Vehicle Routing Problem with Reinforcement Learning

Zhixiang Hu (zh2366) Yunke Gan (yg2631)

Abstract

The vehicle routing problem (VRP) is a combinatorial optimization problem which asks "What is the optimal set of routes for a fleet of vehicles to traverse in order to deliver to a given set of customers?". Finding the optimal solution to such a problem with a slightly big size can be very hard and computationally expensive. Many algorithms proposed so far are heuristic. In this report we used Neural Combinatorial Optimization method combined with the Reinforcement Learning framework to provide solutions to such problems. We proposed an architecture that is based on the work of Nazari et al. (2018) and found that by including dynamic demands information as input to the decoder network, the efficiency of the learned policy can be improved.

1. Introduction

The Vehicle Routing Problem (VRP) generalizes the well-known Travelling Salesman Problem (TSP). It is motivated by answering the following questions:

Given a set of customers with certain delivery demands, what is the optimal set of routes for vehicles to travel in the sense that it can minimize the total route cost, which can be usually measured by the total number of distances the vehicles travel.

In its simplest form, there is only one capacitated vehicle that is responsible for delivery items to a set of customers. During the process if the vehicle runs out of items it must return to the depot to re-load the truck. So in this case the objective is, given the locations of depot, locations of customers, capacity of the vehicle, we need to come up with a set of routes such that they attain the minimum travelling distances.

This can be modelled as a combinatorial optimization and integer programming problem. Determining the optimal solution to VRP is computationally difficult and is actually NP-hard, so we can only use mathematical programming and combinatorial optimization methods to solve such problems with limited sizes. Many algorithms proposed so far are heuristic and providing one that's fast and reliable is still very challenging and attracting researches from applied mathematics and computer science for decades. In this paper, we wish to provide a heuristic solution that is based on Reinforcement Learning (RL) framework.

2. Related Work

The problem was first approached by George Dantzig and John Ramser (1959, cite) in which they proposed a procedure based on linear programming formulation that obtains a near optimal solution to a problem that was applied to petrol deliveries. The exact algorithms for VRP were proposed in 1981 by Christofides et al. (Toth, 1981) where they used dynamic programming with state-space relaxation, q-paths and k-shortest spanning trees. The development of modern heuristics for the VRP started in the 1990s with the advent of metaheuristics. Pisinger and Ropke 2007 (Pisinger Ropke, 2007) several operators, as in adaptive large neighbourhood search. Vidal et al. (2012) (Vidal et al., 2014) proposed the hybrid genetic algorithm which combines genetic search with local search.

Sequence-to-Sequence models (Sutskever et al., 2014) (Luong et al., 2015) (Vinyals et al., 2015a) are useful in tasks for which a mapping from one sequence to another is required.

In more recent time, combinatory optimization problems like VRP and TSP have been tackled using techniques from the Artificial Intelligence community. Such *Neural Combinatorial Optimization* attempts was first approached by Vinayls et al. (Vinyals et al., 2015b) where they introduced the concept of Pointer Network (Ptr-Net). Ptr-Net was originally inspired by sequence-to-sequence models and the change made was that the new model is invariant to the length of the encoder sequence. This makes it possible for Ptr-Net to be applied to find near solutions to combinatory problems, where the length of the output sequence is determined by source sequence.

3. Problem Formulation

We define our problem similar to the framework proposed by Nazari et al. where each input is represented by a sequence of tuples $x_i^t = (s^i, d_t^i) \mid t \in N$ where s^i and d_t^i are static and dynamic elements of the input respectively. We want to generate a stochastic policy π which will minimize the loss objective while satisfying the problem constraints.

Therefore our goal is to make π as close as to the optimal policy π^* such that π^* generates the optimal solution with probability 1. To do this we wish to use a RNN model, that is similar to the Pointer Network proposed by Vinayls et al but also incorporates an attention mechanism.

An attention mechanism is a differentiable structure for addressing different parts of the input. Then the model is trained using well known policy gradient methods.

Formally, we can define our problem as follows:

- Agent States: $X_t = \{x_t^i, i = 1, ..., V; t = 1, .., T\}$, where $x_i^t = (s^i, d_t^i)$. Here x_i^t gives a snapshot of the state information for customer i at time step t where s^i corresponds to the 2-dimensional coordinate of customer i's location (in the case of a 2D graph) and d_t^i stands for his demand at time t.
- Agent Actions: $Y = \{y_t, t = 0, ..., T\}$, where y_{t+1} denotes a pointer that refers to the one of the available customers on time t, i.e.: X_t. The action defines which customer the vehicle should run to given the current information of customers' locations and availability as well as information about past actions $Y_t = \{y_0, ..., y_t\}$.
- Transition of States: $X_{t+1} = f(y_{t+1}, X_t)$ where f is the state transition function that is defined in the following sections. The states in next time step depend on the states at current time as well as the action to be taken by the vehicle.
- Reward Function: the immediate reward after transition from X_t to X_{t+1} with action y_{t+1} can be computed as the negative distance (or consumed time) between the two nodes that the vehicle visited during executing the action.
- Value Function: the value of a sequence of decisions Y is defined simply as the sum of their individual rewards.
- Policy: a stochastic rule π that governs how the decision is made in each time step.

We simplify the problem by first considering the case with only 1 depot and N customers and the corresponding demands are generated uniformly from $\{1, 2, ..., 9\}$.

4. Steps

The aim of this project is to train a random policy to propose solutions to the classic Vehicle Routing Problem (VRP). We want to use Reinforcement Learning to solve the problem with the following steps:

- Choose a problem size, e.g. VRP with 10 customers, namely VRP10.
- Randomly-generate many different datasets of the same problem size, where the datasets have customer locations, customer demands and depot locations, etc.
- Initialize the policy with randomly generated parameters and train the proposed model in those random-generated datasets using the Policy Gradient algorithm.
- Generate another new problem with the same size, e.g. VRP10, and use the trained model to predict the solution to the new problem and the total distance covered.

Note: we can also use the trained model to solve problem with similar size, i.e. we can use the trained model for VRP50 to solve VRP48 problems.

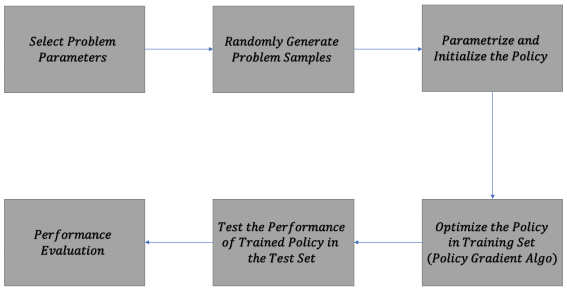

Figure 1. the Process

5. Methods

The core part of this project is to learn the policy π, i.e. what action to take given current state. We parameterize the policy π with parameters θ, where θ is vector of all trainable variables used in the model.

For any given VRP problem with customer locations, customer demands, depot location,trunk capacity, we can solve

the VRP problem by giving a sequence of actions, i.e. the order of going between different locations and depot. The probability of generating action sequence Y (i.e. $y_0, y_1, ..., y_{T+1}$) can be modeled as

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

and we have

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

For this problem, if we allow split delivery, the state transition function f can be explicitly defined as:

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

Here d_t^i is the demand of customer i at time t, and l_t is the remaining loads of the trunk at time t. Essentially, the first state transition function means the demand of the customer being visited will become 0 if the trunk has enough loads, otherwise only part of the demand will be filled. The second state transition function means the demands for customers not visited yet will not change. The third state transition function means the loads of the trunk will become 0 if it's not enough, otherwise there will be some remaining loads for the trunk.

We made use of the attention mechanism in order to take into account the relative importance of information over time, and we followed the approach used by Vinyals et al. The input at time t and $t-1$ are combined with the information from the decoder hidden state to decide how important every input data is in the next step, which is represented by attention vector a_t.

Let $x_t^i = (s^i, d_t^i)$ be the input i, and h_t be the memory state of the RNN cell at decoding step t, then a_t is computed as

$$a_t = a_t(x_t, h_t) = softmax(u_t)$$
$$u_t^i = v_a^T tanh(W_a[x_t^i; h_t])$$

Here ";" means the concatenation of two vectors. Then the context vector c_t can be computed using the information from the input and the attention vector:

$$c_t = \sum_{i=1}^{M} a_t^i x_t^i$$

and then we normalize the values with the softmax function, as follows:

$$P(y_(t+1) \mid Y_t, X_t) = softmax(u_t)$$
$$u_t^i = v_c^T tanh(W_a[x_t^i; c_t])$$

Note that v_a, v_c, W_a and W_c are all parameters to be trained to optimize the policy.

After that, the context vector and input are used to determine the policy π after going through a masking scheme. The masking scheme basically means the customer with no demand with not be considered any more and no customer will be considered if the trunk loads become 0. Specifically, here we come with a policy vector with different probability for different customers under consideration. The customer with highest probability is selected to deliver. If the trunk loads become 0, we will go back to depot to reload. After making the decision, the reward of the action is the opposite number of the distance covered.

After the random initialization of the policy, we will then use the Policy Gradient method to train the model in randomly generate data sets. Policy gradient methods use an estimate of the gradient of the expected return with respect to the policy parameters to iteratively improve the policy. Note that here we use a standard policy gradient method similar to Bello et al.

After that we will randomly generate another test set of the same size and use the trained policy to make decisions and get the action sequence Y. As an example, here we show the result for one randomly generated VRP10 example:

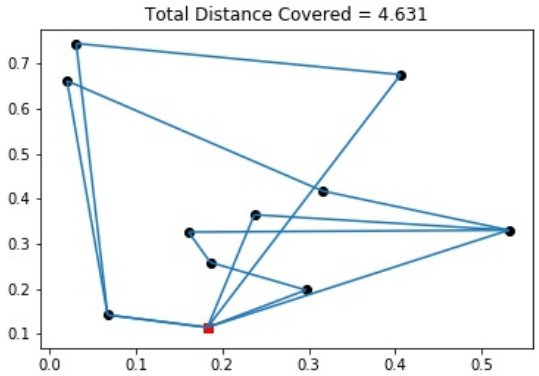

Figure 2. Example VRP10 result

Here the red point is the randomly generated depot and the black point is the randomly generated customer locations. We can see the routes from the graph and, in this case, the total distance covered is 4.631.

6. Architecture

We found some limitations and made improvements based on the work of Nazari et al. (2018). We first give an overview of how the original model works and then explain in further details of the improvement we made as well as the justification for it.

Figure 3 shows the original architecture from Nazari et al. (2018). The embedding layer works like a encoder and maps the inputs to a high-dimensional vector space. Next to the encoder is a RNN decoder that stores the information of the decoded sequence. In this case, the input to the decoder is a sequence of static elements that remember all the previous decisions (the static coordinates of customers on the graph) made by the policy. The results from the encoder and decoder are then fed into a attention layer to produce a probability distribution over the next decision.

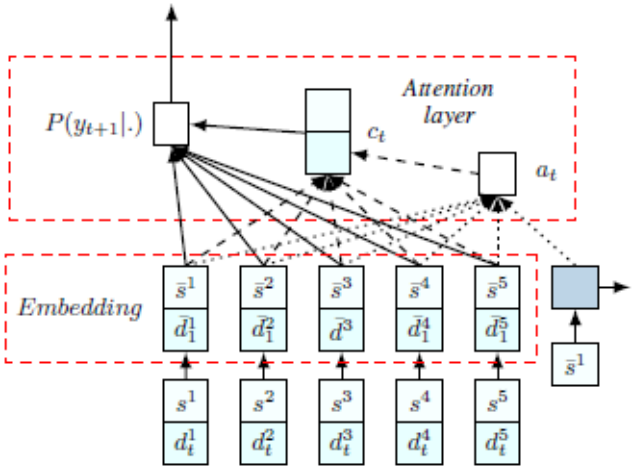

Figure 3. Original Architecture by Nazari et al. (2018)

So what could possibly be improved under this architecture? Remember the encoder part only contains the coordinates and demands information at current iteration and the decoder part only remembers static information from the past. In this case we discard the information about the changing demands of customers from previous iterations. However, changes in the demands could be useful for determining the next position the vehicle should be.

Figure 4 shows the architecture we proposed. The main modification we made is on the inputs to the RNN decoder. In this case we input both static and dynamic elements from previous iterations. The static elements remain the same, which are the previous decoded sequences $\{y_t\}$. The dynamic element is the context vector c_t from last iteration. We used c_t because first since it is a combination of the outputs from the encoder and decoder, it naturally contains the previous information about customers' demands. Second, it also contains information about the relative importance of the input nodes.

Before feeding the context vector into the decoder, we first apply a pooling layer on one of the dimensions of the context tensor and make sure it has the same shape as the decoder input. Then we take a combination of the context and decoder input and map the result into a new tensor and feed it into the decoder. In summary, our proposed change is as follows:

$$c_t^{'} = pooling(c_t)$$
$$I_{t+1} = f_d(c_t^{'}, I_{t+1}; W_d)$$

where c_t the context vector from last iteration, I_{t+1} the static input, $f_d(\cdot; W_d)$ the trainable mapping function. For simplicity, we used mean pooling for c_t and average mapping for $c_t{}'$ and I_{t+1}.

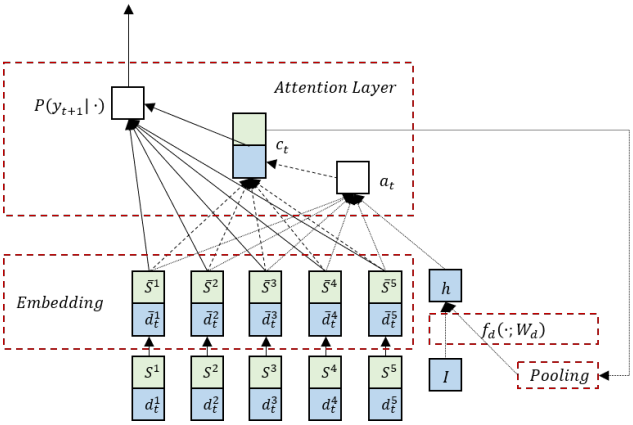

Figure 4. Proposed Architecture

7. Results

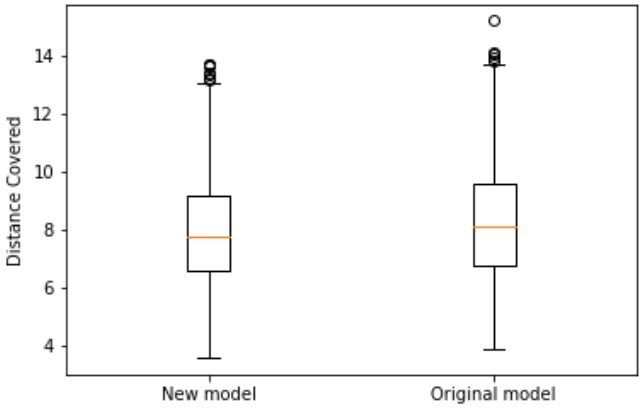

Figure 5. Performance Comparison

After training the original model and new model with 260,000 randomly generated examples, we used the trained policy to make decisions on 1000 randomly generated test examples. In all of the examples, the distances covered are recorded. Let's see how they perform.

As we can see from the table summarizing the statistics of the 1000 test examples, our new model achieves more than 4 percent less distance on average, and the standard deviation also shrinks by 5.6 percent. In addition, if we look at the box-plot, we can easily say that the new model,considering both the best and the worst cases, has significant improvement compared to the old model.

Avg. Distance	Original Model	New Model
mean	8.28	7.94
std	1.96	1.85

Figure 6. Performance Comparison: Statistics

8. Conclusion

In this report we proposed an architecture that is based on the work of Nazari et al. (2018) and tested it on 260,000 randomly generated VRP examples. We added a few pooling and mapping layers before the input layer of the decoder network so as to integrate information from the context vector and the static decoded sequence. We found that by including the dynamic demands information as input to the decoder network, the policy performance as measured by the total traveling distance can be improved by more than 4 percent.

9. Repository

The data and code can be found at:

https://github.com/DeepLearningCUCS/RLVRP

References

Vehicle routing problems with time windows (vrptw) are from open-source solomon problems website., a. URL http://people.idsia.ch/ luca/macs-vrptw/problems/welcome.html.

Capacitated vehicle routing problem (cvrp) datasets from vehicle routing datasets website, b. URL https://www.coin-or.org/SYMPHONY/branchandcut/VRP/data/index.htm.old.

Dantzig GB, R. J. The truck dispatching problem, 1959.

Luong, M., Pham, H., and Manning, C. D. Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025, 2015. URL http://arxiv.org/abs/1508.04025.

Nazari, M., Oroojlooy, A., Snyder, L. V., and Takác, M. Deep reinforcement learning for solving the vehicle routing problem. *CoRR*, abs/1802.04240, 2018. URL http://arxiv.org/abs/1802.04240,https://github.com/OptMLGroup/VRP-RL.

Pisinger, D. and Ropke, S. A general heuristic for vehicle routing problems. *Computers Operations Research*, 34(8):2403 – 2435, 2007. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2005.09.012. URL http://www.sciencedirect.com/science/article/pii/S0305054805003023.

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014. URL http://arxiv.org/abs/1409.3215.

Toth, N. C. A. M. P. *Networks, An international Journal*. 1981.

Vidal, T., Crainic, T. G., Gendreau, M., and Prins, C. A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research*, 234(3):658 – 673, 2014. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2013.09.045. URL http://www.sciencedirect.com/science/article/pii/S037722171300800X.

Vinyals, O., Bengio, S., and Kudlur, M. Order matters: Sequence to sequence for sets, 2015a.

Vinyals, O., Fortunato, M., and Jaitly, N. Pointer networks, 2015b.