2022 IEEE International Conference on
Systems, Man, and Cybernetics (SMC)

A Deep Averaged Reinforcement Learning Approach for the Traveling Salesman Problem

Sirvan Parasteh
Dept. of Computer Science
University of Regina
Regina, Canada
Sirvanparasteh@uregina.ca

Amin Khorram
Dept. of Industrial Systems Engineering
University of Regina
Regina, Canada
akw126@uregina.ca

Malek Mouhoub
Dept. of Computer Science
University of Regina
Regina, Canada
Malek.Mouhoub@uregina.ca

Samira Sadaoui
Dept. of Computer Science
University of Regina
Regina, Canada
Samira.Sadaoui@uregina.ca

2022 IEEE International Conference on Systems, Man, and Cybernetics (SMC) | 978-1-6654-5258-8/22/$31.00 ©2022 IEEE | DOI: 10.1109/SMC53654.2022.9945274

***Abstract*—This work presents a deep averaged reinforcement-learning approach to learn improvement heuristics for route planning. The proposed method is tested on the Traveling Salesman Problem (TSP). While learning improvement heuristics using machine learning models are prosperous, these methods suffer from low generalization and forgetfulness of the agents during the training process. We have applied the stochastic weight averaging method during the training phase to solve these issues, which smothers the training convergence and prevents the forgetting of optimized learned policies, and consequently provides better results. The agent can learn the optimized policy while holding a moving average of the previously learned policies during the training epochs. In order to assess the performance of our proposed approach, we conducted comparative experiments considering other known methods from the literature. The results demonstrate our proposed method's superiority in training trends and optimization.**

***Index Terms*—Deep Reinforcement Learning, Heuristics, Traveling Salesman Problem (TSP), Attention Mechanism, Recurrent Neural Network, Route Planning.**

I. Introduction

Given a map with a set of cities, the goal of TSP is to find the cheapest tour visiting each city only once. More formally, TSP is a combinatorial optimization problem meeting the Hamiltonian cycle rule while minimizing the total travel cost. This routing problem can be defined as a graph with a set V of n nodes (V=$\{1, \ldots, n\}$), where each node has a feature X(V). The solution for TSP is represented with the sequence S=(s_1,...,s_n) where each s_i is assigned a different value from V [1]. In this work, the main focus is on solving TSP using improvement heuristics. In the literature, the conventional methods to solve TSP include exact and approximation algorithms, and heuristics [2]. Exact methods, such as Branch and Bound, guarantee the optimal solution to be returned [3], but require an exponential running time [4]. Approximation methods such as nature-inspired techniques [5]–[8] trade the running time for the quality of the solution returned. For some challenging TSP instances, the solutions can be of poor quality. Heuristics can solve the problem or approximate the solution more quickly compared to the classic methods [9]. However, the conventional heuristic techniques suffer from substantial trial-and-error, and the quality of the solution is highly dependent on the experience of human experts [10]. There are two types of heuristics in the literature, construction heuristics, and improvement heuristics [1]. The construction approach starts with an initial node and extends it until a complete solution is achieved [11]. This method can lead to a good quality solution but needs additional procedures such as beam search and sampling to achieve high-performance [12]. Known instances of construction heuristics methods include Nearest Neighbor and Insertion Heuristics. The improvement heuristics, on the other hand, starts with a complete solution (achieved by a construction algorithm) and tries to improve it through an iterative process [13]. Some of the most famous improvement heuristics methods are Node-and-Edge-Insertion, 2-Opt Exchange, and Lin-Kernighan Moves. We adopt an improvement heuristic based on the 2-Opt Exchange [14]. Recently, using deep learning to solve the heuristics algorithms has been a growing trend [1]. Conventional heuristics are guided by human intervention and manipulated search policies, which may end up with a slight improvement to the solution [1]. In this work, using deep reinforcement learning, we present an algorithm to learn improvement heuristics directly, which removes human intervention to allow complete automation for all problem instances. The deep reinforcement learning methods in the literature are suffering from two issues which are instability during training and forgetting the highly rewarded policy, which leads to deterioration of performance [15]. In order to show that our method is overcoming these two issues while providing appealing performance and quality solutions, we conducted a set of comparative experiments using known benchmarks [1]. The results show that using stochastic weight averaging not only improves the stability of the model during training but also enhances the model generalization [15].

II. Literature Review

There are many attempts to solve TSP using deep neural networks. In [16], Bello *et al*. used a Recurrent Neural Network (RNN) based on a chain rule of sequence-to-sequence supervised training without using optimal solutions for labeling the training samples in order to solve TSP. The solution was costly, and the outcome was not quite

acceptable. Nazari *et al*. in [17] solved the Vehicle Routing Problem (VRP) using Reinforcement Learning and mapping the input using attention mechanism to a high dimensional vector space but did not achieve a state-of-the-art performance. In [18], Kool *et al*. used reinforcement learning to solve TSP but used the attention mechanism in the encoder and decoder, which led to better performance compared to [17]. Deudon *et al*., in [19], used a graph embedding network and attention mechanism for the encoder; however, the performance depends on a 2-opt local search. In [10] Dai *et al*. used a deep Q-learning network for training a node selection heuristics and the greedy algorithm for optimization to solve TSP on a graph. However, the performance is dependent on the size and type of the graph. All the mentioned articles were based on learning construction heuristics and could outperform the conventional methods, which are not based on learning. However, none of them were successful in reaching a state-of-the-art performance. In [20], Chen *et al*. performed an improvement heuristics using deep reinforcement learning and two policies, which are region picking and rule picking policies, to solve combinatorial optimization problems. The policies are used to predict the rewritten region and solution selection. Nevertheless, because significant policy changes occur in each iteration, the solution is very time costly. Finally, in [1], Wu *et al*. presented a deep reinforcement learning based on a self-attention policy network to implement improved learning for solving TSP and cVRP (Capacitated Vehicle Routing Problem). We will present a modification on the [1] to reach a state-of-the-art performance in a shorter calculation time and a lower variance.

III. Proposed Method

In reinforcement learning, an agent interacts with the environment and learns the optimal policy by trial-and-error decision-making. Each action resulting from a decision will be assessed by a reward function that motivates the agent to improve the learned policies and, consequently, improve its decisions to obtain higher rewards. However, reinforcement learning is not efficient, especially for large-size (high dimensional) problems, and learning the policy become infeasible due to the computational complexity of traditional reinforcement learning algorithms. For example, in Q-Learning, the agent needs to hold the Q-Table in memory as it has to use it to estimate the following best action, as shown in Fig. 1. In a sizeable complex situation, the number of states and actions can increase dramatically, and memorizing the table can become impossible. On the other hand, deep learning has shown its ability to handle high-dimensional inputs by automating the feature extraction and learning the underlying patterns in large unstructured datasets. Deep learning can be combined with RL to handle the complexity of high-dimensional state spaces and make learning optimal heuristic policies more effectively through a gradient descent optimization approach. Deep learning significantly reduces the reliance on domain knowledge [21]. This is viable because the instances of a given problem, while are consisted of different values or even distributions, still share common structures. A good representation of input samples and the agent's reward function enable

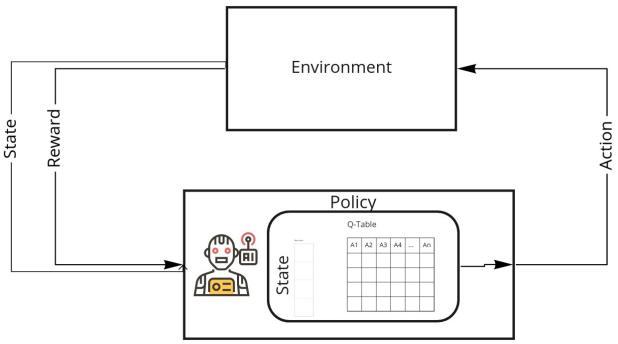

Fig. 1: Q-Learning with a Q-table to calculate the best action

deep learning to discover the relevant features and patterns through a supervised-like approach. Thus dramatically prune state-space complexity and generate policies that outperform human-devised algorithms [22]. Fig. 2 shows the DQN agent in which the Q-Table has been replaced by a network that learns to estimate the Q-function and suggest the following action considering the current state of the environment. This

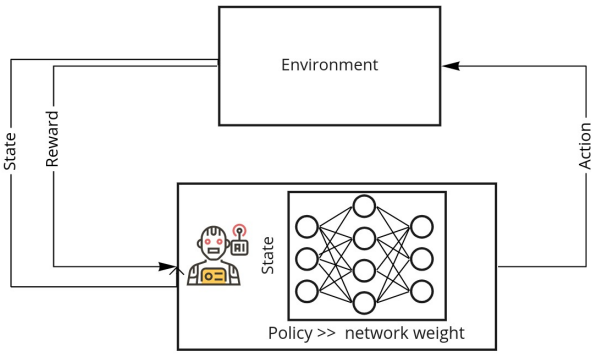

Fig. 2: DQN: Deep Q-Network in which agent optimize its policy by optimizing the network weights

research will use a deep reinforcement learning approach to learn optimal policies for solving TSP. Our work is mainly based on the proposed work in [1] in which we have recognized some drawbacks and provided suggestions to improve the method. We will discuss the base method briefly, name the issues and introduce our approach to solve the issues.

A. Base Method

The algorithm in [1] is selected as the base method, which was published in 2021. In this paper, a deep RL agent learns to solve the routing problem following the process of improvement heuristics. The reinforcement learning formulation of the problem is as below:

- **Goal**: Learning the improvement heuristic policies
- **State**: tuple of TSP instance and a corresponding solution, ordered sequence of nodes, at time step t.
- **Action**: a node pair such (s_i, s_j) from current solution which will be used by 2-opt algorithm to change the current solution.

- **Transition**: applying local operator, 2-opt method, having the selected node pair in action step.
- **Reward**: obtained reward at time step t, that will be calculated using the below equation [1]:

$$r_t = r(s_t, a_t, s_{t+1}) = f(s_t^*) - min\{f(s_t^*), f(s_{t+1})\} \quad (1)$$

  Where s_t^* is known as the finest found solution till step t, and intuitively only if the next solution, s_{t+1}, is better, then the reward is positive; otherwise, it will be zero, which means no progress in the current step [1].
- **policy**: the weights of the deep network designed for selecting the next node pair that is randomly initialized at step $t = 0$ and through an iterative process of the designed reinforcement learning framework will be updated to learn actions that lead to higher rewards.

Considering the above framework, we need to define the structure of the deep learning network and the chosen transition method, the 2-opt operator, which is explained in the next sections.

B. The Node Pair Selection Network

Deep reinforcement learning is utilized to formulate and learn the improvement heuristics to solve TSP. First, we consider the network represented in [1] as a black box and then apply our method to solve the mentioned issues of this approach. The overall structure of the model is shown in Fig. 3. The proposed model follows the actor-critic mechanism where the actor tries to learn to change the order of the nodes in a TSP solution in order to reach a lower cost. At the same time, the critic provides guidance and feedback to the actor to help it learn from its actions. The training phase is an iterative process of a given number of steps involving the actor and critic. Firstly, a new dataset consisting of 10000 TSP samples will be generated at each epoch. A TSP sample is a vector of nodes in a euclidean feature space, while the corresponding solution for TSP is an ordered list of its nodes. For the first epoch, the initialized solutions are generated randomly. During the training, actors and critics learn how to change the initial solutions to minimize the cost of the solutions. The cost of a solution is the path's length goes through all the nodes according to the orders. At the next step, a node embedding layer is applied. Since the order of the nodes is important, along with the node embedding, a positional encoding of the corresponding order of nodes in each TSP solution is required. Thus, to estimate the positional order of the nodes, a sinusoidal position encoding is also added to node embedding. The embedded vectorized inputs are fed into the deep learning-based self-attention layer, followed by a fully connected layer network. The predictions for each input data suggest a pair of nodes, (i, j), that the agent uses for changing the solutions toward improving the solution. If the suggested change improves the results, the agent will receive a positive reward, which is a calculated using the below equation:

$$Reward = max(0, S_{i+1} - S_i) \quad (2)$$

where S_i is the cost of the current solution, and S_{i+1} is the cost of the next solution. Any change offered by the models will be applied regardless of its impact. Thus, there is a chance for getting out of local optima and increase the exploration of the agent and make the process more simple. While the structures for actor and critic network are the same their goals are a bit different. Actor tries to learn the policies to optimize the pair selection for each input, while the critic learns the rewards corresponding to actor's action to guide him improve its actions. Having calculated the rewards and losses accordingly, finally the weights of the networks will be updated using the gradient decent optimization method.

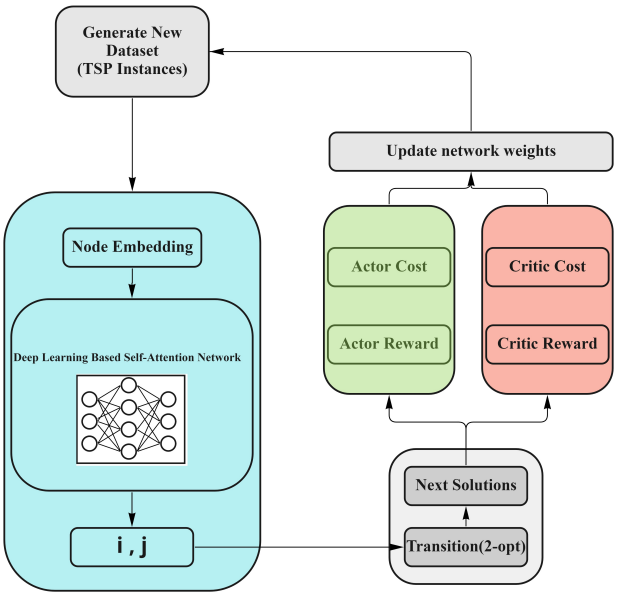

Fig. 3: The overall structure of the network proposed in [1]

C. 2-Opt Local Search

For routing problems, various local search operators are applicable [1]. In this work, we use the most famous operator 2-opt which focuses on pairwise operations. The operator transforms a solution s_t to s_{t+1} by performing an operation K on the node pair (s_{t_i}, s_{t_j}). The operator reverses the node sequence and performs node swap, which exchanges s_{t_i}, s_{t_j} [1]. The 2-opt exchange operates on the routes of individual agents. A basic 2-opt move is shown in Fig. 4 [23].

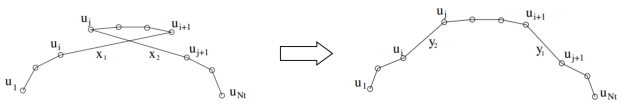

Fig. 4: The 2-opt illustration [23]

D. The Actor Critic Network

An actor-critic algorithm with Adam optimizer is applied. The actor-network is the node pair selection network represented above. The critic is applied to evaluate the cumulative reward at each step. While the critic has the same structure as the actor, there is a subtle difference in their goal. The actor learns to predict the subsequent actions; however, the critic learns to estimate the corresponding rewards of the actor's actions so that it will guide the actor to improve

its heuristic policy. In the meantime, the difference between critic-predicted rewards and the actual reward will be used to calculate the error of the citric and improve its predictions. So, the actor and critic will learn together to update their network weights and optimize them simultaneously.

E. Proposed method

The above framework has shown its ability to learn improvement heuristics effectively. However, deep reinforcement learning models suffer from instability and generalization issues. The average obtained reward of reinforcement learning agent does not monotonically increase during the training where the agent gains more experience. This happens because the agent forgets the highly rewarding policies over time since the training process of reinforcement learning agent is heavy in terms of the number of instances and experience it requires to learn. On the other hand, as the learning agent is trained using a whole new training set on each iteration, the possibility of forgetting the optimized rewarding policy increase due to the distribution difference and natural presence of noise [15].

Forgetting and training cycle on recent data, in turn, can affect the generalization of the model and reduce the model performance on the unseen instances [24]. This issue has been shown in Fig. 5 reporting on the training of the base model where the convergence is not stable, and showing that the agent forgets the learned optimal policies. More precisely, the average validation cost, shown in (a), and the average reward fluctuate as the training proceeds.

Algorithm 1 Our Proposed Algorithm: SWA_DRL TSP Solver

initialize:
$swa_{start} = k$, where k is given by expert
initial model weight, W
initial $epoch_{size}, batch_{size}, lr$
initial $W_{swa} \leftarrow W$
initial optimizer
$N_{swa} \leftarrow 1$
Start Training:
for epoch in range($epoch_{size}$) **do**
    $W \leftarrow train_model(data, W, epoch, lr, optimizer)$

    if then$epoch \leq swa_{start}$:
        $W_{swa} \leftarrow \frac{W_{swa} + (W - W_{swa})}{N_{swa} + 1}$
        $N_{swa} \leftarrow N_{swa} + 1$
Normalize W_{swa}**:**
batch_norm(data, W_{swa})
persist_model(W_{swa}

In order to handle these issues in our proposed method, we have applied a stochastic weight averaging method suggested in [15], [24] to improve the results of the benchmark model. In this method, after a predefined number of training steps, named swa_{start}, the weight averaging process starts. In training, we will utilize two distinct models. One is the primary model and participates in the optimization process through the training to calculate the subsequent optimized weights, W. Another model is used to hold the calculated average weights during the process. Our proposed algorithm can be observed above. The weight averaged model will not participate in the training process. However, the weights of this model need an extra training step that consequently normalized the weights along forwarding the data into the model. Having normalized the weights, we have tested the model and shown in the experimental study that the proposed method can reach a better performance while staying stable during the training process.

IV. Experimentation

We used a system of 64GB RAM, Core I9 CPU, and NVIDIA GPU RTX-3060. The termination steps during the training have been set to 200 due to the long period of training, while we let the agent improve the solution over 1000, 3000, and 5000 steps during the evaluation phase, which is considerably faster.

A. Dataset

The training dataset is populated with 15000 randomly generated instances of euclidean-based TSP with a graph size of 20 nodes. The batch size is 110, and the termination steps are 1000, 3000, and 5000 steps. A new dataset, based on the 10,000 TSPs reported in [1], has been generated on each training step to let the agent experience new data to achieve generalization. This dataset is also used to test traditional algorithms introduced in the next section.

B. Discussion

Table 1 compares the results of our proposed model(Bolted) with several metaheuristics (Nearest Neighbor, Simulated Annealing, Ant colony, Genetic Algorithm , Divide-and-Conquer , and LKH) and the results of the base article. The used metrics are the average cost, the variance of the costs and consumed time for solving TSPs. The parameters of the mentioned methods are provided in the next section. As shown in Table.1, our proposed method outperformed the the base article, named as DRL-LIH in the comparisons, and most of the benchmark methods.

Also the stability of the proposed method has been shown in the Fig. 6. In the diagram shows, the blue line represents our method which shows the stability during the training process and tends to utilize and retain optimized policies. In contrast, the base method, represented by the red line, fluctuates from the optimal policies. The drop in the obtained rewards and the increase in the solution cost are also observed in this model.

1) Nearest Neighbor: The nearest neighbor is a straightforward algorithm. The decision rule assigns to an unclassified sample point and leads to the classification of the nearest of the previously classified points [25]. Using this algorithm, we achieved a mean cost of 4.4874, a variance of 0.291, and a calculation time of 11'19".

2) Simulated Annealing: Simulated annealing is a probabilistic technique for approximating the global optimum. The algorithm is inspired by annealing in metallurgy, which is the same as hill-climbing, except instead of picking the best move, it picks a random move. If the selected move improves the solution, it is accepted. Otherwise, the algorithm makes a move with some probability less than 1. The probability

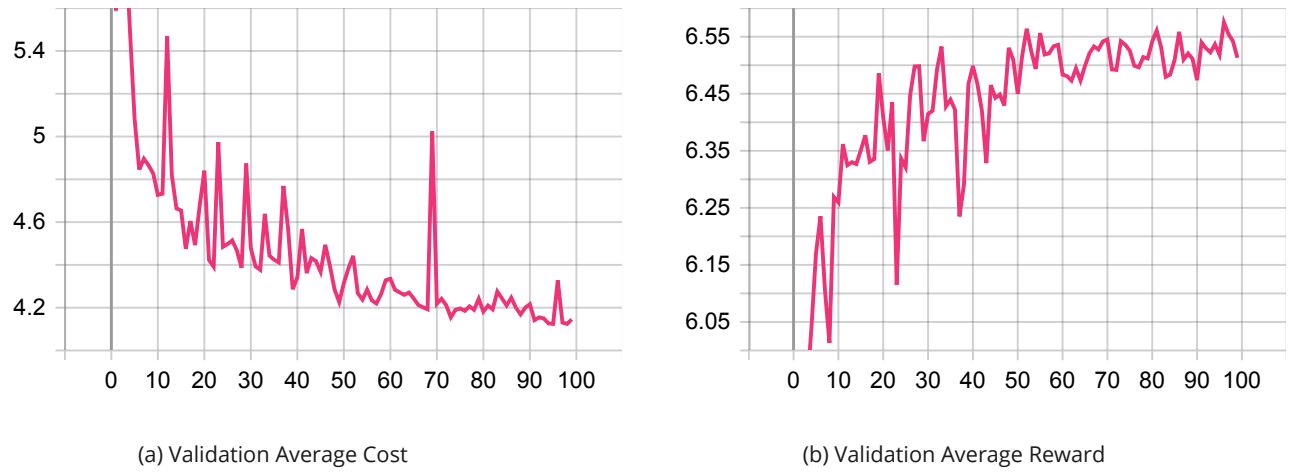

Fig. 5: Instability in improving the averaged cost(a) and averaged reward (b) through training steps.

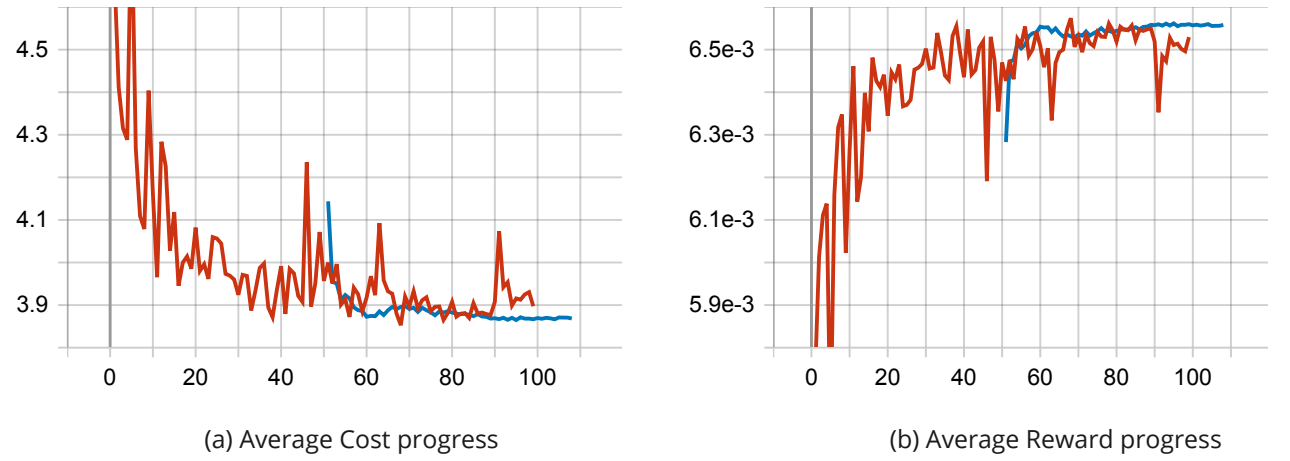

Fig. 6: Stability of the averaged solution cost(a) and averaged obtained reward (b) through training steps.

decreases exponentially with the badness of the move [26]. Using the Simulated Annealing method, we reached the mean cost of 3.919, the variance of 0.117, and the calculation time of 20'58".

3) Ant Colony: And-Colony is a probabilistic technique for solving combinatorial optimization problems based on finding a good path through graphs. Using this algorithm, we reached the mean cost of 3.8749, the variance of 0.106, and the calculation time of 5'32".

4) Genetic Algorithm: Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on the natural selection process. Using this algorithm, we reached the mean cost of 5.688, the variance of 0.304, and the calculation time of 15'27".

5) Divide-and-Conquer: The idea is to recursively break the original problem into smaller sub-problems that are easier to solve. This technique has three steps: 1. Divide: breaking the problem (or feature space) into smaller sub-problems (sub-spaces). 2. Conquer: solving sub-problems by recursively calling them. 3. Combine the sub-problems to find the final solution of the whole problem [27]. Using this algorithm, we achieved the mean cost of 4.1936, the variance of 0.1283, and the calculation time of 29".

6) LKH-3: LKH is a practical implementation of the Lin-Kernighan heuristic for solving the traveling salesman problem [28]. The Lin-Kernighan is a local optimization algorithm and is specified in terms of exchanges or moves that can convert one tour into another. From the beginning state, the algorithm repeatedly performs the exchanges that reduce the length of the current tour till the point which finds a tour where no more improvement is possible. The initial tour is generated randomly, but the convergence is guaranteed [28]. Using this algorithm, we achieved the mean cost of 3.83

and the calculation time of 42".

TABLE I: Experimental Results

Method	Results		
	mean cost	***Variance***	***Calculation Time(s)***
Nearest Neighbor	4.4874	±0.291	11min 19s
Simulated Annealing	3.919	±0.117	20min 58s
Ant Colony	3.8749	±0.106	5min 32s
Genetic Algorithm	5.688	±0.304	15 min 27s
Divide-and-Conquer	4.1936	±0.1283	29s
LKH	**3.83**	-	42s
DRL-LIH [T=1000]	3.8399	±0.0098	3min 6s
Our-Method [T=1000]	**3.8369**	±0.0097	3min 7s
DRL-LIH [T=3000]	3.8325	±0.0097	8min 38s
Our-Method [T=3000]	**3.8302**	±0.0096	8min 36s
DRL-LIH [T=5000]	3.8307	±0.0096	13min 55s
Our-Method [T=5000]	**3.8291**	±0.0096	14min 1s

V. Conclusion and Future Work

We have proposed a new deep reinforcement algorithm with a stochastic weight averaging mechanism for solving the TSP. Having tested our algorithm on a publicly available dataset of 10,000 TSP instances, we have shown that our method not only solved the known issues of the previous methods, including instability and forgetting the optimized policies, but also improved generalization and achieved better performance in terms of lower-cost solutions and higher average obtained rewards. In the near future, we will explore new methods to tackle larger TSP instances. One idea is to consider transfer learning following a divide-and-conquer paradigm. We believe that this new approach will enable to formulate the problem as a multi-agent system in which agents will obtain more generalized policies by sharing their weights while requiring lower training cost. We will also investigate TSPs in a dynamic environment, in case the optimal route is no longer feasible due to unforeseen events such as accidents or severe weather conditions, we need to find the next best feasible route. We will rely on our dynamic vehicle routing algorithms [29] and dynamic constraint solving techniques [30], [31].

References

[1] Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. Learning improvement heuristics for solving routing problems.. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[2] Gregory Gutin and Abraham P Punnen. *The traveling salesman problem and its variations*, volume 12. Springer Science & Business Media, 2006.

[3] Stefan Poikonen, Bruce Golden, and Edward A Wasil. A branch-and-bound approach to the traveling salesman problem with a drone. *INFORMS Journal on Computing*, 31(2):335–346, 2019.

[4] Franz Rothlauf. *Design of modern heuristics: principles and application*, volume 8. Springer, 2011.

[5] Wael Korani and Malek Mouhoub. Discrete mother tree optimization for the traveling salesman problem. In *International Conference on Neural Information Processing*, pages 25–37. Springer, 2020.

[6] Urszula Boryczka and Krzysztof Szwarc. The harmony search algorithm with additional improvement of harmony memory for asymmetric traveling salesman problem. *Expert Systems with Applications*, 122:43–53, 2019.

[7] Eneko Osaba, Javier Del Ser, Ali Sadollah, Miren Nekane Bilbao, and David Camacho. A discrete water cycle algorithm for solving the symmetric and asymmetric traveling salesman problem. *Applied Soft Computing*, 71:277–290, 2018.

[8] Divya Gupta. Solving tsp using various meta-heuristic algorithms. *International Journal of Recent Contributions from Engineering, Science & IT (iJES)*, 1(2):22–26, 2013.

[9] Judea Pearl. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc., 1984.

[10] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017.

[11] Edmund Burke, Graham Kendall, Barry Mccollum, and Paul Mcmullan. Constructive versus improvement heuristics: an investigation of examination timetabling. 01 2007.

[12] Paulo da Costa, Jason Rhuggenaath, Yingqian Zhang, Alp Akcay, and Uzay Kaymak. Learning 2-opt heuristics for routing problems via deep reinforcement learning. *SN Computer Science*, 2(5):1–16, 2021.

[13] Víctor Pacheco-Valencia, José Alberto Hernández, José María Sigarreta, and Nodari Vakhania. Simple constructive, insertion, and improvement heuristics based on the girding polygon for the euclidean traveling salesman problem. *Algorithms*, 13(1):5, 2019.

[14] Paulo R d O Costa, Jason Rhuggenaath, Yingqian Zhang, and Alp Akcay. Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning. In *Asian Conference on Machine Learning*, pages 465–480. PMLR, 2020.

[15] Evgenii Nikishin, Pavel Izmailov, Ben Athiwaratkun, Dmitrii Podoprikhin, Timur Garipov, Pavel Shvechikov, Dmitry Vetrov, and Andrew Gordon Wilson. Improving stability in deep reinforcement learning with weight averaging. In *Uncertainty in artificial intelligence workshop on uncertainty in Deep learning*, 2018.

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

[17] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takác. Reinforcement learning for solving the vehicle routing problem. *Advances in neural information processing systems*, 31, 2018.

[18] Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.

[19] Michel Deudon, Pierre Cournut, Alexandre Lacoste, Yossiri Adulyasak, and Louis-Martin Rousseau. Learning heuristics for the tsp by policy gradient. In *International conference on the integration of constraint programming, artificial intelligence, and operations research*, pages 170–181. Springer, 2018.

[20] Xinyun Chen and Yuandong Tian. Learning to perform local rewriting for combinatorial optimization. *Advances in Neural Information Processing Systems*, 32, 2019.

[21] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[22] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

[23] Justin Jackson, Anouck Girard, Steven Rasmussen, and Corey Schumacher. A combined tabu search and 2-opt heuristic for multiple vehicle routing. In *Proceedings of the 2010 American Control Conference*, pages 3842–3847. IEEE, 2010.

[24] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

[25] TM Cover and PE Hart. Nearest neighbor pattern classification. ieee transactions on information theory. *IT-13*, pages 19–7, 1967.

[26] Saul A Teukolsky, Brian P Flannery, WH Press, and WT Vetterling. Numerical recipes in c. *SMR*, 693(1):59–70, 1992.

[27] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

[28] Keld Helsgaun. An effective implementation of the lin–kernighan traveling salesman heuristic. *European journal of operational research*, 126(1):106–130, 2000.

[29] Eisa Alanazi, Malek Mouhoub, and Mahmoud Halfawy. A new system for the dynamic shortest route problem. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pages 51–60. Springer, 2017.

[30] Malek Mouhoub. Arc consistency for dynamic csps. In *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, pages 393–400. Springer, 2003.

[31] M. Mouhoub and A. Sukpan. A new temporal csp framework handling composite variables and activity constraints. In *17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'05)*, pages 7 pp.–149, 2005.