Reinforcement Learning for Solving the Vehicle Routing Problem

Mohammadreza Nazari
Department of Industrial Engineering
Lehigh University
Bethlehem, PA 18015
mon314@lehigh.edu

Afshin Oroojlooy
Department of Industrial Engineering
Lehigh University
Bethlehem, PA 18015
afo214@lehigh.edu

Martin Takáč
Department of Industrial Engineering
Lehigh University
Bethlehem, PA 18015
takac@lehigh.edu

Lawrence V. Snyder
Department of Industrial Engineering
Lehigh University
Bethlehem, PA 18015
lvs2@lehigh.edu

Abstract

We present an end-to-end framework for solving the Vehicle Routing Problem (VRP) using reinforcement learning. In this approach, we train a single model that finds near-optimal solutions for problem instances sampled from a given distribution, only by observing the reward signals and following feasibility rules. Our model represents a parameterized stochastic policy, and by applying a policy gradient algorithm to optimize its parameters, the trained model produces the solution as a sequence of consecutive actions in real time, without the need to re-train for every new problem instance. On capacitated VRP, our approach outperforms classical heuristics and Google's OR-Tools on medium-sized instances in solution quality with comparable computation time (after training). We demonstrate how our approach can handle problems with split delivery and explore the effect of such deliveries on the solution quality. Our proposed framework can be applied to other variants of the VRP such as the stochastic VRP, and has the potential to be applied more generally to combinatorial optimization problems.

1 Introduction

The *Vehicle Routing Problem* (VRP) is a combinatorial optimization problem that has been studied in applied mathematics and computer science for decades. VRP is known to be a computationally difficult problem for which many exact and heuristic algorithms have been proposed, but providing fast and reliable solutions is still a challenging task. In the simplest form of the VRP, a single capacitated vehicle is responsible for delivering items to multiple customer nodes; the vehicle must return to the depot to pick up additional items when it runs out. The objective is to optimize a set of routes, all beginning and ending at a given node, called the *depot*, in order to attain the maximum possible reward, which is often the negative of the total vehicle distance or average service time. This problem is computationally difficult to solve to optimality, even with only a few hundred customer nodes [12]. For an overview of the VRP, see, for example, [15, 22, 23, 31].

The prospect of new algorithm discovery, without any hand-engineered reasoning, makes neural networks and reinforcement learning a compelling choice that has the potential to be an important milestone on the path toward approaching these problems. In this work, we develop a framework with the capability of solving a wide variety of combinatorial optimization problems using *Reinforcement*

Learning (RL) and show how it can be applied to solve the VRP. For this purpose, we consider the Markov Decision Process (MDP) formulation of the problem, in which the optimal solution can be viewed as a sequence of decisions. This allows us to use RL to produce near-optimal solutions by increasing the probability of decoding "desirable" sequences. A naive approach is to find a problem-specific solution by considering every instance separately. Obviously, this approach is not practical in terms of either solution quality or runtime since there should be many trajectories sampled from one MDP to be able to produce a near-optimal solution. Moreover, the learned policy does not apply to instances other than the one that was used in the training; with a small perturbation of the problem setting, we need to rebuild the policy from scratch.

Therefore, rather than focusing on training a separate model for every problem instance, we propose a structure that performs well on any problem sampled from a given distribution. This means that if we generate a new VRP instance with the same number of nodes and vehicle capacity, and the same location and demand distributions as the ones that we used during training, then the trained policy will work well, and we can solve the problem right away, without retraining for every new instance. As long as we approximate the generating distribution of the problem, the framework can be applied. One can view the trained model as a black-box heuristic (or a meta-algorithm) which generates a high-quality solution in a reasonable amount of time.

This study is motivated by the recent work by Bello et al. [4]. We have generalized their framework to include a wider range of combinatorial optimization problems such as the VRP. Bello et al. [4] propose the use of a Pointer Network [32] to decode the solution. One major issue that prohibits the direct use of their approach for the VRP is that it assumes the system is static over time. In contrast, in the VRP, the demands change over time in the sense that once a node has been visited its demand becomes, effectively, zero. To overcome this, we propose an alternate approach—which is actually simpler than the Pointer Network approach—that can efficiently handle both the static and dynamic elements of the system. Our model consists of a recurrent neural network (RNN) decoder coupled with an attention mechanism. At each time step, the embeddings of the static elements are the input to the RNN decoder, and the output of the RNN and the dynamic element embeddings are fed into an attention mechanism, which forms a distribution over the feasible destinations that can be chosen at the next decision point.

The proposed framework is appealing since we utilize a self-driven learning procedure that only requires the reward calculation based on the generated outputs; as long as we can observe the reward and verify the feasibility of a generated sequence, we can learn the desired meta-algorithm. For instance, if one does not know how to solve the VRP but can compute the cost of a given solution, then one can provide the signal required for solving the problem using our method. Unlike most classical heuristic methods, it is robust to problem changes, meaning that when the inputs change in any way, it can automatically adapt the solution. Using classical heuristics for VRP, the entire distance matrix must be recalculated and the system must be re-optimized from scratch, which is often impractical, especially if the problem size is large. In contrast, our proposed framework does not require an explicit distance matrix, and only one feed-forward pass of the network will update the routes based on the new data.

Our numerical experiments indicate that our framework performs significantly better than well-known classical heuristics designed for the VRP, and that it is robust in the sense that its worst results are still relatively close to optimal. Comparing our method with the OR-Tools VRP engine [16], which is ones of the best open-source VRP solvers, we observe a noticeable improvement; in VRP instances with 50 and 100 customers, our method provides shorter tours in roughly 61% of the instances. Another interesting observation that we make in this study is that by allowing multiple vehicles to supply the demand of a single node, our RL-based framework finds policies that outperform the solutions that require single deliveries. We obtain this appealing property, known as the split delivery, without any hand engineering and no extra cost.

2   Background

Before presenting the model, we briefly review some background that is closely related to our work.

Sequence-to-Sequence Models   *Sequence-to-Sequence* models [30, 32, 24] are useful in tasks for which a mapping from one sequence to another is required. They have been extensively studied in

the field of neural machine translation over the past several years, and there are numerous variants of these models. The general architecture, which is almost the same among different versions, consists of two RNN networks, called the encoder and decoder. An encoder network reads through the input sequence and stores the knowledge in a fixed-size vector representation (or a sequence of vectors); then, a decoder converts the encoded information back to an output sequence.

In the vanilla Sequence-to-Sequence architecture [30], the source sequence appears only once in the encoder and the entire output sequence is generated based on one vector (i.e., the last hidden state of the encoder RNN). Other extensions, for example Bahdanau et al. [3], illustrate that the source information can be used more wisely to increase the amount of information during the decoding steps. In addition to the encoder and decoder networks, they employ another neural network, namely an *attention mechanism* that *attends* to the entire encoder RNN states. This mechanism allows the decoder to focus on the important locations of the source sequence and use the relevant information during decoding steps for producing "better" output sequences. Recently, the concept of attention has been a popular research idea due to its capability to align different objects, e.g., in computer vision [6, 37, 38, 18] and neural machine translation [3, 19, 24]. In this study, we also employ a special attention structure for policy representation. See Section 3.3 for a detailed discussion of the attention mechanism.

Neural Combinatorial Optimization   Over the last several years, multiple methods have been developed to tackle combinatorial optimization problems by using recent advances in artificial intelligence. The first attempt was proposed by Vinyals et al. [32], who introduce the concept of a *Pointer Network*, a model originally inspired by sequence-to-sequence models. Because it is invariant to the length of the encoder sequence, the Pointer Network enables the model to apply to combinatorial optimization problems, where the output sequence length is determined by the source sequence. They use the Pointer Network architecture in a supervised fashion to find near-optimal Traveling Salesman Problem (TSP) tours from ground truth optimal (or heuristic) solutions. This dependence on supervision prohibits the Pointer Network from finding better solutions than the ones provided during the training.

Closest to our approach, Bello et al. [4] address this issue by developing a neural combinatorial optimization framework that uses RL to optimize a policy modeled by a Pointer Network. Using several classical combinatorial optimization problems such as TSP and the knapsack problem, they show the effectiveness and generality of their architecture.

On a related topic, Dai et al. [11] solve optimization problems over graphs using a graph embedding structure [10] and a deep Q-learning (DQN) algorithm [25]. Even though VRP can be represented by a graph with weighted nodes and edges, their proposed model does not directly apply since in VRP, a particular node (e.g. the depot) might be visited multiple times.

Next, we introduce our model, which is a simplified version of the Pointer Network.

3   The Model

In this section, we formally define the problem and our proposed framework for a generic combinatorial optimization problem with a given set of inputs $X \doteq \{x^i, i = 1, \cdots, M\}$. We allow some of the elements of each input to change between the decoding steps, which is, in fact, the case in many problems such as the VRP. The dynamic elements might be an artifact of the decoding procedure itself, or they can be imposed by the environment. For example, in the VRP, the remaining customer demands change over time as the vehicle visits the customer nodes; or we might consider a variant in which new customers arrive or adjust their demand values over time, independent of the vehicle decisions. Formally, we represent each input x^i by a sequence of tuples $\{x_t^i \doteq (s^i, d_t^i), t = 0, 1, \cdots\}$, where s^i and d_t^i are the static and dynamic elements of the input, respectively, and can themselves be tuples. One can view x_t^i as a vector of features that describes the state of input i at time t. For instance, in the VRP, x_t^i gives a snapshot of the customer i, where s^i corresponds to the 2-dimensional coordinate of customer i's location and d_t^i is its demand at time t. We will denote the set of all input states at a fixed time t with X_t.

We start from an arbitrary input in X_0, where we use the pointer y_0 to refer to that input. At every decoding time t ($t = 0, 1, \cdots$), y_{t+1} points to one of the available inputs X_t, which determines the input of the next decoder step; this process continues until a termination condition is satisfied.

The termination condition is problem-specific, showing that the generated sequence satisfies the feasibility constraints. For instance, in the VRP that we consider in this work, the terminating condition is that there is no more demand to satisfy. This process will generate a sequence of length T, $Y = \{y_t, t = 0, ..., T\}$, possibly with a different sequence length compared to the input length M. This is due to the fact that, for example, the vehicle may have to go back to the depot several times to refill. We also use the notation Y_t to denote the decoded sequence up to time t, i.e., $Y_t = \{y_0, \cdots, y_t\}$. We are interested in finding a stochastic policy π which generates the sequence Y in a way that minimizes a loss objective while satisfying the problem constraints. The optimal policy π^* will generate the optimal solution with probability 1. Our goal is to make π as close to π^* as possible. Similar to Sutskever et al. [30], we use the probability chain rule to decompose the probability of generating sequence Y, i.e., $P(Y|X_0)$, as follows:

$$P(Y|X_0) = \prod_{t=0}^{T} P(y_{t+1}|Y_t, X_t), \tag{1}$$

and

$$X_{t+1} = f(y_{t+1}, X_t) \tag{2}$$

is a recursive update of the problem representation with the state transition function f. Each component in the right-hand side of (1) is computed by the attention mechanism, i.e.,

$$P(y_{t+1}|Y_t, X_t) = \text{softmax}(g(h_t, X_t)), \tag{3}$$

where g is an affine function that outputs an input-sized vector, and h_t is the state of the RNN decoder that summarizes the information of previously decoded steps $y_0, \cdots, y_t$. We will describe the details of our proposed attention mechanism in Section 3.3.

Remark 1: This model can handle combinatorial optimization problems in both a more classical static setting as well as in dynamically changing ones. In static combinatorial optimization, X_0 fully defines the problem that we are trying to solve. For example, in the VRP, X_0 includes all customer locations as well as their demands, and the depot location; then, the remaining demands are updated with respect to the vehicle destination and its load. With this consideration, often there exists a well-defined Markovian transition function f, as defined in (2), which is sufficient to update the dynamics between decision points. However, our model can also be applied to problems in which the state transition function is unknown and/or is subject to external noise, since the training does not explicitly make use of the transition function. However, knowing this transition function helps in simulating the environment that the training algorithm interacts with. See Appendix C.6 for an example of how to apply the model to a stochastic version of the VRP in which random customers with random demands appear over time.

3.1 Limitations of Pointer Networks

Although the framework proposed by Bello et al. [4] works well on problems such as the knapsack problem and TSP, it is not applicable to more complicated combinatorial optimization problems in which the system representation varies over time, such as VRP. Bello et al. [4] feed a random sequence of inputs to the RNN encoder. Figure 1 illustrates with an example why using the RNN in the encoder is restrictive. Suppose that at the first decision step, the policy sends the vehicle to customer 1, and as a result, its demand is satisfied, i.e., $d_0^1 \neq d_1^1$. Then in the second decision step, we need to re-calculate the whole network with the new d_1^1 information in order to choose the next customer. The dynamic elements complicate the forward pass of the network since there should be encoder/decoder updates when an input changes. The situation is even worse during back-propagation to accumulate the gradients since we need to remember when the dynamic elements changed. In order to resolve this complication, we require the model to be *invariant to the input sequence* so that changing the order of any two inputs does not affect the network. In Section 3.2, we present a simple network that satisfies this property.

3.2 The Proposed Neural Network Model

We argue that the RNN encoder adds extra complication to the encoder but is actually not necessary, and the approach can be made much more general by omitting it. RNNs are necessary only when
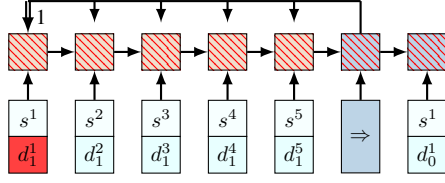
Figure 1: Limitation of the Pointer Network. After a change in dynamic elements (d_1^1 in this example), the whole Pointer Network must be updated to compute the probabilities in the next decision point.
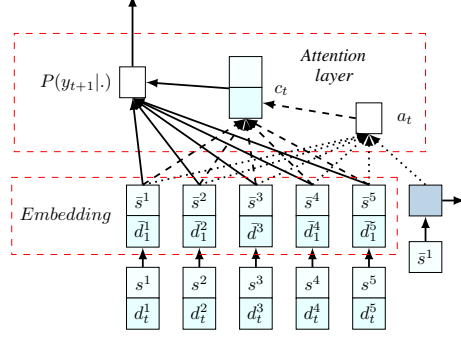

Figure 2: Our proposed model. The embedding layer maps the inputs to a high-dimensional vector space. On the right, an RNN decoder stores the information of the decoded sequence. Then, the RNN hidden state and embedded input produce a probability distribution over the next input using the attention mechanism.

the inputs convey sequential information; e.g., in text translation the combination of words and their relative position must be captured in order for the translation to be accurate. But the question here is, *why do we need to have them in the encoder for combinatorial optimization problems when there is no meaningful order in the input set?* As an example, in the VRP, the inputs are the set of unordered customer locations with their respective demands, and their order is not meaningful; any random permutation contains the same information as the original inputs. Therefore, in our model, we simply leave out the encoder RNN and directly use the embedded inputs instead of the RNN hidden states. By this modification, many of the computational complications disappear, without decreasing the model's efficiency. In Appendix A, we provide an experiment to verify this claim.

As illustrated in Figure 2, our model is composed of two main components. The first is a set of embeddings that maps the inputs into a D-dimensional vector space. We might have multiple embeddings corresponding to different elements of the input, but they are shared among the inputs. The second component of our model is a decoder that points to an input at every decoding step. As is common in the literature [3, 30, 7], we use RNN to model the decoder network. Notice that we feed the static elements as the inputs to the decoder network. The dynamic element can also be an input to the decoder, but our experiments on the VRP do not suggest any improvement by doing so, so dynamic elements are used only in the attention layer, described next.

3.3 Attention Mechanism

An attention mechanism is a differentiable structure for addressing different parts of the input. Figure 2 illustrates the attention mechanism employed in our method. At decoder step i, we utilize a context-based attention mechanism with glimpse, similar to Vinyals et al. [33], which extracts the relevant information from the inputs using a variable-length alignment vector a_t. In words, a_t specifies how much every input data point might be relevant in the next decoding step t.

Let $\bar{x}_t^i = (\bar{s}^i, \bar{d}_t^i)$ be the embedded input i, and $h_t \in \mathbb{R}^D$ be the memory state of the RNN cell at decoding step t. The alignment vector a_t is then computed as

$$a_t = a_t(\bar{x}_t^i, h_t) = \text{softmax}\,(u_t)\,, \quad \text{where } u_t^i = v_a^T \tanh\left(W_a[\bar{x}_t^i; h_t]\right). \tag{4}$$

Here ";" means the concatenation of two vectors. We compute the conditional probabilities by combining the context vector c_t, computed as

$$c_t = \sum_{i=1}^{M} a_t^i \bar{x}_t^i, \tag{5}$$

with the embedded inputs, and then normalizing the values with the softmax function, as follows:

$$P(y_{t+1}|Y_t, X_t) = \text{softmax}(\tilde{u}_t^i), \quad \text{where } \tilde{u}_t^i = v_c^T \tanh\left(W_c[\bar{x}_t^i; c_t]\right). \tag{6}$$

5

In (4)–(6), v_a, v_c, W_a and W_c are trainable variables.

Remark 2: *Model Symmetry*: Vinyals et al. [33] discuss an extension of sequence-to-sequence models where they empirically demonstrate that in tasks with no obvious input sequence, such as sorting, the order in which the inputs are fed into the network matter. A similar concern arises when using Pointer Networks for combinatorial optimization problems. However, the model proposed in this paper does not suffer from such a complication since the embeddings and the attention mechanism are invariant to the input order.

3.4 Training Method

To train the network, we use well-known policy gradient approaches. To use these methods, we parameterize the stochastic policy π with parameters θ. Policy gradient methods use an estimate of the gradient of the expected return with respect to the policy parameters to iteratively improve the policy. In principle, the policy gradient algorithm contains two networks: (*i*) an actor network that predicts a probability distribution over the next action at any given decision step, and (*ii*) a critic network that estimates the reward for any problem instance from a given state. Our training methods are quite standard, and due to space limitation we leave the details to the Appendix.

4 Computational Experiment

Many variants of the VRP have been extensively studied in the operations research literature. (See, for example, the reviews by Laporte [22], Laporte et al. [23], or the book by Toth and Vigo [31] for different variants of the problem.) In this section, we consider a specific capacitated version of the problem in which one vehicle with a limited capacity is responsible for delivering items to many geographically distributed customers with finite demands. When the vehicle's load runs out, it returns to the depot to refill. We will denote the vehicle's remaining load at time t as l_t. The objective is to minimize the total route length while satisfying all of the customer demands. This problem is often called the capacitated VRP (CVRP) to distinguish it from other variants, but we will refer to it simply as the VRP.

We assume that the node locations and demands are randomly generated from a fixed distribution. Specifically, the customers and depot locations are randomly generated in the unit square $[0, 1] \times [0, 1]$. For simplicity of exposition, we assume that the demand of each node is a discrete number in $\{1, .., 9\}$, chosen uniformly at random. We note, however, that the demand values can be generated from any distribution, including continuous ones.

We assume that the vehicle is located at the depot at time 0, so the first input to the decoder is an embedding of the depot location. At each decoding step, the vehicle chooses from among the customer nodes or the depot to visit in the next step. After visiting customer node i, the demands and vehicle load are updated as follows:

$$d_{t+1}^i = \max(0, d_t^i - l_t), \quad d_{t+1}^k = d_t^k \text{ for } k \neq i, \text{ and } \quad l_{t+1} = \max(0, l_t - d_t^i) \tag{7}$$

which is an explicit definition of the state transition function (2) for the VRP.

In this experiment, we have employed two different decoders: *(i)* greedy, in which at every decoding step, the node (either customer or depot) with the highest probability is selected as the next destination, and *(ii)* beam search (BS), which keeps track of the most probable paths and then chooses the one with the minimum tour length [27]. Our results indicate that by applying the beam search algorithm, the quality of the solutions can be improved with only a slight increase in computation time.

For faster training and generating feasible solutions, we have used a *masking scheme* which sets the log-probabilities of infeasible solutions to $-\infty$ or forces a solution if a particular condition is satisfied. In the VRP, we use the following masking procedures: *(i)* nodes with zero demand are not allowed to be visited; *(ii)* all customer nodes will be masked if the vehicle's remaining load is exactly 0; and *(iii)* the customers whose demands are greater than the current vehicle load are masked. Notice that under this masking scheme, the vehicle must satisfy all of a customer's demands when visiting it. We note, however, that if the situation being modeled does allow split deliveries, one can relax *(iii)*. Indeed, the relaxed masking allows split deliveries, so the solution can allocate the demands of a given customer into multiple routes. This property is, in fact, an appealing behavior that is present in many real-world applications but is often neglected in classical VRP algorithms. In

all the experiments of the next section, we do not allow to split demands. Further investigation and illustrations of this property is included in Appendix C.2–C.4.

4.1 Results

In this section, we compare the solutions found using our framework with those obtained from the *Clarke-Wright savings heuristic* (CW), the *Sweep heuristic* (SW), and Google's optimization tools (OR-Tools). We run our test on multiple problem sizes with different vehicle capacities; for example, VRP10 consists of 10 customer. The results are based on 1000 instances, sampled for each problem size.

(a) Comparison for VRP10

(b) Comparison for VRP20

	RL-Greedy	RL-BS(5)	RL-BS(10)	CW-Greedy	CW-Rnd(5,5)	CW-Rnd(10,10)	SW-Basic	SW-Rnd(5)	SW-Rnd(10)	OR-Tools
RL-Greedy		12.2	7.2	99.4	97.2	96.3	97.9	97.9	97.9	41.5
RL-BS(5)	85.8		12.5	99.7	99.0	98.7	99.1	99.1	99.1	54.6
RL-BS(10)	91.9	57.7		99.8	99.4	99.2	99.3	99.3	99.3	60.2
CW-Greedy	0.6	0.3	0.2		0.0	0.0	68.9	68.9	68.9	1.0
CW-Rnd(5,5)	2.8	1.0	0.6	92.2		30.4	84.5	84.5	84.5	3.5
CW-Rnd(10,10)	3.7	1.3	0.8	97.5	68.0		86.8	86.8	86.8	4.7
SW-Basic	2.1	0.9	0.7	31.1	15.5	13.2		0.0	0.0	1.4
SW-Rnd(5)	2.1	0.9	0.7	31.1	15.5	13.2	0.0		0.0	1.4
SW-Rnd(10)	2.1	0.9	0.7	31.1	15.5	13.2	0.0	0.0		1.4
OR-Tools	58.5	45.4	39.8	99.0	96.5	95.3	98.6	98.6	98.6	

(c) Comparison for VRP50

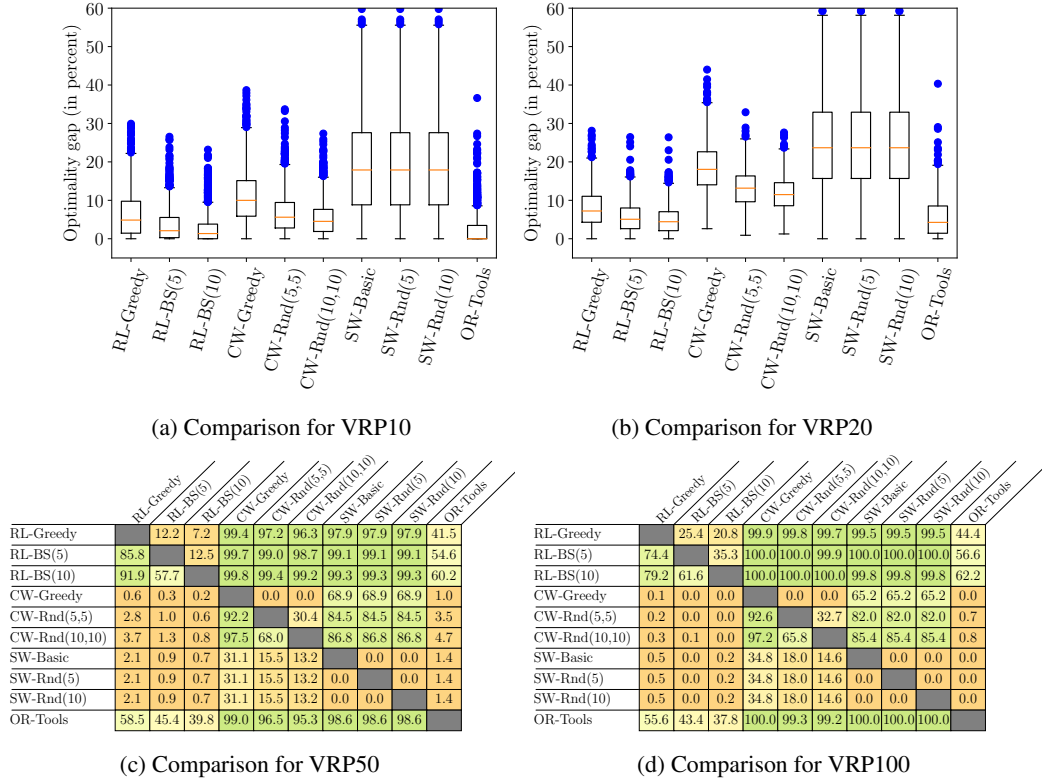|	RL-Greedy	RL-BS(5)	RL-BS(10)	CW-Greedy	CW-Rnd(5,5)	CW-Rnd(10,10)	SW-Basic	SW-Rnd(5)	SW-Rnd(10)	OR-Tools
RL-Greedy		25.4	20.8	99.9	99.8	99.7	99.5	99.5	99.5	44.4
RL-BS(5)	74.4		35.3	100.0	100.0	99.9	100.0	100.0	100.0	56.6
RL-BS(10)	79.2	61.6		100.0	100.0	100.0	99.8	99.8	99.8	62.2
CW-Greedy	0.1	0.0	0.0		0.0	0.0	65.2	65.2	65.2	0.7
CW-Rnd(5,5)	0.2	0.0	0.0	92.6		32.7	82.0	82.0	82.0	0.7
CW-Rnd(10,10)	0.3	0.1	0.0	97.2	65.8		85.4	85.4	85.4	0.8
SW-Basic	0.5	0.0	0.2	34.8	18.0	14.6		0.0	0.0	0.0
SW-Rnd(5)	0.5	0.0	0.2	34.8	18.0	14.6	0.0		0.0	0.0
SW-Rnd(10)	0.5	0.0	0.2	34.8	18.0	14.6	0.0	0.0		0.0
OR-Tools	55.6	43.4	37.8	100.0	99.3	99.2	100.0	100.0	100.0	

(d) Comparison for VRP100

Figure 3: Parts 3a and 3b show the optimality gap (in percent) using different algorithms/solvers for VRP10 and VRP20. Parts 3c and 3d give the proportion of the samples for which the algorithms in the rows outperform those in the columns; for example, RL-BS(5) is superior to RL-greedy in 85.8% of the VRP50 instances.

Figure 3 shows the distribution of total tour lengths generated by our method, using greedy and BS decoders, with the number inside the parentheses indicating the beam-width parameter. In the experiments, we label our method with the "RL" prefix. In addition, we also implemented a randomized version of both heuristic algorithms to improve the solution quality; for Clarke-Wright, the numbers inside the parentheses are the randomization depth and randomization iterations parameters; and for Sweep, it is the number of random initial angles for grouping the nodes. Finally, we use Google's OR-Tools [16], which is a more competitive baseline. See Appendix B for a detailed discussion on the baselines.

For small problems of VRP10 and VRP20, it is possible to find the optimal solution, which we do by solving a mixed integer formulation of the VRP [31]. Figures 3a and 3b measure how far the solutions are far from optimality. The optimality gap is defined as the distance from the optimal objective value normalized by the latter. We observe that using a beam width of 10 is the best-performing method; roughly 95% of the instances are at most 10% away from optimality for VRP10 and 13% for VRP20. Even the outliers are within 20–25% of optimality, suggesting that our RL-BS methods are robust in comparison to the other baseline approaches.

Since obtaining the optimal objective values for VRP50 and VRP100 is not computationally affordable, in Figures 3d and 3d, we compare the algorithms in terms of their winning rate. Each table gives the percentage of the instances in which the algorithms in the rows outperform those in the columns. In other words, the cell corresponding to (A,B) shows the percentage of the samples in which algorithm A provides shorter tours than B. We observe that the classical heuristics are outperformed by the other approaches in almost 100% of the samples. Moreover, RL-greedy is comparable with OR-Tools, but incorporating beam search into our framework increases the winning rate of our approach to above 60%.

Figure 4 shows the log of the ratio of solution times to the number of customer nodes. We observe that this ratio stays almost the same for RL with different decoders. In contrast, the log of the run time for the Clarke-Wright and Sweep heuristics increases faster than linearly with the number of nodes. This observation is one motivation for applying our framework to more general combinatorial problems, since it suggests that our method scales well. Even though the greedy Clark-Wright and basic Sweep heuristics are fast for small instances, they do not provide competitive solutions. Moreover, for larger problems, our framework is faster than the randomized heuristics. We also include the solution times for OR-Tools in the graph, but we should note that OR-Tools is implemented in C++, which makes exact time comparisons impossible since the other baselines were implemented in Python.
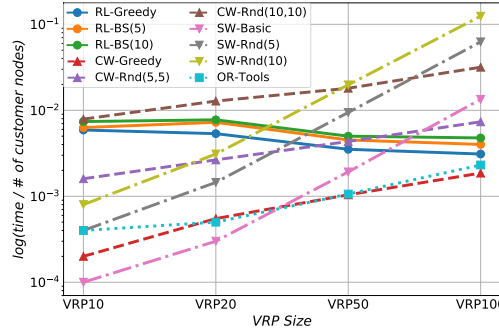

Figure 4: Log of ratio of solution time to the number of customer nodes using different algorithms.

4.2 Extension to Other VRPs

The proposed framework can be extended easily to problems with multiple depots; one only needs to construct the corresponding state transition function and masking procedure. It is also possible to include various side constraints: soft constraints can be applied by penalizing the rewards, or hard constraints such as time windows can be enforced through a masking scheme. However, designing such a scheme might be a challenging task, possibly harder than solving the optimization problem itself. Another interesting extension is for VRPs with multiple vehicles. In the simplest case in which the vehicles travel independently, one must only design a shared masking scheme to avoid the vehicles pointing to the same customer nodes. Incorporating competition or collaboration among the vehicles is also an interesting line of research that relates to multi-agent RL (MARL) [5].

This framework can also be applied to real-time services including on-demand deliveries and taxis. In Appendix C.6, we design an experiment to illustrate the performance of the algorithm on a VRP where both customer locations and their demands are subject to change. Our results indicate superior performance than the baselines.

5 Discussion and Conclusion

We expect that the proposed architecture has significant potential to be used in real-world problems with further improvements. Noting that the proposed algorithm is not limited to VRP, it will be an important topic of future research to apply it to other combinatorial optimization problems such as bin-packing, job-shop, and flow-shop.

This method is quite appealing since the only requirement is a verifier to find feasible solutions and also a reward signal to demonstrate how well the policy is working. Once the trained model

is available, it can be used many times, without needing to re-train for the new problems as long as they are generated from the training distribution. Unlike many classical heuristics, our proposed method scales well with increasing problem size, and has a superior performance with competitive solution-time. It doesn't require a distance matrix calculation which might be computationally cumbersome, especially in dynamically changing VRPs. We also illustrate the performance of the algorithm on a much more complicated stochastic version of the VRP.

Acknowledgment

References

[1] David L Applegate, Robert E Bixby, Vasek Chvatal, and William J Cook. *The traveling salesman problem: a computational study*. Princeton university press, 2006.

[2] Claudia Archetti and Maria Grazia Speranza. The split delivery vehicle routing problem: a survey. In *The vehicle routing problem: Latest advances and new challenges*, pages 103–122. Springer, 2008.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *In International Conference on Learning Representations*, 2015.

[4] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

[5] Lucian Buşoniu, Robert Babuška, and Bart De Schutter. Multi-agent reinforcement learning: An overview. In *Innovations in multi-agent systems and applications-1*, pages 183–221. Springer, 2010.

[6] Kan Chen, Jiang Wang, Liang-Chieh Chen, Haoyuan Gao, Wei Xu, and Ram Nevatia. Abc-cnn: An attention based convolutional neural network for visual question answering. *arXiv preprint arXiv:1511.05960*, 2015.

[7] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *Conference on Empirical Methods in Natural Language Processing*, 2014.

[8] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.

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

[10] Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *International Conference on Machine Learning*, pages 2702–2711, 2016.

[11] Hanjun Dai, Elias B Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in Neural Information Processing Systems*, 2017.

[12] Ricardo Fukasawa, Humberto Longo, Jens Lysgaard, Marcus Poggi de Aragão, Marcelo Reis, Eduardo Uchoa, and Renato F Werneck. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical programming*, 106(3):491–511, 2006.

[13] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[14] Fred Glover and Manuel Laguna. Tabu search*. In *Handbook of combinatorial optimization*, pages 3261–3362. Springer, 2013.

[15] Bruce L Golden, Subramanian Raghavan, and Edward A Wasil. *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43. Springer Science & Business Media, 2008.

[16] Inc. Google. Google's optimization tools (or-tools), 2018. URL https://github.com/google/or-tools.

[17] Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2016. URL http://www.gurobi.com.

[18] Seunghoon Hong, Junhyuk Oh, Honglak Lee, and Bohyung Han. Learning transferrable knowledge for semantic segmentation with deep convolutional neural network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3204–3212, 2016.

[19] Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. On using very large target vocabulary for neural machine translation. 2015.

[20] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Machine Learning*, 2015.

[21] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[22] Gilbert Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European journal of operational research*, 59(3):345–358, 1992.

[23] Gilbert Laporte, Michel Gendreau, Jean-Yves Potvin, and Frédéric Semet. Classical and modern heuristics for the vehicle routing problem. *International transactions in operational research*, 7 (4-5):285–300, 2000.

[24] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *Conference on Empirical Methods in Natural Language Processing*, 2015.

[25] 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.

[26] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[27] Graham Neubig. Neural machine translation and sequence-to-sequence models: A tutorial. *arXiv preprint arXiv:1703.01619*, 2017.

[28] Ulrike Ritzinger, Jakob Puchinger, and Richard F Hartl. A survey on dynamic and stochastic vehicle routing problems. *International Journal of Production Research*, 54(1):215–231, 2016.

[29] Lawrence V Snyder and Zuo-Jun Max Shen. *Fundamentals of Supply Chain Theory*. John Wiley & Sons, 2nd edition, 2018.

[30] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[31] Paolo Toth and Daniele Vigo. *The Vehicle Routing Problem*. SIAM, 2002.

[32] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015.

[33] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. 2016.

[34] Christos Voudouris and Edward Tsang. Guided local search and its application to the traveling salesman problem. *European journal of operational research*, 113(2):469–499, 1999.

[35] Ronald J Williams and Jing Peng. Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 3(3):241–268, 1991.

[36] Anthony Wren and Alan Holliday. Computer scheduling of vehicles from one or more depots to a number of delivery points. *Operational Research Quarterly*, pages 333–344, 1972.

[37] Tianjun Xiao, Yichong Xu, Kuiyuan Yang, Jiaxing Zhang, Yuxin Peng, and Zheng Zhang. The application of two-level attention models in deep convolutional neural network for fine-grained image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 842–850, 2015.

[38] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*, pages 2048–2057, 2015.

A Our Model versus Pointer Network

In this section we use the *Traveling Salesman Problem* (TSP) (a special case of the VRP in which there is only a single route to optimize) as the test-bed to validate the performance of the proposed method. We compare the route lengths of the TSP solutions obtained by our framework with those given by the model of Bello et al. [4] for random instances with 20, 50, and 100 nodes. In the training phase, we generate 10^6 TSP instances for each problem size, and use them in training for 20 epochs. 10^6 is chosen because we want to have a diverse set of problem configurations; it can be larger or smaller, or we can generate the instances on-the-fly as long as we make sure that the instance are drawn from the same probability distribution with the same random seed. The city locations are chosen uniformly from the unit square $[0, 1] \times [0, 1]$. We use the same data distribution to produce instances for the testing phase. The decoding process starts from a random TSP node and the termination criterion is that all cities are visited. We also use a masking scheme to prohibit visiting nodes more than once.

Table 1 summarizes the results for different TSP sizes using the *greedy decoder* in which at every decoding step, the city with the highest probability is chosen as the destination. The results are averaged over 1000 instances. The first column is the average TSP tour length using our proposed architecture, the second column is the result of our implementation of Bello et al. [4] with greedy decoder, and the optimal tour lengths are reported in the last column. To obtain the optimal values, we solved the TSP using the Concorde optimization software [1]. A comparison of the first two columns suggests that there is almost no difference between the performance of our framework and Pointer-RL. In fact, the RNN encoder of the Pointer Network learns to convey no information to the next steps, i.e., $h_t = f(x_t)$. On the other hand, our approach is around 60% faster in both training and inference, since it has two fewer RNNs—one in the encoder of actor network and another in the encoder of critic network. Table 1 also summarizes the training times for one epoch of the training and the time-savings that we gain by eliminating the encoder RNNs.

Table 1: Average tour length for TSP and training time for one epoch (in minutes).

| Task | Average tour length | | | Training time | | |
	Our Framework (Greedy)	Pointer-RL (Greedy)	Optimal	Our Framework (Greedy)	Pointer-RL (Greedy)	% Time Saving
TSP20	3.97	3.96	3.84	22.18	50.33	55.9%
TSP50	6.08	6.05	5.70	54.10	147.25	63.3%
TSP100	8.44	8.45	7.77	122.10	300.73	59.4%

B Capacitated VRP Baselines

In this Appendix, we briefly describe the algorithms and solvers that we used as benchmarks. More details and examples of these algorithms can be found in Snyder and Shen [29]. The first two baseline approaches are well-known heuristics designed for VRP. Our third baseline is Google's optimization tools, which includes one of the best open-source VRP solvers. Finally, we compute the optimal solutions for small VRP instances, so we can measure how far the solutions are from optimality.

B.1 Clarke-Wright Savings Heuristic

The Clarke-Wright savings heuristic [9] is one of the best-known heuristics for the VRP. Let $\mathcal{N} \doteq \{1, \cdots, N\}$ be the set of customer nodes, and 0 be the depot. The distance between nodes i and j is denoted by c_{ij}, and c_{0i} is the distance of customer i from the depot. Algorithm 1 describes a randomized version of the heuristic. The basic idea behind this algorithm is that it initially considers a separate route for each customer node i, and then reduces the total cost by iteratively merging the routes. Merging two routes by adding the edge (i, j) reduces the total distance by $s_{ij} = c_{i0} + c_{0j} - c_{ij}$, so the algorithm prefers mergers with the highest savings s_{ij}.

We introduce two hyper-parameters, R and M, which we refer to as the *randomization depth* and *randomization iteration*, respectively. When $M = R = 1$, this algorithm is equivalent to the original Clarke-Wright savings heuristic, in which case, the feasible merger with the highest savings will be selected. By allowing $M, R > 1$, we introduce randomization, which can improve the performance

of the algorithm further. In particular, Algorithm 1 chooses randomly from the $r \in \{1, \cdots, R\}$ best feasible mergers. Then, for each r, it solves the problem $m \in \{1, \cdots, M\}$ times, and returns the solution with the shortest total distance.

Algorithm 1 Randomized Clarke-Wright Savings Heuristic

1: compute savings s_{ij}, where
$$s_{ij} = c_{i0} + c_{0j} - c_{ij} \qquad\qquad i, j \in \mathcal{N}, \ i \neq j$$
$$s_{ii} = 0 \qquad\qquad\qquad\qquad i \in \mathcal{N}$$
2: **for** $r = 1, \cdots, R$ **do**
3:     **for** $m = 1, \cdots, M$ **do**
4:         place each $i \in \mathcal{N}$ in its own route
5:         **repeat**
6:             find k feasible mergers (i, j) with the highest $s_{ij} > 0$, satisfying the following conditions:
                i) i and j are in different routes
                ii) both i and j are adjacent to the depot
                iii) combined demand of routes containing i and j is $\leq$ vehicle capacity
7:             choose a random (i, j) from the feasible mergers, and combine the associated routes by replacing $(i, 0)$ and $(0, j)$ with (i, j)
8:         **until** no feasible merger is left
9:     **end for**
10: **end for**
11: **Return**: route with the shortest length

B.2 Sweep Heuristic

The sweep heuristic [36] solves the VRP by breaking it into multiple TSPs. By rotating an arc emanating from the depot, it groups the nodes into several clusters, while ensuring that the total demand of each cluster is not violating the vehicle capacity. Each cluster corresponds to a TSP that can be solved by using an exact or approximate algorithm. In our experiments, we use dynamic programming to find the optimal TSP tour. After solving TSPs, the VRP solution can be obtained by combining the TSP tours. Algorithm 2 shows the pseudo-code of this algorithm.

Algorithm 2 Randomized Sweep Algorithm

1: for each $i \in \mathcal{N}$, compute angle α_i, respective to depot location
2: $l \leftarrow$ vehicle capacity
3: **for** $r = 1, \cdots, R$ **do**
4:     select a random angle α
5:     $k \leftarrow 0$; initialize cluster $S_k \leftarrow \emptyset$
6:     **repeat**
7:         increase α until it equal to some α_i
8:         **if** demand $d_i > l$ **then**
9:             $k \leftarrow k + 1$
10:             $S_k \leftarrow \emptyset$
11:             $l \leftarrow$ vehicle capacity
12:         **end if**
13:         $S_k \leftarrow S_k \cup \{i\}$
14:         $l \leftarrow l - d_i$
15:     **until** no unclustered node is left
16:     solve a TSP for each S_k
17:     merge TSP tours to produce a VRP route
18: **end for**
19: **Return**: route with the shortest length

B.3 Google's OR-Tools

Google Optimization Tools (OR-Tools) [16] is an open-source solver for combinatorial optimization problems. OR-Tools contains one of the best available VRP solvers, which has implemented many

heuristics (e.g., Clarke-Wright savings heuristic [9], Sweep heuristic [36], Christofides' heuristic [8] and a few others) for finding an initial solution and metaheuristics (e.g. Guided Local Search [34], Tabu Search [14] and Simulated Annealing [21]) for escaping from local minima in the search for the best solution. The default version of the OR-Tools VRP solver does not exactly match the VRP studied in this paper, but with a few adjustments, we can use it as our baseline. The first limitation is that OR-Tools only accepts integer locations for the customers and depot while our problem is defined on the unit square. To handle this issue, we scale up the problem by multiplying all locations by 10^4 (meaning that we will have 4 decimal digits of precision), so the redefined problem is now in $[0, 10^4] \times [0, 10^4]$. After solving the problem, we scale down the solutions and tours to get the results for the original problem. The second difference is that OR-Tools is defined for a VRP with multiple vehicles, each of which can have at most one tour. One can verify that by setting a large number of vehicles (10 in our experiments), it is mathematically equivalent to our version of the VRP.

B.4 Optimal Solution

We use a mixed integer formulation for the VRP [31] and the Gurobi optimization solver [17] to obtain the optimal VRP tours. VRP has an exponential number of constraints, and of course, it requires careful tricks for even small problems. In our implementation, we start off with a relaxation of the capacity constraints and solve the resulting problem to obtain a lower bound on the optimal objective value. Then we check the generated tours and add the capacity constraint as *lazy-constraints* if a specific subtour's demand has violated the vehicle capacity, or the subtour does not include the depot. With this approach, we were able to find the optimal solutions for VRP10 and VRP20, but this method is intractable for larger VRPs; for example, on a single instance of VRP50, the solution has 6.7% optimality gap after 10000 seconds.

C Extended Results of the VRP Experiment

In this section, we present more detailed results for the VRP, including a comparison with baselines and an illustration of the solutions generated. We demonstrate the flexibility of the model to incorporate split deliveries, as an option, to further improve the solution quality. We also illustrate with an example that our proposed framework can be applied to more challenging VRPs with the stochastic elements.

C.1 Implementation Details

For the embedding, we use 1-dimensional convolution layers for the embedding, in which the in-width is the input length, the number of filters is D, and the number of in-channels is the number of elements of x. We find that training without an embedding layer always yields an inferior solution. One possible explanation is that the policy is able to extract useful features from the high-dimensional input representations much more efficiently. Recall that our embedding is an affine transformation, so it does not necessarily keep the embedded input distances proportional to the original 2-dimensional Euclidean distances.

We use one layer of LSTM RNN in the decoder with a state size of 128. Each customer location is also embedded into a vector of size 128, shared among the inputs. We employ similar embeddings for the dynamic elements; the demand d_t^i and the remaining vehicle load after visiting node i, $l_t - d_t^i$, are mapped to a vector in a 128-dimensional vector space and used in the attention layer. In the critic network, first, we use the output probabilities of the actor network to compute a weighted sum of the embedded inputs, and then, it has two hidden layers: one dense layer with ReLU activation and another linear one with a single output. The variables in both actor and critic network are initialized with Xavier initialization [13]. For training both networks, we use the REINFORCE Algorithm and Adam optimizer [20] with learning rate 10^{-4}. The batch size N is 128, and we clip the gradients when their norm is greater than 2. We use dropout with probability 0.1 in the decoder LSTM. Moreover, we tried the entropy regularizer [35, 26], which has been shown to be useful in preventing the algorithm from getting stuck in local optima, but it does not show any improvement in our experiments; therefore, we do not use it in the results reported in this paper.

On a single GPU K80, every 100 training steps of the VRP with 20 customer nodes takes approximately 35 seconds. Training for 20 epochs requires about 13.5 hours. The TensorFlow implementation of our code will be publicly available.

C.2 Flexibility to VRPs with Split Demands

In the classical VRP that we studied in Section 4, each customer is required to be visited exactly once. On the contrary to what is usually assumed in the classical VRP, one can relax this constraint to obtain savings by allowing split deliveries [2]. In this section, we show that this relaxation is straightforward by slightly modifying the masking scheme. Basically, we omit the condition *(iii)* from the masking introduced in Section 4, and use the new masking with the exactly similar model; we want to emphasize that we do not re-train the model and use the variables trained previously, so this property is achieved at no extra cost.

Figure 5 shows the improvement by relaxing these constraints, where we label our relaxed method with "RL-SD". Other heuristics does not have such option and they are reported for the original (not relaxed) problem. In parts 5a and 5b we illustrate the "optimality" gap for VRP10 and VRP20, respectively. What we refer to optimality in this section (and other places in this paper) is the optimal objective value of the non-relaxed problem. Of course, the relaxed problem would have a lower optimal objective value. That is why RL-SD obtains negative values in these plots. We see that RL-SD can effectively use split delivery to obtain solutions that are around $5 - 10\%$ shorter than the "optimal" tours. Similar to 3, parts 5c and 5d show the winning percentage of the algorithms in rows in comparison to the ones in columns. We observe that the winning percentage of RL-SD methods significantly improves after allowing the split demands. For example in VRP50 and VRP100, RL-SD-Greedy is providing competitive results with OR-Tools, or RL-SD-BS(10) outperforms OR-Tools in roughly 67% of the instances, while this number was around 61% before relaxation.

(a) Comparison for VRP10

(b) Comparison for VRP20

	RL-SD-Greedy	RL-SD-BS(5)	RL-SD-BS(10)	CW-Greedy	CW-Rnd(5,5)	CW-Rnd(10,10)	SW-Basic	SW-Rnd(5)	SW-Rnd(10)	OR-Tools
RL-SD-Greedy		15.8	8.7	99.7	98.0	96.9	98.8	98.8	98.8	46.8
RL-SD-BS(5)	82.1		15.4	99.7	99.4	99.3	99.4	99.4	99.4	60.5
RL-SD-BS(10)	90.4	59.2		99.9	99.5	99.6	99.6	99.6	99.6	66.0
CW-Greedy	0.3	0.3	0.1		0.0	0.0	68.9	68.9	68.9	1.0
CW-Rnd(5,5)	2.0	0.6	0.5	92.2		30.4	84.5	84.5	84.5	3.5
CW-Rnd(10,10)	3.1	0.7	0.4	97.5	68.0		86.8	86.8	86.8	4.7
SW-Basic	1.2	0.6	0.4	31.1	15.5	13.2		0.0	0.0	1.4
SW-Rnd(5)	1.2	0.6	0.4	31.1	15.5	13.2	0.0		0.0	1.4
SW-Rnd(10)	1.2	0.6	0.4	31.1	15.5	13.2	0.0	0.0		1.4
OR-Tools	53.2	39.5	34.0	99.0	96.5	95.3	98.6	98.6	98.6	

(c) Comparison for VRP50

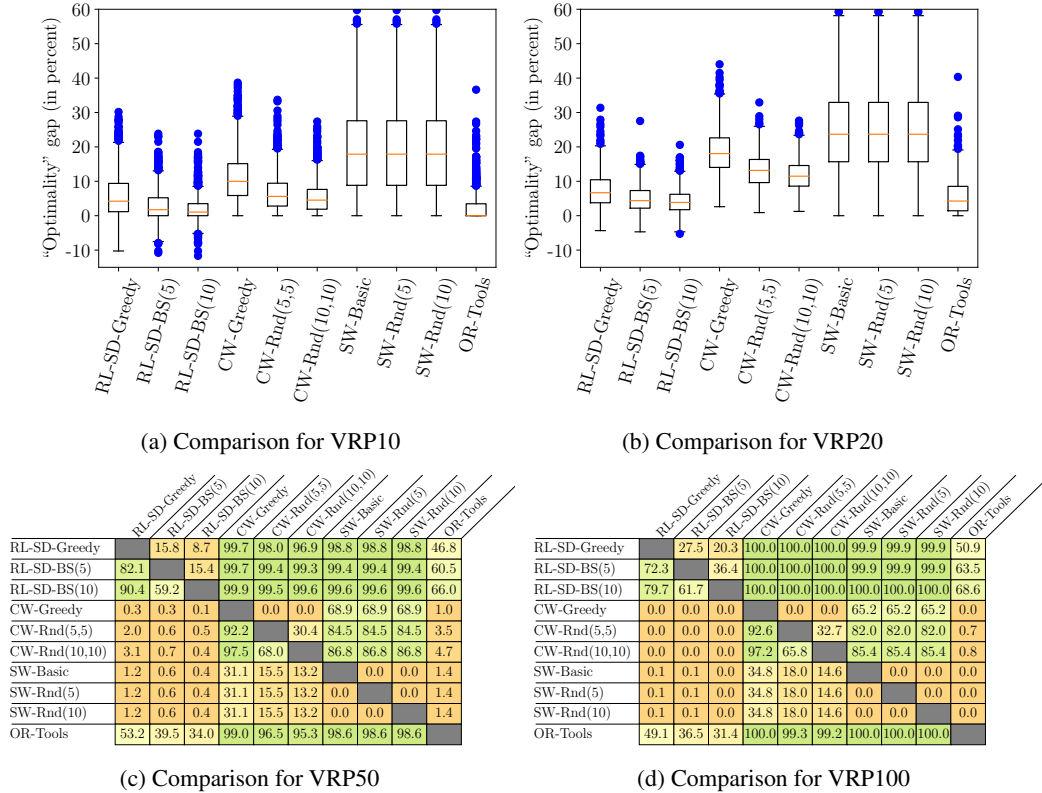|	RL-SD-Greedy	RL-SD-BS(5)	RL-SD-BS(10)	CW-Greedy	CW-Rnd(5,5)	CW-Rnd(10,10)	SW-Basic	SW-Rnd(5)	SW-Rnd(10)	OR-Tools
RL-SD-Greedy		27.5	20.3	100.0	100.0	100.0	99.9	99.9	99.9	50.9
RL-SD-BS(5)	72.3		36.4	100.0	100.0	100.0	99.9	99.9	99.9	63.5
RL-SD-BS(10)	79.7	61.7		100.0	100.0	100.0	100.0	100.0	100.0	68.6
CW-Greedy	0.0	0.0	0.0		0.0	0.0	65.2	65.2	65.2	0.0
CW-Rnd(5,5)	0.0	0.0	0.0	92.6		32.7	82.0	82.0	82.0	0.7
CW-Rnd(10,10)	0.0	0.0	0.0	97.2	65.8		85.4	85.4	85.4	0.8
SW-Basic	0.1	0.1	0.0	34.8	18.0	14.6		0.0	0.0	0.0
SW-Rnd(5)	0.1	0.1	0.0	34.8	18.0	14.6	0.0		0.0	0.0
SW-Rnd(10)	0.1	0.1	0.0	34.8	18.0	14.6	0.0	0.0		0.0
OR-Tools	49.1	36.5	31.4	100.0	99.3	99.2	100.0	100.0	100.0	

(d) Comparison for VRP100

Figure 5: Parts 3a and 3b show the "optimality" gap (in percent) using different algorithms/solvers for VRP10 and VRP20. Parts 3c and 3d give the proportion of the samples (in percent) for which the algorithms in the rows outperform those in the columns; for example, RL-BS(5) is provides shorter tours compared to RL-greedy in 82.1% of the VRP50 instances.

15

C.3 Summary of Comparison with Baselines

Table 2 provides the average and the standard deviation of tour lengths for different VRPs. We also test the RL approach using the split delivery option where the customer demands can be satisfied in more than one subtours (labeled with "RL-SD", at the end of the table). We observe that the average total length of the solutions found by our method using various decoders outperforms the heuristic algorithms and OR-Tools. We also see that using the beam search decoder significantly improves the solution while only adding a small computational cost in run-time. Also allowing split delivery enables our RL-based methods to improve the total tour length by a factor of around 0.6% on average. We also present the solution time comparisons in this table, where all the times are reported on a single core Intel 2.6 GHz CPU.

Table 2: Average tour length, standard deviations of the tours and the average solution time (in seconds) using different baselines over a test set of size 1000.

Baseline	VRP10, Cap20			VRP20, Cap30			VRP50, Cap40			VRP100, Cap50		
	mean	std	time	mean	std	time	mean	std	time	mean	std	time
RL-Greedy	4.84	0.85	0.049	6.59	0.89	0.105	11.39	1.31	0.156	17.23	1.97	0.321
RL-BS(5)	4.72	0.83	0.061	6.45	0.87	0.135	11.22	1.29	0.208	17.04	1.93	0.390
RL-BS(10)	4.68	0.82	0.072	**6.40**	0.86	0.162	**11.15**	1.28	0.232	**16.96**	1.92	0.445
CW-Greedy	5.06	0.85	0.002	7.22	0.90	0.011	12.85	1.33	0.052	19.72	1.92	0.186
CW-Rnd(5,5)	4.86	0.82	0.016	6.89	0.84	0.053	12.35	1.27	0.217	19.09	1.85	0.735
CW-Rnd(10,10)	4.80	0.82	0.079	6.81	0.82	0.256	12.25	1.25	0.903	18.96	1.85	3.171
SW-Basic	5.42	0.95	0.001	7.59	0.93	0.006	13.61	1.23	0.096	21.01	1.51	1.341
SW-Rnd(5)	5.07	0.87	0.004	7.17	0.85	0.029	13.09	1.12	0.472	20.47	1.41	6.32
SW-Rnd(10)	5.00	0.87	0.008	7.08	0.84	0.062	12.96	1.12	0.988	20.33	1.39	12.443
OR-Tools	**4.67**	0.81	0.004	6.43	0.86	0.010	11.31	1.29	0.053	17.16	1.88	0.231
Optimal	4.55	0.78	0.029	6.10	0.79	102.8		—			—	
RL-SD-Greedy	4.80	0.83	0.059	6.51	0.84	0.107	11.32	1.27	0.176	17.12	1.90	0.310
RL-SD-BS(5)	4.69	0.80	0.063	6.40	0.85	0.145	11.14	1.25	0.226	16.94	1.88	0.401
RL-SD-BS(10)	4.65	0.79	0.074	6.34	0.80	0.155	11.08	1.24	0.250	16.86	1.87	0.477

C.4 Sample VRP Solutions

Figure 6 illustrates sample VRP20 and VRP50 instances decoded by the trained model. The greedy and beam-search decoders were used to produce the figures in the top and bottom rows, respectively. It is evident that these solutions are not optimal. For example, in part (a), one of the routes crosses itself, which is never optimal in Euclidean VRP instances. Another similar suboptimality is evident in part (c) to make the total distance shorter. However, the figures illustrate how well the policy model has understood the problem structure. It tries to satisfy demands at nearby customer nodes until the vehicle load is small. Then, it automatically comprehends that visiting further nodes is not the best decision, so it returns to the depot and starts a new tour. One interesting behavior that the algorithm has learned can be seen in part (c), in which the solution reduces the cost by making a partial delivery; in this example, we observe that the red and blue tours share a customer node with demand 8, each satisfying a portion of its demand; in this way, we are able to meet all demands without needing to initiate a new tour. We also observe how using the beam-search decoder produces further improvements; for example, as seen in parts (b)–(c), it reduces the number of times when a tour crosses itself; or it reduces the number of tours required to satisfy all demands as is illustrated in (b).

Tables 3 and 4 present the solutions found by our model using the greedy and beam search decoders for two sample VRP10 instances with a vehicle capacity of 20. We have 10 customers indexed $0 \cdots 9$ and the location with the index 10 corresponds to the depot. The first line specifies the customer locations as well as their demands and the depot location. The solution in the second line is the tour found by the greedy decoder. In the third and fourth line, we observe how increasing the beam width helps in improving the solution quality. Finally, we present the optimal solution in the last row. In 4, we illustrate an example where our method has discovered a solution by splitting the demands which is, in fact, considerably shorter than the optimal solution found by solving the mixed integer programming model.
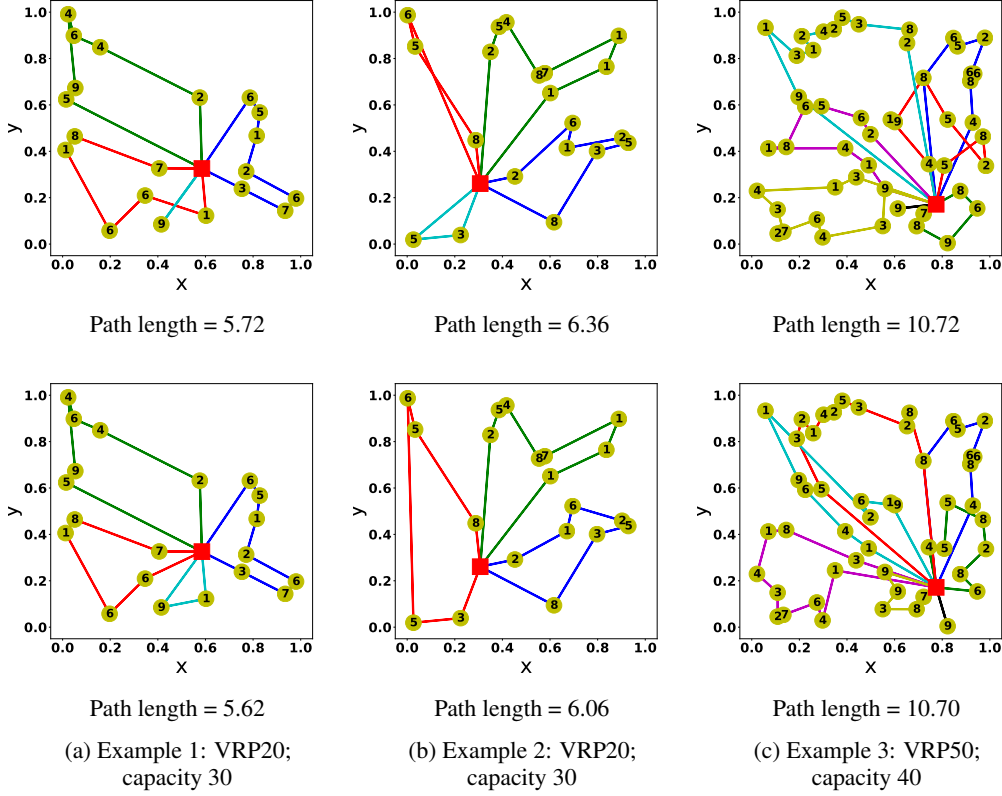
Figure 6: Sample decoded solutions for VRP20 and VRP50 using greedy (in top row) and beam-search (bottom row) decoder. The numbers inside the nodes are the demand values.

C.5 Attention Mechanism Visualization

In order to illustrate how the attention mechanism is working, we relocated customer node 0 to different locations and observed how it affects the selected action. Figure 7 illustrates the attention in initial decoding step for a VRP10 instance drawn in part (a). Specifically, in this experiment, we let the coordinates of node 0 equal $\{0.1 \times (i, j), \forall i, j \in \{1, \cdots, 9\}\}$. In parts (b)-(d), the small bottom left square corresponds to the case where node 0 is located at [0.1,0.1] and the others have a similar interpretation. Each small square is associated with a color ranging from black to white, representing the probability of selecting the corresponding node at the initial decoding step. In part (b), we observe that if we relocate node 0 to the bottom-left of the plane, there is a positive probability of directly going to this node; otherwise, as seen in parts (c) and (d), either node 2 or 9 will be chosen with high probability. We do not display the probabilities of the other points since there is a near-0 probability of choosing them, irrespective of the location of node 0. A video demonstration of the model and attention mechanism is available online at `https://streamable.com/gadhf`.

C.6 Experiment on Stochastic VRP

Next, we design a simulated experiment to illustrate the performance of the framework on the *stochastic VRP* (SVRP). A major difficulty of planning in these systems is that the schedules are not defined in beforehand, and one needs to deal with various customer/demand realizations on the fly. Unlike the majority of the previous literature which only considers one stochastic element (e.g., customer locations are fixed, but the demands can change), we allow the customers and their demands to be stochastic, which makes the problem intractable for many classical algorithms. (See the review of SVRP by Ritzinger et al. [28].) We consider an instance of the SVRP in which customers with random demands arrive at the system according to a Poisson process; without loss of generality we assume the process has rate 1. Similar to previous experiments, we choose each new customer's

Table 3: Solutions found for a sample VRP10 instance. We use different decoders for producing these solutions; the optimal route is also presented in the last row.

Sample instance for VRP10:
Customer locations: [[0.411, 0.559], [0.874, 0.302], [0.029, 0.127], [0.188, 0.979], [0.812, 0.330], [0.999, 0.505], [0.926, 0.705], [0.508, 0.739], [0.424, 0.201], [0.314, 0.140]]
Customer demands: [2, 4, 5, 9, 5, 3, 8, 2, 3, 2]
Depot location: [0.890, 0.252]

Greedy decoder:
Tour Length: 5.305
Tour: $10 \to 5 \to 6 \to 4 \to 1 \to 10 \to 7 \to 3 \to 0 \to 8 \to 9 \to 10 \to 2 \to 10$

BS decoder with width 5:
Beam tour lengths: [5.305, 5.379, 4.807, 5.018, 4.880]
Best beam: 2, Best tour length: 4.807
Best tour: $10 \to 5 \to 6 \to 4 \to 1 \to 10 \to 7 \to 3 \to 0 \to 10 \to 8 \to 2 \to 9 \to 10$

BS decoder with width 10:
Beam tour lengths: [5.305, 5.379, 4.807, 5.0184, 4.880, 4.800, 5.091, 4.757, 4.8034, 4.764]
Best beam: 7, Best tour length: 4.757
Best tours: $10 \to 5 \to 6 \to 1 \to 10 \to 7 \to 3 \to 0 \to 4 \to 10 \to 8 \to 2 \to 9 \to 10$

Optimal:
Optimal tour length: 4.546
Optimal tour: $10 \to 1 \to 10 \to 2 \to 3 \to 8 \to 9 \to 10 \to 0 \to 4 \to 5 \to 6 \to 7 \to 10$

Table 4: Solutions found for a sample VRP10 instance where by splitting the demands, our method significantly improves upon the "optimal" (of which no split demand is allowed).

Sample instance for VRP10:
Customer locations: [[0.253, 0.720], [0.289, 0.725], [0.132, 0.131], [0.050, 0.609], [0.780, 0.549], [0.014, 0.920], [0.624, 0.655], [0.707, 0.311], [0.396, 0.749], [0.468, 0.579]]
Customer demands: [5, 6, 3, 1, 9, 8, 9, 8, 7, 7]
Depot location: [0.204, 0.091]

Greedy decoder:
Tour Length: 5.420
Tour: $10 \to 7 \to 4 \to 9 \to 10 \to 6 \to 9 \to 8 \to 10 \to 1 \to 0 \to 5 \to 3 \to 10 \to 2 \to 10$

BS decoder with width 5:
Beam tour lengths: [5.697, 5.731, 5.420, 5.386, 5.582]
Best beam: 3, Best tour length: 5.386
Best tour: $10 \to 7 \to 4 \to 6 \to 10 \to 6 \to 8 \to 9 \to 10 \to 1 \to 0 \to 5 \to 3 \to 10 \to 2 \to 10$

BS decoder with width 10:
Beam tour lengths: [5.697, 5.731, 5.420, 5.386, 5.362, 5.694, 5.582, 5.444, 5.333, 5.650]
Best beam: 8 , Best tour length: 5.333
Best tours: $10 \to 7 \to 4 \to 9 \to 10 \to 9 \to 6 \to 8 \to 10 \to 1 \to 0 \to 5 \to 3 \to 10 \to 2 \to 10$

Optimal:
Optimal tour length: 6.037
Optimal tour: $10 \to 5 \to 7 \to 10 \to 9 \to 10 \to 2 \to 10 \to 8 \to 10 \to 1 \to 3 \to 4 \to 6 \to 10$

location uniformly on the unit square and its demand to a discrete number in $\{1, \cdots, 9\}$. We fix the depot position to $[0.5, 0.5]$. A vehicle is required to satisfy as much demand as possible in a time horizon with length 100 time units. To make the system stable, we assume that each customer cancels its demand after having gone unanswered for 5 time units. The vehicle moves with speed 0.1 per time unit. Obviously, this is a continuous-time system, but we view it as a discrete-time MDP where the vehicle can make decisions at either the times of customer arrivals or after the time when the vehicle reaches a node.

The network and its hyper-parameters in this experiment are the same as in the previous experiments. One major difference is the RL training method, where we use asynchronous advantage actor-critic (A3C) [26] with one-step reward accumulation. The main reason for choosing this training method is that REINFORCE is not an efficient algorithm in dealing with the long trajectories. The details

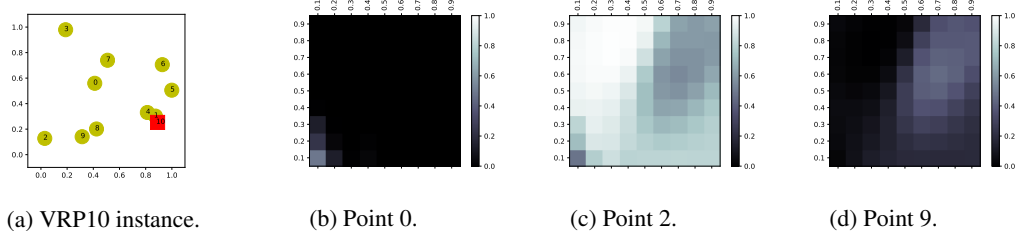| (a) VRP10 instance. | (b) Point 0. | (c) Point 2. | (d) Point 9. |

Figure 7: Illustration of attention mechanism at decoding step 0. The problem instance is illustrated in part (a) where the nodes are labeled with a sequential number; labels 0-9 are the customer nodes and 10 is the depot. We place node 0 at different locations and observe how it affects the probability distribution of choosing the first action, as illustrated in parts (b)–(d).

of the training method are described in Appendix D. The other difference is that instead of using masking, at every time step, the input to the network is a set of available locations which consists of the customers with positive demand, the depot, and the vehicle's current location; the latter decision allows the vehicle to stop at its current position, if necessary. We also add the *time-in-system* of customers as a dynamic element to the attention mechanism; it will allow the training process to learn customer abandonment behavior.

We compare our results with three other strategies: *(i) Random*, in which the next destination is randomly selected from the available nodes and it is providing a "lower bound" on the performance; *(ii) Largest-Demand*, in which the customer with maximum demand will be chosen as the next destination; and *(iii) Max-Reachable*, in which the vehicle chooses the node with the highest demand while making sure that the demand will remain valid until the vehicle reaches the node. In all strategies, we force the vehicle to route to the depot and refill when its load is zero. Even though simple, these baselines are common in many applications. Implementing and comparing the results with more intricate SVRP baselines is an important future direction.

Table 5 summarizes the average demand satisfied, and the percentage of the total demand that this represents, under the various strategies, averaged over 100 test instances. We observe that A3C outperforms the other strategies. Even though A3C does not know any information about the problem structure, it is able to perform better than the Max-Reachable strategy, which uses customer abandonment information.

Table 5: Satisfied demand under different strategies.

Method	Random	Largest-Demand	Max-Reachable	A3C
Avg. Dem.	24.83	75.11	88.60	112.21
% satisfied	5.4%	16.6%	19.6%	28.8%

D Training Policy Gradient Methods

We utilize the REINFORCE method, similar to Bello et al. [4] for solving the TSP and VRP, and A3C [26] for the SVRP. In this Appendix, we explain the details of the algorithms.

Let us consider a family of problems, denoted by $\mathcal{M}$, and a probability distribution over them, denoted by $\Phi_{\mathcal{M}}$. During the training, the problem instances are generated according to distribution $\Phi_{\mathcal{M}}$. We also use the same distribution in the inference to produce test examples.

REINFORCE Algorithm for VRP   Algorithm 3 summarizes the REINFORCE algorithm. We have two neural networks with weight vectors θ and ϕ associated with the actor and critic, respectively. We draw N sample problems from $\mathcal{M}$ and use Monte Carlo simulation to produce feasible sequences with respect to the current policy π_θ. We adopt the superscript n to refer to the variables of the nth instance. After termination of the decoding in all N problems, we compute the corresponding rewards as well as the policy gradient in step 14 to update the actor network. In this step, $V(X_0^n; \phi)$

19

is the the reward approximation for instance problem n that will be calculated from the critic network. We also update the critic network in step 15 in the direction of reducing the difference between the expected rewards with the observed ones during Monte Carlo roll-outs.

Algorithm 3 REINFORCE Algorithm

1: initialize the actor network with random weights θ and critic network with random weights ϕ
2: **for** $iteration = 1, 2, \cdots$ **do**
3:     reset gradients: $d\theta \leftarrow 0, d\phi \leftarrow 0$
4:     sample N instances according to $\Phi_{\mathcal{M}}$
5:     **for** $n = 1, \cdots, N$ **do**
6:         initialize step counter $t \leftarrow 0$
7:         **repeat**
8:             choose y_{t+1}^n according to the distribution $P(y_{t+1}^n | Y_t^n, X_t^n)$
9:             observe new state X_{t+1}^n
10:             $t \leftarrow t + 1$
11:         **until** termination condition is satisfied
12:         compute reward $R^n = R(Y^n, X_0^n)$
13:     **end for**
14:     $d\theta \leftarrow \frac{1}{N} \sum_{n=1}^{N} (R^n - V(X_0^n; \phi)) \nabla_\theta \log P(Y^n | X_0^n)$
15:     $d\phi \leftarrow \frac{1}{N} \sum_{n=1}^{N} \nabla_\phi (R^n - V(X_0^n; \phi))^2$
16:     update θ using $d\theta$ and ϕ using $d\phi$.
17: **end for**

Asynchronous Advantage Actor-Critic for SVRP The *Asynchronous Advantage Actor-Critic* (A3C) method proposed in [26] is a policy gradient approach that has been shown to achieve super-human performance playing Atari games. In this paper, we utilize this algorithm for training the policy in the SVRP. In this architecture, we have a central network with weights θ^0, ϕ^0 associated with the actor and critic, respectively. In addition, N agents are running in parallel threads, each having their own set of local network parameters; we denote by θ^n, ϕ^n the actor and critic weights of thread n. (We will use superscript n to denote the operations running on thread n.) Each agent interacts with its own copy of the VRP at the same time as the other agents are interacting with theirs; at each time-step, the vehicle chooses the next point to visit and receives some reward (or cost) and then goes to the next time-step. In the SVRP that we consider in this paper, R_t is the number of demands satisfied at time t. We note that the system is basically a continuous-time MDP, but in this algorithm, we consider it as a discrete-time MDP running on the times of system state changes $\{\tau_t : t = 0, \cdots\}$; for this reason, we normalize the reward R_t with the duration from the previous time step, e.g., the reward is $R_t/(\tau_t - \tau_{t-1})$. The goal of each agent is to gather independent experiences from the other agents and send the gradient updates to the central network located in the main thread. In this approach, we periodically update the central network weights by accumulated gradients and send the updated weight to all threads. This asynchronous update procedure leads to a smooth training since the gradients are calculated from independent VRP instances.

Both actor and critic networks in this experiment are exactly the same as the ones that we employed for the classical VRP. For training the central network, we use RMSProp optimizer with learning rate 10^{-5}.

Algorithm 4 Asynchronous Advantage Actor-Critic (A3C)

1: initialize the actor network with random weights θ^0 and critic network with random weights ϕ^0 in the master thread.
2: initialize N thread-specific actor and critic networks with weights θ^n and ϕ^n associated with thread n.
3: **repeat**
4:     **for** each thread n **do**
5:         sample a instance problem from $\Phi_{\mathcal{M}}$ with initial state X_0^n
6:         initialize step counter $t^n \leftarrow 0$
7:         **while** episode not finished **do**
8:             choose y_{t+1}^n according to $P(y_{t+1}^n | Y_t^n, X_t^n; \theta^n)$
9:             observe new state X_{t+1}^n;
10:             observe one-step reward $R_t^n = R(Y_t^n, X_t^n)$
11:             let $A_t^n = \left(R_t^n + V(X_{t+1}^n; \phi) - V(X_t^n; \phi) \right)$
12:             $d\theta^0 \leftarrow d\theta^0 + \nabla_\theta A_t^n \log P(y_{t+1}^n | Y_t^n, X_t^n; \theta^n)$
13:             $d\phi^0 \leftarrow d\phi^0 + \nabla_\phi \left(A_t^n \right)^2$
14:             $t^n \leftarrow t^n + 1$
15:         **end while**
16:     **end for**
17:     periodically update θ^0 using $d\theta^0$ and ϕ^0 using $d\phi^0$
18:     $\theta^n \leftarrow \theta^0, \phi^n \leftarrow \phi^0$
19:     reset gradients: $d\theta^0 \leftarrow 0, d\phi^0 \leftarrow 0$
20: **until** training is finished