be replaced to fit company's needs. It is finally important to mention that a parallel implementation has been developed to allow our program to be used in an operational context.

Though we are clearly in the logistics settings, this problem can be extended to ride-hailing services as well as ships routing.

1.5 Structure of the thesis

This thesis is organized in the following way: Chapter 2 presents a literature review organized in three parts. Firstly, an overall background of Dynamic Programming and Approximate Dynamic Programming is given. Secondly, recent elements from the Artificial Intelligence community are discussed, in particular with the studies using Deep Reinforcement Learning techniques. Thirdly, we review more concrete applications of Approximate Dynamic Programming to industrial applications. In Chapter 3, a formal definition of the problem is given along with justifications of using Reinforcement Learning in our problem. Chapter 4 details the two main components of our system: a simulator interacting with a base policy in order to improve it through simulation and retraining. In Chapter 5, we assess the quality of our procedure by comparing it with a myopic policy. Eventually, an overall conclusion of the study is given, including ideas for future research.

Chapter 2

Literature review

This literature review aims at examining three topics. Firstly, we will review existing works on the Dynamic Vehicle Routing Problem (DVRP). For the needs of our study, we will focus on the stochastic version of this well-known problem and we will see how the Dynamic Programming (DP) framework has been used to provide practical solutions. Secondly, we will review some more recent studies on Reinforcement Learning in the Artificial Intelligence (AI) community to highlight the similarities between AI problems and delivery problems. Eventually, we will see that the profusion of data in the delivery industry has driven real-world applications to use Machine Learning (ML) techniques to gain an edge in a very low-margin industry.

2.1 The Dynamic Vehicle Routing Problem

The Vehicle Routing Problem (VRP) introduced in G. B. Dantzig and J. H. Ramser (Dantzig & Ramser, 1959) and its numerous variants have been extensively studied. However, as an extension of the TSP, the VRP is a *NP*-Hard combinatorial optimization problem as shown in (Karp, 1972). In some simple deterministic instances of the problem, optimal solutions can be found as in (Fisher, 1994), where a K-tree method is used. But for larger instances, heuristics are needed to find a high-quality solution in a reasonable amount of time. The well-known Clarke and Wright saving algorithm (Clarke & Wright, 1964) is an example of such a heuristic. However, given the nature of our problem, we need to focus on the dynamic version of this thoroughly studied problem.

2.1.1 From DP to ADP

As mentioned in (Pillac, Gendreau, Guéret, & Medaglia, 2013), there are two dimensions in the dynamic version of the VRP: firstly information can be available from the beginning to the planner (static information), or it can only be available over time (dynamic information), as in our problem where new customer requests arrive over time. Secondly, the nature of information is threefold: it can either be completely known (deterministic), partially known (stochastic) or even unknown. We will see that our problem is dynamic and stochastic in the sense that the information is unveiled through time and is partially known (through distributions).

In this setting, (Pillac, Guèret, & Medaglia, 2012) classifies techniques to tackle these problems as either non-anticipative or anticipative. Non-anticipative methods only exploit currently known information whereas anticipative methods exploit the pattern in the information revealed dynamically. We will not focus on non-anticipative methods as they are mainly used for dynamic and deterministic models. Anticipative methods will be more appropriate in our study as we need to consider stochastic demands.

Among anticipative methods, we can distinguish those based on sampling. These methods use a pool of scenarios generated by sampling distributions, representing a solution to the initial problem. Then, an optimizer improves these solutions. When a real scenario is examined, a decision is taken from the pool of scenarios. These methods include the tabu search heuristic found in (Ichoua, Gendreau, & Potvin, 2006) or the Multiple Scenario Approach (MSA) in (Bent & Van Hentenryck, 2004). Though simple, these methods provide significant improvements for DVRP, especially in high dynamism settings. Their ability to capture the stochasticity of the system will be used in our work.

The other approach among anticipative methods is based on stochastic modeling. The classic way of formulating the DVRP is to use a Markov Decision Process (MDP) as a model for the system and to solve it using Dynamic Programming. The idea is to take decisions that induce a cost-to-go function (immediate cost plus the expected cost) that is minimal, as covered in (Powell, 2013) and (Bertsekas, 2007). The problem is expressed as a sequence of subproblems, solved backwardly in time according to the well-known Bellman's principle of optimality (Bellman, 1954). However, for many real-world applications the number of possible states to be examined to compute the expected cost makes the resolution intractable: the curse of dimensionality. This is where Approximate Dynamic Programming (ADP) comes into play.

In ADP sometimes known as neuro-dynamic programming, the expected cost is estimated instead of being computed. The core methods include the estimation of being in a certain state called Value Approximation, as well as the use of lookahead policies. An interesting study by (Secomandi, 2000) compares these methods in the settings of DVRP with stochastic demands. For the Value Approximation procedure, it creates a Value Approximation function $\tilde{J}$ using a simple linear model. With an initial policy μ (for instance a greedy one), an approximate policy evaluation step is performed. This means that pairs of state x and cost-to-go value $\tilde{J}(x)$ are used to retrain the linear model (least-squares fitting). Secondly, an approximate policy improvement step is performed by simulating the system using the following updated policy:

$$\tilde{\mu}(s) = arg \min_{a \in A} \sum_{s' \in S} p_{ss'}(g(s,a,s') + \tilde{J}(s'))$$

with A the action space available at the current step, S the state space, $p_{ss'}$ the transition probability from state s to state s' and g the immediate cost function. These steps are directly derived from the Policy Iteration algorithm used in Dynamic Programming. The paper shows that this procedure, though memory efficient, is outperformed by a one-step roll-out policy. In a roll-out policy, the former cost-to-go function $\tilde{J}$ is now a closed-form function, with no parameters (for example nearest neighbor). A single approximate policy improvement step is performed. This simple procedure produces very good results on the DVRPSD and should, therefore, be considered in our study.

2.1.2 Curse of dimensionality

(Novoa & Storer, 2009) went further by studying a two-step roll-out policy. The cost-to-go function is again expressed as an immediate cost plus an expected cost obtained from $\hat{J}$, computed as in (Secomandi, 2000):

$$\tilde{J}(s') = \min_{a \in A} g(s', a) + \sum_{s'' \in S} p_{s's''}(u)\hat{J}(s'')$$

It is not surprising to see that the results are further improved at the expense of a higher computational time. This is why the paper introduced the use of Monte Carlo Simulations to compute an approximation of the cost-to-go functions, using a base policy. Given a joint demand scenario d_s, all controls and states from state s'' are simulated until the termination state, leading to an estimation of the cost-to-go function $J(s'', d_s)$. By repeating these simulations, $\hat{J}(s'')$ is approximated by $\sum_{d_s} \frac{J(s'',d_s)}{|d_s|}$ with $|d_s|$ the number of simulations. The paper reports a significant decrease in computational time by about 65% for large instances. This will be considered in our approach.

Given the complexity of our problem's state space, it is important to tackle the curse of dimensionality in several ways. Though Monte Carlo Simulation is an interesting approach, Adaptive Aggregation methods introduced by (Bertsekas & Castanon, 1989) propose to reduce the state space by aggregating it depending on the state of computations. This means that a particular state can be part of different aggregate groups, changing dynamically as time passes. An application in (Grippa, 2016) uses this method for UAV delivery system with stochastic and impatient customers. Features of the state space are aggregated depending on their correlation with the optimal differential net cost of the system. Once the state space is reduced, a Policy Iteration algorithm produces a high-quality solution.

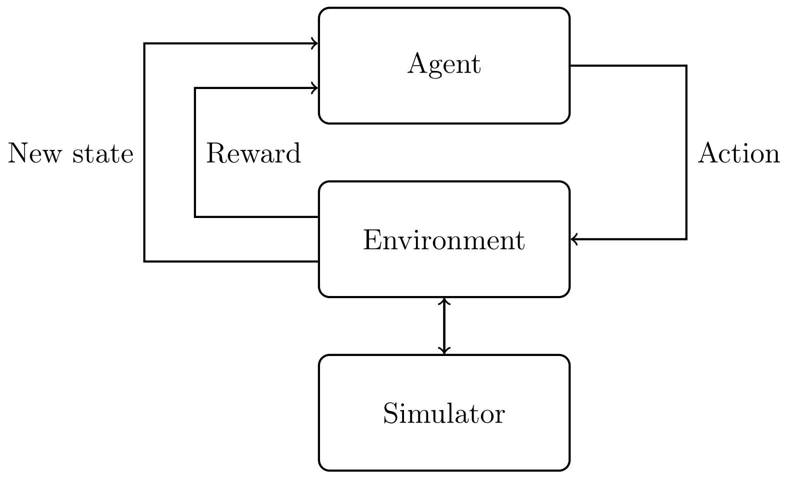


Figure 2.1: Simple representation of the Reinforcement Learning framework

It is important to notice that so far, all the DVRPs were examined using a model (usually a MDP). This is not mandatory.

2.2 Reinforcement Learning

The recent renewed interest for Reinforcement Learning (RL) in the AI community has shown that, under some assumptions, it is possible to tackle many problems without even considering a model as shown in (Sutton & Barto, 1998) and (Szepesvári, 2010). At its core, the RL framework is based on an agent interacting with an environment and receiving rewards depending on the evolution of this environment as shown in Fig. 2.1.

2.2.1 The environment

In the previous subsection, the environment was evolving according to a MDP. In the RL community, it is often replaced by something more complex like a simulator. (Brockman et al., 2016) describes an interesting tool-kit made of various environments called OpenAI Gym, that provides benchmarks to efficiently compare algorithms. It includes Atari games or more complex environments such as a humanoid simulator. This philosophy would be interesting to the research community in Logistics as it gives more modularity to the solution-building process in the DVRP in general. Numerous papers cover their own specific situations. This means resources are to be spent to construct not only a good policy (Agent), but also a complex model (Environment and Simulator) capturing the problem. We will keep in mind this idea of modularity in order to build a solution that can be recreated and compared.

2.2.2 Data encoding and Agent representation

The renewed interest in Reinforcement Learning is also to be found in the impressive results obtained by using Machine Learning and Deep Learning (DL) techniques. In the previously described framework, the cost-to-go function is typically replaced by a neural network. Value Approximation can be combined with the use of Policy Approximation. Policy Approximation consists of learning a function such that given a state of the system, it outputs an action. (Silver et al., 2016) uses both techniques with advanced DL techniques in the context of the Go game. Deep Convolutional Neural Networks (CNN) are used to both approximate the value of positions (value network) and to create a policy (policy network). Taking as input the grid board game, the value network outputs a scalar giving the probability of winning from the current state of the game. The policy network outputs a probability distribution giving for every single board position, its probability of being selected as a move. The training of these networks is complex but is worth exploring.

Firstly, an initial policy network is created from expert moves using supervised learning. This policy is then improved by policy gradient learning in order to maximize the number of wins when playing against previous versions of the policy. Once this policy is trained, many games are generated by self-play to gather a large dataset of state-winning outcome pairs. This eventually allows the training of the value network. These networks are finally used in combination with a Monte Carlo Tree Search by selecting actions using a lookahead search, based on the policy and value network.

Though technically complex, we want to retain from this paper the learning process to create a good policy: initialize policy and value approximations from expert moves, self-play to generate a dataset to train a Value Approximation function and finally perform a Monte Carlo Tree Search.

We focused on the policy learning part. However, an important aspect of these procedures lies in how states are encoded before being fed to the CNN. In the two previous papers, as well as in (Mnih et al., 2013) that applies similar methods to Atari games, states are simply encoded as an image: either the game's grid or the pixels of the Atari games. In (Silver, Hubert, et al., 2017), a more complex approach is used in the context of chess and shogi. Instead of simply providing the chess board to the CNN, a stack of T sets of M planes is provided. Each set contains information of the board positions at each time step between 0 and T (set to zero if not played yet). The M planes represent the presence of each type of piece. On top of this, L planes are added, containing additional information such as player's color, moves count or the legality of castling in chess etc. These complex representations allow capturing a full picture of the game state while being adapted to a learning task using CNN.

This incursion in the field of Deep Reinforcement Learning brings novel ways of designing policies. However, in DVRPs, systems are much more complex than deterministic games such as chess. It is also not straightforward to run millions of simulations as it has been done in the previously cited works. In the next part, we will see that the profusion of data from the industry is an opportunity to use Deep Reinforcement Learning techniques to tackle DVRPs.

2.3 Applications in Industrial settings

2.3.1 Profusion and Importance of Traffic data

As reported in (Horvitz & Mitchell, 2017), rich streams of data are available for an ever decreasing cost of retrieving and storing it. The profusion of internet-connected mobile devices is also an opportunity for the creation of dynamic sensor networks.

(Woodard et al., 2017) introduces a method to exploit GPS data from mobile phones. It creates travel time probability distributions for an arbitrary road segment at an arbitrary time. This technique has been widely used in the past few years by services such as Bing or Google Maps. However, the surge of ride-sharing applications such as Uber or Lyft has required faster and more accurate computations as these services need to perform dynamic pricing (which partly depends on travel times) but also demand forecasting to better match customers needs.

Though the methods to model travel times can be extremely complex and central to the performance of routing systems, it is not the main focus of our study. In light of the fast-evolving techniques in this field, it is a necessity that our study allows travel time models to be easily replaced without impacting the rest of the system, as more accurate models will be built in the future. Therefore, we will be using Bing and Google Maps API as a base model for travel time estimation, while being aware of the existence of possibly better models, especially in Singapore where traffic data is made available online by the Land Transport Authority (LTA).

2.3.2 Stochastic demands, Real-time data and other constraints

Let us now review some works that highlight the importance of online data to build an interesting solution to the DVRP.

As early as in (S. Kim, Lewis, & White, 2005a), a MDP formulation of a DVRP tries to take advantage of real-time information and historical data. The data consists of vehicles average speeds updated every 15 minutes, provided by the Michigan Department of Transportation (MDOT). The paper shows that historical data can already improve delivery costs by as much as 8% during peak hours (9 am to 12 pm) and real-time information can further improve this result to reach 11% for the same period.

More recently, (Cheong & White, 2012) assesses the value of real-time information. It notes that establishing the use of wireless communication and web-based application to communicate with delivery drivers is not a financial issue, but a behavioral one. The study shows also a significant reduction in expected total travel costs due to dynamic tour determination.

(G. Kim, Ong, Cheong, & Tan, 2016) adopts an even more refined approach. Though the approach using a MDP to model the DVRP is common, it considers the arcs between the nodes of the problem, to be the sum of segments where travel times can be modeled as both time-dependent and stochastic. For instance, a route between point A and point B may be comprised of a freeway segment followed by a ramp and then a simple street. Each segment's travel time is modeled as a normal distribution specific to the time of the day, independent of the other segments travel times. The interest and validity of this approach have been established in (Chang, Wan, & OOI, 2009). Eventually, the total travel time is modeled as a sum of normal random variables. Again, the improvements, even small (7% decrease in travel time), are significant given the low margin nature of delivery businesses.

2.4 Industry Examples

Because our problem is being faced by the delivery industry, we need to produce a work that is pragmatic and that can be implemented on a daily basis. There are previous examples of research leading to real-world implementations. As early as in (Attanasio et al., 2007), eCourier Ltd implemented methods from research to help manage a fleet of same-day couriers in London. At its core, one module is dedicated to forecast travel times and demands. Demand forecasting is made using time series extrapolation whereas travel times are forecasted with a time series extrapolation as well, completed by a neural network taking into account real-time traffic and weather. A second module is responsible for courier allocation. It operates in two steps. Firstly, a background optimization procedure optimizes couriers' route in terms of customer waiting time and resource utilization (as vehicles are of different types, they induce different costs) with a Tabu Search. The particularity of the implementation is that it adds to the current jobs, new demands sampled from the demand distribution. This is to create initial routes that are flexible for which deterministic methods can be used. As a second step, it performs a fast insertion procedure to the initial route as developed in (Solomon, 1987) for new incoming requests. Another important point that is not always examined in more theoretical approaches, is the importance of an efficient computer implementation. Parallelism is used to allow for fast decision-making on small devices such as smart phones carried by couriers. This will be another point of interest in our study.

(Simão et al., 2009) develops a complex simulator whose goal is to model the behavior of over 6,000 drivers from Schneider National, the largest truckload motor carrier in the United States. It seems to be the first ADP-based system that produces high quality solutions in this industry setting. The system combines the Value function approximation technique with an aggregation method to reduce the state space's size which is particularly large for this application. It is interesting to notice that policies obtained from suboptimal solutions to the problem, can sometimes closely match policies adopted by the company's dispatchers. These dispatchers' policies can be considered as the equivalent of the expert moves in Go as mentioned in Section 2.2.2. We will build on this remark in our study.

Chapter 3

Problem definition and model formulation

As seen in the previous chapter, online vehicle routing covers a broad range of situations, depending on the problem considered or the assumptions made. This chapter will therefore define the specific problem faced. In Section 3.1, we provide a concise problem description. In Section 3.2, the problem is examined in light of the Reinforcement Learning framework and we justify its use. Eventually, Section 3.3 proposes a formal model of the situation with a mathematical formulation of the problem.

3.1 Problem Description

In the context previously described, it is now possible to formulate our problem. We consider one delivery van operating in a given region (Singapore for the numerical experiments), and serving 2 types of customers during the morning operations:

- deliveries, known since van's departure
- pickups, that appear at random places, over time

Priority is set on deliveries in the sense that they produce higher rewards than pickups. This is to formalize the need to complete the deliveries first during the morning. These rewards remain as parameters that the company can set and modify depending on the needs. Our problem is to maximize the expected accumulated rewards obtained by serving successively any of the available customers (being either a delivery or a pickup), as long as the departure time towards the next customer, does not exceed 12pm whatever the arrival time may be. Indeed, the courier will leave for its lunch break after serving this last customer.

This problem can be formulated as a *Dynamic Vehicle Routing Problem with Stochastic Demands under Time Windows Constraints* (DVRPSDTW). In this work we consider only single agents as well as a 12pm time constraint as mentionned above, but these assumptions can be changed depending on the problem at hand.

3.2 Concepts

This Section shows that our problem's characteristics suggest the use of the Reinforcement Learning framework.

3.2.1 Reinforcement Learning Framework

Our problem consists of a succession of stages: decision, action and observation as shown in Fig. 3.1. Given that the driver is at a certain place, he has to decide among the customers known at the current time, which one to visit next. He then moves to the decided location, with the possibility of having new pickups appearing while driving towards the place. Once arrived on site, the courier delivers to the customer before deciding again which customer to visit next. This process continues as long as the departure time towards the next customer happens before 12pm.

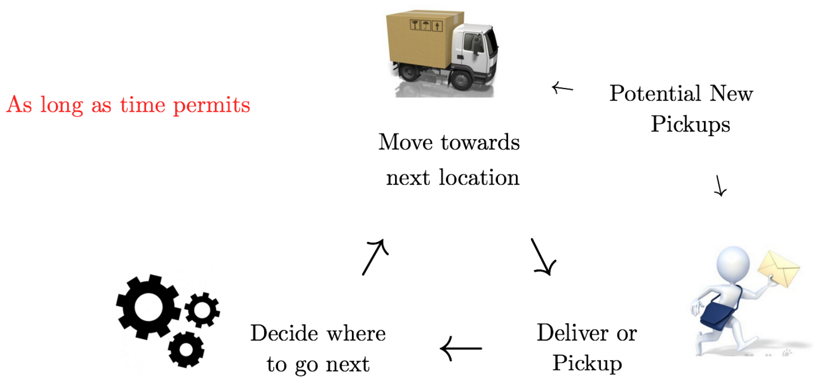


Figure 3.1: Problem description as a cycle

This problem fits well in the paradigm of Reinforcement Learning. As shown in Fig. 3.2, the agent (the courier) interacts with an environment by taking a decision that will produce a new state, and reward the agent. In our case, the process repeats itself over an unknown number of steps, but over a finite horizon (morning operation). Ideally, the objective is to find a policy that produces the most accumulated rewards over time. As the future unfolds in an unknown way, the objective for the agent is in fact to maximize its expected accumulated rewards. The sequential nature of the framework makes it possible to optimize the policy given a particular state. This is desirable as our agent has to be able to adapt to patterns over time, like traffic or pickup trends.

More details concerning states, policy and objective function are given in the Section 3.3.

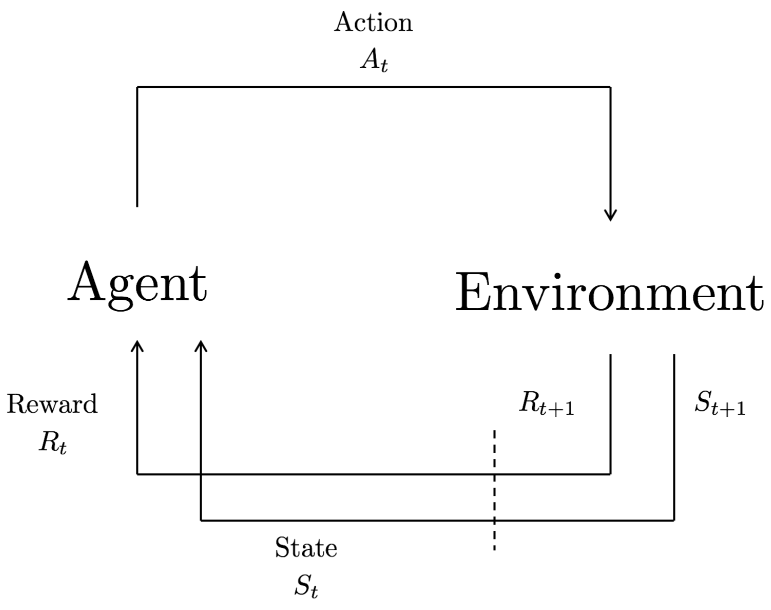


Figure 3.2: Reinforcement Learning Framework

3.2.2 The Agent and its Policy: base policy

After serving a customer, the objective is to know which customer to go next given a specific state of the environment (traffic, jobs remaining to be done etc.). The idea is that a satisfactory policy can be obtained from the courier's knowledge and experience. It is well-known that delivery drivers achieve decent performance at their task by anticipating congested roads or future demands. However, these features are mostly intuition-based. We want to learn these features in order to have a first base policy on which to improve later.

Let us suppose that a driver is available to us. He is able to decide which customer to serve next given the time of the day, the jobs remaining to be done etc. Given a scenario where several customers are available to be served (delivery or pickup), which action should be taken? The idea is simply the following: for each action a, the state that is likely to materialize if choosing this action is generated, and then the driver is asked to decide where to go next. By repeating this process (state generation - driver's decision), a likely virtual trajectory is generated for each action initially available. This trajectory has a certain value (sum of rewards) which provides a rough estimate of the downstream value of choosing an initial action a. This action is finally chosen, producing the highest downstream value.

This entire process relies on having a driver available to make decisions at will. This will be obtained in our first step: supervised learning on past decisions.

3.2.3 Environment

Before going further, we must recreate a realistic environment in which our artificial driver can interact and from which it will learn by exploration. In games like Atari, such interaction between policy and environment (RL) has been successfully applied as it is possible to experiment a lot by virtually playing the game millions of times. It is not as simple for delivery services. We aim at creating an environment that is able to simulate accurately, travel times and pickup trends whatever the current state is. By simulating our agent in this realistic environment, we will be able to explore several possible decisions.

3.2.4 The Agent and its Policy: enhanced policy

Replicating a courier's behavior is a good start but we aim at going further. In Atari (Mnih et al., 2013) or Chess games (Silver, Hubert, et al., 2017), scientists have been able to make the agent learn from playing a lot of games, by allowing some exploration in addition to the pure exploitation of the currently known policy. The objective here is the same: run the virtual courier against a simulator with the possibility for exploration (for instance choose a random next customer, instead of following the policy's decision). From the newly generated data, we would be able to further train our policy initially obtained from historical decisions. Eventually, we can deploy a powerful procedure by coupling this enhanced policy with the process of estimating the downstream value of actions, described above (and known as Sample Average Approximation). To sum up, we present these policy learning steps in Fig. 3.3. The left column describes the successive learning steps (base learning - policy improvement - policy exploitation) while the right column gives the main techniques used to fulfill each learning step.

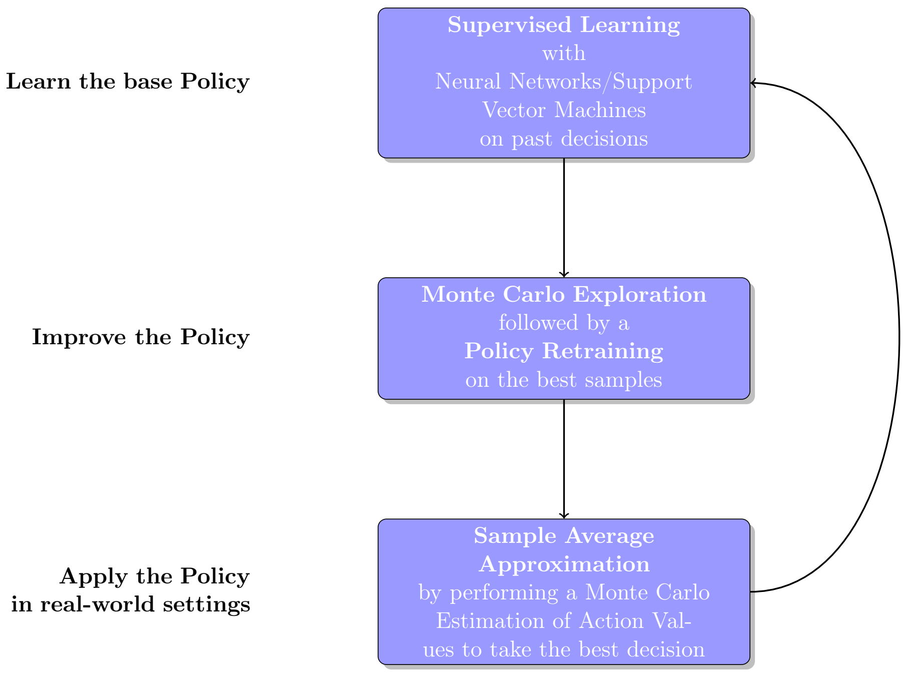

Learn the base Policy
Supervised Learning with Neural Networks/Support Vector Machines on past decisions
Improve the Policy
Monte Carlo Exploration followed by a Policy Retraining on the best samples
Apply the Policy in real-world settings
Sample Average Approximation by performing a Monte Carlo Estimation of Action Values to take the best decision

Figure 3.3: Solution Design Approach

3.3 Model Description

We consider L locations with the set of locations being $\{1, \ldots, L\}$. This number L represents the number of postal codes considered in the experiments and will typically range from a few dozen to a maximum of 16,077.

3.3.1 State

For each stage (decision - action - observation) we define what an agent (the courier) is aware of. At stage k, we define the state $S_k = (l_k, t_k, \mathcal{D}_k, \mathcal{P}_k)$ from the State Space S:

- the last location served: $l_k \in \{1, \ldots, L\}$
- time when leaving to the next customer: t_k (continuous)
- set of remaining customers of type deliveries:
  $\mathcal{D}_k = \{d_k^{(1)}, d_k^{(2)}, \ldots\} \subset \{1, \ldots, L\}$
- set of remaining customers of type pickups:
  $\mathcal{P}_k = \{p_k^{(1)}, p_k^{(2)}, \ldots\} \subset \{1, \ldots L\}$

With these notations, $\mathcal{D}_k \cap \mathcal{P}_k \cap \{l_k\} = \emptyset$ holds, meaning that no location hosts a customer waiting for both being delivered and having its parcel picked-up.

3.3.2 Action

The decision takes place at t_k and tells which location to visit among the Action Space $A_k = \mathcal{D}_k \cup \mathcal{P}_k$ (waiting customers), which is a subset of the action space $A = \{1, \ldots, L\}$

- $x_k \in \mathcal{D}_k \cup \mathcal{P}_k$

3.3.3 Exogenous information

All elements on which we have partial or no control (like travel times or pickups appearance), occurs between t_k and t_{k+1}. It is described as the random variable W_{k+1} and includes:

- travel time and service time: $t_{k+1} - t_k$ modeled as a normal distribution $\mathcal{N}(\mu_k, \sigma_k)$ (see (G. Kim et al., 2016))
- new requests for pickups that can be modeled from real-world data, and depending on $t_{k+1} - t_k$

3.3.4 Transition Function

We define a function which gives the next state, from the current one:

$$S_{k+1} = S^M(S_k, x_k, w_{k+1})$$

As expected, we can already write (decision at step k becomes the current location at step $k+1$):

$$S_{k+1} = (x_k, t_{k+1}, \mathcal{D}_{k+1}, \mathcal{P}_{k+1})$$

3.3.5 Objective Function and Problem Formulation

We define the reward obtained for a single stage:

$$g(S_k, x_k) = \begin{cases} \beta_d & \text{if } x_k \in \mathcal{D}_k \\ \beta_p & \text{if } x_k \in \mathcal{P}_k \end{cases}$$

with w_{k+1} unknown to x_k and $\beta_p < \beta_d$ (as deliveries are more rewarding than pickups in our business case).

We can now define our problem as an optimization problem over an objective function:

Find the best policy $\pi^* : S \to \mathcal{A}$, that maximizes the total expected reward over our specific time horizon:

$$\max_{\pi \in \Pi} \mathbb{E}^{\pi} \left\{ \sum_{k=0}^{T} g(S_k, \pi(S_k)) | S_0 \right\}$$

with

$$T = \max\{k \in \mathbb{N}, k \leq |\mathcal{A}||t_k \leq 12pm\}$$

3.3.6 Optimal Formulation

As a Reinforcement Learning problem, or Dynamic Programming, the optimal way of solving is to start from what is identified as the final state, say S_T, and find the optimal decision at each step following a backward recursion, by solving:

$$\begin{cases} x_k = & arg\max_{x \in \mathcal{P}_k \cup \mathcal{D}_k} \left[g(S_k, x) + \mathbb{E}^{\pi} \left(\sum_{i=k+1}^{T} g(S_i, \pi^*(S_i)) \right) \right] \\ x_T = & \textbf{Final State} \end{cases}$$

However defining what is T in our case is impossible as the number of steps is not known in advance. Thus, x_T cannot be analytically described. Traditional Mixed-Integer Programming methods is a possibility to express a theoretical mean for an optimal resolution.

However, even if a formal and optimal way of solving could be found, it would involve looking through all states, which leads to the traditional problem of the curse of dimensionality. Our focus is not to propose an intractable way of finding the optimal solution, but to create a heuristic leading to very good performance.

Chapter 4

Methodology: Simulator and Heuristic

In this Section, the architecture of the simulator as well as the heuristics are described. Section 4.1 details the construction of an algorithm simulating the environment on which a policy can act. Section 4.2 eventually details the way a base policy is learnt from historical decisions made by couriers.

4.1 Simulator

Before going further into the details of how a policy is learnt as previously described, this Section will detail the construction of a simple and yet realistic simulator. The simulator at step k, is essentially the function presented below:

$$generateNextState(x_k, S_k, \mathcal{S}) = S_{k+1} = (x_k, t_{k+1}, \mathcal{D}_{k+1}, \mathcal{P}_{k+1})$$

where:

- $t_{k+1} - t_k$ is the traveling time
- $\mathcal{D}_{k+1} = \mathcal{D}_k$ if x_k is a pickup, else $\mathcal{D}_{k+1} = \mathcal{D}_k \setminus \{x_k\}$ (delivery served)
- $\mathcal{P}_{k+1}$ is updated from $\mathcal{P}_k$ by unveiling pickups with a time of appearance between t_k and t_{k+1} according to a certain roadmap or *scenario*.
- $\mathcal{S}$ is this scenario

The traveling times and the scenario giving the evolution of pickups from $\mathcal{P}_k$ to $\mathcal{P}_{k+1}$ will be given by sampling specific distributions.

4.1.1 Traveling times

We denote by $\mathcal{T}(S_k, x_k)$ the random variable giving the travel time necessary to travel from the current location l_k to x_k at time t_k of the day. We describe a first method that describes this random variable as a normal distribution whose mean and variance are obtained through regression. A second method consists of directly using Google Maps or Bing API which is costly and cannot scale to millions of simulations.

Regression Models

300 paths have been randomly selected among all the trips done in the past by couriers. These trips range from a few meters to several kilometers. For each trip, the traveling time has been retrieved, considering traffic conditions using the Google Maps Directions API, every 30 minutes (for instance from 8:30 am to 7:00 pm every 30 minutes). Results are presented in Fig. 4.1. We perform a simple least squares 2^{nd} order-polynomial regression for the dataset collected every 30 minutes, as it leads to better results compared to simple models like first order linear regression or exponential regression etc.

Let d be the trip distance. Then, the following model explains the travel time t necessary to travel this distance, given a 30-minute period j of the day:

$$t = f_j(d) = c_j + b_j d + a_j d^2$$

with which a prediction interval is associated. A prediction interval for a travel time t, given the travel distance d^* is defined by:

$$\hat{t} \pm \underbrace{\tau^*_{n_{trip}-2} s_x \sqrt{1 + \frac{1}{n_{trip}} + \frac{(d^* - \bar{d})^2}{(n_{trip}-1)s_d^2}}}_{PI_k}$$

where

- $\hat{t}$ is the average expected value of t
- n_{trip} is the number of trips used to study the relationship between distance and travel time ($n_{trip} = 279$)
- s_x is the standard deviation of the samples

- $\tau^*_{n_{trip}-2}$ is obtained from a t-test
- s_d is the standard deviation of the residuals:

$$s_d = \sqrt{\frac{\sum(t_i - \hat{t}_i)^2}{n-2}}$$

Based on the Appendix of (G. Kim et al., 2016) about the assumption of normality for travel times, we adopt a normal distribution specific to each 30-minute period, $\mathcal{N}(\mu_k, \sigma_k)$ with:

$$\mu_k = f_{i^*}(d_{kk+1}) \text{ and } \sigma_k = PI_k$$

where d_{kk+1} is the distance to be covered by a truck between the current location l_k and the decision x_k, i^* the corresponding 30-minute period which t_k belongs to (for instance if t_k =10:20 am, the closest measurement would be the one at 10:00 am, assuming that data has been retrieved every 30 minutes from 8:30 am).
With these notations and noting d_{kk+1} as $||x_k - l_k||$, it is finally possible to write $\mathcal{T}(S_k, x_k)$ as follows:

$$\mathcal{T}(S_k, x_k) \sim \mathcal{N}(f_{i^*}(||x_k - l_k||), PI_k)$$

Direct use of the Google Maps Directions API

This methods simply queries the Google Maps Directions API on the very specific path for which we need to know the travel time. Among the parameters to be specified in the API call, it is possible to include the time of departure, which is exactly t_k in our model.

4.1.2 Simulating Deliveries and Pickups according to a scenario

As part of the evolving environment, new pickups can appear. But deliveries (still deterministic from the agent perspective) need also to be set before launching the

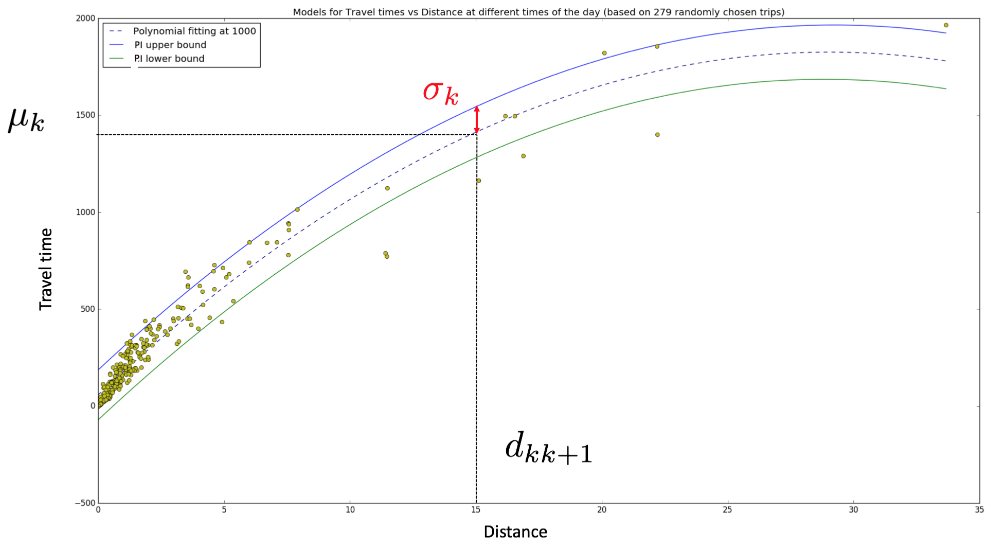


Figure 4.1: Regression obtained for distance vs travel time data, retrieved at 10:00 on the day of reference.

simulation. We present the concept of scenario.

Scenario Replayer

Firstly, we simply replayed past scenarios. $\mathcal{D}_0$ consists of the exact same deliveries found in past operations. The evolution of $\mathcal{P}_k$ is simulated by unveiling the new pickups for which the time of appearance is smaller than t_k, according to the same chronology encountered in the past. This has the advantage of being realistic, but becomes a problem when we test a policy that has been learnt on the scenario dataset: as we will see in the next Section, scenarios used in the simulator need to be separated from the training set of our heuristic.

Scenario Generator

A second way to simulate demands is to generate *scenarios* beforehand: deliveries, as well as pickups and their time of appearance, are sampled from distributions. It is important to note that these scenarios are generated for the Simulator to play them. The agent is not aware of the full scenario when starting operations: he only sees the unveiling of the scenario provided by the simulator depending on his decisions.

Scenarios will be obtained from the following distributions:

1. $\mathcal{N}_d$: number of deliveries during the whole operations period

2. $\mathcal{N}_p(t)$: number of pickups appearing after time t

3. $\mathcal{G}_d$ and $\mathcal{G}_p$: geographical distribution of respectively, deliveries and pickups

4. $\mathcal{N}_d^{(i)}(t)$ and $\mathcal{N}_p^{(i)}(t)$: number of customers at location i ($\in \{1, \dots, L\}$)

5. $\mathcal{R}_i$: distribution giving the ready-time of a pickup at location i ($\in \{1, \dots, L\}$)

Eventually, generating a scenario consists of sampling these distributions as explained in the procedure Alg. 1. We add as inputs the sets of deliveries and pickups ($\mathcal{D}_s$ and $\mathcal{P}_s$). This will be useful when deploying the algorithm where scenarios will have to be generated from a certain time t, given an already known history (see Section 4).
In real-world applications, these distributions will have to be trained. It is also to be noted that the sampling of delivery locations will not be necessary, if the policy is trained once deliveries are done. In this case, only pickups are sampled and fixed deliveries are considered.

Remark: we adopt the C++ syntax to access attributes of objects. For instance, deliveries of a scenario $\mathcal{S}$ are denoted as $\mathcal{S}$.delivery. The practical structure of the scenarios is described in Appendix B.

Algorithm 1 *generateScenario($t_s, \mathcal{D}_s, \mathcal{P}_s$)*

```
Input $t_s$ starting time,
        $\mathcal{D}_s$ existing deliveries,
        $\mathcal{P}_s$ existing pickups
Output Scenario Object
Initialization $\mathcal{S}$ Scenario Object including $\mathcal{D}_s$ and $\mathcal{P}_s$

// Sample number of locations
$n_d \sim \mathcal{N}_d(t_s)$                ▷ number of delivery locations appearing after $t_s$
$n_p \sim \mathcal{N}_p(t_s)$                ▷ number of pickup locations appearing after $t_s$
// Sample deliveries

for $i = 1$ to $n_d$ do
    $g_d^{(i)} \sim \mathcal{G}_d$                                ▷ location
    $n_d^{(i)} \sim \mathcal{N}_d^{(g_d^{(i)})}$                       ▷ number of deliveries at $g_d^{(i)}$
    // Sample pickups
    $\mathcal{S}$.delivery.append( $g_d^{(i)}$ : $n_d^{(i)}$)

for $j = 1$ to $n_p$ do
    $g_p^{(j)} \sim \mathcal{G}_p$                                ▷ location
    $n_p^{(j)} \sim \mathcal{N}_p^{(g_p^{(j)})}$                       ▷ number of pickups at $g_p^{(j)}$
    $\mathcal{S}$.delivery.append( $g_p^{(j)}$ : [ ])

    for $l = 1$ to $n_p^{(j)}$ do
        $r_l^{(j)} \sim \mathcal{R}_{g_p^{(j)}}$                    ▷ Time of Appearance
        $\mathcal{S}$.delivery.$g_p^{(j)}$.append($r_l^{(j)}$)
return $\mathcal{S}$
```

$\mathcal{N}_d$ and $\mathcal{N}_p(t)$

The number of deliveries during the day $\mathcal{N}_d$ is fixed in a first step. The number of pickups appearing after time t, $\mathcal{N}_p(t)$ is also a deterministic number in the simulations, obtained from the dataset (see Section 5).

$\mathcal{G}_d$ and $\mathcal{G}_p$

$\mathcal{G}_d$ and $\mathcal{G}_p$ are obtained using a Kernel Density Estimation technique on a dataset. Considering N_c past deliveries, located at $(c_i)_{i\in[1,N_c]}$ (geographic coordinates), a density estimation of $\mathcal{G}_d$ is given by:

$$\rho(y) = \sum_{i=1}^{N_c} K(\frac{||y - c_i||}{h})$$

with:

- a geographic coordinate $y = (y_1, y_2)$
- a Gaussian Kernel $K(x, h) \propto exp(-\frac{x^2}{2h^2})$
- a bandwidth h

The same method is used to determine $\mathcal{G}_p$ using past pickups samples.

$\mathcal{N}_d^{(i)}(t)$ and $\mathcal{N}_p^{(i)}(t)$

The number of customers at location i, $\mathcal{N}_d^{(i)}(t)$ and $\mathcal{N}_p^{(i)}(t)$, is a normal distribution centered on the average number of customers each location has seen in the past (see next part for a justification of this pattern).

$\mathcal{R}_i$

The ready-time of a pickup at location i, $\mathcal{R}_i$, is also a normal distribution whose average is the mean time of the pickups' times of appearance occurring at location i. The standard deviation of this distribution will be an important parameter to study.

In these Sections, we described how to initialize and run a simulator. In the next Section, an agent is constructed by interacting with the simulator.

4.2 Heuristic Building

In this section, we first propose a method to learn the following function from a real-world dataset (past couriers' decisions). Secondly, we build an improvement procedure of the following policy:

$$\begin{array}{ccc} \pi_0 : S & \rightarrow & \text{A} \\ X = \begin{pmatrix} t\text{: current time} \\ \lambda\text{: current location} \\ \mathcal{D}\text{: remaining deliveries} \\ \mathcal{P}\text{: remaing pickups} \end{pmatrix} & \mapsto & Y = (x \in \mathcal{D} \cup \mathcal{P}) \end{array}$$

4.2.1 Base Heuristic

The idea is that couriers already do a decent work at making successive decisions to serve their customers, anticipating demands and taking into account the traffic conditions for instance. Therefore, it should constitute a good base heuristic to improve on later.

The initial problem is thus a supervised learning task. Indeed, past data provides X and the associated Y. More specifically, it is a classification task as we foresee from X which of the L locations is the decision we eventually take: $Y \in A$. Let us see how data can be encoded before describing the models receiving these inputs.

Data encoding

The input vector X is a mix of a continuous variable (time) with one categorical variable (current location) and two sets of categorical variables (remaining deliveries and remaining pickups). To exploit statistical learning techniques, the input needs to be a vector of real numbers. The data encoding procedure presented in Appendix C gives a way of generating vectors of real numbers. This numerical representation sums up the context and is then used by statistical models.

Statistical Learning Model: Neural Network

The first model we considered is a fully connected Neural Network, with one hidden layer as presented in Fig. 4.2. The input layer has $L + 1$ units (see Appendix C) while the output layer has L units. By applying a softmax function to this final layer, we obtain a probability distribution over all the categories (L locations). In other words, from the Softmax layer, given a certain input, we have the probability that any of the L locations is to be selected as the decision. By sampling or selecting the unit with the highest probability, a decision is finally returned. However, the condition $x \in \mathcal{D} \cup \mathcal{P}$ has not been included yet. That is why a *class eviction* process is implemented. It simply consists of removing the non-valid classes ($x \notin \mathcal{D} \cup \mathcal{P}$) before sampling or taking the maximum to get the decision. This process is implemented only after training the Neural Network.
We explain in more details how we train this Neural Network in the next part in order to get the best training and testing accuracy.

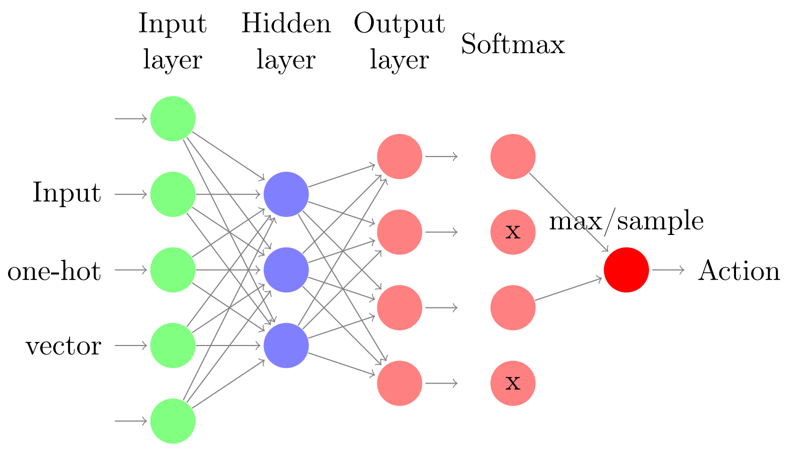


Figure 4.2: Feed Forward Neural Network with class eviction (x units)

More formally, the learned function $f : R^{L+1} \rightarrow R^{L}$, given training samples $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ where $x_i \in \mathbf{R}^{L+1}$ and $y_i \in R^{L}$ (one-hot label vector) is the following:

$$f(x) = \text{softmax}(W_2 g(W_1^T x + b_1) + b_2)$$

where:

- $W_1 \in \mathbf{R}^{L+1} \times \mathbf{R}^m$ (m, number of hidden units)
- $W_2 \in \mathbf{R}^L \times \mathbf{R}^m$
- $\text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_{l=1}^{k} \exp(z_l)}$
- $g(x)_i = max(0, x_i)$, is a rectifier used to decrease the effect of vanishing gradients
- $b_1 \in \mathbf{R}^m$ and $b_2 \in \mathbf{R}^L$

We then determine our parameters $W = \{W_1, W_2, b_1, b_2\}$ by minimizing the categorical cross-entropy function. The cross-entropy function, or loss, is defined as follows for a classification task:

$$Loss(\hat{y}, y, W) = -\sum_i y_i log(\hat{y_i}) + \alpha ||W||_2^2$$

where

- $\hat{y}$: observed samples
- y: labels (from the true underlying distribution)
- $\alpha ||W||_2^2$: regularization parameter to penalize complex models with α a real number

To practically minimize this function, a Stochastic Gradient Descent (SGD) method is used which consists of computing iteratively the loss gradient with respect to the weights W: $\nabla Loss_W$. This gradient is then deducted from the weights according to the following rule:

$$W^{i+1} \leftarrow W^i - \epsilon \nabla Loss_W^i$$

where i is the iteration step and $\epsilon > 0$ the learning rate.

Once the training of f is completed, we can either sample the output distribution probability or select the maximum. These two possibilities will be important for the next section.

4.2.2 Heuristic Enhancement

Once an initial policy is available (either by training it as in the previous section or by a random initialization), it will be used in an exploratory mode (allowing random behavior) in order to produce improved trajectories in terms of rewards. *State-decision* samples of such trajectories are then collected to re-train the initial policy. The overall procedure is presented in Fig. 4.3

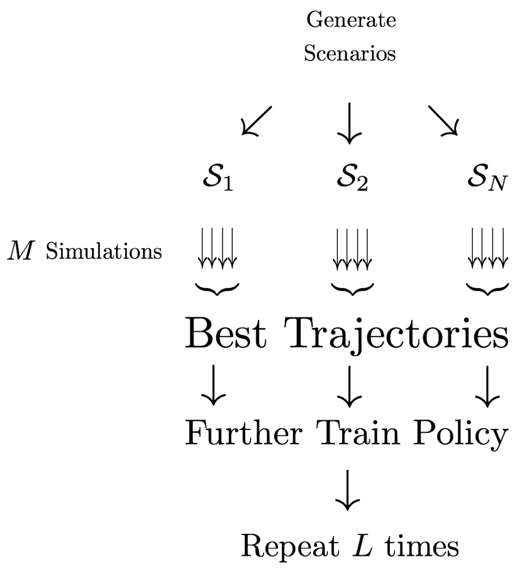


Figure 4.3: Summary of *policyimprovment(*π_0*, L, N, M)*

The policy in exploratory mode is denoted as $\sim \pi_0$ and defined as follows:

$$\sim \pi_0(S_k) = \begin{cases} \tilde{x} & \text{with probability } \epsilon \in [0,1] \\ \pi_0(S_k) & \text{with probability } 1-\epsilon \end{cases}$$

where $\tilde{x}$ is chosen (uniformly) randomly in $\mathcal{D}_k \cup \mathcal{P}_k \setminus \pi_0(S_k)$, the set of possible actions minus the one picked by the policy.

Here is the procedure to perform exploration on a single scenario and retrieve the best path among all the explored paths.

Algorithm 2 *bestTrajFromScenario($\mathcal{S}, M$)*

```
1: Input $\mathcal{S}$, Scenario Object (typically output of generateScenario()),
2:         $M$ integer, number of Scenario simulations
3: Output traj$^\star$, state-decision samples from the best performing trajectory
4: Initialization $i \leftarrow 1$                          ▷ simulation counter
5:             best_score $\leftarrow 0$,
6:             best_traj $\leftarrow$ []           ▷ collect best state-decision samples
7:             $S_k \leftarrow initialState(\mathcal{S})$              ▷ initial state from scenario $S$
8:
9: while $i \leq M$ do
10:     score $\leftarrow 0$                          ▷ store current simulation's score
11:     traj $\leftarrow$ []                     ▷ store current simulation's trajectory
12:
13:     // As long as the deadline is not reached
14:     while $S_k.t_k \leq$12pm do
15:         decision $\leftarrow\sim \pi_0(S_k)$                    ▷ decision with exploration
16:         $S_k \leftarrow generateNextState$(decision, $S_k$,S)
17:         if decision $\in \mathcal{D}_k$ then
18:             score $\leftarrow$ score + $\beta_d$
19:         else
20:             score $\leftarrow$ score + $\beta_p$
21:         traj.append([$S_k$, decision])
22:     // update current highest value trajectory
23:     if score > best_score then
24:         best_score $\leftarrow$ score
25:         best_traj $\leftarrow$ traj
26:     $i \leftarrow i + 1$
27: return traj$^\star$ = best_traj
```

It is now possible to generate new data by repeating the previous procedure on several scenarios and improve the initial policy by training it further at regular intervals. This whole process constitutes a cycle that we also repeat several times as described in Alg. 3.

Algorithm 3 *policyimprovment(π_0, L, N, M)*

```
1:  **Input** $\pi_0$ initial policy
2:          $L$ integer, number of learning cycles
3:          $N$ integer, number of Scenarios
4:          $M$ integer, number of Simulations per Scenario
5:  **Output** $\pi_0$, further trained policy
6:  **Initialization** $l \leftarrow 1$                          ▷ cycle counter
7:              $n \leftarrow 1$                                 ▷ scenario counter
8:
9:  **while** $l < L$ **do**
10:     training_data ← []
11:     // Generate improved state-decision samples
12:     **while** $n < N$ **do**
13:         $\mathcal{S} \leftarrow generateScenario(8am, [], [])$
14:         best_traj = $bestTrajFromScenario(\mathcal{S}, M)$
15:         training_data.append(best_traj)
16:         $n \leftarrow n + 1$
17:
18:     // Further Train the policy
19:     $\pi_0 \leftarrow furtherTrain(\pi_0, training\_data)$
20:     $l \leftarrow l + 1$
21: **return** $\pi_0$
```

4.3 Deployment: Sample Average Approximation (SAA)

In this section, we use the simulator and the scenario generator to obtain an estimate of the downstream value of choosing the available actions at one given state: Sample

Average Approximation.

Algorithm 4 Sample Average Approximation

```
1: Input $S_k$ state
2:         $M$ integer, number of simulations per action
3: Output $x_k$, decision
4: Initialization action_values ← [], store downstream values
5:
6: // for each available action
7: for $a_k$ in $S_k.\mathcal{D}_k \cup S_k.\mathcal{P}_k$ do
8:     // compute next state
9:     $S_{k+1} \leftarrow generateNextState(S_k, a_k)$
10:
11:    downstream_value ← $averageDownstreamValue(\pi_0, S_{k+1}, M)$        ▷ (⋆)
12:    action_value.append(downstream_value)
13:
14: return $\text{argmax}_{a_k \in S_k.\mathcal{D}_k \cup S_k.\mathcal{P}_k}$ [$action\_value$]
```

In Alg. 4, $(\star)$ $averageDownstreamValue(\pi_0, S_{k+1}, M)$ refers to the procedure in Alg. 5:

Algorithm 5 $averageDownstreamValue(\pi_0, S, M_s)$

```
1: Input $\pi_0$ policy
2:          $S$ state
3:          $M_s$ integer, number of scenarios from which to estimate the downstream
   value of being in $S$, following $\pi_0$
4: Output average_score, downstream value of being in a state $S$ and following
   the policy $\pi_0$
5: Initialization scores ← [], contains scores of simulations
6:          $L_{\mathcal{S}}$ ← [], list of scenarios
7:
8: // create $M_s$ scenarios that contain $S_k$
9: for _ in $[1, M_s]$ do
10:    $L_{\mathcal{S}}$.append($generateScenario(S.t, S.\mathcal{D}, S.\mathcal{P})$)
11:
12: // simulate each scenario and retrieve the associated score
13: for $\mathcal{S}$ in $L_{\mathcal{S}}$ do
14:    scores.append($bestScoreFromScenario(\mathcal{S}, 1)$) [1]
15:
16: return $mean$(scores)
```

We present a summary in Fig. 4.4. When the courier is at a state S_k and has to choose among n actions ($a_1, \ldots, a_n$), we simulate many possible paths generated by the interaction between the agent and the policy in exploratory mode. The average value of these paths gives an estimate of the downstream value of an action. Eventually the action with the highest value is chosen (red).

[1] Similar to *bestTrajFromScenario*() but returns the trajectory's score instead of the trajectory's state-decision samples. Only 1 scenario simulation is considered here to alleviate the burden of the computations. This is to achieve fast computations in an operational context. However, integers greater than 1 may be used.

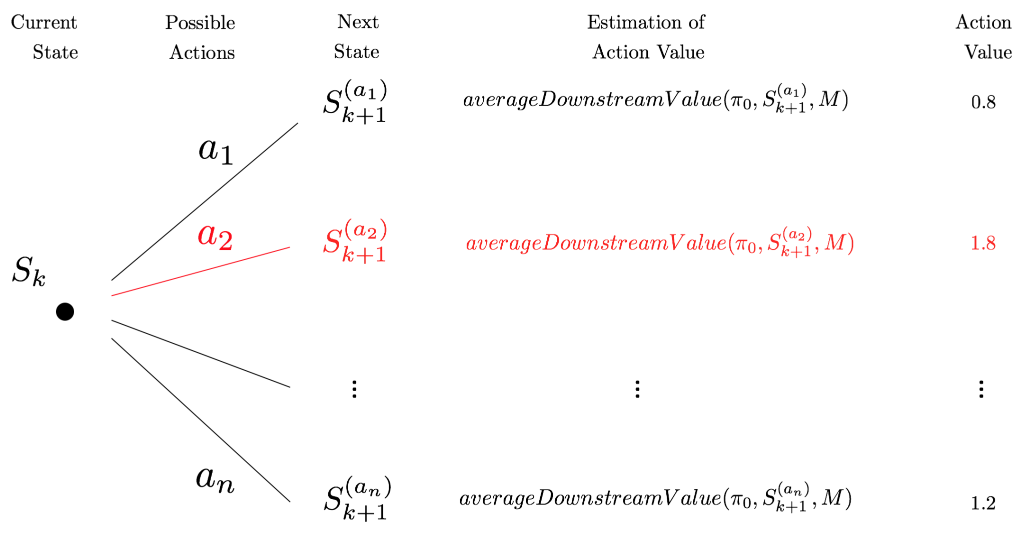

Current State
Possible Actions
Next State
Estimation of Action Value
Action Value
S_k
a_1
a_2
a_n
$S_{k+1}^{(a_1)}$
$averageDownstreamValue(\pi_0, S_{k+1}^{(a_1)}, M)$
0.8
$S_{k+1}^{(a_2)}$
$averageDownstreamValue(\pi_0, S_{k+1}^{(a_2)}, M)$
1.8
$S_{k+1}^{(a_n)}$
$averageDownstreamValue(\pi_0, S_{k+1}^{(a_n)}, M)$
1.2

Figure 4.4: Sample Average Approximation procedure from state S_k

Chapter 5

Computational Experiments and Results

In this chapter, we assess the quality of our approach detailed in the previous chapter. Given the learning and simulation tasks at hand, numerous computational experiments are required in order to estimate the improvements brought by our method and the settings in which it is suitable. Section 5.1 details the software and hardware used throughout the experiments, along with a description of the dataset used in the learning task. Section 5.2 presents the regression models obtained from the dataset. This constitutes the simulator module. In Section 5.3, we select the neural network parameters to obtain a satisfactory base policy. With the simulator and the base policy being properly built, we finally assess our system's performance in Section 5.4.

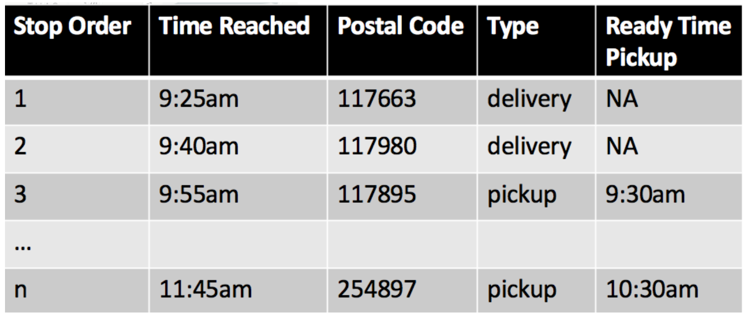
5.1 Generalities

5.1.1 Dataset

Our numerical experiments will be based on a dataset consisting of about 100,000 samples. Each sample represents one stop to service a customer (either delivery or pickup) in Singapore, during one month (December 2015). This represents about 3,000 tours done daily by on average 300 couriers. Close to 6,000 parcels are delivered or retrieved daily. Many features are available such as the time serving the customer, the time when a pickup becomes known to the courier, the truck ID etc. Fig. 5.1 gives an example of the data generated by a single courier on a specific day.

5.1.2 Hardware and Software

The same desktop computer has been used for all the computations. Its processor is an Intel Core i5-6600 running at 3.30GHz with 8GB of RAM. All our system's parts



Stop Order	Time Reached	Postal Code	Type	Ready Time Pickup
1	9:25am	117663	delivery	NA
2	9:40am	117980	delivery	NA
3	9:55am	117895	pickup	9:30am
…				
n	11:45am	254897	pickup	10:30am

Figure 5.1: Example of daily data for a single courier

have been built using Python version 3.6. For speed and memory saving purpose, the Numpy library has been used extensively to represent vectors, matrices and the operations between these objects. For the base policy initialization from historical decisions, the library Tensorflow has been used to exploit its fast underlying C++ implementation. No GPU has been used and this remains a possible extension of this work for a faster implementation. The scikit-learn library has also been used for its easy-to-use API and its large variety of statistical models, as it is useful to identify which models to use in our system. Most of the codes used for the experiments are available on Github[1].

5.2 Simulator Building

In this Section, we detail each module of the simulator presented in Section 4.1: traveling times and scenario generator, both obtained from real-world data. We use two types of regressions: second order polynomial regression and kernel density estimation.

5.2.1 Traveling Time estimations

From 279 trips chosen uniformly randomly among past deliveries, we retrieve travel times as a function of distance, at several periods of the day. This allows the definition of a travel time probability distribution with the parameters μ_k and σ_k, as

[1]Simulator and Statistical Learning: https://github.com/doulouUS/policy_approximation. Travel times and Pickup Probability Distribution https://github.com/doulouUS/foo-Environment_2 in the *dynamics* folder

Table 5.1: Regression Parameters at various times of the day

Date	**Time**	c	b	a	$\tau^*_{n_{trip}-2}$	s_x	n_{trip}	$\bar{d}$	s_d
20150314	0928	61.80258495	119.39268608	-1.97871606	1.65041343	80.21387969	279	2.06638489	3962.17476982
20150314	1000	57.85077531	121.79934186	-2.09783781	1.65039322	77.82829678	279	2.07179211	3964.44253194
20150314	1031	57.58979393	119.66229242	-1.87988113	1.65039322	79.17258823	279	2.07179211	3964.44253194
20150314	1103	60.35024020	117.80647206	-1.98264065	1.65039322	79.51625952	279	2.07179211	3964.44253194
20150314	1135	61.20563238	117.91676701	-2.06737683	1.65039322	77.22963306	279	2.07179211	3964.44253194
20150314	1207	64.94283256	116.54402851	-2.00949129	1.65039322	80.23824689	279	2.07179211	3964.44253194
20150314	1239	58.10411789	120.16016664	-2.16122218	1.65039322	75.07799628	279	2.07179211	3964.44253194
20150314	1311	60.30947380	116.78939684	-2.04804424	1.65039322	75.07351021	279	2.07178853	3964.44471852
20150314	1343	61.26578258	116.91450533	-2.04944668	1.65039322	75.21139458	279	2.07179211	3964.44253194
20150314	1414	56.82970495	121.82378078	-2.19145419	1.65039322	75.15014007	279	2.07179211	3964.44253194
20150314	1446	61.19829030	118.26482388	-2.07166564	1.65039322	77.13305296	279	2.07179211	3964.44253194
20150314	1518	58.99825580	121.37056356	-2.22827180	1.65039322	76.54692802	279	2.07179211	3964.44253194

explained in Section 4.1.1

It is to be noticed that the coefficient of determination (known as R^2) is very good: at least 0.93 whatever the period of measurement is. Fig. 5.2 shows all the measurements done throughout one day.

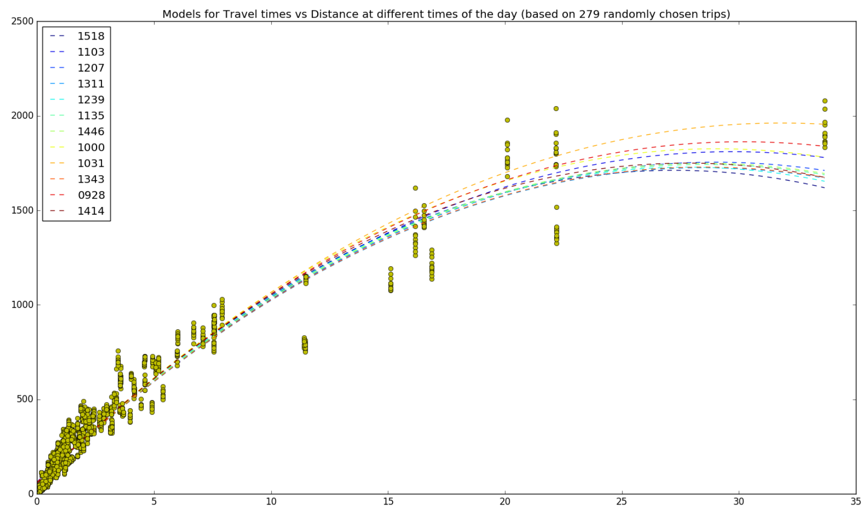


Figure 5.2: All regression models obtained at various times of the day.

5.2.2 Scenario generator

Pickup ready-time distributions $\mathcal{R}_i$

Fig. 5.3 presents diagrams obtained from the dataset's column giving the time at which pickups become known.

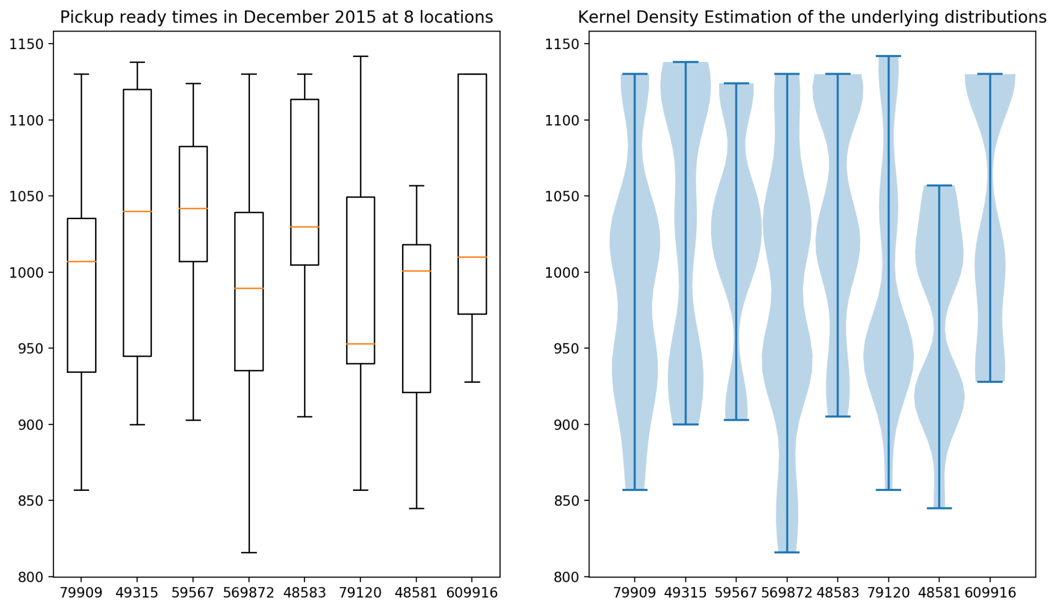


Figure 5.3: Boxplot and Violin plot of pickup appearance times in 8 locations (postal codes)

Kernel Density methods (as explained in Section 4.1.2) and boxplots give a good idea of how the data is distributed over time. It is interesting to see that the data has a relatively high variance. However, for some locations, a pattern appears with pickups having a tendency to appear regularly around the same time (e.g. 79120, where pickups tend to appear early in the morning). For the needs of our study, simple normal distributions are adopted. Variances of these distributions remain an important parameter and are set in Section 5.4.

Kernel Density Estimation

In Figs. 5.4 and 5.5, we present the pickup probability distributions obtained by doing a Kernel Density Estimation as described in Section 4.1.2.

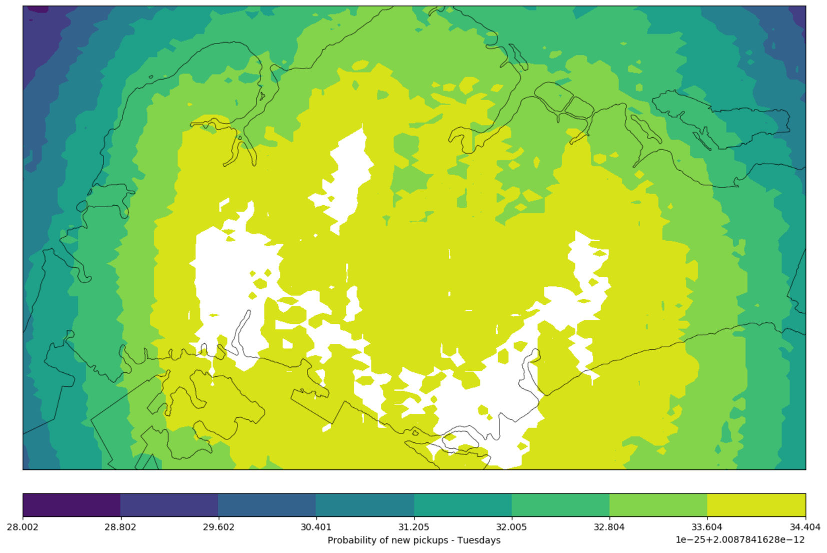


Figure 5.4: Pickup Probability Distributions based on Tuesdays data - $h = 0.1$

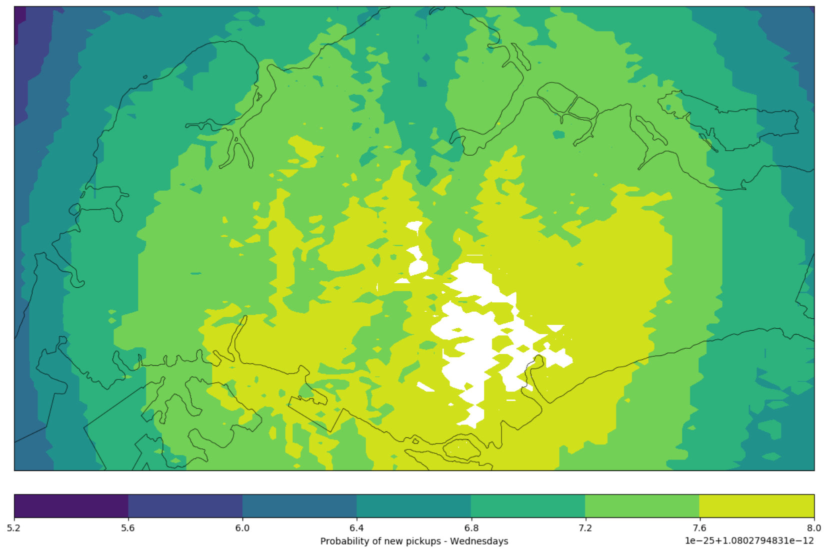


Figure 5.5: Pickup Probability Distributions based on Wednesdays data - $h = 0.1$

Different trends between Tuesdays and Wednesdays appear. It is also important to mention the special role played by the bandwidth h. It acts as a smoothing parameter, which sets the balance between bias and variance in the obtained distributions. Here, the bandwidth has been chosen relatively small, in order to capture local trends.

5.2.3 Simulator Implementation

The simulator uses the Object-Oriented Programming paradigm to fit our need for modularity. As shown in Appendix D, our program is made of modules interacting together with a possibility to modify each one independently from the others.

5.2.4 Discussion

In this Section, we built second order polynomial models to model travel times at various times of the day. These models are surprisingly very accurate ($R^2 = 0.93$). However, our modular implementation allows for future model refinement.

The geographical distributions $\mathcal{G}_p$ and $\mathcal{G}_d$ obtained from a Kernel Distribution Estimation give a broad idea of demand patterns over Singapore. However, our system is operating at smaller levels: more samples are required to model trends in residential neighborhoods. Again, our modular implementation allows for future improvements on these models, independent of the other modules.

5.3 Heuristic Training: Supervised Learning

This Section focuses on setting the right parameters of the neural network described in Section 4.2.1.

5.3.1 Neural Network Training

The Neural Network structure described in Section 4.2.1 has a large number of parameters to be chosen. They can be divided among the following four categories:

- Batch or online learning
- Gradient Descent: we implemented an Adam optimizer, as it is a well known technique to speed up learning
- Code implementation: trying to exploit the sparse structure of our inputs
- Architecture: number of layers, number of hidden units for each layer

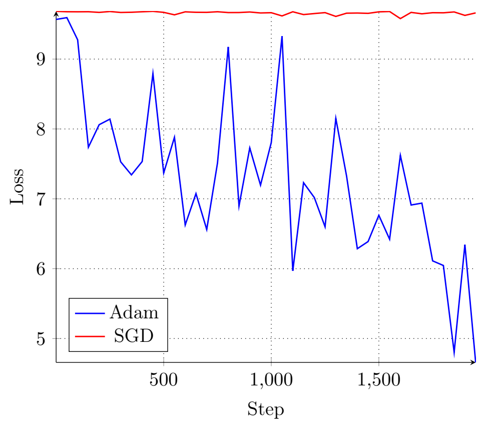
Several architectures have been implemented using the Python library Tensorflow as shown in Appendix A.

Optimizer, Sparsity and learning mode choices

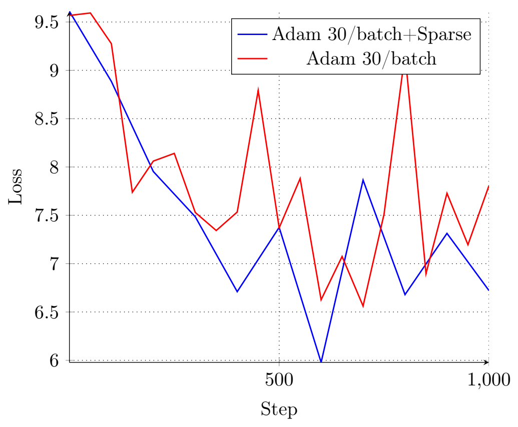
We show the cross-entropy losses presented in Section 4.2.1, obtained for various implementation choices in Fig. 5.6.

Using the Adam optimizer presented in (Kingma & Ba, 2014), a sparse implementation as well as 30 samples per batch for retraining are the choices we retain for our system. These are the parameters showing the best performance on the training set.

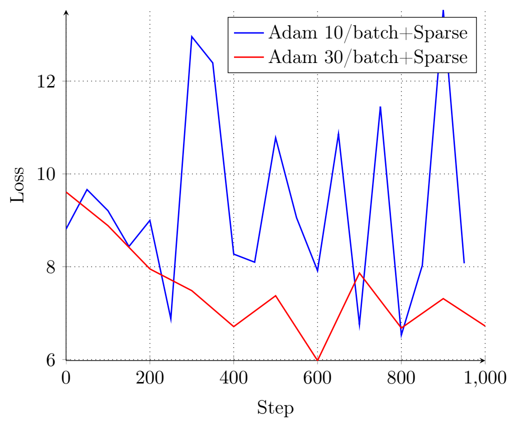
Figure 5.6: Parameters tuning



a: Adam vs SGD



b: Plain vs Sparse



c: 10 and 30 per batch

Architecture choices

We present in Table 5.2, some training results using an Adam optimizer, a sparse implementation and 30 samples per batch, considering all the locations from our dataset: 16,077.

Table 5.2: Experiment Results - 20k steps

Input size L	Hidden Units	Batch size	Training Accuracy	Testing Accuracy
16,077	1,000	30	90%	15%

The results are poor, which is explained by the fact that the training data is too broad and does not exhibit a pattern. Instead, we reduced the dataset to tours done by a single truck with the following characteristics:

- 1 month of daily operations
- 37 different postal codes for 957 stops (delivery or pickup)

This new dataset leads to the results shown in Table 5.3, with losses given in Fig. 5.7.

Table 5.3: Experiment Results - 60k steps

Input size L	Hidden Units	Batch size	Training Accuracy	Testing Accuracy
37	1,000	30	98%	40%

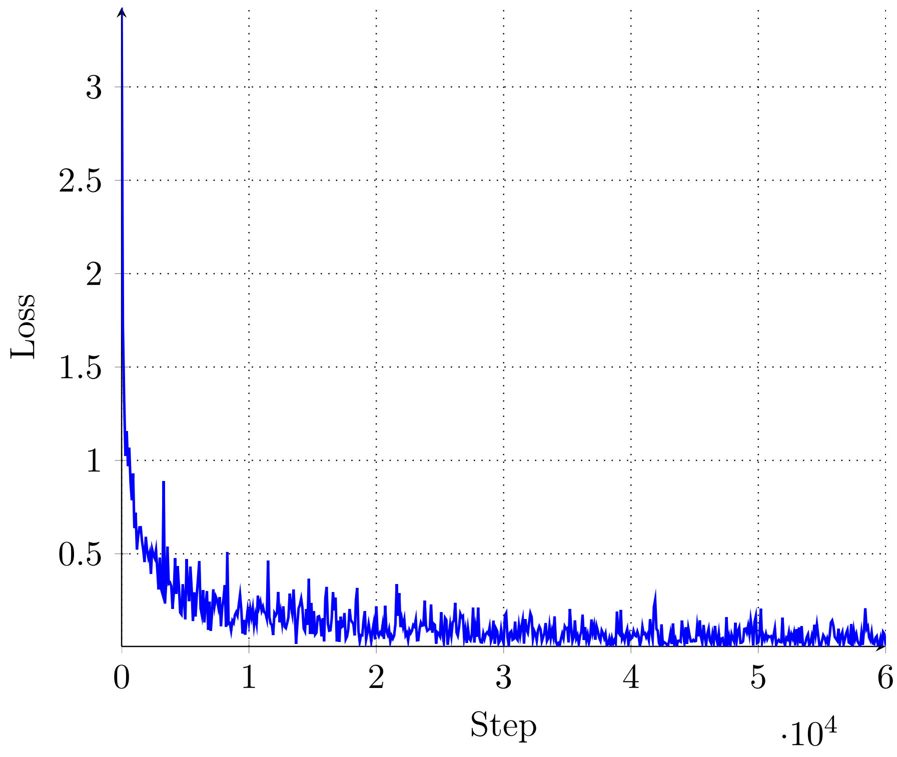


Figure 5.7: Adam optimizer - 30 samples per batch - 60k steps

5.3.2 Discussion

The training set is well learnt but performance remains poor on the test set despite using techniques such as dropout or early stopping to prevent overfitting.

These results are explained by some reasons. Firstly, the reduced dataset is too small (barely 1000 samples). Data from at least one year of operations from the same driver would be needed. Secondly, the way of measuring performance (accuracy) is too restrictive. Indeed, only the highest probable neuron (see Section 4.2.1) is returned from the last network's layer and compared to the corresponding label. It is thus not surprising to see such poor results. However, when considering that an output is correct when the label is among the three most likely returned neurons, the accuracy reaches 70%. At these levels of performance, the policy learnt by the neural network is satisfactory and can constitute a base policy.

It is important to finally mention that this part of our system can be improved using the latest developments in Deep Learning. Firstly, the way data is encoded (see Appendix C) is simple and can certainly be improved to fit well into the chosen neural network's architecture. Secondly, the complexity of our model is relatively low (two layers). Some network architectures can comprise hundreds of layers using GPU as well as distributed systems to train massive models.

5.4 Enhanced Policy Performance and Comparison with a myopic policy

In this Section, we describe the improvement procedure of the base policy obtained from the previous Section. In some particular settings, Alg. 3 is implemented. Then, the parameters N, M and L are set empirically. We finally identify in which demand settings our procedure performs better compared to the Nearest-Neighbor heuristic.

5.4.1 Experiment Settings

Our procedure is tested in a simple grid world (here $[1, 10] \times [1, 10] \subset N \times N$ [2]) where each node of the grid is a location (designated by a pair of integers) and the Manhattan distance is used to establish the distance between the nodes. The delivery van always starts from the top left corner, with coordinates $(0, 0)$. Travel times are obtained by imitating patterns seen in the real world (see Appendix E) using a scale factor. A fixed set of deliveries is chosen and remains the same throughout the simulations. $\mathcal{G}_p$ and the $\mathcal{R}_i$ distributions are initialized as shown in Fig. 5.8 and Fig. 5.9 where the red color indicates a high probability and blue color a low probability.

[2] N represents the set of all natural integers

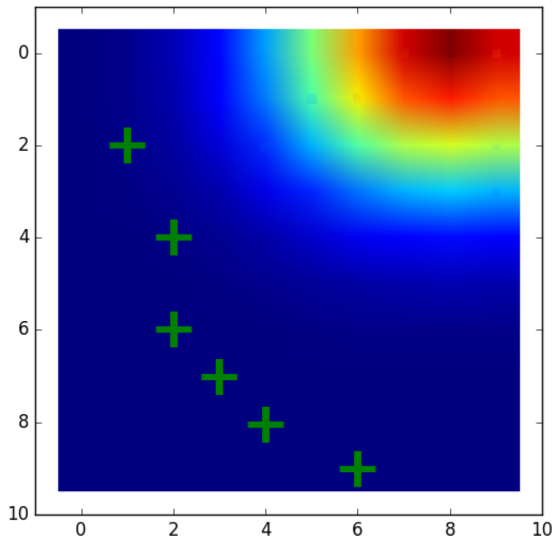


Figure 5.8: $\mathcal{G}_p$ with deliveries (green crosses)

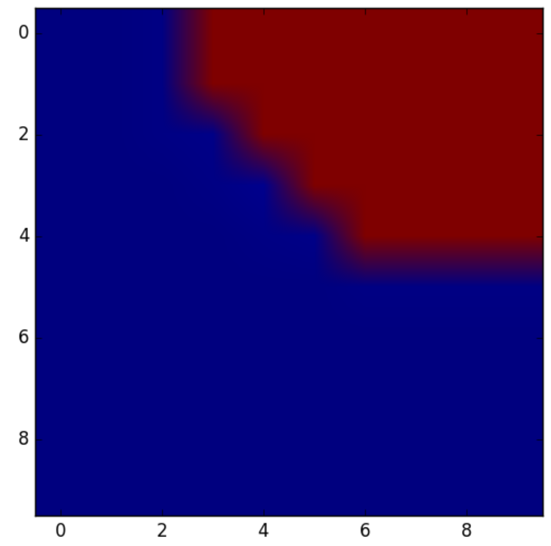


Figure 5.9: Mean of $\mathcal{R}_i$, $i \in [1, 10] \times [1, 10]$

Deliveries are deliberately placed in regions where the probability for pickups to appear is low. However, pickups are extremely likely at the top right corner and a larger number of them is set to appear. It is, therefore, more rewarding to make a move towards this region in the long term. A nearest-neighbor policy would tackle the deliveries first (see Section 5.4.4). In these simple settings, we test the ability of our system to anticipate choices leading to more rewards.

To follow the performance of the system during the experiment, 10 test scenarios are initialized with the same number of deliveries and pickups. Only the pickups' locations and times of appearance vary. These scenarios are regularly simulated with the policy currently being trained. At the end of each simulation, each test gives the number of customers (deliveries or pickups) the policy succeeded to serve. The average number of served jobs on all the test scenarios is the metric used to assess how well the policy performs.

Our procedure will be referred as *System* and the myopic policy will be the *Nearest-Neighbor* policy.

5.4.2 Computation time

We present in this Section, the improvements in computation time brought by a parallel implementation.

Alg. 2 is mainly a loop of independent processes. Indeed, each pass (M in total) into the loop consists of simulating from the start the current policy on a scenario. It is therefore highly parallelizable and Fig. 5.10 shows the significant gains obtained for high values of M.

This result is not surprising: Alg. 2 consists of two nested loops repeated L times. Parallelizing one of these loops speeds up each pass, but does not remove the linear dependence in L. However, parallel computations requires more memory writings which slow down the processes for low values of M. Given the duration involved, the model can easily be trained at night.

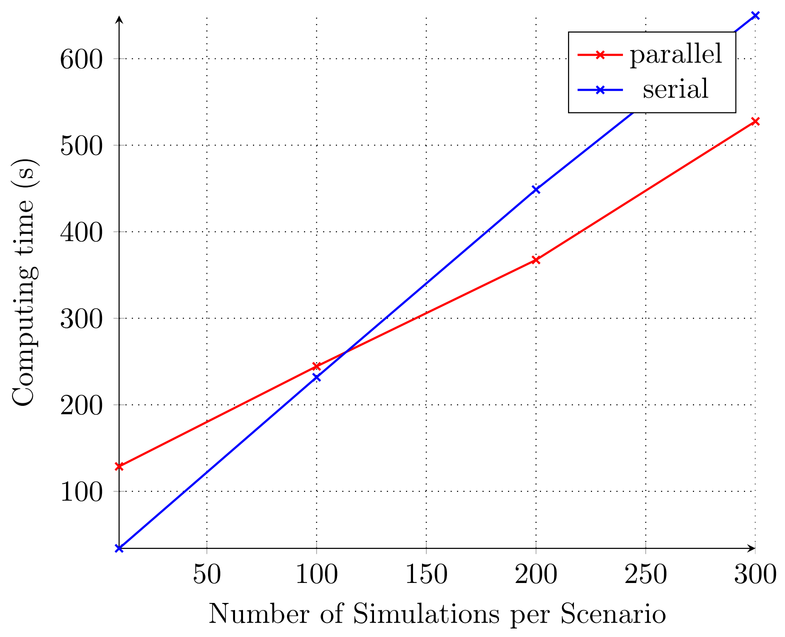


Figure 5.10: $N = 10$ for $L = 100$ steps

5.4.3 Model parameters tuning

In this part, we select the model's parameters leading to the best performance, independent of the environment's patterns. Heuristic parameters are also considered fixed because they have been obtained in the previous Section (Section 5.3).

Number of Scenarios N

Intuitively, N gives an indication of the variety of scenarios encountered during learning by the system. The higher the number of scenarios, the more situations have been explored, which leads us to think that a large N is necessary. However, the results presented in Fig. 5.11 show differently.

Performance increases fast during the first steps but then stops for higher values of N. Only $N = 1$ shows a sustained increase in performance. This problem is well-known in Machine Learning. High values of N mean that each retraining period has a lot of samples at once (batch training). When $N = 1$, only one sample is used at each retraining period: it is an online training. In our case, batch learning (especially for high values of N) leads to over-fitting in our model which explains

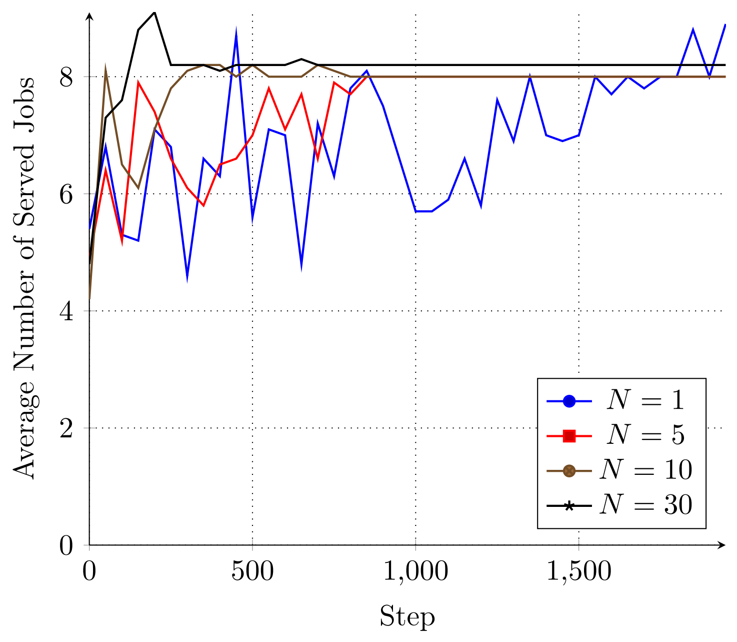


Figure 5.11: System performance during learning for different values of N

why performance stalls. Therefore, N should be kept low to allow for longer learning phases.

Number of Parallel Simulations M

M is an indication of the degree of exploration carried out during the simulation of one scenario. Intuition suggests that M should be high to allow for sufficient exploration, leading to high-quality decisions. Experiments confirm this statement. The higher M is, the better the performance is. With this remark, parallel computations are all the more needed.

5.4.4 Comparison with Nearest-Neighbor

In this Section, we finally examine the performance of our system in various environment conditions. The results are compared with the Nearest-Neighbor policy, which consists of going to the nearest available customer at each step.

Influence of duration factor

The duration factor, presented in more details in Appendix E, is a parameter representing the inverse of the travel time between the two most distant locations of the environment. Fig. 5.12 presents the performance of our system compared to the Nearest-Neighbor policy. Only the duration factor varies, while the other parameters are the ones obtained from the previous Section ($N = 1$ and $M = 1000$) with $L = 1,000$.

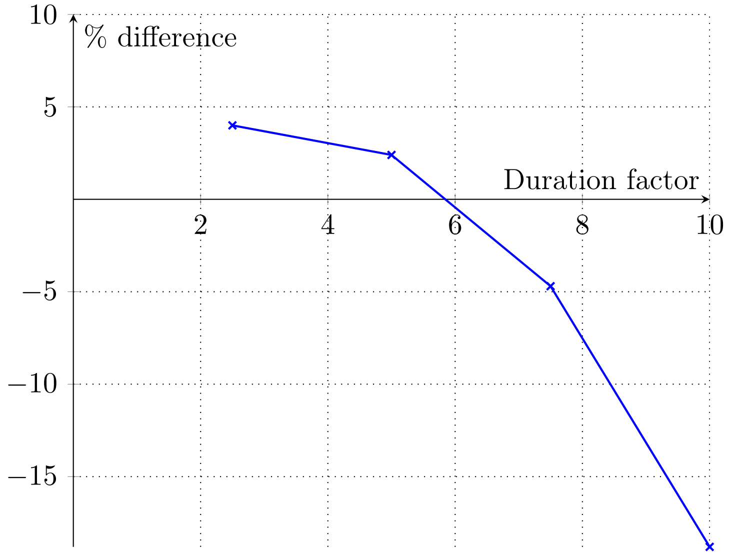


Figure 5.12: Duration factor influence on the Percentage difference between Nearest-Neighbor and System scores $(\frac{Score_{Nearest-Neighbor} - Score_{System}}{Score_{Nearest-Neighbor}})$

Our procedure outperforms the Nearest-Neighbor policy in congested settings (duration factor lower than 6) by up to 4.0%. For fluid traffic, the Nearest-Neighbor policy performs better by up to 18.8%.

Influence of the standard deviation of times of appearance

The standard deviation of pickups' times of appearance controls how new pickups are spread in time. Fig. 5.13 shows the influence of this parameter, while $N = 1$, $M = 1$ and the duration factor is set to 5.

Our system outperforms the Nearest-Neighbor policy for larger standard deviations of times of appearance (standard deviation larger than ~0.25) by up to 2.7%.

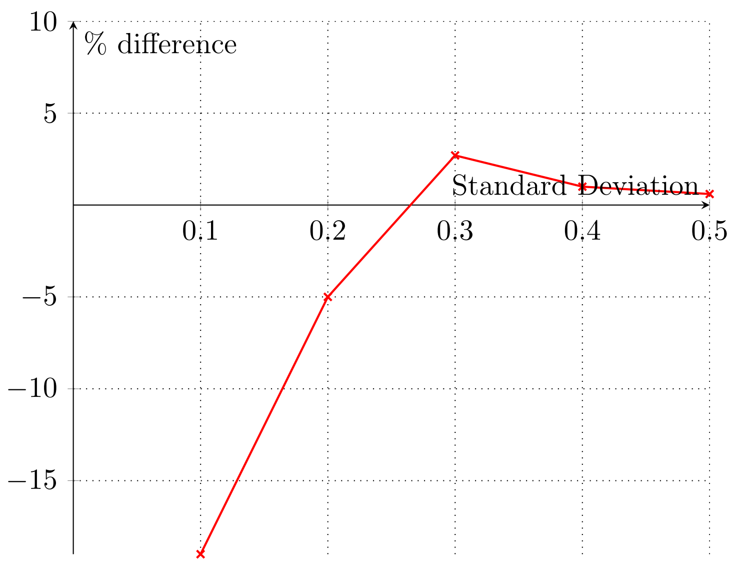


Figure 5.13: Standard Deviation Influence: Percentage difference between Nearest-Neighbor and System scores

For lower standard deviations, the Nearest-Neighbor policy performs better by up to 19%.

Influence of L

We run $L = 50,000$ cycles to evaluate the benefits of a longer training period. Fig. 5.14 presents the Average Number of Jobs served on 10 test scenarios, evaluated every 50 cycles during the training period of our system.

At the end of training, our system outperforms the Nearest-Neighbor policy by 4.4% in terms of the average number of jobs served (12.40 versus 11.87). 80% of the test scenarios saw improvements compared to the Nearest-Neighbor policy. Among these scenarios, our system served on average 1 to 2 more customers for which the Nearest-Neighbor policy served 13 to 14 customers, representing an increase of more than 10%.

5.5 Summary

This chapter described implementations of the different modules built in the previous chapter. Firstly, an environment is simulated to reproduce realistic travel times

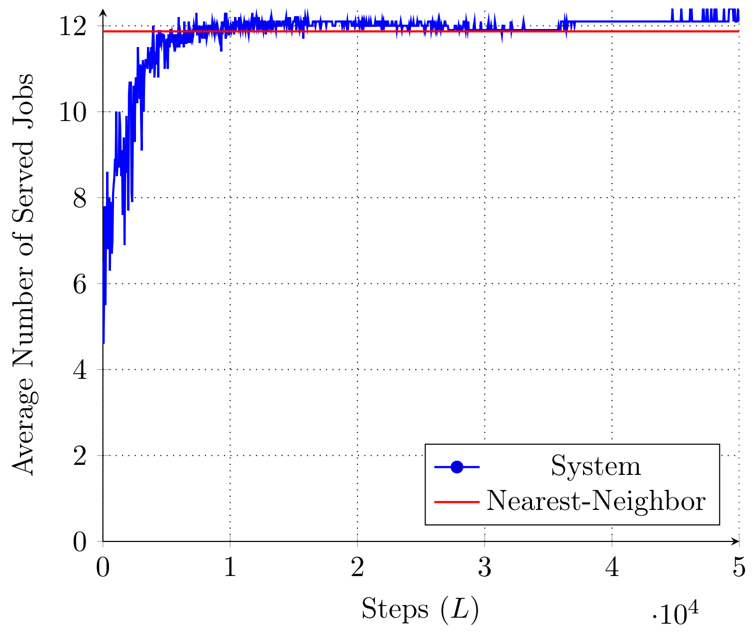


Figure 5.14: System performance during learning period

and demand scenarios. Secondly, a base policy is learnt from historical decisions made by experienced drivers. This policy takes as input the state of the environment and outputs the decision to take. Eventually, the policy and the environment interact to produce trajectories. By allowing exploration during the simulation of these trajectories as well as retraining of the policy, we improved the base policy. The improved policy performs better than the Nearest-Neighbor policy by 4.4% on average, on particular conditions. Indeed, congested road networks and relatively high standard deviation for pickups' times of appearance lead to better performance from our system.

Chapter 6

Conclusion and future research

This thesis brought a new perspective to the treatment of last-mile delivery problems with uncertain demands. Specifically, it shows that Deep Reinforcement Learning methods from the Artificial Intelligence community can be used to build suboptimal policies. The profusion of data generated by couriers is an opportunity to incorporate their knowledge in statistical models. We succeeded to reproduce decisions made by drivers in up to 70% of the test cases, using a moderately complex neural network.

A policy represented as a statistical model can be further improved by simulating its actions in world-like scenarios. Simulations allow the policy to explore its environment in order to improve decisions according to a chosen objective function. These improved decisions are then used to further train the statistical model. We succeeded to improve by 4.4% the average number of jobs served by the nearest-neighbor policy with improvements reaching more than 10% on specific scenarios.

Beyond these results, this work should be seen as an introduction towards a more complex use of Deep Reinforcement Learning techniques in DVRP. Significant work and improvements remain to be done and we give some directions hereafter.

Firstly, we believe it would be interesting to explore state encodings representing the environment. An almost infinite amount of information can be included in addition to the state of demand, current location and current time as we did in this study. This aspect is intimately related to the architecture of the neural network representing the agent. Recent works in the RL community use architectures with dozens of layers which take advantage of complex but well designed state encodings.

Secondly, an emphasis should be made on the practical implementation of the framework. To allow for the use of complex models in an industrial setting, it would be interesting to exploit Graphics Processing Units (GPUs) or distributed computing implementations. Our approach is well suited for highly parallel implementations.

This could bring the strength of recent RL techniques to a fast-growing industry, that needs to improve its efficiency.

References

Adding value to parcel delivery. (2015). Retrieved from https://www.accenture.com/_acnmedia/Accenture/Conversion-Assets/DotCom/Documents/Global/PDF/Dualpub_23/Accenture-Adding-Value-to-Parcel-Delivery.pdf

Attanasio, A., Bregman, J., Ghiani, G., & Manni, E. (2007). Real-time fleet management at Ecourier Ltd. *Operations Research/Computer Science Interfaces Series*, *38*, 219-238.

Bellman, R. (1954). Dynamic programming and a new formalism in the calculus of variations. *Proceedings of the National Academy of Sciences*, *40*(4), 231-235.

Bent, R. W., & Van Hentenryck, P. (2004). Scenario-based planning for partially dynamic vehicle routing with stochastic customers. *Operations Research*, *52*(6), 977-987.

Bertsekas, D. P. (2007). *Dynamic programming and optimal control* (3rd ed., Vol. II). Athena Scientific.

Bertsekas, D. P., & Castanon, D. A. (1989). Adaptive aggregation methods for infinite horizon dynamic programming. *IEEE Transactions on Automatic Control*, *34*(6), 589-598.

Bieding, T., Gortz, S., & Klose, A. (2009). On line routing per mobile phone a case on subsequent deliveries of newspapers. *Lecture Notes in Economics and Mathematical Systems*, 29-51.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). OpenAI Gym. *CoRR (Computing Research Repository)*, *abs/1606.01540*. Retrieved from http://arxiv.org/abs/1606.01540

Cao, Z., Guo, H., Zhang, J., Niyato, D., & Fastenrath, U. (2016). Improving the efficiency of stochastic vehicle routing: A partial Lagrange Multiplier Method. *IEEE Transactions on Vehicular Technology*, *65*(6), 3993-4005.

Chan, Y., & Baker, S. (2005). The multiple depot, multiple traveling salesmen facility-location problem: Vehicle range, service frequency, and heuristic implementations. *Mathematical and Computer Modelling*, *41*(8-9), 1035-1053.

Chang, T.-S., Wan, Y.-W., & OOI, W. T. (2009). A stochastic dynamic traveling salesman problem with hard time windows. *European Journal of Operational Research*, *198*(3), 748-759.

Cheong, T., & White, C. C. (2012). Dynamic traveling salesman problem: Value of real-time traffic information. *IEEE Transactions on Intelligent Transportation Systems*, *13*(2), 619-630.

Christiansen, C. H., & Lysgaard, J. (2007). A branch-and-price algorithm for the capacitated vehicle routing problem with stochastic demands. *Operations Research Letters*, *35*(6), 773-781.

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.

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

Ferrucci, F., & Bock, S. (2015). A general approach for controlling vehicle en-route diversions in dynamic vehicle routing problems. *Transportation Research Part B: Methodological*, *77*, 76-87.

Fisher, M. L. (1994). Optimal solution of vehicle routing problems using minimum k-trees. *Operations Research*, *42*(4), 626-642.

Goodson, J. C., Ohlmann, J. W., & Thomas, B. W. (2013). Rollout policies for dynamic solutions to the multivehicle routing problem with stochastic demand and duration limits. *Operations Research*, *61*(1), 138-154.

Grippa, P. (2016). Decision making in a UAV-based delivery system with impatient customers. *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Horvitz, E., Apacible, J., Sarin, R., & Liao, L. (2012). Prediction, expectation, and surprise: Methods, designs, and study of a deployed traffic forecasting service. *CoRR (Computing Research Repository)*, *abs/1207.1352*. Retrieved from http://arxiv.org/abs/1207.1352

Horvitz, E., & Mitchell, T. (2017). *From data to knowledge to action: A global enabler for the 21st century.* Retrieved from https://www.microsoft.com/en-us/research/publication/from-data-to-knowledge-to-action-a-global-enabler-for-the-21st-century/

Ichoua, S., Gendreau, M., & Potvin, J.-Y. (2006). Exploiting knowledge about future demands for real-time vehicle dispatching. *Transportation Science*, *40*(2), 211-225.

Jean, S., Cho, K., Memisevic, R., & Bengio, Y. (2014). On using very large target vocabulary for neural machine translation. *CoRR*, *abs/1412.2007*. Retrieved from http://arxiv.org/abs/1412.2007

Joerss, M., Schroder, J., Neuhaus, F., Klink, C., & F., M. (2016, September). Parcel delivery the future of last mile. *Travel, Transport and Logistics. McKinsey-Company*.

Juliani, A. (2017). *Simple Reinforcement Learning with Tensorflow.* Retrieved from https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-0-q-learning-with-tables-and-neural-networks-d195264329d0

Karp, R. (1972). Reducibility among combinatorial problems. *Complexity of Computer Computations*, 85–103.

Kim, G., Ong, Y. S., Cheong, T., & Tan, P. S. (2016). Solving the dynamic vehicle routing problem under traffic congestion. *IEEE Transactions on Intelligent Transportation Systems*, *17*(8), 2367-2380.

Kim, S., Lewis, M., & White, C. (2005a). Optimal vehicle routing with real-time traffic information. *IEEE Transactions on Intelligent Transportation Systems*, *6*(2), 178-188.

Kim, S., Lewis, M., & White, C. (2005b). State space reduction for nonstationary stochastic shortest path problems with real-time traffic information. *IEEE Transactions on Intelligent Transportation Systems*, *6*(3), 273-284.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, *abs/1412.6980*. Retrieved from http://arxiv.org/abs/1412.6980

Maddison, C. J., Huang, A., Sutskever, I., & Silver, D. (2014). Move evaluation in go using deep convolutional neural networks. *CoRR*, *abs/1412.6564*. Retrieved from http://arxiv.org/abs/1412.6564

Manna, C., & Prestwich, S. (2014). Online stochastic planning for taxi and ridesharing. *2014 IEEE 26th International Conference on Tools with Artificial Intelligence*.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. A. (2013). Playing Atari with Deep Reinforcement Learning. *CoRR*, *abs/1312.5602*. Retrieved from http://arxiv.org/abs/1312.5602

Nguyen, A., Yosinski, J., & Clune, J. (2016). Understanding innovation engines: Automated creativity and improved stochastic optimization via deep learning. *Evolutionary Computation*, *24*(3), 545-572.

Novoa, C., & Storer, R. (2009). An approximate dynamic programming approach for the vehicle routing problem with stochastic demands. *European Journal of Operational Research*, *196*(2), 509-515.

Pattberg, B. (2017). *Sumo simulation of urban mobility.* Retrieved from http://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/

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

Pillac, V., Guèret, C., & Medaglia, A. L. (2012). An event-driven optimization framework for dynamic vehicle routing. *Decision Support Systems*, *54*(1), 414-423.

Powell, W. B. (2009). What you should know about approximate dynamic programming. *Naval Research Logistics*, *56*(3), 239-249.

Powell, W. B. (2013). *Approximate dynamic programming: Solving the curses of dimensionality.* Wiley.

Proper, S., & Tadepalli, P. (2006). Scaling model-based average-reward reinforcement learning for product delivery. *Lecture Notes in Computer Science*, 735-742.

Secomandi, N. (2000). Comparing neuro-dynamic programming algorithms for the vehicle routing problem with stochastic demands. *Computers & Operations Research*, *27*(11-12), 1201-1225.

Secomandi, N., & Margot, F. (2009). Reoptimization approaches for the vehicle-routing problem with stochastic demands. *Operations Research*, *57*(1), 214-230.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., ... Lanctot, M. e. a. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, *529*(7587), 484-489.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., ... Hassabis, D. (2017, December). Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *ArXiv e-prints*.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... Bolton, A. e. a. (2017). Mastering the game of go without human knowledge. *Nature*, *550*(7676), 354-359.

Simão, H. P., Day, J., George, A. P., Gifford, T., Nienow, J., & Powell, W. B. (2009). An approximate dynamic programming algorithm for large-scale fleet management: A case application. *Transportation Science*, *43*(2), 178-197.

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

Sungur, I., Ren, Y., Ordóñez, F., Dessouky, M., & Zhong, H. (2010). A model and algorithm for the courier delivery problem with uncertainty. *Transportation Science*, *44*(2), 193-205.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction* (Vol. 1) (No. 1). MIT Press Cambridge.

Szepesvári, C. (2010). Algorithms for reinforcement learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, *4*(1), 1-103.

Tadepalli, P., & Ok, D. (1998). Model-based average reward reinforcement learning. *Artificial Intelligence*, *100*(1-2), 177-224.

Tensorboard: Graph visualization. (2017). Retrieved from https://www.tensorflow

.org/versions/r0.12/how_tos/graph_viz/

Tomis, R., Rapant, L., Martinovic, J., Slaninová, K., & Vondrák, I. (2016). Probabilistic Time-Dependent Travel Time Computation Using Monte Carlo Simulation. *Lecture Notes in Computer Science*, 161-170.

Van Hentenryck, P., Bent, R., & Upfal, E. (2009). Online stochastic optimization under time constraints. *Annals of Operations Research*, *177*(1), 151-183.

Woodard, D., Nogin, G., Koch, P., Racz, D., Goldszmidt, M., & Horvitz, E. (2017). Predicting travel time reliability using mobile phone GPS data. *Transportation Research Part C: Emerging Technologies*, *75*, 30-44.

Yuyan, M., Jiafu, T., & Yang, Y. (2013). A management platform for testing benchmarking and algorithm in weighted vehicle routing problems. In *2013 25th Chinese Control and Decision Conference (CCDC).*

Appendices

Appendix A.

Neural Network Architectures

We present two architectures tested in Section 5.3.1. (*TensorBoard: Graph Visualization*, 2017) explains how to read these flow charts.

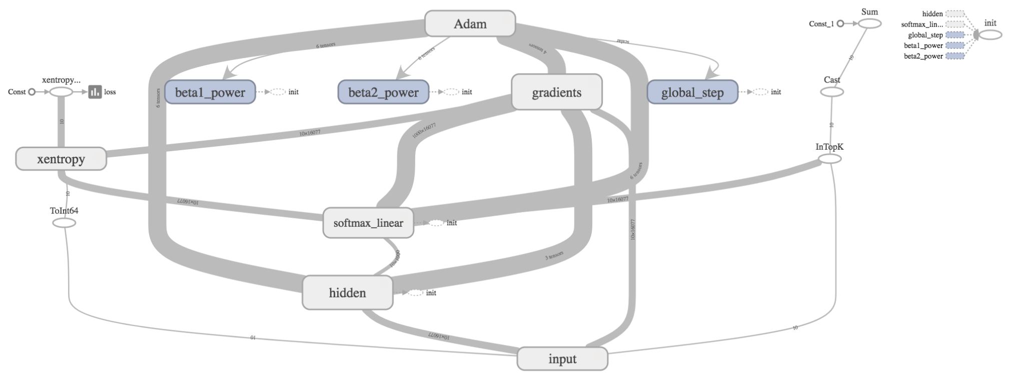


Figure A.1: Adam Optimizer - 30 samples per batch - Large Experiment

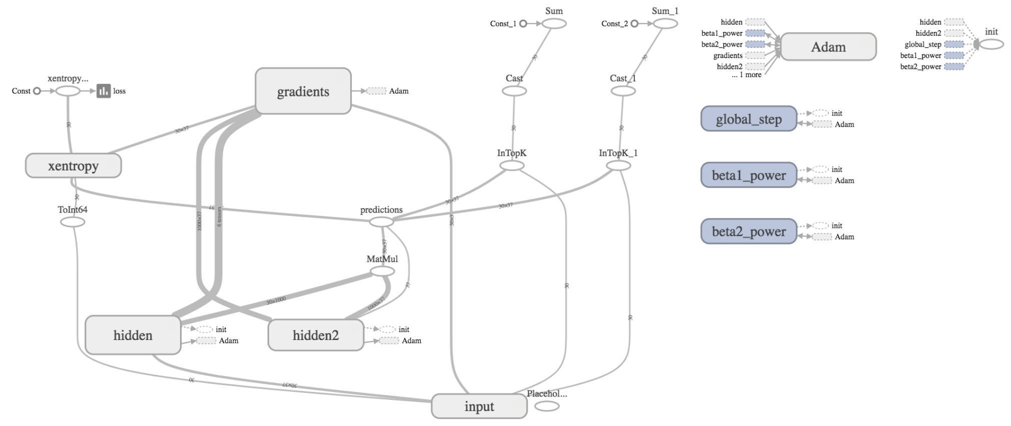


Figure A.2: Adam Optimizer - 30 samples per batch - Small Experiment

Appendix B.

Scenario Representation

Scenarios are described using a nested map data structure as shown in Fig. B.1.

$$\begin{array}{lll}
\{\text{"delivery"} & : & \\
 & & \{location1 : nb1, \\
 & & location2 : nb2, \\
 & & \qquad \ldots \qquad\qquad \}, \\
\text{"pickup"} & : & \\
 & & \{locationA : [readyTime1, \ldots], \\
 & & locationB : [readyTime1, \ldots], \\
 & & \qquad \ldots \qquad\qquad \}\}
\end{array}$$

Figure B.1: Scenario Object: Nested map data structure

Appendix C.

Data encoding: one-hot vectors

We adopt the one-hot-encoding technique traditionally used in Machine Learning for categorical variables and extend it to fit our needs. The one-hot-encoding technique sets up a vector having the same size as the number of categories, and sets the index, say i to the category which has been indexed as i. For instance, in a vocabulary of 5 words, one word said to be the first will be encoded as $(1, 0, 0, 0, 0)$. The second one will be encoded as $(0, 1, 0, 0, 0)$ and so on.

In our case, categories have a different status: it can be a current location, a remaining delivery or a remaining pickup. We extend the technique described in the previous paragraph by setting the i^{th} location (among $\{1, \ldots, L\}$) to 1 if it is the current location, 0.5 if it is one of the remaining deliveries and -0.5 if it is one of the remaining pickups. An example is given in Fig. C.1, where the current time has been added as a number between -1 et 1. This number is obtained by normalizing the state's number of seconds since midnight.

Input entry	**Encoded vector**
$X = \begin{pmatrix} t\text{=9:25am} \\ \lambda\text{=6} \\ \mathcal{D}\text{=\{2\}} \\ \mathcal{P}\text{=\{3,4\}} \end{pmatrix}$	$\left(\underbrace{0.568}_{\text{time}}; \underbrace{0; 0.5; -0.5; -0.5; 0; 1; 0}_{\text{locations}}\right)$

Figure C.1: Encoding of an input on a 7-location world ($L = 7$)

Appendix D.

Object-Oriented Program: Structure

Fig. D.1 shows the structure of the system through the implemented classes in the object-oriented framework. Simulations are made possible thanks to the generation of scenarios inside the *ScenarioLayout* class. This class reflects past historical data with its initialization functions *initialize_ distrib_ appearance initialize_ distrib_ time_ mean* and *initialize_ time_ var.* Once scenarios are generated (and stored under the form shown in Appendix B.1), it is possible to simulate these scenarios, using the get_travel_time function by generating the successive *States.* All these simulations are done inside the *Simulator* class and eventually provide more training data to train the *Policy* object. This whole process can then be repeated several times to obtain an interesting policy.

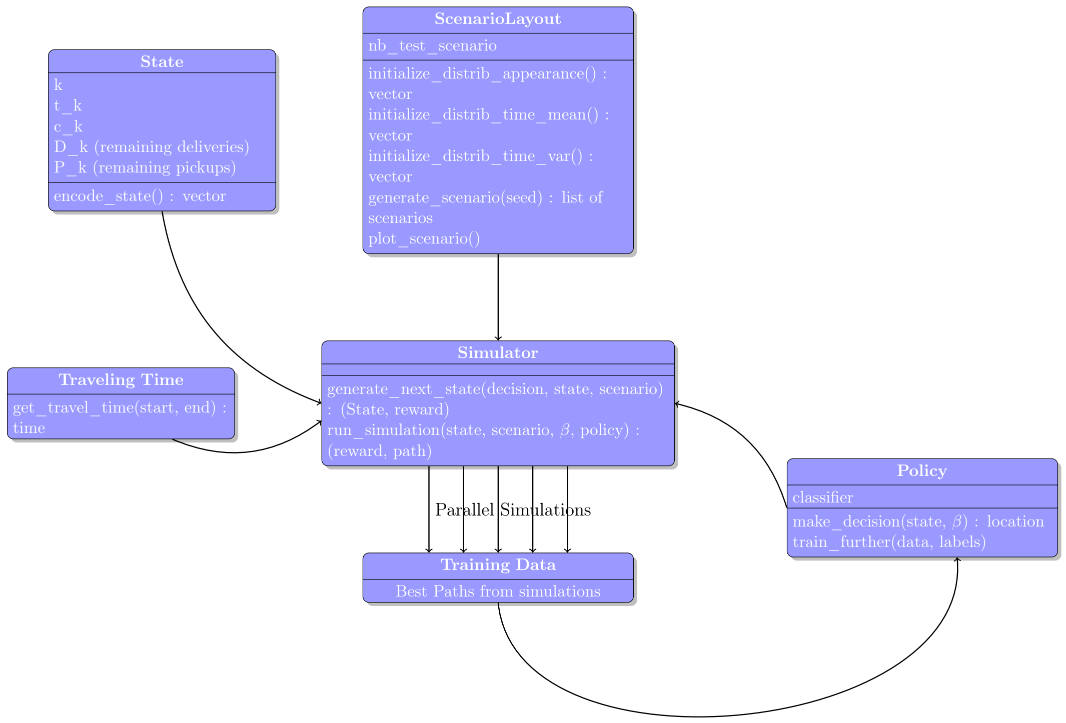

State
k
t_k
c_k
D_k (remaining deliveries)
P_k (remaining pickups)
encode_state() : vector
ScenarioLayout
nb_test_scenario
initialize_distrib_appearance() : vector
initialize_distrib_time_mean() : vector
initialize_distrib_time_var() : vector
generate_scenario(seed) : list of scenarios
plot_scenario()
Traveling Time
get_travel_time(start, end) : time
Simulator
generate_next_state(decision, state, scenario) : (State, reward)
run_simulation(state, scenario, β, policy) : (reward, path)
Parallel Simulations
Policy
classifier
make_decision(state, β) : location
train_further(data, labels)
Training Data
Best Paths from simulations

Figure D.1: Object-Oriented Structure of the system

Appendix E.

Travel Times in the grid world

Travel times between locations in Singapore can be modeled as a 2^{nd} order polynomial function of the distance between the locations (see Section 4.1.1). To preserve this pattern in our grid world, we scale the function giving travel times, using the following variables:

- m_d: maximum distance between existing locations
- g_v: duration factor

In our model, the function giving the travel times will be given by:

$$f(D) = -\frac{1}{m_d g_v} D^2 + \frac{2}{g_v} D$$

so that the following properties hold:

- $f(0) = 0$
- $f(m_d) = \frac{1}{g_v}$

When inverted, the duration factor gives the travel time corresponding to the trip between the farthest apart locations. It is thus a way to control the degree of congestion in the model.

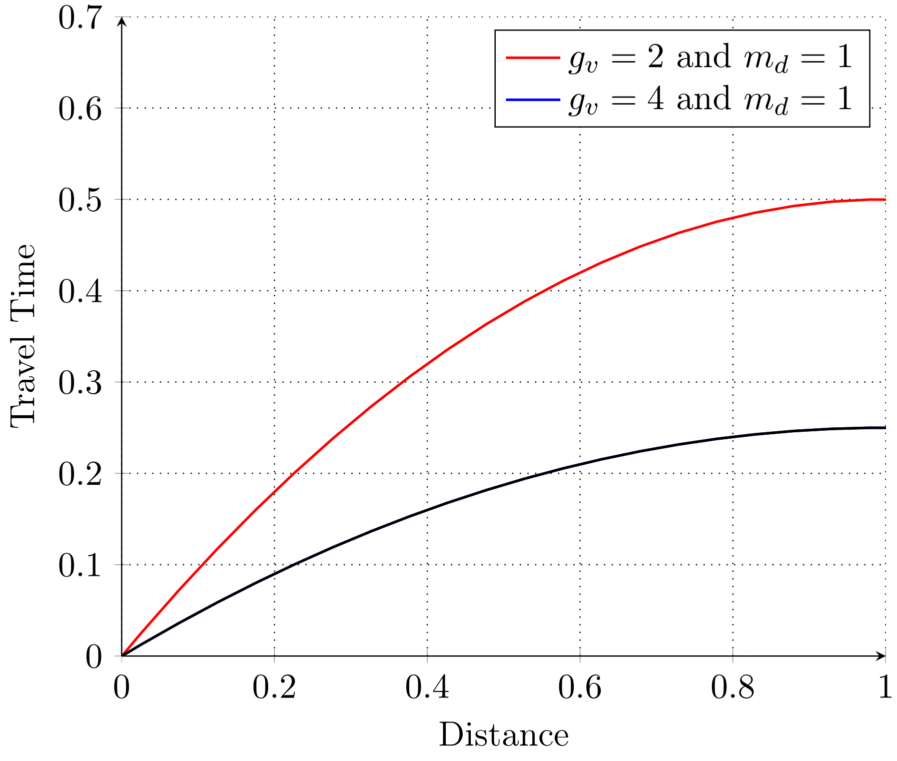


Figure E.1: Examples of f functions