Deep Reinforcement Learning for Stochastic Orienteering Problem with Metaheuristic Enhanced Experience Replay

Kai Zhuo Lim
School of Computing and Information Systems
Singapore Management University
Singapore, Singapore
kz.lim.2023@mitb.smu.edu.sg

James Koh
School of Computing and Information Systems
Singapore Management University
Singapore, Singapore
jameskohby@smu.edu.sg

Aldy Gunawan
School of Computing and Information Systems
Singapore Management University
Singapore, Singapore
aldygunawan@smu.edu.sg

Cédric Verbeeck
Department of Strategy
EDHEC Business School
Roubaix, France
cedric.verbeeck@edhec.edu

Bing Tian Dai
School of Computing and Information Systems
Singapore Management University
Singapore, Singapore
btdai@smu.edu.sg

***Abstract*—In this work, we address the Stochastic Orienteering Problem with Time Windows (SOPTW) using an enhanced reinforcement learning (RL) framework. We employ a Quantile Regression Double Deep Q-Network (QR-DDQN) with Set Transformers for graph representation to capture the problem's stochastic nature. To improve learning efficiency, we introduce a Metaheuristic Enhanced Experience Replay (MEER) mechanism that incorporates metaheuristic operators to generate diverse experiences. Results show that the RL agent achieves higher rewards by adapting in real time to stochastic variations, outperforming benchmark methods and standard RL baselines. We also analyze the underlying mechanisms to explain the observed performance gains.**

***Index Terms*—deep reinforcement learning, stochastic orienteering problem, time windows, experience replay, metaheuristics, QR-DDQN**

I. Introduction

Combinatorial Optimisation Problems (COPs) are widely applied in logistics, scheduling, and resource allocation [1]. Among these, routing problems such as the Orienteering Problem (OP) [2] is particularly significant. The OP seeks a path that maximises total reward within a budget constraint (e.g., time or distance). Its many variants effectively model real-world scenarios such as trip planning, vehicle routing, surveillance, and logistics [3]. Traditional methods for solving the OP and its variants typically use exact algorithms for small instances and metaheuristics for larger ones. Metaheuristics, such as Adaptive Large Neighbourhood Search (ALNS) [4], have been successfully applied. However, they are usually offline, generating complete solutions before execution [5]. This limits their effectiveness in dynamic environments where travel times or rewards may change.

Deep Reinforcement Learning (DRL) has become a powerful approach for sequential decision-making and has shown strong results in routing problems [6], [7]. DRL agents learn by interacting with the environment to maximise cumulative rewards. Off-policy methods, which use experience replay, improve sample efficiency by learning from diverse data.

This paper investigates the use and enhancement of off-policy RL for the SOPTW, addressing challenges in representing problem structure, handling stochastic rewards and costs, managing time windows, and improving exploration efficiency. The goal is to develop an RL-based framework for online, adaptive decision-making in dynamic environments as an alternative to traditional offline methods. Our contributions are threefold. First, we extend classical RL by incorporating set transformers for graph embeddings to better capture the stochastic nature of SOPTW. Second, we propose Metaheuristic Enhanced Experience Replay (MEER), which integrates metaheuristic operators to generate diverse experiences and improve learning efficiency. Finally, we show that RL agents can achieve higher rewards by adapting in real time to stochastic changes rather than relying on precomputed fixed paths.

A core variant of the OP is the OP with Time Windows (OPTW), which has been widely studied. Numerous heuristics and metaheuristics have been developed to obtain high-quality, near-optimal solutions within reasonable computational times, especially for solving large instances. Many metaheuristic methods are inherently offline, computing complete paths before execution [5]. This limits their effectiveness in dynamic or stochastic settings, where parameters such as travel times or rewards may change, making real-time adaptation difficult without costly re-planning. The challenge is even greater for stochastic variants like the SOPTW [3], where handling uncertainty during execution remains a key issue.

Off-policy RL algorithms, which learn from data generated by a different behaviour policy, are generally more sample-

efficient than on-policy methods [7]. A core component is experience replay, as used in Deep Q-Networks (DQN), where past experiences (tuples of state, action, reward, next state) are stored and sampled to improve learning efficiency and reduce data correlation [6]. While widely adopted in routing problems, most existing approaches rely on episodes generated by the RL agent and store them directly in the replay buffer without further modification. At the time of writing, no documented approach leverages metaheuristics to enhance replay buffer content based on agent-generated data.

Some works combine metaheuristics with RL or ML, e.g., to generate training solutions [5], guide evolutionary algorithms [8], or refine ML-generated solutions via local search [9]. Unlike these, our approach implements a feedback loop where RL-generated trajectories are enhanced with metaheuristics and reintegrated into the replay buffer.

II. Problem and Background Definitions

The SOPTW can be formally defined on a graph $G = (V, E)$, where V is a set of vertices representing potential locations, with start vertex $v_s \in V$ and an end vertex $v_f \in V$. Each vertex $v_i \in V$ has an associated reward r_i. Each edge $(v_i, v_j) \in E$ has a traversal cost $c_{i,j}$ which in the SOPTW is a random variable. Additionally, each vertex v_i has a time window $(e_i,\ l_i)$, specifying the earliest and latest time at which the agent can receive the associated reward r_i from the location, and it already takes into account the service time (defined later). If the agent arrives before the opening hours, it simply waits until e_i, and we still term that as the arrival time t_i. If the agent misses the opening hours, there is a penalty incurred p_i. The service time s_i is the duration spent by the agent at the visited location within the opening hours. The problem also includes a total travel time budget B.

A solution to the SOPTW is represented as a path $P = (v_s, v_{i_1}, v_{i_2}, \ldots, v_{i_k}, v_f)$, starting at v_s and finishing at v_f. Let $\mathbf{1}\{\cdot\}$ be the indicator function, the total collected reward is

$$R(P) = \sum_{v_i \in P} r_i \cdot \mathbf{1}\{t_i \in [e_i, l_i]\}, \tag{1}$$

The total travel cost as defined in Eq. (2) is the sum of the realised stochastic edge traversal costs along the path and service time for nodes visited within opening hours.

$$C(P) = \sum_{(v_i, v_j) \in P} c_{i,j} + \sum_{v_i \in P} s_i. \tag{2}$$

A feasible path P must satisfy both Eqs. (3) and (4). A path must satisfy the time window constraints for all visited vertices. We allow violations to the budget by introducing a lateness penalty in Eq. (5) [3].

$$t_{i+1} \geq t_i + c_{i,i+1} + s_i, \quad \forall (v_i, v_{i+1}) \in P, \tag{3}$$

$$t_i \in [e_i, l_i] \quad \text{for reward collection at } v_i. \tag{4}$$

$$\text{Penalty}(P) = -\alpha \left(\max\{0,\ C(P) - B\} \right)^2. \tag{5}$$

The objective is to find a path P that maximises the expected net reward

$$\max_P \ \mathbb{E}\Big[R(P) + \text{Penalty}(P) \Big]. \tag{6}$$

Algorithm 1 Deep Q-Learning with Experience Replay

1: Initialize network Q
2: Initialize target network $\hat{Q}$
3: Initialize replay memory D
4: Initialize the *Agent* to interact with the Environment
5: **for** episode = 1 to N_T **do**
6: **/* (1) Collecting Experience */**
7: $\epsilon \leftarrow$ set new epsilon with decay
8: Choose a from s using ϵ-greedy or Boltzmann exploration
9: From s, act a, observe r and s', store experience tuple $(s, a, r, s', \text{done})$ in replay buffer D
10: If episode terminates with reward within top-N episodes, add the episode into the top-N episode heap.
11: **if** enough experiences in D **then**
12: **/* (2) DQN Training */**
13: Sample minibatch of N transitions from D
14: **for all** $(s_i, a_i, r_i, s'_i, \text{done}_i)$ **do**
15: **if** done_i **then**
16: $y_i \leftarrow r_i$
17: **else**
18: $y_i \leftarrow r_i + \gamma \max_{a'} \hat{Q}(s'_i, a'; \theta^-)$
19: **end if**
20: **end for**
21: $\mathcal{L} \leftarrow \frac{1}{N} \sum_{i=0}^{N-1} \left(Q(s_i, a_i; \theta) - y_i \right)^2$
22: Update Q by minimizing $\mathcal{L}$ (SGD)
23: Every C steps, copy weights $Q \rightarrow \hat{Q}$
24: **end if**
25: **/* (3) MEER Augmentation */**
26: **if** enough experiences in D and every N_M episodes to apply MEER **then**
27: Sample 1 (for intra-MEER) or 2 (for inter-MEER) episodes from the top-N episodes heap.
28: Sample an intra/inter-MEER operator using roulette wheel selection
29: Apply MEER operator on sampled top-N episode
30: Insert the MEER modified episode into replay buffer D
31: **end if**
32: **end for**

III. Methodology

Reinforcement learning involves an agent interacting with an environment by observing a state and selecting actions based on a policy. Each action yields a reward and transitions the agent to a new state, and this process repeats until a terminal state is reached. This interaction is illustrated in the innermost (blue) layer of the MEER framework in Fig. 1. To address the challenges of the SOPTW environment, we incorporate several enhancements, including QR-DDQN, Boltzmann search, action masking, and a novel feature extractor using set transformers and Q,K,V projection layers to encode the graph features. The underlying codes, hyperparameters used to obtain the results, and mathematical notations are shared on Github for reproducibility.

1) State Representation: The agent selects the next node based on node-level features, including binary indicators for visited status, current position, and terminal node identity, as well as time window constraints and rewards for each location. In addition, it incorporates first- and second-degree estimates of the remaining time budget and travel time variance for each candidate node, enabling it to evaluate both feasibility and stochastic uncertainty.

Based on the experiences collected (line 10 of Algorithm 1), sample experiences are drawn from the replay buffer to

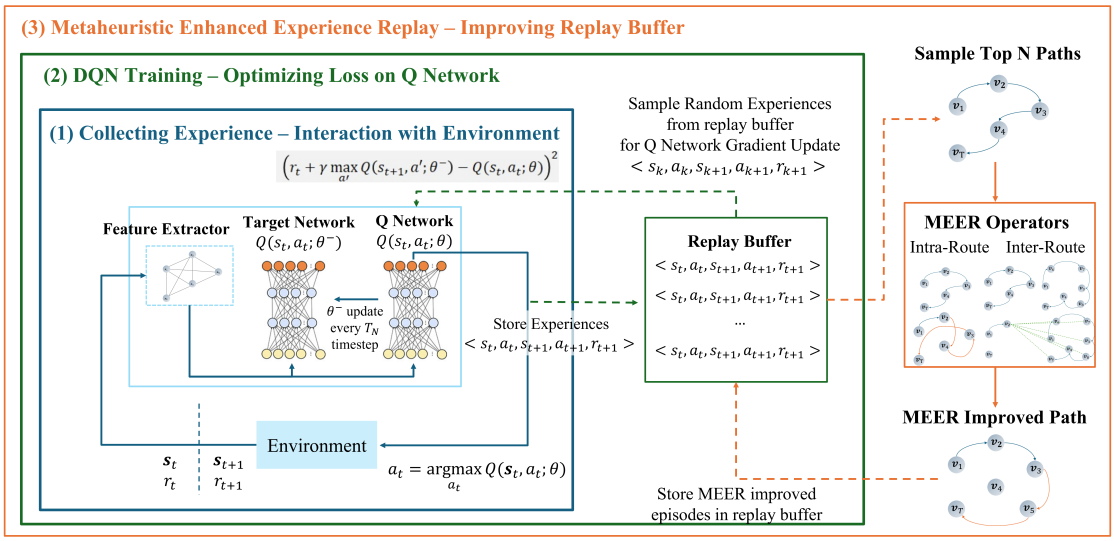

Fig. 1. MEER RL Framework.

optimize the loss on the Q network (2nd green layer in Fig. 1; lines 11 to 24 of Algorithm 1). After every 10,000 gradient step, the parameters of the Q network are copied over to the target network through a hard update. In off-policy methods epsilon-greedy, during the collection of experience (blue inner 1st layer), the agent acts optimally according to its policy most of the time with a small probability to perform a random exploratory action. We introduce MEER (3rd layer in Fig. 1; lines 25 to 31 of Algorithm 1) which aims to perform effective exploration by augmenting the collected experience with metaheuristic operators to increase the likelihood of finding a better path with higher cumulative rewards. Under MEER, the mutated action sequence is re-executed in a dedicated environment instance, where stochastic travel times are sampled according to the same dynamics.

To apply deep neural networks to graph problems, effective contextual embeddings are required to encode graph information into fixed-size vectors. In this feature extraction process, we use a dense fully connected graph, where each node is represented by 16 features (Appendix - Table **??**).

Although Set Transformers are permutation-invariant, node identity and position remain important. We address this by adding learned positional embeddings to each node's features, allowing the model to distinguish nodes—crucial for routing tasks. The DQN then outputs a Q-value per node; without positional encoding, inputs would be indistinguishable. The Set Transformer [10] processes unordered node features via Set Attention Blocks (SABs), enabling the agent to capture global graph features and node interactions independent of input order. Node features with positional encodings are projected through Q, K, V linear layers into high-dimensional vectors, allowing richer representations. SAB outputs are max-pooled to produce a fixed-size graph embedding, which is passed to the RL agent's policy and target networks, resulting in an expressive, order-invariant architecture.

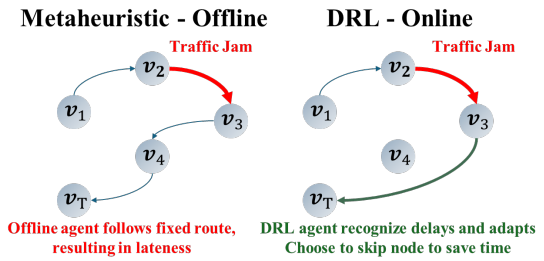

Fig. 2. Example of Offline Metaheuristics vs Online DRL

In many metaheuristic approaches to OP, a complete route is first initialized and then iteratively improved using operators. In contrast, our RL approach constructs the solution sequentially, allowing the agent to select one action at a time starting from the start node. The trained RL agent uses its policy network to decide on the best action to take (the next node to go to) based on $\arg\max_a Q(s, a; \theta)$. For example, if the agent had exceptionally long travel times from v_2 to v_3 (Fig. 2), the budget remaining will be lower and the agent can make use of this information to omit node v_4 and proceed directly to v_T to reach the terminal node in time. This is opposed to constructing a full solution then forcing the agent to stick to the planned route,in which case the agent might be late or the full solution has to be more conservative to allow for an acceptable buffer.

2) RL Agent Formulation and Training: Reinforcement Learning (RL) is a machine learning paradigm where an agent learns optimal decisions through trial and error by interacting with an environment modeled as a Markov Decision Process (MDP) [11]. The MDP is defined by the following tuple, where S is the set of states, A is the set of actions, $P(s_{t+1} \mid s_t, a_t)$:

$S \times A \times S \to \mathbb{R}_+$ is the transition probability distribution, μ_0 is the distribution of initial states, $R(s_t, a_t) : S \times A \to \mathbb{R}$ is the reward function, $\gamma \in [0, 1]$ is the discount factor, and T is the horizon.

$$\mathcal{M} = \langle S, A, P, \mu_0, R, \gamma, T \rangle. \tag{7}$$

The action space is discrete, representing the choice of the next node, with visited nodes masked out. Rewards (Section III-3rd paragraph), granting node rewards if visited within time windows and applying penalties for constraint violations. The RL agent acts in $\mathcal{M}$ by following its policy $\pi(a \mid s; \theta)$ which denotes the probability of taking action a when in state s. The objective is to learn an optimal policy [12] that maximizes the expectation of the Q-value over the initial state and the trajectory when following the policy as such:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{s \sim \mu_0^t,\, a \sim \pi} \left[Q^{\pi}_{\mathcal{M}}(s, a) \right] \tag{8}$$

Deep Q-Network (DQN) [13] approximates the state–action value $Q(s, a; \theta)$ by minimizing temporal-difference (TD) error, but tends to overestimate values since the same network is used for action selection and evaluation. Double DQN (DDQN) [14] addresses this by decoupling the two, using separate networks. However, both methods estimate only expected returns, which is limiting in stochastic settings like SOPTW. To capture uncertainty, Quantile Regression DQN (QR-DQN) [15] extends DDQN by modeling the full return distribution via quantile regression, allowing the agent to account for variance and make more robust decisions. To better balance exploitation and exploration in a large action space, we used a hybrid ϵ-greedy and Boltzmann exploration policy to improve sample efficiency in structured domains by biasing exploration toward promising regions of the action space [16]. Lastly, action masking is applied to prevent the agent from revisiting nodes to improve training stability and convergence [17]–[19].

Training the QR-DDQN involves optimizing the loss on the Q-network (second green layer in Fig. 1) using a similar training process to the standard Deep Q-Network (DQN). Here, the neural network θ is trained by minimizing the temporal difference (TD) loss:

$$L(\theta) = \left(y - Q(s, a; \theta) \right)^2 \tag{9}$$

and the TD target y uses the target network θ^-:

$$y = R + \gamma Q \left(s, \arg\max_{a} Q(s, a; \theta); \theta^- \right) \tag{10}$$

However, our architecture differs from the original QR-DQN. We replace CNNs with a set transformer for feature extraction, incorporate action masking, adopt a hybrid epsilon-greedy and Boltzmann exploration strategy, and introduce MEER for experience replay. As the agent interacts with the environment, it collects experiences stored as tuples:

$$(s_t, a_t, r_{t+1}, s_{t+1}) \tag{11}$$

where s_t is the state, a_t is the action taken, r_{t+1} is the reward received, and s_{t+1} is the next state.

Algorithm 2 Intra-Path Metaheuristic Mutation

1: **Procedure** IntraMutatePath($path, env\_idx, operation$)
2: Copy $path$ into new_path
3: **if** operation = swap **then**
4: Select two internal nodes and exchange their positions
5: **else if** operation = reverse **then**
6: Select a subsequence of nodes and reverse its order
7: **else if** operation = insert **then**
8: Remove one internal node and reinsert it at another position
9: **else if** operation = Destroy & RL-Reconstruct **then**
10: Truncate the path at a random node, then use the RL policy to rebuild the remainder based on $\arg\max_{a_t \in \Omega_{nv}} Q(s_t, a_t; \theta^-)$. RL policy can choose any node in Ω_{nv} which are nodes that has not been visited yet.
11: **end if**
12: **return** new_path

These experiences are stored in the experience replay buffer ($\mathcal{D}$). During training, mini-batches of experiences are sampled uniformly from $\mathcal{D}$ to update the Q-network θ parameters via stochastic gradient descent (SGD). The use of experience replay improves data efficiency and breaks temporal correlations, stabilizing the training process [13].

3) MEER: We introduce one of the central innovations of this work, the Metaheuristic Enhanced Experience Replay (MEER) mechanism, designed to improve the quality and diversity of experiences used for off-policy training. Traditional off-policy exploration methods like epsilon-greedy rely on random action selection, which can be inefficient. MEER improves this by using metaheuristics to guide exploration more effectively. Every N_M training iterations, MEER selects high-quality paths from past exploration and refines them using metaheuristic operations. The enhanced routes are added back to the replay buffer, where they help accelerate learning. This creates a feedback loop between the RL agent and the metaheuristic module.

To enhance replay buffer episodes, MEER employs several techniques. First, adaptive operator selection chooses metaheuristic operators based on success rates using softmax sampling, allowing flexibility across problems. Second, it adapts to different training stages using exponentially weighted moving averages to prioritize recently effective operators via a temperature-controlled softmax (roulette wheel). Finally, it leverages a diverse set of intra- and inter-route operators. The MEER framework uses two classes of metaheuristic operators—intra-route and inter-route—to modify existing trajectories. Applied to top-performing paths from the replay buffer, they generate diverse yet high-quality trajectories for replay augmentation.

Intra-route operators adjust the order of node visits within a single tour. Classical operators such as reverse, swap, and insert enable local improvements and diversification, producing varied yet feasible path permutations. In addition to standard intra-route operators (lines 3–8, Algorithm 2), we introduce an RL-based destroy-reconstruct operator that leverages the trained Q-network for efficient local search. A random split point destroys the latter portion of the tour, retaining the prefix. The tour is then reconstructed by selecting the next node with

Algorithm 3 Inter-Path Metaheuristic Mutation (Crossover with RL Guidance)

1: **Procedure** InterMutatePath($path_1, path_2$)
2: Copy $path$ into new_path
3: **Destroy:** Truncate $path_1$ at a random node
4: **Masking:** allow only nodes from $path_2$ that are not yet visited in new_path to be in the set of possible nodes for reconstruction Ω_{path_2};
5: **Reconstruct:** roll out using the RL policy to rebuild the remainder based on $\arg\max_{a_t \in \Omega_{path_2}} Q(s_t, a_t; \theta^-)$.
6: **Completion:** continue until termination when RL policy chooses to return to terminal node.
7: **return** new_path

the highest Q-value from the RL network, excluding already visited or destroyed nodes (lines 9–11, Algorithm 2).

Inter-route operators extend the search space by combining information from multiple tours. Two top-performing tours (P_1 and P_2) are selected from the replay buffer. A random split point destroys the latter portion of P_1, retaining its prefix. Reconstruction of the destroyed portion uses the RL Q-network to guide node selection, but candidates are restricted to nodes from P_2 (lines 4–5, Algorithm 3). At each reconstruction step, the node from P_2 with the highest Q-value is inserted. Conceptually, this resembles classical crossover or segment-exchange operators in metaheuristics, but instead of performing computationally expensive exhaustive search, the Q-network efficiently selects promising nodes in linear time, allowing P_1 to be reconstructed from high-quality segments of P_2 while promoting exploration of new tour variations.

IV. Experiments Analysis and Results

We evaluated the reinforcement learning framework on a benchmark SOPTW dataset based on real road networks in the Netherlands and Luxembourg [20]. It contains 36 instances (20, 50, or 100 nodes) with rewards, penalties, service and time-window constraints, and two 15-minute adjacency matrices capturing mean and variance. RL agents with different configurations were trained for 1.2 million steps per instance and evaluated over 3000 trials, with average rewards compared against the benchmark SAC algorithm [3].

Fig. 3 shows a comparison between the mean rewards achieved by RL and those obtained by the benchmark SAC algorithm [3]. The RL agent consistently outperforms across all scenarios with 100 nodes, and almost most 50-node scenarios. Larger graph size problems, the search space grows exponentially, leaving more room for the RL agent to exploit its learning capabilities to discover superior solutions. Larger graphs tend to exhibit greater stochastic variability, which amplifies the advantage of the RL agent's ability to perform online adaptive decision-making.

The benefits of including MEER in the RL training can be seen in Fig. 4, where the mean, minimum, and maximum RL rewards improved compared to without MEER when aggregating in all scenarios for 50 and 100 nodes. Furthermore, the performance spread is narrower, indicating that MEER achieves consistently strong results across different scenarios.

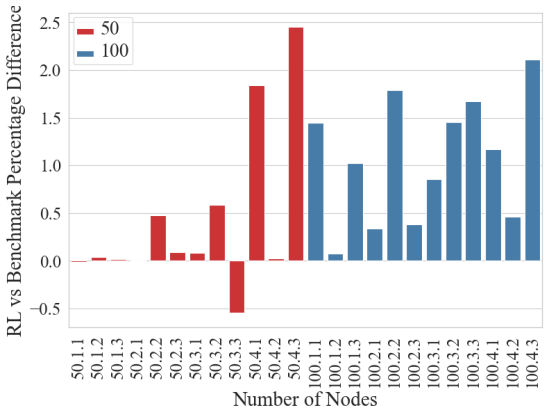

Fig. 3. Comparing RL Model Rewards vs Benchmark Baseline SAC Results.

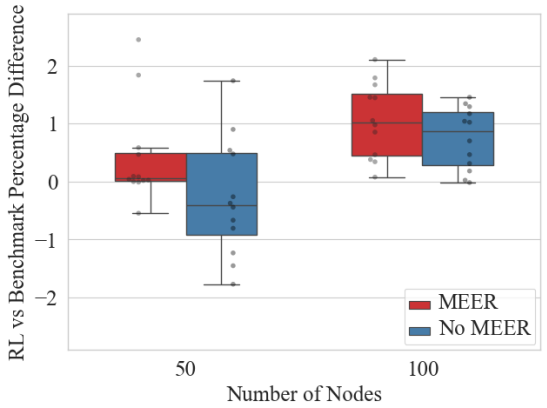

Fig. 4. Comparison of MEER vs No MEER, by number of nodes. Wilcoxon signed-rank test has a p-value of 0.001 and 0.021 for 50 and 100 nodes, respectively.

Next, we examine why inter-route operators yield notable improvements. Intra-route operators such as A_swap and A_insert have low Hamming distances, meaning only minor changes are made and routes remain similar, resulting in little improvement in normalized MEER rewards. In contrast, A_reverse produces larger modifications, reflected in a wider spread of Hamming distances (Fig. 5), leading to better normalized MEER rewards and higher win fractions. We computed that among the time taken in performing the RL updates along with rollout and MEER, 19.6% goes to MEER, 25.5% goes to the rollout, and 54.9% goes to the updating of neural networks, showing the the computational costs due to MEER is relatively small.

We distinguish 'RL' from 'RLd' ('d' denotes a deterministic approach). During reconstruction of the partially destroyed path using nodes from a second path, RLd selects the node with the highest Q-value, while RL uses roulette wheel selection. Fig. 5 shows many positive normalized MEER rewards, indicating improvements over the original route. As typical in evolutionary algorithms, most mutations are worse and fall below zero, but are discarded while better ones are retained. RL_crossover_1 shows a wider range of Hamming distances

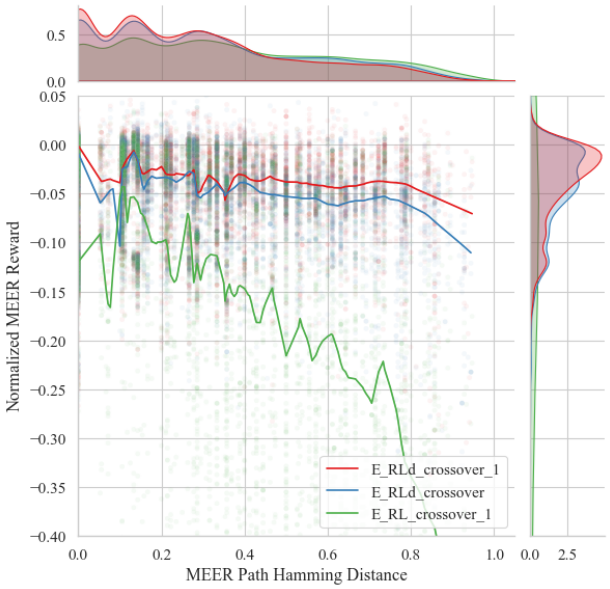

Fig. 5. Correlation plot for inter-route operators showing the Normalized MEER Reward vs MEER Hamming Distance (measure of dissimilarity between two paths). Positive values indicate the MEER-enhanced path yielded a higher reward than the original RL-generated path.

than its deterministic counterpart, reflecting greater exploration beneficial in early training stages.

The variant without the _1 suffix is more constrained, selecting the highest Q-value node from the second path at every reconstruction step, whereas the _1 variant does so only once before choosing from all feasible unvisited nodes. However, RLd_crossover_1 shows slightly lower path diversity, with a tighter KDE distribution near 1, suggesting that when given more flexibility, the RL agent tends to favor higher Q-value nodes, limiting exploration. Nevertheless, including both variants helps maintain diversity. These results indicate that integrating MEER with reinforcement learning improves solution quality and consistency, with benefits becoming more pronounced in larger and more stochastic SOPTW instances.

V. Conclusions

This paper demonstrates that deep reinforcement learning effectively solves the Stochastic Orienteering Problem with Time Windows. By combining graph features, Set Transformers, QR-DDQN, and the MEER mechanism, the approach leverages both DRL and metaheuristics. The proposed agent outperforms state-of-the-art methods, adapts to stochastic environments, and improves exploration and learning efficiency, offering a practical and promising solution. We aknowledge that comparing QR-DQN with vanilla DQN could provide additional insights; however, this work focuses on the proposed MEER mechanism. QR-DQN is selected for its ability to model stochastic returns, with further comparison left for future work. Potential extensions include advanced operator selection, curriculum learning, and transfer learning.

References

[1] W. Liu, R. Wang, T. Zhang, K. Li, W. Li, H. Ishibuchi, and X. Liao, "Hybridization of evolutionary algorithm and deep reinforcement learning for multiobjective orienteering optimization," *IEEE Transactions on Evolutionary Computation*, vol. 27, no. 5, pp. 1260–1274, 2022.

[2] A. Gunawan, H. C. Lau, and P. Vansteenwegen, "Orienteering problem: A survey of recent variants, solution approaches and applications," *European Journal of Operational Research*, vol. 255, no. 2, pp. 315–332, 2016.

[3] C. Verbeeck, P. Vansteenwegen, and E.-H. Aghezzaf, "Solving the stochastic time-dependent orienteering problem with time windows," *European Journal of Operational Research*, vol. 255, no. 3, pp. 699–718, 2016.

[4] S. T. W. Mara, R. Norcahyo, P. Jodiawan, L. Lusiantoro, and A. P. Rifai, "A survey of adaptive large neighborhood search algorithms and applications," *Computers & Operations Research*, vol. 146, p. 105903, 2022.

[5] M. A. Zuzuárregui and S. Carpin, "Solving stochastic orienteering problems with chance constraints using a gnn powered monte carlo tree search," *arXiv preprint arXiv:2409.04653*, 2024.

[6] S. Balhara, N. Gupta, A. Alkhayyat, I. Bharti, R. Q. Malik, S. N. Mahmood, and F. Abedi, "A survey on deep reinforcement learning architectures, applications and emerging trends," *IET Communications*, 2022.

[7] A. Farooq and K. Iqbal, "A survey of reinforcement learning for optimization in automation," in *2024 IEEE 20th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2024, pp. 2487–2494.

[8] R. Qi, J.-q. Li, J. Wang, H. Jin, and Y.-y. Han, "Qmoea: A q-learning-based multiobjective evolutionary algorithm for solving time-dependent green vehicle routing problems with time windows," *Information Sciences*, vol. 608, pp. 178–201, 2022.

[9] Z. Xing and S. Tu, "A graph neural network assisted monte carlo tree search approach to traveling salesman problem," *Ieee Access*, vol. 8, pp. 108 418–108 428, 2020.

[10] J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh, "Set transformer: A framework for attention-based permutation-invariant neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 3744–3753.

[11] J. Beck, R. Vuorio, E. Z. Liu, Z. Xiong, L. Zintgraf, C. Finn, and S. Whiteson, "A survey of meta-reinforcement learning," *arXiv preprint arXiv:2301.08028*, 2023.

[12] Z. Zhu, K. Lin, A. K. Jain, and J. Zhou, "Transfer learning in deep reinforcement learning: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 11, pp. 13 344–13 362, 2023.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[14] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.

[15] W. Dabney, M. Rowland, M. Bellemare, and R. Munos, "Distributional reinforcement learning with quantile regression," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[16] N. Cesa-Bianchi, C. Gentile, G. Lugosi, and G. Neu, "Boltzmann exploration done right," *Advances in neural information processing systems*, vol. 30, 2017.

[17] S. Huang and S. Ontañón, "A closer look at invalid action masking in policy gradient algorithms," *arXiv preprint arXiv:2006.14171*, 2020.

[18] C.-Y. Tang, C.-H. Liu, W.-K. Chen, and S. D. You, "Implementing action mask in proximal policy optimization (ppo) algorithm," *ICT Express*, vol. 6, no. 3, pp. 200–203, 2020.

[19] Y. Hou, X. Liang, J. Zhang, Q. Yang, A. Yang, and N. Wang, "Exploring the use of invalid action masking in reinforcement learning: A comparative study of on-policy and off-policy algorithms in real-time strategy games," *Applied Sciences*, vol. 13, no. 14, p. 8283, 2023.

[20] C. Verbeeck, K. Sörensen, E.-H. Aghezzaf, and P. Vansteenwegen, "A fast solution method for the time-dependent orienteering problem," *European Journal of Operational Research*, vol. 236, no. 2, pp. 419–432, 2014.