# Learning online combinatorial stochastic policies with deep reinforcement

Teo Stocco & Alexandre Alahi*

**Abstract** We study on-demand delivery services in stochastic environments and propose one representation of the online vehicle routing problem that can be learned with supervised and reinforcement learning. Using a simulation framework with topology projections, we evaluate different models and show that stochastic policies can be learned. Fine-tuning the models through the use of the `REINFORCE` rule and with a deterministic critic suggests that this approach can lead to promising policies to solve the problem.

## 1. Introduction

The last decade has seen a rapid growth of on-demand delivery services such as Uber (people), Deliveroo (food) and Glovo (groceries). These services are usually characterized by a time-varying number of parties involved (drivers, restaurants/shops, customers) and face strong asymmetries. The customers tend to share the same behaviors, triggering congestions and peaks as they order similar items at the same time. These environments are also under strong stochastic influences due to periodic traffic conditions and meteorological events. All of these raises an interesting online combinatorial challenge.

Each time a new task fires, a decision needs to be taken in order to match it with the best agent according to an established objective. In the setting where all tasks are known in advance (also known as offline), you may have recognized the classical Vehicle Routing Problem (VRP, Dantzig and Ramser 1959). The VRP tries to find the optimal delivery routes for a fleet of vehicles and a set of places, generalizing the famous Traveling Salesman Problem (TSP) to multiple agents and constraints. Solutions to this NP-hard problem usually rely on greedy approaches, linear integer programming relaxations, heuristics and meta-heuristics (Kumar and Panneerselvam 2012). Among the dozens of VRP flavors studied in the literature, the followings characterize our case of study: capacited (CVRP), with time windows (VRPTW), with pickup and delivery (VRPPD), with heterogeneous fleet (HFVRP), online/dynamic (OVRP/DVRP) and stochastic (SVRP) (Pillac et al. 2013).

We are interested in the online VRP optimization with multiple agents and tasks. Each task requires the agent to *pickup* and *dropoff* some items at targeted times. The agents have different capacities and are affected by stochastic delays at both the pickup and the dropoff points. They cannot refuse a

*VITA laboratory, École Polytechnique Fédérale de Lausanne, EPFL.

task, and all the tasks have the same level of priority. According to the previous nomenclature, this configuration can be characterized as O-C-HF-S-VRP-PD-TW in the operation research taxonomy. Although we relax the capacited and heterogenous fleet constraints by having a single item to pickup and the same type of vehicle (with a capacity of one), all the methodology and principles we suggest can be extended to the fully featured case.

Formally we have a stream $T = \langle t_1, t_2, \dots t_m \rangle$ of $m$ time-order tasks and a set of $n$ agents $A = \{a_1, a_2, \dots, a_n\}$ for an episode $E = (T, A)$ in an original metric space $M$. Each task $i$ can be written as a tuple $t_i = (p_{up}, \mathbf{w}_{up}, p_{off}, \mathbf{w}_{off}, \tau_c, \tau_d)$ where:

- $p_{up} \in M$ a *pickup* location;
- $\mathbf{w}_{up}$ a vector of *pickup* features (e.g. mean pickup time at this location, item types, etc.);
- $p_{off} \in M$ a *dropoff* location;
- $\mathbf{w}_{off}$ a vector of *dropoff* features (e.g. dropoff estimated distance, customer types, etc.);
- $\tau_c$ the time at which the task was created;
- $\tau_d$ the time at which items should be delivered.

The agent $i$ can be described as a tuple $a_i^{(k)} = (p_a^{(k)}, \mathbf{w}_a^{(k)}, \tau_a^{(k)})$ at evolving step $k$ where:

- $p_a^{(k)} \in M$ current or future agent position, initialized at random;
- $\mathbf{w}_a^{(k)}$ a vector of agent current or future features (e.g. agent rating, distance driven, etc.);
- $\tau_a^{(k)}$ the next time at which the agent is available for next task.

The goal is to find a matching stream $P = \langle (t_1, a_j^{(1)}), (t_2, a_k^{(2)}), \dots, (t_m, a_l^{(m)}) \rangle$ such that the average delay $\tau_{a_j}^{(i)} - \tau_{d_i}$ is minimized in a stochastic environment.

Although there are many possible applications, relatively few pieces of work combining both stochastic and online VRP have been done yet (Pillac et al. 2013). Traditional solutions involve heuristic, meta-heuristic and evolutionary strategies (Garrido and Riff 2010). Looking for an alternative and avoiding hand-crafted or general strategies, we propose a framework for learning online policies both in supervised and reinforced settings. Our contributions are threefold:

1. we explore how the spatial structure of the problem can be exploited to transpose the online VRP into an easily extendable and learnable representation;
2. we show that deterministic policies can be approximated using the suggested representation;
3. we investigate how learned policies can take advantage of reinforcement while evolving in a simulated stochastic environment based on real data.

## 2. Related work

The operation research community has extensively studied the vehicle routing problem. Ascheuer, Krumke, and Rambau (2000) compared the `IGNORE` strategy that ignores new requests until current are completed and the `REPLAN` (also known as `REOPT`) strategy that replans everything as soon as a new request arrives. They showed that both are 5/2-competitive and proposed the `SMARTSTART` strategy which combines the two previous ones, assuming the requests do not last too long. The latter has a competitive ratio of 2 that matches their lower bound. Lipmann (2013) extended the analysis to cases with incomplete ride information (e.g. final destination not known) and suggested a case-specific 3-competitive algorithm for these situations.

In the stochastic variant, various configurations having either stochastic demands or requests have been reviewed, and near-optimal heuristics developed (Gendreau, Laporte, and Séguin 1996). Bertsimas and Van Ryzin (1991) studied the stochastic and online VRP in the Euclidean plane. They propose

case-specific policies that try to capture the dynamics of the environments and compare them with the nearest neighbors policy which they show performing well in all situations. Bent and Van Hentenryck (2004) investigate later the problem with service guarantees where some requests can be rejected and avoid sub-optimal choices. They showed improved performance over the nearest neighbors in such examples.

The recent advances in neural combinatorial optimization show that deep learning and deep reinforcement learning are promising tools for tackling the offline VRP. The pointer network architecture was introduced by Vinyals, Fortunato, and Jaitly (2015) to solve discrete combinatorial problems. This new approach is length-invariant and allows to learn a sequential output with each element pointing to a corresponding input through a RNN encoder/decoder architecture. They applied it successfully to the planar TSP ($n \leq 20$) and other discrete problems. Bello et al. (2016) built on the top of this to present a deep reinforcement learning approach based on policy gradient and thus avoiding the cost of labeling. They showed near-optimal results on bigger instances of TSP ($n \leq 100$).

Choosing Q-learning instead of policy gradient Dai et al. (2017) used graph embeddings to provide a more natural way to deal with these discrete problems and obtain results as good as deterministic policies for even larger TSP instances ($n \leq 300$). While simplifying the pointer network architecture by removing RNN dynamic in the encoder layers, Nazari et al. (2018) proposed a network invariant to the input sequence and applied it to the offline VRP with attention mechanism. They showed close to optimal results using the `REINFORCE` rule on instances up to 50 customers. Finally, Kool, Hoof, and Welling (2019) showed how to beat the best offline TSP-baseline known as the farthest insertion by combining graph, attention and policy gradient with a greedy baseline.

## 3. Methods

The previously mentioned machine learning literature evaluates their algorithms in cases where positions are sampled from the Euclidean plane. The coordinates are then used either as raw feature or transformed into a distance matrix encoding the pairwise distances. This approach requires the use of length invariant models as the number of agents is likely to evolve (due to the natural dynamics of demand/offer or unpredictable events such as incidents and network connectivity loss) and invariant to the input sequence as there is no clear motivation why the agents should be ordered. Additional complexity might also occur when having a broader set of agents (e.g. $j \geq 100$) as in real cases.

Another approach would be to feed the network directly with the full environment in a representation space $S$. Assuming all the agents $a_i^{(k)}$ only have a single position $p_a^{(k)}$ and a task $t_i$ two positions $p_{up}, p_{off}$ one can project them from the original metric space $M$ directly to $S$. The hard thing is to design such spaces by keeping the bijectivity, as multiple agents could share the same projected positions. Surjectivity can nonetheless be enough, as agents projected onto the same spot can be assumed to be fairly equivalent when the distance is more impactful than other features. The policy can then use the projection map to find which position is the best and select the agents arbitrarily within the shared position.

**Representation** An effective way of projecting features is to take advantage of images and tensors structures. We first define a bounding box containing the area of interests and use the projection $\phi_p : M \mapsto S$ as the Mercator projection[1] (*EPSG:3857*). This transformation preserves angles but might distort distances and areas. While keeping bounding box small (dozens of kilometers), those effects

[1]The Mercator projection was chosen as it is the *defacto* standard to maps on the web and phones. Distance preserving projections (geodesic to planar) for all the points in the bounding box is geometrically not possible. However, the gnomonic projection can preserve the shortest route property and replace the Mercator one.

should not be significant. Each projected points are then clipped into the projected bounding box to give a rectangular $W \times H$ (width x height) plane as shown in fig. 1.

The planar representation suggests efficient use of 2D convolution to derive translation invariant model. The agents can be shaped into sparse tensors of size $(|\mathbf{w}_a^{(k)}|, H, W)$ and tasks into a pickup tensor of size $(|\mathbf{w}_{up}|, H, W)$ and another dropoff tensor of size $(|\mathbf{w}_{off}|, H, W)$. The additional features are only set on projected positions, the rest being zeroed. Stacking all three together gives an input tensor of size $(|\mathbf{w}_a^{(k)}| + |\mathbf{w}_{up}| + |\mathbf{w}_{off}|, H, W)$, which can be considered as an image with multiple channels.

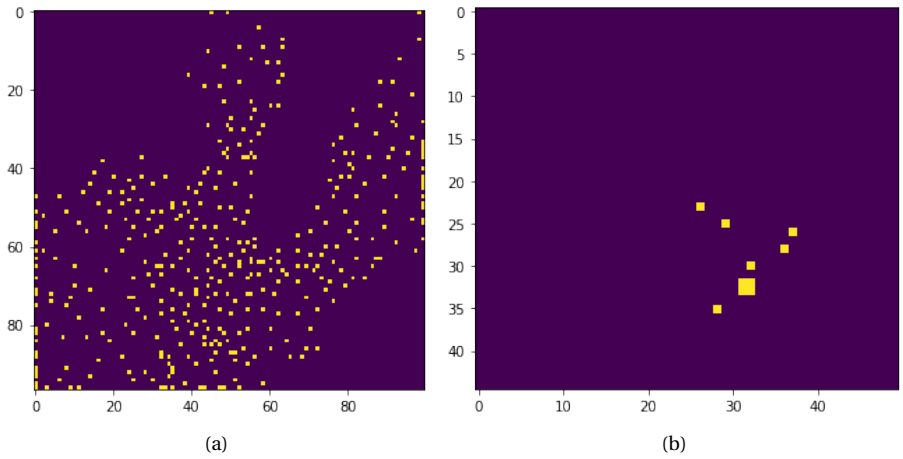


Figure 1: *(a) 391 road crossings in Geneva projected onto the image representations. (b) 11 agents projected leading to 10 active sites on the image.*

**Policy modeling** Now that we have tensorial representations of the full environment and invariant to the number of agents, we can focus on learning architecture. To keep results comparable with potential future work, we only make use of blocks composed of standard 2D convolution (Krizhevsky, Sutskever, and Hinton 2012) followed by batch normalization (Ioffe and Szegedy 2015) and rectified linear units (Nair and Hinton 2010). The model ($3'203$ parameters) consists of 3 successive blocks of $5 \times 5$ convolutions with 5 channels followed by 3 successive blocks of $5 \times 5$ de-convolution to go back to the original input shape.

**Reinforcement** When lacking labeled data, reinforcement learning can be an exciting alternative. By defining a positive or negative signal/reward $R_t$ at the end of each action or simulation, the process can be viewed as a Markov decision process that improves the estimate of state-action-values (dynamic programming/Q-learning) or directly action-values (policy gradient, Arulkumaran et al. 2017). The reward is especially interesting for combinatorial problems as it allows to model soft constraints by penalizing the reward, which is usually hard to define in deterministic policies.

Inspired by Silver et al. (2016), we investigate further to see whether reinforcement learning can improve on a previously learned supervised policy. More recently Kool, Hoof, and Welling (2019) suggested that the use of a deterministic baseline as a critic can also help to find better policies. The `REINFORCE` rule can thus be generalized to integrate the feedback of a deterministic or previously

learned policy as shown in algorithm 1.

**Algorithm 1:** REINFORCE with baseline

**Input:** epochs $E$, batch size $B$, differentiable policy $\boldsymbol{\theta}$, baseline policy $b$, discount factor $\gamma$

**for** *epoch in* $1,\dots,E$ **do**
  $\ell \leftarrow \mathbf{0}$
  **for** *run in* $1,\dots,B$ *(parallel)* **do**
    generate episode $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}$ from $\pi(\cdot \mid S, \boldsymbol{\theta})$
    **for** *step* $t = 0,\dots,T-1$ **do**
      $r \leftarrow \sum_{k=t+1}^{T} R_k$
      $\delta \leftarrow r - b(S_t, A_t)$
      $\ell \leftarrow \gamma^t \delta \nabla \ln \pi(A_t \mid S_t, \boldsymbol{\theta})$
    **end**
  **end**
  $\boldsymbol{\theta} \leftarrow \mathrm{ADAM}(\boldsymbol{\theta}, \ell)$
**end**

## 4. Experiments

All the experiments were run on an Intel Xeon 24x2.60GHz E5-2690v3 with 220Go of RAM and 4xTesla M60 8Go. Models were trained with PyTorch (Paszke et al. 2017) using the ADAM optimizer (Kingma and Ba 2014) with default parameters ($\beta_1 = 0.9$, $\beta_2 = 0.999$ and no weight decay) and later specified learning rates.

**Supervised policies** While the original problem has been defined with any number of tasks and agents features, we will concentrate our efforts on 4 simple channels. Each agent has an indicator variable for its position and another indicator variable to know whether they already are handling a task or not $\mathbf{w}_{up} = (\mathbb{1}\ \mathbb{1}_{free})^\top$. The features for task pickup and dropoff also have each an indicator variable for their locations $\mathbf{w}_{up} = \mathbf{w}_{off} = (\mathbb{1})^\top$.

To compare[2], we apply a deterministic policy and label the historical demand data lend by a local delivery company. The policy selects the nearest neighbor to the pickup location in the original metric space. After projection, we reach better results with noticeably higher accuracies and interesting distances. Tbl. 1 indicates that about at least 25% of the information is lost or cannot be learned due to the projection. This loss might be more important as counterbalanced by the accuracy gained through agents collisions. However, the distances give satisfactory results and show that our proposed architecture can indeed learn from these representations.

Table 1: *Experimental data relabelled with the nearest neighbor strategy. Batch size 128, learning rate 0.001.*

| Topology | Accuracy ± 0.3% | Distance ± 0.2 |
|---|---|---|
| Image 30x27 | 75.0% | 1.0 pixel |
| Image 50x45 | 77.0% | 1.2 pixel |
| Image 80x72 | 72.9% | 2.5 pixel |

[2] One could also compare those policies to an open source offline solver such as Google or-tools (Omme, Perron, and Furnon 2014), but it is not clear how the objective of minimizing the total completion time should relate to minimizing the average delay, especially under soft constraints and stochasticity which characterize our case of study.

**Reinforcement** While learning a policy for imitation might be useful in some cases, the ability to learn something without previous data and potentially more powerful strategies is appealing. As previously stated, the reinforcement learning approach opens a way of integrating soft constraints by modulating the reward signal. One such constraint could be to penalize order delivered too early and too late. However this kind of goal would require the ability to delay the task assignment (e.g. not immediately selecting an agent for a given task) and a more complex simulation, we thus focus on minimizing the delivery delay for each task.

In order to compare within the original metric space, we define 3 deterministic policies based on the same features used in learned policies that will serve as a baseline. More complex policies are also possible but they might use more features, and the comparison would not be as fair as with the following ones:

1. Random policy (`dt-random`): selecting an agent at random
2. Parallel policy (`dt-parallel`): selecting the free agent first
3. Parallel duration policy (`dt-parallel-duration`): selecting the free agent which is the closest travel time-wise

The number of agents in a policy trained by reinforcement from scratch can slightly vary because it has to generalize over the projected collisions. However, obtaining a true invariance in the number of agents require advanced fine-tuned training with a mixed and evolving number of agents. Pre-training first the policy in the supervised setting can avoid these struggles and give superior results as shown in tbl. 2 and illustrated in fig. 2.

Table 2: *Average delay (in minutes) of all policies over 50 simulations. The policies are ordered by best mean.* `dt` *prefix stands for deterministic and* `dl` *for deep learning (supervised). 40 tasks for 4 agents, batch size 16 and learning rate of 0.0005.*

| Policy | Min | Q25 | Mean | Median | Q75 | Max |
|---|---|---|---|---|---|---|
| dt-parallel-duration | -36.24 | -26.97 | -24.54 | -25.14 | -21.76 | -8.67 |
| **rl-image-critic** | -34.94 | -26.42 | **-23.57** | -23.57 | -21.34 | -11.25 |
| **rl-image-pretrained** | -36.00 | -26.26 | **-23.41** | 24.01 | -20.91 | -6.06 |
| dt-parallel | -35.58 | -25.91 | -22.92 | -23.30 | -20.14 | -6.83 |
| dl-image-data | -35.82 | -26.35 | -22.79 | -23.22 | -21.33 | -1.76 |
| rl-image | -34.18 | -25.35 | -20.11 | -22.10 | -17.62 | 7.70 |
| dt-random | -26.63 | -17.69 | -11.10 | -10.56 | -6.37 | 23.80 |

The performance of policies is more or less what we expected. All learned policies are by far smaller than noise and close to the deterministic parallel policies. The ones learned from scratch tends to be less complete as the supervised one and pre-training lead to even better-performing ones. Reinforced ones seem to manage better some good task combinations but also lack some performance in the worst cases.

The best-learnt policy `rl-image-critic` was trained first using supervised learning and then using reinforcement learning with the best policy as feedback. As previously suggested, it forces the policy to take risks and finding strategies that are better than deterministic ones. Not to mention that there must be some loss due to the projections, this shows that a mix of training can lead to promising results.

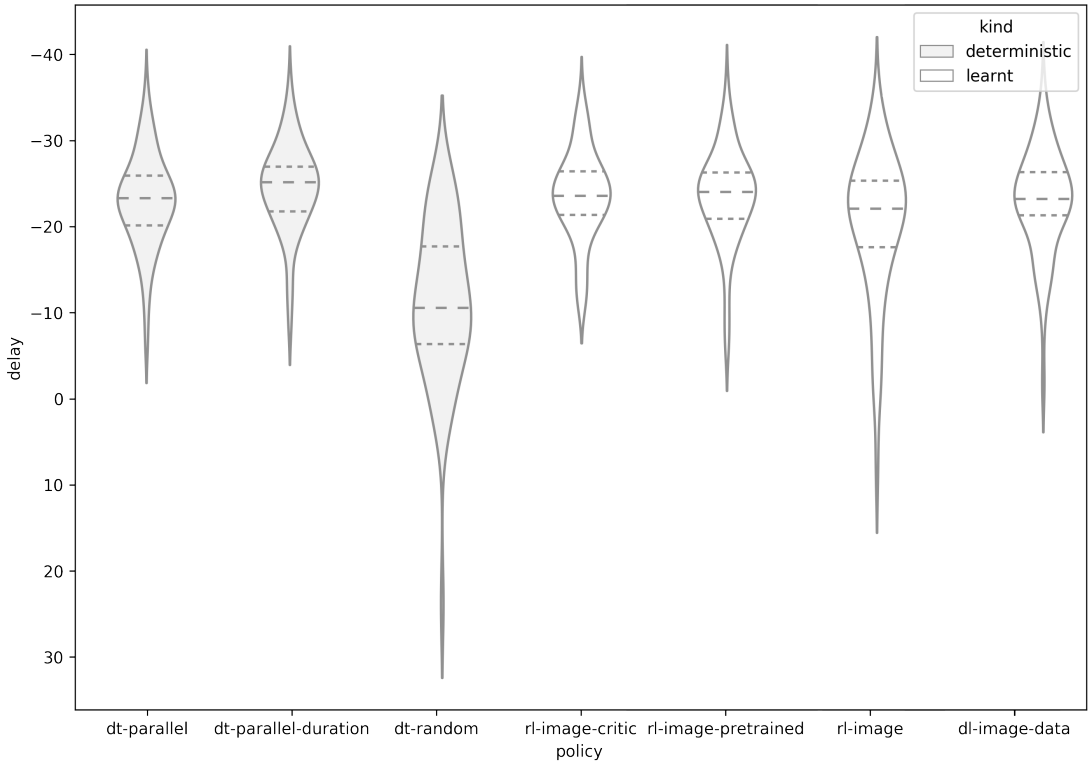


Figure 2: *Results from tbl. 2 illustrated as violin plots. The symmetric shape represents the distribution of delays (in minutes) for the given policies. Dashed lines represents the quartiles.*

## 5. Conclusions

We explored how structured representation can be learned by policies both from using supervised and reinforcement learning in a stochastic environment. Although there is an inherent loss due to the projections and training might be non-trivial, these learned policies show promising results. Nonetheless, further analysis and evaluations are needed to provide evidence in other contexts. The generalization allowing multiple tasks being delivered by the same agent in parallel might be one of the many potential future directions to study.

## 7. References


Arulkumaran, Kai, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. 2017. "A Brief Survey of Deep Reinforcement Learning." *arXiv Preprint arXiv:1708.05866*.

Ascheuer, Norbert, Sven O Krumke, and Jörg Rambau. 2000. "Online Dial-a-Ride Problems: Minimizing the Completion Time." In *Annual Symposium on Theoretical Aspects of Computer Science*, 639–50. Springer.

Bello, Irwan, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. 2016. "Neural Combinatorial Optimization with Reinforcement Learning." *arXiv Preprint arXiv:1611.09940*.

Bent, Russell, and Pascal Van Hentenryck. 2004. "Online Stochastic and Robust Optimization." In *Annual Asian Computing Science Conference*, 286–300. Springer.

Bertsimas, Dimitris J, and Garrett Van Ryzin. 1991. "A Stochastic and Dynamic Vehicle Routing Problem in the Euclidean Plane." *Operations Research* 39 (4): 601–15.

Dai, Hanjun, Elias B Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. 2017. "Learning Combinatorial Optimization Algorithms over Graphs." *arXiv Preprint arXiv:1704.01665*.

Dantzig, George B, and John H Ramser. 1959. "The Truck Dispatching Problem." *Management Science* 6 (1): 80–91.

Garrido, Pablo, and María Cristina Riff. 2010. "DVRP: A Hard Dynamic Combinatorial Optimisation Problem Tackled by an Evolutionary Hyper-Heuristic." *Journal of Heuristics* 16 (6): 795–834.

Gendreau, Michel, Gilbert Laporte, and René Séguin. 1996. "Stochastic Vehicle Routing." *European Journal of Operational Research* 88 (1): 3–12.

Ioffe, Sergey, and Christian Szegedy. 2015. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." *arXiv Preprint arXiv:1502.03167*.

Kingma, Diederik P, and Jimmy Ba. 2014. "Adam: A Method for Stochastic Optimization." *arXiv Preprint arXiv:1412.6980*.

Kool, Wouter, Herke van Hoof, and Max Welling. 2019. "Attention, Learn to Solve Routing Problems!" In *International Conference on Learning Representations*. https://openreview.net/forum?id=ByxBFsRqYm.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton. 2012. "Imagenet Classification with Deep Convolutional Neural Networks." In *Advances in Neural Information Processing Systems*, 1097–1105.

Kumar, Suresh Nanda, and Ramasamy Panneerselvam. 2012. "A Survey on the Vehicle Routing Problem and Its Variants." *Intelligent Information Management* 4 (03): 66.

Lipmann, Maarten. 2013. *On-Line Routing*.

Nair, Vinod, and Geoffrey E Hinton. 2010. "Rectified Linear Units Improve Restricted Boltzmann Machines." In *Proceedings of the 27th International Conference on Machine Learning (Icml-10)*, 807–14.

Nazari, MohammadReza, Afshin Oroojlooy, Lawrence Snyder, and Martin Takac. 2018. "Reinforcement Learning for Solving the Vehicle Routing Problem." In *Advances in Neural Information Processing Systems*, 9861–71.

Omme, Nikolaj van, Laurent Perron, and Vincent Furnon. 2014. "Or-Tools User's Manual." Google.

Paszke, Adam, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. "Automatic Differentiation in Pytorch." In *NIPS-W*.

Pillac, Victor, Michel Gendreau, Christelle Guéret, and Andrés L Medaglia. 2013. "A Review of Dynamic Vehicle Routing Problems." *European Journal of Operational Research* 225 (1): 1–11.

Silver, David, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, et al. 2016. "Mastering the Game of Go with Deep Neural Networks and Tree Search." *Nature* 529 (7587): 484.

Vinyals, Oriol, Meire Fortunato, and Navdeep Jaitly. 2015. "Pointer Networks." In *Advances in Neural Information Processing Systems*, 2692–2700.