Cooperative Route Planning Framework for Multiple Distributed Assets in Maritime Applications

Sepideh Nikookar
Paras Sakharkar
Sathyanarayanan Somasunder
Senjuti Basu Roy
sn627,ps863,ss4327,senjutib@njit.edu
New Jersey Institute of Technology
Newark, NJ, USA

Adam Bienkowski
Matthew Macesker
Krishna R. Pattipati
adam.bienkowski,matthew.macesker,
krishna.pattipati@uconn.edu
University of Connecticut
Storrs, CT, USA

David Sidoti
david.sidoti@nrlmry.navy.mil
US Naval Research Laboratory,
Marine Meteorology Division
Monterey, CA, USA

ABSTRACT

This work formalizes the *Route Planning Problem (***RPP***)*, wherein a set of distributed assets (e.g., ships, submarines, unmanned systems) simultaneously plan routes to optimize a team goal (e.g., find the location of an unknown threat or object in minimum time and/or fuel consumption) while ensuring that the planned routes satisfy certain constraints (e.g., avoiding collisions and obstacles). This problem becomes overwhelmingly complex for multiple distributed assets as the search space grows exponentially to design such plans. The **RPP** is formalized as a Team Discrete Markov Decision Process (TDMDP) and we propose a *Multi-agent Multi-objective Reinforcement Learning* (MaMoRL) framework for solving it. We investigate challenges in deploying the solution in real-world settings and study approximation opportunities. We experimentally demonstrate MaMoRL's effectiveness on multiple real-world and synthetic grids, as well as for transfer learning. MaMoRL is deployed for use by the Naval Research Laboratory - Marine Meteorology Division (NRL-MMD), Monterey, CA.

CCS CONCEPTS

• **Information systems** → **Data management systems**; **Database design and models**.

KEYWORDS

Route planning; Multi-Agent Reinforcement Learning; Function Approximation; Scalable Solution Design; Data Management for AI.

ACM Reference Format:
Sepideh Nikookar, Paras Sakharkar, Sathyanarayanan Somasunder, Senjuti Basu Roy, Adam Bienkowski, Matthew Macesker, Krishna R. Pattipati, and David Sidoti. 2022. Cooperative Route Planning Framework for Multiple Distributed Assets in Maritime Applications. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22), June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3514221.3526131

SIGMOD '22, June 12–17, 2022, Philadelphia, PA, USA

ACM ISBN 978-1-4503-9249-5/22/06…$15.00
https://doi.org/10.1145/3514221.3526131

1 INTRODUCTION

Planning routes for multiple agents, such as ships submarines, and unmanned aerial/surface/underwater vehicles (UAVs, USVs, UUVs), considering multiple objectives, such as fuel, battery usage, time taken, and progress towards a goal, is a complex, but highly-relevant, problem for search and rescue, reconnaissance, and interdiction missions in maritime applications. These problems involve trade-offs among different objectives and require a coordinated search for an object in a very high-dimensional space by multiple geographically distributed searchers (herein referred to as assets or agents). In this vein, automated tools are needed to aid human decision makers in planning courses of action (COAs).

A canonical formulation of the problem is as follows: we are given a set of distributed assets over a discrete grid, each with a given starting location (described by its (lat, long) value), a respective speed limit, and a sensing radius. The problem is to cooperatively discover an object at an unknown location (described by a (lat, long) value). The assets communicate among themselves periodically after a fixed interval of time or communicate asynchronously by broadcasting their locations when an asset locates the destination. When two assets communicate, they get to know each other's current locations. The goal is to decide a sequence of moves, including waiting at nodes, for each asset (route plan) such that the total fuel consumption by the assets and the maximum time for reaching the mission goal over all assets (Makespan) is minimized, while avoiding collisions among themselves. We formalize the *Route Planning Problem* (**RPP**) in **Section 2** as a team decision-making problem and describe the challenges in solving it.

Formalizing the **RPP** requires developing a principled model that captures the nuances of simultaneous movement of the assets, along with how that impacts the objectives and satisfaction of constraints. At each step, the **RPP** solution produces the direction of move for each asset and the corresponding speed (note that a speed of 0 corresponds to the waiting option) while accounting for its sensing radius and the locations of other assets. For distributed assets, since the locations of the other assets are known intermittently, each asset has to *anticipate* the locations and moves of other assets through an appropriate "internal model".

Contribution I: A Distributed RPP Model - We model the **RPP** problem as a Team Discrete Markov Decision Problem (TDMDP) [20], where the states are the nodes in the grid and an asset's

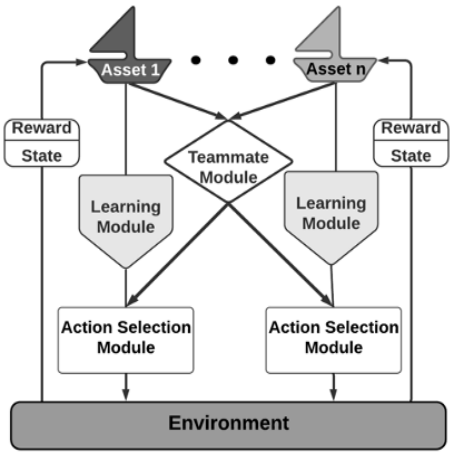


Figure 1: MaMoRL **Framework**

action is the decision to transit to one of the neighboring grid points at a particular speed or wait to avoid a collision. For each transition, there exist vector rewards that capture the multiple objectives of the team. **Section 2** contains further details.

Contribution II: MaMoRL - We propose a *Multi-agent Multi-objective Reinforcement Learning* (MaMoRL) framework for cooperative route planning under constraints. We identify its data centric challenges (**Section 2.4**). The MaMoRL framework contains *a. Teammate Module* (TMM) that captures the probability distribution of the belief of each asset on the locations and actions of other assets at each time step; b. *An Action Selection Module* (ASM) to enable distributed decision making by each asset to optimize multiple objectives, while satisfying constraints at each time step; and c. *Learning Module* (LM) that allows each asset to incrementally learn the functional approximations to the Q-functions (current best values of state-action pairs) from the environment and incorporating that learning inside the other two modules. **Section 3** contains further details.

We also realize that solving MaMoRL exactly is infeasible in a realistic maritime setting, because the memory and CPU bottlenecks of TMM and LM increase exponentially with the increase in the number of assets, the number of nodes and neighbors of the assets, and the cardinality of the set of speeds an asset can choose from.

We therefore develop two function approximation techniques - one using Linear Regression and the other using a Neural Network to effectively *approximate the Teammate Module* (TMM) *and the Learning Module* (LM) [16]. The Linear Regression based approximate solution, Approx-MaMoRL, is shown to be more effective than its Neural Network based counterpart NN-Approx-MaMoRL. It turns out to be as effective as the exact MaMoRL, while overcoming both the memory and CPU bottlenecks (refer to **Section 4**). The design of Approx-MaMoRL involves extensive feature engineering efforts.

Contribution III: Evaluation and Deployment. We perform an extensive experimental evaluation (**Section 4**) using multiple real-world and synthetic grids and implement several baselines for comparisons. We identify the computational bottlenecks of MaMoRL (**Section 4.3**) and demonstrate how Approx-MaMoRL overcomes those limitations. We design two variants of Approx-MaMoRL - one with no knowledge of the destination and the other with partial knowledge of the destination. The experimental results demonstrate the superiority of Approx-MaMoRL compared to the baseline solutions with a 95% statistical significance (**Section 4.4**). Our results additionally demonstrate that the policy learned by the Approx-MaMoRL is transferable to various sized grids **(Section 4.6)**. Finally, in Section 4.7, we describe how we deploy Approx-MaMoRL inside TMPLAR [22], an existing tool for Multi-objective Planning and Asset Routing used by NRL-MMD, Monterey, CA.

2 PRELIMINARIES AND PROBLEM DEFINITION

2.1 Preliminaries

Let N be a set of distributed assets. Each asset operates on a discrete grid $\mathcal{G} =< V, E >$. The weight of each directed edge e ($weight(e)$) between two nodes, $v_p \rightarrow v_q$, denotes the distance between v_p and v_q. The assets explore the grid cooperatively to discover the location of an object, represented as $d(x, y)$. Each asset $i \in N$ is represented as a quintuple:

$$i =< r_i, sp_i, source_i(x, y), cur_i^t(x, y), d_i(x, y) >$$

where r_i is the sensing radius of asset i, sp_i is the maximum speed/velocity, $source_i(x, y)$ is the starting point, $cur_i^t(x, y)$ is the location of asset i at time t, and $d_i(x, y)$ is the destination from the viewpoint of asset i. All assets have the same destination, that is unknown in the beginning, but gets revealed through exploration. Therefore, $d_i(x, y) = d(x, y); \quad \forall i \in N$.

2.2 Problem Setting and Definition

Each asset i is characterized as follows:

- It moves at a speed $\leq sp_i$.
- It observes the grid up to its sensing radius r_i.
- It makes a decision on its next action at each step, i.e., which neighboring node it moves to and at what speed or to wait. Decisions are made only when i is at a node in $\mathcal{G}$.
- Two assets (i, j) exchange their respective locations when they communicate periodically every k time steps or when the destination $d(x, y)$ has been found.

Fuel Consumption Model: The fuel consumption of asset i to move from node $v_p \rightarrow v_q$ at speed $sp'_i \leq sp_i$ is $fuel_i(v_p \rightarrow v_q, sp'_i)$, where sp_i is the maximum allowable speed of asset i. The fuel consumption depends on the distance between v_p, v_q and the speed sp'_i. There exist analytical models [3] for computing the fuel consumption as a function of distance traveled and speed. Thus, the fuel consumption of asset i to explore $\mathcal{G}$ can be computed and we denote it by T_{Fuel_i}.

Definition 1. *Total fuel consumption F_{total} : The overall fuel consumption by the set of $|N|$ assets while exploring $\mathcal{G}$ to discover $d(x, y)$ is $\sum_{\forall i} T_{Fuel_i}$.*

Time Model: The time taken for asset i to move from node $v_p \rightarrow v_q$ at speed $sp'_i \leq sp_i$ is $\frac{weight(v_p \rightarrow v_q)}{sp'_i}$. Thus, the total time taken by asset i to explore $\mathcal{G}$ can be computed and is represented by T_{Time_i}.

Definition 2. *Overall time expended T_{total} : The overall time expended by $|N|$ assets during exploration over $\mathcal{G}$ to discover $d(x, y)$ is the maximum over T_{Time_i}, i.e., $max_{\forall i} T_{Time_i}$.*

Definition 3. *Collision: Two assets i, j collide if they are at the same location simultaneously, i.e.,* $cur_i^t(x,y) = cur_j^t(x,y)$

Problem 1. ***(Route Planning Problem (RPP))*** *Plan routes of* $|N|$ *cooperative distributed assets (agents) with respective starting points (asset* i *with* $source_i(x,y)$*) over* $\mathcal{G}$ *to discover an initially unknown destination* $d(x,y)$ *to minimize* F_{total} *and* T_{total}*, while at no point in time during the exploration,* $cur_i^t(x,y) = cur_j^t(x,y), \forall i,j,t$.

2.3 Toy Running Example

Consider a toy example involving two assets (Table 1), their respective current positions, and their visited paths after 2 moves in Figure 2. Each asset can sense up to its sensing radius which is based on the distance and represented by double circled green and blue nodes for Asset1 and Asset2, respectively.

	r_i	sp_i	$source_i(x,y)$	$cur_i^2(x,y)$	$d_i(x,y)$
Asset1	2	3	(0,0)	(0,4)	(4,3)
Asset2	3	2	(8,7)	(5,4)	(4,3)

Table 1: A toy example using 2 assets

From starting location $source_1$, Asset1 can go to one of its two neighboring sensed nodes. It also can move at 3 different speeds, $(sp'_1 = 1,2,3 \leq sp_1)$ or wait; so the number of possible actions for Asset1 is 7. Evidently, without any further information, it is beneficial to move in the direction that explores more of the not-yet-sensed nodes, as long as collisions are avoided. Asset1 takes action a_0, because more nodes will be sensed from the new position and moves at speed 2 based on Table 2. Its position is changed to $cur_1^1(x,y) = (0,2)$. Similarly, Asset2 moves from $source_2$ to node $(6,6)$ by taking action a'_0 to sense 3 additional nodes. Moreover, it chooses speed 2, because that minimizes the average of time and fuel consumption (see Table 2).

Asset1				Asset2			
Speed	**Time**	**Fuel**	**Average**	**Speed**	**Time**	**Fuel**	**Average**
1	2	3.7664	2.8832	1	2.24	4.2184	3.2292
2	1	4.2714	**2.6357**	2	1.12	4.7840	**2.9520**
3	0.66	4.7286	2.6943	3	—	—	—

Table 2: Time and fuel consumption of the Assets

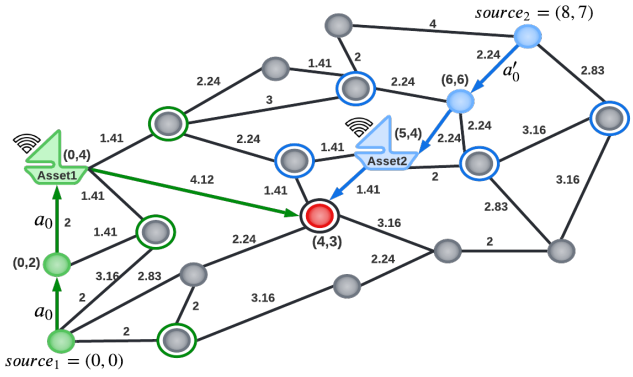


Figure 2: Assets' traveled paths and positions after 2 moves

2.4 Data Centric Challenges

The proposed work falls under the broad category of data management/engineering for efficient AI. The major data-centric challenges to solve the **RPP** stem from two sources: (a) Model and reward design; (b) Memory and CPU Bottlenecks.

2.4.1 Model and Reward Design. The problem of designing automated plans has been recently explored in several research works [2, 18, 21] that broadly fall under the umbrella of Exploratory Data Analysis (EDA). These efforts investigate the design of RL agents to discover user groups [21], to generate exploratory sessions for data scientists [2], or to design complex recommendation tasks, such as courses or trip plans [18]. Similar to these problems, we model the **RPP** as a sequential decision-making/control problem, where the fundamental model consists of a state-action pair for which an appropriate reward function needs to be designed. Unlike existing works, a state of the **RPP** corresponds to the current locations of all $|N|$ assets and an action constitutes deciding respective actions of all of assets (including waiting). Moreover, the problem requires the consideration of multiple objectives (fuel and time) and the simultaneous planning of routes by multiple distributed assets in a spatial domain with limited communication capabilities while satisfying multiple constraints (avoiding collisions and intermittent communication). Therefore, our proposed model `MaMoRL`, which involves extensive data engineering for the reward design process is significantly different from existing works. From a technical standpoint, `MaMoRL` contains a Teammate Module (Section 3.2.1) that captures the belief of each asset on the locations and actions of other assets at each time step, which is unique to this distributed planning problem and has not been studied in the aforementioned prior efforts.

2.4.2 Memory and CPU Bottlenecks. As we shall describe in Section 3.3, `MaMoRL` requires storing very large (multiple) P and Q tables in memory, and the sizes of these tables increase exponentially with the number of assets, grid size and the number of neighbors, and the cardinality of the set of speeds an asset can choose from (refer to Lemmata 1, 2), causing a memory bottleneck (refer to Table 6 for experimental results). We design two function approximation methods (based on Linear Regression and a Neural Network) (Section 3.3) that avoid pre-computing and storing these tables. The CPU bottleneck arises from the need to learn effectively in a setting with exponential number of environmental states and actions. To the best of our knowledge, some existing works [2, 29] proposes Deep Neural Network based RL framework to overcome the large state space problem, but do not comprehensively evaluate the scalability challenges. Naturally, the Neural Network based function approximation requires more training data to be effective, which is harder to obtain for the **RPP**, compared to the EDA problem studied in [2].

3 PROPOSED SOLUTION

We present our proposed model, algorithms for `TMM`, `LM` and `ASM`, and describe approximation opportunities.

3.1 Modeling

We model the **RPP** problem as a Team Discrete Markov Decision Process (TDMDP) (S, A, R) [20]:

a. The state space $S = |V|^{|N|}$, where $|V|$ is the number of nodes in $\mathcal{G}$. At a given time t, state s_t is the locations of all $|N|$ assets.

$$s_t \leftarrow \{cur_1^t(x,y), cur_2^t(x,y), ..., cur_{|N|}^t(x,y)\}$$

b. A is the set of actions. Given a state s, where each asset i is at its corresponding node v_{pi}, an action a_i corresponds to making decisions to move along $v_{pi} \rightarrow v_{qi}$ at speed $sp_i' \leq sp_i$ or staying at v_{pi}, i.e., $v_{pi} \rightarrow v_{pi}$; Let $a = [a_i]$ denote the actions over all assets. The effect of an action is a state transition T_r, which is assumed to be deterministic, that is, $T_r : S \times E \rightarrow S$, a new state is obtained by taking an action at each state.

c. $R(s_t, a_t, s_{t+1})$ is the reward obtained for transitioning from state s_t to state s_{t+1} by taking action a_t.

3.1.1 Reward Design. The process of designing the reward must be guided by the following intuition: (1) proportional to the number of newly explored nodes by the assets, (2) inversely proportional to the time taken, (3) inversely proportional to the fuel consumption.

1. Exploration Reward: of an action a relates to how much of the unexplored grid an asset can sense and is formulated as

$$r_{explore}^a = \frac{\sum_{i=1}^{|N|} Sensed(i)^{a_i}}{D_{max} \times |N|} \tag{1}$$

where $Sensed(i)^{a_i}$ is the number of newly sensed nodes by asset i by taking action a_i, normalized by D_{max}, which is the maximum out-degree in $\mathcal{G}$. Using example in Section 2.3, the exploration reward of taking the first action is $r_{explore}^{a_0} = \frac{2+3}{5\times 2} = 0.5$

2. Time Reward: of an action a over all assets is the inverse of the maximum time each asset needs as part of a (where $\frac{weight(v_{pi} \rightarrow v_{qi})}{sp'_i}$ is the time asset i needs for action a_i)

$$r_{time}^a = \frac{1}{max_{i=1,...,|N|} \frac{weight(v_{pi} \rightarrow v_{qi})}{sp'_i}} \tag{2}$$

For the toy example, $r_{time}^{a_0} = \frac{1}{max(1,1.2)} = 0.83$ for the first action of the example in Section 2.3.

3. Fuel Reward: of an action a is the inverse of the sum of fuel consumed by all $|N|$ assets.

$$r_{fuel}^a = \frac{1}{\sum_{i=1}^{|N|} weight(v_{pi} \rightarrow v_{qi}) \times fuel_i(1, sp'_i)} \tag{3}$$

For our implementation, we use the following model [3] to capture fuel per unit distance.

$$fuel_i(1, sp'_i) = 0.2525 \times {sp'_i}^2 + 1.6307 \times sp'_i \tag{4}$$

Based on the model, the fuel reward of the initial set of actions in the toy example is $r_{fuel}^{a_0} = \frac{1}{2\times 4.2714 + 2.24 \times 4.784} = 0.052$.

Evidently, at state s, an action a corresponds to two things: 1. The assets should move to the neighboring nodes to maximize the Exploration Reward (Equation 1); 2. The speed of each asset is to be chosen to optimize the average of Fuel Reward (Equation 3) and Time Reward (Equation 2).

3.2 Algorithm

Our proposed solution, Multi-agent Multi-objective Reinforcement Learning or MaMoRL in short, contains three different interacting modules (Figure 1) and is inspired by model free Reinforcement Learning for multiple agents [28].

3.2.1 ***Teammate Module (TMM):.*** Assets are distributed and communicate among themselves intermittently. Consequently, when an asset decides on its next move, it needs to have an internal model that anticipates the current locations and moves of other assets. The teammate module is designed to represent that internal model of other team members. The purpose of this internal model is to produce the probability distribution of actions of the remaining $|N| - 1$ assets. This module is gradually updated as asset i observes the actions of asset j (a_j^*) $\forall j \in N \setminus \{i\}$ for each executable action a_j in state s, using the following formula:

$$P_{i_t}(s, a_j) = \begin{cases} P_{i_{(t-1)}}(s, a_j) + \beta^{T-t+1} \sum_{a_t \in A_j(s) \setminus \{a_j\}} P_{i_{(t-1)}}(s, a_t), & a_j = a_j^* \\ (1 - \beta^{T-t+1}) P_{i_{(t-1)}}(s, a_j), & \text{otherwise} \end{cases} \tag{5}$$

Here $\beta \in [0, 1]$ is the learning rate for determining the effect of the previous action, T is the iteration number, A_i and A_j are the sets of possible actions for assets i and j, respectively. Clearly, these P values are to be stored and updated periodically while learning from the environment.

Lemma 1. *TMM has P tables for (exploration, time, and fuel) rewards, each with size*

$$|P| = (|V|^{|N|} \times |A| \times sp).$$

where $|V|$ is the size of the grid, $|A|$ is the number of actions, sp is the max speed and $|N|$ is the number of assets.

At the initial state, Asset1 and Asset2 take actions a_0 and a_0', respectively, thus state s_1 is obtained. At that time, for the exploration reward, Asset1 updates the $P_1(s_0, a')$ values in the system for all Asset2's actions a' in the action set using Equation 5 as follows:

$$\begin{aligned} P_{1_1}(s_0, a_j') &= (1 - 0.3^{(3-1+1)} P_{1_0}(s_0, a_0)) \\ &= (1 - 0.3^3) \times 0.2 = 0.1946, \quad \forall a_j' \in A_2(s_0) \setminus \{a_0'\} \\ P_{1_1}(s_0, a_0') &= 0.2216 \end{aligned}$$

We consider $T = 3$ and $\beta = 0.3$ in our example. Asset2 also does the same for updating P values of Asset1's actions.

3.2.2 ***Learning Module (LM):.*** After the transition

$$(s, a_1, ..., a_i, ..., a_{|N|}) \rightarrow (s', r),$$

asset i will update its Q-function table using the following equation:

$$\begin{aligned} Q(s, a_1, ..., a_i, ..., a_{|N|}) = (1 - \alpha) Q(s, a_1, ..., a_i, ..., a_{|N|}) \\ + \alpha (r + \gamma \max_{a' \in A_i} Q(s, a_1', ..., a_i', ..., a_{|N|}')) \end{aligned} \tag{6}$$

where $a_j' = \underset{b \in A_j}{\text{argmax}}\ P_i(s', b); \quad j = 1, ..., i-1, i+1, ..., |N|$

Clearly, LM must leverage TMM, and update it periodically.

Lemma 2. *There exist different Q tables, one for each reward in LM, and each of size*

$$|Q| = (|V| \times |A| \times sp)^{|N|}.$$

In example 2.3, the Q values are initialized as

$$Q(s, a, a') = \frac{1}{|A(s)||A'(s)|} = 0.0286, \quad \forall a \in A, \forall a' \in A', \forall s \in S$$

Here, we take $\alpha = 0.9$ and $\gamma = 0.8$. After Asset1 and Asset2 take actions a_0 and a'_0, respectively, the new state s_1 and exploration reward $r = \frac{2+3}{5\times2} = 0.5$ are obtained. Then, at $t = 1$, Asset1 will update its Q value for actions a_0 and a'_0 using Equation 6 as follows:

$Q(s, a_0, a'_0) = (1 - 0.9) \times 0.0286 + 0.9(0.5 + 0.8 * 0.0286) = 0.47$

3.2.3 ***Action Selection Module (***ASM***):.*** Asset i uses a greedy policy for selecting the next action using the following equation:

$$a_i^* = \underset{a_i \in A_i}{\text{argmax}}\ V(a_i|A^*) \tag{7}$$

where $A^* = \{a_1^*, ..., a_{i-1}^*, a_{i+1}^*, .., a_{|N|}^*\}$ and $V(a_i|A^*)$ is the conditional expectation of an action given by:

$$V(a_i|A^*) = \begin{cases} \sum\limits_{\forall j \neq i} P(s, a_j) Q(s, a_1^*, ..., a_i, ..., a_{|N|}^*), & t \leq T \\ (\underset{\forall j \neq i}{\text{argmax}}\ P(s, a_j^*)) Q(s, a_1^*, ..., a_i, ..., a_{|N|}^*), & t > T \end{cases} \tag{8}$$

Here, T represents an iteration threshold. In order to see which action Asset1 is going to take in state s_1 (Figure 2) in example 2.3, we calculate the conditional expectation for all possible actions that Asset1 can take in the current state using the first formula in equation 8 considering $T = 3$ and $t = 1$:

$$V_1(a_i|\{a'_j\}) = 4 \times (0.1946 \times 0.0286) + 0.2216 \times 0.0286 = 0.0286$$

$$V_1(a_0|\{a'_j\}) = 4 \times (0.1946 \times 0.0286) + 0.2216 \times 0.47 = 0.1264$$

For $i = 1, ..., 6$ and $j = 0, ..., 4$. Thus, Asset1 chooses action a_0 for the next step and state s_2 is obtained (see Figure 2).

3.3 Function Approximation

It is infeasible to compute MaMoRL exactly in a realistic setting simply because of the exponential size of the P and Q tables (Refer to Lemmas 1 and 2). Indeed, the computational bottlenecks lie in exactly computing or traversing TMM *and* LM. Therefore, we study function approximations for TMM *and* LM. The approximate solution, Approx-MaMoRL, is designed with extensive feature engineering efforts to effectively approximate TMM *and* LM without actually building it, and the assets make decisions based on that approximation.

3.3.1 Function Approximation for TMM.

1. Linear Regression. : As opposed to computing $P_{i_t}(s, a_j)$, the action of each teammate j at state s and time t by asset i is approximated as a linear function [5, 24] $\hat{f}_{i,a_j,s}$, given by

$$\hat{f}_{i,a_j,s} = \omega_1 degree(v_j, s) + \omega_2 \theta(v_j, s) + \omega_3 \alpha(a_j, s) + \omega_4 \beta(a_j, d, s) + \omega_5 (sp_j, s) \tag{9}$$

where:

1. v_j = latest location of asset j,
2. $\theta(v_j, s) = 1$ if there is another asset within m hops, else 0,
3. $\alpha(a_j, s)$ is 1, if a_j leads to unsensed nodes; else 0,
4. $\beta(a_j, d, s) = 1$, if a_j leads to d; else 0,
5. sp_j = speed of asset j,
6. $\omega_l \in [0, 1], \quad l = 1, ..., 5$ are weights of the features.

The function approximation considers features that are useful before the goal is discovered (all features excluding β), and ones that are only useful afterward (all excluding α). β in Equation 9 is designed for the latter purpose.

Training: In the absence of the historical data, we obtain a sample of the original ($P_{i_t}(s, a_j)$ values) coming from MaMoRL to approximate TMM. The features are hand-crafted and the goal here is to learn the weights ω_l that minimize the following error function:

$$Minimize \sum_{\forall i,a_j,s} [f_{i,a_j,s} - \hat{f}_{i,a_j,s}]^2 \tag{10}$$

Route Planning: During the actual route planning, at a given state s by asset i, an action is simply considered to be the action a_j for asset j that has the highest $P_{i_t}(s, a_j)$ value per Equation 9.

2. Neural Network: Using the same training instance as that of Linear Regression, a Neural Network is trained by collecting the P values and the concomitant feature values for each possible action-state pair. The trained Neural Network model is then used to predict the P values for each possible action in order to predict the action a_j with the highest $P_{i_t}(s, a_j)$ in route planning.

3.3.2 Function Approximation of LM.*.*

1. Linear Regression. : Similarly, as opposed to storing the entire Q table, asset i learns a reward function as a linear combination of features:

$$\hat{r}_{i,a_i,s} = \omega_1 degree(v_i, s) + \omega_2 \theta(v_j, s) + \omega_3 \alpha(a_i, s) + \omega_4 \beta(a_i, d, s) + \omega_5 (sp_i, s) + \omega_6 sp'_i \tag{11}$$

Here, sp'_i is the speed of asset i that causes collision and the remaining features are as described in Equation 9.

Training: Similar to the function Approximation for the TMM, we sample from the training grid to learn ω_l by minimizing the least squares objective function:

$$Minimize \sum_{\forall i,a_i,s} [r_{i,a_i,s} - \hat{r}_{i,a_i,s}]^2 \tag{12}$$

Route Planning: During route planing, the reward for each action a_i is calculated using Equation 11 and the action resulting in the highest $\hat{r}_{i,a_i,s}$ is chosen by asset i at state s.

2. Neural Network: Using the same training data as that of function approximation using Linear Regression of LM, a Neural Network is also trained.

4 EXPERIMENTAL EVALUATION

Most of the experiments are run on OS Big Sur with 2.4 GHz Quad-Core Intel Core i5 Processor and 16 GB RAM. Experiments in Table 6 are run on a 64-bit OS server with 3.5 GHz 11th Gen Intel Core i9 and 128 GB RAM. Our code and data are available on GitHub.[1] All results are presented as an average of 10 runs.

4.1 Experimental Setup

4.1.1 Datasets. **I. Real World Data.** A discrete grid of the entire world is produced by taking high-resolution shoreline data from the GSHHG data set and generating a mesh using Gmsh over the world's oceans[12][26]. In this graph, each node has an out-degree

[1] https://github.com/RoutePlanningProblem/MaMoRL

of at most 6. To represent the greater amount of navigational adjustments necessary near land, the mesh is generated such that regions closer to coastlines have a higher resolution than those in the middle of the ocean. This grid is then split into three datasets of increasing resolution: the Caribbean Grid, the North America Shore Grid, and the Atlantic Grid (refer to Table 3).

Datasets	Region	$\|V\|$	$\|E\|$
1	Caribbean Grid	710	1684
2	North America Shore Grid	3291	7811
3	Atlantic Grid	14655	35061

Table 3: Datasets Description

II. Synthetic Data. We use the NetworkX library in Python to generate grids by varying one of the three parameters at a time. (a) number of nodes; (b) number of edges; (c) max out-degree.

4.1.2 Implemented Algorithms. We implement `Approx-MaMoRL`, `NN-Approx-MaMoRL`, `MaMoRL` and compare them with several baselines. Additionally, we implement `MaMoRL` with partial knowledge, which works as follows:

1. `MaMoRL` with partial knowledge: We assume that under *partial knowledge*, the destination is inside a specified region (described by a bounding box of lat, long) that the assets are aware of, but the exact location (lat, long) is unknown. For that, all assets make use of Dijkstra's Shortest Path [9] algorithm to find the shortest path to the boundary of the region containing the destination. Then the assets leverage `MaMoRL` solutions inside the region to reach the destination.
2. `Baseline-1`: Assets plan their routes *one-by-one in a non-simultaneous round robin fashion.* The reward functions are identical to that of the ones described in Section 3.1.1. The baseline is likely to require less fuel for the assets to reach the destination (i.e., smaller F_{total}) because of their long wait at the nodes and no unnecessary moves at the expense of taking a longer time, thereby giving rise to a larger T_{total} value.
3. `Baseline-2`: In this RL based implementation, assets use the same reward functions as before but plan their routes independently, without taking into account the actions of others. This fully distributed planning is akin to the ALOHA protocol and is prone to collisions.
4. `Random Walk-Baseline`: The implementation designs a random walk based solution, where an action in each step and the assets' speed are decided randomly from a uniform probability mass function (for actions) and density (for speed).

Experimentation Goals. Our effort attempts to answer the following questions:

- Q1. How function approximation methods `Approx-MaMoRL` and `NN-Approx-MaMoRL` compare with each other?
- Q2. What are the bottlenecks of implementing the proposed solutions and the baselines?
- Q3. How effective `Approx-MaMoRL` is with or without partial knowledge w.r.t. different baselines and parameters?
- Q4. The Accuracy vs. Speed Trade off of `Approx-MaMoRL`.
- Q5. How effective is `Approx-MaMoRL` in Transfer Learning?

Table 4 contains default values for the parameters that are part of `MaMoRL`. We present T_{total}, F_{total}, relative improvement in objective function values (defined in Section 4.4), memory usage, and running time of the corresponding solution for evaluation purposes. Comparison results are presented with paired t-test [13] with 95% statistical significance.

4.1.3 Summary of Results. Our first and foremost observation is that the Linear Regression based function approximation `Approx-MaMoRL` requires less training time (15× faster) and less training data and is more effective than its Neural Network counterpart `NN-Approx-MaMoRL` (Refer to Section 4.2). Hence the rest of our results primarily focus on `Approx-MaMoRL`. Consistent with our theoretical analysis, in Section 4.3, we demonstrate the memory and CPU bottlenecks of `MaMoRL` and compare that with `Approx-MaMoRL` along with other baselines (Table 6). These results demonstrate that `MaMoRL` is not a practical solution beyond a small grid with only 2 assets, whereas `Approx-MaMoRL` produces reasonable performance to the exact model while being an order of magnitude faster. `Baseline-2` results in asset collisions for more than 97% of the runs, due to its unique design choice, making it infeasible in practice. Next, from extensive synthetic data experiments, we observe in Section 4.4 that the proposed solution `Approx-MaMoRL` **with or without partial knowledge** is effective and robust to problem settings. We also observe that `Approx-MaMoRL` returns the pareto front which is better at optimizing both fuel and time when compared to the baselines. Then, in section 4.5, we explore the trade-off between training time and accuracy of `Approx-MaMoRL`. Our results demonstrate that `Approx-MaMoRL` is effective as a real world solution, making substantial improvement on T_{total} with a 95% statistical significance with a moderate compromise on F_{total}. In section 4.6, we demonstrate that our proposed model is suitable for transfer learning.

4.2 Function Approximation Methods

We compare `Approx-MaMoRL` with `NN-Approx-MaMoRL` from two standpoints - training time, and objective function values. The training data is obtained from `MaMoRL` on a small grid (50 nodes, 93 edges) for 2 assets. Additional information is presented in Table 5. These results demonstrate that, unsurprisingly, for the same amount of training data, `Approx-MaMoRL` is more effective than `NN-Approx-MaMoRL` and is 15x faster in training time (Figure 3). In fact, the Neural Network based approximation requires large amount of training data to be effective which is hard to obtain for our problem since `MaMoRL` could only be run on small instances because of its memory and CPU bottlenecks. For the remainder of the paper, we therefore primarily focus on `Approx-MaMoRL`.

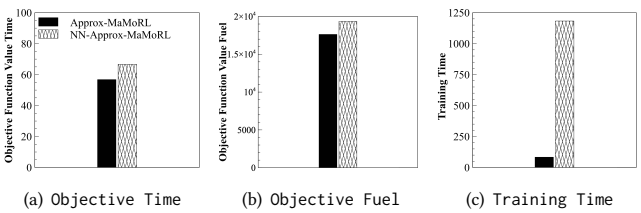


Figure 3: `Approx-MaMoRL` **vs.** `NN-Approx-MaMoRL`

Parameters	#Nodes($\|V\|$)	#Edges($\|E\|$)	#Neighbors(D_{max})	#Assets($\|N\|$)	Speed of Assets(sp)	#Episodes(T_B)	Communication Frequency(k)
Default Value	400	846	9	6	5	10	3

Table 4: Default Parameters Values

Parameters	# Layers	# Nodes		Activation Function		Batch Size	# Epoch
		Layer 1	Layer 2	Layer 1	Layer 2		
Default Value	2	5	1	ReLU	Linear	1000	10000

Table 5: Neural Network Parameters Setting

4.3 Bottlenecks of the Implemented Solutions

We compare Approx-MaMoRL with the exact model (MaMoRL) and the baselines. We present the objective function values as well as CPU time and memory usage (instances that could not be run are represented as N/A in Table 6). Consistent with our theoretical analysis, Table 6 shows that MaMoRL suffers from significant memory and CPU bottlenecks. These bottlenecks are fully averted in Approx-MaMoRL through a "lightweight" function approximation, whereas, the objective function values of Approx-MaMoRL are very close to their exact counterpart in MaMoRL. Unsurprisingly, Baseline-2 results in a majority of collisions (more than 97%). These results demonstrate that Approx-MaMoRL is indeed a suitable alternative to the intractable exact solution.

Scenario	Algorithm	T_{total}	F_{total}	CPU Time	Memory Usage
$\|V\| = 704$	MaMoRL	**N/A**	**N/A**	**N/A**	**205 GB**
$\|N\| = 2$	Approx-MaMoRL	158.85	46482.9	0.8 min	1056 B
$D_{max} = 7$	Approx-MaMoRL with Partial Knowledge	177.94	49353.8	0.9 min	1056 B
	Baseline-1	255.5	38912.75	0.8 min	576 B
	Baseline-2	N/A	N/A	0.6 min	576 B
	Random Walk-Baseline	995.3	54391070.5	N/A	N/A
$\|V\| = 400$	MaMoRL	**N/A**	**N/A**	**N/A**	**17000 TB**
$\|N\| = 3$	Approx-MaMoRL	87.8	28891.4	1.9 min	2304 B
$D_{max} = 9$	Approx-MaMoRL with Partial Knowledge	69.74	13338.7	1.9 min	2304 B
	Baseline-1	128.2	19528.7	1.8 min	864 B
	Baseline-2	N/A	N/A	0.95 min	576 B
	Random Walk-Baseline	153.9	987406.5	N/A	N/A
$\|V\| = 400$	MaMoRL	**317.7**	**80898.4**	**208 min**	**38.5 GB**
$\|N\| = 2$	Approx-MaMoRL	508.1	126919.5	0.8 min	1056 B
$D_{max} = 6$	Approx-MaMoRL with Partial Knowledge	623.3	163528.6	0.85 min	1056 B
	Baseline-1	746.3	113666.9	0.8 min	576 B
	Baseline-2	N/A	N/A	0.6 min	576 B
	Random Walk-Baseline	2442.55	58416741.8	N/A	N/A
$\|V\| = 200$	MaMoRL	**36.3**	**8306.1**	**57 min**	**40 GB**
$\|N\| = 2$	Approx-MaMoRL	48.9	12907.5	0.8 min	1056 B
$D_{max} = 9$	Approx-MaMoRL with Partial Knowledge	48.95	12139.9	0.8 min	1056 B
	Baseline-1	75.4	11479.6	0.8 min	576 B
	Baseline-2	N/A	N/A	0.6 min	576 B
	Random Walk-Baseline	115.1	460182.9	N/A	N/A

Table 6: Comparison Among Implemented Algorithms

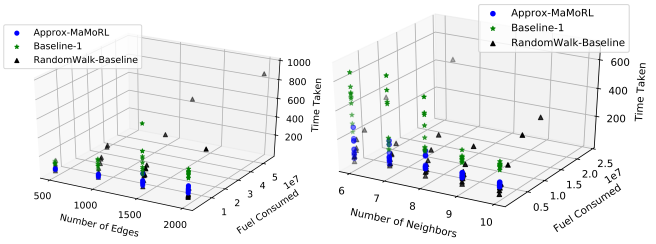
4.4 Effectiveness of Approx-MaMoRL

In these experiments, we vary a large number of parameters systematically that are part of our proposed solution (see Table 4). Figure 4 shows that our proposed framework is capable of returning pareto-front based on both the objectives F_{total} and T_{total}, where Approx-MaMoRL convincingly outperforms the baselines. Additionally, we measure the percentage of relative improvement ($RI(.)$) of F_{total} and T_{total} for Approx-MaMoRL with or without partial knowledge w.r.t other baselines using the following measure.

$$RI(Obj_{total}) = \frac{Obj_{total}^{Baseline} - Obj_{total}^{\text{Approx-MaMoRL}}}{Obj_{total}^{Baseline}} \times 100$$

As demonstrated in Figure 5(a) with increase in the number of nodes, Approx-MaMoRL has around 60% relative improvement for Time w.r.t other baselines. However, it also has negative improvement for Fuel w.r.t Baseline-1 which is understandable. Similar observation holds when the number of edges is increased in Figure 5(b). Figures 5(c) and 5(e) show that having more neighbors or moving assets at different speeds does not change the % relative improvement drastically. By increasing the number of assets in Figure 5(d), the % relative improvement for Time is positive while it is negative for Fuel. According to Figure 5(f), assets take better actions leading to an increase in relative improvement percentage for both objective values, when they are trained more. When assets communicate more often, the objective values improve in general, as a result, % relative improvement of Approx-MaMoRL w.r.t other baselines does not increase as shown in Figure 5(g). Figure 6 shows Approx-MaMoRL with partial knowledge behaves in a similar way.

Baseline-2 results are omitted in Figures 5 and 6, since it results in asset collisions for 97% of the runs.



(a) Varying Number of Edges (b) Varying Number of Neighbors

Figure 4: Pareto Front of F_{total} and T_{total}

Overall, these figures demonstrate the suitability and superiority of Approx-MaMoRL as a real world solution.

4.5 Accuracy and Speed Trade off

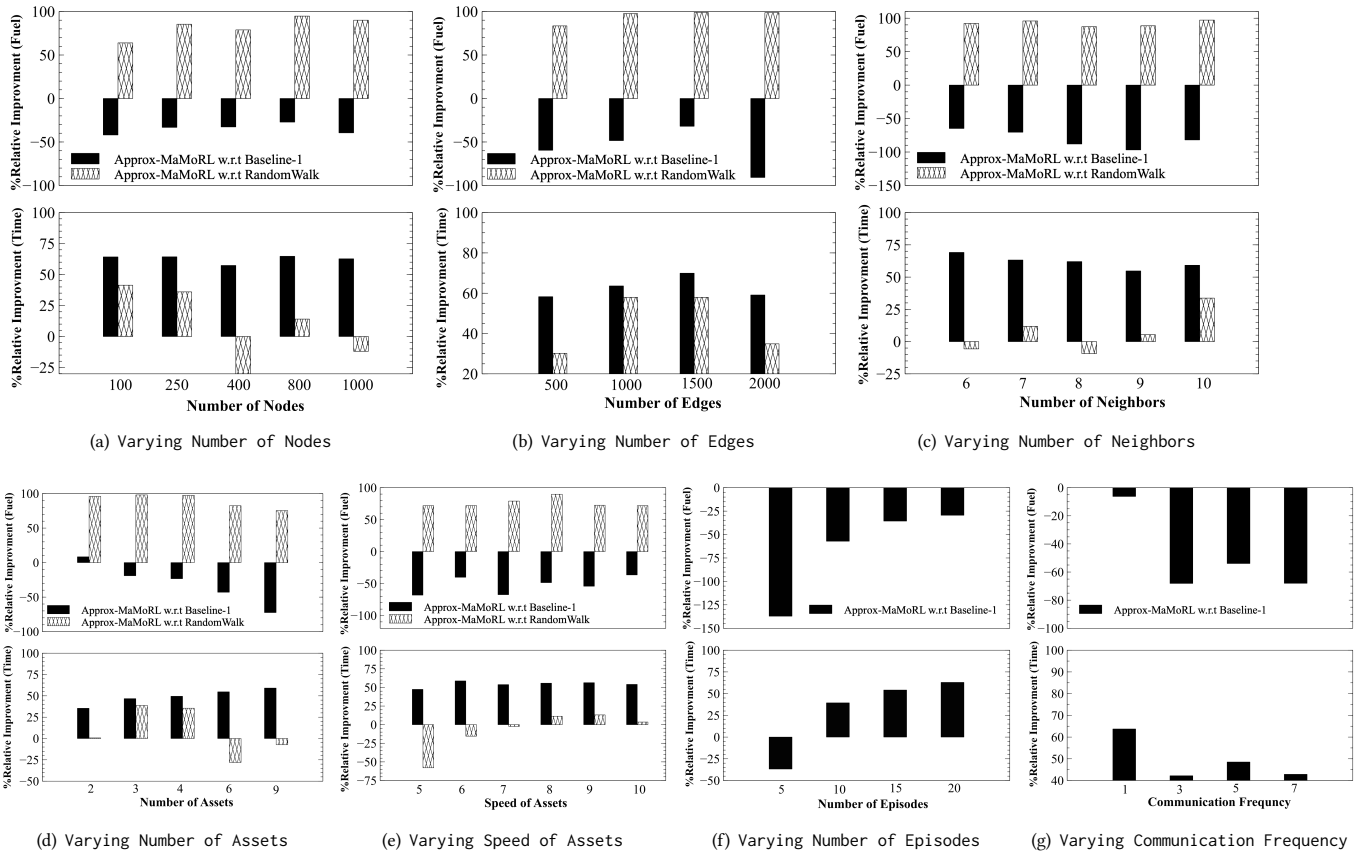
In these experiments, we measure the route planning time of our model and the baselines as seen in Figure 7. We do not include the Random Walk-Baseline as it fails to exhibit consistent behavior in optimizing F_{total} and T_{total}. We consistently observe that Approx-MaMoRL convincingly outperforms Baseline-1 in running time, especially for larger problem settings. These Figures 7(a-g) demonstrate that Approx-MaMoRL is an effective and realistic model to deploy inside TMPLAR.

4.6 Transfer Learning

We present the effectiveness of Approx-MaMoRL in transfer learning. We learn a policy on the Caribbean Grid and use it for navigating the assets in the North America Shore Grid and vice versa. Figure 8 presents these results that corroborate that Approx-MaMoRL is highly effective in transfer learning, i.e., T_{total} and F_{total} of the transferred model are close to those calculated in the original grid.

4.7 Deployment inside TMPLAR

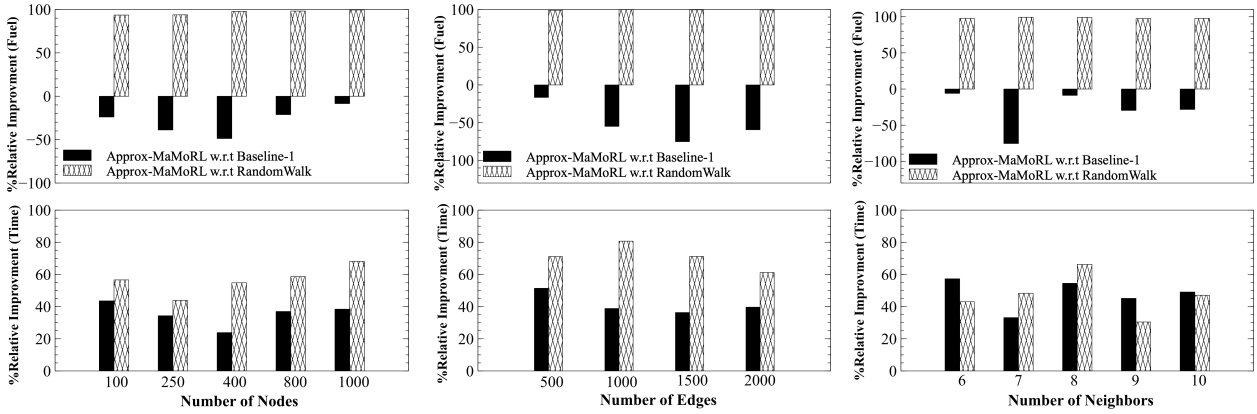
MaMoRL inside TMPLAR [22][4] enables simultaneous route planning for *multiple assets* that are robust to uncertainty under multiple contexts (mission, environment, asset, and threat). It has two views: *a global view*, to facilitate the holistic planning of all the assets



(a) Varying Number of Nodes (b) Varying Number of Edges (c) Varying Number of Neighbors

(d) Varying Number of Assets (e) Varying Speed of Assets (f) Varying Number of Episodes (g) Varying Communication Frequency

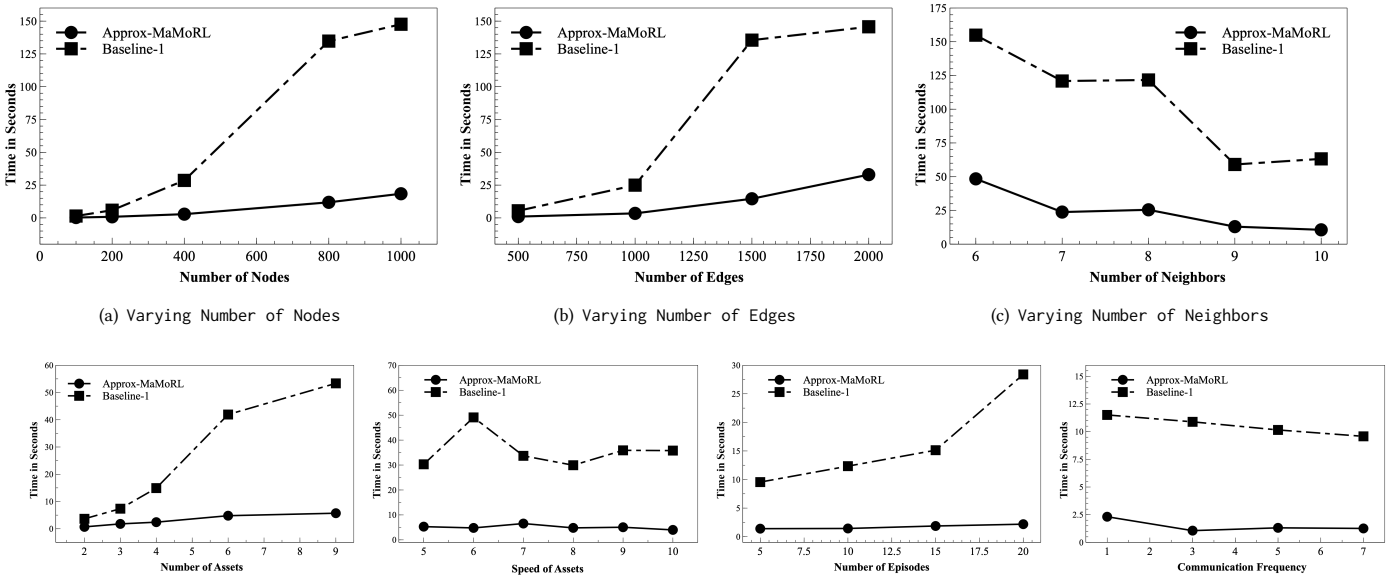
Figure 5: F_{total} and T_{total} Varying Parameters for Approx-MaMoRL



(a) Varying Number of Nodes in Grid (b) Varying Number of Edges in Grid (c) Varying Number of Neighbors in Grid

Figure 6: F_{total} and T_{total} Varying Parameters for Approx-MaMoRL with Partial Knowledge

simultaneously, as well as *a local view* designed for single asset planning. The developed interface is designed through extensive user studies involving human subjects. It is fast, intuitive, logical, and designed to abstract the complex nature of the underlying solution. TMPLAR is written in Python and is intended to be used as a back-end service with JavaScript Object Notation (JSON) objects. MaMoRL is deployed as a docker container simplifying and accelerating the application workflow while giving developers the freedom to innovate with their choice of tools, application stacks, and deployment environments. The efficacy of the tool is evaluated in the Naval Postgraduate School considering different contexts,



(a) Varying Number of Nodes (b) Varying Number of Edges (c) Varying Number of Neighbors

(d) Varying Number of Assets (e) Varying Speed of Assets (f) Varying Number of Episodes (g) Varying Communication Frequency

Figure 7: Running Time Results

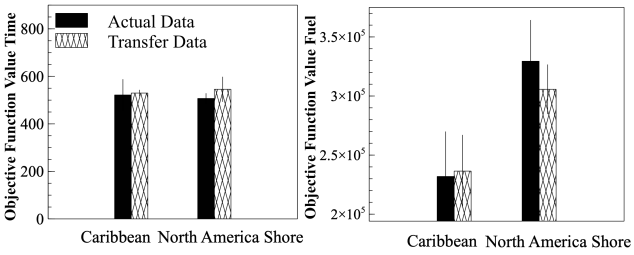


Figure 8: Transfer Learning

hazards, and threats and is now being analyzed by studying the click and eye tracking data of the users, to quantify and compare its ease-of-use and robustness in multi-asset decision making.

5 RELATED WORKS

Reinforcement Learning. Reinforcement Learning (RL) has a broad range of applications in dynamic and uncertain environments [19] [15][10], including traffic lights control and monitoring[25] , exploratory data analysis [2], user group discovery [21], and complex task planning [18]. Authors in [14] and [11] study Markov Decision Problems (MDPs) for a single agent considering multiple objectives with constraints and propose a weighted RL. In [14], authors introduce a Lagrange function for solving utility constrained MDPs. In [27], the author discusses the inefficiencies associated with sample cost in Reinforcement Learning.

Multi-Agent RL. Coordinating multiple agents in a dynamic environment is a complicated problem - some recent efforts have been made to solve this problem using RL [7][8][28][6][23]. [28] studies a cooperative Multi-Agent RL by using a teammate model and reward allotment. [7] considers multiple agents in a task-oriented environment as a Decision-Theoretic Planning (DTP) problem and adapts RL for solving it. [8] shows how to adapt Generalized Learning Automata (GLA) in multi-agent systems. [6] sheds light on the potential applications of Multi-Agent Reinforcement Learning (MARL) in a variety of different areas from robotics to different static games, whilst raising concerns about scalability. [23] points out the absence of existing work where agents reuse knowledge acquired from one another.

Path Planning. Path Planning for ships and aerial vehicles has been studied extensively in [1] [4][22][1][17]. In [4], authors leverage Q-factor as an approximate dynamic programming method for navigating ships in uncertain environments. TMPLAR [22] is developed as a system for Multi-objective Planning and Asset Routing. *We non-trivially adapt these works to address a problem of significant interest to maritime navigation involving multiple distributed assets, objectives, and constraints, as well as study scalability challenges.*

6 CONCLUSION

We study the *Route Planning Problem* (**RPP**), which is formalized as a Team Discrete Markov Decision Process and we propose a *Multi-agent Multi-objective Reinforcement Learning* (MaMoRL) framework for solving it. We demonstrate why exact MaMoRL is computationally expensive and study approximation opportunities. As an ongoing work, we are analyzing the post-deployment data collected through click and eye tracking behavior of the users to quantify its ease-of-use, flexibility, and robustness in multi-asset decision making.

ACKNOWLEDGMENTS

The work of NJIT is supported by the NSF, CAREER Award #1942913, IIS #2007935, IIS #1814595, PPoSS: Planning #2118458, and by the Office of Naval Research Grants No, N000141812838, N000142112966. The work at UConn is supported by the U.S. Office of Naval Research and the U.S. Naval Research Laboratory under Grants N00014-18-1-1238, N00014-21-1-2187 and N00173-18-1-G012.

REFERENCES

[1] Gopi Vinod Avvari, David Sidoti, Lingyi Zhang, Manisha Mishra, Krishna Pattipati, Charles R Sampson, and James Hansen. 2018. Robust multi-objective asset routing in a dynamic and uncertain environment. In *2018 IEEE Aerospace Conference*. IEEE, 1–9.

[2] Ori Bar El, Tova Milo, and Amit Somech. 2020. Automatically generating data exploration sessions using deep reinforcement learning. In *Proceedings of 2020 ACM SIGMOD International Conference on Management of Data*. 1527–1537.

[3] Nicolas Bialystocki and Dimitris Konovessis. 2016. On the estimation of ship's fuel consumption and speed curve: a statistical approach. *Journal of Ocean Engineering and Science* 1, 2 (2016), 157–166.

[4] Adam Bienkowski et al. 2018. Path Planning in an Uncertain Environment Using Approximate Dynamic Programming Methods. In *FUSION*. IEEE.

[5] Lucian Buşoniu et al. 2011. Approximate reinforcement learning: An overview. In *ADPRL*.

[6] Lucian Buşoniu, Robert Babuška, and Bart De Schutter. 2010. Multi-agent reinforcement learning: An overview. *Innovations in multi-agent systems and applications-1* (2010), 183–221.

[7] Gang Chen, Zhonghua Yang, Hao He, and Kiah Mok Goh. 2005. Coordinating multiple agents via reinforcement learning. *Autonomous Agents and Multi-Agent Systems* 10, 3 (2005), 273–328.

[8] Yann-Michaël De Hauwere, Peter Vrancx, and Ann Nowé. 2010. Generalized learning automata for multi-agent reinforcement learning. *Ai Communications* 23, 4 (2010), 311–324.

[9] Edsger W Dijkstra et al. 1959. A note on two problems in connexion with graphs. *Numerische mathematik* 1, 1 (1959), 269–271.

[10] Javier Garcıa and Fernando Fernández. 2015. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research* 16, 1 (2015), 1437–1480.

[11] Peter Geibel. 2006. Reinforcement learning for MDPs with constraints. In *European Conference on Machine Learning*. Springer, 646–653.

[12] Christophe Geuzaine and Jean-Francois Remacle. 2009. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. *Internat. J. Numer. Methods Engrg.* 79, 11 (2009), 1309–1331.

[13] Henry Hsu and Peter A Lachenbruch. 2014. Paired t test. *Wiley StatsRef: statistics reference online* (2014), book.

[14] Yoshinobu Kadota, Masami Kurano, and Masami Yasuda. 2006. Discounted Markov decision processes with utility constraints. *Computers & Mathematics with Applications* 51, 2 (2006), 279–284.

[15] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. 1996. Reinforcement learning: A survey. *Journal of artificial intelligence research* 4 (1996), 237–285.

[16] Francisco S Melo, Sean P Meyn, and M Isabel Ribeiro. 2008. An analysis of reinforcement learning with function approximation. In *Proceedings of the 25th international conference on Machine learning*. 664–671.

[17] Manisha Mishra, David Sidoti, Gopi Vinod Avvari, Pujitha Mannaru, Diego Fernando Martínez Ayala, Krishna R. Pattipati, and David L. Kleinman. 2017. A Context-Driven Framework for Proactive Decision Support With Applications. *IEEE Access* 5, 12475–12495.

[18] S. Nikookar, P. Sakharkar, B. Smagh, S. Amer-Yahia, and S. Basu Roy. 2022. Guided Task Planning Under Complex Constraints. In *38th IEEE International Conference on Data Engineering*. accepted for publication.

[19] Edwin Pednault et al. 2002. Sequential cost-sensitive decision making with reinforcement learning. In *SIGKDD*.

[20] Martin L Puterman. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

[21] Mariia Seleznova, Behrooz Omidvar-Tehrani, Sihem Amer-Yahia, and Eric Simon. 2020. Guided exploration of user groups. *Proceedings of the VLDB Endowment (PVLDB)* 13, 9 (2020), 1469–1482.

[22] David Sidoti et al. 2016. A multiobjective path-planning algorithm with time windows for asset routing in a dynamic weather-impacted environment. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* (2016).

[23] Da Silva et al. 2018. Autonomously Reusing Knowledge in Multiagent Reinforcement Learning.. In *IJCAI*. 5487–5493.

[24] Richard S Sutton et al. 1999. Policy gradient methods for reinforcement learning with function approximation.. In *NIPs*.

[25] Hua Wei et al. 2018. Intellilight: A reinforcement learning approach for intelligent traffic light control. In *SIGKDD*.

[26] Paul Wessel et al. 1996. A global, self-consistent, hierarchical, high-resolution shoreline database. *Journal of Geophysical Research* (1996).

[27] Yang Yu. 2018. Towards Sample Efficient Reinforcement Learning.. In *IJCAI*. 5739–5743.

[28] Pucheng Zhou et al. 2011. Multi-agent cooperation by reinforcement learning with teammate modeling and reward allotment. In *FSKD*, Vol. 2. 1316–1319.

[29] Xinyuan Zhou, Peng Wu, Haifeng Zhang, Weihong Guo, and Yuanchang Liu. 2019. Learn to navigate: cooperative path planning for unmanned surface vehicles using deep reinforcement learning. *IEEE Access* 7 (2019), 165262–165278.