Rethinking OR with RL

Maxime Cauté*

June 19, 2020

Abstract

In this paper, we use deep learning to learn policies in delivery service problems. We focus on the dynamic Dial-a-Ride Problem with nearest neighbour as a target policy. Our goal is to evaluate and explore the methods and architecture to replicate such a policy. To evaluate the performance of our network, we will mostly evaluate the accuracy of its selection compared to the policy, but also evaluate its deviations from this policy.

1 Introduction

On the one hand, Operational Research (OR) is a long-time studied domain that has proved efficient in finding good solutions in complex problems through modelization [1–3]. On the other hand, Reinforcement Learning (RL) is a more recent field where (local) optimal solutions can be found on these same problems through interaction-based optimization. RL has thus been used to tackle several theoretical situations, formerly resolved with pure OR, with great success. Among them stand general Atari game-playing, resolved by deep model-free Q-learning [4], control of robot motors, by end-to-end policy search through deep neural networks and search trees [5], and go-playing, leading to a famous victory over a human expert, by a combination of strategic policies and RL search. [6]

As remarkable as they are, these feats fail to generalize to real world situations. In fact, RL algorithms generally require large data sets and many optimization steps before displaying somewhat good results. The problems presented above came with accurate simulators, which are not to be found in real-world situations. In these cases, agents have to learn by interacting with the real world, with whatever potentially harmful consequences this might have. At this limitation of RL lies the interest of model-based OR, which can produce efficient policies without extensive trials. As both display noticeable advantages, combining them seems promising: a starting OR policy would ensure good enough results from the beginning while subsequent RL would locally improve

*under the supervision of Alexandre Alahi, VITA laboratory, EPFL, Lausanne, Switzerand

this policy. Combination of OR and RL has thus recently been an increasing exploration ground. One of the most intuitive method relies on Imitation Learning (IL): with this method, neural networks are able to learn and replicate results from expert, OR policies. They can hence be pre-trained with an expert policy, as in [7] for Atari Games.

We propose here to evaluate the learning of OR policy in the context of the Dial-a-Ride Problem (DaRP). More specifically, we consider maps of DaRP situations and train our network to replicate an expert policy. The next step is to use RL from this trained network to get the benefits of this method. For this we use standard algorithms and architectures of deep IL and RL.

2 Background: The Dial-a-Ride Problem

The Dial-a-Ride Problem (DaRP) is a NP-hard problem part of the Vehicle Routing Problems (VRPs), where a fleet of agents (vehicles) has to be routed to achieve a set of deliveries. DaRP instances specify these deliveries as transportation services. This means that agents have to pickup customers from a certain point A and get him brought to another point B. While VRPs are traditionally linked to graphs, in real life DaRP instances often take place within dense transportation networks (e.g. cities), so that their pickup and delivery locations can be considered nodes of a grid G. Although agents can be modelled as mere points of the grid to code their location, further characteristics can be considered for them. Therefore, agents can be seen as a set of feature vectors $\mathbf{v}$ located on this predefined grid. The situation further involves a request made of a pickup and a delivery locations, with potential additional characteristics. We also specifically consider dynamic instances, which means that requests are not known beforehand. This makes our problem especially difficult as there can not be any predetermined optimal solution. Computations thus have to be done and updated through the whole course of the issue.

3 Related Work

DaRPs have been extensively studied since their first formalization in 1986 by Jaw et al. [8], due to their application to industrial concerns. The first approaches have been focused on OR resolutions. An extensive 2018 review can be found in [9]. A deterministic, exact approach was first presented by Cordeau in 2006 [10]. The proposed algorithm relied on branch-and-cut methods. However, exact method suffer from the NP-Hardness of DaRP: their computation time is exponential to the entry size!

Many modern OR approches to tackle this problem are therefore now based on heuristics.

In 2019, Claudia Bongiovanni et al. [11] proposed a policy for electricity-powered vehicles. This policy was slightly explored throughout the course of this work, but a much simpler one was considered in the end for exploration

purposes. It relies on a two-phase insertion algorithm: the requests are, if possible, first assigned to the vehicle best able to respond to it through limited schedule reorganization (e.g. segments shifting); after a certain amount of new requests, an optimization process is then started with intra- and extra-route modifications, for previously approved requests only. This optimization process notably employs Large Neighborhood Search.

Reinforcement Learning has also been considered as another solution method for this very problem. This prospect has only been recently explored, hence a limited amount of related works.

Its potential was yet shown in 2015 by Vinyals et al. [12], who achieved remarkable accuracy in discovering the optimal policy for several combinatorial problems through deep neural networks. Their architecture, called Pointer Network, adapts Long Short-Term Memories Recurrent Neural Networks for variable input size.

In the field of DaRP application, in 2019, Al-Abbasi et al. [13] considered a deep RL, model-free approach. They use convolutional, deep Q-networks in order to learn optimal fleet dispatch policies with regards to both customer (ride time) and company (resource use) points of view. A notable feature is the use of a distributed decision-making, as the vehicles learn individually from the environment. The framework is further exploiting customer statistics, trying to predict further demand. These two key points lead the authors to better results than frameworks omitting them.

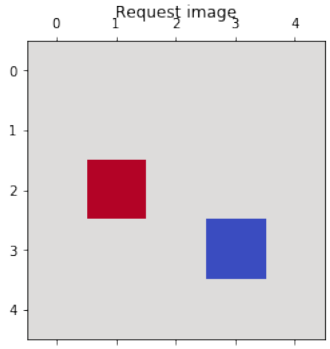
The methods considered above try to compare policies learnt from the environment to expert-given, optimal policies. In 2018, Stocco and Alahi [14] considered a novel approach for DaRP, which rather aimed at directly learning the expert policy. To this extent, they used supervised reinforcement learning through a deep neural network. As the input was turned into an image, the considered architecture consisted in 3 successive, batch normalized, linearly rectified convolutional layers of size 5x5, followed by 3 similar deconvolutional ones. Compared to nearest neighbor policy, they report a promising accuracy around 75%, image projection being likely responsible for a 25% information loss.

4 Experimental setup

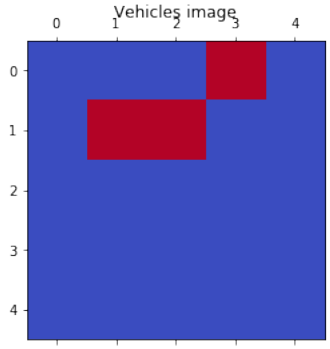
4.1 Data Representation

Our first concern was to depict our DaRP instances. As image processing is a well-explored and promising field in Deep Learning (DL), we settled for image-like structures like in [14], similarly to building a situation map. Instances have thus been represented as a $n \times n$-sized grid G. The cells of the grid represent nodes of the network (e.g. city crosses/neighborhoods, depending on the chosen resolution). Like an image, our grid is composed of different channels to represent the relevant information of the situation.

The first channel G_1 is dedicated to the customer request. Locations of interest (such as the pickup and delivery points) are represented by unique



(a) Request map



(b) Vehicle map

Figure 1: Representation of the 2-channel situation map for 4×4 grid with 3 agents

values on their respective cells while other irrelevant cells just share a single other value. Interestingly, further parameters can be introduced by changing the values accordingly: the values of pickup and delivery might represent the number of passengers by their absolute value while unoccupied cells valuate 0. For normalization purposes, we chose to represent pickup node by a $+1$ value and unoccupied nodes by -1 value. Delivery location was in the end not represented as it is not relevant to our policies.

Other channels are dedicated to vehicle information. For complex situations, it would probably be very relevant to use one channel or more per agent, as it would increase the description power of the map. However, we settled for a more simple model of one single channel for all the agents, as the situations we dealt with were rather simple. Therefore we have a second channel G_2 for the vehicles, where occupied cells are represented by a $+1$ value and unoccupied ones by -1 values, once again for normalization purposes. Supplementary information could be integrated such as the amount of vehicles in cases of overlapsing. However, our data set did not contain such cases.

A display of the images on a 10×10-sized example with a fleet of size 3 can be observed in figure 1.

4.2 Data set

We generated a whole data set to train and test our network. To generate an example, we first generate a request by randomly selecting a pickup cell. Then we generate the vehicles fleet iteratively. Our aim is to have only one nearest neighbour at the end. In order to do this, we ensure that we only have one at all time, and generate a vehicle again if it is as close as the current nearest neighbour. Note that this is affects the structure of the data set somehow, although it is probably not harming our results by much. We also ensure that no vehicle overlaps by regenerating them in this case. The algorithm is described as algorithm 1.

Data: grid size n, fleet size k
Result: a DaRP instance
initialize an empty vehicle list;
initialize minimal distance as $+\infty$;
for *i = 0 to k* **do**
  repeat
    take a random cell;
    compute distance to request;
  until *the cell is not occupied AND its distance is not the minimal one.*;
  if *the distance is lesser than the minimal one* **then**
    replace it
  end
end

Algorithm 1: Situation generation algorithm

With this generation algorithm, we did independantly generate 4.000.000 examples on a 30×30 grid with 10 agents. This number is highly correlated to the evolution of our results.

We then generated the solutions to these examples that the network would have to replicate. We settled for the Nearest Neighbour (NN) policy for the network to learn, as this policy was rather simple, yet not trivial. Previous tests did involve Bongiovanni's policy [11] on simplified situations. However it was discarded after average results, in favor of a simpler policy for exploration purposes. We thus generated, for every example, its solution as the coordinates of the cell to select.

60% of the test set was used for training, while the rest stood for performance evaluations.

4.3 Training and testing protocols

We evaluated the policy-learning ability of the network with a Python and PyTorch implementation. Given a chosen architecture, it would be given as input the 2-channels map of the situation and would be expected to ouput a 1-channel map on the basis of which a vehicle would be selected.

This map is basically a score map for each cell, and the cell with the highest score ends up being the selected one. Note that the selected cell does not necessarily hold a vehicle in it.

A few loss criterions were experimented early for backpropagation. As it yielded the best results at the time, the cross-entropy criterion was kept for loss computation.

The learning rate was set at 5×10^4 for baseline. In order to refine the results, it has been decreased up to 1×10^4 at fixed epochs. Warm-up was also introduced for 20 epochs as a classical result-improvement method of the field.

5 Network architecture for policy learning

Our aim is to learn what (hopefully occupied) cell should be assigned to a given request from a situational map. We took inspiration from the Monoloco network for architecture, presented in [15]. We performed our test on a network with 12 hidden, fully-connected layers of size 512 each. Each hidden layer was followed by batch normalization, Rectifier Linear Units (ReLU) and 20% dropout to increase regularization, as classical DL methods. Input layer was necessarily of size $2 \times (30 \times 30) = 1800$ and output layer of size $(30 \times 30) = 900$. Due to limited learning efficiency, we did further introduce skip connections between odd-indexed layers. This amounts to a total of 2.966.916 parameters. The architecture can be visualized on figure 2. Several changes of parameters were also considered and studied throughout the course of our research, and will be described in the Results section.

Convolutional layers were also considered as a link to image processing. However, as our NN policy was rather simple and as they seemed not much of interest on early trials, they were deemed non-necessary and left aside for the time being.

6 Results

6.1 Evaluation methods

As mentionned before, we tested a DL approach to try to learn our NN policy in DaRP instances. We thus had to come up with methods for evaluation of the results.

Our main evaluation is a basic accuracy evaluation of the output. In other words, the cell chosen by the network (as the cell with highest score on the output) is compared to the cell selected by our policy. The percentage of matching

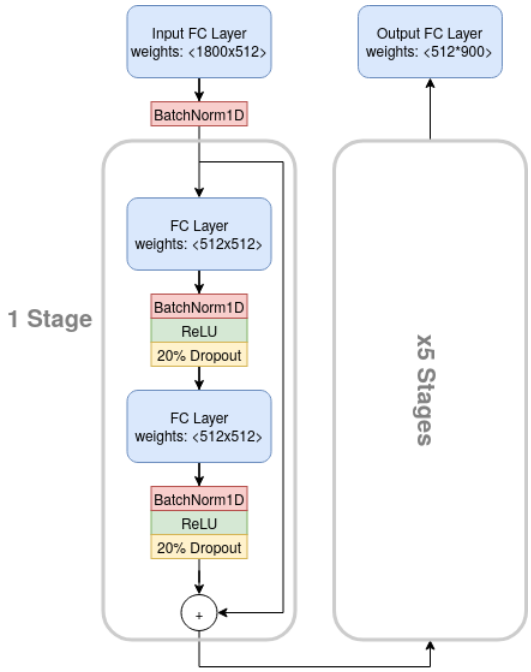


Figure 2: Representation of our main architecture

selections is then considered network accuracy, and is computed separately on both training and testing data set.

We also developped methods to gain better insight on the output. Our first auxiliary test is a vehicle selection accuracy. This means that we compute the percentage of the instances where a vehicle is actually selected. Our second auxiliary test is a mean distance evaluation. This means that we compute the average distance from the right cell. Note that both these computations are run on the whole training set (even on accurate responses). It is however possible to compute the value for inaccurate ones only, as the value is known for accurate ones. If a is the main accuracy and a_v the vehicle accuracy, then the wrong vehicle accuracy is $a'_v = \frac{a_v - a}{1-a}$. Similarily, if m_d is the mean distance, the mean non-null distance is $m'_d = \frac{m_d}{1-a}$.

Another method for result evaluation is loose accuracy evaluation. This evaluation actually consider another cell selection method from the network output. Rather than selecting the cell with maximal score, it selects the closest vehicle to this cell.

6.2 Architecture results

Our experimental results for the Monoloco architecture [15] can be observed in table 1.

Table 1: Testing accuracy evaluation for the Monoloco architecture

(fully-connected layers of same size with skip connections)

Architecture ID	Layers	Layers Size	Parameters	Test accuracy[1]
$MNLC01$	6	512	$2,966,916$	72%
$MNLC02$	6	256	$1,090,692$	58%[2]
$MNLC03$	16	128	$615,172$	52%[2]
$MNLC04$	10	256	1.355.908	62%[2]

$MNLC02$, $MNLC03$, and $MNLC04$ architectures get stuck at respectively 60%, 50% and 70%. training accuracy. As changing learning rates seems to have no effect on it and seeing that the testing accuracy is very close, our guess is that they actually lack the ability to capture the structure of our problem. Yet the runtime is already pretty long as computations last over several days, so we did not run.

However, these architectures show promising results. They are in fact far better in comparison to random selection. A random vehicle selection, for a 10 vehicle fleet would lead to a probability $\frac{1}{10} = 10\%$ of choosing the right one. A random cell selection is even worse as a 30×30 grid would end up with $\frac{1}{30\times30} = 0.33\%$ right selection chance! Deeper architectures are more interesting as they have less weights to update, however this problem seems to require some layer width as underlined by the results of $MNLC03$.

Further auxiliary evaluations can be observed in table 2. As described in 6.1, vehicle accuracy reflects the frequency to which a vehicle (any of the fleet) is selected, mean distance evaluates the mean distance to the right cell from the network's choice, and loose accuracy evaluates the accuracy when selecting the closest vehicle to the network's choice.

Table 2: Auxiliary evaluations for the Monoloco architecture

(fully-connected layers of same size with skip connections)

Architecture ID	Parameters	Vehicle accuracy	Mean distance	Loose Accuracy
$MNLC01$	$2,966,916$	96%	2.46	77%
$MNLC02$	$1,090,692$	N/A	N/A	N/A
$MNLC03$	$615,172$	86%	4.82	53%
$MNLC04$	$1,355,908$	94%	4.218	65%

An immediate observation is that the networks prove efficient in selecting vehicles. The loose accuracy is thus only slightly better than the basic test accuracy, improving it nonetheless. However, the mean distances are also somewhat encouraging. On a 30×30 grid, the average distance between two points randomly selected is approximately 15.63. Our results are below this threshold even

[1] on NN policy

[2] training not complete (either stuck or too long)

when considering wrong selections only (resulting in a 8.66 average distance). It is most likely that the vehicles positions are determining this outcome, but the network seems to take some sens of distance into account.

6.3 Baseline comparison results

We compared those results to the one obtained with baseline comparison models. Those can be observed in table 3.

Table 3: Accuracy evaluation for the fully-connected architecture

Architecture ID	Layers	Layers Size	Parameters	Test accuracy[1]
$FC01$	1	2000	$5,408,700$	45%
$FC02$	3	512	$1,913,996$	37%
$FC03$	6	256	$1,026,188$	18%[2]

We compared our architecture to a fully connected one *without* skip connections. We ran tests on such architecture, although it has to be noted that only 500.000 examples were involved (a quarter of the total set) for computational and exploration purposes.

- Although $FC01$'s accuracy could be considered promising, its number of parameters actually makes it a terrible choice due to long computational time.
- The absence of skip connections is logically impacting deeper architectures such as $FC03$. The difference is here very significant with $MNLC02$: without skip connections, the results are more than 3 times worse! Therefore, in this problem, skip connections seems to be mandatory for a fully-connected architecture to learn accurately, for a cheap computational cost (about 7% more parameters).

Convolutional networks were also considered, but discarded after no real improvement on smaller data set due to time constraints.

7 Conclusion

We did develop a framework to gain insight on policy-learning through DL in the case of the nearest neighbour for dynamic Dial-a-Ride Problems with map representation. Even though we evaluated our networks on simple situations and policies, reaching close-to-perfect accuracy seems to require a few designing tricks to avoid enormous computation times. While deep fully connected networks prove interesting in handling this, they seem to still require skip connections even at average depth. Thanks to these designs, we were able to achieve

[1] on NN policy
[2] training not complete (either stuck or too long)

pretty good accuracies on a Nearest Neighbor policy for simplified DaRP situations. This ground exploration will most likely bear useful observations for more complex problems. A direct application of this is policy-improvement on dynamic DaRP instances, as our representation allows for much more complex situations through feature vectors. Using Reinforcement Learning after our policy-learning methods could likely lead to improved results.

Acknowledgements

Thank you to Prof. Alexandre Alahi for his kind and helpful supervision over this project, and to Natalie Bolón Brun for her shared insight from her previous work. Thank you also to Claudia Bongiovanni for providing tools for testing and for further explaining them.

References

[1] Denis Bouyssou, Silvano Martello, and Frank Plastria. Eleven surveys in operations research: Ii. *Annals of operations research*, 175(1):3–8, 2010.

[2] Gang Yu, Michael Argüello, Gao Song, Sandra M McCowan, and Anna White. A new era for crew recovery at continental airlines. *Interfaces*, 33(1):5–22, 2003.

[3] Robert Stahlbock and Stefan Voß. Operations research at container terminals: a literature update. *OR spectrum*, 30(1):1–52, 2008.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[5] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[6] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[7] Gabriel V Cruz Jr, Yunshu Du, and Matthew E Taylor. Pre-training neural networks with human demonstrations for deep reinforcement learning. *arXiv preprint arXiv:1709.04083*, 2017.

[8] Jang-Jei Jaw, Amedeo R Odoni, Harilaos N Psaraftis, and Nigel HM Wilson. A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Transportation Research Part B: Methodological*, 20(3):243–257, 1986.

[9] Sin C Ho, WY Szeto, Yong-Hong Kuo, Janny MY Leung, Matthew Petering, and Terence WH Tou. A survey of dial-a-ride problems: Literature review and recent developments. *Transportation Research Part B: Methodological*, 111:395–421, 2018.

[10] Jean-François Cordeau. A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, 54(3):573–586, 2006.

[11] Claudia Bongiovanni, Mor Kaspi, Jean-François Cordeau, and Nikolas Geroliminis. A learning large neighborhood search for the dynamic electric autonomous dial-a-ride problem.

[12] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in neural information processing systems*, pages 2692–2700, 2015.

[13] Abubakr O Al-Abbasi, Arnob Ghosh, and Vaneet Aggarwal. Deeppool: Distributed model-free algorithm for ride-sharing using deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 20(12):4714–4727, 2019.

[14] Teo Stocco and Alexandre Alahi. Learning online combinatorial stochastic policies with deep reinforcement. Technical report, 2019.

[15] Lorenzo Bertoni, Sven Kreiss, and Alexandre Alahi. Monoloco: Monocular 3d pedestrian localization and uncertainty estimation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6861–6871, 2019.