Deep Q-Learning with Structure2Vec for solving the Vehicle Routing Problem

Xiangyu Wang, Amlesh Sivanantham, Bistra Dilkina

University of Southern California

Abstract. We would like to extend the work done by Dai et al. [1] to solve the Vehicle Routing Problem (VRP). Current approaches to solving the vehicle routing problem all utilize policy gradient methods to train a policy. Work by Bello et al. [2] first shows that pointer networks can be trained to solve the travelling salesman problem (TSP). Nazari et al. [3] shows that by relaxing the pointer network architecture by replacing the encoder recurrent neural network (RNN) with an attention mechanism, it is able to solve the VRP. Kool et al. [4] uses an architecture known as the Transformer [5] that uses only graph attention layers and is also able to solve the VRP. These three approaches have only looked at improvements in the architecture of the network and not the learning algorithm. Dai et al. [1] utilizes a graph-embedding network known as `structure2vec` (S2V) and Deep n-step Q-learning that can be shown to solve Minimum Vertex Cover, Maximum Cut, and the TSP. We believe that this approach can also solve the VRP. We believe that S2V can provide us with better generalization to larger instance sizes which the prior approaches were not able to do. We also believe that by using n-step Q-learning, we will be able to train the model faster. The training would be off-policy so model parameters need not be updated with respect to the solution like in policy gradient methods.

Keywords: Deep Q Learning, Graph Embedding, Vehicle Routing Problem

1 Introduction

The *Vehicle Routing Problem* (VRP) is a combinatorial optimization problem that is known to be NP-hard. It is a more general version of the Travelling Salesman Problem (TSP). Like the TSP, we have a vehicle, a collection of cities the vehicle needs to get to, and we would of course like to minimize the travel path of the vehicle. What makes the VRP different is that we have a capacitated vehicle that must traverse the nodes of the graph and supply them resources. Each node has a certain amount of demand that must be satisfied. The vehicle starts at a special node labelled the depot and must end its path here. The vehicle will likely not be able to satisfy all the nodes in a single run. It must return back to the depot to restock on resources and then proceed to go back to supplying the demands of the nodes. We also do not allow a vehicle to travel to a node if it doesn't have enough items to supply to that node. There are many variants of the VRP such as stochastic VRP and split-delivery VRP. In split-delivery VRP, we remove the restriction we placed on letting the vehicle supply a node that has a greater demand than the current amount of resource it is carrying. In stochastic VRP, we allow nodes and their demands to be stochastic. We will not be considering these alternate versions of the VRP and will just look at the capacitated VRP. Our goal is to find a route that supplies all the nodes such that we minimize the total vehicle distance. If we frame the VRP as a Reinforcement Learning (RL) environment, we can say that our RL goal objective is to maximize the expected reward. The details of this formulation will be explained in Section 3.

Researchers have extensively studied the problem and have found many exact and heuristic algorithms for solving the VRP, but it still has been a computationally difficult problem to solve. Many of these heuristic algorithms are hand engineered. Recently, there has been an interest in learning these heuristics instead of manually hand-engineering them. With the current advances in Deep Learning and Reinforcement Learning, these ideas seem much more fruitful of an endeavour. Neural Networks can be trained on a myriad of tasks in both a supervised and unsupervised setting. When they we can formulate the problem as a Markov Decision Process (MDP), the Neural Networks can also solve RL tasks. This work will focus primarily on the Q-learning algorithm to learn the VRP in an RL paradigm. We utilize the Q-learning framework alongside the `structure2vec` embedding over an instance of the VRP.

2 Related Work

The application of Neural Networks(NNs) for combinatorial optimization problems has started to become popular recently. The first application can be dated back to 1985 where Hopfield and Tank deployed Hopfield Network to solve TSPs[6]. Although NNs have been used in many ways for combinatorial optimization[7], most of its applications have been implemented in an online manner. The solutions are instances-specific and the must be retrained once we encounter a new problem instance. More recently, researchers have started using NNs to learn heuristics for an entire class of problem instances instead of engineering ad-hoc solutions for each problem type.

Due to the resurgence of Neural Network thanks to recent advancements in the field of Deep Learning, there is a returned interest in using NNs to solve combinatorial tasks. Vinyals et al. first proposed Pointer Networks[8]. Very similar to the sequence-to-sequence learning proposed by Sutskever et al.[9] but instead of directly outputting a probability distribution over a fixed vocabulary, it utilizes a non-parametric softmax modules which is used to point to a specific position in the input sequence. This allowed the new architecture to support problems where the vocabulary size was the input sequence. They also trained TSP instances in a supervised manner. While the training paradigm may not be ideal, Bello and Pham et al. trained the Pointer Networks on TSP instances with policy gradient with actor-critic instead[2]. They showed that they were able to get better results than Vinyals et al. They performed inference in two ways; they either did a lot of sampling with the trained policy, or they refined the parameters of a trained stochastic policy network on a single test input with an exponential moving average baseline instead of a critic. This is called the Active Search method[2]. The time it took to perform either of those actions however downplayed the results. The average tour length may have been better, but there was significantly more computational cost.

Then Nazari et al.[3] extended the use of the Pointer Network to VRP without the extra sampling or active search used in the previous approach. They replaced the RNN encoder with non-recurrent graph attention layer because the input to a combinatorial optimization problem is not ordered. The order of cities do not matter for the TSP or the VRP. Furthermore, training an RNN to learn an embedding where the order of the input sequence does not matter is much more computationally expensive than using graph attention layers which can learn a simple embedding over the input sequence. Their method can also be applied on VRP with split deliveries and stochastic demands. In Kool et al.'s work[4], they adapted the Transformer Architecture[5], which is entirely based on graph attention layers and no recurrent networks. Their approach can consistently outperform the farthest insertion baseline, and can also be applied to Captivated VRP and Split Delivery VRP.

Most of the approaches above are trained with policy gradient. It must be trained off policy which means that the model parameters have to be updated with respect to the whole solution. It is not a sample-efficient method and requires a lot of training instances to achieve desirable behaviour in the policy network. Dai et al.[10] tries to solve graph combinatorial optimization problem with n-step Deep Q-learning and a graph embedding known as structure2vec. Their work can't be directly applied to VRP but our goal for this paper is to adapt Dai et al.'s [10] work so that it solves the Vehicle Routing Problem.

3 Method

Figure 1 shows an update step computed by the proposed framework as applied to an instance of the Vehicle Routing Problem. At the beginning of each step, the embedding of each node is just some random number. Then, it's updated after a few iterations of message passing through the `structure2vec`[11] architecture. With the updated node embedding, the Q-value is calculated at each node. Finally, the node with the maximum Q-value is added to the current partial tour. At each step, some nodes may be masked based on the masking mechanism introduced in the following section.

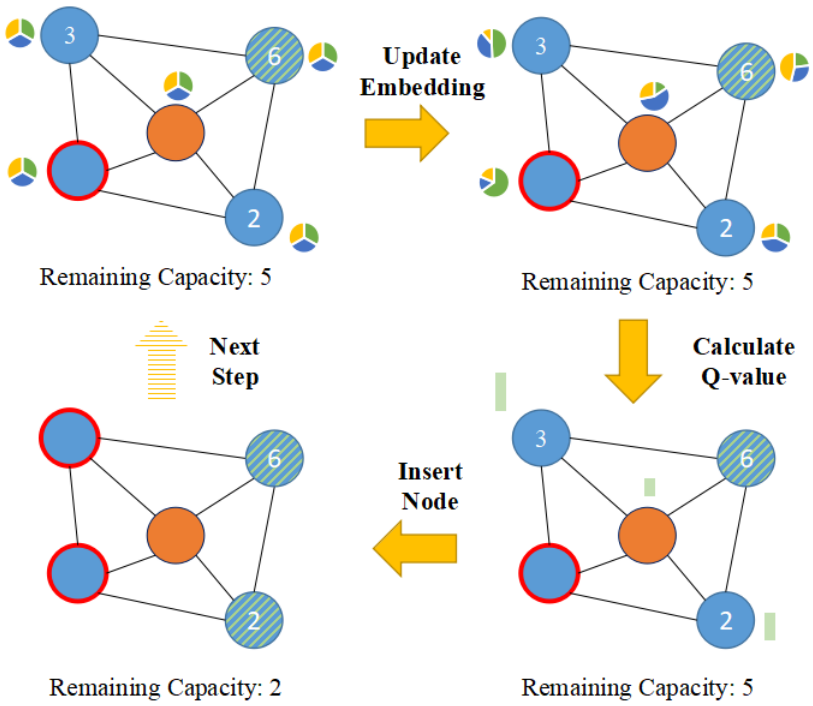

Fig. 1. Illustration of the proposed framework as applied to an instance of the Vehicle Routing Problem. The number in each node shows the current demand of this node. After a few iterations of message passing, the node embedding(pie chart) is updated. Then, the Q-value(green bar) is calculated for each node. The nodes with dashes are masked at the current step.

3.1 Structure2Vec Graph Embedding

From Dai et al.'s work, the `structure2vec`[11] embedding is defined by the recursive message passing function given below, where μ_u is a p-dimensional vector.

$$\mu_v^{t+1} \longleftarrow relu(\theta_1 x_v + \theta_2 \sum_{u \in N(v)} \mu_u^{(t)} + \theta_3 \sum_{u \in N(v)} relu(\theta_4 w(v,u))) \tag{1}$$

But from the implementation of their code, the equation is actually in the form below.

$$\mu_v^{t+1} \longleftarrow relu(\theta_1 \sum_{u \in N(v)} \mu_u^{(t)} + \theta_2 \sum_{u \in N(v)} relu(\theta_3 w(v,u))) \tag{2}$$

Where μ_u is the node embedding for the node u and $w(v,u)$ is the edge embedding for the edge $e(v,u)$. These two kinds of embedding are learned from the corresponding feature vectors. The node feature vector denotes whether the node is the start node, whether it's already the visited and its coordinate. The edge feature vector denotes the weight of the edge and whether any end has already been visited.
So in the setting of VRP, we basically only need to change the node feature vector. Besides adding the demand value into the node feature vector, we also added some binary terms according to the following masking rules[3]:

1. Nodes with zero demand are not allowed to be visited.
2. Customers whose demand is greater than the current vehicle capacity are masked.
3. The depot can't be visited in two consecutive timesteps.

We enforce rule (2) because our VRP setting is that of the non split delivery kind. It may be relaxed when we allow split delivery in our future experiments. Once the embedding in Equation (2) is updated after T

iterations, we can use this embedding to define the $\hat{Q}(h(S), v; \Theta)$, where S is the set of all the nodes that has been visited, and the helper function $h(S)$ helps maintain a tour from the set of nodes. In our case, the helper function inserts the node into the current partial tour where there will be minimum increase of the partial tour length. The $\hat{Q}(h(S), v; \Theta)$ depends on the node embedding μ_u, the pooled embedding over the entire graph $\sum_{u \in V} \mu_u^{(T)}$ and the helper function $h(S)$ as below

$$\hat{Q}(h(S), v; \Theta) = \theta_6^\top relu([\theta_4 \sum_{u \in V} \mu_u^{(T)}, \theta_5 \mu_v^{(T)}]) \tag{3}$$

Where $\{\theta_i\}_{i=1}^6 \in \mathbb{R}^{p \times p}$ vector and $\theta_6 \in \mathbb{R}^{2p}$. The number of iterations T is set to $T = 4$. In the current implementation, the same embedding is used for the depot u_0 and the other nodes $u \in V \backslash \{0\}$. For the depot, we encode the remaining capacity as its demand value. But since the depot is quite different from the other nodes which also carries the information about the remaining capacity, we may try to separate it from the other nodes later

$$\hat{Q}(h(S), v; \Theta) = \theta_6^\top relu([\theta_4 \sum_{u \in V \backslash \{0\}} \mu_u^{(T)}, \theta_5 \mu_v^{(T)}, \theta_7 u_0]) \tag{4}$$

Then, we train theses parameters end-to-end using the Q-learning framework.

3.2 Q-learning

In the definition from [10], states are defined as a sequence of actions(nodes) on the graph. In the setting of VRP, a single state must also include the information of the remaining demand at each node and the remaining capacity of the vehicle. Once the state definition has been determined, the transition function, action, reward and policy architecture are the same as in the TSP setting used by Dai et al. [10]. We use a helper function to help of transition between states in the MDP. It is an insertion operation into the partial tour, and it lets us define the reward which should be the negative of the increase in the partial tour length. However, in the experiments, we find it work better if we do in the opposite way. The reason is explained in [1] as flipping the sign of the reward function makes the RL agent focus on the future rewards instead of being toward most recent rewards.

Then we use a combination of n-step Q-learning and fitted Q-learning to learn the function approximation model for $\hat{Q}(h(S), v; \Theta)$. In general, we will try to minimize the squared loss below by performing batch gradient descent.

$$(y - \hat{Q}(h(S), v; \Theta)) \tag{5}$$

where $y = \sum_{i=0}^{n-1} r(S_{t+i}, v_{t+i}) + \gamma \, max_{v'} \hat{Q}(h(S), v'; \Theta)$. The same as in [12], we will use *experience replay* to update the function approximator with a batch of samples from the memory buffer E. The memory buffer E is collected from the previous episodes, where each sample in E is a state action reward tuple $(S_t, a_t, R_{t,t+n}, S_{t+n})$ for timestep t. Then we perform batch gradient descent on this batch of samples. Because the Q-learning is trained in an off-policy manner, it is much more sample efficient than on-policy methods such as policy gradient.

4 Experiment

We compared against the very same tests that was used in the previous work [4], where we trained and tested our model against problem instances with 10, 20, 50, and 100 nodes with vehicle capacities 20, 30, 40, and 50 respectively. The customers and depot locations are randomly generated in a unit square, while the demands are sampled from discrete integer intervals. For our experiments, we did not consider the split demand scenario which allows deliveries to be split over multiple visits to the same node. Basically, We ran two experiments that gave us insight on the optimality and generality of our method.

4.1 Optimality of the Method

In this section, we compared our method against the prior approaches mentioned before in this paper. The most important approach we compared against is Kool et al. [4] as it is the current best. We also compared against Nazari et al. [3] for a sake of completeness. Furthermore, we also compared against the Clarke-Wright savings heuristic, the Sweep heuristic, and Google's OR Tools. We did not implement these results ourselves, but used the results reported by Kool et al. [4] and Nazari et al. [3] instead. We did however verify the results of their codebase provided on GitHub and we were able to get results very close to the ones they've reported. The results of optimality are shown in table 1.

Method	VRP10,Cap20	VRP20,Cap30	VRP50,Cap40	VRP100,Cap50
S2V-DQN (without constraint)	5.60	9.00	14.00	−*
S2V-DQN (with constraint)	5.54	7.68	13.75	20.89
AM, Greedy [4]	4.65	6.40	10.98	16.78
RL, Greedy [3]	4.84	6.59	11.39	17.23
RL, Beam Search (Size 10) [3]	4.68	6.40	11.15	16.96
Randomized Clarke-Wright [3]	4.80	6.81	12.25	18.96
Randomized Sweep [3]	5.00	7.08	12.96	20.33
Google OR-Tools [3]	4.67	6.43	11.31	17.16
Optimal [3]	4.55	6.10	–	–

Table 1. Results for the Capacitated Vehicle Routing Problem comparing this work against the optimality from Kool et al. [4] and Nazari et al. [3]. We also show how it performs compared to Randomized Clarke-Wright heuristic, Randomized Sweep heuristic, and Google OR Tools. The result from the last version of S2V-DQN on VRP100 is missing because it's too unstable which ranges from 30 to 50.

In our first trial, we found that the learned policy tends to go back to the depot very often even if there's no need to refill. So we added the constraint that the vehicle can't go back to depot until necessary in our latest code. From the first two rows above, we can see that it did improve a bit but still far from satisfactory. Maybe it's better to add a penalty term in the loss function instead of hard coding it.

Figure 2 shows an example of the greedy policy in action on the VRP10Cap20 test set. The black box represents the depot. The to-and-from edges from the depot are not drawn on the figure to make things more clear. Each individual route is then given a color. The ordering of the routes are defined by the legend. In figure 2, there are also bars at each node. The bar represents the capacity of the vehicle. The black segment of the bar represents how much resource the vehicle has supplied to the node. The white segment of the bar represents how much extra resource was available at the end of that route. Figure 3 compares resultant routes produced by our policy with the routes produced by Kool et al. [4]. We notice that when comparing our selected routes to Kool et al.'s selected routes, we find that our policy favors creating fewer routes, while theirs produces more routes. Ultimately, their method produces the better average tour length.

4.2 Generalization of the Method

Thanks to the nature of the framework, we are not limited to the number of nodes in the graph at test time. This means that we can train a model for some limited number of nodes, and evaluate how it will perform in scenarios with more nodes. The other methods we discussed that use policy gradients would not be able to generalize well do to their architectural limitations, but we are able to do that because of the architecture of `structure2vec` does not suffer from those pitfalls. In order to test for generality of the model, we tested various trained versions of the model in a various test scenarios and see how close it comes to optimality. The results are shown in table 2. It compares the values against the average tour lengths when trained and tested on the graph with the same size.

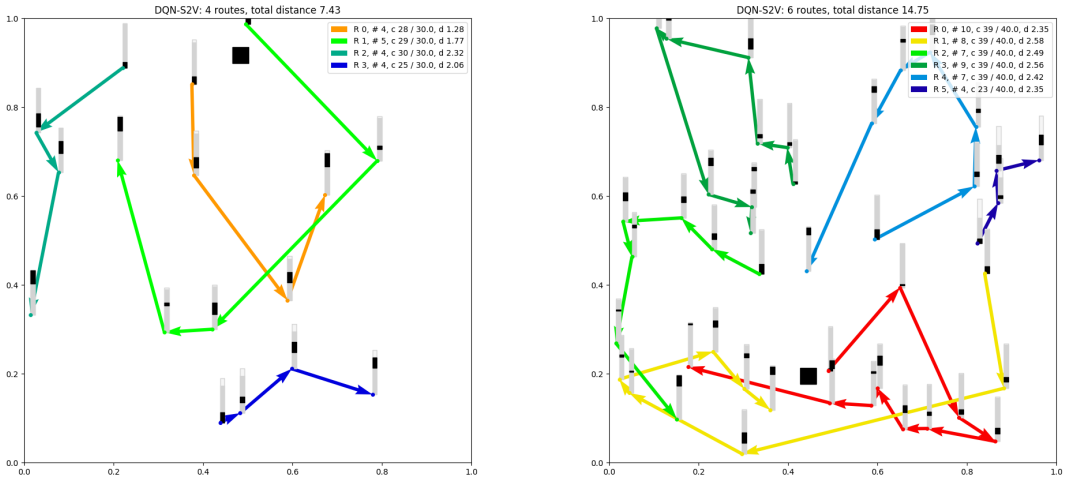

Fig. 2. Here are two sample runs of S2V-DQN(50) on the VRP20Cap30 Test Set (left) and VRP50Cap40 Test Set (right). Edges to and from the depot (the black box) are not drawn for clarity. Legend ordering indicates what order the solution was generated. Each bar is representative of capacity of the vehicle and the black segment represents the demand of the node. As we follow the path a vehicle took, we notice that the black segment moves up showing us how the vehicle provides for the nodes. The white segment (if it exists) represents what portion of the resource was not used in the current route.

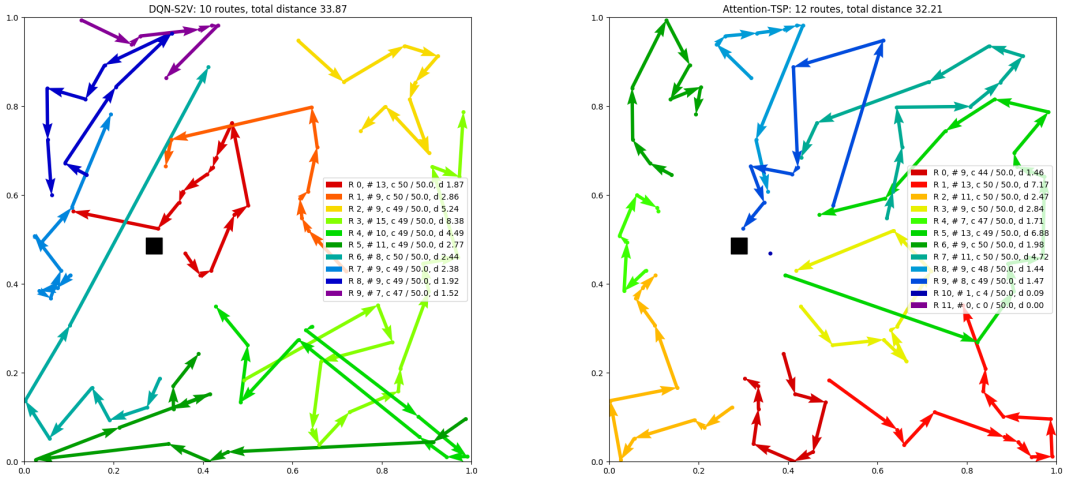

Fig. 3. Here is an instance of the VRP10Cap20 Test Set run on both S2V-DQN (left) and Attention-TSP (right). We can notice that our method tends to favor creating fewer routes, but as a consequence, this makes our overall routes much longer, thus we tend to not be Kool et al. [4]. They manage to create more individual routes, but produce less overall tour cost.

Test Size	10	20	50	100	150	200	250
S2V-DQN(10)	**5.54**	7.86	15.05	24.9	27.73	30.99	34.04
S2V-DQN(20)	5.8	**7.68**	14.09	21.78	26.36	30.08	33.36
S2V-DQN(50)	5.58	7.73	13.75	20.91	25.68	29.44	32.78
S2V-DQN(100)	5.67	7.57	**13.65**	**20.89**	26.09	30.26	32.90

Table 2. S2V-DQN's generalization ability. The last row shows the best result of the model when trained and tested on the same graph size.

5 Conclusions

In this paper, we expand [1]'s work to make it work on the Vehicle Routing Problems. The original framework has really bad performance in that the vehicle tends to go back to the depot much more often than necessary. With the constraint that the vehicle can only go back to the depot when it's necessary, the average tour length is improved but still far from state of the art. We plan to add a penalty term in the loss function to decrease the possibility of such unnecessary actions. On the other hand, we can verify the generality of our method that the size of the training graphs makes little difference on the performance on the testing graphs of different sizes. In the future, we may also test the generalization ability of our method to the other VRP types, such as with dynamic demands and distances.

References

1. Dai, H., Khalil, E., Zhang, Y., Dilkina, B., Song, L.: Learning combinatorial optimization algorithms over graphs. In Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R., eds.: Advances in Neural Information Processing Systems 30. Curran Associates, Inc. (2017) 6348–6358
2. Bello, I., Pham, H., Le, Q.V., Norouzi, M., Bengio, S.: Neural combinatorial optimization with reinforcement learning. (2017)
3. Nazari, M., Oroojlooy, A., Snyder, L.V., Takác, M.: Reinforcement learning for solving the vehicle routing problem. (2018)
4. Kool, W., Welling, M.: Attention solves your tsp, approximately. arXiv preprint arXiv:1803.08475 (2018)
5. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in Neural Information Processing Systems. (2017) 5998–6008
6. Hopfield, J.J., Tank, D.W.: neural computation of decisions in optimization problems. Biological cybernetics **52**(3) (1985) 141–152
7. Smith, K.A.: Neural networks for combinatorial optimization: a review of more than a decade of research. INFORMS Journal on Computing **11**(1) (1999) 15–34
8. Vinyals, O., Fortunato, M., Jaitly, N.: Pointer networks. In: Advances in Neural Information Processing Systems. (2015) 2692–2700
9. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., Weinberger, K.Q., eds.: Advances in Neural Information Processing Systems 27. Curran Associates, Inc. (2014) 3104–3112
10. Dai, H., Umarov, R., Kuwahara, H., Li, Y., Song, L., Gao, X.: Sequence2vec: a novel embedding approach for modeling transcription factor binding affinity landscape. Bioinformatics **33**(22) (2017) 3575–3583
11. Dai, H., Dai, B., Song, L.: Discriminative embeddings of latent variable models for structured data. (2016)
12. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013)

Appendix

Here are some more sample runs using both our policy and the policy generated by Kool et al.'s [4] approach. The graphs on the left are from our model, while the graph on the right is generated from theirs.

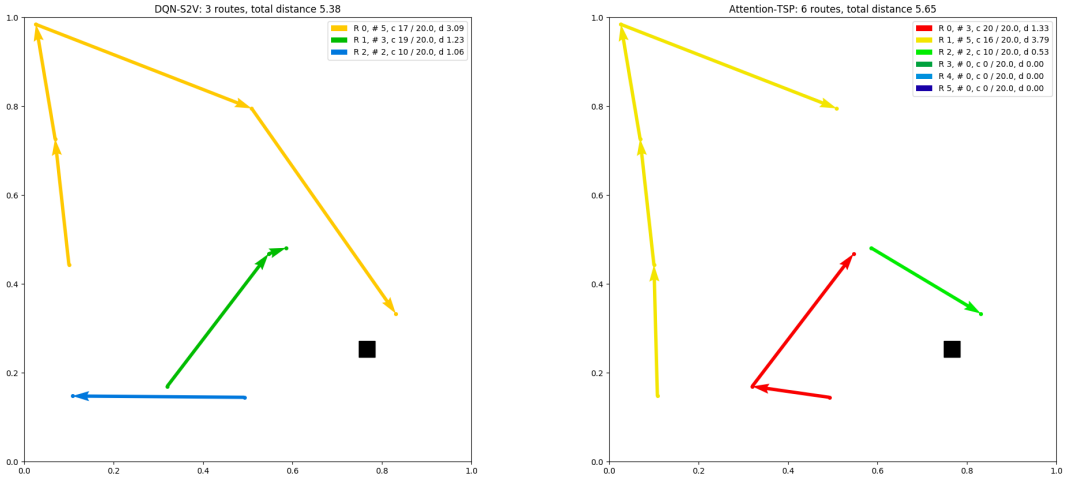

Fig. 4. VRP10

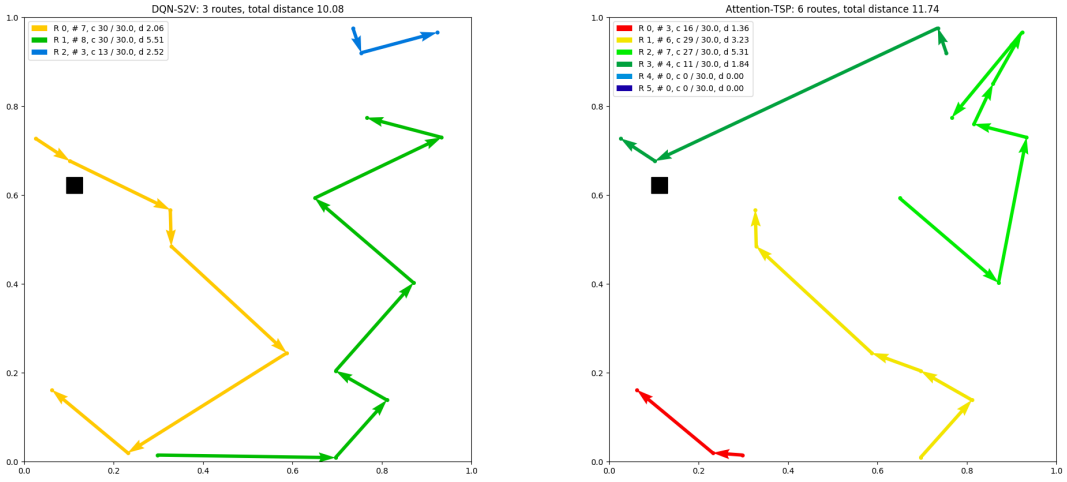

Fig. 5. VRP20

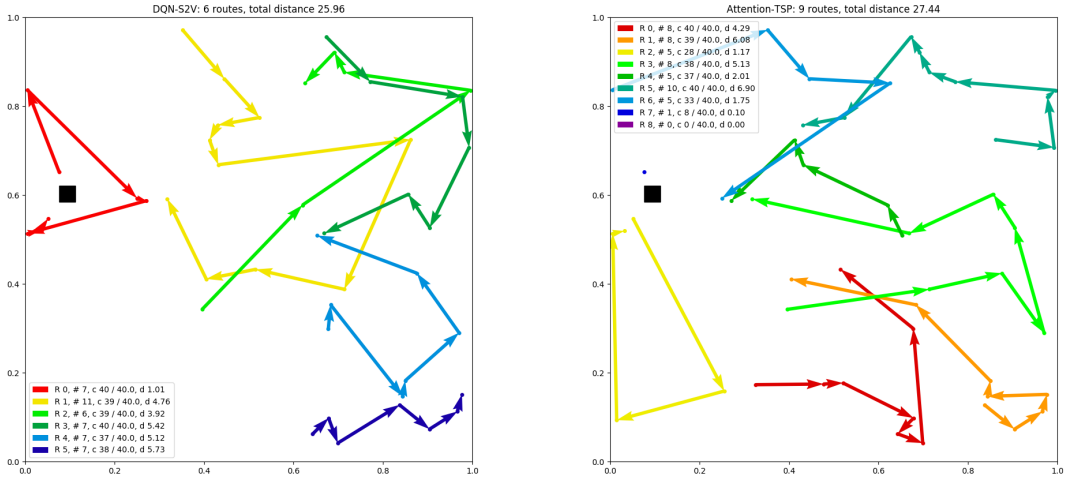

Fig. 6. VRP50

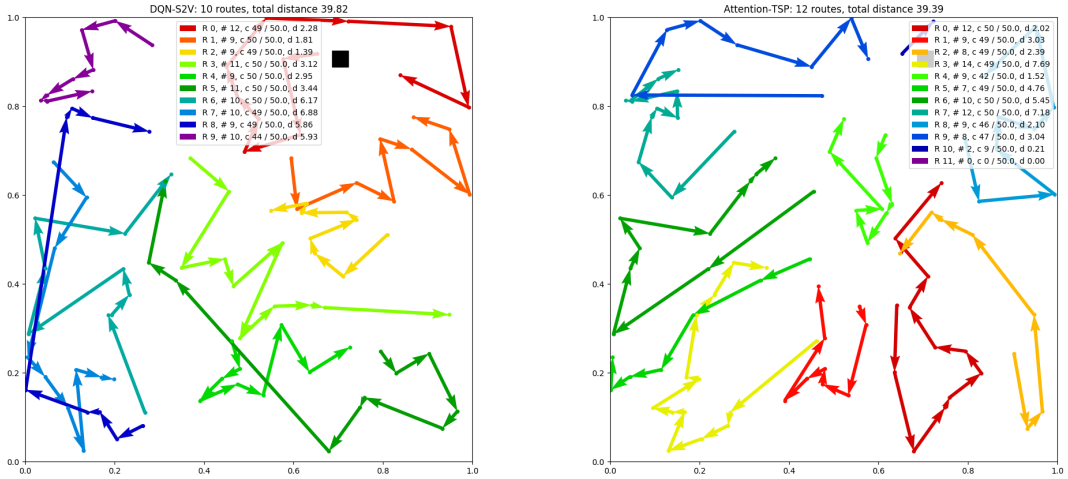

Fig. 7. VRP100