Coordinated Underwater Orienteering with Multiagent Evolutionary Reinforcement Learning

Nathan Butler[1], Hyuntaek Oh[1], and Tanner Wells[1]

***Abstract*—Rising sea-levels due to climate change has led to an increasing need for teams of autonomous underwater robots to explore and monitor the world's oceans. For these robots successfully localize in remote marine environments, each robot must maintain connectivity with a surface node assumed to have perfect localization information. However, achieving coordinated actions the both achieve tasks and maintain connectivity within a team of robots is challenging in the communication-restricted underwater setting. To address this, we turn to communication-free learning based approaches. In this paper, we propose applying Multiagent Evolutionary Reinforcement Learning (MERL) to learn a multiagent control policy in an environment with both connectivity maintenance and multiagent orienteering constraints. We use an Evolutionary Algorithm (EA) to train a baseline policy that optimizes for completed tasks and connected robots, then add reinforcement learning (RL) to influence desired agent behaviors. Results indicate that using MERL yields up to a 7.89% improvement in final model performance over using only EA in our modeled underwater environment.**

I. Introduction

Rising sea-levels threaten coastal communities worldwide with flooding, storms, and displacement. Polar ice melt is a key contributor to sea-level rise, with ocean-terminating glaciers providing significant ice melt forecasting challenges due to the limited data available from the marine environment warming the ice from below [1]. Teams of Autonomous Underwater Robots (AUVs) have been deployed to complete tasks in marine environments, but successful aquatic localization often requires a surface connection to reduce error [2]. While individual AUVs are unable to surface in the under-ice domain, the issue may be mitigated through the use of a surface-to-water base station capable of providing localization data to AUVs connected to it through single or multi-hop communication. While this connection constraint addresses the localization issue, it presents a new complex coordination challenge in which agents must decide between taking actions toward maximizing a global task completion reward or actions that maintain connectivity within the system.

In this paper, we focus on a team of robots that must deploy from and return to a single base station. These robots must maximize the number of tasks completed in an environment such that each robot returns to base within its battery life and maintains a connection with the base station during the entirety of its deployment. Fig. 1 describes this mission concept. While such a problem could be solved in a centralized manner, we desire a method that is robust to local agent disturbances and scales with an increasing number of AUVs. With this, we formulate this problem as a variant of the Multiagent Orienteering Problem (MOP) with base station connectivity constraints [3].

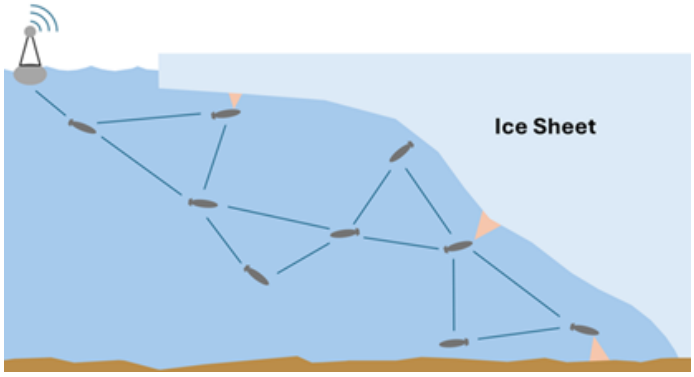

Fig. 1: Under-ice exploration with multi-hop surface base station connections.

Existing work has explored base station connectivity and the MOP as separate problems. Base station connectivity problems for mobile robot systems have typically been addressed using sampling-based or approximation-based methods [4] [5] [6]. Similarly, the MOP has largely been solved with sampling-based planning methods such as Monte Carlo Tree Search [7] [8] [9]. However, combining the MOP with base station connectivity constraints produces a complex multiagent coordination problem that, to the best of our knowledge, has not yet been solved in literature.

To this end, this work investigates the application of learning-based methods for training a multiagent team to maximize task completion while maintaining connectivity. In particular, we apply Multiagent Evolutionary Reinforcement Learning (MERL) to learn agent behaviors in a sparse-reward setting [10]. By assigning each agent a control policy that guides their movements through the environment, we use a MOP global task completion and agent connetivity reward at the end of each simulation to provide a fitness value to the policy. Unlike traditional evolutionary algorithms, MERL also provides a method to improve the policy using a dense reward function, accelerating convergence. We explore the ability of the RL feature to provide meaningful learning guidance when provided with simple, misaligned local rewards. In our results, we show that dense reward functions can be designed with little domain knowledge and, when applied to the MERL framework, can provide valuable feedback to accelerate convergence.

[1]Nathan Butler, Hyuntaek Oh, and Tanner Wells are with the Collaborative Robotics and Intelligent Systems (CoRIS) Institute, Oregon State University, Corvallis OR 97331, USA {butlnath, ohhyun, wellstan}@oregonstate.edu

By relying on the globally-aligned EA half of MERL to ultimately select optimal solutions, we show that we can employ misaligned, dense local rewards that encourage specific behaviors via RL to accelerate learning performance. We show that this framework can produce the coordinated behaviors necessary to solve the MOP with connectivity constraints.

Altogether, the contributions of this work are as follows:

1) Defining the Multiagent Orienteering Problem with Base Station connectivity constraints (MOP-BS)
2) Formatting this problem in the MERL framework with custom dense reward functions
3) Results and discussion on the ability of custom rewards to promote varying system behaviors

II. Background

We organize our related works into three sections that explore the background and current works relevant to our problem. The first section covers multiagent variants of the Orienteering Problem (OP). The second addresses works in base station connectivity maintenance for multi-robot systems. Finally, the third explores multiagent learning-based solutions, with an emphasis on MERL.

A. MOP with Base Station Connection Constraints

The MOP extends the OP to problems consisting of multiple planning agents. The classic NP-Hard OP combines the Knapsack Problem and Traveling Salesman Problem by requiring a planner to find 1) a subset of tasks to visit, and 2) a tour through this task set, with the added constraint that a tour's cost must not exceed a specified budget (usually time or energy). The Team OP extends this formulation to solve tours for multiple entities using a centralized solver. The MOP is similar to the TOP, but grants each agent the ability to contribute information until a global solution is reached [3].

Recent works have addressed the MOP through a variety of distributed approaches. Works by Best at al. and Skrynnik et al. use a decentralized Monte Carlo Tree Search method to allow multiple agents to solve tours in an anytime manner by exchanging partial solutions [7] [8]. Shi et al. address robust multiple-path OP with both approximation-based and Monte Carlo Tree Search solutions [9]. Unfortunately, these approaches struggle in the underwater domain, as they all require several rounds of communication between agents, which is infeasible with limited acoustic bandwidth.

Compared to other OP variants, work towards the MOP is relatively limited. Consequently, to the best of our knowledge no works thus far have explored an extension of this problem that incorporates connectivity maintenance constraints. Specifically, MOP-BS adds an additional constraint to MOP which requires that agents maintain single or multi-hop connection with a base station at all times. As such, the final global reward G is defined as the sum of tasks visited and connected agents evaluated at the end of a mission.

B. Base Station Connectivity Maintenance

Often, connectivity maintenance is modeled with respect to a fixed node called a base station [4] [5] [6] [11]. This base station is where all agents originate from and is where each agent must maintain a single or multi-hop connection with [4]. Various methods have been proposed for maintaining connectivity to a base station. Some methods model the base station as the root of a tree with agents breaching outwards and forming connections [4] [5] [6]. This allows for the use of area partitioning [4] and dynamic algorithms such as the minimum Steiner Tree [5] to optimally connect agents in a known geographic environment. Another method uses a two-phase tree propagation method for modeling and optimizing connectivity maintenance to a base station [6]. Role switching algorithms have also been used to task some agents with maintaining connectivity while others complete tasks [11].

In general, we note that these base station connectivity maintenance approaches provide little consideration for broader goals outside of purely maintaining connections, such as completing tasks in the orienteering setting discussed previously. Additionally, many approaches require multiple communication rounds, which are again infeasible in the underwater domain.

C. Evolutionary Algorithms and Reinforcement Learning

Learning-based methods can be used to train agents to make decisions independently in environments where communication is limited or denied. In problems like MOP-BS where we have sparse rewards, evolutionary algorithms (EAs) have shown success in learning effective control policies. Conversely, if dense reward signals are available, reinforcement learning (RL) methods can be combined with EA approaches to reach solutions faster [12].

Many recent works have seen success in combining EAs with RL to solve complex hierarchical objective problems. Khadka and Tumer propose a EA+RL (ERL) framework that uses experiences gathered during EA simulations to provide diverse experiences to an RL learner. This method takes advantage of the EA to optimize for sparse global rewards while using RL to refine policies with dense feedback. Separately, Zhan et al. use an RL-based genetic algorithm for coordinating a team of unmanned aerial vehicles in maritime search and rescue operations [13]. Finally, Majumdar et al. and Aydeniz et al. present a methods for extending the EA+RL approach to multiagent settings, with the first introducing MERL for training multiagent teams through multi-headed policies [10], and the second proposing a novelty-based fitness function to guide individual agent's exploration [14].

These works highlight the potential of EA+RL and MERL approaches for handling challenging multiagent problems. As MERL has produced agent and team-level behaviors necessary for complex tasks, we explore the application of this method for our problem.

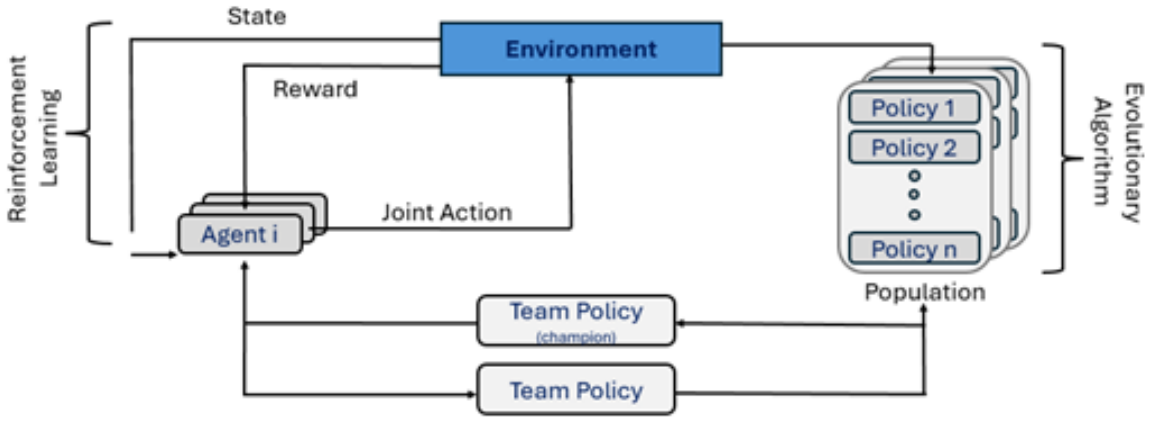

Fig. 2: MERL training process. EA loop runs over a population of policies (right side of figure). RL loop optimizes a champion policy using replay buffer containing experiences gained during EA training (left side of figure).

III. Method

In this section we discuss our implementation of the MERL algorithm. We also describe our multiagent policy and highlight how MERL updates different aspects of the policy during training.

A. Multiagent Evolutionary Reinforcement Learning

MERL solves for optimal policies by combining an EA search in parameter space with an RL search in action space. Fig. 2 represents the high-level MERL architecture. This framework uses an EA loop to iterate over a population of policies, performing selection, evaluation, crossover, and mutation functions as are common for EAs. However, during policy evaluations experiences in the form of state transitions are stored in a replay buffer. Each iteration, an RL loop acts on a batch of experiences sampled from the buffer to update the current champion policy.

The particular reinforcement learning algorithm used is deep Q-learning (DQN). This algorithm works in discrete action spaces, and can handle complex continuous observation spaces. These qualities make it a good candidate for handling the MOP-BS problem characteristics. Using DQN for reinforcement learning allows the model to reinforce agent-specific behaviors and teaming that are difficult to learn with only the global reward used in the EA.

Alg. 1 (MERL) provides a detailed description of the MERL training loop. The EA process begins by first randomly generating a population of k multi-headed policies with n heads each (line 1). We initialize an empty replay buffer to store experiences during training (line 2). Each policy is then rolled out in multiple environment simulations with $Rollout$ (line 5) and assigned a fitness calculated as the average global reward $G(s)$ obtained by the end of each simulation (line 6). We also store experiences (state, action, reward, next state, done) encountered during training in the replay buffer (line 7). When policy rollouts are finished, we obtain the highest-fitness policy using $GetBest$ (line 9) and create a copy of that policy with $CloneBest$ for RL training (line 10).

Algorithm 1 MERL

Input: population size k, number of agents n, batch size b, discount factor γ
Output: best policy π^*

1: $\Pi \leftarrow InitPols(k, n)$ ▷ Init. population
2: $buf \leftarrow \emptyset$ ▷ Init. replay buffer
3: **while** training **do** ▷ Enter training loop
4: **for** each unevaluated policy $\pi \in \Pi$ **do**
5: $exp, f \leftarrow Rollout(\pi)$ ▷ Run sim with π
6: $\pi.f \leftarrow f$ ▷ Update policy fitness
7: $buf.append(exp)$ ▷ Store experiences
8: **end for**
9: $\pi^* \leftarrow GetBest(\Pi)$ ▷ Get best π
10: $\pi^{RL} \leftarrow CloneBest(\pi^*)$ ▷ Copy best for RL
11: $batch \leftarrow buff.sample(b)$ ▷ Create RL batch
12: **for** $s, a, r, s' \in batch$ **do** ▷ Enter RL loop
13: **for** $head \in \pi^{RL}.heads$ **do** ▷ Train heads
14: $Q_s \leftarrow \pi^{RL}(s)[head]$
15: $Q_{s'} \leftarrow \pi^{RL}(s')[head]$
16: $target = r + \gamma * \max Q_{s'}$
17: $loss = MSE(target, Q_s(a))$
18: $\pi^{RL}[head].backprop(loss)$
19: **end for**
20: **end for**
21: $\Pi' \leftarrow [\pi^*, \pi^{RL}]$ ▷ Start next gen.
22: **while** length $\Pi' < k$ **do** ▷ Create children
23: $pair \leftarrow Select(\Pi)$
24: $child \leftarrow Crossover(pair)$
25: $Mutate(child)$
26: $\Pi'.append(child)$
27: **end while**
28: $\Pi \leftarrow \Pi'$
29: **end while**
30: **return** π^* ▷ Return best policy

The RL loop performs training on a batch of experiences sampled from the replay buffer (line 11). The details of the multi-headed policy will be discussed in the next section, so for now we just note that RL operates on each head individually. We first obtain the state-action values at each head for state s and next state s' (lines 14 and 15). Then we get our target value using agent-specific local rewards r and the highest-value action at the next state (line 16). We compute MSE loss between the state-action values and target values and then backpropagate this loss through the corresponding head (lines 17 and 18).

With the RL policy updated, we initialize a new population with the best-performing EA policy and refined RL policy (line 21) and then fill the new population with children. The algorithm then uses a tournament selection process to select two "parent" policies from the population, performs crossover with the parent policies' parameters to generate a child, randomly mutates the child's parameters, and adds the child into the new population until the population size reaches k (lines 23-26). When this occurs, we move to operating on the new policy and restart the MERL loop (line 28). Finally, upon completion we return the best-fit policy (line 30).

B. Multi-Headed Policy for Centralized Training and Decentralized Execution

Policies trained using RL for multiagent systems can be susceptible to instability when agents must execute coordinated actions, as is the case for the MOP-BS problem. We alleviate this issue using the multi-headed policy framework presented by Majumdar et al. [10]. Fig. 3 displays the shared and agent-specific components of this policy network. The policy takes as input the environment state and outputs an action distribution at each head, where each head corresponds to a specific agent. With, the network can be used to return a joint action for centralized training or queried on an agent-specific basis for multiagent decentralized execution.

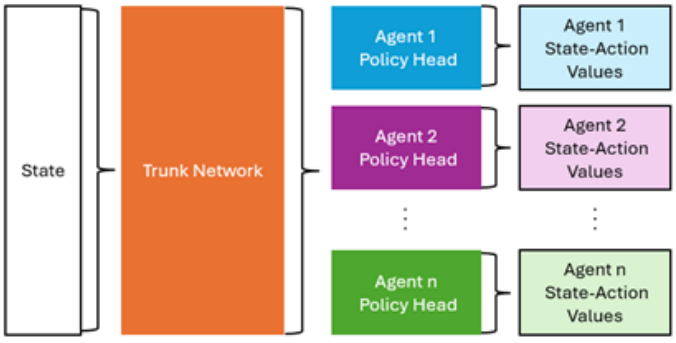

Fig. 3: Multi-headed policy for agent action selection. Shared trunk provides training stability while agent-specific heads enable agents to refine their individual policies during RL loop.

The multi-headed policy is divided into a trunk component and agent heads. The trunk is shared by all heads, and is responsible for initial environment processing. During EA crossover and mutation, all policy parameters are eligible for adjustment. However, in RL only the heads may be impacted so that the trunk of the network can maintain stability across agents.

IV. Experimental Setup

This section describes our evaluation domain and the experiments run for measuring the impact of MERL with different RL rewards versus using an EA on its own.

A. Experimental Domain

In the MOP-BS underwater exploration problem, tasks are dispersed over a wide area. In our experimental setup, we model these tasks as points that agents must get close to in order to "complete". As described previously, underwater communications are difficult over large distances, and a base station is often used to localize agents. This constraint is modeled in the simulation as a maximum allowable communication distance between any agent and a static base station. When this maximum distance is exceeded, the agent is considered "dead" and no longer can move during the simulation. In our experimental setup, the maximum communication distance between agents and the base station is less than the distance from the base station to a task. This means that in order to complete any task, agents must collaborate to form a network between themselves and the base station. In this way, a policy must be learned that navigates both the MOP and connectivity maintenance constraints that exist in multiagent underwater expeditions.

We define our experimental environment as a 2D grid, with four tasks, four agents, and a base station. Fig. 4 shows an example visualization of this environment where all four agents leave the base station and head out into the environment until they break connection with the base station. The agents in this example disconnect before they can complete a task.

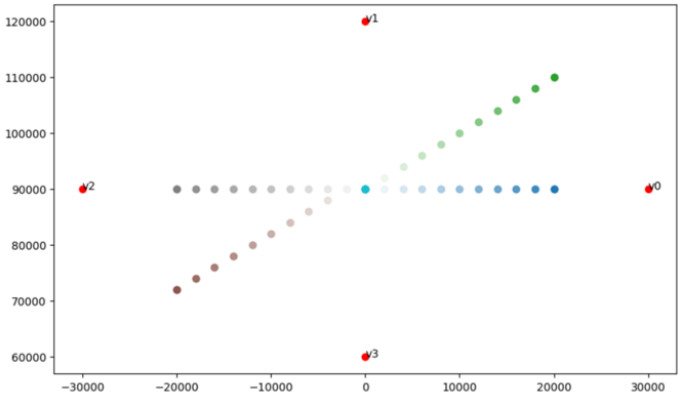

Fig. 4: Example environment used for experiments. The four tasks, shown as red dots, are placed a set distance away from the base station, shown as the blue dot. These other colored dots represent the position of each agent over time. Lighter colored dots show where the agent was earlier in the simulation, and darker colors show agent location later in the simulation.

For this domain, we consider a continuous state space consisting of agent positions relative to the base station and their connectivity status. We allow agents to select from a discrete action space representing eight constant-velocity movements in the cardinal and intermediate directions.

B. Reward Functions and Experiments

The MOP-BS problem requires agents to complete tasks while maintaining connection to the base station. As such, the global reward for this problem is defined as

$$G(s) = tasks_{completed} + agents_{connected}, \tag{1}$$

where $tasks_{completed}$ is the total number of tasks completed during the simulation and $agents_{connected}$ is the total number of agents who remain connected to the base station by the end of the simulation. Here, in order to maximize score, agents must learn to keep themselves connected while also completing tasks in the environment.

The global fitness equation provides only a sparse reward. A dense reward is therefore needed to support frequent RL feedback. We are interested in exploring how changing this dense reward function effects the final behaviors learned during MERL. To achieve this, we created two different reward functions, each with the goal of motivating different types of agent behavior.

The first function aims to motivate agents to maintain an optimal distance from other agents or the base station. We define this function as

$$L1_i(s) = \exp\left(-\frac{(d_{\text{agent\_min}} - dist_{opt})^2}{2 \cdot scale^2}\right), \tag{2}$$

where $d_{\text{agent\_min}}$ denotes the euclidean distance of agent i to the nearest other agent or to the base station if all other agents are disconnected, $dist_{opt}$ denotes the desired distance between agents, and $scale$ denotes the scaling factor. This equation makes the reward $L1_i$ a Gaussian curve where 1 is the maximum reward, and any difference between $dist_{opt}$ and $d_{\text{agent\_min}}$ decreases this reward from a maximum of 1 to a minimum of zero. This is scaled by $scale$. For our experiments, $dist_{opt}$ was set to 15000 and $scale$ was set to 3000. This incentivizes agents to be around a desired distance of 15000 from the nearest closest agent, comfortably within the maximum communication range 20000.

The second function aims to motivate agents to move towards incomplete tasks. We this function as

$$L2_i(s) = 1/\exp\left(\frac{d_{\text{task\_min}}}{\text{distance\_scalar}}\right), \tag{3}$$

where $d_{\text{task\_min}}$ denotes the euclidean distance from i to the nearest incomplete task and distance_scalar scales the minimum distance between 0 and 1. The result is a reward that is at a maximum of 1 when an agent is directly at an incomplete task and goes to 0 the further away an agent is from an incomplete task. This reward was designed to reinforce behaviors that allow agents to go towards the closest incomplete task.

We note here that each function is defined using limited domain knowledge, and as such is not necessarily aligned with (1). Rather, we design these functions to motivate "desired" behaviors that are then refined toward an optimal policy by the MERL EA loop. Both reward functions are tested separately and their overall performance and policy behaviors are compared in the following results section.

V. Results

Fig. 5 shows the performances for the EA only loop, MERL with the L1 dense RL reward, and MERL with the L2 dense RL reward. These three methods were trained for 100 epochs, and the best global reward is reported for each epoch. Each method was tested 10 times, and the results in 5 show the average results of each method and their respective standard errors. MERL with L1 sees a 7.89% increase in average final scores compared to only the EA while MERL with L2 sees a 5.26% increase over only the EA.

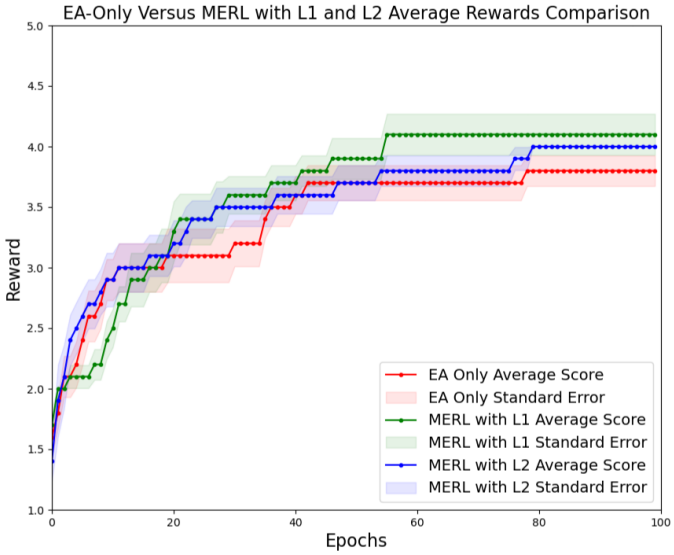

Fig. 5: Average training performance from EA-Only, MERL with L1 dense reward, and MERL with L2 dense reward plotted with standard error. Each method was tested over 10 trials with 100 epochs each. The number of epochs versus the average global award are plotted and indicate how each method learns over time as well as their average best rewards.

These results indicate a slightly better final performance of MERL with both L1 and L2 rewards over 100 epochs. MERL with L1 on average converged in more epochs than EA only, but had a significantly higher final average reward. MERL with L2 converged in less epochs on average than EA only and had a slightly higher average final reward.

In addition, Figs. 6,7,8 show example trajectories of agents given the best policy learned during training for EA, MERL with the L1 dense reward, and MERL with the L2 dense reward respectively.

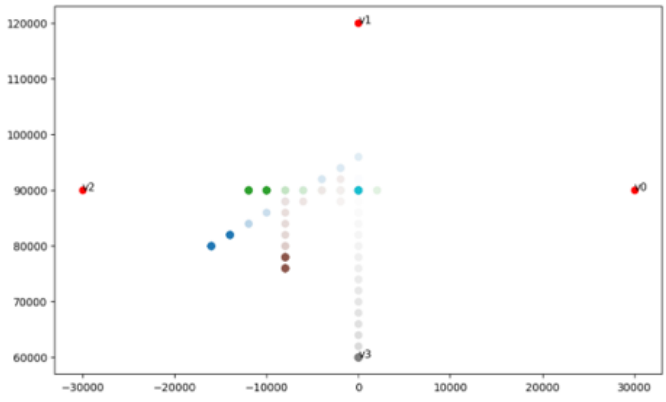

Fig. 6: Example behavior of best policy trained with EA only. In this example, three agents stayed connected with each other and the base station. The fourth agent completed one task and disconnected, leading to a score of 4.

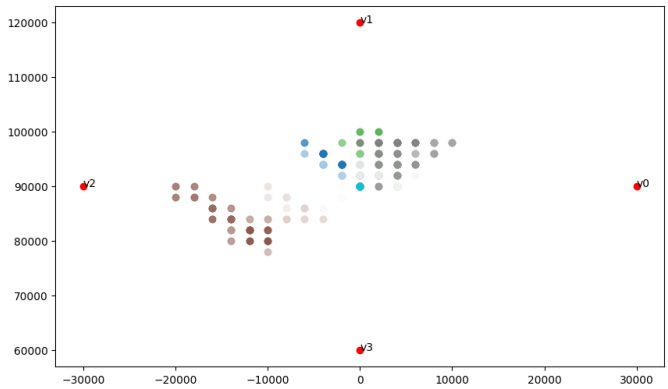

Fig. 7: Example behavior of best policy trained on MERL with the L1 dense reward function. In this example, all four agents stayed connected and three agents stayed close to each other, but no tasks were completed.

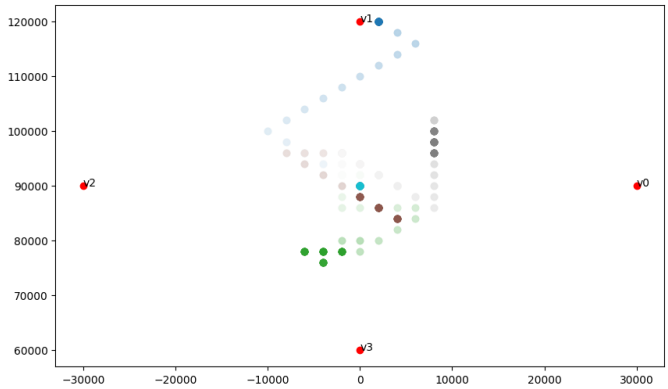

Fig. 8: Example behavior of best policy trained on MERL with the L2 dense reward function. In this example, two agents move towards the bottom task while two agents move towards the top task. One agent at the top completes a task then disconnects, and the other agent remains connected. Both bottom agents remain connected but do not complete a task.

Figure 6 shows learned behavior of agents for an example of training using EA alone. Three agents learned to stay near the base station on the left, while one agent stayed connected long enough to complete the bottom task before being disconnected. This led to a best G of four, with one task completed and three agents remaining connected. This example shows the baseline behavior of agents trained with the EA loop.

Figure 7 shows the behavior of agents for an example of using MERL with the L1 dense reward. In this example, all four agents stayed connected for the duration of the simulation, leading to a total score of four. However, no agent was able to complete a task. Three agents stayed relatively close to each other and the base station, while a fourth agent ventured towards a task before returning closer to the base station. This example shows the change of agent behavior that can result from using MERL with the L1 reward as compared to just using the EA.

Figure 8 shows the behavior of agents for an example of using MERL with the L2 dense reward. In this example, agents appear to split off in pairs of two. One group goes towards the top reward and the other group goes towards the bottom reward. One agent in the top group completes a task but disconnects, while the other agent remains connected. The two agents on the bottom remain connected but do not complete any tasks. This leads to a total score of four. Again, this example shows how changing the dense reward can significantly alter learned agent behaviors, even if the final total score is the same.

VI. Discussion

We draw several key insights from these results. First, we remark that MERL shows promise as an approach to the MOP-BS problem. Through only 100 epochs of training, the algorithm reached global rewards between three and five out of a maximum possible reward of eight (four complete tasks plus four connected robots). We can expect with longer training times to see results closer to optimal. Next, we note the success of dense local rewards engineered with little domain knowledge. For both L1 and L2, the average global reward achieved by the end of training was higher than that reached by the EA alone. This result is particularly interesting as neither L1 or L2 are fully aligned with G. This results highlights MERL's resilience to misaligned RL rewards, but also shows how users can apply somewhat naive RL reward functions with MERL and still see benefits. Finally, we observe in our trajectory plots how the benefits of varying local reward functions manifest in the system's behavior.

In the future, we envision exploring improvements to agent feedback and more complex problem scenarios. The RL portion of MERL may be able to more effectively learn if agent-specific rewards capture credit assignment information. As such, we would like to explore designing a difference reward for agents' dense feedback. Additionally, we plan to evaluate this method on randomized problem configurations to determine its robustness to a variety of task and agent configurations. With this, there are additional environmental

characteristics from the motivating domain that we could incorporate, such as ocean currents that modify agent trajectories in unexpected ways. Last, we believe another step toward realism is to incorporate a continuous action space that returns agent headings rather than discrete movements.

VII. Conclusion

In this paper, we have presented and evaluated a learning-based approach to solving the MOP-BS problem. We have described our implementation of the MERL algorithm, which combines an EA and DQN to train a control policy for a multiagent team. We analyzed this algorithm in a communication-constrained experimental setup inspired by the underwater domain. We presented custom dense reward functions for the RL portion of MERL and evaluated their impact on overal performance and the characteristics of the trajectories produced. Overall, our MERL implementation saw up to a 7.89% improvement in total global reward achieved over EA alone, highlighting the effectiveness of the added RL component, even with misaligned local rewards. Going forward, we aim to increase realism through the experimental domain design and policy configuration.

References

[1] R. B. Alley, S. Anandakrishnan, K. Christianson, H. J. Horgan, A. Muto, B. R. Parizek, D. Pollard, and R. T. Walker, "Oceanic Forcing of Ice-Sheet Retreat: West Antarctica and More," *Annual Review of Earth and Planetary Sciences*, vol. 43, pp. 207–231, May 2015. Publisher: Annual Reviews.

[2] X. Su, I. Ullah, X. Liu, and D. Choi, "A Review of Underwater Localization Techniques, Algorithms, and Challenges," *Journal of Sensors*, vol. 2020, no. 1, p. 6403161, 2020. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1155/2020/6403161.

[3] 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, pp. 315–332, Dec. 2016.

[4] J. Scherer and B. Rinner, "Multi-Robot Persistent Surveillance With Connectivity Constraints," *IEEE Access*, vol. 8, pp. 15093–15109, 2020. Conference Name: IEEE Access.

[5] E. Stump, N. Michael, V. Kumar, and V. Isler, "Visibility-based deployment of robot formations for communication maintenance," in *2011 IEEE International Conference on Robotics and Automation*, pp. 4498–4505, May 2011. ISSN: 1050-4729.

[6] P. Mukhija, K. M. Krishna, and V. Krishna, "A two phase recursive tree propagation based multi-robotic exploration framework with fixed base station constraint," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4806–4811, Oct. 2010. ISSN: 2153-0866.

[7] G. Best, O. M. Cliff, T. Patten, R. R. Mettu, and R. Fitch, "Decentralised Monte Carlo Tree Search for Active Perception," in *Algorithmic Foundations of Robotics XII: Proceedings of the Twelfth Workshop on the Algorithmic Foundations of Robotics* (K. Goldberg, P. Abbeel, K. Bekris, and L. Miller, eds.), pp. 864–879, Cham: Springer International Publishing, 2020.

[8] A. Skrynnik, A. Andreychuk, K. Yakovlev, and A. Panov, "Decentralized Monte Carlo Tree Search for Partially Observable Multi-Agent Pathfinding," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, pp. 17531–17540, Mar. 2024. Number: 16.

[9] G. Shi, L. Zhou, and P. Tokekar, "Robust Multiple-Path Orienteering Problem: Securing Against Adversarial Attacks," *IEEE Transactions on Robotics*, vol. 39, pp. 2060–2077, June 2023. Conference Name: IEEE Transactions on Robotics.

[10] S. Majumdar, S. Khadka, S. Miret, S. Mcaleer, and K. Tumer, "Evolutionary Reinforcement Learning for Sample-Efficient Multiagent Coordination," in *Proceedings of the 37th International Conference on Machine Learning*, pp. 6651–6660, PMLR, Nov. 2020. ISSN: 2640-3498.

[11] K. Kobayashi, T. Higuchi, and S. Ueno, "Connectivity maintenance for robotic swarms by distributed role switching algorithm," *Artificial Life and Robotics*, vol. 27, pp. 876–884, Nov. 2022.

[12] H. Bai, R. Cheng, and Y. Jin, "Evolutionary Reinforcement Learning: A Survey," *Intelligent Computing*, vol. 2, p. 0025, Jan. 2023. Publisher: American Association for the Advancement of Science.

[13] H. Zhan, Y. Zhang, J. Huang, Y. Song, L. Xing, J. Wu, and Z. Gao, "A reinforcement learning-based evolutionary algorithm for the unmanned aerial vehicles maritime search and rescue path planning problem considering multiple rescue centers," *Memetic Computing*, vol. 16, pp. 373–386, Sept. 2024.

[14] A. A. Aydeniz, R. Loftin, and K. Tumer, "Novelty Seeking Multiagent Evolutionary Reinforcement Learning," in *Proceedings of the Genetic and Evolutionary Computation Conference*, (Lisbon Portugal), pp. 402–410, ACM, July 2023.